On Dynamic Distributed Computing

Sébastien Gambs¹, Rachid Guerraoui², Florian Huc², and Anne-Marie Kermarrec³
¹Université de Rennes 1 – INRIA/IRISA
²École Polytechnique Fédérale de Lausanne (EPFL)
³INRIA Rennes Bretagne-Atlantique

Abstract. The history of distributed computing is strongly tied to the assumption of a static network composed predominantly of honest nodes. Yet, many modern distributed systems not only are not static but exhibit a high level of churn. This paper demonstrates how to achieve distributed computing in a highly dynamic environment despite the presence of a static Byzantine adversary controlling a large fraction of the nodes. Somewhat surprisingly, we prove that it is possible to maintain clusters of nodes with a majority of honest ones in each in an efficient manner, within a system whose size can change polynomially compared to the initial size of the network. As a corollary of our construction, we solve an open problem in distributed computing. Our construction also provides a basic abstraction enabling for the first time several types of distributed coordination in a highly dynamic setting, such as peer sampling, broadcast, Byzantine agreement and aggregation.

More specifically, we assume a static Byzantine adversary controlling a fraction \( \tau \leq \frac{1}{2l^2} - \epsilon \) of the nodes (for some fixed constants \( l > \sqrt{2} \) and \( \epsilon > 0 \), independent of \( N \), the maximal size of the system). To the best of our knowledge, we are the first to show how to maintain at low cost, in terms of computation and communication, an overlay partitioning the nodes into clusters of size \( O(\log^2 N) \), each containing a majority of honest nodes with high probability. Another algorithm displaying a lower complexity, which we call NOW (for Neighbors On Watch), preserves this property in the presence of high churn (up to a polynomial increase and decrease of the number of nodes). The algorithm NOW itself relies on another novel algorithm called OVER (for Over: Valued Erdős Rényi graph), which maintains an overlay with high expansion coefficient and low degree. In a nutshell, NOW can achieve dynamic clustering with a low complexity, namely with a communication cost induced by each node arrival or departure that is polylogarithmic with respect to the maximal size of the system.
1 Introduction

Distributed computing has a long history, being more than thirty years old, and a large part of the work in this field has been devoted to devising algorithms that perform reliable computations on a network of nodes, despite some of them malfunctioning. The most robust of such algorithms typically tolerate the behavior of so called Byzantine nodes, which are nodes that behave in an arbitrary manner. Some fundamental assumptions that throughout the years have accompanied the design of most of those algorithms are that 1) the network is static and 2) it contains a majority of honest nodes. Yet in reality, many networks are dynamic rather than static, and furthermore sometimes exhibit a high level of churn with many nodes leaving and entering the system without notice and in a frequent manner.

To the best of our knowledge, we are the first to address the issue of partitioning a network subject to high churn into clusters with a minority of Byzantine nodes (i.e., majority of honest nodes) in each. Our protocol, which we call NOW (for Neighbors On Watch), maintains clusters with small size (i.e., polynomial in the actual number of nodes in the system) in a network whose size can vary polynomially compared to the initial size. This question has remained open for high churn [KS10], even if constructions are known when the size of the network vary linearly [AS09,KS10]. For this simpler case, it is possible to maintain a partition using a constant number of clusters, and hence that can have a static structure, which is no longer true when one considers a high level of churn. More precisely, under high churn, one has to deal with a number of clusters that varies, thus making this problem very challenging. The overlay resulting from our construction can be seen as a fundamental abstraction for performing reliable distributed computing. For instance, we show how it can be used to solve fundamental problems from distributed computing such as peer sampling [JGKVS04], computation of aggregation function [VR03], broadcast [PSL80] and agreement [LSP82] in a network with an important fraction of Byzantine nodes and high level of churn.

More specifically, we consider a dynamic network of (current) size $n$, in which an active adversary can control up to a fraction $\tau$ of the nodes. Moreover, we assume that the size of the network evolves in a polynomial manner as $n$ can take any value between $N^{1/y}$ (for some positive constant $y$), the initial size of the network, and $N^z$ (for some positive constant $z$), an upper bound on its maximum size, through a number of join and leave operations polynomial in $N$. The challenge solved by NOW is to structure such a network into clusters of small size (i.e., $O(\log^2 N)$), each containing a majority of honest nodes with high probability. This construction is achieved in a highly dynamic system (whose size may vary polynomially), in a fully distributed manner and without requiring any node in the network to have a global knowledge of the structure of the network.

In a nutshell, NOW works in two phases. During the initialization phase, in which the network is small (i.e., of size $O(\sqrt{N})$), the algorithm constructs a first partition of the nodes into clusters of size $O(\log^2 N)$ together with an unstructured overlay built on top of these clusters. Each cluster is guaranteed to contain a majority of honest nodes with high probability. Afterwards, NOW maintains this partition and the overlay when nodes leave or join the network (maintenance phase). In order to maintain the overlay with a low complexity in terms of communication and computation, NOW relies on OVER (for Over: Valued Erdős Rényi graph), a construction based on random graphs that dynamically maintains an expander graph whp. This construction is similar in spirit to other previous works [LS03,GMS04,AY08], with however the fundamental difference that the graph maintained is not required to be regular, and hence has even less structure, thus making it resilient to a huge number of simultaneous crashes (see Section 8.3 for more details). A key ingredient for the success of OVER is the possibility of achieving efficient random walks, which is guaranteed by the good expansion properties and the low maximum degree, two properties that OVER ensures on the graphs it maintains. In NOW, each node has only a local knowledge of $O(\log^4 N)$ neighbors and the algorithm can tolerate a static Byzantine adversary controlling at most $\tau \leq (\frac{1}{2^l}) - \epsilon$ of the nodes, for fixed constants $l > \sqrt{2}$ and $\epsilon > 0$ (independent of $N$), provided that the honest nodes form a connected component during the initialization phase.

Contrary to some previous approaches, our protocols do not assume that (1) each node knows the identities of all other nodes, (2) the availability of a broadcast channel, (3) all users are honest at initialization, or (4) the size of the network is linear with respect to the initial size of the network. The initial construction of this overlay has a communication cost of $O(N^{3/2} \log N)$ bits and any join or leave
operation induces a cost of $O(Polylog N)$. Furthermore, these operations are done in a load balanced manner, in the sense that each node receives and sends asymptotically the same number of messages.

We believe that our method for creating and maintaining such an overlay has a tremendous range of applications and that it can be applied to solve various fundamental problems in distributed computing. For instance, by relying on this overlay and adapting existing protocols, we have designed the following algorithms:

- **NOW-Agree** is a protocol solving the Byzantine agreement problem with a communication cost\(^1\) of $\tilde{O}(n(\log N + \log \delta^{-1})(\log \log N + M))$ bits, in which $n$ is the current number of nodes in the network, $N$ an upper bound on its maximum size, $M$ the size of the string on which to agree and $\delta$ a parameter related to the probability of success of the algorithm (NOW-Agree as well as the algorithms below are probabilistic algorithms).
- **NOW-Aggregate** solves the computation of aggregation functions (including the leader election problem) in $\tilde{O}(nM)$, with $M$ being the maximum size of the aggregate.
- **NOW-Sample** is the first peer sampling technique tolerating a Byzantine adversary, which has a polylogarithmic complexity.
- **NOW-Broadcast-Local** and **NOW-Broadcast-Global** are two different broadcast protocols. More precisely, NOW-Broadcast-Local enables to broadcast a message even if the sender does not have full knowledge of the network and has a communication cost of $\tilde{O}(n(\log N + \log \delta^{-1})(\log \log N + M))$ bits, in which $M$ is the size of the message to be broadcast and $\delta$ a parameter related to the probability of success of the algorithm. NOW-Broadcast-Global assume that the sender can communicate with all the nodes of the network, which makes it possible to further reduce the communication cost of the broadcast protocol to $\tilde{O}(nM + n(\log N + \log \delta^{-1})(h(B.M) + k))$, for $h(B.M)$ the size of the output of a universal hash function which is related to the probability of success as well as $k$ an other security parameter and $\delta$ a parameter related to the probability of success of the NOW-Broadcast-Local which is used as a sub-routine.

Each of these protocols is interesting in its own right as we discuss in the related work section. Figure 1 describes the interactions between these different protocols. More precisely, a given protocol uses the protocol below it as displayed in this figure.

![Fig. 1. Overview of the protocols stack.](image)

The rest of the paper is organized as follows. First, we describe the model in Section 2, before giving some background notions about continuous time random walk in Section 3. Afterwards in Section 4, we describe OVER, a protocol dynamically maintaining an expander, before giving an overview the protocol NOW (Section 5), which we further detail in Section 6 (initialization) and Section 7 (maintenance). In Section 8, we analyze the protocol NOW and prove its validity. We also further discuss how to weaken some of our system assumptions in order to make our constructions more generic. Finally, we describe possible applications of our overlay (Section 9), and review related work in Section 10 before concluding.

\(^1\) $\tilde{O}()$ is the same as $O()$ ignoring the logarithmic factors. The details of the analysis of the complexity can be found in the corresponding section.
2 Model

2.1 System assumptions

We consider a dynamic network with a discrete time variable $t_i$ (for $i \in \mathbb{N}$). We assume synchronous nodes that can send or receive a message from all their neighbors at each time step. For the sake of clarity, we assume in this paper that initially there are $\sqrt{N}$ nodes in the network and that the number of nodes in the network always remains between $\sqrt{N}$ and $N$. However this assumption can be relaxed in the sense that the lower bounds and upper bounds can be replaced by $N^{1/y}$ and $N^z$ respectively for any positive constant $y$ and $z$.

For the sake of clarity, we analyze our protocol under the assumption that at each time step only one node can leave or join the network. However, our protocol can be generalized to the setting in which, at each time step, a polylogarithmic number of nodes can join or leave the network. We further ignore in the analysis the computational time required by a node and source-to-end delay for message delivery, although these two hypothesis could also be discarded as explained in Section 8.3. Finally, nodes do not need to make any specific action when leaving the network, therefore a crashed node can be considered as a node that has left. Instead, we assume a mechanism enabling a node to detect if one of its neighbors has crashed or left the network without notice (which once again is the same from our point of view).

NOW can worked even in the presence of an active adversary controlling a fraction $\tau = \frac{1}{2\sqrt{t}} - \epsilon$ (for some $\epsilon > 0$) of the nodes in the network (the exact meaning of the variable $t$ will be detailed later). The nodes controlled by the adversary may not follow the rules of the protocol and can behave in a arbitrary manner, which corresponds to a Byzantine adversary. In our setting, a typical objective for the adversary is to gain the lead in one (or more) of the clusters constructed. At the beginning of the protocol (i.e., at time $t_0$), the adversary can choose a fraction $\tau$ of the nodes to corrupt. We also assume that the honest nodes form a connected component and that the adversary cannot split this connected component into disjoint parts. Moreover during the execution of the protocol, each time a node joins the network, the adversary can choose at this time whether or not to corrupt it. However, this decision does not change over time (in this respect the adversary is static and not adaptive). Concerning OVER (Section 4), we assume that all the vertices of the graph it deals with are honest and that each node leaving the network is chosen at random. These two requirements will be ensured whenever it is required.

We assume that each node has a unique identifier together with an associated pair of private and public keys and that malicious nodes cannot forge identities. For instance, the generation of private and public keys can be done by a (trusted) Certification Authority, and the public key can be used as the identifier of a node. The pair of keys is required to tolerate an adversary controlling a fraction $\tau = \frac{1}{2\sqrt{t}}$ of the nodes. If the fraction of nodes controlled by the adversary is instead $\tau = \frac{1}{\sqrt{t}}$, then the assumption that each node has a unique identity is no longer required and can be replaced by the hypothesis that each pair of neighbors in the graph can communicate through a secure channel. Unlike most of the previous works, we do not assume that each node has a global knowledge of the identities of all nodes in the network (except during the initialization phase in which the global knowledge of the identities is computed once in a network of small size). Instead, each node has only a local knowledge of a few nodes in the network (typically $O(Polylog N)$) and may not even be aware of the current size of the network.

2.2 Notation

We use the time step as a subscript of a variable to indicate the moment at which the variable is considered. For instance, $n_{t_i}$ represents the number of nodes in the system at time $t_i$, while $\#C_{t_i}$ stands for the number of clusters in the network at the same time, and $|C_j|_{t_i}$ is the size of the cluster $j$ at time $t_i$. (We omit the index of the time step when it is not relevant, hence $n$ stands for the current number of nodes in the network.). Therefore, taking into account the upper and lower bounds we assume on the network, we get $\sqrt{N} \leq n_{t_i} \leq N$ at any time step $t_i$. During the analysis of the protocols, we will be interested in the communication cost of a protocol, which corresponds to the number of bits exchanged during the protocol, as well as the round complexity of the protocol, which is the number of steps (i.e., rounds) required by the protocol to terminate.
Given a graph \( G = (V, E) \), and a vertex \( v \in G \), we denote by \( d_v \) its degree. Similarly, for a particular cluster \( C \), \( d_C \) is the number of clusters adjacent to \( C \) (i.e., the degree of \( C \) in the graph of clusters). We will refer to \( d_C \) as the degree of cluster \( C \). By abuse of language, we will also sometimes refer to a cluster when we want to refer to all of its nodes. For instance, when we write “a cluster does . . . ”, we mean “all the nodes from the cluster do . . . ”. In a similar manner, when we write that there is an edge between two clusters, we really mean that there is an edge between all pairs of nodes, whose arrival and departure nodes are not located in the same cluster.

3 Background on Continuous Time Random Walk

In this section, we review fundamental results on Continuous Time Random Walk [AF02] that we will use as our building blocks for both protocols OVER and NOW.

Given an undirected graph \( G = (V, E) \), in which \( G \) is the set of nodes and \( E \) the set of edges, a Continuous Time Random Walk (CTRW) on \( G \) consists in the following stochastic process: a virtual agent walks from nodes to nodes through edges of the graph chosen uniformly at random from the ones incident to the node on which the agent is currently positioned. The walk is performed for a given amount of time \( T \) and when the agent visits a node \( v_j \), it decrements a counter representing the remaining time of the random walk by \( \log(1/U)/d_{v_j} \), where \( U \) is a number chosen uniformly at random from \((0, 1)\) and \( d_{v_j} \) is the degree of node \( v_j \). If the value of the counter is still positive, the agent chooses at random a new neighbor and walks to this node. Otherwise, the walk stops. We denote by \( \psi_t(v_i) \) the probability vector of the position of the agent at time \( t \) if the agent starts the walk at the node \( v_i \in V \). This type of CTRW has a uniform stationary distribution \( \pi = (1/n)_i \) [AF02], and the speed of convergence towards this stationary distribution is characterized by the mixing time of the walk.

Definition 1 (Mixing time [Lin02]). For every \( \epsilon > 0 \), the \( \epsilon \)-mixing time of a random walk is

\[
T_{\text{mix}}(\epsilon) = \max_{v_i \in V} \min \{t \mid d(\psi_t(v_i), \pi) \leq \epsilon, \forall t' > t\}
\]

The mixing time can be seen as the time required to get the probability distribution of the current position of the agent \( \psi \) close to the stationary distribution \( \pi \). The interpretation of [Lin02] (Theorem 5.2) states that \( 1/\epsilon \) represents the expected number of samples needed before retrieving an improperly selected node compared to the stationary distribution \( \pi \). As we rely on a random walk process to generate the samples, and that we perform a number of samples that is polynomial in \( n \), we need to set \( \epsilon = \Theta(1/n \log n) \).

The choice of this value for \( \epsilon \) ensures that with high probability all our samples can be considered as picked uniformly at random.

To analyze the mixing time of a CTRW, one can simply study the Laplacian matrix \( L \) of \( G \) that is defined as follow: \( L_{ij} = -1 \) if \( i \neq j \) and \((v_i, v_j) \in E \), \( L_{ij} = d(v_i) \) if \( i = j \) and \( L_{ij} = 0 \) otherwise. The mixing time of a CTRW is related to the spectral gap of the Laplacian matrix. We denote the eigenvalues of the Laplacian matrix by \( \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n \). Since \( \lambda_1 = 0 \), the spectral gap is \( |\lambda_1 - \lambda_2| = \lambda_2 \) in this case and we have the following theorem [GKLMM07]:

Theorem 1 ([GKLMM07]). Given a CTRW \( \psi_t(v_i) \) with spectral gap \( \lambda_2 \), and stationary distribution \( \pi = (1/n)_i \), we have:

\[
d(\psi_t(v_i), \pi) \leq \frac{\lambda_2}{\lambda_1} e^{-\lambda_2 t}
\]

We will use \( T_{\text{mix}} = \log^2 n / \lambda_2 \), as for this walk time, we get \( d(\psi_t(v_i), \pi) \leq \frac{1}{2n \log n - 177} \). The immediate consequence is that using this time, before we get an erroneous sample, we have to make an exponential number of walks in expectation. From Theorem 1, we obtain that to upper bound the mixing time of our CTRW, it is sufficient to estimate the second eigenvalue of the Laplacian matrix, \( \lambda_2 \), which is related to the isoperimetric constant of \( G \) as defined below. In short, the isoperimetric constant is a measure of how fast data propagates through the resulting graph.

Definition 2 (Isoperimetric constant [KS11]). Given a graph \( G = (V, E) \), the isoperimetric constant of \( G \) is the constant \( I(G) = \inf_{S : |S| \leq n/2} E(S, S) / |S| \) where \( E(S, S) \) is the number of edges between \( S \) and \( \bar{S} = V \setminus S \).
Following this, we have $\lambda_2 \geq I(G)^2/2\Delta(G)$, where $\Delta(G)$ is the maximum degree of the graph ([MLMKG06]). To illustrate this, consider a graph from the Erdős-Rényi $\mathcal{G}(n, p)$ model, which corresponds to a graph with $n$ vertices where each edge is present with probability $p$. From Theorem 5.4 of [GMT05], we obtain that for this graph, for $p = \log(n)^2/n$ and $d = np$, with high probability $\lambda_2 \geq d^2/8\Delta(G)$, and if $\Delta(G) \leq d\log(n)^2$, we have $\lambda_2 \geq d/8\log(n)^2$, which gives $T_{\text{mix}} = 8\log^2 n$.

4 OVER: Maintaining Expander Graphs

Our objective is to dynamically maintain a partition of nodes into clusters of small size. To realize this in an efficient manner, we build an overlay in which the clusters are organized in a specific manner. While many overlays have been proposed during the last decade, none of them have considered an adversary that is as powerful as ours. We provide an overview of the different overlay constructions proposed in the past in Section 10.1. While we could rely on the overlay construction proposed in [GMT05] (further analyzed in [GMS04,AY08]), this would not be enough to construct a protocol robust against an adversary that can force many nodes to leave the network without initiating a leave operation (for instance by making them crash through a denial-of-service attack). Therefore, we have designed a novel protocol called OVER (Over: a Variation on Erdős-Rényi graphs), which maintains an unstructured overlay based on random graphs from the Erdős-Rényi model.

4.1 OVER in a nutshell

OVER relies mainly on two subroutines, called Add and Remove (detailed in the following subsection), to maintain the overlay. The main objective of the overlay is to connect the different clusters of nodes in a manner that allows for efficient communication and coordination between the different clusters. For the remaining of the paper, one can think of the overlay as as graph that is constructed dynamically and such that the vertices of this graph represent disjoint clusters of nodes.

Starting from a graph that is a random graph drawn from the Erdős-Rényi model, we prove that with high probability, after a sequence of vertices addition and removal polynomial in $N$, the resulting graph will have a large expansion factor and a low degree. Furthermore, we prove than this graph is robust even against a number of vertices removal that are performed without calling the subroutine Remove (which corresponds to a crash of a node of this graph). More precisely, even if a number of removals proportional to its size occurs, the resulting graph will still behave as desired.

This section deals with a graph which represents the overlay, i.e. which represents how the clusters are interconnected. Since the clusters will be guaranteed to be composed of a majority of honest nodes, we can assume in this section, that the graph uniquely composed of honest vertices. Moreover, the time is discrete and subdivided into time steps such that at each time step at most one vertex is added or removed from the graph. (In Section 8.3, we will show how to weaken this assumption in order to obtain a more generic result). The evolution of the graph is represented by a sequence $G_{t_0}, \ldots, G_{t_i}, \ldots$ in which $t_i$, for $i \in N$ indicates the time step at which the addition or removal operation considered has been performed. Finally, we denote by $n_{t_i}$ the number of vertices in $G_{t_i}$.

4.2 Primitives from OVER

We detail thereafter the different primitives that are used to build the OVER algorithm:

- CTRW$(v)$ returns a vertex chosen by a CTRW starting at vertex $v$. The communication cost and round complexity of this subroutine are equal to twice the length of the path performed by the CTRW, which is $O(\log^3 N)$ as proven in the next subsection.
- Link$(u, v)$ adds an extra edge between the vertices $u$ and $v$. The communication cost and round complexity of this subroutine are equal to the length of the path used to communicate from $u$ to $v$.
- Add$(v)$ is the operation executed by a vertex $v$ contacted by a vertex $u$ upon joining the network. When this operation is performed, $2\log^2 N$ edges are added at random to connect $u$ to the rest of the graph using the subroutine Link$(u, \text{CTRW}(v))$. 

6
Algorithm 1 Continuous Time Random Walk: CTRW(v).

Require: A connected graph $G = (V, E)$ whose Laplacian second eigenvalue is $\lambda_2$ and a starting vertex $v$.
Ensure: The returned vertex is chosen uniformly at random.

$v$ sets $T = \log^2 n/\lambda_2$.
$v$ chooses at random a neighbor $u$ and moves to $current\_node = u$.
while $T > 0$ do
    $current\_node$ chooses at random a number $U \in (0, 1)$.
    $current\_node$ updates $T = T - \log(1/U)/d_{current}$.
    $current\_node$ chooses at random a neighbor $u$ at random and moves to $current\_node = u$.
end while
Return $current$ to the original vertex $v$ by following in a backward manner the path constructed by the CTRW.

Algorithm 2 Adding a new edge: Link($u, v$).

Require: A connected graph $G = (V, E)$ and two vertices $u$ and $v$.
Ensure: The addition of an edge between $u$ and $v$.

$u$ adds $v$ to its list of neighbors.
$v$ adds $u$ to its list of neighbors.

Algorithm 3 Adding a vertex: Add($v$).

Require: A connected graph $G = (V, E)$, an new vertex $u$ that contacts a vertex $v$ already present in the graph.
Ensure: The addition of $2 \log^2 n$ edges at random.

for $i = 0; i = i + 1; i < 2 \log^2 n$ do
    $v$ executes Link($u, CTRW(v)$).
end for

– Remove($v$) is the operation executed by a vertex leaving the network without crashing. When this operation is performed, the edges connected to $v$ are removed and $2 \log^2 N$ new edges are added at random using the subroutine Link($CTRW(v), CTRW(v)$). This addition of edges in case of a removal is fundamental as otherwise the number of remaining edges in the graph may not be sufficient to guarantee connectivity after an important number of vertex removals. This is particularly true in our setting in which the size of network may vary polynomially compared to the initial size.

Algorithm 4 Removing a vertex: Remove($v$).

Require: A connected graph $G = (V, E)$ and a vertex $v$ that has left $G$ in a proper manner (i.e., without crashing).
Ensure: The addition of $2 \log^2 n$ edges at random.

for $i = 0; i = i + 1; i < 2 \log^2 n$ do
    $v$ executes Link($CTR\tilde{w}(v), CTR\tilde{w}(v)$).
end for

4.3 Analysis of the OVER graph

We now prove that at each time step, the graph constructed by OVER exhibits good expansion properties and a small maximum degree. These results are proved under the assumption that the random choices made during the construction of $\tilde{G}^R$ are perfectly uniform (i.e., the small bias induced by the random walk is ignored). This assumption is justified by the fact that we consider a mixing time after which the distance from the distribution of the sample to the uniform distribution is $O(n^{-\log n})$. We further demonstrate it for a single addition or removal of vertex at each time step, but it can be straightforwardly extended to a higher number of additions and removals that could be performed in parallel.

Theorem 2 (Isoperimetric constant of $\tilde{G}^R$). With high probability, $G_{t_i}$ has an isoperimetric constant $I(G_{t_i}) \geq \log^2 N/2$.

Proof. To prove this theorem, we demonstrate that at each time step $t_i$, $G_{t_i}$ can be seen as an instance of a graph from the Erdős-Rényi model $\mathcal{G}(n_{t_i}, p(n_{t_i}))$ with $p = \log(N)^2/n_{t_i}$ to which some edges have been added.
If $p(n)$ is decreasing (i.e., $p(n+1) < p(n)$), a graph generated from the model $\mathcal{G}(n+1, p(n+1))$ can be considered as a sub-graph of a graph of $\mathcal{G}(n, p(n))$ to which a new vertex $v$ has been added and such that each new potential edge is created with probability $p(n+1)$. Therefore by drawing on this analogy, one can proceed as follow: first choose a degree $d_v$ for $v$ according to the binomial distribution $Bi(n+1, p(n+1))$, and then choose $d_v$ neighbors uniformly at random. We follow this procedure when we add a new vertex to $G_t$ (join operation), with $p(n_t) = (\log^2 N) / n_t$. The added vertex has a degree equals to $2\log^2 N$, which with high probability leads to a larger degree than $Bi((n_t, \log^2 N / n_t))$.

Similarly, when $p(n) = (\log^2 N) / n$, a graph issued from the model $\mathcal{G}(n, p(n))$ can be seen as a sub-graph of a graph of $\mathcal{G}(n, p(n+1))$ to which less than $2\log^2 N$ edges have been added at random. We follow this procedure when a vertex of $G_t$ is removed (leave operation). Therefore, $G_t$ can be seen as an instance of $\mathcal{G}(n_t, p(n_t))$ to which some edges have been added. From [GMT05] and as $p(n_t) n_t > \log(n_t)$, we have $I(G_t) \geq p(n_t) n_t / 2 = (\log^2 N) / 2$. □

**Theorem 3 (Maximum degree of $\hat{G}^\infty$).** With high probability, at any moment after a polynomial number of time steps $t$, $G_t$ has maximal degree at most $\log^4 N$.

**Proof.** Given a sequence of graphs of the form $G_t$, we want to compute the sequences of degrees of a specific vertex $v$. Let $t_{\text{join}}$ be the time at which $v$ joins the network. If $t_{\text{join}} = t_0$, then $v$ is in $G_0$ the initial graph. Otherwise if $t_{\text{join}} > t_0$ then $v$ belongs to $G_{t_{\text{join}}}$ but not to $G_{t_{\text{join}} - 1}$. We now focus on a sequence during which $v$ does not leave the network. During this sequence, the addition and removal of vertices has the following impact on the degree of $v$, for $n_t$, the number of vertices in $G_t$ before the action performed at step $t + 1$ is executed:

- When a vertex is added, $n_{t+1} = n_t + 1$, and the degree of $v$ increases by one with probability $(2\log^2 N) / n_t$.
- When a vertex is removed, $n_{t+1} = n_t - 1$, and the degree of $v$ decreases by one with probability $d_v(n_t) / (n_t - 1)$ as the vertex removed at random may be connected to $v$. Afterwards, as $2\log^2 N$ edges are added at random, the degree of $v$ increases by at most a value corresponding to the hyper-geometric distribution as $2\log^2 N$ trials are performed to select $n_t - 2$ edges among $(n_t - 1)$ possible edges in total.

By assumption, the number of vertices in $G_t$ at a particular time step $t$ verifies $N^{1/2} \leq n_t \leq N$. If we consider a sequence of addition and removal of vertices starting from $n_{t_0} = N^{1/2}$, then there are at most $N - N^{1/2}$ more addition than removal operations. Moreover, each removal occurring at time $t_j$ can be associated to an addition that has occurred at time $t_i < t_j$ such that $n_{t_i} = n_t$.

Considering the event $\{d_v(n_t \geq \log^4 N\}$, we want to prove that its probability of occurrence is very low. Such an event would be preceded by another event $\{d_v(n_t \geq \log^3 N\}$ such that from $t'$ to $t$ the degree of $v$ remains higher than $\log^3 N$. The probability that such an event occurs can be upper bounded by the probability of the following random variable being larger than $\log^4 N$. For all time steps $t_i$, we have $N^{1/2} \leq n_{t_i} \leq N$ and we define $X = \sum_{j=N^{1/2}}^N W_j + \sum_{t=0}^T (X_t + Y_t + Z_t)$ in which:

- $W_j = +1$ with probability $\log^2 N / j$.
- $X_t = +1$ with probability $\log^2 N / n_t$.
- $Y_t = -1$ with probability $\log^3 N / n_{t_i}$ (as $d_v(n_t \geq \log^3 N$, we can lower bound the probability that it decreases).
- $Z_t$ follows the hyper-geometric distribution corresponding to $2\log^2 N$ trials to select $n_{t_i} - 2$ elements among $(n_{t_i} - 1)$.

Using standard Chernoff bounds, we obtain that in order to reach $\sum_{t=0}^T X_t \geq \log^4 N$, $T$ needs to be sufficiently large with respect to $N$ so that $1/n_{t_i}$ is large enough. From this, it is possible to infer that the probability of the event $\{X \geq \log^4 N\}$ is exponentially small in $N$. Therefore, the maximum degree of $G_t$ is upper bounded with high probability by $\log^4 N$ at each time step during a polynomial number of join and leave operations. □

\[\text{Starting from a larger size would result only in adding less edges, therefore the degree would have less probability to go over } \log^4 N.\]
Remark 1 (Comparison of OVER versus [LS03]). If we consider a graph $G_t$ of size $n_t$, then this graph is robust against $\epsilon n_t$ crashes of vertices selected at random while preserving the above mentioned properties. This property can be deduced from the proof of Theorem 2. Indeed, the graph obtained after $\epsilon n_t$ random crashes can be considered as the union of edges constructed at random and of an instance of a graph $\mathcal{G}(n_t, p'(n_t))$ with $p'(n_t) = \log^2 N/(1+\epsilon)n_t$ (here $p'(n_t)$ replaces $p(n_t) = \log^2 N/n_t$). In comparison, a graph obtained using techniques presented in [LS03,GMS04,AY08] is composed of an union of cycles. Therefore if even a single node crashes, the cycles are cut and the protocol does not work anymore.

Remark 2. From Theorems 2 and 3, we have that the second eigenvalue of the Laplacian matrix of $G_t$ verifies the property that $\lambda_2 \geq I(G_t)^2/2\Delta(G_t) \geq 1/8$. We choose for the duration of a CTRW $T_{mic} = 8\log^2 N$. For this specific duration, with high probability, the cluster reached after a CTRW can be considered as chosen uniformly at random from $G_t$, and the number of vertices visited during the CTRW is $O(\log^3 N)$.

5 NOW: Outline

While OVER maintains a structured overlay on the graph of clusters, the objective of NOW (Neighbours On Watch) is to ensure that with high probability each cluster contains a majority of honest nodes. We coin this protocol as NOW as it relies on partitioning the nodes into clusters so that honest nodes inhibit the behavior of malicious ones. NOW consists of two phases: the initialization phase and the maintenance phase. In a nutshell, the initialization phase generates an initial overlay satisfying the desired requirements, while the maintenance phase ensures that even after a polynomially long sequence of leave and join operations, these desired properties still hold.

The initialization phase (Section 6) is itself divided into two sub-phases. First, an algorithm is run in order for the nodes to acquire a global knowledge of the nodes in the network. Afterwards, a Byzantine agreement algorithm [KS10] is used to construct an initial partition forming the basis of the overlay $\hat{G}^R$. The maintenance phase ensures that the nodes are partitioned into clusters of size $\Theta(\log^2 N)$ such that, with high probability, all clusters contain a majority of honest nodes. In order to achieve this, we design rules to manage the clusters based on OVER, the distributed protocol described in Section 4 which maintains an overlay over the clusters which has good expansion properties and low degree.

**Initialization**

- Small graph ($n = \sqrt{N}$)
- Compute global knowledge
- Apply robust Byzantine Agreement

*complexity: $O(N^{3/2} \log N)$*

**Maintenance**

- Local knowledge and $\sqrt{N} \leq n \leq N$
- Preserve a good partition of the nodes
- Maintain the overlay

*complexity: Polylog($N$)*

![Fig. 2. NOW.](image)

**Overlay.** Using OVER, we maintain an overlay $\hat{G}^R$ corresponding to a graph on the vertex set composed of the set of the clusters previously computed. More precisely, $\hat{G}^R$ is a random graph constructed recursively as described in Section 4. In practice, it can be considered as a graph from the Erdős-Rényi model $\mathcal{G}(\#C_1, p)$ with $p = \log^2 N/\#C_1$ to which some extra edges have been added. When an edge links two clusters $C_i$ and $C_j$ in $\hat{G}^R$, it means that all the nodes from $C_i$ know the identities of all the nodes from $C_j$ (and vice-versa), as shown on Figure 3. A node only needs to know the identities of nodes in its cluster and the neighboring ones (but not the identities of all the nodes in the network as most previous works). In Theorem 2 and 3 (Section 4), we have proved that with high probability, at each step, the isoperimetric constant of $\hat{G}^R$ verifies $I(\hat{G}^R) \geq \log^2 N/2$, and that its maximum degree is at most $\log^4 N$. 


an edge in $\hat{G}$

an edge between two nodes

Fig. 3. Illustration of the different types of edges.

6 NOW: Initialization Phase

Network Discovery. At initialization, the network is composed of $n_0 = \sqrt{N}$ nodes. The protocol starts by running an algorithm that informs each node of all the other nodes identifiers (Algorithm 5). The initialization phase is the only moment at which the global knowledge of all the nodes in the network is computed and this computation is performed while the size of the network is still “small”. Afterwards, the computation of the global knowledge is performed, it is possible to use standard off-the-shelf Byzantine agreement protocol that are robust to malicious nodes such as [KS10] to construct an initial partition which forms the basis of the overlay $\hat{G}^R$. Given a graph $G = (V, E)$, in which $V$ is the set of vertices and $E$ the set of edges connecting these vertices, and a subset of vertices $X \subset V$, we denote by $G[X]$ the sub-graph of $G$ induced by $X$.

Algorithm 5 Global knowledge computation

| Require: A graph $G = (V, E)$ in which honest nodes form a connected component |
| Ensure: All honest nodes know the identifiers of all the other nodes in the system |
| Set the request list of each node to be the empty set $\emptyset$ |
| while a node did not put all of its neighbors in its request list or a node has non-empty request list do |
| A node with an empty request list chooses at random a neighbor with whom it has not communicated yet and adds its to its request list |
| A node sends its list of neighbors to the first node of its request list |
| A node receiving a message, adds the sender of the message to its request list and merge the list of received nodes with its list of neighbors |
| end while |

Theorem 4 (Global knowledge computation). In a graph composed of $n$ vertices, Algorithm 5 stops after $O(n^2)$ steps. When the algorithm terminates, it is guaranteed that all honest nodes know the identities of all nodes in the network.

Proof. Each node sends a message to each other node exactly once, therefore the algorithm stops after $n(n - 1)$ steps. Moreover, due to the assumption that the adversary cannot forge false identifiers, no honest node will try uselessly to contact an imaginary node. An edge $(x, y)$ is said to be unchecked if the nodes $x$ and $y$ have not communicated directly to each other using this edge. We denote by $H$ (resp. $M$) the set of honest (respectively malicious) nodes. Furthermore, $E^?$ is the set of unchecked edges, $E^\hat{v}$ the set of checked edges and $H^?$ the set of honest nodes with degree less than $|H| - 1$ belonging to $G^? = (H, E^?)$.

10
We now prove the theorem by induction. The induction hypothesis is that the unchecked edges induce a connected graph over all honest nodes that have not checked all the edges going to the other honest nodes yet (i.e., \( G^i = (H^i, E^j) \)) forms a connected graph. Initially, the induction hypothesis is true by assumption. For an edge \((x, y)\) that becomes checked, if it was not initially a cut edge in \(G^i\), the graph induced by the unchecked edges, the hypothesis remains true. On the contrary, if it is a cut edge, \(x\) is connected via an unchecked edge to another node \(z\). Therefore, by checking the edge \((x, y)\), the edge \((y, z)\) is added to \(E\) and becomes unchecked. Hence, it is also added to \(E^j\) and the induction hypothesis remains true. By checking the edges one after the other, we are guaranteed that at the end of the process all honest nodes know the identities of all the other nodes in the system and that they will all have the same view of the graph of the network (which by assumption remains static during this phase).

The communication cost of Algorithm 5 in terms of the number of exchanged bits, is \(O(n^3 \log n)\). Indeed, in total \(O(n^2)\) edges are checked, generating each time a message of size \(O(n \log n)\) containing all the identifiers of the neighbors. When \(n = \sqrt{N}\), this gives an overall complexity of \(O(N^{3/2} \log N)\).

**Clusterization.** Once all the honest nodes know the identities of all the nodes in the network, any Byzantine agreement protocol can be used, such as for instance [KS10] whose complexity is \(O(n \sqrt{n})\). This protocol selects a representative cluster of logarithmic size containing an honest majority. Afterwards, the nodes of this representative cluster can partition the network into \(\#C\) clusters, \(\{C_1, \ldots, C_{\#C}\}\), each of size \(k \log^2 N\), for some constant \(k\) that can be considered as a security parameter of the protocol and which is chosen apriori depending on the application requirements. The higher \(k\), the less chance the adversary has to control a majority of nodes in one of the clusters. Choosing the partition at random ensures that with high probability, there is a majority of honest nodes in each cluster. The representative cluster can generate a random number such that the malicious nodes cannot influence it by using the primitive distributed random number generation (see next section for more details).

It is fundamental for the security of our protocol that each cluster has a majority of honest nodes. Indeed, a node receiving a message from all the nodes of a particular cluster consider this message has valid if and only if, it receives the same message from more than half of the nodes of this cluster. Using this rule for inter-cluster communication, together with the condition that each cluster has a majority of honest nodes, is sufficient to ensure the correctness of the protocol. To summarize, a node accepts a message if and only if it receives the same message from more than half of the nodes of this cluster.

### 7 NOW: Maintenance Phase

While the initialization phase of NOW ensures the desired properties for both the overlay and the clusters, maintaining them under a high level of churn is challenging. In this section, we describe how to preserve the property that each cluster is composed of an honest majority in presence of nodes join and leave operations. To realize this objective, we rely mainly on three primitives: **rand\&num** (to distributively generate random numbers), **rand\&cl** (to distributively choose a cluster at random), and **exchange** (to distributively exchange nodes between clusters).

It is essential for the security of our protocol to induce churn in the clusters in which nodes have joined or left. Indeed, as mentioned in [Sch05,AS09], without additional churn, the adversary could control a majority of nodes in a cluster after a few steps by using a very simple strategy: the adversary chooses a specific cluster and keeps adding and removing the malicious nodes until they fall into that cluster. Similarly, it is crucial to introduce churn if nodes may leave the network due to some actions of the adversary (for instance through a denial-of-service attack). The required churn is induced by the **Join** and **Leave** operations. Complementary, the **Split** and **Merge** operations ensure that the clusters remain of size \(\Omega(\log^2 N)\), and that the required properties of \(G^R\) (i.e., expansion and low maximum degree) are preserved. As an alternative to our approach, we could also adapt the procedure from [AY08] for the **Split** and **Merge** operations.
7.1 Building blocks

Distributed random number generation. The primitive randNum enables all the nodes from a cluster to agree on a common integer chosen uniformly at random from the interval \((0, r)\). This protocol is secure as long as the malicious nodes are in minority in the cluster. First, each node generates a session pair of cryptographic keys of a semantic cryptosystem and encrypts, using the public session key, an integer chosen uniformly at random from \((0; r)\). Each node then commits the encrypted value using a secure broadcast protocol [HZ10]. Once all the commitments have been sent-out or after some predefined timeout (to account for malicious nodes who may not participate to the protocol), all committed numbers are revealed by each node securely broadcasting its private key to all other nodes in the cluster. Afterwards, each node can add the numbers revealed modulo \(r\), which generates a number uniformly at random in \((0; r)\). With clusters of size \(O(\log^2 N)\), the communication cost of this primitive is \(O(\log r \log^3 N)\), which is quadratic in the size of the cluster as long as \(r\) remains a constant (its round complexity in our model is \(O(1)\)).

Algorithm 6 Distributed random number generation: randNum.

Require: A cluster \(C\) with a majority of honest nodes and an integer \(r\).

Ensure: The generation of an integer chosen at random from the interval \((0; r)\).

Each node of the cluster \(C\) generates a session pair of cryptographic keys for a semantic cryptosystem.

Each node of \(C\) chooses an integer chosen uniformly at random from \((0; r)\) and encrypts it using its session public key.

Each node of \(C\) broadcasts the encrypted value to all the other nodes of \(C\) using a secure broadcast protocol [HZ10].

Once all encrypted values have been broadcast, each node of \(C\) broadcasts its session private keys to all other nodes of \(C\).

Each node of \(C\) decrypts all the encrypted values received and add them modulo \(r\).

Output the obtained value.

Randomly choosing a cluster. In the protocol, it is required that clusters are chosen uniformly at random. However, each cluster only has a local knowledge (i.e., it only knows a small subset of the clusters). Therefore in order to choose a cluster at random, we perform a CTRW\(^3\) on \(\hat{G}^R\), the graph of clusters. From Theorems 2 and 3, we have \(\lambda_2 \geq I(G)^2 / 2\Delta(G) \geq 1/8\) and \(T_{mix} \leq 8 \log^2 N\). Therefore with high probability, the cluster reached after a CTRW of duration \(8 \log^2 N\) can be considered as chosen uniformly at random from \(\hat{G}^R\). With clusters of size \(O(\log^2 N)\), this primitive called randCl, has a communication cost of \(O(\log^7 N \log \log N)\) with high probability. The number of nodes visited during the walk is \(O(\log^3 N)\) with high probability. At each step a random integer from the range \((0, O(\log N))\) is generated at a cost of \(O(\log^4 N \log \log N)\). The round complexity of this protocol is \(O(\log^2 N)\) with high probability.

\(^3\) Recall that a vertex \(C_i\) of \(\hat{G}^R\) is actually a cluster in \(G\). A node of \(C_i\) participates to the random walk if and only if it receives an identical message from at least half plus one of the nodes of the neighboring cluster from which the random walk comes.
Algorithm 7 Randomly choosing a cluster: \texttt{randCl}.
\begin{algorithm}
\textbf{Require:} A graph connecting clusters each with a majority of honest nodes and an initial cluster $C$.
\textbf{Ensure:} The choice of a cluster chosen uniformly at random among all the clusters.

The nodes from $C$ choose an integer $i$ at random using \texttt{randNum} with the integer $r = d_C$ and set $T = \log^2 n/\lambda_2$.

The nodes from the current cluster $C$ initiate a CTRW by sending a message to all the nodes of the cluster $C'$ for $C$ the $i^{th}$ neighbor of cluster $C$ on the overlay.

$C'$ becomes the current cluster.

\textbf{while} $T > 0$ \textbf{do}

\hspace{1em}The nodes from the current cluster choose a number $U$ from $(0, 1)$ using \texttt{randNum}, and reduce $T$ by $\log(1/U)/d$, for $d$ being the degree of the current cluster.

\hspace{1em}The nodes from the current cluster choose a number $i$ using \texttt{randNum} with the integer $d$ being the limit of the interval.

\hspace{1em}The nodes from the current cluster $C$ send a message to all the nodes of the cluster $C'$, which is the $i^{th}$ neighbor on the overlay of cluster $C$.

\hspace{1em}$C'$ becomes the current cluster.

\textbf{end while}

Output the identity of the current cluster in which the walks has ended.
\end{algorithm}

Exchange of nodes. In order to induce churn, a limited number of clusters exchange their nodes with nodes chosen at random from other clusters. This procedure, which we call \texttt{exchange}, is repeated a polylogarithmic number of times whenever a node leaves or joins the system. More precisely, for each node exchanged from cluster $C$, a random cluster is first picked at random using the primitive \texttt{randCl}.

The chosen cluster, $C'$, is informed via a message that it will receive a given node with id $x$. Subsequently, the cluster $C'$ chooses one of its node (using the primitive \texttt{randNum}) that is sent in replacement of $x$. Once each node to be exchanged have been assigned to a new cluster and that their replacing nodes have been identified, the exchange of the nodes is performed. During this exchange, if $C$ is adjacent to another cluster, the nodes of this cluster are informed of the change (i.e., of the new composition of $C$).

This step is fundamental since a node from a neighboring cluster accepts a message from $C$ if and only if at least half plus one of the nodes of $C$ send it. Therefore before the exchange, the nodes of $C$ send to all the nodes of neighboring clusters a message containing the new composition of the cluster. Clusters exchanging nodes proceed in the same manner. Finally, the new nodes of $C$ are informed by the former nodes of this cluster of the local structure of the overlay (i.e., the direct neighboring clusters of $C$ in the overlay). The round complexity of \texttt{exchange} is $O(\log^3 N)$ with high probability.

Algorithm 8 Exchange of nodes: \texttt{exchange}.
\begin{algorithm}
\textbf{Require:} A graph connecting clusters with a majority of honest nodes and a cluster $C$.
\textbf{Ensure:} All the nodes of $C$ are exchanged with nodes chosen at random.

\textbf{for} nodes $x$ in $C$ \textbf{do}

\hspace{1em}Choose a cluster $C_x$ using \texttt{randCl}.

\hspace{1em}The nodes from $C_x$ choose an integer $i_x$ using \texttt{randNum} with $r = |C_x|$, which corresponds to a node $y_x$ of $C_x$.

\textbf{end for}

\textbf{for} nodes $x$ in $C$ \textbf{do}

\hspace{1em}All the nodes from $C_x$ send a message to all the nodes of the neighboring clusters that $x$ replaces $y_x$.

\hspace{1em}All the nodes from $C$ send a message to all the nodes of the neighboring clusters saying that $y_x$ replaces $x$.

\textbf{end for}
\end{algorithm}

7.2 Interface

We describe in this section the operations used to maintain the overlay. These operations are either called by the nodes upon joining or leaving the network or simultaneously by all the nodes of a cluster that splits or merges.

Join operation. This operation, which is initiated by a node joining the network, is inspired by [Sch05,AS09] (and so is the leave operation). When a node $x$ joins the network, we assume that there is a mechanism that allows it to get in contact with a cluster of the overlay. This cluster chooses another cluster uniformly
at random using $\text{randCl}$ in which $x$ is inserted. Afterwards, the chosen cluster proceeds by inserting $x$ and exchanging all its nodes at random with nodes from other clusters using $\text{exchange}$. The operation of the addition of a node has an overall communication cost of $O(\text{Polylog } N)$.

**Algorithm 9 Join operation.**

**Require:** A node $x$ contacting a cluster $C$ to join the network.

**Ensure:** The preservation of the properties of the overlay and of the clusters.

The nodes of $C$ choose a cluster $C'$ using $\text{randCl}$.

All the nodes from $C'$ add $x$ to their local view of $C'$.

All the nodes from $C'$ send a message to all the nodes from the neighboring clusters informing that $x$ is added to $C'$.

All the nodes of $C'$ send their neighborhood to $x$ using the path used to find $C'$ in $\text{randCl}$.

```plaintext
if $|C'| > kl \log^2 n$ then
    The nodes of $C'$ compute a partition of $C'$ into two parts of roughly the same size using $\text{randCl}$: $C_1$ and $C_2$.
    The nodes of $C_1$ keep their neighborhood.
    Both the nodes of $C_1$ and $C_2$ send a message informing that $C'$ is replaced by $C_1$ to the neighbors of $C_1$.
    The nodes of $C_2$ are given a new neighborhood using $\text{Add}(C_2)$ (Algorithm 3).
end if
```

**Split operation.** This operation is initiated simultaneously by all the nodes of a cluster $C$ if after a join operation, the size of this cluster is larger than $lk \log^2 N$ for some fixed parameter $l$, then $C$ has to be split in two, the old and the new clusters. To achieve this, the nodes of the cluster $C$ generate a random partition of $C$. The old cluster keeps its neighbors in the overlay $\hat{G}^R$, whereas the new cluster is added to the overlay using the operation $\text{Add}$ described in Section 4. This procedure has a global communication of $O(\text{Polylog } N)$ and round complexity of $O(\log^3 N)$.

**Leave operation.** This operation occurs when a node from a cluster $C$ leaves the network or when the other nodes of $C$ detect its disappearance if this node has crashed. The cluster $C$ exchanges all its nodes with nodes chosen at random from other clusters using the primitive $\text{exchange}$. Afterwards, a cluster receiving one or more nodes from $C$ also exchanges all its nodes with other nodes using the same primitive $\text{exchange}$. This process has a communication cost of $O(\text{Polylog } N)$ and round complexity of $O(\log^3 N)$.

**Cluster merging.** This operation is initiated simultaneously by all the nodes of a cluster $C$ containing less than $\frac{k \log^2 N}{l}$ users (for the same fixed parameter $l$ described previously). In this situation, a cluster has to be removed, and moreover on order to be able to apply Theorems 2 and 3, this cluster has to be chosen at random, which is achieved using the primitive $\text{randCl}$. The nodes contained in $C$ proceed as if they were joining the network while the nodes from the chosen cluster $C'$ become the nodes of $C$. In $\hat{G}^R$, $C'$ is removed by using the operation $\text{Remove}$ described in Section 4.

### 8 NOW: Analysis

In this section, we prove that after a polynomial sequence of join and leave operations (some of them inducing some splitting and merging of clusters), each cluster has a majority of honest nodes as long as the fraction of malicious nodes $\tau$ controlled by the adversary is smaller than $(\frac{1}{2\ell^2}) - 2\epsilon$ (for some constant $l > \sqrt{2}$ and $\epsilon > 0$ independent of $n$).

#### 8.1 Status of a cluster after exchange

At each time step, we assume that either a join or leave operation takes place or nothing occurs. These operations may in turn induce the splitting or merging of clusters. A split operation is done directly at

---

4 One can think of $l$ has being equal to 2, but any constant greater than $\sqrt{2}$ (e.g., $l = 1.415$) will also work. This parameter influences the number of split and merge operations in the network. The closest is $l$ to $\sqrt{2}$, the higher is the number of this operations.
Lemma 1 (Majority of honest nodes in a cluster). If a cluster \( C \) has exchanged all its nodes at time step \( t_i \), we have \( \text{Prob}(p^C_{t_i} > l^2\tau(1 + \epsilon)) \leq n^{-\gamma} \), for any positive constant \( \gamma \), as long as the security parameter \( k \) is large enough.

Proof. When a cluster \( C \) exchanges one of its node with another cluster, this cluster is first selected at random and then a node is chosen out of it. In this scenario, the probability of performing an exchange with a malicious node is \( \frac{\sum_{j=1}^{|C|} p^C_j}{\sum_{j=1}^{|C|} p^C_j} \). By assumption, we have \( \sum_{j=1}^{|C|} p^C_j |C_j| t_i \leq \tau n \) and \( (k \log^2 N)/l \leq \sum_{j=1}^{|C|} p^C_j |C_j| t_i \leq k \log^2 N \). Therefore, \( \sum_{j=1}^{|C|} p^C_j \leq \frac{\tau n}{k \log^2 N} \) and \( |C| \geq \frac{n}{k \log^2 N} \).

Setting \( f = l^2\tau \), we have:
\[
\frac{\sum_{j=1}^{|C|} p^C_j}{|C|} \leq l^2\tau = f
\]

Using standard Chernoff bound arguments, we can derive the following result on the number \( X \) of malicious nodes among \( |C| t_i \): \( \text{Prob}(X > (1 + \epsilon)\tau|C| t_i) \leq e^{-\epsilon^2|C| t_i/(2(1+\epsilon/3))} \). Therefore as \( |C| t_i \geq (k \log^2 N)/l \), we have \( P(X > (1 + \epsilon)\tau|C| t_i) \leq N^{-\gamma} \) when \( k \) is sufficiently large.

This lemma is a consequence of the Chernoff bound arguments [HMRAR98] and implies that to obtain a majority of honest nodes in a cluster with high probability, it is sufficient that \( l^2\tau + \epsilon < 1/2 \), which is true by assumption on \( \tau \).

Remark 3 (Increasing the robustness). In order to tolerate a fraction of malicious nodes up to \( 1/2 - \epsilon \), one can bias the CTRW so that the stationary distribution becomes \((|C|)/n\). To realize this, it is sufficient that at each cluster, the counter representing the remaining time of the random walk is decreased by \( \log(1/U)|C|/(d(C) \log^2 N) \), where \( U \) is a number chosen uniformly at random from \((0, 1)\), \( |C| \) is the current size of the cluster and \( d(C) \) its degree in \( G^R \). The convergence speed of this CTRW is determined by the second eigenvalue of the matrix \((L_{ij} \log^2 N)/|C|_{ij})\), in which \( (L_{ij})_{ij} \) is the Laplacian matrix as defined previously in Section 3. However, our results do not give directly an upper bound on this eigenvalue.

8.2 Evolution of the divergence

To summarize, we have seen that each time a cluster exchanges all of its nodes, as long as \( l^2\tau(1 + \epsilon) < 1/2 \), we obtain a majority of honest nodes with high probability in the resulting cluster. We now proceed by
proving that in between two exchanges, this property also holds. To realize this, we focus on a specific cluster $C$ and consider a sequence of $s$ join and leave operations.

When a join operation occurs at time step $t_i$, the cluster in which the new node is added is chosen uniformly at random, and then between $k/(l \log^2 N)$ and $kl \log^2 N$ nodes are exchanged with other nodes that are also chosen at random. A specific cluster $C$, has a probability $1/\#C_{t_i}$ to be chosen for both these two types of events. Similarly, for each leave operation, the cluster $C'$ from which the node has left exchanges between $k/(l \log^2 N)$ and $kl \log^2 N$ nodes with other clusters chosen at random, which results in $O(\log^2 N)$ exchanges. Afterwards, if a cluster was involved in an exchange of nodes with $C'$, then it also exchanges all its nodes.

We now prove that if a cluster has been selected many times for performing a node exchange, then with high probability, it has been selected for a node insertion or a node exchange during a leave operation. We first analyze only sequences of join operations before extending it to sequence of join and leave operations.

Lemma 2 (Probabilistic bound on the number of exchanges after a sequence of join operations). If after a sequence of join operations, a given cluster $C$ is affected by $\Theta(\log^3 N)$ nodes exchanges, then with high probability a node is inserted in $C$.

Proof. Let $T_j^{\text{join}}$ be the number of times that $C$ is chosen for a join operation, and $T_j^{\text{exchange}}$, the number of times it is involved in an exchange. Using standard Chernoff arguments, it is possible to prove that $T_j^{\text{join}}$ and $T_j^{\text{exchange}}$ do not deviate too much from their expected values $\bar{T}_j^{\text{join}}$ and $\bar{T}_j^{\text{exchange}}$. Indeed, $P(\lvert T_j^{\text{join}} - \bar{T}_j^{\text{join}} \rvert > \epsilon \bar{T}_j^{\text{join}}) \leq e^{-\epsilon^2 \bar{T}_j^{\text{join}}/2(1+\epsilon/3)}$, and similarly $P(\lvert T_j^{\text{exchange}} - \bar{T}_j^{\text{exchange}} \rvert > \epsilon \bar{T}_j^{\text{exchange}}) \leq e^{-\epsilon^2 \bar{T}_j^{\text{exchange}}/2(1+\epsilon/3)}$.

For a join operation occurring at time step $t_i$, if cluster $C_j$ is selected, this results in $|C_j|_{t_i}$ exchanges. This induces a maximum of $kl \log^2 N$ exchanges per join operations. In this case, $\bar{T}_j^{\text{exchange}} \leq kl \log^2 N \bar{T}_j^{\text{join}}$, and similarly $\bar{T}_j^{\text{join}} \leq \frac{1}{k \log^2 N} \bar{T}_j^{\text{exchange}}$. After a sequence of $s$ join events, and if $\bar{T}_j^{\text{exchange}} = \Theta(\log^3 N)$, then, $T_j^{\text{join}} = \Theta(\log N)$. Therefore with high probability, every $\Theta(\log^3 N)$ exchanges, there will be a join operation during which the new nodes are inserted in the cluster $C$.

As a consequence of this lemma, we can deduce that it is sufficient to prove that after $\Theta(\log^3 N)$ exchanges, the fraction of malicious nodes has not grown too much, which we will use to prove that it always remain a majority of honest nodes in each cluster.

Lemma 3 (Upper bound on the fraction of malicious nodes after several join operations). Given a cluster $C$ whose fraction of malicious nodes is upper bounded by $\ell^2 \tau (1 + \epsilon)$ (for some constant $\epsilon > 0$ independent of $n$), then with high probability after $\Theta(\log^3 N)$ exchanges, the fraction of malicious nodes in this cluster does not exceed $\ell^2 \tau (1 + 2\epsilon)$.

Proof. A cluster $C$ with a fraction $p$ of malicious nodes has a probability at most $p(1 - f)$ to have this fraction decreased by $1/|C|$, and at least $(1 - p)f$ to have it increased by the same amount. As after the insertion of a node, this fraction is at most $f(1 + \epsilon)$, we now prove that it increases by $\epsilon$ with probability $o(1/N^2)$, for $\gamma$ being arbitrarily large depending on the chosen value of $k$.

The fraction of malicious nodes in the cluster is dominated by the martingale with starting state $f(1 + \epsilon)$, which increases or decreases by $1/|C|$ with probability $f$. With high probability, this martingale will not exceed $f(1 + 2\epsilon)$ after $\Theta(\log^3 N)$ steps (recall that $k \log^2 N)/l \leq |C| \leq kl \log^2 N$). If $k$ is chosen to be large enough and for an arbitrarily large constant $M$, we can derived from Azuma-Hoeffding’s inequality that:

$$\text{Prob}(p^C > f(1 + 2\epsilon)) < e^{-\epsilon^2/\sum_{i=1}^{M} \tau_{\text{exchange}}^i |C|^2}$$

$$\leq e^{-\epsilon^2(k/l)^2 \log^4 N/(M \log^3 N)}$$

$$= e^{-\epsilon^2(k/l)^2 \log(N)/M} = n^{-\gamma}$$
To summarize, we have proved in Lemma 2 and 3 that after a sequence of join operations, the probability of a cluster having a majority of malicious nodes is upper bounded by \( n^{-\gamma} \) (recall that by assumption \( \ell^2 \tau + 2\epsilon < 1/2 \)). Therefore by applying the union bound, we have that all clusters have a majority of honest nodes with probability at least \( 1 - n^{-\gamma+1} \). In the situation in which a node leaves the network, the protocol induces \( c\log^2 N \) exchanges (for some constant \( c \) such that \( k/l \leq c \leq kl \)), and \( c' \log^2 N + 1 \) clusters exchange all their nodes, for \( c' \log^2 N \) being the size of the cluster that the node has left (\( k/l \leq c' \leq kl \)). Therefore the number of nodes being exchanged divided by the number of cluster exchanging all their nodes remains constant (approximately \( c/c' \)), and as a consequence with high probability the fraction of malicious nodes does not increase too much.

**Corollary 1 (Majority of honest nodes in all the clusters).** With high probability, after a number of steps polynomial in \( N \), at each time step, all the clusters have a majority of honest nodes.

**Remark 4 (Limiting the power of the adversary).** Considering an adversary controlling at most a fraction \( 1/r^2 - \epsilon \) of the nodes for some constant \( \epsilon > 0 \) and \( r \geq 2 \) independent of \( n \), it is possible to strengthen Corollary 1 to obtain that in all the clusters the adversary controls at most a fraction \( 1/r \) of the nodes.

### 8.3 Weakening the assumptions

In this section, we discuss how it is possible to weaken some of the assumptions upon which NOW has been analyzed in order to increase the generality of the construction. In particular, we describe how to adapt NOW so that it can tolerate a high number of join and leave operations at each time step and how to discard the assumption that the message delay is not taken into account in the analysis.

**Occurrence of several join and leave operations at a particular time step.** In order to accommodate a high number of nodes joining and leaving at each time step, it is sufficient to consider clusters of larger size. For instance, if one want to be able to cope with \( \log^i N \) nodes joining or leaving the network at each time step for some constant \( i \geq 0 \), it needs to use clusters of size \( \log^{i+1} N \) nodes instead of \( \log^2 N \). As a consequence at each time step, the number of nodes susceptible to leave a cluster is negligible compared to its size. Hence, the adversary cannot control a majority of the nodes of in a cluster by forcing all the nodes from this cluster to leave at the same time step. All the proof that we have developed can also be adapted to cope with this new size of cluster. Moreover, with respect to the operations used by the protocols OVER and NOW, they can all be done in parallel and therefore no further adaptation is required. Finally, the new construction does not impact the round complexity and increases only the communication cost by a polylogarithmic factor.

**Taking into account the delay of message transmission.** Currently, we have analyzed our protocols under the assumption that the computational time required by node and the source-to-end delay for message delivery are null. This assumption can be lift by using the following trick; first the join or leave operations are processed every \( \delta \geq \max(\text{round complexity}) \) steps, in which \( \max(\text{round complexity}) \) stands for the maximal round of the procedures used, which is \( O(\log^3 N) \) in our case. Therefore each \( \delta \) steps, \( \delta \) join and leave operations are performed simultaneously. A node leaving the network will induce a Remove operation when it actually leaves.

**Multiple crashes.** Our protocol tolerates an adversary that an \( \epsilon n \) random nodes crash simultaneously if we suppose that it controls at most \( 1/p \) nodes. This type of crashes can be used to simulate a failure of some critical links in a network. To prove this, we have to show that even after this number of crashes, the adversary cannot gain the lead in a cluster that has kept more than half of its nodes (compared to the original size of the cluster before the crash occurs). Due to Remark 4, we have that with high probability in each cluster there is less than a quarter of the nodes that are controlled by the adversary. Hence, for the adversary to control the majority of the nodes in a cluster, at least half of the nodes of this cluster need to crash. However in this scenario, it remains in total less than half of the nodes from the cluster. Therefore the neighboring cluster will ignore the messages from this cluster and act as if all the nodes of this cluster have crashed and this cluster is dead. Thanks to the properties of our overlay (see
Remark 3), NOW will keep working even after such an important number of crashes. Further remark that the assumption that the crashes are random is necessary as otherwise the adversary can split the honest nodes into disconnected components, which will make NOW (and any other protocol) fail.

9 Applications

NOW and OVER construct and maintain an overlay ensuring that the network is partitioned into clusters each containing a majority of honest nodes in the context of a large scale dynamic system, which closes the following fundamental open problem in distributed computing asked in [KS10]:

“Can we [...] address problems of robustness in networks subject to churn? An idea is to assume that: 1) the number of processors fluctuates between \(n\) and \(\sqrt{n}\) where \(n\) is the size of name space; 2) the processors do not know explicitly who is in the system at any time; and 3) that the number of bad processors in the system is always less than a 1/3 fraction. In such a model, can we 1) do Byzantine agreement; and 2) maintain small (i.e. polylogarithmic size) quorums of mostly good processors?”

Moreover, we believe that these two algorithms can be applied to solve a wide range of other problems in distributed computing. Therefore in this section, we briefly review how to apply our algorithms to obtain efficient and robust algorithms for other distributed tasks in the context of a highly dynamic network.

9.1 NOW-Broadcast

The broadcast problem, first introduced in [PSL80], is one of the fundamental primitive in distributed computing, in the sense that it can be used as a building block for constructing more complex protocols for other tasks. Therefore, the design of efficient protocols solving this problem is of paramount importance.

In a nutshell, the broadcast problem is defined as follows: the sender, a specific node in the network, aims at sending a message to all the nodes in the network such that (a) either all the nodes received the message of the sender and thy are sure that all the messages received are the same (i.e., all the messages received are consistent) or (b) the broadcast is aborted and all the nodes are aware of this issue (this could happens for instance if the sender tries to send different messages to different nodes).

The broadcast problem becomes particularly challenging in the context in which the adversary controls some of the nodes of networks and can make them act maliciously. Most of the current broadcast protocols that can tolerate an adversary controlling a constant fraction of the nodes have a communication cost per bit broadcast that is at least quadratic in the size of the network (compared to linear when there is no adversary). The design of a broadcast protocol with a lower communication cost would increased the efficiency of protocol heavily relying on the access to a broadcast channel has a primitive. However, such a protocol cannot be deterministic. In the following, we will show how to use NOW to design robust and efficient probabilistic broadcast protocols.

Broadcasting in a network composed only of honest nodes. If the network is composed only of honest nodes (i.e., there is no adversary), it is easy to design an efficient broadcast protocol. For instance, if all users can communicate via pairwise channels (i.e., the network is fully connected), it is sufficient for the sender to send the message to all the receivers by using the pairwise channels he shares with them. In the situation in which the nodes are not all directly connected to each others, gossip protocols can be used to have a very efficient protocol provided that the topology of the network as good expansion properties. More precisely, a result of Mosk-Oayama and Shah (Theorem 2 of [MAS06]) bounds the number of rounds required to spread a message in particular network. In order to use this theorem, we need to briefly introduce some new notations. Considering an information-spreading algorithm \(\mathcal{P}\) (e.g., a gossip protocol) specifying a pattern of communication between nodes, for each node \(u\) we denote by \(S_u(t)\) the set of nodes that have the message initially detained by \(u\) at time \(t\). We can now define the \(\delta\)-information-spreading time in the following manner:

**Definition 3 (Information-spreading time [MAS06]).** For \(\delta \in (0, 1)\), the \(\delta\)-information-spreading time of a spreading algorithm \(\mathcal{P}\) is \(T_{\mathcal{P}}^{\delta \text{pr}}(\delta) = \inf\{t : \Pr(\bigcup_{i=1}^{\delta} \{S_i(t) \neq V\}) \leq \delta\}\).
The information-spreading time of a particular algorithm can be upper bounded be relying on the notion of conductance that we defined thereafter.

**Definition 4 (Conductance [MAS06]).** Given a graph $G = (V, E)$, in which $V$ is the set of vertices of the graph and $E$ its set of edges, the conductance of $G$ is defined as $\phi(G) := \min_{S \subseteq V: 0 \leq e(S) \leq e(V)/2} \frac{e(S, \bar{S})}{e(S)}$, in which $e(S)$ stands for the number of edges in $S$ and $e(S, \bar{S})$ is the number of edges from $S$ to $\bar{S} = V \setminus S$.

We can now cite the result (Theorem 2 of [MAS06]) that directly upper bounds the information-spreading time of an algorithm.

**Theorem 5 ([MAS06]).** Given a graph $G$, in which $V$ is the set of vertices of the graph and $E$ its set of edges, there is an information-spreading algorithm $\mathcal{P}$ such that, for any $\delta \in (0, 1)$, it disseminates an information in $T^{\text{spr}}_{\mathcal{P}}(\delta) = O\left(\frac{\log n + \log \delta^{-1}}{\phi(G)}\right)$, for $n$ the current number of nodes in the network.

**Broadcasting using the overlay $\hat{G}^R$.** Thereafter, we describe two protocols that can broadcast messages in a network in the presence of an active adversary controlling a constant fraction of nodes. For an adversary controlling a fraction $\frac{1}{27} \delta$ of the nodes, it is fundamental that the messages exchanged are signed (we guarantee that there is a majority of honest nodes in each cluster). However, if the adversary controls a fraction $\frac{1}{27} \delta$ of the nodes, this assumption can be relaxed and the messages do not need to be signed. This result can be proven by extending Corollary 1 to demonstrate that in this situation there are in each cluster strictly less than one third of malicious nodes (see Remark 4).

**NOW-Broadcast-Local.** Relying on the overlay built by NOW and OVER, it is possible to derive a broadcast protocol with a communication cost of $O(n \log^6 N (\log N + \log \delta^{-1})(\log \log N + M))$ bits for $\delta \in (0, 1)$ and $M$ the size of the message to be broadcasted, by adapting a gossip protocol from [MAS06]. The modifications done to the algorithm are the following:

- When a node $u$ wants to broadcast a message, it first broadcasts it to all the other nodes of its cluster using the secure broadcast protocol from [HZ10], whose complexity is quadratic in the size of the cluster.
- Afterwards, the information-spreading algorithm of [MAS06] is run on the overlay $\hat{G}^R$. When a cluster propagates the message to one of its neighbors, it means that each of its nodes send the message to all the nodes of the selected cluster. If the node of a cluster receives different messages, it performs a majority vote to select the message to keep.

For each round and each cluster, the complexity of this procedure is $O(\log^4 N \log \log N)$ for the random number generation (i.e., with high probability there are at most $\log^4 N$ neighbors for each cluster), and $O(M \log^4 N)$ for forwarding the message $M$ to neighboring clusters. Therefore, the communication cost of this algorithm is $O(\log^3 N + \log^2 N \log \delta^{-1})$. Indeed, the conductance of $\hat{G}^R$ can be bounded by using its maximum degree and its isoperimetric constant $\phi(\hat{G}^R) \geq I(\hat{G}^R)/\Delta(\hat{G}^R)$, for $\delta(\hat{G}^R)$ the maximum degree of $\hat{G}^R$, thus giving $\phi(\hat{G}^R) \geq 1/\log^2 N$. As a consequence, the global communication cost of the algorithm is $O(n \log^6 N (\log N + \log \delta^{-1})(\log \log N + M))$.

**NOW-Broadcast-Global:** While the previous protocol requires only a local knowledge of the network topology (i.e., the node in particular cluster needs only to know the identities of the nodes in its cluster and in the neighboring ones), we consider now the situation in which the sender has a global knowledge of the identities of all the nodes in the network and its topology, in other words that there are channels in-between all pairs of nodes. Using this knowledge, one can design a protocol whose round complexity is lower than the previous one. This protocol works as follow:

- When a node $u$ wants to broadcast a message $M$, it sends it to all the nodes that will be receivers.
- Inside a given cluster $C$, each node of $C$ broadcasts the message it received from $u$ to all the other nodes of the cluster $C$ using a secure broadcast protocol. The message that is received by a majority of nodes is considered as being the message effectively received by the cluster from $u$ (we denote this message by $M_C$).
The same message will be broadcasted to all the nodes of the network with high probability.

O communication cost of knowledge of the network, the algorithm proposed by King and Saya [KS10] can solve this problem with adversary controlling a fraction of the nodes in the network and assuming that each node has a global (honest nodes, the agreement problem can be easily solved by having one of the node in the network e.g., the one with the smallest identifier) sending its input to all the other nodes. In context of an active Byzantine agreement protocol) proceeds as follow:

---

**Algorithm 11 NOW-Broadcast-Local**

**Require:** A network with a partition and an overlay maintained by NOW, and two parameters $\delta \in (0, 1)$ and $c$ that are large enough.

**Ensure:** The same message will be broadcasted to all the nodes of the network with high probability.

The sender sets $t = 0$ and sends its message $M$ together with $t$ using a secure broadcast protocol to all the nodes of its cluster.

```plaintext
while $t \leq c \frac{\log n}{\delta N} + \log \delta^{-1}$ do
    – A node that has received $M$ and $t$ during the previous step updates $t = t + 1$.
    – A cluster $C$ whose nodes have received $M$ and $t$ at the previous step chose a neighboring cluster $C'$ at random using the primitive $\text{randNum}$.
    – All nodes of $C$ send $M$ and $t$ to all nodes of $C'$.
end while
```

– The cluster to which $u$ belongs generates a random string $B$ of $k$ bits at random using the primitive $\text{randNum}$. Afterwards, the cluster computes the hash of length $k$ of the message $M$ concatenated with random string $h(B, M)$. The length of the random string $k$ is related to the security of the protocol as the bigger $k$ is, the more difficult it is for an adversary to generate $h(B, M) = h(B, M')$ for some other message $M'$.

– Afterwards, $u$ calls the NOW-local-broadcast protocol described in the previous section to disseminate $h(B, M)$ along with $B$.

– If a cluster $C$ receives a hash $h(B, M)$ that does not correspond to the hash of the message it received concatenated to $B$ ($h(B, M) \neq h(B, M_C)$), $C$ broadcasts an alarm (also using the NOW-broadcast-local protocol) and the current protocol is aborted.

The communication cost of this broadcast protocol is $O(nM + n \log^6 N (\log N + \log \delta^{-1})(\log \log N))$ and its round complexity is $O(\log^3 N + \log^2 N \log \delta^{-1})$.

---

**Algorithm 12 NOW-Broadcast-Global**

**Require:** A network with a partition and an overlay maintained by NOW, as well as two parameters $\delta \in (0, 1)$ and $c$ that are large enough. All nodes know the identities of the other nodes in the system and have agree on a common cryptographic hash function $h$.

**Ensure:** The same message will be broadcasted to all the nodes of the network with high probability.

The sender $u \in C$ sends its message to all the other nodes of the network.

Each node that has received a message broadcasts it to all the nodes in its cluster.

A node $v \in C'$ performs a majority vote on the messages received from the nodes of its cluster and sets $M_{C'}$ as being the remaining one (note that one can have $M_{C'} = \emptyset$).

The nodes of $C$ generates a random string $B$ of $k$ bits using the primitive $\text{randNum}$.

The nodes of $C$ computes $h(B, M)$.

The sender $u$ calls NOW-Broadcast-Local to broadcast $B$ and $h(B, M)$ to all the other nodes in the network.

A node $v \in C'$ that receives $B$ and $h(B, M)$ verifies whether or not $h(B, M) = h((B, M_C))$. If this equality does not hold, the node broadcasts an alarm to all the nodes of its cluster.

A cluster in which a majority of nodes have send an alarm broadcasts this information using the protocol NOW-Broadcast-Local, and the current protocol NOW-Broadcast-Global is aborted.

---

**9.2 NOW-Agree**

In this subsection, we show how it is possible to leverage on the protocols NOW and OVER to implement an efficient solution to the problem of Byzantine agreement [LSP82]. In a network composed only of honest nodes, the agreement problem can be easily solved by having one of the node in the network (e.g., the one with the smallest identifier) sending its input to all the other nodes. In context of an active adversary controlling a fraction of the nodes in the network and assuming that each node has a global knowledge of the network, the algorithm proposed by King and Saya [KS10] can solve this problem with a communication cost of $O(n \sqrt{m})$. In our adaptation of this algorithm, we suppose that the algorithms NOW and OVER have been run and that one cluster (such as $C_0$ or the cluster that has initiated the Byzantine agreement protocol) proceeds as follow:

---

20
– The nodes of this first cluster run a Byzantine agreement protocol at the level of the cluster.
– This cluster broadcasts the result of the Byzantine agreement to the rest of the nodes of the network by calling the Now-Broadcast-Local described in the previous subsection.

The communication cost and round complexity of this algorithm is equal to the one of the broadcast protocol. When the identity of the cluster initiating the agreement protocol or the cluster with the smallest identifier is not clear, we need a procedure to initiate the agreement protocol on at least one cluster and not too many. By assumption, we know that the size of the network is in between \( \sqrt{N} \) and \( N \). Therefore, in the case for which each node initiates the agreement protocol with probability \( \log N/N \), if after \( \log^4 N \) steps no output is received, it means that no node has initiated the protocol. In this case, each cluster proceeds by initiating the protocol with a probability that is twice the previous one. One can show that this procedure has to be repeated at most \( \log N \) times in order to ensure that each cluster receives at least one output with high probability. In this case, \( O(\log N) \) clusters will have broadcasted a message, which results in the communication cost being increased by a factor \( \log N \) compared to the original broadcast protocol. In order for a node to choose among the multiple outputs it receives, a cluster \( C \) broadcasting a message attaches to it a tag that corresponds to the lowest id of all the ids of the nodes within the cluster \( C \). Therefore, the final output selected by a node is the message received whose attached id is the lowest.

Algorithm 13 Now-Agree

Require: A network with a partition and an overlay maintained by NOW. All nodes have an input bit. A node \( u \in C \) initiates the protocol NOW-Agree.
Ensure: All the honest nodes agree on a bit that was proposed initially by one of them.

The nodes of \( C \) run a Byzantine Agreement protocol among themselves (such as [KS10]) and output \( b \).
The cluster \( C \) broadcasts \( b \) to all the other nodes in the network by using NOW-Broadcast-Local.

9.3 NOW-Aggregate

In [GGH+], a protocol has been proposed to compute aggregate functions in a secure and scalable manner by relying on a ring overlay (i.e., an overlay in which the clusters are organized in a ring). This type of overlay can either be maintained along with \( \hat{G}^R \) during our protocol or computed from scratch when needed. Alternatively, one can construct a binary tree via a breadth-first search started on a chosen cluster (for instance via a Byzantine agreement protocol such as the one described in the previous subsection). Such a protocol would produced a structure of small diameter and therefore it would improved the round complexity of the aggregation protocol. Using the protocol of [GGH+] with clusters of size \( O(\log^2 N) \) (which corresponds to the size of the clusters in our overlay) leads to a communication cost of \( O(nM \log^4 N) \), for \( M \) the maximum size of the aggregate.

If privacy is not a primary concern, in the sense that the adversary can learn information about the input of honest nodes, one can rely on the algorithm proposed in [MAS06] to compute efficiently an aggregation function. Thereafter, we describe the pseudo code of such an algorithm in the situation in which the sum is the aggregation function considered. However, it is straightforward to adapt it to any separable function as defined in [MAS06]. We refer the reader to the original paper for the description of the full algorithm and its detailed analysis.

The propagation of the minimum of each \( W^C_i \), for \( 1 \leq i \leq r \), leads to a communication cost for the broadcast algorithm of \( O(nr \log^6 N(\log N + \log \delta^{-1})(\log \log N + M)) \), for \( M \) the maximum size of the \( W^C_i \).

9.4 NOW-Sample

A peer sampling service provides each node with a sample of nodes picked uniformly at random [JGKVS04]. We believe that one of the most promising applications of our overlay network consists in a protocol providing a peer sampling service in presence of an active adversary, such that each node can draw a fresh
Algorithm 14 NOW-Aggregate

Require: A network with a partition and an overlay maintained by NOW. Each node \( u \) has a positive integer \( y_u \) as input and knows a common integer \( r \) measuring the accuracy of the estimated value obtained.

Ensure: Each node learns an estimate of the sum of all the values, i.e. \( \sum_{v \in V} y_v \).

For each cluster \( C \), the nodes of \( C \) broadcast their input to all the other nodes of \( C \).

Each node of \( C \) computes \( y_C = \sum_{v \in C} y_v \).

The nodes of \( C \) collaboratively generate \( r \) independent random numbers \( W^C_1, \ldots, W^C_r \) using the primitive \( \text{randNum} \), such that the distribution of each \( W^C_l \) is exponential with rate \( y_C \) for \( l = 1, \ldots, r \).

Each cluster broadcasts \( W^C_1, \ldots, W^C_r \) using NOW-Broadcast-Local.

Each node computes \( \tilde{W}^C_l = \min_{C} W^C_l \) for \( l = 1, \ldots, r \).

Each node \( u \) computes \( r \sum_{l=1}^r \tilde{W}^C_l \), which is outputted as the estimate of \( \sum_{v \in V} y_v \).

Algorithm 15 NOW-Sample

Require: A network with a partition and an overlay maintained by NOW, as well as two parameters \( \delta \in (0, 1) \) and a node \( u \) requiring a sample.

Ensure: The node \( u \) receives the id of a node chosen at random.

The node \( u \in C \) broadcasts a message to all the nodes of its cluster requiring a sample.

The cluster \( C \) initiates \( \text{randCl} \), which returns \( C' \).

The cluster \( C' \) selects one of its nodes \( v \) at random using the primitive \( \text{randNum} \).

The cluster \( C' \) sends \( v \) to \( u \) following the path used by \( \text{randCl} \) in a backward manner.

10 Related Work

10.1 Overlays for dynamic networks

Many protocols for constructing and maintaining an overlay over a dynamic network have been proposed in the literature. Thereafter, we give an overview of these protocols and we compare them depending on the assumptions made.

With node joining and leaving at each time step. Several protocols have been proposed to maintain an overlay interconnecting the nodes in a P2P network. Some of these protocols focus on offering efficient routing properties as well as tolerating unexpected crashes such as CAN, Pastry or Tapestry [RFH’01,RD01,ZHS’04]. However, the communication cost of these protocols for maintaining the overlay is rather high. Other protocols have been designed that focus on other characteristics such as SHELL [SS09] that organizes the peers into a heap structure resilient against large Sybil attacks.

In [KSW10], the authors have presented an overlay resilient to an adversary that can force several peers to crash and to join in an arbitrary manner. The number of join and leave operations tolerated at each turn is proportional to the degree of the nodes in the overlay, which can be shown to be optimal. Two different overlays have been proposed, one based on hypercube and the other on “pancake” graph (cf [KSW10] for the definition). The communication cost for maintaining both overlays are quite high as all the nodes of the network exchange messages at each step.

All the protocols previously described construct overlays with a specific structure, while others consider the idea of maintaining unstructured overlays [LS03], which is the approach taken for OVER. The protocol of [LS03] builds an overlay corresponding to an expander graph obtained from the union of several random cycles. This protocol has been further extended and analyzed in [GMS04,AY08].
Maintaining unstructured overlays induces fewer message exchanges as compared to structured ones [RFH+01,RD01,ZHS+04,KSW10] because only a polylogarithmic number of nodes are involved in the communication when a node joins or leaves the network. However, previous protocols [LS03,GMS04,AY08] did not tolerate the crashes of nodes as they require that each node that left performs a special operation.

We could have relied on this protocol, but in order to tolerate the accidental crashes of nodes, we propose OVER to maintain an unstructured overlay based on a random graph from the Erdős-Rényi family. Our procedure induces a small increase in the communication cost compared to previous work, but also provides enhanced robustness against nodes failures (i.e., nodes that leave the network without taking any action). Indeed, in our setting, a node of $\tilde{G}^R$ leaving without further notice results in a loss of $2 \log^2 N$ edges, which does not jeopardize the expansion property of the graph as long as it happens a number of times that is a fraction of the current size of the network. Instead, in the solution based on Hamiltonian cycles, the whole process is endangered since the cycles are usually broken when a node crashes. As a consequence, a graph constructed with OVER is still reliable even after the simultaneous loss of a constant fraction of its nodes, which is not the case for most of the overlays based on previous work [LS03,GMS04,AY08]. However some construction [BBC+06] are able to tolerate a linear number of failures in the sense that a component of linear size still keeps good expansion properties (to be compared with the whole graph when using OVER).

Topological changes. Previous works have studied the influence of different types of churn on the overlay. For instance in [BCF09,KMO11], the authors consider a dynamic network in which the communication links may be modified by the adversary at each time step under some connectivity restrictions. In [APRU11], the authors study the scenario in which at each time step, the adversary can force an important number of nodes to leave the network while other nodes naturally join the network at the same time. These join and leave operations change the topology, but still with the constraint that the size of the network remains constant. Furthermore, the authors assume that the nodes are connected via an expander graph. Depending on whether the adversary has to decided in advance his strategy or not, the authors propose almost everywhere agreement protocols tolerating at each time step a churn of, respectively $O(n)$ and $O(\sqrt{n})$. However, the two main differences with our work are that 1) all the nodes are assumed to be honests (i.e., the adversary is only external) and 2) the nodes are connected via an expander initially by assumption. In contrast, our protocol tolerate an active adversary controlling a constant fraction of the nodes in the network and the expander graph is dynamically constructed.

Tolerating an active adversary. The model of dynamic network that is the closest to our is the one developed by Awerbuch and Scheideler [AS04,Sch05,AS07,AS09]. In their works, these researchers have considered a network in which with nodes join and leave at each time step, with the constraint that the number of nodes in the network is always within a constant factor of the initial size. Their protocols further require that initially the network is composed of only honest nodes and that the malicious nodes start to join the network only after a particular initialization phase has taken place. Within this model, the authors propose a technique to maintain clusters of size $O(\log n)$ composed of a majority of honest ones using a trusted central entity. Our approach improves upon these previous works in several ways as it allows to maintains a partition of the nodes when the size of the network varies polynomially and without relying on any central entity.

10.2 Aggregation

Some problems such as the election problem, which consists in choosing an output based on the vote of all the nodes of the network, can be reduced to the more generic problem of computing the aggregate of the inputs of the nodes in the network. Many efficient distributed protocols exist for networks composed of only honest nodes, such as gossip protocols. For instance, in [KDG03] a protocol has been proposed that converges exponentially fast to the exact value, and [MAS06] proposes a protocol that computes an estimate of the exact value using the property that if we have several random variables $W_i$, each with exponential distribution of rate $\lambda_i$, then the minimum of these random variables has an exponential distribution with rate $\sum \lambda_i$. 

23
In [GGHK10] it is assumed that the nodes controlled by adversary are rational and that they will not misbehave if they are a chance that they will caught. In this setting, the authors proposed a protocol ensuring that with high probability the adversary does not learn information any information on the input of a specific node except from what can de deduced from the aggregate value. This work was further extended in [GGH+] in which the authors proposed a protocol with the same guarantee in presence of a stronger adversary.

10.3 Broadcast

The broadcast problem was introduced in [PSL80] and it is known that broadcast is possible in general if and only if the adversary control strictly less than a third of the nodes in the network and that signatures are necessary to tolerate a larger fraction of malicious users. Current protocols designed to tolerate the presence of an adversary controlling a constant fraction of the nodes have a communication cost that is at least quadratic in the size of the network but some efficiency improvements can be obtained when the message to be broadcasted is long [FH06]. Moreover, most of the secure broadcast protocols assume that the network is fully connected and that each node knows the identities of all the other nodes in the network. While this assumption can be realistic in some scenarios, there are plenty of other settings in which the sender only has a partial knowledge of the network (for instance in P2P network).

The two broadcast protocols that we have designed (NOW-Broadcast-Local and NOW-Broadcast-Global) improve the previous known protocols in several ways. First, the communication cost is lower with $O(n \text{Polylog } n)$ for both protocols. Second, we are the first to propose a broadcast protocol (NOW-Broadcast-Local) tolerating an active adversary in the setting in which the nodes have only a partial knowledge of the network. The main drawback of our protocols is that they can tolerate less malicious nodes that some previous works (namely $\frac{n}{3^2} - \epsilon$ with signatures versus $\frac{n}{3} - 1$; and $(\frac{n}{2^2} - \epsilon)$ without signatures versus $n - 1$) Remark that similarly to other works our protocols are probabilistic [FH06] and not deterministic.

10.4 Byzantine Agreement

Solving deterministically the agreement problem in the presence of an active adversary controlling $m$ malicious nodes when a broadcast channel is not available as a primitive, requires a communication cost of $\Omega(mn)$ bits, since each message should be sent to at least $m + 1$ nodes so that one honest node receive it with certainty. However, if one is willing to go for a probabilistic guarantee on the correctness of the output (instead of a deterministic one), the communication cost can be drastically reduced. We briefly review the main families of techniques addressing this problem available in the literature.

First, the universe reduction technique ([GVZ06]) consists in electing a small subset of nodes to represent the other nodes in a distributed computation. A fundamental requirement is that the proportion of the malicious nodes contained in this set should be very close to the proportion of malicious nodes in the overall system. In [Fei99], a protocol is proposed to select randomly a subset of nodes but it requires the availability of a broadcast channel. Under this assumption, the communication cost of the protocol is $O(n)$. However, simulating such a broadcast channel costs $O(n^2)$, and thus leads to a global cost of $O(n^3)$, which becomes impractical for large-scale systems. Alternative protocols by [KKK+10] and [KS10] have a sub-quadratic communication complexity, but on the other hand assume that each node has a global knowledge of the identities of all the other nodes in the system, which itself hides an $\Omega(n^2)$ communication complexity. Moreover, it is known that Byzantine agreement is possible with a communication cost of $O(n \log n)$ in the setting of global knowledge when the adversary can only send a limited number of messages, or $O(n \sqrt{n})$ when the protocol is balanced (i.e., same number of messages received and sent by each node) and that there is no restriction on the power of the adversary [KS10].

Another solution to solve the Byzantine agreement problem is to use an overlay partitioning the nodes such that each cluster of the partition contains a majority of honest nodes such as in [AS04, Sch05, AS07, AS09] which we mentioned previously in Section 10.1. This is the approach taken in this paper, which has lead to the design of NOW-Agree that has a communication cost of $O(n \text{Polylog } n)$ and does not require the assumption of global knowledge, thus decreasing the communication cost compared to previous works while weakening the assumptions at the same time.
10.5 Peer Sampling

A peer sampling service provides each node with a sample of nodes identifiers picked uniformly at random. Such a service can be implemented easily using gossip-based protocols or random walks techniques when all the nodes are honest and acts as a basic building block for many contemporary distributed applications. While this problem has been widely studied in systems composed of honest nodes, it is challenging to solve it in dynamic network in the presence of an active adversary. One of the few protocol that has addressed this issue is Brahms [BGK+09]. However, this protocol accounts only for an adversary that has a limited power (typically assuming that each node can send a limited number of messages). In addition, the analysis is also restricted to two types of attacks, one that aims at splitting the honest nodes in two disconnected components, and the second that aims at over-representing the nodes controlled by the adversary in the sample.

NOW-Sample improves on Brahms on several aspects. First, NOW-Sample requires less memory at each node ($O(\log^6 N)$ with high probability) since each node of the overlay has maximum degree $\log^4 N$ and that each cluster contains $O(\log^2 N)$ nodes. Second, NOW-Sample is resilient to all possible misbehaviors of malicious nodes, e.g. the number of messages sent by the adversary is not limited. Furthermore, as the adversary controls at most a fraction $\tau \leq \frac{1}{2} - \epsilon$ of the nodes for some $\epsilon > 0$ and $l \leq \sqrt{2}$, our solution guarantees that with high probability there will be a majority of honest nodes in the sample. In order to tolerate more malicious nodes, one could do as suggested in Remark 3, although the resulting random walk would be longer.

11 Conclusion

In this paper, we propose a novel protocol called OVER, that allows to dynamically maintain a graph with good expansion properties and low degree under a high level of churn. We think that this protocol is interesting in its own right as it allows to create a graph with good resilience properties. Afterwards we have introduced NOW, which is based on OVER and can be used to maintain a partition of the nodes of a dynamic network into small clusters such that each of them contain a majority of honest nodes.

We have illustrated the usefulness of NOW by showing how it can be used to solve efficiently some fundamental problems of distributing computing. In particular, we proposed NOW-Broadcast (Local and Global), NOW-Aggregate, NOW-Agree and NOW-Sample and each of these algorithms improve significantly on the best known state-of-the-art algorithms for these problems. The following table summarizes the communication cost and round complexity of these different protocols.

| Protocol                | Communication cost                                                                 | Round complexity |
|-------------------------|-----------------------------------------------------------------------------------|------------------|
| NOW-Broadcast-Global    | $nM + \tilde{O}(n(\log N + \log \delta^{-1}) (h(B.M) + k))$                      | $\tilde{O}(\log N + \log \delta^{-1})$ |
| NOW-Broadcast-Local     | $\tilde{O}(n(\log N + \log \delta^{-1}) (\log \log N + M))$                     | $\tilde{O}(\log N + \log \delta^{-1})$ |
| NOW-Agree$^7$          | "                                                                                 | "                |
| NOW-Aggregate$^8$       | $O(nr(\log N + \log \delta^{-1}) (\log \log N + M))$                           | "                |
| (not privacy preserving)|                                                                                  |                  |
| NOW-Aggregate$^{privacy preserving}$ | $\tilde{O}(nM)$                  | $\tilde{O}(n)$  |
| NOW-Sample (per sample) | Polylog $N$                                                                       | Polylog $N$     |

This work opens several new perspectives and avenues of research. In particular, we are especially interested in the problem of dealing with the stronger adversary model in which the adversary can be

$^5 M$ represents the maximal size of the output on which the nodes need to agree.

$^6 M$ represents the maximal size of the aggregate.
adaptive. Coping with such an adversary will necessitate to rethink the strategy that inhibits the behavior of malicious nodes. In particular, under such an hypothesis, it is useless to partition the nodes into clusters because as soon as the partition is computed, the adversary can choose to corrupt all the nodes from a given cluster. Another fundamental question is whether or not it is possible to devise a procedure for the initialization phase of NOW with a cost of only $O(n^2_t)$ (as opposed to $O(n^3_t)$ in our case). Finally, it would interesting to consider the situation in which nodes are asynchronous.

References

[AF02] D. Aldous and J. Fill. Reversible markov chains and random walks on graphs. http://stat-www.berkeley.edu/users/aldous/RWG/book.html, 2002.
[APRU11] J. Augustine, G. Pandurangan, P. Robinson, and E. Upfal. Towards robust and efficient computation in dynamic peer-to-peer networks. *Arxiv preprint, arXiv:1108.0809*, 2011.
[AS04] B. Awerbuch and C. Scheideler. Group spreading: A protocol for provably secure distributed name service. *Theory of Computing Systems*, 39(6):903–928, 2006.
[BCF09] H. Baumann, P. Crescenzi, and P. Fraigniaud. Parsimonious flooding in dynamic graphs. In *Proceedings of the 28th ACM symposium on Principles of distributed computing (PODC’09)*, pages 260–269, 2009.
[BGK+09] E. Bortnikov, M. Gurevich, I. Keidar, G. Kliot, and A. Shraer. Bramhs: Byzantine resilient random membership sampling. *Computer Networks*, 53(13):2340–2359, 2009.
[Fei99] U. Feige. Noncryptographic Selection Protocols (Extended Abstract). In *40th Annual Symposium on Foundations of Computer Science (FOCS’1999)*, page 142, 1999.
[FH06] M. Fitzi and M. Hirt. Optimally efficient multi-valued byzantine agreement. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing (PODC’06)*, pages 163–168, 2006.
[GGH+*11] S. Gambs, R. Guerraoui, H. Harbous, F. Huc, and A-M. Kermarrec. Secure and Scalable aggregation in dynamic networks. In *submitted*.
[GGHK10] A. Giurgiu, R. Guerraoui, K. Huguenin, and A-M. Kermarrec. Computing in Social Networks. *Lecture Notes in Computer Science*, 6366:332–346, 2010.
[GKLMM07] A. Ganesh, A.-M. Kermarrec, E. Le Merrer, and L. Massoulié. Peer counting and sampling in overlay networks based on random walks. *Distributed Computing*, 20:267–278, 2007.
[GMS04] C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks. In *23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM’2004)*, volume 1. IEEE, 2004.
[GMT05] A. Ganesh, L. Massoulié, and D. Towsley. The effect of network topology on the spread of epidemics. In *24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM’2005)*, volume 2, pages 1455–1466, 2005.
[GVZ06] R. Gradwohl, S. Vadhan, and D. Zuckerman. Random selection with an adversarial majority. *Advances in Cryptology-CRYPTO 2006*, pages 409–426, 2006.
[HMRAR98] M. Habib, C. McDiarmid, J. Ramirez-Alfonsin, and B. Reed. *Probabilistic Methods for Algorithmic Discrete Mathematics*. Springer Verlag, Berlin, 1998.
[HZ10] M. Hirt and V. Zikas. Adaptively secure broadcast. *Advances in Cryptology EUROCRYPT 2010*, pages 466–485, 2010.
[JGKVS04] M. Jelasity, R. Guerraoui, A.M. Kermarrec, and M. Van Steen. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In 5th ACM/IFIP/USENIX international conference on Middleware, pages 79–98, 2004.
[KDG03] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *44th Annual IEEE Symposium on Foundations of Computer Science (FOCS’03)*, pages 482–491, 2003.
[KKK+10] B.M. Kapron, D. Kempe, V. King, J. Saia, and V. Sanwalani. Fast asynchronous byzantine agreement and leader election with full information. *ACM Transactions on Algorithms (TALC)*, 6(4):1–28, 2010.
[KMO11] F. Kuhn, Y. Moses, and R. Oshman. Coordinated consensus in dynamic networks. In *Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 1–10, 2011.
[KS10] Valerie King and Jared Saia. Scalable Byzantine Computation. *ACM SIGACT News*, 41(3):89–104, 2010.
[KS11] M. Krebs and A. Shaheen. *Expander Families and Cayley Graphs: A Beginner’s Guide*. Oxford University Press, USA, 2011.
[KSW10] F. Kuhn, S. Schmid, and R. Wattenhofer. Towards worst-case churn resistant peer-to-peer systems. *Distributed Computing*, 22(4):249–267, 2010.
[Lin02] T. Lindvall. *Lectures on the coupling method*. Dover, 2002.
[LS03] C. Law and K.Y. Siu. Distributed construction of random expander graphs. In 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM’2003), pages 2133–2143, 2003.

[LSP82] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. ACM Transactions on Programming Languages and Systems (TOPLAS), 4(3):382–401, 1982.

[MAS06] D. Mosk-Aoyama and D. Shah. Computing separable functions via gossip. In Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing (PODC ’06), pages 113–122, New York, NY, USA, 2006.

[MLMKG06] L. Massoulié, E. Le Merrer, A-M. Kermarrec, and A. Ganesh. Peer counting and sampling in overlay networks: random walk methods. In Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing, pages 123–132. ACM, 2006.

[PSL80] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. Journal of the ACM, 27(2):228–234, 1980.

[RD01] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In Middleware 2001, pages 329–350. Springer, 2001.

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In Conference on Applications, technologies, architectures, and protocols for computer communications, pages 161–172, 2001.

[Sch05] C. Scheideler. How to spread adversarial nodes? Rotate! 37th annual ACM symposium on Theory of computing (STOC’05), page 704, 2005.

[SS09] C. Scheideler and S. Schmid. A distributed and oblivious heap. Automata, Languages and Programming, pages 571–582, 2009.

[VR03] R. Van Renesse. The importance of aggregation. In Future Directions in Distributed Computing, pages 87–92, 2003.

[ZHS+04] B.Y. Zhao, L. Huang, J. Stibbling, S.C. Rhea, A.D. Joseph, and J.D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. IEEE Journal on Selected Areas in Communications, 22(1):41–53, 2004.