Mind the Gap: Autonomous Detection of Partitioned MANET Systems using Opportunistic Aggregation

Simon Bouget, Yérom-David Bromberg, Hugues Mercier, Etienne Rivière, François Taïani

Mind the Gap: Autonomous Detection of Partitioned MANET Systems using Opportunistic Aggregation

Simon Bouget*, Yérom-David Bromberg*, Hugues Mercier†, Etienne Rivière‡ and François Taiani*

*University of Rennes, Inria, CNRS, IRISA, France;
†University of Neuchâtel, Switzerland; ‡Université catholique de Louvain, Belgium
first.last@{irisa.fr,unine.ch,uclouvain.be}

***Abstract*—Mobile Ad-hoc Networks (MANETs) use limited-range wireless communications and are thus exposed to partitions when nodes fail or move out of reach of each other. Detecting partitions in MANETs is unfortunately a nontrivial task due to their inherently decentralized design and limited resources such as power or bandwidth. In this paper, we propose a novel and fully decentralized approach to detect partitions (and other *large* membership changes) in MANETs that is both accurate and resource efficient. We monitor the current composition of a MANET using the lightweight aggregation of compact membership-encoding filters. Changes in these filters allow us to infer the likelihood of a partition with a quantifiable level of confidence. We first present an analysis of our approach, and show that it can detect close to 100% of partitions under realistic settings, while at the same time being robust to false positives due to churn or dropped packets. We perform a series of simulations that compare against alternative approaches and confirm our theoretical results, including above 90% accurate detection even under a 40% message loss rate.**

I. Introduction

Mobile Ad-hoc Networks (MANETs) do not rely on any fixed network infrastructure, but instead exploit mesh networking protocols [1], [2] to overcome the mobility of nodes and the imperfect and unpredictable nature of wireless communications. Because they operate in open, dynamic and sometimes hostile environments, MANETs can easily become *partitioned*, for instance because some key nodes have failed, because nodes have moved out of reach from one another, or because environmental conditions hinder wireless communications. When this occurs, the network becomes disconnected, greatly hampering its overall operation.

A partition can be mitigated using different techniques. For instance, the transmission range of the wireless nodes may be increased; some of the nodes might fall back on an alternative wireless technology (e.g. a broadband cellular network) [3]; or the system might leverage additional resources such as a supporting Flying Ad-Hoc Network (FANET) [4], [5] to extend the routing coverage [6].

Before any explicit mitigating action can be taken, however, the partition must first be *detected*. Detecting a partition in a MANET is unfortunately a challenging task: MANETs typically lack any central element, forcing each node to build its own perception of the network's current state. This decentralized monitoring must also remain extremely lightweight in order to meet the memory, CPU, and energy constraints of mobile nodes.

Previous proposals to detect partitions in MANETs have either assumed extended node capabilities [7], [8] (such as GPS sensors or accelerators), thus limiting their applicability to high-end deployments, or have attempted to construct explicit membership and path information [9]–[11]. As MANETs reach several hundreds of nodes, gathering explicit node lists is increasingly problematic: explicit representations incur important communication costs and lead to a rapid depletion of energy resources.

In this work, we tackle partition detection in MANETs by coupling a *probabilistic compact representation* of a network's composition with a *periodic and opportunistic aggregation procedure* inspired by gossip protocols [12], [13] and designed to piggyback on existing communications required by routing protocols. These two primitives in tandem allow us to arbitrate the inherent tradeoffs arising between speed, accuracy, and cost of the detection. They offer an adaptable range of guarantees tailored to each system's requirements in a lightweight, decentralized and accurate fashion. We instantiate them so that partitions can either be self-detected by a MANET by identifying temporal discrepancies, or detected by a second external network by monitoring spatial discrepancies. The choice between these two use cases depends on whether mitigating measures should be triggered by the partitioned network (such as switching to an alternative wireless technology [3]), or by some external system (such as offering bridging capabilities by a FANET [4], [6]).

More precisely, we make the following contributions:

1) We use random bit signatures to concisely encode a MANET's set of nodes, or *membership list* into a *filter*, and show that this compact and stochastic representation can detect large connectivity changes.
2) We present two partition-detection algorithms that combine our probabilistic representation with a periodic aggregation procedure. Our algorithms provide an internal self-detection mechanism (in which a MANET A detects it has partitioned), and an external third-party detection service (in which a system B detects that a MANET A has partitioned).

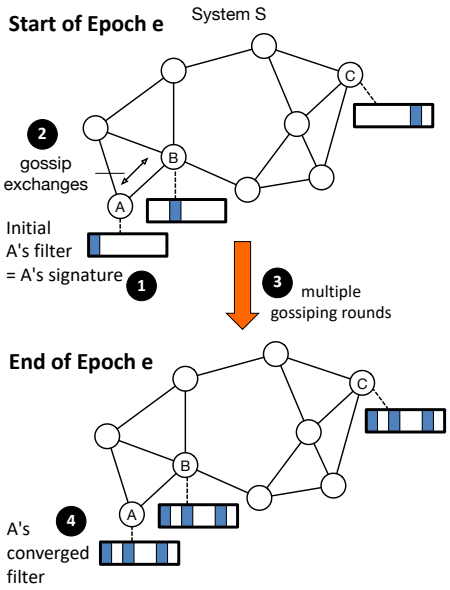

Figure 1. Starting from their individual signature, nodes progressively aggregate other nodes' signatures in their local filter. At an epoch's end, they converge to a summary representing the composition of the subnetwork they were able to hear from (shown only for the nodes A, B, C for simplicity).

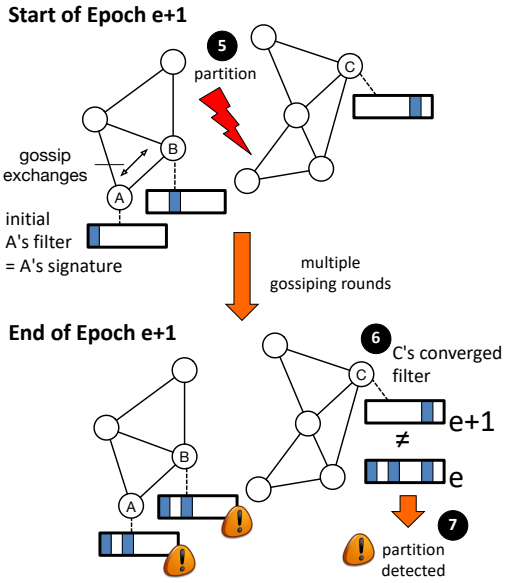

Figure 2. When a partition occurs, the summaries between two successive epochs change suddenly, as the signatures of unreachable nodes are no longer aggregated in the converged summary. This sudden change can be detected, and a partition detection event raised.

3) We develop a theoretical analysis to tune implementation parameters based on the environment in which the network is deployed. We show that as long as the filters are not completely saturated with '1's , we detect close to 100% of partitions and almost none of the non-partition events.
4) We demonstrate the practical relevance of our approach through an extensive series of simulations. As an example, we show that even with 40% message loss, performances are still satisfying with an error rate below 10%.

In the following, we present our approach in Section II, analyze it formally in Section III, and present an in depth experimental evaluation in Section IV. We discuss related work in Section V and conclude in Section VI.

II. Approach

We start with a general overview of our approach, and then detail the two variants of our partition detection mechanisms.

A. Overview

A simple approach to partition detection consists in maintaining and propagating a list of a system's connected nodes [9]–[11]. If two lists for the same system are sufficiently different, the system is likely partitioned. Unfortunately, this direct approach is, in most cases, not tractable: it incurs a high overhead in large systems, both in terms of memory usage, bandwidth consumption, and hence energy consumption, a prime limiting factor in MANETs.

To overcome this difficulty, our approach (which we term *MtG* for *Mind the Gap*, as partitions create gaps in connectivity) replaces the explicit representation of reachable nodes by a potentially inaccurate but compact *summary* of a system's connectivity. More precisely, each node of a system repeatedly constructs a summary of the currently reachable network, and nodes infer a partition when two summaries about the same system differ markedly. In practice, this will detect any large change in system's connectivity, not only partitions, but non-partition events often need to be addressed in the same way and can be easily filtered out with minor changes we won't detail here.

1) Constructing summaries: To be practical and scalable, summaries should ideally be accurate, compact, and robust to network delays and interference. Our approach uses fixed-size bit arrays, termed *filters*, which we construct over recurring periods of times, called *epochs*, using a wireless gossip aggregation procedure. This aggregation procedure is designed to leverage existing periodic communications in the MANET system, such as the periodic *beacon* messages employed by many MANET routing protocols [2]. The size of the filters is fixed in advance and uniform in the system, but only requires a rough estimate of the network size (see Section III).

Figure 1 illustrates this construction in a small network of nine nodes. Upon initialization, each node is assigned an initial bit array (the node's *signature*) of the size of the summaries to be constructed. This initial filter contains only unset bits except for one of them. The set bit is selected uniformly at random when a node is configured. [1]

When an epoch starts, nodes initialize their local filter with their signature (Label ❶ in Fig. 1, here shown for nodes A, B, C). They then broadcast their current filter (Label ❷), and aggregate the filters received from other nodes with

[1]Note we could have used a hash function on the node identifier to derive a node's signature, in effect constructing a Bloom filter with a single hash function [14]. Bloom filters offer additional but unneeded features, so an initial random bit is both simpler and sufficient.

Table I
NOTATIONS AND VARIABLES

Constants and functions

Δ_{epoch}	Duration of an epoch.
γ	Threshold used to detect a partition.
f	Size of the bit arrays
$\text{hdist}(s_1, s_2)$	Hamming distance between the bit arrays s_1 and s_2.
$\text{PARTITION}(i, e)$	Event representing a partition in system i at epoch e.

Variables maintained by a node p_i in a monitored system

$sysID_i$	The ID of the system the node p_i belongs to.
clock_i	p_i's local clock.
$epoch_i$	p_i's current epoch number.
$node\_sig_i$	The one-bit signature of p_i.
$filter_i$	The system summary being constructed by p_i
$sum_i[]$	An array of the system summaries observed by p_i at the end of each past epoch, indexed by epoch numbers (used for self-detection)

Variables maintained by a node p_i in a monitoring system

$sumSet_i$	The set of summaries propagated to the monitoring node. $(id, ep, s) \in sumSet_i$ means that a node from system id generated a system summary s at the end of epoch ep, and that p_i is aware of this summary.

Algorithm 1: MtG/Self-detect: Filter aggregation (at p_i)

```
every X seconds do
    BROADCAST AGG⟨sysID_i, epoch_i, filter_i⟩
on receive AGG⟨sysID, epoch, filter⟩ do
    if epoch_i = epoch and sysID_i = sysID then
        filter_i ← OR(filter_i, filter)
```

Algorithm 2: MtG/Self-detect: Change of epoch (at p_i)

```
on expiration EPOCH_TIMER do
    sum_i[epoch_i] ← filter_i
    for t ∈ 0..epoch_i − 1 do
        if hdist(sum_i[epoch_i], sum_i[t]) > γ then
            raise PARTITION(sysID_i, epoch_i)
    filter_i ← node_sig_i                    ▷Resetting p_i's filter
    epoch_i ← ⌊clock_i / Δ_epoch⌋            ▷New epoch
    set timer EPOCH_TIMER at (epoch_i + 1) × Δ_epoch
```

OR operations on each bit. This procedure is repeated over multiple asynchronous rounds during an entire epoch (Label ❸). Eventually, provided the system is connected and the epoch is long enough, each node converges to a summary of the currently reachable network [12], [13]. This summary contains the bit-signatures of all participants the local node was able to hear from (Label ❹, for clarity the figure only shows the signatures of nodes A, B, C). In the following we often use the words *filter* and *summary* interchangeably, privileging *summary* when denoting the state of a filter at the end of an epoch.

2) Detecting partitions: The system summaries constructed by individual nodes can be exploited in two ways: *(i) Self-Detection* to detect partitions from within a partitioned system, and *(ii) Assisted-Detection* to detect the partition of a monitored system from an external monitoring system.

Self-detection is illustrated in Figure 2. Suppose that a partition occurs just after the construction of the summaries of Figure 1 when Epoch e ends (Label ❺). The summaries constructed by each node during Epoch $e+1$ will therefore only encompass signatures from its connected subnetwork (Label ❻). The summary obtained by Node C for Epoch $e+1$ will not contain the signatures of A or B. This summary will thus differ sufficiently from that of Epoch e, allowing C to detect a partition (Label ❼).

Assisted-detection works along the same lines but involves an external monitoring system, and uses discrepancies in space rather than in time. Both types of detection are presented more formally in Sections II-B and II-C.

3) Parameter tradeoffs: So far, we have assumed that the construction of summaries was done perfectly. In practice, two summaries of the same system might diverge for other reasons than a partition. First, individual nodes might crash, leading to small changes in individual summaries. Second, filters might propagate imperfectly over an epoch due to network failures or if an epoch is too short with respect to the network diameter. Such imperfect propagation will lead to variations in the summaries constructed by a same node over successive epochs (used for self-detection), and by different nodes over the same epoch (used for assisted-detection).

Another cause of inaccuracy stems from the compact nature of node signatures and summaries. Signatures can collide, and a partition might only cause small changes in a network's summaries. Consider, in the worst case, a large network using small filters. It is highly probable that all the bits of the summaries will be set to one before and after a large partition. Using a very large filter solves this problem but is a waste of precious resources. The size of system summaries, the length of an epoch, the frequency of gossiping rounds, and the threshold used to detect partitions must therefore be selected with care, depending on the size, dynamism, memory and energy constraints of the system.

B. Self-detection protocol (MtG/Self-detect)

The first way we use summaries is to allow a system to monitor its own evolution over time and detect when it becomes partitioned, an ability we have termed *self-detection*. The protocol implementing this behavior, called *MtG/Self-detect*, is described by Algorithms 1 and 2. Table I provides a summary of the variables and notations used.

When a node p_i starts participating to the system, it does not take part in the current epoch (its $epoch_i$ variable is set to $\perp$), and waits for the next epoch to start at time $\left(\left\lfloor \frac{\text{clock}_i}{\Delta_{\text{epoch}}} \right\rfloor + 1\right) \times \Delta_{\text{epoch}}$ before joining the protocol, where Δ_{epoch} is the duration of an epoch, and clock_i represents p_i's local clock. We assume that clocks are loosely synchronized between all nodes, with a drift remaining small compared to the duration of an epoch Δ_{epoch}, so that all nodes in the MANET work for the same epoch during a large fraction of Δ_{epoch}. Using a date instead

Algorithm 3: MtG/Assisted: Change of epoch at a node p_i belonging to a monitored system

14 **on expiration** EPOCH_TIMER **do**
15 BROADCAST SUMMARY$\langle sysID_i, epoch_i, filter_i \rangle$
16 $filter_i \leftarrow node\_sig_i$ ▷*resetting p_i's filter*
17 $epoch_i \leftarrow \left\lfloor \frac{\text{clock}_i}{\Delta_{\text{epoch}}} \right\rfloor$ ▷*New epoch*
18 **set timer** EPOCH_TIMER **at** $(epoch_i + 1) \times \Delta_{\text{epoch}}$

Algorithm 4: MtG/Assisted: Signature aggregation at a node p_i belonging to a monitoring system

19 **on receive** SUMMARY$\langle sysID, epoch, filter \rangle$ **do**
20 **if** $sysID_i \neq sysID$ **then**
21 $sumSet_i \leftarrow sumSet_i \cup \{(sysID, epoch, filter)\}$
22 CHECKPARTITION()

23 **every** Y seconds **do**
24 BROADCAST SUMMARY_SET$\langle sysID_i, sumSet_i \rangle$

25 **on receive** SUMMARY_SET$\langle sysID, sumSet \rangle$ **do**
26 **if** $sysID_i = sysID$ **then**
27 $sumSet_i \leftarrow sumSet_i \cup sumSet$
28 CHECKPARTITION()

29 **procedure** CHECKPARTITION() **is**
30 $P \leftarrow \left\{ (i, e, s_1, s_2) \mid \begin{array}{r} \text{hdist}(s_1, s_2) > \gamma \ \wedge \\ \{(i, e, s_1), (i, e, s_2)\} \subseteq sumSet_i \end{array} \right\}$
31 **For all** $(i, e, s_1, s_2) \in P$ **do**
32 $sumSet_i \leftarrow sumSet_i \setminus \{(i, e, s_1), (i, e, s_2)\}$
33 **raise** PARTITION(i, e)

of a duration for our timers ensures we leverage that loose synchrony and that the drifts don't accumulate over multiple epochs. The code of the joining mechanism is not shown in the presented pseudo-code for the sake of clarity.

We assume that nodes know the identifier of the system they belong to, and that they can ignore messages sent by nodes belonging to other systems, if necessary. When a node actively participates in an epoch, it periodically broadcasts its current filter (lines 1-2 of Alg. 1), typically by leveraging existing periodic beaconing messages used by the routing protocol. When a node receives a neighbor's filter for the system it belongs to, and for the epoch it currently participates in (lines 3-4), it incorporates the received filter into its own filter by using a logical OR over the two bit fields (line 5). Otherwise, the message is simply dropped. This simple process implements a push-based aggregation [12], and is robust and efficient.

At the end of an epoch, each node stores its final filter as the system summary for this epoch (line 7 of Alg. 2) and compares it to prior summaries by calculating the Hamming distance between the two filters ($\text{hdist}(-,-)$, line 9), i.e. counting the number of bits in which they differ. The current filter is then reset (line 11) and a new aggregation epoch starts (lines 12-13).

If there is "enough" difference between filters (using the threshold γ at line 9), this is a sign of significant change in the MANET over the corresponding epoch and a partition is detected (with the PARTITION event at line 10). The main difficulty, and an important contribution of this work, is to determine the proper threshold parameter γ to detect actual partitions while avoiding false positives, a point we revisit in detail in our analysis of Section III.

C. Assisted-detection protocol (MtG/Assisted)

As an alternative to self-detection, summaries can also be used to provide *assisted-detection*, i.e. the detection by a monitoring MANET that the MANET being monitored has partitioned. We assume the monitoring system is strongly connected and does not suffer from the same issues as the monitored system. The corresponding protocol (termed *MtG/Assisted*) is detailed in Algorithms 3 and 4. The main idea consists in propagating *within* the monitoring MANET summaries constructed by nodes of the monitored system. This propagation makes it then possible to detect whether two nodes from the monitoring system have observed a large enough discrepancy in two summaries of the same monitored system for the same epoch, hinting at a partition.

More specifically, nodes in the target system execute the same aggregation gossip as previously (cf. Alg. 1). At the end of an epoch, however, the nodes do not store the filter but instead broadcasts them with a SUMMARY message (line 15 in Alg. 3).

When a node from the monitoring system receives a SUMMARY message (line 19 in Alg. 4), it stores it (line 21). The monitoring system then executes its own gossip aggregation *of the target system's signatures* with SUMMARY_SET messages (lines 23-28 in Alg. 4). When a node receives a SUMMARY_SET message, it stores the new summaries (line 27) and then checks if it has two different enough summaries from the same epoch and for the same target system, indicating a potential partition (procedure CHECKPARTITION(), lines 29-33). Note that a more optimized and energy-aware variant could reduce the number of calls to CHECKPARTITION() and make sure that any partition event is raised only once.

III. ANALYSIS

To understand the behavior of summaries and therefore the MtG protocols, we first discuss how the threshold parameter γ should be set assuming ideal network conditions (Sec. III-B), before proposing a more in-depth analysis when considering an imperfect and hostile but realistic network (Sec. III-C).

A. Overview: The role of γ

Both MtG/Self-detect and MtG/Assisted use the parameter γ to decide when the hamming distance $\text{hdist}(-,-)$ between two summaries is large enough to denote a partition with high probability. The difference between the two variants stems from how the two compared summaries are obtained. In MtG/Self-detect, the two summaries $sum_i[e]$ and $sum_i[e+1]$

are computed by the same node n_i over two consecutive epochs e and $e+1$ (line 9 in Algorithm 2), whereas in MtG/Assisted, $sum_i[e]$ and $sum_j[e]$ are computed by two different nodes n_i and n_j over the same epoch e.

In both cases the threshold parameter γ captures the trade-off between false positives and false negatives: a low γ will cause both protocols to react to small changes in summaries, decreasing the risk of missing actual partitions (false negatives), but increasing the risk of raising an alert when no partition has occurred (false positives). A high γ has a reverse effect.

The most challenging situation is when the two summaries sum_1 and sum_2 represent two sets of nodes S_1 and S_2 that have similar sizes. They are more likely to have bits in common, and hence a low Hamming distance. This situation is more likely to occur with the MtG/Assisted protocol, when the same network S becomes split into two equal parts S_1 and S_2.

To investigate this case, we will consider the following scenario. A monitoring node n_{mon} receives two summaries sum_1 and sum_2 for the same system i and same epoch e (i.e., (i, e, s_1, sum_1) and (i, e, s_1, sum_2) both belong to P at line 30 of Algorithm 4). Let us note S_1 the connected (sub)network from which sum_1 originated, and S_2 the one of sum_2. n_{mon} must decide based on the hamming distance between sum_1 and sum_2 whether they come from the same connected network ($S_1 = S_2 = S$, no partition), or if they originate from two disconnected subnetworks ($S_1 \neq S_2$, there is a partition).

B. Operating in paradise: ideal network conditions

We first consider the case where network conditions are ideal, i.e., there is no churn, no noise, and no packet loss. If the duration of an epoch Δ_{epoch} is sufficiently long, all the summaries produced with a connected (sub)network are identical at the end of each epoch.

In this scenario, $S_1 = S_2$ implies that $\text{hdist}(sum_1, sum_2) = 0$, and therefore, $\text{hdist}(sum_1, sum_2) > 0$ implies $S_1 \neq S_2$. In other words, under these ideal conditions, choosing $\gamma = 0$ in Algorithm 4 guarantees all PARTITION events raised by n_{mon} correspond to an actual partition, i.e., there cannot be any false positive. The same reasoning applies to Algorithm 2.

$\text{hdist}(sum_1, sum_2) = 0$ does not imply, however, that $S_1 = S_2$. sum_1 and sum_2 may still originate from different subnetworks due to collisions in the random choice of the original node signatures, yielding a false negative (an undetected partition). We now evaluate the likelihood of such a scenario. Let $B(sum)$ be the number of nonzero bits in a summary of size f when inserting n node signatures into it. Because the node signature are uniformly random, the expected number of bits set to '1' in the summary is $\mathbb{E}\big(B(sum)\big) = f \cdot \left(1 - \left(1 - \frac{1}{f}\right)^n\right)$. More precisely, it was shown in [15] that the probability of the same summary to contain j non-zero bits is $\mathbb{P}\big(B(sum) = j\big) = \frac{\binom{f}{j} \cdot j! \cdot \left\{ {n \atop j} \right\}}{f^n}$, where $\left\{ {a \atop b} \right\}$ stands for the Stirling numbers of the second kind [16]. Stirling numbers are gruesome to handle and are usually tackled asymptotically, but since in this work we are ultimately interested in efficient implementations using small summaries, we resort instead to mathematical simulations.

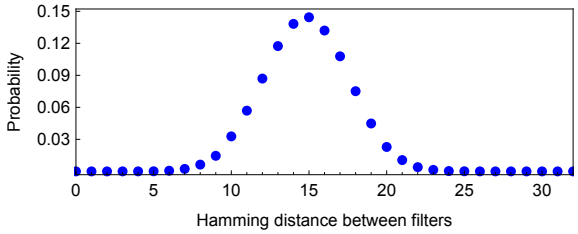

Figure 3. Distribution of the Hamming distance of two summaries of size $f = 32$ with $n = 64$ inserted signatures coming from independent subnetworks.

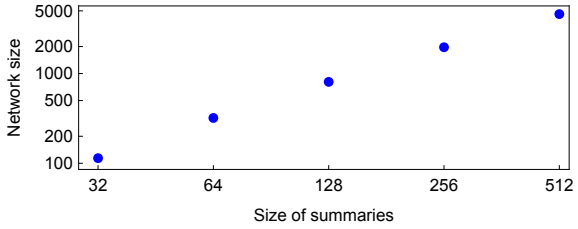

Figure 4. Maximum network size for a probability of false negative of 10^{-5} when partitioned into two partitions of equal size for different summary sizes.

As n and f increase, the distribution of $B(sum)$ is sharply concentrated around its expected value [17]. At first sight, this is disappointing: for instance with $f = 32$ and $n = 64$ the expected number of '1's in the summaries is 27.8. However, Figure 3 shows the distribution of the Hamming distance of the summaries for two disjoint subnetworks S_1 and S_2 of 64 nodes each (corresponding to a global network $S = S_1 \cup S_2$ of 128 nodes partitioned in two equal halves) with summaries sum_1 and sum_2 of size 32 bits. In this example, the probability that the Hamming distance between sum_1 and sum_2 is zero (and hence the rate of false positives) is approximately 10^{-5}.

Two partitions of equal size is the worst possible scenario since the probability that both summaries are identical decreases quickly as the size of each subnetwork diverges. Note that in the self-detection variant, there is no central authority that can compare the summaries from both partitions; if the size of the partitions greatly differs, the small subnetwork will easily detect the partition and trigger mitigating measures. This is further discussed in the next section.

Figure 4 shows the maximum network size allowing a probability of false negative of 10^{-5} when partitioned into two partitions of equal size for different summary sizes (again this is the worst possible scenario). Under perfect network and convergence assumptions, MtG can thus easily handle partition detection with high accuracy in networks of 128 nodes or less with 32-bit summaries, in networks of 800 nodes or less with 128-bit summaries, and in networks of 4500 nodes or less with 512-bit summaries. The sub-linear trend is due to summaries being wrongly identical when they are entirely filled with '1's. This corresponds to the coupon collector problem, and it takes $n.log(n)$ random node signatures to fill a n-bit filter.

C. Operating in reality: dynamism and imperfect aggregation

In a real deployment, of course, summaries may diverge within the same system connected S due to churn, node crashes, and imperfect aggregation caused by network contention, node mobility or a hostile environment. In other words, we might

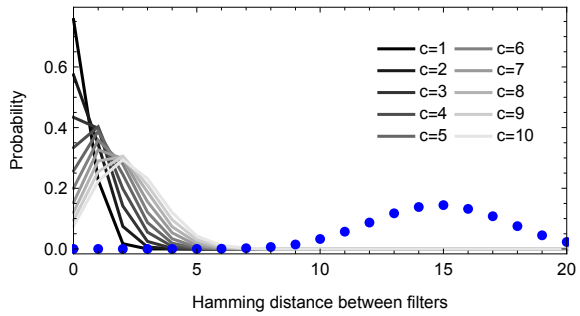

Figure 5. Distribution of the Hamming distance of two summaries of size $f = 32$ with $n = 64$ inserted signatures and churn parameter $c \in \{1, 2, 3, \ldots, 10\}$ (gray curves). The blue points correspond to two partitioned systems of 32 nodes. Probability mass functions calculated with 10^5 randomized experiments.

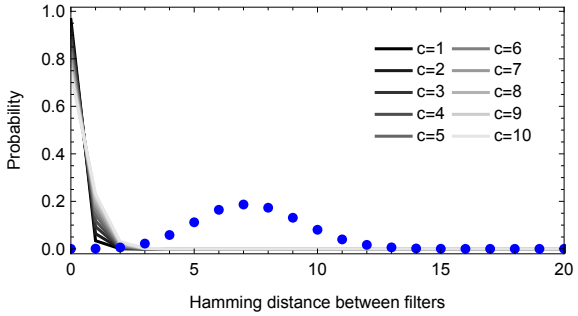

Figure 6. Distribution of the Hamming distance of two summaries of size $f = 32$ with $n = 128$ inserted signatures and churn parameter $c \in \{1, 2, 3, \ldots, 10\}$ (gray curves). The blue points correspond to two partitioned systems of 64 nodes. The probability mass functions are calculated with 10^5 randomized experiments.

have $\text{hdist}(sum_1, sum_2) \neq 0$, even though there is no partition. One should therefore set the threshold γ to a strictly positive value to account for these causes of divergence, thus reducing the chances that MtG/Self-detect and MtG/Assisted might erroneously detect partitions that do not exist. For our analysis, we model the effect of the above imperfections through a generic "churn" parameter c that seeks to capture the differences between sum_1 and sum_2 caused by isolated node failures and imperfect communication. More precisely, we will assume that sum_1 and sum_2 contain the nodes signatures of two sets S_1 and S_2 such that $|S_1 \setminus S_2| = |S_2 \setminus S_1| = c$. In the case of MtG/Assisted, this correspond to the situation in which c nodes of the global network S did not make it into sum_1, and c other and distinct nodes did not make it into sum_2. In the case of MtG/Self-detect, this correspond to a scenario in which c nodes present in epoch e fail in epoch $e+1$, and conversely c nodes that were failed in epoch e appear in epoch $e+1$. This simple model completely ignores topology because all nodes in a connected subset all have the same filter, so it doesn't matter where the churn happens.

Figure 5 shows the distribution of the Hamming distance of two summaries of size $f = 32$ used by a connected system of $|S| = 64$ nodes. Each gray curve corresponds to a churn level c varying from 1 to 10. For comparison, the blue points correspond to the situation in which S becomes partitioned into two equal halves of 32 nodes. The gray and blue probability mass functions are almost disjoint, thus if we set the partition threshold to $\gamma = 7$, we essentially detect 100% of the partitions and no false positives. In figure 6, we repeat the process with the same summary size but with systems of $n = 128$ nodes. There is a small overlap between the gray and blue curves, but by setting the partition threshold to $\gamma = 2$, we still detect 99% of the worst possible partitions and treat less than 1% of the normal events at $c = 10$ as partitions.

D. Discussion

It is important to note that the choice of c is arbitrary. What is "normal churn" and what is "a partition" is entirely dependent on the context and the specific circumstances of a real world network, and the value of c should be chosen by a network operator depending on their objectives. Our goal is to show that the operator of a very noisy network *can* set c to a high value, so MtG won't raise too many false alerts, and that our approach still works in adversarial circumstances. In other words, in a real world deployment, the value of c is context-specific, but in this analysis, we want c as high as possible.

The use of c to bound the amount of change under which our protocol should not detect any partition means that if a system S looses c nodes that become disconnected from the rest of S, but do not crash, S will no raise any alert when using MtG/Self-detect. However, in this case, the c nodes are still active and, from *their* point of view, a drastic partition just occurred. They will therefore produce an alert, and allow for mitigation actions.

Finally, we also emphasize that these results are conservative for two reasons. First, $c = 10$ and $n = 64$ corresponds to a churn rate of 15% per epoch. The operator of a network in such an environment might want to trigger mitigating measures even in the absence of a partition. Second, it assumes that the network is partitioned in two pieces of equal size. As mentioned earlier, this is the most pessimistic scenario (i.e., it will systematically yield the lowest Hamming distances between summaries for the different subsystems). It is thus clear that like for many applications of bit fields as a compact representation, such as Bloom filters [18], the efficiency of our approach is uniquely determined by the size of the filters/summaries and the number of nodes in the network: with very small filters and very large networks, the filters, even once partitioned equally, will be filled with '1's. As a result, and similarly to the situation where we consider ideal network and convergence assumptions, under realistic settings MtG can still reliably detect partitions in networks of 128 nodes or less with 32-bit filters, in networks of size 800 nodes or less with 128-bit filters.

IV. Evaluation

We evaluate our proposal along three dimensions. We first investigate whether filters can effectively distinguish between joined and partitioned networks (Section IV-B). We then assess the ability of MtG to detect partitions and compare them to two alternative approaches (Section IV-C). Finally, we explore the parameters space in Section IV-D.

A. Experimental setup and metrics

Unless stated otherwise, we configure MtG to use 32-bit filters, asynchronous rounds of 0.3 second, and epochs of 16 rounds (i.e., roughly 5 seconds per epoch). Naturally, our results apply to lower communication frequencies, provided that the epoch time be adjusted accordingly to comprise the same number of rounds. Typically, the communication frequency is set by the beaconing frequency of the routing protocol, and aggregation messages piggyback on these beacon messages. During each round, each node can push its current filter to all neighbors in its communication range, which we set to be 100m. Nodes are positioned over a 400m×400m area. The exact number of nodes, their position, and their mobility depend on the experiment, as we detail below.

We use the Omnet++/Inet framework [19] for our simulations. Presented results are averages over 10 runs using different random seeds for nodes signatures and positions. We do not show error bars as they prove to be negligible. We use the following metrics:

- **The normalized maximum pair-wise internal distance** (*internal distance* for short) measures the maximum Hamming distance between the filters being maintained by two nodes of the same monitored system S, normalized by the size of the filters f (32 in our experiments). When S is connected, this distance should converge to zero at the end of each epoch. Formally, we have: $\text{internal\_d}(S) = \max_{(p_i,p_j)\in S^2}\left(\frac{\text{hdist}(filter_i, filter_j)}{f}\right)$.
- **The normalized minimum pair-wise inter-partition distance** (*external distance* for short) measures the minimum Hamming distance between the filters of two partitions S_1 and S_2 of a system, normalized by the size of the filters. When S_1 and S_2 are disconnected, this distance should remain above the γ threshold, even in a converged state, in order to distinguish a partition from a connected configuration. Formally we have: $\text{external\_d}(S_1, S_2) = \min_{(p_i,p_j)\in S_1\times S_2}\left(\frac{\text{hdist}(filter_i, filter_j)}{f}\right)$.
- **The error rate** is the number of nodes belonging to a partitioned system that do not detect the partition (*false negatives*) summed with the number of nodes belonging to a fully connected system that raise a partition alert (*false positives*), over the total number of nodes.
- **The per-node per-round bandwidth** (*bandwidth* for short) represents the amount of information sent by each node per round, measured in bits.

B. Effective representation

We set up a system S of 120 nodes divided into two groups S_1 and S_2 of 60 nodes each. S_1 and S_2 are initially distributed over the same 400m×400m area, but drift apart in opposite direction at 25 m/s, S_1 heading North and S_2 South. S_1 and S_2 thus become unable to reach each other at around 15 seconds into the experiment (end of the 3rd epoch). We monitor over time the internal distance of S and the external distance between S_1 and S_2. Figure 7 shows that before the partition (0-15 seconds), the internal distance goes down to 0 very rapidly, in less than 2 seconds (6 rounds). Once the partition occurs (15-24 seconds), the external distance between the two partitions is always strictly positive, and converges quickly to a value noticeably above 0.

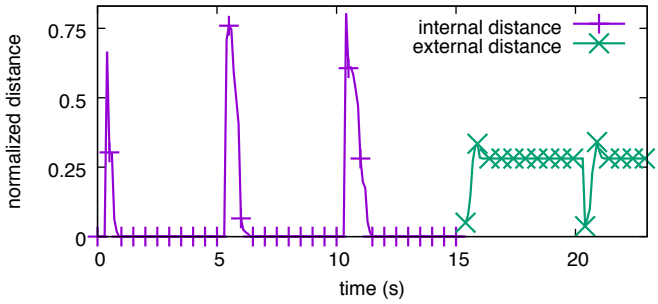

Figure 7. Filters are an effective representation of a network membership. They converge when the system is connected, and they are clearly distinct when there is a partition.

These two results combined demonstrate that filters are efficient to represent a system membership, in a quick, accurate, and compact manner. Even after only a few rounds, all nodes in a system agree on the same filter, while disconnected subsystems have very distinct filters, all the while using 32-bit filters for systems containing over a hundred nodes.

C. Partition detection

In this second set of experiments we compare the filters used by MtG with those of two baseline approaches:

- **The *graph-coloring* baseline:** Each node randomly choses a 16-bit integer, a *color*, and starts spreading it. When it receives a color announce from another node, it keeps the color with the biggest integer value and gossips this value thereafter. If a network is not partitioned, all nodes will eventually agree on the same color; if there is a partition, each subsystem will agree on a different color. Hence, the color can serve as a simple way to distinguish subsystems. Similarly to MtG, it uses a fixed-size information per node and for any gossip exchange.
- **The *full-list* baseline:** Each node maintains an exhaustive list of all node identifiers it has encountered, and gossips it around. Nodes merge incoming lists with their local list using an union operator. At the end of an epoch, all connected nodes agree on the same list, which is the membership list of their subsystem. If two different lists are observed, it is a sign of a partition. Unlike MtG and the graph-coloring baseline, the space cost of this method is linear in the number of nodes in the system.

We repeat the setup of the previous subsection: two 60-node groups moving away from each other. We run four experiments in which nodes execute either one of the variants of MtG or one of the two other protocols.

- In the first experiment, all 120 nodes execute MtG/Self-detect, the self-detection version of our approach.
- In the second one, we add a third group of nodes from an independent system in the middle, serving as the monitoring group, and we run MtG/Assisted, the

Table II
PARTITION DETECTION PERFORMANCE.

Approach	Error rate	Bandwidth (bits sent/round/node) average	max
graph-coloring	**50%**	16	16
full-list	0%	**1995**	**3840**
MtG (self-detect)	0%	32	32
MtG (monitoring node)	0%	32	64
(monitored node)		32	32

third-party version of our protocol. The third group is constituted of 20 fixed nodes, set up to ensure the coverage of the whole area of the experiment.

- In a third experiment, we used the graph-coloring baseline instead of our proposed filters to represent the network membership, in self-detect mode.
- Finally in the fourth experiment, we used the full-list baseline instead of filters, in self-detect mode.

The results are summarized in Table II. The bandwidth consumption assumes the following: filters are 32 bits long, node identifiers use 32 bits (size of an IPV4 address), and colors are 16-bit integers.

In both modes, our protocols accurately detect the partition when it happens, with modest bandwidth consumption, comparable to the graph-coloring approach. In contrast, the graph-coloring approach only detects the partition for half of the nodes. Indeed, the nodes in the group of the "winning" color do not see any change when the partition happens.

The extensive-list approach detects partitions accurately, but its bandwidth consumption is orders of magnitude bigger than with our approach, even with a system featuring as little as 120 nodes. Moreover, this bandwidth usage varies considerably over the execution of the protocol, with a low bandwidth usage at the beginning of an epoch (using small lists) and an unbounded usage as lists are aggregated to the full membership, e.g. up to 1,920 bits for a 60-node group, or 3,840 for a 120-node group. This bandwidth usage is orders of magnitude higher than with the filters, and increases linearly with the network size. Furthermore, it is in the expected common case when the lists are converged and that there is no partition that the usage is the highest, which is the opposite of the desired behavior.

D. Pushing the limits

In Section III, we mentioned that a network split perfectly in half was the worst case for MtG, because the summaries of each half are the closest to each other. This is clearly true in MtG/Assisted, however, in MtG/Self-detect a node compare its own summary to its summary from the previous epoch. When the network does not split exactly in half but instead in two subnetworks, one large and one small, the small group will converge to a significantly different summary. It will therefore detect the partition easily. The large subset on the other hand will observe a limited change between the summaries, and might miss a partition detection. To analyze this effect, we set up the same experiment as in subsection IV-B except that instead of 60 nodes in S_1 and S_2 each, we use varying proportions for the group sizes, from 50%-50% ($S_1 = S_2 = 60$ nodes) to 5%-95% ($S_1 = 6$ nodes, $S_2 = 114$ nodes). The results are given by Table III.

Table III
DETECTION IN CASE OF UNEVEN NETWORK SPLIT.

Share of nodes in S_1	Detection by S_1 (small group)	Detection by S_2 (large group)
50%	✔	✔
20%	✔	✔
15%	✔	✘
5%	✔	✘

As expected, the small group S_1 always detects the partition. Even with only 20% of nodes leaving (24 nodes in S_1), nodes in the larger group S_2 still detect the partition as well. It is only when S_1 contains 15% or less of the network (18 nodes or less) that S_2 begins to miss the partition. It is interesting to look more precisely at the distance between the signatures of each group *before* and *after* the partition. This is shown by Figure 8. The distances for both groups converge toward the same value when the original group is split exactly in half. This confirms that the even split is the worst scenario, even in MtG/Self-detect. In other cases, the detection and the triggering of mitigation measures is systematically easier for the smaller group resulting from the split.

We now explore specific adversarial conditions that MtG may encounter, and suggest countermeasures. First, the length of the epochs is a tradeoff between the quality of convergence between filters and the freshness of the information they contain. While a long epoch provides better convergence guarantees, shorter epochs avoid keeping information about crashed or unreachable nodes. The length of the epochs needs to be adapted to the size of the considered systems. To illustrate this, we set up an experiment with systems of increasing sizes. We keep the density of nodes constant between all experiments, simply distributing a larger number of nodes over a larger area, resulting in systems of increasing diameters, from 4 to 16 hops. The length of rounds and epochs is constant between experiments. Figure 9 shows the evolution of the internal distance of these systems over time. As the diameter increases, the convergence is slower. Past a certain diameter, filters do not have enough time to converge anymore, as the distance does not reach zero.

Given the number and types of devices deployed in a MANET and the physical area over which they are deployed, a network operator can get a rough estimation of its system's diameter. Based on our experiments, we recommend to set the length of an epoch (in rounds) to twice the expected maximal diameter of the system (in hops).

Finally, we test the resilience of MtG to bad communication conditions and message losses. We set up a system of 120 nodes distributed in two groups of 60 nodes each and make them drift apart under the scenario previously described. In order to stress the system, we reduce the length of the epoch to only 6 rounds, and we tune the drifting speed so that a

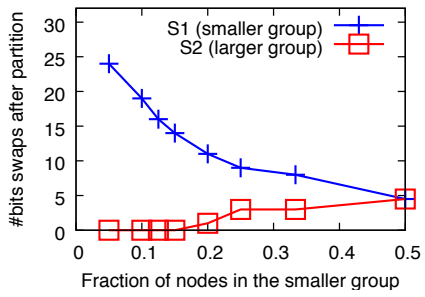

Figure 8. Numbers of bits swapping in the signature of each group after a partition (120 nodes overall, in the 2-group scenario of Sec. IV-B)

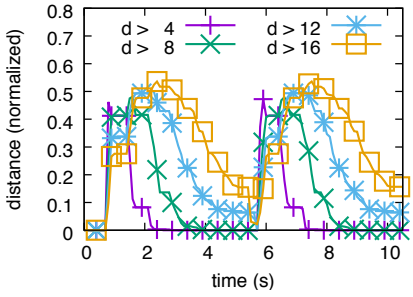

Figure 9. With fixed epoch and round durations, the larger the network is, the longer it takes to converge. If the network is too large, the filters do not converge anymore, so it is important to adapt the length of an epoch to the size of the network.

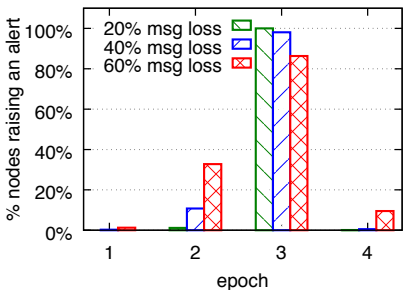

Figure 10. Fraction of nodes detecting a partition, under varying message loss levels. A real partition happens at the end of epoch 2. MtG resists message loss until a certain level where it can start to confuse the effect of message loss with partition.

partition happens at the end of the second epoch. MtG should hence raise a partition alert during epoch 3. Figure 10 shows the fraction of nodes that raise an alert during each epoch. With 20% of lost messages, the detection rate is not affected. With 40% message loss, barely 10% of the nodes confuse the effect of bad communication conditions with a real partition in epoch 2. It is however only when reaching a tremendous level of 60% message loss that a noticeable fraction of MtG nodes raise a false alert before the partition actually happens. We argue that under such harsh conditions, a system operator has more pressing issues than partitions, and will want to trigger available counter-measures just as if a real partition was happening. This makes us confident that MtG is resilient to message loss rates under all the circumstances where its detection capabilities can prove useful.

V. Related Work

Partitions are known to be problematic for MANETs and Wireless Sensor Networks, and have therefore been investigated in the past. To the best of our knowledge, however, none of these earlier approaches considered the use of compact filters as MtG does. In the following we review related work on partition detection, as well as on related problems, such as cut detection, partition prediction, and partition recovery.

a) Membership and partition detection: The work of Arantes *et al.* [9], [10] formalizes the notions of partition detector and partition participants detector in a manner similar to the classical formalization of failure detectors [20]. The two algorithms they propose accumulate information about broadcast propagation paths over epochs in order to construct local reachability information. When the set of nodes in the system is known beforehand [9], a partition is detected when some of these nodes become unreachable (possibly because of crashes). This approach is extended to systems with an arbitrary number of unknown participants [10], for which the detector is able to return the set of nodes present in the local partition, provided that the local partition eventually stabilizes. The accumulation of network participants in a list is similar to the way we accumulate members in our filters. For large networks, however, such an explicit approach is likely not to scale as we show in our evaluation.

Ritter *et al.* [11] propose an approach to detect partitions in MANETs in which a subset of active nodes exchange beacon messages that traverse the network. The proposed heuristic tends to position active nodes at the border of the network, in order to maximize the network nodes covered by a beacon propagation path. When beacons repeatedly fail to propagate between two active nodes, a partition is suspected. In contrast to MtG, this strategy assumes that border nodes can be reliably detected, and only change slowly.

Khelil *et al.* [21] present a broadcast strategy for partitionable MANETS based on hypergossiping, the selective rebroadcasting of partially broadcast messages. This strategy includes a mechanism to detect partition joins, i.e. the rejoining of the two parts of a previously disconnected network. This mechanism exploits Last Broadcast Received (LBR) lists, a list of the identifiers of the k last broadcast messages received by a node. Nodes periodically exchange this list, and conclude that they are rejoining a partitioned subnetwork when their local LBR substantially differ from that of their neighbors. Because the main goal of this approach is to maximize the delivery ratio of system-wide broadcasts, the partition join detection mechanism tends to err on the side of over-detection, with numerous wrong detection decisions in some instances [22].

b) Cut detection: Cut detection is a problem related, but distinct from partition detection in MANETs, and focuses on mostly static Wireless Sensor Networks, in which sensors forward their readings to dedicated sink nodes. A cut occurs when some nodes become disconnected from the sink.

The work of Barooah *et al.* [23] allows each sensor node to detect when it becomes disconnected from the sink, and whether other sensor nodes have become disconnected. Won *et al.* [24] target only the second problem and adds consideration about energy and robustness to malicious nodes. Because of the specific topology of sink-based sensor networks, these approaches are however not applicable here.

c) Partition prediction: Some works try to predict partitions before they happen, but require more powerful primitives than MtG. Some use GPS information (regarding both location and speed) to build a mobility model of the network and predict when nodes are likely to get out of range [7]. Other

proposals [8] assume the existence of a distributed algorithm that returns the set of disjoint paths between two nodes, and predict partitions based on the number of paths and their length. By comparison, our solution makes no assumption regarding the higher level capabilities of a network, and only assumes a one-hop broadcast primitive.

d) Address allocation: A few papers focus on allocating addresses to nodes, including after a network partition. The work of Singh *et al.* [25] only deals with graceful partitions (i.e. nodes detect and announce when they will leave/merge with the network), and propose a protocol to elect a new cluster head when a partition looses access to the initial head, but does not consider the kind of abrupt partitions we deal with. The solution of Zhou *et al.* [26] is designed to address this problem even in case of partitions, and so does not consider the problem of detecting them.

e) Partition recovery: Some works look at the problem of partition recovery, once a partition has been detected. For example, Han, Swindlehurst and Liu [5] consider the optimal placement of Unmanned Aerial Vehicles (UAV) to overcome the disconnection of ground-based MANETs. Their approach is based on heuristics that seek to maximize different metrics of connectivity, but assumes a central location where all relevant information about MANETs nodes and UAV positions can be aggregated and processed, an assumption we do not make.

VI. Conclusion & Future Work

We presented in this article MtG, a lightweight method to detect partitions in a MANET, either by the nodes forming the MANET themselves or by an external supporting system. Our analysis and evaluation show the ability of our approach to detect such partitions even under aggregation imperfection and imperfect networking conditions. For our future work, we are interested in developing a more formal method to derive the partition threshold γ based on the filter and network sizes, or at least to provide these thresholds for a larger number of parameters. In particular, as mentioned in Section III we do not leverage the number of bits set to '1' in our summaries. When a system is partitioned, two general cases can occur. If both partitions have the same size, the summaries from one epoch to the next will very likely contain significantly less bits set to '1'. If one partition is small and the other is large, the large partition might view its evolution as a normal churn event, but the small partition will see an even higher reduction in its number of summary bits set to '1' between consecutive epochs. Considering the number of bits set to '1' may allow us to tackle higher noise levels, as well as larger systems with good accuracy, without increasing the filter size. It would also avoid triggering an alert when a large number of nodes join the network or when two partitions reconnect, since this events *increase* the number of '1' in the filters, but are still detected with our current approach as large changes in membership.

Acknowledgment

The research leading to these results has received funding from the French Agence Nationale de la Recherche (ANR) under project PAMELA (ANR-16-CE23-0016), from CHIST-ERA under project DIONASYS, from the EU's H2020–The EU Framework Program for Research and Innovation 2014-2020 under Grants 653884 and 692178, and from the Swiss National Science Foundation (SNSF) under grant 155249.

References

[1] P. Ruiz and P. Bouvry, "Survey on broadcast algorithms for mobile ad hoc networks," *ACM Computing Surveys*, vol. 48, no. 1, Jul. 2015.
[2] K. Akkaya and M. Younis, "A survey on routing protocols for wireless sensor networks," *Ad Hoc Networks*, vol. 3, no. 3, pp. 325 – 349, 2005.
[3] P. Grace, D. Hughes, B. Porter, G. S. Blair, G. Coulson, and F. Taiani, "Experiences with open overlays: A middleware approach to network heterogeneity," in *European Conf. on Comp. Sys.*, ser. EuroSys, 2008.
[4] I. Bekmezci, O. K. Sahingoz, and S. Temel, "Flying ad-hoc networks (FANETs): A survey," *Ad Hoc Networks*, vol. 11, no. 3, 2013.
[5] Z. Han, A. L. Swindlehurst, and K. R. Liu, "Optimization of MANET connectivity via smart deployment/movement of unmanned air vehicles," *IEEE Trans. on Vehicular Tech.*, vol. 58, no. 7, pp. 3533–3546, 2009.
[6] G. Blair, Y.-D. Bromberg, G. Coulson, Y. Elkhatib, L. Réveillère, H. B. Ribeiro, E. Rivière, and F. Taïani, "Holons: Towards a systematic approach to composing systems of systems," in *Int. Workshop on Adaptive and Reflective Middleware*, ser. ARM, 2015.
[7] B. Milic, N. Milanovic, and M. Malek, "Prediction of partitioning in location-aware mobile ad hoc networks," in *38th Annual HICSS*, 2005.
[8] M. Hauspie, J. Carle, and D. Simplot, "Partition detection in mobile ad-hoc networks using multiple disjoint paths set," in *International Workshop on Objects models and Multimedia technologies*, 2003.
[9] D. Conan, P. Sens, L. Arantes, and M. Bouillaguet, "Failure, disconnection and partition detection in mobile environment," in *7th IEEE NCA*, 2008.
[10] L. Arantes, P. Sens, G. Thomas, D. Conan, and L. Lim, "Partition participant detector with dynamic paths in mobile networks," in *9th IEEE NCA*, 2010.
[11] H. Ritter, R. Winter, and J. Schiller, "A partition detection system for mobile ad-hoc networks," in *1st IEEE ComSoc Conference on Sensor and Ad Hoc Communications and Networks*, ser. SECON, 2004.
[12] M. Jelasity, A. Montresor, and Ö. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM TOCS*, 2005.
[13] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *44th Annual IEEE FOCS*, 2003.
[14] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communication of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.
[15] R. Barazzutti, P. Felber, H. Mercier, E. Onica, and E. Rivière, "Efficient and confidentiality-preserving content-based publish/subscribe with prefiltering," *IEEE TDSC*, 2017.
[16] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics*. Addison-Wesley, 1994.
[17] R. Motwani and P. Raghavan, *Randomized algorithms*. Chapman & Hall/CRC, 2010.
[18] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," *Internet Mathematics*, vol. 1, no. 4, pp. 485–509, 2002.
[19] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," in *SIMUTools*, 2008.
[20] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *Journal of the ACM*, 1996.
[21] A. Khelil, P. J. Marrón, C. Becker, and K. Rothermel, "Hypergossiping: A generalized broadcast strategy for mobile ad hoc networks," *Ad Hoc Networks*, vol. 5, no. 5, pp. 531–546, 2007.
[22] A. Khelil, P. J. Marrón, R. Dietrich, and K. Rothermel, "Evaluation of partition-aware manet protocols and applications with ns-2," in *SPECTS*, 2005.
[23] P. Barooah, H. Chenji, R. Stoleru, and T. Kalmár-Nagy, "Cut detection in wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 3, pp. 483–490, 2012.
[24] M. Won, S. M. George, and R. Stoleru, "Towards robustness and energy efficiency of cut detection in wireless sensor networks," *Ad Hoc Networks*, vol. 9, no. 3, 2011.
[25] S. Singh, N. Rajpal, and A. Sharma, "Address allocation for MANET merge and partition using cluster based routing," *SpringerPlus*, 2014.
[26] H. Zhou, L. M. Ni, and M. W. Mutka, "Prophet address allocation for large scale manets," *Ad Hoc Networks*, vol. 1, no. 4, pp. 423–434, 2003.