Stability Detection in Mobile Ad-Hoc Networks

Roy Friedman†
Department of Computer Science
Technion - Israel Institute of Technology
Haifa 32000
Israel
roy@cs.technion.ac.il

Galya Tcharny‡
Intel Israel LTD
MATAM
Haifa 31015
Israel
galya.tcharny@intel.com

November 25, 2003

Abstract

Mobile Ad-Hoc Networks (MANET) are formed in an ad-hoc manner by collections of devices that are equipped with wireless communication capabilities such as the popular WiFi (IEEE 802.11b) standard. Given that the hardware technology and networking protocols for MANETs become mature and ubiquitous, the main barrier for MANETs to become widely used is applications. Like in other areas of distributed computing, in order to expedite the development of applications, there is a need for middleware services that support these applications.

Reliable multicast has been identified as a basic service for many reliable distributed applications. Detecting when messages become stable, i.e., have been received by all their intended recipients, is an important aspect of reliable multicast, as it enables garbage collecting such messages from buffers and advancing flow control send windows. Moreover, as been motivated by previous work, being able to detect both when messages become stable and when they become fuzzy-stable can lead to improved system performance in MANETs. This paper presents a novel stability detection protocol. The protocol is specifically adapted to MANETs, as it operates in a gossip based manner, and as it detects both stable and fuzzy-stable messages. The paper also includes an evaluation of this protocol by extensive simulations under various network parameters.

Keywords: Stability Detection, Reliable Distributed Systems, Reliable Multicast, Mobile Ad-Hoc Networks, Gossip Protocols

†This research was supported by grants from Intel, IBM faculty partnership award, the European Commission IST Project No. IST-2001-37610, and the Bar-Nir Bergreen Software Technology Center of Excellence
‡The author is currently also with IRISA/VINRIA Rennes, France
§The work was done while this co-author was a graduate student in the Department of Computer Science at the Technion
1 Introduction

Mobile Ad-Hoc Networks (MANET), as their name suggests, are networks of mobile devices that are formed in an ad-hoc manner. The devices that participate in such networks have wireless communication capabilities with a limited range transmitters, and therefore can directly communicate with other devices in their range. Some of the devices occasionally volunteer to forward some of the messages they receive, or in other words, act as routers, thereby forming a network. Yet, there is no fixed infrastructure, the network is continuously changing, and routers are elected on demand. In other words, the networking issues are handled ad-hoc.

Developing services for MANET is becoming important for several reasons. First, the enabling hardware technology, Wireless Ethernet (WiFi) Network Interface Cards (NICs), is becoming a standard feature in laptops, notebooks, and PDAs. In particular, WiFi NICs are being integrated into laptop chip-sets and even into mobile processors’ CPUs. Second, MANETs offer a potential for new applications and improved services for mobile users, especially since mobile devices are increasingly equipped with both cellular communication and WiFi, and their computing power is becoming stronger. Example applications include collaborative caching of Internet based services, interactive games, ad-hoc transactions and e-commerce, collaborative (shared white-board and video conferencing like) applications, and enhancing the bandwidth and reach of cellular communication. Third, ad-hoc networks are self-managed and easily deployable, as they require no pre-existing infrastructure.

Applications operating over MANETs are fully distributed by nature, and cannot rely on a centralized server. Moreover, many of the applications we envision, as described above, can benefit from a reliable multicast service, that can disseminate information reliably among all participants. Realizing reliable multicast often involves buffering received messages by some or all processes for possible retransmission of these messages to nodes that failed to receive them during the first attempt. Such messages are typically stored in the receivers’ buffers until they become stable, i.e., it is known that they have been received by everyone. Additionally, many reliable multicast protocols, e.g., [3, 6, 14, 11, 17, 16, 21, 22, 23, 25], as well as group communication (GC) toolkits, e.g., [1, 2, 4, 5, 13, 18, 19, 20, 26, 27], employ flow control mechanisms. These mechanisms limit the rate of multicasts allowed by each sender to ensure that no more than some threshold of such messages is unstable at any point in time. This is done in order to avoid overflowing the network and the receivers’ buffers, something that would reduce both the performance of the system and its reliability.

Thus, stability detection, i.e., the act of detecting when messages have become stable, is a key building block in many toolkits and protocols aiming to ensure reliable multicast. Yet, when considering existing stability detection techniques, it is easy to see that they do not match well the characteristics of MANETs. For example, some approaches involve exchanging stability detection protocol messages among all processes [8, 12]. These approaches are not very scalable, due to the large communication overhead they impose. Moreover, in a MANET, even with a moderate number of participants, the cost of such global exchanges is prohibitively high, due to the fact that in MANETs routing is done by the devices themselves, and many devices are battery powered.

More scalable approaches to stability detection have been proposed, e.g., in [8, 12]. However, these approaches were developed for wired fixed networks. When attempting to utilize them in MANETs,
one encounters two immediate drawbacks. First, these approaches assume point-to-point communication. Given that wireless communication always generates a limited range broadcast, it is a pity not to utilize it. Second, existing scalable approaches impose some logical topology that is overlayed on the nodes. Yet, due to mobility, the topology of a MANET is constantly changing, so the assumed logical topology will either require a lot of maintenance effort to evolve with the physical one, or risk being completely out of touch with the physical one. Third, most scalable approaches are based on rounds such that stability is detected at the end of each round. These rounds end whenever a node has heard transitively from all other alive nodes in the system. However, in a MANET where transient disconnection may be common, each such round may take a long time, leading to very long stability detection delays.

Another problem with relying on existing stability detection techniques for maintaining flow control is that one slow node that fails to acknowledge messages in a timely manner can slow down all others. This problem often appears in large local area networks (due to the increased probability of load imbalance), in applications running over wide area networks (due to the unpredictability of the Internet), and in MANETs (due to transient disconnections and frequent changes in routing paths). In [7], we have suggested that using fuzzy group membership can help overcome this problem. Briefly, the idea is to associate a fuzziness level factor to each multicast group member, rather than using a binary membership value; for example, the fuzziness level can be an integer greater or equal to 0 where the higher this value is, the worse is the connectivity of the corresponding node. This way, it is possible to declare all messages that were acknowledged by all members whose fuzziness level is below some threshold as fuzzy-stable. In return, this allows the buffer management and flow control mechanisms of reliable multicast to employ various optimizations on such messages [10]. For example, fuzzy-stable messages can be saved in compressed form, or only by a few nodes, and the flow control mechanism may allow advancing the send windows beyond fuzzy-stable messages (that are not fully stable yet).

In this paper we present a novel stability detection protocol that has the following nice characteristics, making it suitable for MANETs. The protocol operates in a gossip manner, based on the temporal physical structure of the network. The protocol can detect both fuzzy-stable and stable messages, and while it is round based, it allows rounds to end early, in order to avoid arbitrarily long stability detection times caused by transiently disconnected nodes. In fact, our protocol can be viewed as combining ideas from the FullDisr algorithm described in [8, 12] with gossip based approaches, but in a way that matches MANET environments. We analyze this protocol and study its performance with various network parameters. The results of these simulations validate the benefits of our approach.

# 2 Basic Concepts

We assume a distributed environment, composed of a known number $n$ of nodes, sometimes also called processes, processors, or devices. These nodes are placed in some physical domain, and each of them may move arbitrarily in that domain. Nodes can send messages with some known transmission range. When the distance between two nodes $p$ and $q$ is less than the transmission range of $p$, we say that $q$ is within the transmission range of $p$. A message $m$ sent by a node $p$ can be received by all other nodes that are within the transmission range of $p$ when $m$ is sent.
Yet, the system is not completely reliable, and some failures may occur. These include node crashes, in which case a node ceases to function completely, or message omissions. Omissions can happen either if a process fails to send a message it is expected to send, a message is lost in transit, or a process fails to receive a message. Yet, we assume that omissions are relatively rare, and occur in a random manner. In the literature, there exists another type of failures, known as Byzantine failures [24], in which processes behave in an arbitrary manner, including sending messages they are not supposed to, but these are beyond the scope of this work.

In addition, we assume a fuzzy failure detection mechanism, as proposed in [7]. That is, this mechanism reports to each node the fuzziness level of each other node, which is an integer larger or equal to 0. Nodes whose fuzziness level is above a given threshold are considered fuzzy. Messages that are received by all their intended recipients are considered stable. Similarly, if only fuzzy nodes are not known to receive a message, then the message is said to be fuzzy-stable.

Additionally, we assume the existence of a protocol that attempts to ensure reliable multicast for application messages, or in other words, that each application message that is sent is eventually delivered to every process that does not crash. Moreover, we assume that this mechanism deliver messages sent by the same sender in FIFO order. The goal of our work is to provide a stability detection mechanism of application messages that can be used by the reliable multicast protocol and other possible middleware services. That is, our mechanism detects both stable messages and fuzzy-stable messages. Finally, we would like to emphasize that messages generated by the stability detection mechanism itself need not be reliably delivered, and may be received in arbitrary order, although for liveness, we assume that most of them are received within a bounded delay.

3 MANET Based Stability Detection

3.1 The FuzzyFullDist Protocol

FuzzyFullDist uses a gossip mechanism (similar to [8, 12]) to disseminate stability information. However, as elaborated below, it adapts existing gossip ideas from wired networks to MANET settings. Thus, we start by describing how gossip stability detection protocols usually work, and then focus on the changes that were made to obtain FuzzyFullDist.

Existing gossip stability detection protocols proceed in rounds, executed sequentially. At the beginning of each round, each node \(i\) initializes an array \(M_i\) to hold the sequence numbers of the last consecutive message it received from each other node. The array \(M_i\) is periodically sent to the neighbors of \(i\). When \(i\) receives an array \(M_j\) from one of its neighbors, \(i\) sets \(M_i\) to be the element wise minimum of both arrays. This way, \(M_i\) eventually holds the last messages that were received by all nodes at the beginning of the round, at which point \(i\) can move to the next round. In order to detect this eventuality, \(i\) also maintains a boolean array \(G_i\) in which \(G_i[j]\) is \(true\) when the value of \(M_i\) reflects \(j\)'s information at the beginning of the current round. Thus, \(G_i[j]\) is initialized to \(true\), and all other entries in \(G_i\) are initialized to \(false\). \(G_i\) is piggybacked on every message, and when \(i\) receives \(G_j\), it sets \(G_i\) to be the element wise logical or of both vectors. When \(G_i\) is all \(true\), \(i\) knows that it has
(transitively) heard from all nodes, and therefore $M_i$ holds the sequence numbers of the last known stable messages. At this point these messages are declared stable by storing their sequence number in an array $S_i$ and a new round can begin. Note that a non-round based gossip protocol can also be devised, but that would require sending matrices on each protocol message, which is often considered prohibitively expensive.

The problem is that in an ad-hoc environment, where disconnections can be frequent, a protocol round may last a very long time. During a round, the stability detection protocol disseminates information about messages that were received prior to its beginning, hence long round durations delay disseminating information about newly received messages. In order to allow efficient detection of fuzzy-stable messages, we introduce the ability to have early virtual round termination, even before a node (transitively) heard from every other node, if the round has lasted for too long. The purpose of introducing the virtual rounds is to allow network nodes to advertise newly received messages sooner. That is, nodes do not need to wait to hear from all other nodes to start telling about newly received messages, but it is enough to hear only from the currently connected (non-fuzzy) nodes. Hence, virtual rounds facilitate fuzzy-stability detection, which is performed at the end of each virtual round. So in fact, $FuzzyFullDist$ runs two protocols rounds concurrently: one round as assumed by known stability detectors (that is, a round lasts until $G_i$ is all true), and another virtual round, which always terminates after at most $T_{\text{detection}}$ time units – the virtual round uses its own set of variables, denoted $vM_i$ and $vG_i$. $T_{\text{detection}}$ is chosen so that all nodes that belong to a well-connected component of the network will know about messages that were received by all other nodes in the same component at the beginning of a virtual round with high probability. Note also that a virtual round may terminate when $vG_i$ is all true (if this occurs faster than $T_{\text{detection}}$).

In order for the information gathered in virtual rounds to be useful, it should be saved carefully. If a virtual round terminates as a result of all true $vG_i$, then $vM_i$ holds authentic stability information. On the other hand, if the virtual round terminates early, the gathered information will be lost, since at the beginning of a new round, $vM_i$ is initialized with local information. In particular, if all virtual rounds end early, then nothing meaningful will be obtained from them. This problem is solved in $FuzzyFullDist$ as follows: each node $i$ remembers in a matrix $R_i$ the maximal sequence numbers reported by any node from which it received a stability detection protocol message. $FuzzyFullDist$ relies on the fact that when a protocol message is received, messages with sequence numbers smaller than those appearing in the received array $M_j$ were received by all the nodes for which $G_j[k]$ is true. Thus, when a protocol message is received, a node (in addition to the actions described above) updates the $R_i[k]$ row for every node for which $G_j[k]$ is true as the element-wise maximum of $M_j[k]$ and $R_i[k]$. Hence, each entry of the stability array $S_i$ is computed at any point in time as a minimum of the corresponding column in $R_i$ (if it is greater than the current entry of $S_i$).

In order to detect fuzzy-stable messages, $FuzzyFullDist$ relies on the failure detection service that was described in Section 2 to identify fuzzy nodes. A message is declared fuzzy-stable when the fuzziness level of all processes that are not known to have received this message is above some predefined threshold $\text{fuzzthresh}$. Formally, each node $i$ maintains the following variables:
• An $n \times n$ matrix $R_i$ of sequence numbers. $R_i[i,j]$ holds the array of sequence numbers of messages that $i$ knows that $j$ received. In particular, $R_i[i]$ is a sequence numbers array of messages received by $i$ itself.

• An integer array $M_i$ containing sequence numbers of messages acknowledged by all the participants of the current round as known to $i$ so far.

• A boolean array $G_i$ recording the nodes whose stability information was learned by $i$ in the current round.

• Arrays $\nu M_i, \nu G_i$ - same as $M_i$ and $G_i$, respectively. These arrays are used in virtual rounds.

• A stability array $S_i$ corresponding to $i$’s stability information at the moment. $S_i$ is the output array of the protocol, which is used by other protocols that need stability information.

• An array $F_i$ containing the maximum sequence numbers of messages heard so far by all nodes except those with fuzziness level above $\text{fitthresh}$. If $m$ is a sequence number of the message sent by node $k$ and $m$ is in the interval between $S_i[k]$ and $F_i[k]$ when the virtual round ends, then $m$ is $\text{fuzzy-stable}$ and hence buffer management optimizations may be applied on it. Obviously, at the end of the virtual round $F_i[k]$ is always greater or equal to $S_i[k]$. $F_i$ is an output array (similar to $S_i$).

• An integer $r_i$, which holds the round index.

• An integer $\nu r_i$, which holds the virtual round index.

• A timer $\text{round\_timer}_i$, which indicates when it is time to move to the next virtual round. $\text{round\_timer}_i$ timeouts every $T_{\text{detect}}$ time units.

• An integer $\text{max\_round}$, remembering the maximal virtual round index $i$ has heard from.

As indicated above, $\text{FuzzyFullDist}$ includes two sub-protocols: $\text{GossipFullDist}$, which runs only full rounds, and $\text{VGossipFullDist}$, which may also early-terminate a virtual round. The pseudocode for $\text{GossipFullDist}$ is given in Figures 1 and 2.

In $\text{GossipFullDist}$, if a node $i$ receives a protocol message from round $r_i + 1$, it adopts the attached $M_j$ and increments $r_i$ (Figure 2, lines 15–22). If a node $i$ receives a protocol message from round $r_i - 1$, it ignores the message (Figure 2, lines 24–26). Intuitively, if node $i$ receives a protocol message from round $r_i + 1$, it means that its sender already knows all information that nodes advertised in round $r_i$, and the resulting stability information appears in the stability vector attached to the protocol message. Therefore, it is pointless for node $i$ to continue round $r_i$, as any information it could have learned already appears in the received round $r_i + 1$ protocol message. For similar reasons, protocol messages marked with earlier rounds can be ignored.

On the other hand, we claim that if the same policy for handling messages with higher or lower protocol round indices would be employed in $\text{VGossipFullDist}$, then this could result in needless stability detection delays. The problem stems from the fact that the relevance of information cannot be devised solely from the round index, as elaborated in Example 3.1.
Notation:

- ArrayMin - element-wise minimum of the input arrays
- ArrayMax - element-wise maximum of the input arrays
- \( A > B \) - evaluates to true for arrays \( A \) and \( B \) if \( \exists k \), \( A[k] > B[k] \) and \( \forall j, A[j] \geq B[j] \)
- \( \min \{ R \} \) - the minimum value in a set of numbers \( R \). If \( R \) is an empty set, the result is \(-1\)
- \( \max(A, B) \) - maximum of two numbers \( A \) and \( B \)
- \( A_1 \lor A_2 \) - element-wise OR of arrays \( A_1 \) and \( A_2 \)

Initialization of node \( i \):

\[
G_i[i] := \text{true}, \text{for all } j \neq i \quad G_i[j] := \text{false}
\]

\[
M_i := R_i[i]
\]

\[
S_i := [0, 0, \ldots , 0]
\]

\[
r_i := 0
\]

procedure \text{Start New Round}

\[
\text{broadcast(stability, } r_i, G_i, M_i, S_i)\]

\[
\text{for all } j \text{ do}
\]

\[
G_i[j] := \text{false}
\]

\[
\text{done}
\]

\[
G_i[i] := \text{true}
\]

\[
M_i := R_i[i]
\]

\[
r_i := r_i + 1
\]

\[
\text{broadcast(stability, } r_i, G_i, M_i, S_i)\]

\[
\text{end}
\]

Figure 1: *GossipFullDist* sub-protocol: Notation and Initialization

Example 3.1 In this example we present a scenario in which if a node joins a higher round upon receiving its protocol messages and ignores messages from lower rounds, it may cause a needless stability detection delay. Recall that this optimization is employed by *GossipFullDist*, but not by *VGossipFullDist*, so here we explain why.

Assume the network topology of Figure 3, where nodes are vertices and there is an edge between any pair of nodes that may receive each other messages. Assume that each node sends *GossipFullDist* and *VGossipFullDist* messages each \( \Delta \) time units in one physical packet. Assume also that all nodes have just begun the same *GossipFullDist* round \( r \) and the same virtual round \(vr\), and now prepare to send their first messages in a new round. Node 1 broadcasts its first message while node 2 processes it and then broadcasts its own message. At this point, nodes 1 and 2 disconnect from the rest of the network, while they remain connected with each other (that is, the edge between nodes 2 and 3 fails). Note that node 3 received *GossipFullDist* and *VGossipFullDist* messages from node 2 that include information on nodes 1 and 2, which allows nodes 3, 4...N to move to the next full round \( r + 1 \) and virtual round \( vr + 1 \) fast, as they can transitively hear from all nodes. Nodes 1 and 2 cannot complete the full round, and have to wait for their virtual round to timeout. During this virtual round, node 2 broadcast a bundle of messages \( B \). When nodes 1 and 2 move to the next virtual round \( vr + 1 \), node 2 learns that messages \( B \) were received by node 1. Soon after that, nodes 3, 4...N also move to their
1: Every $\Delta$ time units, \texttt{broadcast}(stability, $r_i$, $G_i$, $M_i$, $S_i$) \\
2: \\
3: upon receive(stability, $r_i$, $G_i$, $M_i$, $S_i$) from node $j$ do \\
4: \hspace{1em} $S_i := \text{ArrayMax}(S_i, S)$ \\
5: \\
6: \hspace{1em} \textbf{for all} $j$ do \hspace{1em} /* keep $R_i$ consistent with last stability information */ \\
7: \hspace{2em} $R_i[j] := \text{ArrayMax}(R_i[j], S)$ \\
8: \\
9: \hspace{1em} \textbf{if}($r > r_i$) then \hspace{1em} /* receive a message from current round */ \\
10: \hspace{2em} $M_i := \text{ArrayMin}(M_i, M)$ \\
11: \hspace{2em} \hspace{1em} \textbf{for all} $j$ such that $G_i[j] = \text{true}$ do \hspace{1em} /* remember $M$ for all participants of the round */ \\
12: \hspace{3em} $R_i[j] := \text{ArrayMax}(R_i[j], M)$ \\
13: \hspace{2em} $G_i := G_i \lor G$ \\
14: \\
15: \hspace{1em} \textbf{else if}($r < r_i$) then \hspace{1em} /* receive a message from the greater round */ \\
16: \hspace{2em} $M_i := \text{ArrayMin}(M_i, R_i[i])$ \hspace{1em} /* move to the next round */ \\
17: \hspace{2em} $G_i := G$ \\
18: \hspace{2em} $G_i[i] := \text{true}$ \\
19: \hspace{2em} $r_i := r$ \\
20: \hspace{2em} \hspace{1em} \textbf{for all} $j$ such that $G_i[j] = \text{true}$ do \hspace{1em} /* remember $M$ for all participants */ \\
21: \hspace{3em} $R_i[j] := \text{ArrayMax}(R_i[j], M)$ \\
22: \hspace{1em} \textbf{endif} \\
23: \\
24: \hspace{1em} \textbf{else if}($r < r_i$) then \\
25: \hspace{2em} \textbf{return} \\
26: \hspace{1em} \textbf{endif} \\
27: \\
28: \hspace{1em} $\forall j, S_i[j] := \max(S_i[j], \min\{\forall k, R_i[k][j]\})$ \\
29: \\
30: \hspace{1em} $\forall k$, Label all messages received from node $k$ with sequence number $P$, \\
31: \hspace{1em} $P \leq S_i[k]$ as stable \\
32: \\
33: \hspace{1em} \textbf{if}(G_i is all true) then \hspace{1em} /* start new round */ \\
34: \hspace{2em} \textbf{Start-New-Round} \\
35: \hspace{1em} \textbf{endif} \\
36: \hspace{1em} \textbf{do} \hspace{1em} /* receive(stability, $r_i$, $G_i$, $M_i$, $S_i$) */ \\
37: \\
38: \\
39: \\
40: \textbf{Figure 2: GossipFullDist sub-protocol - code for node $i$}
next virtual round \( v_r + 2 \), and prepare to send their new round messages. At this point node 1 remains disconnected and node 2 joins the rest of the network as appears in Figure 4. Node 2 retransmits messages \( B \) to the rest of the nodes. Now node 3 sends its protocol message, which causes node 2 to join the full round \( r + 1 \) and the virtual round \( v_r + 2 \), while it remains the only node that knows that node 1 received messages \( B \). After that, nodes 2, 4, 5, \ldots, \( N \) simultaneously send their protocol messages and node 2 learns that messages \( B \) are stable. At this point, node 2 joins node 1 and they both remain disconnected. Hence, only nodes 1 and 2 are able to detect stability of messages \( B \). Nodes 3, \ldots, \( N \) will be able to detect their stability only after the partition is healed. That is, messages \( B \) are needlessly buffered by a majority of network nodes.

Note that GossipFullDist cannot speed up stability detection in this example. This is because in GossipFullDist, node 1 does not start its full round after it delivered messages \( B \).

VGossipFullDist strives to keep the already gathered information from being lost, while also gossipping newly arrived information as soon as possible. As we explained in Example 3.1, in VGossip-
Initialization of node $i$:

- $vG_i[0] := \text{true}$, for all $j \neq i$, $G_i[j] := \text{false}$
- $vM_i := R_i[k]$
- $S_i := [0, 0, \ldots, 0]$  /* shared with GossipFullDist sub-protocol */
- $F_i := [0, 0, \ldots, 0]$
- $vr_i := 0$
- $\text{max round} := vr_i$
- $\text{round timer}_i := T_{\text{det}}$

**procedure vStartNewRound**

- $\text{broadcast}(\text{vstability}, vr_i, vG_i, vM_i, S_i)$
- for all $j$ do
  - $\text{fuzzy stable} := j$'s latest message that was acknowledged by all nodes except processes with fuzziness level at least $f\text{thresh}$
  - $F_i[j] := \max(F_i[j], \text{fuzzy stable})$
- done
- for all $j$ do
  - $vG_i[j] := \text{false}$
- done
- $vG_i[0] := \text{true}$
- $vM_i := R_i[k]$
- $\forall k$, Label all messages received from node $k$ with sequence number $P$
- $S_i[k] < P \leq F_i[k]$ as $\text{fuzzy stable}$
- $\text{max round} := vr_i := \text{max round} + 1$
- $\text{broadcast}(\text{vstability}, vr_i, vG_i, vM_i, S_i)$
- $\text{round timer}_i := T_{\text{det}}$
- end

Figure 5: VGossipFullDist sub-protocol: Initialization

FullDist nodes do not move to the next round as a result of a protocol message from a higher round. Yet, if a protocol message is received from higher or lower round and it includes new information on some nodes, then these nodes are included in the round as if information about them has arrived from the same round as the local one. In Example 3.2, we discuss how our approach allows to relieve the problem of Example 3.1. At the same time, nodes remember the highest round index they heard from in a variable $\text{max round}$, and the next round they start is indexed by $\text{max round} + 1$. This is done in order to reduce the number of concurrently existing round indices and the gap between them.

The pseudocode for VGossipFullDist is given in Figures 5 and 6. Also, VGossipFullDist may mark messages as stable or fuzzy-stable. The protocol shown here assumes the fuzzy-stability detection with fuzziness level threshold $f\text{thresh}$. FuzzyFullDist runs GossipFullDist and VGossipFullDist in parallel. Also, the two sub-protocols produce gossips with the same interval. Hence, its only natural to piggyback their messages on each other; the partition of the sub-protocols is done for the clarity of presentation.
1: Every $\Delta$ time units, broadcast($\mathit{vstability}$, $\mathit{vr}_i$, $\mathit{vG}_i$, $\mathit{vM}_i$, $S_i$)
2: 
3: upon $\mathit{round\_timer}$, $\mathit{timeout}$ do
4: \hspace{1em} $\mathit{gossip\_timer}$, $\mathit{cancel}()$
5: \hspace{1em} $\mathit{vStart\_New\_Round}$
6: done;
7: 
8: upon receive($\mathit{vstability}$, $r$, $G$, $M$, $S$) from node $j$ do
9: \hspace{1em} $S_i := \text{ArrayMax}(S_i, S)$
10: for all $j$ do \hfill /* keep $R_i$ consistent with last stability information */
11: \hspace{1em} $R_i[j] := \text{ArrayMax}(R_i[j], S)$
12: 
13: if($r = \mathit{vr}_j$) then \hfill /* receive a message from current round */
14: \hspace{1em} $\mathit{vM}_i := \text{ArrayMin}(\mathit{vM}_i, M)$
15: for all $j$ such that $\mathit{G}[j] = \text{true}$ do \hfill /* remember $M$ for all participants of the round */
16: \hspace{1em} $R_i[j] := \text{ArrayMax}(R_i[j], M)$
17: \hspace{1em} $\mathit{vG}_i := \mathit{vG}_i \vee G$
18: 
19: else if($r \neq \mathit{vr}_j$) then \hfill /* receive a message from another round */
20: \hspace{1em} $\mathit{max\_round} := \text{max}(r, \mathit{max\_round})$
21: \hspace{1em} for all $j$ such that $\mathit{G}[j] = \text{true}$ do \hfill /* remember $M$ for all participants of the round for which $M$ contains new information */
22: \hspace{2em} if($M > R_i[j]$) then
23: \hspace{3em} $R_i[j] := \text{ArrayMax}(R_i[j], M)$
24: \hspace{3em} $\mathit{vM}_i := \text{ArrayMin}(\mathit{vM}_i, M)$
25: \hspace{2em} $\mathit{vG}_i[j] := \text{true}$
26: endif
27: 
28: endif
29: 
30: $\forall j$, $S_i[j] := \text{max}(S_i[j], \text{min}\{\forall k, R_i[k][j]\})$ (30)
31: 
32: $\forall k$, Label all messages received from node $k$ with sequence number $P$, $P \leq S_i[k]$ as stable
33: 
34: if($\mathit{vG}_i$ is all true) then \hfill /* start new round */
35: \hspace{1em} $\mathit{round\_timer}$, $\mathit{cancel}()$
36: \hspace{1em} $\mathit{vStart\_New\_Round}$
37: endif
38: 
39: done; \hfill /* receive($\mathit{vstability}$, $r$, $G$, $M$, $S$) */

Figure 6: $\mathit{VGossipFullDist}$ sub-protocol - code for node $i$
Example 3.2. In this example, we show that by using our approach, the stability detection delay can be reduced in the scenario of Example 3.1. At the time when node 2 reconnects to the majority of the nodes and node 1 remains disconnected (Figure 4), node 2 runs virtual round \( vr + 1 \), and nodes 3...N run virtual round \( vr + 2 \). As a result from a message of node 3, node 2 moves to the full round \( r + 1 \), while it remains in virtual round \( vr + 1 \). When nodes 2, 4, 5,...N simultaneously send their protocol messages, nodes 3...N learn that node 1 has messages B, while node 2 starts a new virtual round \( vr + 2 \), since it collected information about all nodes. When node 3 issues its next protocol message, nodes 4...N will detect the stability of messages B (while nodes 2 and 3 learned it earlier). When node 2 joins node 1, node 1 will also detect the stability of messages B. Thus, when our approach is used, all nodes know that messages B are stable at the end of the scenario.

3.2 Liveness and Correctness

Liveness of a stability detection protocol implies that stability of a stable message \( m \) is eventually detected. For the liveness of our protocol, we must rely on the failure detection mechanism. Clearly, if any node fails, stability of new messages might never be detected since the failing node will never acknowledge them. Yet, when using a failure detector, as soon as node \( i \) suspects node \( j \), it sets \( G_{i}[j] \) to true (this was dropped from the formal presentation for the sake of simplicity). In particular, when an application runs above a group communication toolkit, once a node is considered faulty, it is eliminated from the view with a view change operation. This mechanism also allows to handle network partitions. When a network partition occurs, as a result of a view change, each component of the partition forms its own view, and message stability is defined only among members of the same view. Note that when a view change occurs, nodes agree on the system's state, after which previously received messages can be discarded. Hence, in the rest of this section, we assume that there are no failures and all network partitions are short-lived.

Lemma 3.1. In all infinite runs of the protocol that do not experience any failure suspicions, all rounds of the GossipFullDist sub-protocol are finite.

Proof: Assume, by way of contradiction, that round \( r \) of GossipFullDist is infinite. That is, there exist node \( i \) and node \( j \) such that \( i \) never receives a gossip from \( j \) after round \( r \) began at time \( T_r \). Let \( T_j > T_i \) be the earliest time after which no new nodes receive a gossip on \( j \) (such a moment exists since the number of nodes is finite). Let \( J \) be a subgroup of nodes that transitively hear from \( j \) at \( T_j \), \( J \neq V \) since \( i \in V \setminus J \). Since \( J \) does not grow, it means that no node \( k \in V \setminus J \) ever hears from any node \( v \in J \), neither directly, and as a result, not transitively. That is, \( J \) and \( V \setminus J \) are permanently partitioned. However, given that the failure detector does not suspect any node, it means that there are no permanent partitions. A contradiction.

Lemma 3.2. In all infinite runs of the protocol, the stability of any stable message is eventually detected by every node.
Proof: Let $m$ be a message from $j$ that had stabilized at time $T_m$. Let $r$ be a smallest round that ends after $T_m$. Since all rounds of the GossipFullDist sub-protocol are finite, such a round $r$ exists. At the beginning of round $r+1$, for every $j$, it holds that $M_t[j] \geq m$. According to the code of GossipFullDist, at the end of the round $r+1$, for every $i$ it holds that $M_t[i] \geq m$ as well. Hence, $m$’s stability is eventually detected by all nodes.

Lemma 3.3 If the FuzzyFullDist protocol at node $i$ marks a message stable, then it was received by all view members.

Proof: According to the protocol, only node $k$ can originate spreading a gossip that $k$ delivered $j$’s message $m$. Hence, if node $i$ detects that $j$’s column of a matrix $R_i$ is at least $m$, then all nodes have definitely delivered $m$.

The necessity of GossipFullDist sub-protocol GossipFullDist is a mandatory part of the protocol that ensures liveness of the protocol. In Example 3.3 we show that if FuzzyFullDist includes only VGossipFullDist, then it cannot ensure liveness in the lifetime of a given view.

Example 3.3 In this example we present a scenario under which a protocol that includes only VGossipFullDist can never detect the stability of a stable message $m$. Assume that a virtual round lasts at most $T_0 = R \ast (\Delta + \delta)$ time units. We also assume that nodes send their protocol messages simultaneously. Hence, information initiated by node $i$ can be gossiped along at most $R$ hops during a round. We also assume that a view change occurs only if a network partition is not resolved in $T' >> T_0$ time units after it occurs. An initial configuration of a network is described by a graph of $N$ vertices $V = \{1, 2, 3, .., N\}$, where for every $i < N$ there is an edge between vertices $i$ and $i + 1$, and there are no other edges. Assume that all nodes are at the beginning of the same virtual round, they have not yet sent any protocol messages, and they all have just received a message $m$. $T_1 = \frac{T_0}{N} \ast (\Delta + \delta)$ time units after a virtual round began all edges fail, and then the failed edges are reconstructed $T_0 - T_1 + 1$ time units after the failure (that is, after this virtual round ends). During the virtual round, no node can gather information from more than $\frac{2N}{3} + 1$ nodes before new virtual round starts. Hence, $m$’s stability is not detected during that round. If the same scenario happens in every virtual round, $m$’s stability is never detected.

Discussion An ad-hoc network can be very dynamic, and thus it may happen that all messages of GossipFullDist will be lost while messages of other protocols and the application will be successfully delivered. In this case, the network is not truly partitioned (for more details see Example 3.4), yet the protocol cannot detect stability. This situation is very unlikely. Yet, in order to completely avoid this possibility, messages of GossipFullDist can be occasionally piggybacked on other protocols and application messages.

Example 3.4 This example shows adversarial behavior of an ad-hoc network that is always partitioned for FuzzyFullDist messages, but successfully delivers all other messages. A simple topology example
appears in Figure 7. Node 2 is most of the time in the neighborhood of Nodes 1 and 3, as shown. When one of the nodes sends FuzzyFullDist protocol messages, Node 2 always moves as appears in Figure 8 and then returns to its previous position. As a result, each node is partitioned from the other two with respect to stability messages. Other than that, all other messages are successfully routed to all nodes.

3.3 Analysis

During one protocol step, each sub-protocol sends at most two messages — a gossip message at the beginning of a step and another message in case a round termination was detected. Actually, since the gossiping interval $\Delta$ is the same for both sub-protocols, only two messages need to be produced, where each message is a union of the sub-protocol messages. This is in contrast to the original FullDist protocol of [12], in which each node sends $n$ messages.

The number of messages received by each node during one step of the protocol is therefore at most twice the number of its neighbors (since each neighbor sends at most two messages). In the worst case, the number of neighbors is $n$, and hence the number of received messages is bounded by $2n$. In [9] we have shown that the expected number of neighbors is $N_{\text{neighbor}} = (n - 1) \cdot \pi R^2 / N^2$, where $R$ is the transmission range. Thus, the expected number of received messages is $2N_{\text{neighbor}}$. This is in comparison to the FullDist protocol, in which each node receives $n$ messages during time period equal to a protocol step.
<table>
<thead>
<tr>
<th>Protocol</th>
<th>$\text{FuzzyFullDist}$</th>
<th>$\text{FullDist}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of sent messages per node per step</td>
<td>2</td>
<td>$n$</td>
</tr>
<tr>
<td>Number of received messages per node per step</td>
<td>$2 \times (n - 1) \times \frac{\sqrt{n}}{n}$</td>
<td>$n$</td>
</tr>
<tr>
<td>Total messages size per step</td>
<td>$O(n)$</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>

Table 1: $\text{FuzzyFullDist}$ vs. $\text{FullDist}$

$\text{FuzzyFullDist}$ is more scalable on the average than the $\text{FullDist}$ protocol both in the network load and buffer space. At the same time $\text{FuzzyFullDist}$ is resilient to failures, since each protocol message is received by all the neighbors of a sender. The size of messages sent by both $\text{FuzzyFullDist}$ and $\text{FullDist}$ is $O(n)$. Also, both algorithms require $O(n^2)$ memory in order to save the sequence numbers’ matrix. The summary of comparison of $\text{FuzzyFullDist}$ and $\text{FullDist}$ appears in Table 1.

4 Experimental Performance

Clearly, the main measure for stability detection protocols’ effectiveness is the stability detection time, since this allows garbage collection in receive buffers and advancing flow control windows, as we discussed in the Introduction. We performed simulations using the GloMoSim simulator [28]. In all our simulations, all nodes are initially placed randomly inside the simulation area of size 250m x 250m. The mobility of the nodes is modeled according to the Random-Waypoint Scheme [15]. According to this model, each node travels to a randomly selected point inside the area with a speed selected randomly from the interval defined by minimum and maximum allowed speeds. Then, a node may choose to stay at that point for a randomly selected period between 0s and 90s. Each simulation lasted 30 minutes. At the MAC layer, the IEEE 802.11 protocol was used with a bandwidth of 11Mbps.

Also, the network consists of a single multicast group. All nodes join this group at the beginning of the simulation and remain members of the group until the end of the simulation. In our experiments, a single sender sends messages to all group members. Each data point in every diagram is obtained as an average of 10 different runs with 10 different network topologies and movement patterns.

We used a simple flooding protocol for the underlying multicast. That is, every node relays a message to all its members when it receives it for the first time. Recall that reliable delivery of application messages (only) is required for the stability detection protocol, in order to ensure that indeed all messages eventually stabilize. In our simulations we used the lost (application) message recovery mechanism that is described in [10].

In all measurements, each node sends an application message every second and a $\text{FuzzyFullDist}$ protocol message every 5 seconds. The protocol virtual round timeout varies from 10s to 60s in steps of 5s. The failure detector allows temporary disconnections of at most 45s before suspecting a node, while the fuzziness level starts to grow after 5s. For fuzzy-stability marking, we used the fuzziness
level threshold of $\frac{1}{8}$ the allowed disconnection period, for a total of $5s + 45s/8 = 10.5s$ (this is the maximal time a node can be non-responsive before being considered fuzzy).

### 4.1 Performance of Stability Detector vs. Number of Nodes

The simulations in this section assume a transmission range of 60m. The minimum speed is 0m/s and maximum speed is 2m/s.

Figure 9 presents maximal, average and minimal stability detection times obtained in simulations with 60 nodes. First, the stability detection time decreases as the virtual round timeout of VGossipFullDist grows. Then, after the minimum is reached for a virtual round timeout of 20s, the stability detection time starts to grow almost linearly in the virtual round timeout. If the round is short, VGossipFullDist is not very efficient as round changes are frequent, which means that previously collected stability information is no longer propagated. On the other hand, the larger the virtual round timeout
Figure 11: FuzzyFullDist - fuzziness duration time vs. number of nodes

Figure 12: FuzzyFullDist - average fuzziness duration time vs. average buffering time (30 nodes)
Figure 13: FuzzyFullDist - the ratio of messages that go through fuzzy state vs. number of nodes

Figure 14: FuzzyFullDist - the dynamics of fuzzy marking during the simulation with 30 nodes

Figure 15: Maximum stability detection time vs. number of nodes
Figure 16: Average stability detection time vs. number of nodes

Figure 17: Minimum stability detection time vs. number of nodes
is, the later nodes start to propagate information about recently received messages. The graph shows that the damage caused by a large timeout is greater than the gain from it. The average detection time is approximately half the difference between maximal and minimal detection times. This means that occurrences of short and long stability detection times are equally frequent.

Figure 10 depicts the average ratio of early terminated VGossipFullDist rounds for simulations with different number of nodes. It appears that the probability that all nodes participate in a round is in direct relation to the number of nodes, and in direct relation to the virtual round timeout. Note that for simulations with 60 nodes and a virtual round timeout of 20s (corresponding to the minimum stability detection time), at least 96% of the rounds terminated early. Even for a virtual round timeout of 60s, more than 80% of the rounds terminated early. This indicates that the early round termination option is highly beneficial.

Figure 11 presents the average time a message is marked fuzzy-stable until its stability is detected. One of the main purposes of fuzzy-stable marking is to detect messages that need long-term buffering. In a network with 30 nodes and a virtual round timeout of 20s, messages remain fuzzy-stable during about 85s. In order to evaluate the quality of this result, we need to know how long a message needs to be actually buffered. Note that the buffering time depends not only on the stability detection time, which is measured from the moment a message becomes stable, but also on the reliable delivery protocol.

Figure 12 presents the average buffering time (that is, the average time that passes from the moment a message is received until it is declared stable) and the average time that a message is marked fuzzy-stable when the network consisted of 30 nodes. For example, with an optimal virtual round timeout of 20s, messages that are marked fuzzy-stable remain fuzzy-stable for about 45% of the average buffering time. This means that if buffering optimizations would be applied to fuzzy-stable messages when they are marked as such, then buffer space can be saved during a significant period of the buffering time. Note also that the period during which messages are marked fuzzy-stable shortens as either the virtual round timeout or the number of nodes grow. The first dependence is due to the fact that messages are marked fuzzy-stable only at the end of a round, while message stability may also be detected during a round. Hence, when rounds become longer, fuzzy-stable marking occurs closer to the stability detection. As the number of nodes grow, the connectivity becomes better. That is, if a disconnection occurs, the network will become connected again sooner when the number of nodes is greater. As a result, stability detection happens sooner after fuzzy-stability detection.

Figure 13 indeed shows that the ratio of messages that go through the fuzzy-stable state decreases as the virtual round timeout grows and the number of nodes increases. With large virtual round timeouts, more messages can be detected stable before the round ends. Similarly, with a larger number of nodes, the connectivity of the network is improved, so message stability is detected faster, and hence the number of fuzzy-stable messages decreases.

We now focus on the ratio of fuzzy-stable messages obtained for simulations with 30 nodes. For virtual round timeout of 20s, about 30% of messages are marked fuzzy-stable (without becoming stable before hand). Figure 14 presents the number of messages that are marked fuzzy-stable during the simulation time. It can be seen that such markings occur in bursts. That is, there are periods when only a few messages are marked and periods when many messages are marked. This can be explained by
the behavior of an ad-hoc network. When the network is connected, almost no messages are marked fuzzy-stable. However, when temporary partitions occur, messages take longer to stabilize, and so is stability detection, causing a burst of fuzzy-stable markings. This shows that the fuzzy-stable marking of \textit{FuzzyFullDist} meets its purpose of identifying messages that require long buffering.

Figures 15, 16, 17 present maximum, average and minimum stability detection times respectively, with numbers of nodes varying from 30 to 60. The greater the number of nodes is, the smaller are the minimal, average and maximal stability detection times due to improved network connectivity. Note also that initially there is a decrease in stability detection time as virtual round timeout increases, since the information gathered by \textit{VGossipFullDist} is allowed to propagate further. Then, after a minimum is reached, it continuously grows as the virtual round timeout increases, independently of the number of nodes, which matches our expectations based on the discussion above.

4.2 Performance of Stability Detector vs. Nodes’ Speed

The simulations of this section include 40 nodes. The transmission range is set to 60m in all executions. The maximal speed of nodes varies from 2m/s to 8m/s.

Figures 18, 19, 20 present maximum, average and minimum stability detection times for various values of nodes’ maximal speed. Overall, the average and minimal stability detection times are very close for different settings of maximal speed. Maximal stability detection time, on the other hand, obviously decreases as the maximal speed grows. Interestingly, the ratio of fuzzy-stable messages (see Figure 21) is also higher when nodes move fast and the virtual round timeout is short. This is probably caused as a result of more frequent disconnections and the inability to reconstruct new routes during the short virtual round timeout. On the other hand, when the virtual round timeout becomes longer, the ratio of fuzzy-stable messages decreases as nodes move faster. In addition, when nodes move fast, the time during which messages are fuzzy-stable becomes shorter (see Figure 22). It appears that as nodes move faster, disconnections may be more common, but they are shorter, and in fact, nodes’ movement act as another mode of carrying information across different areas of the network.
Figure 19: Average stability detection time vs. nodes’ speed

Figure 20: Minimum stability detection time vs. nodes’ speed

Figure 21: FuzzyFullDist - ratio of messages that go through fuzzy state vs. nodes’ speed
4.3 Performance of Stability Detector vs. Transmission Range

The simulations of this section include 40 nodes. The maximal speed of nodes is set to 2m/s.

Figures 23, 24, 25 present maximum, average and minimum stability detection times for various values of nodes’ transmission range. As expected, a longer transmission range accounts for better connectivity, which results in faster stability detection. As appears in Figure 26, the ratio of fuzzy-stable messages decreases as the transmission range increases. For a transmission range that ensures good network connectivity (i.e., 70m and above) and optimal virtual round timeout of 20s, only about 5% of the messages become fuzzy-stable. As appears in Figure 27, the period of time during which messages are marked fuzzy-stable also decreases as the transmission range increases.
Figure 24: Average stability detection time vs. nodes' range

Figure 25: Minimum stability detection time vs. nodes’ range

Figure 26: FuzzyFullDist - ratio of messages that go through fuzzy state vs. nodes’ range
5 Conclusions

In this paper, we have presented a novel stability detection protocol, suitable for MANETs, and studied its performance by simulation. In particular, the protocol uses round based gossip mechanisms, and introduces the notions of early round termination and fuzzy stability detection. Our protocol is more efficient than previous round based gossip protocols in terms of message sizes, and more efficient than non-round based fault-resilient protocols in terms of the number of messages they generate. The protocol is now being integrated in our JazzEnsemble group communication toolkit (http://www.cs.technion.ac.il/Labs/dsl/projects/JazzEnsemble/). Our simulations validated the importance of these optimizations, and provided the following observations. As can be expected, the better the network connectivity is, either due to a larger number of nodes or due to long transmission ranges, the shorter the stability detection time is. Additionally, fast nodes movement can improve the stability detection time (and reduce the number of fuzzy stable messages) since when nodes move fast, disconnections are likely to heal quickly and, in fact, the movement itself starts acting as another way of propagating information in the network.

The early round termination optimization proved to be highly useful as even with very long virtual round timeouts many rounds ended early. This is because a failure of a single process to acknowledge a message can prevent the message from being declared stable. This is exactly why in traditional protocols “one slow process can slow down the entire group”, and one of the main strengths of our approach compared to previously studied protocols. Similarly, the simulations have shown that when a few nodes become disconnected, many messages are indeed marked fuzzy-stable. Moreover, messages that are declared fuzzy-stable remain in this state for a long time, and detection of fuzzy-stable messages occurs in bursts. This indicates that indeed flow control mechanisms and buffer management protocols that utilize fuzzy stability can gain considerable performance, and in particular alleviate the “one slow process” problem. Of course, as can be expected, the benefits of fuzzy stability becomes less significant when the network connectivity is very good.
References


