Consensus in Sparse, Mobile Ad-hoc Networks

Khaled Alekeish and Paul Ezhilchelvan

TECHNICAL REPORT SERIES

No. CS-TR-1208  July, 2010
Consensus in Sparse, Mobile Ad-hoc Networks

K. Alekeish and P. Ezhilchelvan

Abstract

Consensus is central to several applications including collaborative ones which a wireless ad-hoc network can facilitate for mobile users in terrains with no infrastructure support for communication. We solve the consensus problem in a sparse network in which a node can at times have no other node in its wireless range and useful end-to-end connectivity between nodes can just be a temporary feature that emerges at arbitrary intervals of time for any given node pair. Efficient one-to-many dissemination, essential for consensus, now becomes a challenge: enough number of destinations cannot deliver a multicast unless nodes retain the multicast message for exercising opportunistic forwarding. Seeking to keep storage and bandwidth costs low, we propose two protocols. An eventually relinquishing (<>RC) protocol that does not store messages for long is used for attempting at consensus, and an eventually quiescent (<>QC) one that stops forwarding messages after a while is used for concluding consensus. Use of <>RC protocol poses additional challenges for consensus, when the fraction, (f/n), of nodes that can crash is: (1/4)<=(f/n)<(1/2). Consensus latency and packet overhead are measured through simulations and both decrease considerably even for a modest increase in network density.

© 2010 University of Newcastle upon Tyne.
Printed and published by the University of Newcastle upon Tyne,
Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.
Bibliographical details

ALEKEISH, K., EZHILCHELVAN, P.
Consensus in Sparse, Mobile Ad-hoc Networks
[By] K. Alekeish, P. Ezhilchelvan
(University of Newcastle upon Tyne, Computing Science, Technical Report Series, No. CS-TR-1208)

Abstract
Consensus is central to several applications including collaborative ones which a wireless ad-hoc network can facilitate for mobile users in terrains with no infrastructure support for communication. We solve the consensus problem in a sparse network in which a node can at times have no other node in its wireless range and useful end-to-end connectivity between nodes can just be a temporary feature that emerges at arbitrary intervals of time for any given node pair. Efficient one-to-many dissemination, essential for consensus, now becomes a challenge: enough number of destinations cannot deliver a multicast unless nodes retain the multicast message for exercising opportunistic forwarding. Seeking to keep storage and bandwidth costs low, we propose two protocols. An eventually relinquishing (<>RC) protocol that does not store messages for long is used for attempting at consensus, and an eventually quiescent (<>QC) one that stops forwarding messages after a while is used for concluding consensus. Use of <>RC protocol poses additional challenges for consensus, when the fraction, (f/n), of nodes that can crash is: (1/4)<=(f/n)<(1/2). Consensus latency and packet overhead are measured through simulations and both decrease considerably even for a modest increase in network density.

About the author
Khaled Alekeish is a PhD student within the School of Computing Science, Newcastle University
Paul Devadoss Ezhilchelvan received Ph.D. degree in computer science in 1989 from the University of Newcastle upon Tyne, United Kingdom. He received the Bachelor of Engineering degree in 1981 from the University of Madras, India, and the Master of Engineering degree in 1983 from the Indian Institute of Science, Bangalore. He joined the School of Computing Science of the University of Newcastle upon Tyne in 1983 where he is currently a lecturer.

Suggested keywords
MANET
NETWORK DENSITY
NODE CONNECTIVITY
CRASH-TOLERANCE
CONSENSUS
QUIESCENT MULTICASTING
COVERAGE ASSURANCE
Consensus in Sparse, Mobile Ad-hoc Networks
Khaled Alekeish, Paul Ezhilchelvan

Abstract—Consensus is central to several applications including collaborative ones which a wireless ad-hoc network can facilitate for mobile users in terrains with no infrastructure support for communication. We solve the consensus problem in a sparse network in which a node can at times have no other node in its wireless range and useful end-to-end connectivity between nodes can just be a temporary feature that emerges at arbitrary intervals of time for any given node pair. Efficient one-to-many dissemination, essential for consensus, now becomes a challenge: enough number of destinations cannot deliver a multicast unless nodes retain the multicast message for exercising opportunistic forwarding. Seeking to keep storage and bandwidth costs low, we propose two protocols. An eventually relinquishing (\( QC \)) protocol that does not store messages for long is used for attempting at consensus, and an eventually quiescent (\( RC \)) one that stops forwarding messages after a while is used for concluding consensus. Use of \( RC \) protocol poses additional challenges for consensus, when the fraction, \( \frac{1}{n} \), of nodes that can crash is: \( \frac{1}{4} \leq \frac{1}{n} < \frac{1}{2} \). Consensus latency and packet overhead are measured through simulations and both decrease considerably even for a modest increase in network density.

Index Terms—Manet, network density, node connectivity, crash-tolerance, consensus, quiescent multicasting, coverage assurance

1 INTRODUCTION

MOBILE ad-hoc networking (Manet) technology is perhaps unique in its ability to facilitate collaboration amongst mobile users in terrains that have no fixed infrastructures for communication support. One of the challenging problems in supporting collaboration is to enable the users to reach an identical collective decision while their preferred options are all different but equally appropriate. This problem, considered in a failure-prone context, is widely known as the consensus [1]. The broader aim of our work is to build a consensus module for supporting collaborations in a Manet.

Of the several consensus protocols proposed for a Manet, only a few have been subject to performance evaluation. A closer look at these works reveals that they all assume a network density that would leave the Manet almost always connected. Network density can be defined as the average number of nodes within a disc of radius equal to the nodes’ wireless range. Informally, two nodes are said to be directly connected if they are in each other’s wireless range, and simply connected if they are either directly connected or a sequence of one or more directly connected nodes is between in them. (In Fig 1(a), \( N_i, 1 \leq i \leq 4 \), are connected to each other, where \( \leftrightarrow \) is a direct connectivity.) Thus, the denser the network, the more likely that any two nodes remain connected despite node mobility.

Table 1 lists the density values used for studying the consensus protocol performance. (Those of [2] were adjusted to match our density definition.) A density value of, say, 8.7 means that each node is expected to be directly connected with 7.7 other nodes at any moment. Even when nodes move at moderate speeds, any two nodes will have, at any moment, at least one path connecting them and also lasting long enough for supporting message transmission. (In [3], connections last longer also for acknowledgements to be received.)

At the opposite end to dense networks is the delay tolerant networking (DTN) [6]. Density is so low that a multi-hop connectivity is rare and direct connectivity between a given node pair can take an arbitrarily long time to emerge. So, routing a message, say, \( m \) to a given destination itself is a challenge with DTN (see [7] for a survey). It typically relies on opportunistic forwarding as Fig 1(b) illustrates: direct connections that commence at different (and arbitrary) instances, \( t_1, t_2, t_3 \) are treated as ‘opportunities’ for forwarding \( m \) from \( N_1 \) to \( N_2 \), with intermediaries \( X_1 \) and \( X_2 \) willing to retain \( m \) and wait for such an opportunity. Message latencies are typically long and are not a criterion for measuring the efficacy of a routing strategy [7]. To our knowledge, consensus has not been solved with DTN.

This paper solves consensus for Manets of density values larger than DTN but not large enough to keep the network always connected. The Manet we consider keeps distinct subsets of nodes connected at distinct intervals, and the sequence of connectivity formed over some (unknown) period of time, would allow any node to transmit its message to any other node if opportunistic forwarding is used. Referring to Fig 1(c), connectivity involving \( \{N_1, X_1, X_2, N_2\} \), \( \{N_1, X_3, X_4, N_3\} \) and \( \{N_1, X_5, X_6, N_4\} \) prevail during intervals \( (t_1, t'_1) \), \( (t_2, t'_2) \) and \( (t_3, t'_3) \) respectively, and these intervals may be arbi-
The earlier works listed in Table 1 use a class of consensus protocols that require that at least one node in G remains connected to all other nodes in G for a sufficiently long duration. Figure 2(a) suggests that a sparse Manet cannot be relied upon to meet this requirement: the probability that a node at any moment is connected to, say, more than 80% of nodes is only 10% (1 − P(DR ≤ 0.8) ≈ 0.1). So, we will use a randomized consensus protocol [8] that imposes no such requirement.

A consensus protocol, randomized or otherwise, requires that when a node multicasts its initial proposal, at least a majority of collaborating nodes receive it. With this probability being as small as 40% in Fig 2(a), consensus requires a multicast support that indulges in opportunistic forwarding which in turn requires nodes to retain messages potentially for long durations. Moreover, nodes typically engage in several rounds of message exchange before reaching consensus and this increases the number of messages being retained.

So, we develop two types of multicast protocols to support consensus: an eventually relinquishing (◊RC) protocol that is guaranteed to discard a message once a target on DR is achieved; and an eventually quiescent (◊QC) protocol that is guaranteed not to transmit a message once a target on DR is achieved, but may retain the message. The use of the latter is hence kept to the minimum necessary. (More on Fig 2(b) in Subsection 2.4.)

The third and final challenge arises due to an interplay between the possibility of node crashes and the maximum DR the ◊RC protocol can guarantee. Even if the latter delivers a multicast message to a majority of nodes, some of these nodes could subsequently crash, leaving the number of operative nodes that received the multicast to fall below the threshold needed for consensus to make progress. To deal with such unfortunate possibilities, the consensus protocol of [8] will have to be appropriately adapted.

The paper is organized as follows. Next section describes the system context, fault and connectivity assumptions, specifications of multicast protocols and our approach to reaching consensus. Sections 3, 4 and 5 respectively present the consensus, ◊QC and ◊RC protocols and rigorous correctness arguments. Finally, Section 6 evaluates the cost and delay in reaching consensus.
2 System Model and The Approach

The system $S$ is made up of mobile nodes collaborating towards a common goal in a terrain that has no fixed infrastructure for supporting communication between nodes. The nodes communicate using the omnidirectional wireless transmission functionality of a CSMA/CA-like MAC layer protocol (e.g., IEEE 802.11b). Exchange of information between nodes is thus limited strictly to ad-hoc networking.

A small group $G$ of $n$ nodes is formed at time $t_0$ for the purpose of reaching consensus whenever a unanimous choice out of different possibilities needs to be made during the collaboration. Nodes of $(S,G)$ cooperate to discover and maintain connectivity between nodes of $G$; that is, nodes of $(S,G)$ act as routers for nodes of $G$ to exchange messages and execute consensus.

We assume that nodes of $S$ have unique identifiers and let $G=\{N_1, N_2, \ldots, N_n\}$ and $S-G=\{X_1, X_2, \ldots\}$. We also assume that $n \geq 3$ and $|S-G| \gg n$. Nodes of $G$ are referred to as consensus nodes or simply as nodes, and those of $S-G$ as routing nodes or routers, for short.

Of the $n$ nodes of $G$, at most $f$, $0 < f < \frac{n}{2}$, can crash over the lifetime of $G$ and a crashed node does not recover. Thus, a node is either working or crashed permanently. A working node is also referred to as an operative node and functions according to its specification. Every operative node knows the identity of all other nodes in $G$ and also knows $f$ and $n$.

We define $W(t)$ as the set of nodes that are operative at time $t > t_0$. Since a crashed node does not recover, $W(t)$ is non-increasing over time: $W(t') \subseteq W(t)$, $\forall t' \geq t$.

Since $f < \frac{n}{2} < n$, $G$ has at least $(n-f)$ nodes that never crash. These nodes are called permanently working nodes or simply the correct nodes of $G$. $PW$ denotes the set of all correct nodes. $PW \subseteq W(t)$ for all $t \geq t_0$. Consistent with fault-assumptions in the consensus literature, we will assume that it is the adversary who solely decides which nodes of $G$ crash and when.

2.1 Node Connectivity

Consider two operative nodes that are in the wireless range of each other. Let $\delta$ be the maximum delay that any one of them may take to transmit an application message to the other, despite possible collisions and interferences. The nodes are said to be $1$-hop connected or simply $1$-Connected at time $t \geq t_0$, if they remain operative and also in each other’s range at least until $(t+\delta)$. Note that $1$-Connectivity at time $t \geq t_0$ is a binary relation on $W(t+\delta)$ which is reflexive, symmetric and intransitive.

Two operative nodes, $N_i$ and $N_j$, are said to be $h$-Connected if a path of at most $h$, $h \geq 1$, $1$-Connections, exists between them for the next $h\delta$ duration at least. $N_i$ and $N_j$ are said to be $(B,h)$-Connected if they are operative and $h$-Connected for a duration of length at least $B$, where $B > h\delta$ is a parameter specified by the application.

The intuition behind $(B,h)$-Connectivity is that $h$-connectivity between two nodes is useful to an application, only if it lasts for an additional duration of at least $(B-h\delta)$ time. Applications that are of interest here are the multicast protocols used by the consensus protocol. Note that, by definition, $(B,h)$-Connectivity will imply $(B,h')$-Connectivity for $h' > h$ if $B + (h' - h)\delta \approx B$, i.e., if $\delta$ is negligibly small compared to $B$. If $(B,h)$-Connectivity $\Rightarrow$ $(B,h')$-Connectivity, then $(B,h)$-Connectivity is a reflexive, symmetric and transitive relation.

2.2 Liveness Requirement

From an application’s perspective, a node $N_i$ is permanently isolated from the rest of $G$ starting from time $t$, if it never enjoys $(B,h)$-Connectivity with any operative node for any $h$, $h \geq 1$, at any time after $t$. Of course, the above notion of node isolation assumes that $N_i$ remains operative and that the absence of $(B,h)$-Connectivity emergence is not because $N_i$ crashed sometime after $t$.

Solving consensus requires that operative nodes are not isolated permanently; the Manet must fulfill a liveness condition that eliminates such permanent node isolations. This condition can be informally stated as follows. For every $t$, $t \geq t_0$, any $N_i \in W(t)$ that remains operative for a ‘long enough’ duration will have $(B,h)$-Connectivity for some $h$, $h \geq 1$, with some operative node $N_j$ before $t + h\delta$, where $I_{h_\delta}$, $I_{h} \geq B$, is finite but unknown.

Observe that there are two unknowns in the informal statement of the liveness condition: $h$ and $I_h$. To keep the protocol design tractable, we assume that only $I_h$ is unknown and the condition holds for all $h$, $h \geq 1$. More precisely, our assumption will be as stated below:

For every $t$, $t \geq t_0$, any $N_i \in W(t)$ that remains operative for a ‘sufficiently long’ time will make $(B,h)$-Connectivity, for every $h$, $h \geq 1$, with some operative node before $t + I_{h}$, where $I_{h_\delta}$, $I_{h} \geq B$ is finite but unknown.

When $(B,h)$-Connectivity is assured for all $h \geq 1$, applications can choose to work with a particular value of $h$, say $H$. Note that if $H$ is chosen to be small, $(B,H)$-Connectivity may take longer to emerge. That is, the smaller the $H$, the more likely that $I_H$ is large and the more delay-tolerant the application needs to be.

Formally, the liveness condition assumed can be stated as:

$$\forall t \geq t_0, \forall h \geq 1, \exists I_h, B \leq I_h \neq \infty :$$

$$(\forall N_i \in W(t + I_h), \exists N_j, N_j \in W(t + I_h))$$

$$(1) \quad N_i \text{ and } N_j \text{ are } (B,h_\delta)\text{-Connect at some time in } [t, t + I_{h}]$$

In words, for every $t$, $t \geq t_0$, for every $h \geq 1$, there exists a finite but unknown $I_{h_\delta}$, $I_h \geq B$, such that for every $N_i \in W(t + I_{h})$ there exists $N_j \in W(t + I_{h})$ which $(B,h)$-Connects with $N_i$ at some time during the interval $[t + I_{h_\delta}, t + I_h]$.

Node isolation is a special case of group partitioning in which a subset, say, $G'$, of operative nodes are unable to $(B,h)$-Connect with any of the operative node not in $G'$. To avoid $G'$ from being permanently partitioned, the Manet must allow some node in $G'$ to $(B,h)$-Connect with some operative node in $(G-G')$ before $t + I_{h_\delta}$, for all $h \geq 1$. So
generalizing (1) gives the required Liveness Condition that is assumed to be satisfied by the Manet so that $G$ is never permanently partitioned from the application’s perspective:

**Liveness Condition (LC)**

$$\forall t_0, \forall h \geq 1, \exists I_h, B \leq I_h \neq \infty :$$

$$\forall G', W(t + I_h) \supseteq G' \neq \emptyset, \exists N_i, N_j :$$

$$(N_i \in G', N_j \in W(t + I_h) - G' \land N_i, N_j (B,h)\text{-Connect at some time in } [t, t + I_h])$$

In words, for every $t$, $t \geq t_0$, for every $h \geq 1$, there exists a finite but unknown $I_h$, $I_h \geq B$, such that for every non-empty $G' \subset W(t + I_h)$ the following holds: there are operative nodes $N_i$ and $N_j$ such that $N_i \in G'$, $N_j \in W(t + I_h) - G'$ and $(B,h)$-Connectivity exists between them starting from some time in the interval $[t, t + I_h]$.

**Remark.** $I_h = \infty$, $\forall h \geq 1$ and $I_H = B$, for some finite $H$, represent two extreme cases of interest.

When $I_h = \infty$, $\forall h \geq 1$, an application will never see correct nodes having $(B,h)$-Connectivity for durations that are as long as it needs. That is, it is pointless for the application to be delay indulgent because the Manet is never going to oblige its requirement.

On the other hand, $I_H = B$ means that, at every $t$, new $(B,H)$-Connectivity between some $N_i \in G'$ and some $N_j \in W(t + I_h) - G'$ is emerging, or an existing $(B,H)$-Connectivity prolongs beyond $t$ for a further $B$ time or more, or both. That is, the Manet keeps the operative nodes of $G$ always $(B,H)$-Connected.

### 2.3 Multicast Services

A multicast protocol is said to be eventually quiescent ($\Diamond Q$, for short), if it ensures that there is a time after which operative nodes permanently stop disseminating a multicast message.

A multicasting protocol is said to be eventually relinquishing ($\Diamond R$), if it ensures that there is a time after which operative nodes do not retain a multicast message for dissemination purposes.

Note that $\Diamond R \Rightarrow \Diamond Q$. Basic flooding and its variations, such as [9], are both $\Diamond Q$ and $\Diamond R$. However, they cannot guarantee anything on the number or the set of nodes that would deliver a multicast message. We will use the term coverage to refer to the number or the type of nodes that deliver a multicast message. We will define two classes of protocols that offer the maximum possible guarantee on coverage while also being $\Diamond Q$ or $\Diamond R$ respectively.

In expressing the protocols’ properties (and also throughout the paper), we will use the term deliver to refer to a node taking possession of a message by executing a protocol. We will use $m$ to denote a message; we will also assume that a node that initiates multicasting of $m$ also delivers its own $m$. The term transmit will be used for referring to a node’s MAC level, wireless transmitting of a packet that can be received by all devices (nodes and routers) that are in the wireless range.

**Definition.**

$\Diamond QC$ is a class of multicast protocols which satisfy:

1. dissemination of $m$ stops permanently within some finite time after multicast of $m$ is initiated $(\Diamond Q)$, and
2. in any execution in which at least one correct node delivers $m$, all correct nodes deliver $m$ (coverage, C).

**Remark 1:** Executions in which no correct node delivers $m$ involve nodes crashing shortly after delivering $m$ and not executing the protocol long enough for $m$ to reach a correct node. They have $\Diamond Q$ and $\Diamond R$ features, by default. So, we will limit our interest to those executions in which at least one correct node delivers $m$.

**Remark 2:** No multicast protocol can offer a stronger coverage guarantee than 2) above, unless it also implements uniform delivery property. The latter requires delaying delivery of $m$ until it is deduced that at least $(f+1)$ nodes can deliver $m$. This incurs time and bandwidth overhead.

As per [10], transforming a $\Diamond QC$ protocol into a $\Diamond R$ equivalent with the same coverage guarantee as 2), is not possible unless operative nodes can accurately know which nodes have crashed. Such a knowledge is in turn not possible because $I_h$ in (2) can be arbitrarily long. So, a crashed node is indistinguishable from an operative one that is out of other nodes’ wireless range for an arbitrarily long period of time. So, 2) must be weakened for the $\Diamond R$ counterpart.

**Definition.**

$\Diamond RC$ is a class of multicast protocols which satisfy:

1. nodes that deliver $m$ discard $m$ within some finite time after multicast of $m$ is initiated $(\Diamond R)$, and
2. in any execution in which at least one correct node delivers $m$, at least $(n-f)$ nodes deliver $m$ (coverage, C).

**Remark 3:** All nodes that deliver $m$ need not be correct, and at most $f$ can go on to crash after delivering $m$. So, in the worst case, just $(n-2f)$ correct nodes may deliver $m$. It is shown below that the 2) above is the strongest coverage guarantee any $\Diamond R$ protocol can offer.

**Lemma 1:** In any execution in which at least one correct node delivers $m$, a $\Diamond R$ multicast protocol that has no support for detecting node crashes, cannot guarantee that more than $(n-f)$ nodes deliver $m$ if more than $(n-f)$ nodes are operative throughout that execution.

**Proof** By Contradiction. Suppose that there exists a $\Diamond R$ protocol that cannot detect node crashes but nevertheless guarantees that more than $(n-f)$ nodes deliver $m$ if more than $(n-f)$ nodes are operative throughout that execution.

**Execution 1:** All nodes of $F$ have crashed before $t_0$. Let $t_s$, $t_s > t_0$, be the timing instance before which any node that delivers $m$ has relinquished $m$. $t_s$ must exist since the protocol is $\Diamond R$; also, no more than $(n-f)$ nodes can deliver $m$. 
Execution 2: No node crashes before or after \( t_b \). However, the Manet keeps all nodes of \( F \) outside the wireless range of every node in \( F \) at least until \( t_e \). This is possible if, in this execution, the unknown \( I_h > t_e - t_b \) for every \( h \geq 1 \); that is, \( LC \) is met for \( I_h > t_e - t_b \).

Nodes of \( F \) cannot distinguish execution 2 from execution 1 until \( t_e \) for three reasons: (1) the multicast initiator is in \( F \), (2) nodes of \( F \) do not execute the protocol for \( m \) (certainly until \( t_e \) in execution 2), and (3) nodes of \( F \) cannot detect whether a node in \( F \) is crashed or operative.

So, in execution 2, as in execution 1, only at most \( (n - f) \) nodes can deliver \( m \) with all delivering nodes discarding \( m \) by \( t_e \); additionally, nodes of \( F \) do not execute the protocol until \( t_e \) and there is no \( m \) after \( t_e \) for them to deliver. This is a contradiction, since at least \( f \) nodes do not deliver \( m \) even though all \( n \) nodes are operative.

\( \Box_{\text{Lemma 1}} \)

2.4 Approach to Consensus and the Rationale

Given that every operative node in \( G \) proposes an initial value or initial estimate, a consensus protocol satisfies the following properties which lead to nodes arriving at an identical decision despite their potentially different initial estimates:

- **Validity**: If a node decides \( v \), then \( v \) was proposed by some node.
- **Agreement**: No two nodes decide differently.
- **Termination**: Every correct node decides.

As discussed in Section 1, consensus is typically solved using multicast protocols that assume that the network remains connected always (or almost always), and assure that correct nodes deliver each other’s messages with a probability that is 1 (or close to 1, respectively). Using these consensus solutions as such for sparse Manets would involve deploying a \( QC \) protocol for multicasting. This could lead to an unaffordable storage overhead for two reasons: the \( QC \) protocols can retain messages for ever and consensus may take several rounds of message exchange, using a large number of \( QC \) multicasts.

An alternative approach, which is pursued here, will be to use a \( RC \) multicast protocol and address the challenges that arise thereof. In particular, a \( RC \) protocol, in the worst case, can deliver a correct node’s \( m \) just to \( (n - 2f) \) correct nodes (Remark 3 in Subsection 2.3); consequently, if \( (n - 2f) \) is not a majority in \( n \), which will be the case if \( n \leq 4f \), then executions of a traditional consensus protocol may deadlock. Choosing the alternative approach requires addressing these issues.

It is also not entirely feasible to rule out the use of a \( QC \) protocol. When a correct node decides during an execution of a consensus protocol, it is common for another correct node not to be able to decide at or around the same time; the latter needs to be ‘helped’ by the former by sending the decision to it. A \( QC \) protocol, will be used (only) for this purpose. Figure 2(b) depicts the role of \( QC \) and \( RC \) protocols in our approach.

3 Consensus Protocol

3.1 The EMR Protocol

Consensus protocol will be derived from that by Ezhilchelvan, Mostefaoui and Raynal [8]. The latter will be referred to as the EMR protocol and works using dissemination primitives designed for a (connected) wired network. The protocol derivation will address the implications that arise when \( QC \) and \( RC \) protocols are used instead.

The communication facilities used by EMR protocol are: a uniform reliable broadcast service, \( RCast \) for short, and a simple, non-crash-tolerant broadcast service. Both operate within \( G \) and are \( R \). \( RCast \) ensures that even if an operative node delivers \( m \) and subsequently crashes, \( m \) is delivered by every correct node. It thus offers delivery guarantees stronger than \( QC \) multicast whose coverage guarantee applies only in executions where at least one correct node delivers \( m \).

The simple broadcast service in [8] ensures that all correct nodes deliver \( m \) if the broadcasting of \( m \) by the source node is not interrupted by a crash; otherwise, some correct nodes may deliver \( m \), while some others may not. This guarantee is stronger than that of \( RC \) in which even a correct node’s \( m \) is not necessarily delivered by all other correct nodes.

The EMR protocol mostly uses simple broadcasting and reserves \( RCast \) only for the following two activities. First, each node \( N \), \( RCast \) its initial value to every other node at the start of the protocol and collects the values delivered through \( RCast \) in a bag called ValueBag or VBag. Secondly, if \( N \) decides, it \( RCast \) its decision to others so that the latter can decide swiftly.

The main part of the execution involves \( n \)-to-\( n \) message exchanges until some node can decide. It is briefly described as follows. It proceeds in rounds, with each round having two phases. During the first phase of a round, every node broadcasts a value called its estimate, \( est \) for short, and waits to obtain estimates from a majority of nodes in \( G \). If estimates from a majority are the same, that identical value is adopted as the node’s new estimate; otherwise a default value \( \bot \) (different from any possible estimate) is taken as the new estimate.

During the second phase, as in the first, nodes exchange their estimates and wait to obtain estimates from a majority of nodes of \( G \). If a node obtains the same value from a majority, it decides on that value and \( RCast \) its decision. Otherwise, it proceeds to the first phase of the next round with a new estimate computed as follows: if it has obtained at least one \( v, v \neq \bot \), \( v \) is its new estimate; else, i.e., if all the values obtained are \( \bot \), the node chooses its new estimate randomly from its VBag.

The EMR protocol achieves termination with probability 1 by using two facts: all operative nodes eventually have identical VBag (thanks to the uniform delivery property of \( RCast \)) and there is a small probability that all nodes randomly select the same value from their
**3.2 Protocol Derivation and Challenges**

The new protocol is derived from the EMR protocol by addressing the implications of replacing RBcast and broadcast support with QC and RC multicasting respectively; moreover, the use of QC multicasting is kept to a minimum.

3.2.1 Using QC Multicast

It is done only for disseminating the decision value. By the definition of consensus, the value decided must be the same, irrespective of which or how many nodes decide. Therefore, several deciding nodes initiating QC multicast need not be distinguished based on sender identifiers and can be easily optimized to equivalent of handling a single QC multicast.

The new consensus protocol does not require nodes to exclusively disseminate their initial estimates for the sake of building identical VBags. However, it requires, just like EMR, that operative nodes eventually have identical VBags. While EMR achieves this requirement through accumulation of values disseminated by RBcast, the new protocol does it through elimination of values from VBags so that all operative nodes eventually have identical VBags containing a single value. The scheme is explained below.

Nodes make a random selection at the end of each phase, concurrently to their attempts to decide based on est exchanged as per the logic of the EMR protocol. Each \( N_i \) maintains a variable \( cand \), which is its input candidate for the random selection process. \( cand_i = est_i \) for phase 1, round 1.

At the start of each phase, \( N_i \) sends \( cand_i \) together with its \( est_i \). Once \( N_i \) delivers \( \{ est, cand \} \) pair from a majority of nodes (including its own), VBags becomes the set of all distinct \( cand \) values \( N_i \) delivered in the current phase. A value selected randomly from VBags becomes the new \( cand \), which will be \( N_i \)'s input candidate for the selection to be held at the end of next phase. It is later shown that if random selections are repeated frequently enough, VBags of operative nodes decrease in size and eventually become identical singleton sets, returning the same \( cand \) after a ‘random’ selection.

3.2.2 Using RC Multicast

At least \((n-\alpha)\) nodes deliver a RC multicast \( m \). In the worst case, \( f \) of those that \( RC_{\text{delivered}} m \) may crash, leaving only \((n-2f)\) correct nodes with \( m \). This worst-case possibility may block a correct node delivering \( m \) from a majority of nodes, when \( n-2f \leq \frac{\alpha}{2} \), i.e., for all \( n, \frac{2f+1}{n} \leq \frac{\alpha}{4} \). Note that if a node cannot \( RC_{\text{deliver}} \{ est, cand \} \) pair from a majority of nodes, it cannot complete a phase. (Completing a phase means deciding or moving onto the next phase/round.) A deadlock arises when no node can complete a phase and this is illustrated by the scenario described below.

Consider \( G = \{N_1, N_2, N_3\} \) and \( f = 1 \). Say, \( N_1 \) and \( N_2 \) deliver a message for round \( r \) and phase \( ph \), denoted as \((r, ph)\) message for short. Let \((r, ph)\) messages from \( N_1 \) and \( N_2 \) be \( RC_{\text{delivered}} \{ \{ N_1, N_3 \}, \{ N_2, N_3 \} \} \) respectively. Suppose that \( N_3 \) crashes after \( RC_{\text{delivering}} \{ \{ N_1, N_3 \}, \{ N_2, N_3 \} \} \) message of its own. Both \( N_1 \) and \( N_2 \) cannot now deliver two \((r, ph)\) messages, if their own \((r, ph)\) messages have been discarded by the \( RC \) multicast protocol.

To break the deadlock, it is necessary for operative nodes to repeat their \( RC_{\text{mcasting}} \) \((r, ph)\) messages at regular intervals, if they judge themselves being unable to deliver \((r, ph)\) messages from a majority of nodes. In the above scenario, the \((r, ph)\) messages \( RC_{\text{mcast}} \) by \( N_1 \) and \( N_2 \) after \( N_3 \) has crashed will allow \( N_1 \) and \( N_2 \) to make progress.

This repetitive \( RC_{\text{mcasting}} \) of \((r, ph)\) messages must be done in a judicial manner, mindful of overheads involved. Our protocol manages this using a time-driven mechanism that can also be event-driven one when \( 2f+1 < n \leq 4f \): a node distinguishes its \( RC \) multicasts for a given \((r, ph)\) message using a message field, called the attempt sequence number and denoted as \( \alpha, \alpha \geq 1 \). The \( RC \) multicast system will treat \( RC \) multicasts from a given node with distinct \( \alpha \) as distinct multiset messages. Every time a node \( RC_{\text{delivers}} \) \((r, ph)\) message, it increments a count by 1. When the count reaches \((n-2f)\), the \((r, ph)\) message will be \( RC_{\text{mcast}} \) after \( \Delta \) time and the count is decremented by \((n-2f)\).

In the scenario considered earlier with \( G = \{N_1, N_2, N_3\} \) and \( f = 1 \), only \( \Delta \)-driven mechanism is effective as \( n = 2f+1 \): both \( N_1 \) and \( N_2 \) will perform a \( RC_{\text{mcast}} \) once every \( \Delta \) interval until \( N_3 \) crashes or no longer delivers the \( RC_{\text{mcasts}} \) from \( N_1 \) and \( N_2 \).

It is shown later that if a correct node begins a phase \( ph \) of round \( r \), then some correct node \( RC_{\text{delivers}} \) \((r, ph)\) from a majority of nodes, so long as no correct node is decided. The proofs make no assumption as to how long a faulty node, such as \( N_3 \) in the earlier scenario, can take to crash or to stop delivering correct nodes’ repeated \( RC \) multicasts; they only require the interval to be finite.

3.3 The Protocol

The protocol is expressed in Figure 3 as a function that takes a node \( N_i \)'s proposed estimate and returns the decision \( v \) (line 4). It assumes the use of QC and RC protocols which respectively provide \( QC_{\text{mcast}} \) and \( RC_{\text{mcast}} \) for multicasting \( m \), and \( QC_{\text{deliver}} \) and \( RC_{\text{deliver}} \) for delivering a multicast \( m \) exactly once.

The round-based activities are carried out by thread \( C() \) which is activated in line 2 of Fig 3 for phase \( ph = 1 \) of round \( r = 1 \), \((r = 1, ph = 1)\) for short. Thread activation supplies five input parameters which are all initialized in line 1 of Fig 3, where, it should be noted, \( cand_i \) and \( est_i \) are both set to \( N_i \)'s own proposal \( v_i \).
Thread $C(t)$ is designed to die after either deciding or spawning a new instance of itself for a later phase. If decision is made, it is $\Diamond QC$ multicasting which, when $\Diamond QC\_delivered$ (line 3), terminates Function consensus.

Function consensus ($\nu_i$)
1. $r_i = ph_i = \alpha_i = 1$; $est_i = cand_i = v_i$;
2. activate Thread $C(r_i, ph_i, \alpha_i, est_i, cand_i)$;
3. wait until $\Diamond QC\_delivered$ (decision, $v$) occurs;
4. return ($v$);

Fig. 3. The Consensus Function

Thread $C(r_i, ph_i, \alpha_i, est_i, cand_i)$
1. while (true) do
2. $m = (r_i, ph_i, \alpha_i, est_i, cand_i); \Diamond RC\_mcast (m)$;
3. Bag$_i = \{i\}; V\ Bag = \{\}$; count$_i = 0;
4. while ($|Bag_i| \leq 2$) do
5. while (count$_i < n - 2f$) do
6. (wait until $(r_j, ph_j, \alpha_j, est_j, cand_j)$ $\Diamond QC\_delivered$ ($(r_j = r_i \land ph_j > ph_i) \lor (r_j > r_i)$);
7. (if $(r_j = r_i \land ph_j > ph_i) \lor (r_j > r_i)$ then
8. { activate Thread $C(r_j, ph_j, 1, est_j, cand_j)$; die; }
9. count$_i$ + 1;
10. if $est_j \not\in V\ Bag_j$ then enter $est_j$ in $V\ Bag_j$;
11. if $(\exists j, est_j > \notin Bag_j$ then enter $\notin Bag_j$;
12. if $(|Bag_j| > 2)$ then { exit }
13. if $(count_i \geq n - 2f \land |Bag_j| \leq 2)$ then
14. { count$_i$ = count$_i$ + ($n - 2f)$; $\alpha_i = \alpha_i + 1$;
15. schedule $\Diamond RC\_mcast (r_i, ph_i, \alpha_i, est_i, cand_i)$ at clock $\Delta$
16. } cancel any pending $\Diamond RC\_mcast()$;
17. cand$_i$ = PickRandom($V\ Bag_j$);
18. if (ph$_i = 1$) then
19. { if ($v = est_j$ for all $<j, est_j >$ in $Bag_j$) then $est_i = v$;
20. else $est_i = v$;
21. ph$_i$ = 2; $\alpha_i = 1$; }
22. if (ph$_i = 2$) then
23. { if ($v = est_j$ for all $<j, est_j >= Bag_j$) then $\Diamond QC\_mcast$ (decision, $v$); die; }
24. else if ($v = est_i$ $\land i \neq j$ for some $<j, est_j >= Bag_j$) then
25. $est_i = v$;
26. else $est_i = cand_i$;
27. $r_i = r_i + 1; ph_i = 1; \alpha_i = 1$; }
25. $\theta_j$ + 1; ph$_i$ = 1; $\alpha_i = 1$;
27. }
25. }
24. }
22. }
21. }
20. }
19. }
18. }
17. }
16. }
15. }
14. }
13. }
12. }
11. }
10. }
9. }
8. }
7. }
6. }
5. }
4. }
3. }
2. }
1. }
16. }
15. }
14. }
13. }
12. }
11. }
10. }
9. }
8. }
7. }
6. }
5. }
4. }
3. }
2. }
1. }

Fig. 4. Pseudo-Code for Consensus Thread

The pseudo-code for thread $C()$ of some $N_i \in G$ for a given $(r, ph)$, $r \geq 1$ and $ph \in \{1, 2\}$, is presented in Figure 4. It can be understood in three parts: $N_i$ $\Diamond RC$ multicasting own $(est, cand)$ (line 2), awaiting $(est, cand)$ pairs to be $\Diamond RC\_delivered$ from a majority of nodes (lines 3-15) and acting on the $(est, cand)$ pairs delivered (lines 16-27).

To deduce the termination of waiting in part 2, Bag$_i$, is maintained for storing est$_j$ that was $\Diamond RC\_delivered$ from $N_j \in G$, as a 2-tuple $<j, est_j>$. Having initialized Bag$_j$, V Bag$_j$ and count$_j$ (in line 3), the thread waits (line 6) to $\Diamond RC\_deliver$ a $m_j$ from any $N_j \in G$ for some $(r_j, ph_j)$ that is either future to or the same as $(r_i, ph_i)$.

A future $m_j$ will have either $(r_j = r_i$ and $ph_j > ph_i$) or $r_j > r_i$ (line 7) and will expedite the execution: a new thread instance is created with future $m_j$ as the input (except for $\alpha_i$ which is initialized to 1), and the current thread dies (line 8).

If $m_j$ is current with $r_j = r_i$ and $ph_j = ph_i$, count$_i$ is incremented (line 9); Bag$_j$ and $V\ Bag_j$ are updated, if $<j, est_j>$ and est$_j$ are not already present, respectively (lines 10-11). If count$_i \geq n - 2f$ without termination of part 2 (lines 12-13), then a repeat of $\Diamond RC\_mcast(m)$ after $\Delta$ time is scheduled (line 15), once $m.\alpha$ is increased by 1 and count$_i$ decreased by $(n - 2f)$ (line 14).

Once Bag$_j$ has more than $\frac{n}{2}$ entries, part 3 begins: any pending $\Diamond RC\_mcast(m)$ is canceled (line 16) and cand$_i$ is set to a random value picked from $V\ Bag_j$ (line 17). The rest of part 3 faithfully implements the EMR logic as described in §3.1 for both $ph = 1$ and 2, except for one aspect: when $ph = 2$, if est$_i$ has to be a random selection then est$_i$ is simply set to cand$_i$ (line 27).

3.4 Correctness Arguments

They will be presented in two parts, focusing on the two major changes we have made to the EMR protocol [8]. The first part addresses the implications of using $\Diamond RC$ multicast and will establish that, as in the EMR protocol, every operative node obtains estimates from a majority of nodes in any phase unless it is expedited. The second part is about the equivalent support for eventual termination: with probability 1, operative nodes eventually have the same value for cand$_i$ and thus enabling them, as in the EMR protocol, to start a round with the same estimate. The rest of the proofs simply follow that in [8] and only an outline is provided for completeness.

Throughout this subsection, we consider an execution in which all correct nodes $\Diamond RC\_mcast$ their initial estimates (in line 2, Fig 4) for phase $ph = 1$ and round $r = 1$. For the first part, we choose the context where no node is decided and some phase $ph$ and round $r$, $r \leq 1, (r, ph)$ for short, is the latest phase for which at least one correct node has $\Diamond RC\_delivered$ a message $m = (r, ph, *, *, *)$. That is, no node is (yet) to execute a later $(r', ph')$ such that $r' > r$ or $ph' > ph$ if $r' = r$.

We define time $t_s$ as the moment when the first correct node delivered a $(r, ph)$ message; $t_s$ must exist at least for $(r = 1, ph = 1)$. We show that if nodes stop crashing for a sufficiently long period of time after $t_s$, then some node $\Diamond RC\_delivers (r, ph)$ messages from a majority of nodes and thus can complete phase $ph$ of round $r$.

We define $t_1, t_2 \geq t_s$, as the moment when operative nodes stop crashing; that is, $W(t) = W(t_1), \forall t \geq t_1$. Note that $(t_1 - t_s)$ may be arbitrarily long. A crashed node may have delivered any number of $\Diamond RC$ messages for $(r, ph)$ and have acted on none. Proofs are structured as follows.

Lemma 2 in Appendix A proves that the consensus protocol keeps the operative nodes continually
\( RC_{mcast} \) (in line 15 of Fig 4) a \((r, ph)\) m (for increasing values of \(\alpha\)) during \([t_l, t_f]\) so that when crashes stop occurring at \(t_f\), completion of phase \((r, ph)\) is facilitated.

Lemma 3 in Appendix A establishes that there exists an instance \(t_2 > t_1\) such that at least \((n - f)\) nodes \( RC_{mcast} (r, ph) \) m at least once during \([t_1, t_2]\). Note that \(n - f > \frac{n}{2}\). That a majority of nodes \( RC_{mcast} (r, ph) \) messages, does not automatically mean that some node must deliver \((r, ph)\) messages from a majority of nodes at some time after \(t_2\); this is because a correct node’s \( RC\) multicast is not necessarily \( RC\) delivered by all other correct nodes. Lemma 4 in Appendix A proves what is required.

The node that delivers \((r, ph)\) messages from a majority of nodes, should either decide or continue to the next phase by \( RC_{mcast} \) for \((r', ph')\) where \( r' = r \) and \( ph' = 2 \) if \( ph = 1 \) or \( r' = r + 1 \) and \( ph' = 1 \) if \( ph = 2 \); in the latter case, some correct node must deliver the \((r', ph')\) message. Thus, by recurrence, it is shown that nodes complete any phase, if no node is decided and no node is executing a later phase.

**Termination.** For this part, we will consider a phase in which no node decides. We define \( CAND^0 \) and \( CAND^1 \) be the set of all \( cand \) values the operative nodes have at the start and the end of that phase, respectively. Lemma 5 and its corollary (both in Appendix A) together establish that the probability that \( CAND^1 \) is a singleton set, i.e., \(|CAND^1| = 1\), given that \(|CAND^0| > 1\), is not zero.

More precisely, Lemma 5 proves (by contradiction) that the probability that \( CAND^1 = CAND^0 \) cannot be 1 when \(|CAND^0| > 1\). Since \(|CAND^0|, |CAND^1| \leq n\), is finite, the probability that \( CAND^1 \) is a singleton set given that \( CAND^1 \subset CAND^0 \), cannot also be zero. Using these observations, the corollary of Lemma 5 establishes what needs to be shown.

Note that \(|CAND^1| = 1\) at the end of any given phase is a stable property: \(|CAND^1| = 1\) is also true at the end of every undecided phase that follows, because a selection from a singleton set can only return one value.

In any execution, \(1 \leq |CAND^0| \leq n\). Let \( p_i, 1 \leq i \leq n \), be defined as the probability that \(|CAND^1| = 1\) given that \(|CAND^0| = i\). As noted above, \( p_1 = 1 \) and \( p_i > 0, \forall i : 2 \leq i \leq n\). Let \( p = \min (p_1, p_2, \ldots, p_n) \).

Let \( P(\phi) \) be the probability that \(|CAND^1|\) becomes 1 by the end of \( \phi, \phi > 0\), undecided phases, given that \( cand \) values of operative processes were different at \( ph = 1 \) and \( r = 1.1 - P(\phi) \leq (1 - p)^2\); i.e., \( P(\phi) \geq 1 - (1 - p)^2\). As \( \lim_{\phi \to \infty} P(\phi) = 1\), all operative nodes must eventually start a phase with the same \( cand \) value.

Lemma 3 in [8] proves that if all operative processes start a given round with the same estimate in phase 1 of that round, they must decide on that estimate. Using this lemma and the assertion that random selections leads to an identical outcome with a non-zero probability, Theorem 2 in [8] proves that every correct node eventually decides with probability 1.

The two-phase structure within a round and the intersecting nature of majority subsets are used in [8] to prove that if a node decides \( \phi \) in round \( r\), then any node that does not decide in that round, must start any round \( r' > r \) with \( est = \phi \); i.e., at the end of phase 2 of round \((r' - 1)\), an undecided node cannot set its \( est \) to a random selection in [8] or cannot execute \( est = cand \) in line 27 of Fig 4. This property is used in proving that no two nodes can decide differently (Theorem 1 in [8]).

### 4 A QC MULTICAST PROTOCOL

#### 4.1 A Basic Protocol

The major challenge in designing a QC protocol is in preventing a node \( N_i \) that delivered \( m \) from disseminating \( m \) once the following property is met: all nodes that were operative at some time have delivered \( m \). Note that it is a stable property (once true, always true); but a node cannot deduce it without an ability to identify all crashed nodes. There lies the challenges of QC design and a basic protocol is outlined below.

\( N_i \) maintains knowledge regarding which other nodes have \( m \). This knowledge is denoted as \( K_i(m) \) or simply \( K_i \). \( K_i \) is maintained such that if it indicates that \( N_i \) has \( m \), then it is certainly true; it need not be true the other way round: when \( N_i \) does have \( m \), \( K_i \) may indicate that \( N_i \) does not have \( m \). Obviously, \( K_i \) is undefined if \( N_i \) has never ‘seen’ \( m \), i.e., never delivered \( m \).

\( N_i \) also maintains \( H\)-hop neighborhood view, \( Neigh_H \), which contains all other nodes that are connected to \( N_i \) by at most \( H \) hops, for some chosen \( H \geq 1 \). Note that a node \( N_j \) may get connected to, then disconnected from, and reconnected to \( N_i \). Consequently, \( N_j \) may appear, then disappear and reappear in \( Neigh_H \) of \( N_i \).

Whenever \( N_j \) appears in \( Neigh_H \) of \( N_i \) or re-appears after a period of disappearance, \( N_i \) is said to encounter \( N_j \). (When \( N_i \) encounters \( N_j \), \( N_j \) may either already have \( N_i \) in its \( Neigh_H \) or encounter \( N_i \) as well.) On encountering \( N_j \), \( N_i \) checks its \( K_j \) to see if \( N_j \) has \( m \). If not, \( N_i \) uncasts \( m \) to \( N_j \). \( N_j \) will do the same, if it has \( m \); otherwise, it receives \( N_i \)’s unicast and unicasts back an acknowledgement packet, \( ack \) for short. When \( N_i \) receives \( m \) or \( ack \) from \( N_j \), it includes \( N_j \) in its \( K_i \).

By the liveness requirement (LC in subsection 2.2), operative nodes that delivered \( m \) cannot for ever be partitioned from those that have not delivered \( m \); some of the former set must (\( B, H \)-Connect) with some node of the latter, facilitating transfer of \( m \). So, if a correct node delivers \( m \), all other correct ones must deliver \( m \).

Suppose that at some time after delivering \( m \), \( N_i \) has encountered every other node that has been operative until that time. From that time onwards, \( N_i \) will not encounter a node that has not delivered \( m \) as per its \( K_i \); \( N_i \) will therefore cease unicasting \( m \). (\( N_i \) may not know this and hence will retain \( m \).) QC can thus be ensured, if it can be shown that such a moment in time must exist for every operative node.

Before going quiescent, a node could unicast \( m \) to every other node; so, the message overhead of the basic
In the improved version, $N_i$ maintains not just $K_i$ but additional information as to whether other nodes have the same $K$ as its own. The latter is called knowledge about $K$ and is denoted as $KK_i$. If an encountered $N_j$ is deemed not to have the same $K$ as $K_i$, then $N_i$ unicasts only $K_i$ to $N_j$. If $N_i$ has not delivered $m$, it learns of the existence of $m$ and requests for $m$ to be unicast. Thus, $N_i$ unicasts $m$ to a $H$-hop neighbor only on request.

$K_i(m)$ is a vector of $n$ boolean bits: $K_i(m)[j] = 1$ if $N_i$ knows that $N_j$ has delivered $m$, $K_i(m)[j] = 0$ if $N_i$ does not know if $N_j$ has delivered $m$ or not. Note that once $K_i(m)[j]$ becomes 1, it cannot become 0 again.

Let $|K_i(m)|$ denote the total number of 1s in $K_i(m)$. We define a bit-wise OR operator, denoted as $\lor$ for convenience, between two $K$ vectors, $K_1$ and $K_2$, which returns a $K$ vector whose $j^{th}$ bit is $K_1[j] \lor K_2[j]$, $\forall j, 1 \leq j \leq n$. Further, $N_i$ maintains its $KK_i$ also as a vector of $n$ boolean bits: if it knows that $N_j$ has the same $K(m)$ as itself, then $KK_i(m)[j]$ is set to 1; otherwise, $KK_i(m)[j]$ will be 0. Both $K_i(m)$ and $KK_i(m)$ are formed when $N_i$ first delivers $m$ and are initialized with all bits being set to 0 except the $j^{th}$ bit which is set to 1.

Note that, unlike in $K_i(m)$, the number of 1s in $KK_i(m)$ can decrease, because contents of $KK_i(m)$ will change every time $K_i(m)$ changes. Similarly, while $KK_i(m)[j] = 1$, $N_i$ may have added more 1s to its $K(m)$ without $N_i$ being aware of this addition. So, the only certainty that $N_i$ can derive from $KK_i(m)[j]$ being 1 is that $N_j$ had the same $K(m)$ as itself in the past.

Finally, each $N_i$ maintains a list $L_i$ of $3$-tuples for each delivered message $<m.id, K(m), KK(m)>$: it also maintains a message pool $M_i$, in which all delivered $m$ are kept for possible unicasting to an encountered node. The protocol specifies $B$ for $(B, H)$-Connectivity and also chooses $\beta$ as: $\beta + H\delta < \frac{B}{2}$. Protocol related packets and messages are not processed on their arrival but at discrete instances: the protocol thread sleeps for a random interval on $(0, \beta)$; when it wakes up, it processes all packets delivered to it while it was asleep and then goes to sleep again.

Control Packets: In addition to the $ack$ packets, following types are also used: a $Kpkt(m)$ simply contains $m.id$ and the transmitting node’s $K(m)$ and a $Req(m)$ packet is used for requesting $m$ to be unicast to the requester.

Support Services: The $QC$ protocol described here uses three, simple underlying services. (Subsection 4.4 describes how we implemented these services.) The first service provides an up-to-date view on the $H$-hop neighborhood $Neigh_H$ which would typically be changing due to node mobility. The second service provides the unicast support using which a node can send $m$ to a specific node in its $Neigh_H$. It is assumed to offer two primitives: $unicast()$ and $unideliver()$ for sending and delivering unicast messages, respectively.

The third service is optional and it can be any $R$ protocol such as basic flooding. Using this service reduces the delivery latency of $QC$ multicast messages, albeit at the expense of message overhead. Its presence offers a leverage for making a trade-off between latency and overhead. $R_{mcast()}$ and $R_{deliver()}$ are the two primitives assumed for multicasting and delivering messages through a $R$ protocol. If no $R$ protocol is to be used, then a $R_{mcast}(m)$ invocation will simply cause $R_{deliver()}$ to return $m$ only to the invoker.

### 4.3 Protocol Description

The pseudo-code for $QC\_mcast(m)$ is shown in figure 5 where $K(m)$ is initialized and $QC\_mcast()$ to disseminate the $(m, K(m))$ pair as a single message entity.

$QC\_deliver()$ has three concurrent threads (see Fig. 6). Thread 1 responds to a $R_{deliver}()$ of $(m, K(m))$ by delivering $m$ to the local application(s), initializing $KK(m)$ and entering $m$ and $<m.id, K(m), KK(m)>$ in $M_i$ and $L_i$, respectively (lines 2-3).

Threads 2 and 3 both operate on a sleep-and-act basis. They process packets concerning $m$ for which a $<m.id, K, KK>$ is and is not present in $L_i$, respectively. When Thread 2 wakes up, it performs 5 tasks for every $<m.id, K, KK>$ in $L_i$.

In Task 2.1, a $Kpkt$ (containing $K_i$) is unicast to any neighbor $N_j$ for which $KK_i[j] = 0$ (lines 6,7). Task 2.2 processes each incoming $Kpkt$ containing $K_j$: it involves incorporating the knowledge in $K_j$ with the local $K_i$ (by setting $K_i = K_i \lor K_j$) and reinitializing $KK_i$ appropriately (lines 8-11). Task 2.3 responds to any $Req(m)$ delivered during sleep. Tasks 2.4 and 2.5 cater for (special) situations that allow $m$ and $<m.id, K, KK>$ to be relinquished, when $|K_i| = n$ and $|KK_i| = n$ hold, respectively. For $|K_i|$ to become $n$, all nodes of $G$ must deliver $m$ and not crash until $N_i$ becomes aware of it; similarly, $|KK_i| = n$ (line 15) can come true only after $|K_i|$ reaches $n$ and if nodes do not crash until $N_i$ becomes quiescent.

Thread 3 carries out two Tasks. Task 3.1 responds to having delivered a $K\_pkt$ for a $m$ that has no entry in $L_i$. A $Req(m)$ is unicast to the source node of that $K\_pkt$, unless the $K$ vector within the $K\_pkt$ is a vector of all 1s. The latter implies that all nodes have already delivered $m$; the entry for $m$ is missing in $L_i$ because the local Thread 2 has already executed line 15 while the source node of the $K\_pkt$ is yet to have $|KK_i| = n$. Returning a $K\_pkt$ with a vector of all 1s (line 19) enables the remote node to have $|KK_i| = n$ eventually.

Task 3.2 deals with a $unideliver(m, K(m))$ for which there is no entry in $L_i$: this has to be the first $unideliver$ since a $Req(m)$ had been $unicast$ earlier. $m$ is delivered to applications and appropriate entries are made in $L_i$ and $M_i$ so that Thread 2 can handle dissemination of $m$. protocol is $O(n^2)$. In the improved version described below, $m$ is disseminated only $O(n)$ at the expense of using more control packets such as $ack$. 
4.4 Implementing Support Services

Constructing $\text{Neigh}_H$, for $H = 1$, takes advantage of MAC level beacons being periodically sent by mobile devices to announce their presence to their immediate (1-hop) neighbors. Node $N_i$ retains or enters $N_j$ in its $\text{Neigh}_1$ if a beacon is received from $N_j$ and removes $N_j$ from its $\text{Neigh}_1$ if a few consecutive beacons (3 in our implementation) are missing.

For $H > 1$, nodes must explicitly transmit a $\text{Ghello}$ packet for every $\beta$ interval with the time-to-live (TTL) field set to $H$. A transmitted packet is received by all devices, routers and nodes, that are within the wireless range of the transmitting node.) A router node that receives a $\text{Ghello}$ packet reduces TTL by 1 and, if TTL > 0, appends its own identifier to the packet which is then transmitted. When $N_i$, receives a $\text{Ghello}$ from another $N_j$, it adds or retains $N_j$ in its $\text{Neigh}_H$ and records the reverse of the appended identifier sequence as a route for unicasting $m$ to $N_j$. Membership of $\text{Neigh}_H$ and a route information are lost, unless renewed within $3/\beta$ time.

For $\Diamond R$ multicasting, we use an adaptation of our earlier work on encounter based broadcasting [11]. The original work treats $G$ as $S$ itself. We chose to adapt [11] for three reasons: it performs well for a wide range of node speeds and network densities, requires $O(\ln(n))$ transmissions of $m$ and no control packets, and uses only $\text{Neigh}_1$ that can be maintained with just MAC level beacons and does not require $\text{Ghello}$ packets.

In [11], a node transmits $m$ whenever it encounters any other node in its $\text{Neigh}_1$. The expected number of these encounter-based transmissions needed for all destinations to receive $m$ turns out to be: $\tau = 2(\ln(|S|) + \gamma)$, where $\gamma = 0.5772...$ is the Euler-Mascheroni’s number. A node terminates the protocol once it deduces that at least $\tau$ transmissions of $m$ have taken place.

$$\Diamond QC\_multicast(m)$$

{ initialize $K(m)$ for $m$;
 $\Diamond R\_multicast(m, K(m))$;
 }

Fig. 5. $\Diamond QC$ multicast

A simple adaptation of [11] for $G \subset S$, will be to allow a node $N_i \in G$ to transmit $m$ only when it encounters another $N_j \in G$ in its $\text{Neigh}_1$, and do nothing if the encounter involves a router node $X \in (S - G)$. But it could turn out to be highly delay-indulgent when $|G| \ll |S|$. So, we allow occasional flooding of $m$.

$N_i$ initiates a flooding of $m$ if the number of consecutive encounters that did not involve any $N_j \in G$ (i.e., involved only router nodes) exceeds a threshold called the encounters-to-flood and denoted as $E2F$. The value of $E2F$ is also adaptively varied, similar in style to additive increase and multiplicative decrease in TCP congestion control, so that flooding is rarely resorted to. A node doubles its $E2F$ soon after it has initiated a flood; this reduces the chances of doing another flood before protocol termination; to prevent $E2F$ staying too large due to an absence of necessary encounters caused by rare, 1-off network conditions, $E2F$ is reduced by 1 whenever a node in $G$ is encountered after $\tau$, $\tau < (E2F - 1)$, successive encounters involving only routers.

Note that the $\Diamond R$ protocol is used for disseminating $(m, K(m))$ pair (not just $m$). This is because the $\Diamond R$ protocol is assumed to update the $K(m)$ in the received $(m, K(m))$ to include the local node before the received $(m, K(m))$ is transmitted; furthermore, if, after having $\Diamond R\_delivered(m, K(m))$, which must be exactly once, the $\Diamond R$ protocol receives $(m, K(m))$ with $K(m)$ having more information, then the latter is passed to Thread 2 in the form of a $K\_Pkt$.

4.5 Correctness Arguments

Claim 1: If $B > 4(\beta + H\delta)$, then a $(B,H)$-Connectivity is long enough to support the sequence of information exchange triggered by the $\Diamond QC$ protocol.
Suppose that \( N_t \) has delivered \( m \) and \( N_j \) has not, when 
\((B,H)\)-Connectivity between them begins at time, say, \( t \). The following sequence of events can occur, each with the worst possible delay:

1. \( N_t \) emits \textit{Hello} at \( t + \beta \) which reaches \( N_j \) at \( t + \beta + H\delta \), ensuring the presence of \( N_j \) in \textit{Neighbor} of \( N_t \);
2. Thread 2 of \( N_t \) wakes-up at \( t + (\beta + H\delta) + \beta \) and unicasts a \( K_P\textit{Pkt} \) which reaches \( N_j \) by \( t + 2(\beta + H\delta) \);
3. Thread 3 of \( N_t \) processes the \( K_P\textit{Pkt} \) from \( N_j \) at \( t + 2(\beta + H\delta) + \beta \) and unicasts a \textit{Req}(\( m \)) which

\[ n_j \text{ reaches } N_j \text{ at } t + 3(\beta + H\delta) \text{; and,} \]
4. Thread 2 of \( N_j \) responds to \textit{Req}(\( m \)) at \( t + 3(\beta + H\delta) + \beta \) and \( m \) arrives at \( N_j \) at \( t + 4(\beta + H\delta) < t + B \).

\textbf{Claim 2:} Consider two operative nodes \( N_i \) and \( N_j \). If \( K_i(m) \neq K_j(m) \) at some time \( t \) during an execution, then both \( K_{K_i}(m)[j] \) and \( K_{K_j}(m)[i] \) cannot be 1 at \( t \); at least one of them must be 0. Two cases must be considered: by time \( t \), (i) only one node has delivered \( m \), and (ii) both have delivered \( m \).

For case (i), let us assume, with no loss of generality, that \( N_i \) has delivered \( m \) and \( N_j \) has not. So, \( K_i(m) \) does not exist. This means that \( N_i \) cannot have received a \( K_P\textit{Pkt} \) from \( N_j \) by \( t \); \( N_i \) cannot set \( K_{K_i}(m)[j] \) to 1 without first receiving one. So, \( K_i(m)[j] = 0 \).

To consider the second case, suppose, with no loss of generality, that \( K_{K_i}(m)[i] = 1 \) at \( t \). This is possible only if \( N_j \) has processed a \textit{uniledelivered} \( K_P\textit{Pkt} \) from \( N_i \) at or before \( t \) and has found \( K_i(m) = K_j(m) \). But at \( t \), \( K_i(m) \neq K_j(m) \). That is, \( N_j \) has changed its \( K_i(m) \) since it had \textit{unicast} its \( K_P\textit{Pkt} \), which must have caused its \( K_{K_i}(m) \) to be initialized (line 9, Fig 6). Further, \( N_i \) could not have learnt that \( N_j \) also changed its \( K_i(m) \) exactly in the same way as it did, because, by \textit{given}, \( K_i(m) \neq K_j(m) \) at \( t \). Therefore, \( K_{K_i}(m)[j] \) cannot be 1, and can only be 0.

By both the claims, if \( N_i \) and \( N_j \) \((B,H)\)-Connect during an execution of the \( QC \) protocol and if at least one of them has \( m \) before the \((B,H)\)-Connectivity commences, then at least one 0 in \( K_i(m) \) or \( K_j(m) \) becomes 1 (irreversibly), unless \( K_i(m) = K_j(m) \) at the start of the \((B,H)\)-Connectivity itself.

Using this observation and the Liveness Condition (2) in Sub-section 2.2, Lemma 6 in Appendix B proves that there must exist a timing instance \( t_Q \), following a correct node delivering \( m \) during an execution, such that all nodes of \( W(t_Q) \) have identical \( K(m) \). This proves that all correct nodes, which must be in \( W(t_Q) \), \( QC\textit{deliver} m \); moreover, when two nodes of \( W(t_Q) \) \((B,H)\)-Connect, neither will \textit{unicast} \textit{Req}(\( m \)) and hence \( m \) to the other.

5 \( QC \) RC Multicast Protocol

It is derived from the \( QC \) protocol and the derivation will be driven by the two important ways \( RC \) differs from \( QC \): message relinquishing and a restricted coverage guarantee of only at least \( (n - f) \) nodes.

\textbf{Definition.} A node is said to \textit{realize} \( m \) if it knows that at least \( (n - f) \) nodes have delivered \( m \).

Note that the definition does not require a node to have delivered \( m \) in order to realize \( m \). Thus, in the \( RC \) protocol, a node \( N_i \) can have \( K_i(m) \) without having \( m \) in \( M_i \) or without ever having delivered \( m \); i.e., with \( K_i(m)[i] = 0 \).

Central ideas behind derivation are 2-fold. On realizing \( m \), \( N_i \) discards \( m \) from its \textit{Msg-pool} \( M_i \) (if \( N_i \) had delivered \( m \)). It continues to execute the protocol only to spread the realization of \( m \) so that \( m \) is not retained in \( M \) of operative nodes of \( G \). Three simple changes to the code are needed to put these ideas into effect.

\textbf{Fig. 7. Changes in Task 3 for RC}.

First, in line 14 of Fig 6, Task 2.4 of Thread 2 must execute: if \( (K_i(m) \geq n - f \land m \notin M_i) \) then \textit{delete} \( m \) from \( M_i \). Once \( m \) is deleted from \( M_i \), a node cannot respond to \textit{Req}(\( m \)). So, generation of \textit{Req}(\( m \)) by Thread 3 in response to \textit{unidelivering} \( K_P\textit{Pkt}(m) \) is suppressed, if the \( K \) contained in \( K_P\textit{Pkt}(m) \) indicates that the source has realized \( m \). So, the second change is in Task 3.1 of Thread 3 is shown in Fig 7.

Even with Thread 3 carefully suppressing the generation of \textit{Req}(\( m \)), Thread 2 may have \textit{Req}(\( m \)) \textit{uniledelivered} to it for a short period after \( m \) has been realized. This is explained below.

Say, \( |K_i| < n - f \) when Thread 2 of \( N_i \) just begins to execute Task 2.1. Suppose that a \( K_P\textit{Pkt}(m) \) is \textit{unicast} to \( N_i \) containing \( K_i, K_i < n - f \). Next, the thread processes the \( K_P\textit{Pkt} \) \textit{uniledelivered} to \( N_i \) in Task 2.2. Say \( |K_i| \) increases due to this processing and \( N_i \) realizes \( m \). Suppose also that \( N_i \) discards \( m \) straight after realization.

Meanwhile, Thread 3 of \( N_j \) \textit{unicasts Req}(\( m \)) to \( N_i \) based on the \( K_P\textit{Pkt}(m) \) it \textit{uniledelivered} with \( K_i, |K_i| < n - f \). \( N_i \) cannot respond to \textit{Req}(\( m \)) coming from \( N_j \). To avoid this situation, Task 2.4 discards a realized \( m \) only after the elapse of a grace period - specified in terms of a number, \textit{Grace}(\( m \)), of the timeout intervals.

\textbf{Correctness Arguments.} They follow mainly from the fact that executions of \( QC \) and \( RC \) protocols are identical until some node realizes and remains operative. Given that no node has (yet) realized in an execution of the \( QC \) protocol in which a correct node delivers \( m \), the claims and the arguments in Subsection 4.5 are used by Lemmas 7 and 8 (Appendix C) to show that at least one correct node must realize \( m \) after delivering it.

Lemma 9 uses this result and the Liveness Condition (2) in Sub-section 2.2 to show that there is a timing instance \( t_R \) such that all nodes operative at \( t_R \) have realized \( m \);
i.e., all operative nodes stop retaining $m$ at some time during any execution of interest.

6 A Performance Study

Table 2 lists the important simulation parameters used for studying the consensus protocol performance. 50 mobile nodes of $S$ were randomly placed in a fixed size terrain. The network density was varied by varying the nodes’ wireless range as 100, 150 and 200 meters, resulting in the density values shown in the Table.

| TABLE 2 |
| Simulation parameters |
| Simulator | SWANS [12] |
| Area size | 1000m x 1000m |
| Mobility style | Random Waypoint |
| Pause time | 0s |
| $n = |G|$, $f$ | 50, 10, 3 |
| Density (wireless range in m) | 1.6 (100), 3.5 (150), 6.3 (200) |
| Maximum Node Speed | 5 m/s - 40 m/s |
| Simulation runs | 1000 |
| Pathloss model | Free-Space |

A simulation for a particular set of parameters involves 1000 runs using distinct random seeds. Thus, a point in the graphs we present is the average on measurements taken over 1000 runs. Moreover, each run commenced after 1000 seconds of node movement to avoid any initial bias in node placement.

Consensus protocol always starts with each node proposing a distinct initial estimate. Also, at the start, three nodes are randomly chosen for crashing at randomly chosen moments. $\Delta$ in the consensus protocol was chosen to be 10 seconds - a large value because the failure scenario (see § 3.2.2) that calls for repeated $RC$ multicasting of a given $(r, ph)$ message was judged to occur rarely. (It was never observed in our study.)

The value of $\beta$ in $\Delta QC$ and $\Delta RC$ protocols was varied as: 2, 5, and 10 seconds. No significant difference in performance was observed and the results presented here use $\beta = 5$. The value of $H$ in $\text{Neigh}_H$ is 2, if a node has an unrealized multicast, whether it is $\Delta QC$ or $\Delta RC$; $H$ is set to 1, once all on-going multicasts are realized. Finally, when a node begins to execute a given phase, it kills any on-going $\Delta RC$ multicasts for an earlier phase.

We measured 5 performance metrics for 3 different densities, and 8 different maximum node speeds. They are presented in three categories: number of rounds, time overhead and packet overhead.

**Average number of rounds** taken for reaching consensus is the only metric that benefited from low density and remained immune to node speeds. It was around 3.1, 2.4 and 2.2 for densities 6.3, 3.5 and 1.6 respectively. (It showed a noticeable fluctuation with node speed only for density value 6.3.)

It appears that when the network is denser, more nodes deliver each other’s estimates and simultaneously complete a round/phase. As the density drops, a few nodes complete a round/phase much ahead of the rest, resulting in them expediting the slow ones and, in doing so, enforcing their random choice on to the latter. Consequently, estimates of different nodes converge faster and consensus is reached in fewer rounds.

**Time overhead** was measured in terms of (i) time taken for the first nodes to decide, and (ii) time taken for nodes to become totally quiescent. The latter occurs when all operative nodes have identical $K K$ for a multicast $m$ and hence not even a control packet (such as $K pkt$ or $Chello$) needs to be unicast for that $m$. Figures 8(a) and 8(b) show these latencies respectively.

The lower the density, the longer the latencies, in particular, the longer it takes to complete a few rounds. Increasing the node speed (up to a threshold) helps reduce the latencies. This is because as the speed increases ($B > 20 m/s$, $H \geq 2$)-Connectivity is formed quickly; beyond the threshold, connectivity does not last for the required $B$ duration due to fast node movement. Longest latencies are observed when density and maximum speed are at their smallest, 1.6 and 5 m/s; it takes 20 and 34 minutes for the first node to decide and for total quiescence, respectively. Doubling the density value reduces latencies by more than half at all node speeds.

**Packet overhead.** We measure the average control and data packets per node as: total number of control/data packets transmitted (by nodes and routers) until total quiescence, divided by $|S| = 50$. A data packet refers to any $m$ containing $(est, cand)$ pair or the decision, while $|S|$ pair or the decision, while $H$ is the value of $H$ in $\text{Neigh}_H$ is 2, if a node has an unrealized multicast, whether it is $\Delta QC$ or $\Delta RC$; $H$ is set to 1, once all on-going multicasts are realized. Finally, when a node begins to execute a given phase, it kills any on-going $\Delta RC$ multicasts for an earlier phase.

We measured 5 performance metrics for 3 different densities, and 8 different maximum node speeds. They are presented in three categories: number of rounds, time overhead and packet overhead.

**Average number of rounds** taken for reaching consensus is the only metric that benefited from low density and remained immune to node speeds. It was around 3.1, 2.4 and 2.2 for densities 6.3, 3.5 and 1.6 respectively. (It showed a noticeable fluctuation with node speed only for density value 6.3.)

It appears that when the network is denser, more nodes deliver each other’s estimates and simultaneously complete a round/phase. As the density drops, a few nodes complete a round/phase much ahead of the rest, resulting in them expediting the slow ones and, in doing so, enforcing their random choice on to the latter. Consequently, estimates of different nodes converge faster and consensus is reached in fewer rounds.

**Time overhead** was measured in terms of (i) time taken for the first nodes to decide, and (ii) time taken for nodes to become totally quiescent. The latter occurs when all operative nodes have identical $K K$ for a multicast $m$ and hence not even a control packet (such as $K pkt$ or $Chello$) needs to be unicast for that $m$. Figures 8(a) and 8(b) show these latencies respectively.

The lower the density, the longer the latencies, in particular, the longer it takes to complete a few rounds. Increasing the node speed (up to a threshold) helps reduce the latencies. This is because as the speed increases ($B > 20 m/s$, $H \geq 2$)-Connectivity is formed quickly; beyond the threshold, connectivity does not last for the required $B$ duration due to fast node movement. Longest latencies are observed when density and maximum speed are at their smallest, 1.6 and 5 m/s; it takes 20 and 34 minutes for the first node to decide and for total quiescence, respectively. Doubling the density value reduces latencies by more than half at all node speeds.

**Packet overhead.** We measure the average control and data packets per node as: total number of control/data packets transmitted (by nodes and routers) until total quiescence, divided by $|S| = 50$. A data packet refers to any $m$ containing $(est, cand)$ pair or the decision, while $|S|$ pair or the decision, while $H$ is the value of $H$ in $\text{Neigh}_H$ is 2, if a node has an unrealized multicast, whether it is $\Delta QC$ or $\Delta RC$; $H$ is set to 1, once all on-going multicasts are realized. Finally, when a node begins to execute a given phase, it kills any on-going $\Delta RC$ multicasts for an earlier phase.

We measured 5 performance metrics for 3 different densities, and 8 different maximum node speeds. They are presented in three categories: number of rounds, time overhead and packet overhead.

**Average number of rounds** taken for reaching consensus is the only metric that benefited from low density and remained immune to node speeds. It was around 3.1, 2.4 and 2.2 for densities 6.3, 3.5 and 1.6 respectively. (It showed a noticeable fluctuation with node speed only for density value 6.3.)

It appears that when the network is denser, more nodes deliver each other’s estimates and simultaneously complete a round/phase. As the density drops, a few nodes complete a round/phase much ahead of the rest, resulting in them expediting the slow ones and, in doing so, enforcing their random choice on to the latter. Consequently, estimates of different nodes converge faster and consensus is reached in fewer rounds.

**Time overhead** was measured in terms of (i) time taken for the first nodes to decide, and (ii) time taken for nodes to become totally quiescent. The latter occurs when all operative nodes have identical $K K$ for a multicast $m$ and hence not even a control packet (such as $K pkt$ or $Chello$) needs to be unicast for that $m$. Figures 8(a) and 8(b) show these latencies respectively.

The lower the density, the longer the latencies, in particular, the longer it takes to complete a few rounds. Increasing the node speed (up to a threshold) helps reduce the latencies. This is because as the speed increases ($B > 20 m/s$, $H \geq 2$)-Connectivity is formed quickly; beyond the threshold, connectivity does not last for the required $B$ duration due to fast node movement. Longest latencies are observed when density and maximum speed are at their smallest, 1.6 and 5 m/s; it takes 20 and 34 minutes for the first node to decide and for total quiescence, respectively. Doubling the density value reduces latencies by more than half at all node speeds.

**Packet overhead.** We measure the average control and data packets per node as: total number of control/data packets transmitted (by nodes and routers) until total quiescence, divided by $|S| = 50$. A data packet refers to any $m$ containing $(est, cand)$ pair or the decision, while all other packets are counted as control packets (which do not however include MAC level beacons).

Control packet overhead shows a similar trend as the latencies, suggesting that the longer it takes to reach consensus, the more control packets are being expended. The data packet overhead is fairly constant over node speeds but influenced very much by the network density.
7 Conclusion


We addressed the challenges in a systematic manner, beginning with formally stating the liveness condition that a sparse Manet must support. This requirement assumes that only B and δ are known; this is similar to assumptions in [16] where a contact oracle outputs B and queueing and traffic demand oracles δ. Design of multicast and consensus protocols presented here are major extensions of our earlier works [17], [18] where there was no multicasting as G = S and the protocol of [8] was not adapted for the challenging range of L values.

The simulation runs, though carried out in a cluster, were quite time-consuming because of low density values used. For density values smaller than 1,6, our protocol will work if the liveness condition is met but one should expect latencies to be in the order of hours, in particular, with slow moving nodes. A consensus-centric application, it appears, has to be well and truly delay-tolerant in sparser Manets. As a future work, we would like to investigate how our protocol performance compares with others, such as [4] and [5], when the network becomes moderately dense.

References


APPENDIX A

LEMMAS FOR CONSENSUS PROTOCOL

To recap, the lemmas are stated and proved considering an execution in which all correct nodes participate and in the following context: no node is decided and some phase ph and round r, r ≥ 1, (r, ph) for short, is the latest phase for which at least one correct node has ✷RC-delivered a message m = (r, ph), s, ♦, ♦. That is, no node is (yet) executing (r′, ph′) such that r′ > r or ph′ > ph if r′ = r. ts is the moment when the first correct node delivered a (r, ph) message, and t1, t1 ≥ ts, is the moment when nodes stop crashing.

Definitions. For all t ≥ ts, we let A(t) as the set of nodes that have ✷RC-mcasts at least one (r, ph) message before time t. That is, nodes of A(t) have been active in phase ph of round r before t. We let a(t) = |A(t)|. Similarly, we denote a (r, ph) message simply as m when the context is obvious.

A(t) = \{N_k, 1 ≤ i ≤ a(t) : N_k executed ✷RC-mcast(m) before t for some \(a(t) ≥ 1\). Finally, we let \(\Delta_t\) be some unknown bound on the latency for ✷RC multicast messages: m is delivered at any destination within \(\Delta_t\) time after ✷RC-mcast of m is initiated.

Lemma 2: So long as no node delivers (r, ph) message m from a majority of nodes in G, there is at least one correct node that ✷RC-mcasts m during \([t, t + \Delta_t + \Delta]\) for every t ≥ ts.

Proof By the definition of ts, some node must have ✷RC-mcast m before t ≥ ts; so A(t) ≠ \{\}. Total number of ✷RC-mcasts by nodes of A(t) is:

\[\sum_k \alpha_k = \sum_k \alpha_k\]
Total number of $\diamond RC\_deliver()$ events generated by these multicasts is at least:

$$ (n - f) \times \sum_{i=1}^{a(t)} \alpha_{k_i} $$

(4)

Of the delivery events in (4), at most $f \times \sum_{i=1}^{a(t)} \alpha_{k_i}$ can occur in nodes that deliver $m$ but go on to crash without ever acting on any $m$ they delivered. Clearly, these nodes cannot be in $A(t)$ and cannot also be correct. So, the total number of $\diamond RC\_deliver()$ events that must occur at nodes of $A(t)$ and at correct nodes not in $A(t)$ is at least:

$$ (n - 2f) \times \sum_{i=1}^{a(t)} \alpha_{k_i} $$

(5)

There are two cases to consider regarding $\diamond RC\_deliver()$ events counted in (5):

(a) some occur at some correct $N_q \notin A(t)$, and
(b) none occurs at any correct $N_q \notin A(t)$.

Case (a): Recall that we consider the context where no node is decided and no node is executing $(r', ph')$ for $r' > r$ or $ph' > ph$ if $r' = r$. So, correct $N_q$ must be delivering $m$ of $(r, ph)$ for the first time, otherwise it would have expedited itself and be in $A(t)$; moreover, since $N_q \notin A(t)$, $N_q$ must $\diamond RC\_deliver()$ $m$ during $[t, t + \Delta_L]$. In lines 7-8 of Fig 4, it expedites itself to $(r, ph)$ and the new thread will start by $\diamond RC\_mcast$ $m$. So, the lemma holds for this case.

For case (b), each deliver event counted in (5) occurs only at nodes in $A(t)$ and hence increments $count_{k_i}$ of some $N_{k_i} \in A(t)$ (see line 9 in Fig 4). Whenever $N_{k_i} \diamond RC\_mcast m$ with $\alpha_{k_i} > 1$, $count_{k_i}$ is reduced by $(n - 2f)$ (lines 14-15). So, by $t + \Delta_L$,

$$ \sum_{i=1}^{a(t)} count_{k_i} \geq (n - 2f) \times \sum_{i=1}^{a(t)} \alpha_{k_i} - (n - 2f) \times \sum_{i=1}^{a(t)} \alpha_{k_i} = (n - 2f) \times a(t) $$

(6)

It is not possible for every $N_{k_i} \in A(t)$ to have $count_{k_i} < (n - 2f)$ by $t + \Delta_L$. Some $N_{k_i}$ must have $count_{k_i} \geq (n - 2f)$ and must $\diamond RC\_mcast m$ by $t + \Delta_L + \Delta$. □

Lemma 3: So long as no node delivers $(r, ph)$ message $m$ from a majority of nodes, there exists $t_2 > t_1$ such that at least $(n - f)$ nodes $\diamond RC\_mcast m$ at least once during $[t_1, t_2]$.

Proof Assume, contrary to the lemma, that $t_2$ does not exist; i.e., the number of nodes that $\diamond RC\_mcast m$ after $t_1$ never exceeds $(n - f - 1)$. By Lemma 2, a continuous stream of $\diamond RC\_mcasts$ of $m$ is launched, starting from $t_1$, with at least one $\diamond RC\_mcast$ for every $(\Delta_L + \Delta)$ time.

Once a node has its $count \geq (n - 2f)$, it must $\diamond RC\_mcast m$ unless it crashes. But after $t_1$, no node crashes. Therefore, the assumption made contrary to this lemma requires at most $n - (n - f - 1) = (f + 1)$ nodes do not execute more than $(n - 2f - 1)$ deliver events after $t_1$ so that their $count \leq (n - 2f - 1)$ at every $t \geq t_1$. This means that all except at most $(n - 2f - 1) \diamond RC\_mcasts$ are delivered only by $(n - f - 1)$ nodes. This is a contradiction of $\diamond RC$ protocol which must deliver $m$ to at least $(n - f)$ nodes. □

Lemma 4: At some time $t \geq t_2$, some node $\diamond RC\_delivers (r, ph)$ messages from at least $\lceil \frac{n + 1}{2} \rceil$ distinct nodes.

Proof Let $G_j(t)$ be the set of all nodes that have $\diamond RC\_mcast a (r, ph)$ message $m$ at least once in $[t_1, t]$. Since $t \geq t_2$, by lemma 3, $|G_j(t)| \geq (n - f)$. For each $N_{j_i} \in G_j(t)$, consider any one $\diamond RC$ multicast carried out by $N_{j_i}$ after $t_1$; let that multicast be denoted as $m_{j_i}$. Define $M_j(t) = \{m_{j_i}, \forall N_{j_i} \in G_j(t)\}$.

Case 1: All $m$ in $M_j(t)$ were delivered only by nodes in $G_j(t)$.

In this case, at least $|M_j(t)| \times (n - f) \diamond RC\_deliver$ events must occur among $|G_j(t)|$ nodes, with no $m$ in $M_j(t)$ delivered more than once to any $N_{j_i} \in G_j(t)$.

Since $|M_j(t)| = |G_j(t)|$, at least one $N_{j_i}$ must deliver at least $\frac{|G_j(t)|}{2}$ messages from $M_j(t)$. Otherwise, the total $\diamond RC\_deliver()$ events occurred cannot exceed $|G_j(t)| \times (\lceil \frac{n + 1}{2} \rceil)$ which is less than the minimum that must occur which is $|M_j(t)| \times (n - f)$, because $n - f > \lceil \frac{n + 1}{2} \rceil = \lceil \frac{3}{2} \rceil$ when $n > 2f$.

Case 2: Some $m$ in $M_j(t)$ are delivered by some node(s) not in $G_j(t)$, i.e. by some $N_q \in W(t_1) - G_j(t)$.

If $N_q$ has never $\diamond RC\_mcast m$, it will expedite itself to $(r, ph)$ after delivering some $m$ in $M_j(t)$ in lines 7 and 8. In that case, it enters $G_j(t')$ for some $t' > t$. Alternatively, $N_q$ does not $\diamond RC\_mcast$ after delivering some $m$ in $M_j(t)$. This means that $N_q$ has already $\diamond RC\_mcast m$, possibly more than once; these $\diamond RC\_mcasts$ by $N_q$ must be done before $t_1$, because $N_q \notin G_j(t)$. Each of $N_q$’s $\diamond RC$ multicast is delivered by at least $(n - f)$ nodes.

The minimum number of nodes in $G_j(t)$ which deliver $N_q$’s $\diamond RC$ multicasts is:

$$ n - 2f $$

(6)

If ever $N_q$ has $count_q \geq (n - 2f)$, it must $\diamond RC\_mcast m$ and join $G_j(t')$ for some $t' > t$. Let us consider two situations: (s1) no $N_q \in W(t_1) - G_j(t)$ joins $G_j(t')$, $t' > t$, and (s2) each $N_{j_i} \in G_j(t)$ delivers only at most $\lceil \frac{n}{2} \rceil$ messages from distinct nodes. We show that both s1 and s2 cannot prevail, by assuming to the contrary.

When s1 prevails, $count \leq (n - 2f - 2)$ for all $N_q \in W(t_1) - G_j(t)$. So no $N_q$ can $\diamond RC\_deliver$ more than $(n - f - 2)$ multicasts from $M_j(t)$, as it has already $\diamond RC\_delivered$ its own $m$ it $\diamond RC\_mcast$ before $t_1$. Messages of $M_j(t)$ generate a total of at least $|M_j(t)| \times (n - f) \diamond RC\_deliver()$ events; of these, the minimum number of deliver events which must occur within $G_j(t)$ will be:

$$ |M_j(t)| \times (n - f) - |W(t_1) - G_j(t)| \times (n - 2f - 2) $$

(7)

From (6), the deliver events which must occur with $G_j(t)$ due to $\diamond RC$ multicasts by $N_q \in W(t_1) - G_j(t)$ must at
least be:
\[(|W(t_1) - G_j(t_1)|) \times (n - 2f)\]

Adding (7) and (8), the minimum number of deliver events that must occur within \(G_j(t)\) is:
\[|M_j(t)| \times (n - f) + |W(t_1) - G_j(t_1)|\]

When \(s_2\) also prevails, the deliver events within \(G_j(t)\) cannot exceed \(|M_j(t)| \times (\frac{n}{2})\), which must at least be equal to (9); but it is not, since \(n - f > \frac{n}{2}\) when \(n > 2f\) and \(|W(t_1) - G_j(t_1)| \geq 0\). So, both \(s_1\) and \(s_2\) cannot prevail. Whenever \(s_1\) prevails, \(s_2\) cannot prevail; some \(N_{j_i} \in G_j(t)\) delivers messages from at least \([2^{n+1}]\) of distinct nodes and the lemma holds.

Alternatively, whenever \(s_2\) prevails, \(s_1\) cannot prevail and some \(N_{q} \in W(t_1) - G_j(t_1)\) joins \(G_j(t')\), leading to \(G_j(t') = W(t_1)\) for some \(t' > t\). At \(t'\), \(G_j(t')\) can then have only one case where the lemma is true. \(\square\)

**Lemma 5**: Let \(CAND^0\) and \(CAND^1\) be the set of all \(cand\) values of operative nodes at the start and at the end of some phase \(ph\), respectively. There is a non-zero probability \(P\) that \(CAND^1 < CAND^0\) given that \(CAND^0 > 1\).

**Proof** By contradiction. Let \(P\) be zero. This means that every \(v \in CAND^0\) is guaranteed to be the only value in \(VBag_{i}\) of some operative \(N_i\), which computes \(cand\) at the end of phase \(ph\).

Let \(w, w \neq v\) and \(v \in CAND^0\), be the only element in \(VBag_{\overline{i}}\) when \(N_{j_i}\) computes \(cand\) at the end of \(ph\). The latter is carried out only after \(N_{j_i}\) has delivered \(\{est, cand = w\}\) pair from a majority of nodes. So, a majority of nodes have \(\bowtie RC_{\overline{meas}}\{est, cand = w\}\). Any two majority sub-sets intersect. Therefore, any \(N_i\) must deliver \(\{est, cand = w\}\) at least once before it \(\bowtie RC\_delivers\) from a majority of nodes. It means that it is not possible for \(N_i\) to have only \(v, v \neq w\), in its \(VBag_{\overline{i}}\) before it computes \(cand\) at the end of phase \(ph\). This is a contradiction. \(\square\)

**Corollary**: The probability that \(|CAND^1| = 1\) given that \(|CAND^0| > 1\), cannot be zero.

**Proof** There can be \(2^{CAND^0}\) subsets in \(CAND^0\), of which \(|CAND^0|\) are singleton subsets. So, the probability that a subset of \(CAND^0\) is a singleton is given by \(\frac{CAND^1}{2^{CAND^0}}\), which is non-zero as \(1 \leq |CAND^0| \leq n\) and \(2^n\) is finite for finite \(n\). So, the probability that \(|CAND^1| = 1\) given that \(|CAND^0| > 1\), is given by \(P \times \frac{CAND^1}{2^{CAND^0}}\) which is non-zero.

**APPENDIX B**

**Lemma for ♦QC Protocol**

**Definition D(t)**: We define function \(D(t)\) as the set of all nodes that have delivered \(m\) at or before \(t\). By definition, \(D(t)\) is non-decreasing over time.

To recapp, executions of interest are those in which at least one correct node delivers \(m\). Let \(t_1\) be the earliest instance when a correct node delivers \(m\). Let \(t_2 = t_1 + (n^2 - 1)I_H\) when \(I_H\) is as defined in the Liveness Condition (2) of Sub-section 2.2. It is finite albeit unknown. Since \(n\) is also finite, \((n^2 - 1)I_H\) is a bounded time interval.

**Lemma 6**: In any execution of ♦QC in which the first correct node delivers \(m\) at time \(t_1\), there exists an instance \(t_Q, t_1 \leq t_Q \leq t_2 = t_1 + (n^2 - 1)I_H\), when all nodes in \(W(t_Q)\) have identical \(K(m)\)

**Proof** At any time \(t, t \geq t_1\), there can be three types of nodes.

**Category 1**: Correct nodes that delivered \(m\) at or before \(t\). That is, \(\{N_i : N_i \in D(t) \cap FW\}\). By given, there is at least one node in this category.

**Category 2**: Operative nodes that delivered \(m\) at or before \(t\) but do not survive until \(t_2\). That is, \(\{N_i : N_i \in D(t) - W(t_2)\}\). There can be zero or more nodes in this category.

**Category 3**: Nodes that are operative at time \(t\) but have not delivered \(m\) until \(t\): \(\{N_i : N_i \in W(t) - D(t)\}\). These nodes will not be executing the protocol at time \(t\). We will suppose that \(K(m)\) and \(KK(m)\) of these nodes contain only zeros and remain inaccessible until a node (if at all) starts executing the protocol for \(m\).

Let \(NZ_i\) denote the number of zeros in the \(K(m)\) of all nodes of \(W(t)\). A node that is in \(W(t)\) has at most \((n - 1)\) zeros in its \(K(m)\). There is at least one such node and at most \((n - 1)\) nodes may be in category 3. Thus, \(NZ_i < (n - 1) \times (n - 1) = n = (n - 1)(n + 1) = n^2 - 1\).

**Lemma** is true if \(NZ_i\) becomes zero at any time \(t\). For \(1 \leq j \leq (n^2 - 1)\), define \(G'_j\) as any set of nodes that are in \(W(t_1 + jI_H)\) and have identical \(K(m)\) at \(t_1 + jI_H\). \(W(t_1 + jI_H)\) cannot be empty as it contains Category 1 nodes. \(G'_j = W(t_1 + jI_H)\) implies that lemma is true for all \(t, t \geq t_1 + jI_H\). Suppose that \(G'_j \subseteq W(t_1 + jI_H)\) at \(t_1 + jI_H\). By liveness condition, \(G'_j\) will have \(N_{j_i}(B,H)\)-Connecting with some \(N_{j_i} \in W(t_1 + jI_H)\) at \(t_1 + jI_H\). Two cases to consider:

**Case 1**: \(N_{j_i}\) has not delivered \(m\) at \(t_1 + (j - 1)I_H\). That is, \(N_{j_i} \notin D(t_1 + (j - 1)I_H)\). \(NZ_i\) will reduce by at least 2.

**Case 2**: \(N_{j_i} \in D(t_1 + (j - 1)I_H)\). \(NZ_i\) is never set to 0. So, \(NZ_i\) must be zero, if \(G_i = W(t_1 + jI_H)\) does not come true before \(t_2\). So, there exists \(t_Q, t_1 \leq t_Q \leq t_2\) when all nodes of \(W(t_Q)\) have identical \(K(m)\). \(\square\)

**APPENDIX C**

**Lemmas for ♦RC Protocol**

**Lemma 7**: In an execution of ♦RC in which a correct node delivers \(m\), there exists a node that realises \(m\) and remains operative for at least \(fI_H\) duration thereafter.
Proof So long as no operative node has realised \( m \), the executions of both \( QC \) and \( RC \) are identical after \( t_1 \) which, as in Appendix B, denotes the earliest instance when a correct node delivers \( m \). \( NZt \), the number of zeros in the \( K(m) \) vectors of nodes of \( W(t) \), will be irreversibly reducing by at least 1 for every \( IH \) interval, starting from \( t_1 \). So, some operative node must have the number of zeros in its \( K(m) \) reaching \( f \) and must realize \( m \). If that node survives for at least \( fIH \), then the lemma becomes true. Otherwise, the execution of \( RC \) reverts to being identical to \( QC \) and \( NZt \) starts to fall, allowing another node to realize \( m \). The number of nodes realizing \( m \) and crashing within \( fIH \) is bounded by \( f \). Moreover, there is at least one correct node executes \( RC \). Hence, the lemma is true.

\( \Box \) Lemma 7

Definition: \( Z(t) \) is a function that returns the set of nodes that have realized \( m \) at or before \( t \).

Lemma 8: In any execution in which at least one correct node delivers \( m \), at least one correct node realizes \( m \) after delivering \( m \).

Proof By lemma 7, there is an operative node, say, \( N_r \) that realizes \( m \) at or before \( t_r \) and remains operative at least until \( t_r + fIH \). \( \forall t: t_r \leq t \leq t_r + fIH \), \( N_r \in Z(t) \cap W(t) \). If \( N_r \) is correct, then lemma is true. So, we will assume that it need not be so. Let \( G'_j \) be \( Z(t_r + (j-1)IH) \cap W(t_r + jIH) \) for \( 1 \leq j \leq f \). \( G'_j \) cannot be empty due to \( N_r \). If \( G'_j = W(t_r + jIH) \), then lemma is true.

Say \( G'_j \subset W(t_r + jIH) \). During \( [t_r + (j-1)IH, t_r + jIH] \), some node \( N_j \in G'_j \) will \( (B,h)-Connect \) with \( N_j \in W(t_r + jIH) \). This will result in \( N_j \) realizing \( m \), making \( |Z(t_r + jIH)| \geq |Z(t_r + (j-1)IH)| + 1 \). Thus, if \( G'_j \subset W(t_r + jIH) \) for all \( j, 1 \leq j \leq f \), then \( Z(t_r + jIH) \geq f + 1 \) and at least 1 of these realized nodes is correct. \( \Box \) Lemma 8

Lemma 9: In an execution of \( RC \) protocol in which at least one correct node delivers \( m \), there is a timing instance \( t_R \) such that \( Z(t_R) \cap W(t_R) = W(t_R) \).

Proof Let \( t_{1R} \) be the earliest instance when a correct node realizes \( m \). As per lemma 8, \( t_{1R} \) exists in any execution of interest.

Let \( t_2 = t_{1R} + (n-1)IH \). Define \( G'_j = Z(t_{1R} + (j - 1)IH) \cap W(t_{1R} + jIH) \), if \( G'_j = W(t_{1R} + jIH) \) the lemma becomes true at \( t_{1R} + jIH \). Say \( G'_j \subset W(t_{1R} + jIH) \). By (2), some node \( N_i \in G'_j \) will \( (B,h)-Connect \) with some \( N_j \in W(t_{1R} + jIH) \). During \( [t_{1R} + (j-1)IH, t_{1R} + jIH] \), \( N_j \) must realize this connectivity by Claim 1. \( |Z(t_{1R} + jIH)| \geq |Z(t_{1R} + (j - 1)IH)| + 1 \) for every \( j \), if \( G'_j \subset W(t_{1R} + jIH) \). So, \( Z(t_{1R} + (n-1)IH) = n \) if \( G'_j \neq W(t) \) is not true for some \( t < t_{1R} + (n-1)IH \). Hence the lemma. \( \Box \) Lemma 9