Design and Performance-Study of Crash-Tolerant Protocols for Broadcasting and Supporting Consensus in MANETs

Einar W Vollset and Paul D Ezhilchelvan

School of Computing Science

University of Newcastle

{einar.vollset, paul.ezhilchelvan}@ncl.ac.uk [*]

Abstract

Self-organized collaborative applications in terrains with no infrastructure support for untethered communication are long known to be feasible only with the mobile ad-hoc networking (MANET) technology. Supporting collaboration however requires a solution to the consensus problem, using which collaborating users with different initial opinions can decide identically. Efficient consensus solutions require efficient broadcast support. This paper presents four crash-tolerant broadcast protocols which are designed (i) to provide the maximum broadcast coverage that can ever be guaranteed, and (ii) to suit a wide range of MANET types: from a connected MANET (no partitions) to intermittently disconnected one (partitions occurring rarely and healing swiftly) to an intermittently connected one (partitions taking longer to heal and re-appearing swiftly). The resulting design challenges are addressed systematically, beginning with formulating a MANET liveness property and deriving two foundational results that would guide the protocol design. The protocols' performance is then studied through simulations for a range of node speeds and network densities. The one with the least overhead among them is used to host a known, randomized consensus protocol as a broadcast application. The consensus overhead and the latency are found to be surprisingly small even when each node has distinct initial opinion. The underlying reason is attributed to the specific characteristics of MANETs and the features of the broadcast protocol.

keywords. Crash-tolerance, Ad-hoc networking, Partitions and Connectedness, Broadcasting, Consensus.

[*]Tel: +44-191-222-8546, Fax: +44-191-222-8232

1 Introduction

Ad-hoc Networking is perhaps the only technology available for a group of mobile wireless users to engage on a collaborative task in terrains which can offer no fixed infrastructure support for untethered communication. Supporting collaboration in such mobile ad-hoc networks (MANETs) however is not an easy task, and one of the difficult problems to enable the mobile and wireless users, called from now on the *nodes* for short, to agree on the same course of action even if the action-plans thought of initially by them can be different.

For example, a new node may request to join the collaboration process, and the collaborating nodes may have different opinions as to whether or not the join request be accepted, and if accepted, what should be the *status* accorded to the new node within the group. The status may be a rank order, an IP address (chosen to be unique [16]), wireless channels allocated exclusively to the joiner for transmission (to avoid channel interference [17]), and so forth. Despite any differences in their opinions over the join request, the nodes must decide identically.

It is well-known [18] that such decision problems encountered in distributed computing can be solved as variations of the generic problem known as the *consensus* problem [9]. So, a consensus module that is evaluated to be efficient in diverse MANET environments (e.g., from sparse to dense), is a vital tool for supporting a variety of distributed applications. The aim of our work is to build this tool for hosting distributed MANET applications. More specifically, the contributions are two fold: we will (**i**) design a family of broadcast protocols which are appropriate to a diverse range of MANET characteristics, and study their performance; and, (**ii**) use the most efficient of the family to support a consensus protocol and demonstrate the performance of the latter.

Consensus problem has been extensively studied under the *asynchronous* communication model wherein the message transfer delay between any pair of operative nodes at any given instance is finite but cannot be bounded with certainty. A MANET, with its arbitrary topological changes due mainly to application-driven node mobilities, conforms to this model, *provided* that any partition that disconnects operative nodes is not permanent. We assume that a group of collaborating nodes can be partitioned and that partitions heal eventually, i.e., after some arbitrary amount of time. Therefore, an attempt to transfer a message between two operative nodes can take an arbitrary amount of time to succeed, if the nodes were initially not connected (either directly or transitively).

The assumption that the partitions heal eventually, is realistic since collaboration has a sense of purpose which typically requires that the users strive to be in touch with one another; further, our system model (in Section 2) admits node crashes and treats them as events that cannot be accurately detected [10]; so, if there is a user who wanders astray and becomes permanently disconnected from others, his node can be regarded to have crashed.

(The partition-centric approach [7, 6, 16] does not force permanent partitions to be considered in terms of node crashes, but is not pursued here for reasons stated in Section 6.)

When MANETs are modeled as asynchronous communication networks, the vast amount of literature on fixed-network consensus protocols ([12] presents a consensus 'tour') can offer valuable guidance to achieving our aim. We first note that the randomized protocols are more decentralized than their failure-detector (e.g., $\Diamond w$ [3]) based counterparts that take a (rotating) coordinator based approach. Since decentralization is appropriate for MANET environments, we would prefer randomized protocols, bearing in mind that a decentralized (or symmetric) consensus protocol requires each node to broadcast to every other node and the underlying broadcast protocol needs to be bandwidth-efficient. We also note that at least a majority of nodes must be *consulted* before a consensus protocol can decide; so, the overhead and latency for consensus will be small if the underlying broadcast protocol can achieve high coverage with small latencies. All these observations indicate that efficient consensus requires efficient broadcast support.

To provide an efficient broadcast support, we design and performance-study four protocols, and select the best for studying the performance of a randomized consensus protocol [8]. The design will make use of the fact that the nodes can be uniquely ranked using consensus; this allows the broadcast protocols to represent message dissemination status as a boolean vector and to offer maximum coverage with small bandwidth overhead.

A novelty of our work is that our broadcast protocols are designed to suit a wide range of MANET types: from a *connected* MANET (no partitions) to an *intermittently disconnected* one (partitions occurring rarely and healing swiftly) to *intermittently connected* ones (partitions taking longer to heal and re-appearing swiftly). Specifically, the protocols will be primarily designed for the last type, and then adopted to the two former types by incorporating simple techniques (e.g., ack suppression) commonly used in networking protocols for improving efficiency. So, our design approach will be different from the traditional approach (e.g., [11]) of maintaining a routing structure for message dissemination, and the reasons for this essential difference are succinctly presented below.

Suppose that the route between (source) node s and (destination) node d is found to be via an intermediary node i; that is, the route is made up of two *contemporaneous* wireless links or direct-connections namely: between s and i, and between i and d. When the MANET is only intermittently connected, these links may not exist simultaneously and a multi-hop route connecting s and d may not be formed or may not be identified if formed only for a brief period. Therefore, the protocol design should not rely on the possible existence or identification of multi-hop routes between s and d, but rather make effective use of 1-hop, *direct-connections* which various node pairs experience at different timing instances.

Suppose, for example, that when node i enters the radio range of d, it had already lost the connection it had with s; that is, the link between s and i and that between i and d come into existence not simultaneously but one after the other. Consequently, message dissemination must involve node i *retaining* a message m it received from s for a while, and *transmitting* it at appropriate moments so that d could receive m if the link between i and d is formed within the retention period. This approach has been used in [4, 14, 15] when MANETs are expected to be intermittently connected. We will take this approach but our design will differ in two important aspects: (a) how a node decides when to stop retaining a given m, and (b) guaranteeing the maximum attainable coverage for m.

The design challenges that ensue are addressed in a systematic manner: we first derive some foundational results that will guide the design process and also influence the guarantees offered by the protocols. For example, we observe that a protocol that is designed to tolerate at most f node crashes, cannot guarantee that all operative nodes receive a broadcast m when less than f nodes have actually crashed, unless nodes retain m for an unbounded amount of time (for possible re-transmission). Since nodes are wireless devices with natural constraints on memory and battery usage, the retention period cannot be unbounded. Thus, it is possible that an operative node receives a broadcast m or decides in a consensus run, with another operative node being totally unaware of that broadcast or the consensus run. This is different to the fixed-network broadcast and consensus protocols which ensure identical outcome for *all* operative nodes.

The paper is organized as follows: Section 2 presents the system model, assumptions, and the network liveness property. Section 3 describes the rationale for our design approach and identifies two results that influence the protocol design; the properties of broadcast and consensus protocols for MANETs are also formulated. Section 4 is devoted to the description of broadcast and consensus protocols. Simulation results are presented in Section 5; the consensus performance is surprisingly faster and the reason seems to be due to the features of MANETs and the broadcast protocol. Section 6 concludes the paper, with an examination of the literature.

2 System Model

We consider a group $\mathcal{G}$ of mobile nodes collaborating towards a common goal in a terrain that has no fixed infrastructure for supporting communication between nodes. The nodes can however communicate using the omnidirectional wireless transmission functionality of a CSMA/CA-like MAC layer protocol (e.g. IEEE 802.11b). Thus, the information exchange is limited strictly to ad-hoc networking.

A new node can join $\mathcal{G}$ and a collaborating node can leave $\mathcal{G}$ only after the nodes of $\mathcal{G}$ have approved the join/departure requests. Thus, the number, n, of nodes involved in collaboration at any given time can vary. A node can crash (i.e., cease to be operative) at any moment. When a collaborating node crashes, it effectively makes an unapproved departure from $\mathcal{G}$ and its absence is not assumed to be detectable with certainty [10]. The number of nodes that can crash while engaged in collaboration does not exceed a known bound $f > 0$. That is, $\mathcal{G}$ contains at least $n - f$ operative nodes at any time and we assume $n \gg f$.

Direct Connectivity. Consider two operative nodes that are in wireless range of each other. A *congestion-* and *collision-resilient* (CCR) channel is said to exist between them, if at least one of a few consecutive attempts made by each node to send a packet to the other, is successful. (These attempts are typically made at the MAC layer.) Let δ be the maximum delay which a packet can experience to be received over a CCR channel.

Two operative nodes are said to be *directly connected* at any given moment, if a CCR channel exists between them for B or more time starting from that moment, where $B \gg \delta$ is an application-specified parameter. The intuition here is that two nodes being in each other's wireless range can be of any use to an application, only if that gives raise to a CCR channel that lasts for at least B time. (The applications that are of interest to us will be broadcast protocols.)

2.1 MANET Liveness Property

We assume that the ad-hoc network formed by the operative nodes of $\mathcal{G}$ satisfies a liveness property that does not allow any partition to become permanent. For the sake of exposition, we will assume that there are no requests for joins/departures. Let $\mathcal{O}$ be the set of all nodes of $\mathcal{G}$ that are operative at time t. Let $\mathcal{P}$ be any non-empty and proper sub-set of $\mathcal{O}$, and $\overline{\mathcal{P}}$ be its complementary set in $\mathcal{O}$; that is, $\overline{\mathcal{P}}$ contains those nodes that are operative at t but not in $\mathcal{P}$. Since we will be concerned about operative nodes of $\mathcal{P}$ and $\overline{\mathcal{P}}$ being possibly disconnected and eventually re-connected, let us assume that $\mathcal{P}$ and $\overline{\mathcal{P}}$ each have some node(s) that never crash.

If no node in $\mathcal{P}$ ever has direct connectivity with any node in $\overline{\mathcal{P}}$, then $\mathcal{P}$ and $\overline{\mathcal{P}}$ are said to be permanently partitioned (from the perspective of application that has specified B). The liveness property disallows it by requiring that direct connectivity must emerge between some nodes of $\mathcal{P}$ and $\overline{\mathcal{P}}$ within some arbitrary amount of time (I) after t. More precisely, at least one node in $\mathcal{P}$ must directly connect with some node(s) in $\overline{\mathcal{P}}$ at least once during $[t, t + I]$, where $I \geq B$ is finite but unknown.

Remark 1. By letting I be unknown, little is assumed to be known about network density, node mobility patterns and node speeds. (Network density is the number of nodes within a disc of radius equal to nodes' radio range.) For example, when the network density is small or when nodes move at very high speeds relative to each other, a CCR channel lasting continuously for at least B time, will take longer to emerge, i.e., I tends to be large. If the density is high and nodes move at low or medium speeds, direct connectivity between nodes of $\mathcal{P}$ and $\overline{\mathcal{P}}$ is likely to emerge quickly, if it does not exist already; i.e., I tends to be small.

Remark 2. $I = \infty$ and $I = B$ represent extreme cases of interest. The former implies that direct connectivity between nodes of $\mathcal{P}$ and $\overline{\mathcal{P}}$ may take for ever to emerge. $I = B$ means that new direct connectivity between nodes of $\mathcal{P}$ and $\overline{\mathcal{P}}$ emerges at t, or existing direct connectivity prolongs beyond t for a further B time or more, or both.

Network Liveness Property rules out permanent partitioning of *any* $\mathcal{P}$ defined at *any* instance t during the collaboration process that is assumed to be initiated at t_0. It is stated formally as:

$\forall \mathcal{P}$, $\forall t \ \geq t_0$, $\exists I$, $B \leq I \ \neq \ \infty$: $\exists\, i \in \mathcal{P}$, $j \in \overline{\mathcal{P}}$: nodes i and j have direct connectivity during $[t, t + I]$.

3 Design Approach and Protocol Specifications

A MANET remains a connected network throughout the collaboration, if the liveness property is satisfied for $I = B$: for any given $\mathcal{P}$, some operative node in $\mathcal{P}$ is beginning or continuing to have direct connectivity with some operative node in $\overline{\mathcal{P}}$ at every $t \geq t_0$ (see also Remark 2 above). That is, some nodes of $\mathcal{P}$ and $\overline{\mathcal{P}}$ are in direct connectivity at any given moment, and this holds despite node mobility.

As I becomes larger (compared to B), the MANET becomes intermittently disconnected and then intermittently connected (see also Remark 1 above). Since I is unknown, broadcast protocols need to be designed to account for the possibility that $I \gg B$. As observed in Section 1, a multi-hop route between a node pair requires the simultaneous existence of its constituent 1-hop links, which is less likely when $I \gg B$. Therefore, in conformance with the earlier works in intermittently connected MANETs [4, 14, 15], our protocols will require that a node which has received m retain m *for a while* and transmit it at *appropriate* moments so that m gets disseminated. Two design issues that arise thereof are: when a node that has received m, should (i) transmit m and (ii) stop retaining/transmitting m. These issues are addressed together with the goal of attaining the maximum possible *coverage* for m, where coverage (denoted as c) refers to the number of operative nodes (other than the broadcaster) that receive m at least once.

The three core protocols designed here address the issue (i) by combining the timer and the event driven approaches in varying degrees, where an event can be receiving a control packet or deducing the presence of another node in the neighborhood for the first time since m was received. To address (ii), the protocols are designed to have the *storage subsidence property* (SSP) defined below.

Let m be broadcast at time t_b. A broadcast protocol satisfies the ***storage subsidence property*** (SSP) only if there is a finite time t_e ($t_e > t_b$) after which nodes that received m are not required to retain m for propagation. That is, all (broadcast-related) transmissions of m end by t_e.

3.1 Foundational Results

The requirement of the SSP distinguishes our protocols from the fixed-network, asynchronous protocols which do not have it explicitly imposed on them as a design objective. This, together with the I of the liveness property being unknown, gives rise to two important results. The first helps identify the basic dissemination strategy that ought to be employed when nodes have no access to any neighborhood information, and the second the maximum guaranteeable coverage when nodes can crash and crashes are not accurately detectable.

Let us first note that the well-known flooding scheme has the SSP: the broadcaster transmits m once, soon after m is ready for transmission; any other node that receives m, transmits m after a random delay; soon after performing the single, mandatory transmission, nodes can discard m. If the network is sparse (i.e., $I \gg B$), transmissions of m are less likely to be received by nodes that have not yet received m; so, the flooding scheme can provide poor coverage, and even $c = 0$ if no node receives m. This means that when $I \gg B$ is likely, the nodes may have to transmit m *more than once* to achieve high coverage, and this inferrence is generalized as:

Proposition 1. *Suppose that nodes never crash (each node is operative) and have no knowledge about immediate neighborhood. A broadcast protocol that has the SSP cannot guarantee $c \geq 1$, unless every node with m transmits m at least once every τ time, $\tau < (B + \delta)$, until it decides not to retain m.*

The proof can be seen in the Appendix. The first proposition thus identifies τ-periodic transmissions as essential to achieve higher c. The second proposition establishes the upper bound that can be guaranteed on c to be $(n-f-1)$ when crashes of at most f nodes are assumed. (Detailed correctness arguments are given in the Appendix.) This means that a broadcast protocol terminating its propagation efforts once $(n - f - 1)$ nodes are known to have received m, is a justified design option if the protocol is to have the SSP.

Proposition 2. *Any crash-tolerant broadcast protocol that has the SSP, cannot guarantee that more than $(n - f - 1)$ nodes receive a given broadcast, even if more than $(n - f)$ nodes, including the broadcaster, do not crash.*

3.2 Broadcast Protocol Specification

For a broadcast m initiated at time $t_b \geq t_0$, the following guarantees are offered despite at most f, $0 < f \ll n$, nodes crashing before the broadcast completes:

1. **Delivery**: at least $(n - f - 1)$ nodes receive m within some bounded time after t_b, if the broadcaster does not crash (i.e., remains operative), or if the broadcaster crashes and an operative node receives m; and,
2. **Termination**: if a node that receives m remains operative, it discards m (**Storage Subsidence**) and stops transmitting any packet concerning the broadcast of m (**Bandwidth Subsidence**) at some time after t_b.

A broadcast protocol ensures that at least $(n-f-1)$ nodes receive an operative node's m, and this lower bound reflects the maximum guaranteed coverage identified in proposition 2. It is possible that the broadcaster crashes before completing the protocol and a few nodes, if any, that receive m also crash likewise. In that case, no delivery guarantees can be given; however, if an operative node receives m, then at least $n - f - 1$ nodes receive m.

3.3 Consensus Protocol Specification

A consensus protocol enables nodes to reach a common decision. It guarantees the following when (1) nodes of $\mathcal{G}$ can make potentially different initial proposals or *values*, (2) at most f nodes can (undetectably) crash before or during the protocol execution, and (3) $n > 2f$:

Termination. With probability 1, at least $n - f$ nodes of $\mathcal{G}$ irreversibly decide on a value.

Validity If a node decides on v, then v is proposed initially by some node.

Agreement No two nodes that decide, decide differently.

In wired networks, when the SSP-like requirements are not rigorously enforced, an operative node's message is typically ensured to reach every other operative one through selective transmissions followed up by acknowledgements. So, traditional consensus protocols guarantee that *all* operative nodes decide. In a crash-prone MANET, as pointed out in proposition 2, a broadcast can be guaranteed to reach only $n - f$ nodes if subsidence properties have also to be upheld with I being unknown. Hence, the termination guarantee is weaker for MANETs. This has two implications.

First, at most f of the decided nodes could crash if no node has crashed before the consensus execution started. So, in the worst case, only $n - 2f$ operative nodes have the consensus outcome. Since $n > 2f$, $n - 2f \geq 1$.

Second, suppose that node i decided during a consensus run and that it now intends to leave $\mathcal{G}$; since, it may currently be the *only* operative node to have the consensus outcome, it must broadcast the outcome before it departs $\mathcal{G}$ so that there are some operative nodes in $\mathcal{G}$ that know the outcome. However, the following (worst-case) scenario is possible: no node in $\mathcal{G}$ has crashed when i intends to leave, only $n - f$ nodes (including i) decided during the consensus run, only the nodes that decided receive i's broadcast, i leaves $\mathcal{G}$ and then f of the remaining $(n - f - 1)$ decided nodes crash. If $n = 2f + 1$ when i left $\mathcal{G}$, then there will be no operative node in $\mathcal{G}$ which knows the consensus outcome despite i's broadcast. Therefore, node i should (be allowed to) leave $\mathcal{G}$ only if $n > 2f + 1$.

4 A Family of Broadcast Protocols

We first present three core protocols: *proactive dissemination* protocol (*PDP*), *reactive dissemination* protocol (*RDP*), and a hybrid version called the *proactive knowledge and reactive message* (*PKRM*) protocol. The *RDP* assumes that the nodes know their immediate neighbors. Of the three, the PKRM appears to possess the best features of the other two and avoid the worse aspects of each. (This is also confirmed by simulations.) Hence, it is optimized and the resulting protocol is termed as the *optimized PKRM* and denoted also as PKRM$_o$.

For the sake of exposition, we will assume that $\mathcal{G}$ is made up of n nodes, each with a unique sequence number in $[0 \ldots (n-1)]$. (See also the discussions in Section 6.) Each node i knows n and its sequence number i; it maintains a boolean vector $K_i(m)$ of n bits for m. $K_i(m)[j]$ = 1 means that node i knows (for sure) that node j has m, $K_i(m)[j]$ = 0 if node i does not know if node j has m or not. Thus, $K_i(m)$, more precisely, the 1-bits in it, indicate the *knowledge* of node i on the propagation of m. Note that $K_i(m)[j]$ = 1 indicates certainty of node i concerning node j and m. Since a node cannot 'undo' receiving m, this certainty remains valid for ever.

Realization: When $K_i(m)$ contains $(n - f)$ or more 1 bits, node i *realizes* that m needs to be propagated no longer, or node i is simply said to have *realized* m. A realized m can be discarded. For each protocol, an operative node that has m, realizes m at some time after m is broadcast at t_b.

4.1 Proactive Dissemination Protocol (PDP)

In the PDP, nodes that have m transmit m once every β seconds. (The broadcaster of m has m at the time of broadcast, t_b.) β is a fixed parameter and $2(\beta + \delta) \leq B$. This ensures that when two operative nodes experience direct connectivity, they can, within B seconds, exchange information and also each other's response to the information exchanged. (Recall that $B \gg \delta$.) The protocol described below has six steps and the correctness arguments are presented in the Appendix.

Step 1. The broadcaster initializes $K(m)$ as a vector of zeros and then sets its own bit to 1; it transmits m with its $K(m)$ as a message field $m.K$ and with a unique $m.id$.

Step 2. When node i receives m for the first time, it initializes $K(m)$ to the received $m.K$ and sets its own bit in $K(m)$ to 1. After waiting for a random time interval distributed uniformly in $(0, \beta)$, it transmits m with $m.K$ being a copy of its $K(m)$.

Step 3. A node that transmitted m once, will thereafter check once every β seconds whether a transmission is needed for the propagation and realization of m:

if the node has not realized m: it transmits m (with $m.K$ set to its $K(m)$);

if the node has realized m: if it has received m in the past β seconds, it transmits an *infectious* packet ***realize***(m) that contains only $m.id$; otherwise, it does nothing.

Step 4. When a node that has unrealized m receives m, it updates its $K(m)$ as per the contents of the received $m.K$: if $K(m)[j] = 0$ and $m.K[j] = 1$, then $K(m)[j]$ is set to 1. If the node has $(n - f)$ or more 1-bits in its $K(m)$ or receives ***realize***(m), it realizes m (i.e., gets infected).

Step 5. When ***realize***(m) is received after m is realized, the received packet is ignored.

Step 6. When a node that has not received m even once, receives ***realize***(m), it ignores the received packet.

4.2 Reactive Dissemination Protocol (RDP)

Each node i has information ($Neigh_i$) on immediate neighborhood, expressed in terms of nodes' sequence numbers. Let $\{K_i(m)\}$ denote the set of nodes whose bits are 1 in $K_i(m)$. Node i propagates m if it has m and only if $(Neigh_i - \{K_i(m)\})$ is not empty, which is evaluated once every β seconds.

When nodes that have m, thus transmit m only on the *need to propagate* basis, it is possible that $(n - f)$ or more nodes have received m but nodes with m cannot realize m. Consider, for example, a MANET of 3 nodes (n= 3) arranged in a straight line, with each node having only its immediate neighbor(s) in its $Neigh$. When the middle node broadcasts m, each of its two neighbors (the end nodes) receives m and forms $K(m)$ with two 1 bits (see step 2 of the PDP). Since each end node has only the broadcaster in its $Neigh$, it will find $Neigh - \{K(m)\} = \{\}$ and choose not to transmit m. If $f = 1$, the broadcaster, which cannot know (for sure) whether its neighbors have received the broadcast, cannot realize m even though all three have received m.

It is thus obvious that RDP requires additional data structures and control packets than PDP. Nodes maintain two more (boolean) vectors: knowledge on the propagation knowledge of m ($KK(m)$) and knowledge on the realization of m ($KR(m)$). If node i knows that node j has the same $K(m)$ as itself, then $KK_i(m)[j]$ is set to 1; otherwise, $KK_i(m)[j]$ will retain the initialized value of 0. Similarly, if node i knows that node j knows of the realization of m, then $KR_i(m)[j]$ is set to 1; otherwise, $KR_i(m)[j]$ will retain the initialized value of 0.

Note that, unlike in $K_i(m)$ and $KR_i(m)$, the number of 1s in $KK_i(m)$ can decrease, because $KK_i(m)$ will be re-set every time $K_i(m)$ changes. Similarly, while $KK_i(m)[j] = 1$, node j may have added more 1s to its $K(m)$ without node i being aware of this addition. Therefore, the only certainty that node i can derive from $KK_i(m)[j] = 1$ is that node j had the same $K(m)$ as itself at some (past) time in an execution.

Packets of the following types are also used: *K_pkt*(m) contains $m.id$ and the transmitting node's knowledge $K(m)$ and $KK(m)$; *realize(m)* contains $m.id$ and the transmitting node's $KR(m)$; and *realize_ack*(m) is transmitted in response to receiving *realize(m)*. Finally, β is fixed to be (as in the PDP) $2(\beta + \delta) \leq B$. The protocol steps are:

Step 1. The broadcaster initializes $K(m)$, $KK(m)$ and $KR(m)$ as a vector of zeros and then sets its own bit in the former two vectors to 1; it transmits m with its $K(m)$ as a message field $m.K$ and with a unique $m.id$.

Step 2. When node i receives m for the first time, it initializes $K(m)$ to the received $m.K$ and sets its own bit to 1; it initializes $KK(m)$ and $KR(m)$ as a vector of zeros; it sets its own bit in $KK(m)$ to 1. After waiting for a random time interval distributed uniformly in $(0, \beta)$, it transmits m with $m.K$ being a copy of its $K(m)$.

Step 3. A node that transmitted m once, will thereafter check once every β seconds whether a transmission is needed, unless $\mid \{KR(m)\} \mid= n$:

node has not realized m: If $Neigh - \{K(m)\} \neq \{\}$ then m (with $m.K$ set to its $K(m)$) is transmitted; if $Neigh - \{K(m)\} = \{\}$ and $Neigh - \{KK(m)\} \neq \{\}$, *K_pkt*($m.id$) is transmitted. Nothing is transmitted if $Neigh - \{K(m)\} = \{\}$ and $Neigh - \{KK(m)\} = \{\}$.

node has realized m: If $Neigh - \{KR(m)\} \neq \{\}$ then a $realize(m)$ (containing its $KR(m)$) is transmitted;

Step 4. When a node that has an unrealized m receives m or *K_pkt*(m), it updates its $K(m)$ as per the contents of the received and re-sets its $KK(m)$ appropriately; If $\mid \{K(m)\} \mid \geq (n - f)$ or if it receives a *realize(m)* or a *realize_ack*(m), it realizes m and updates its $KR(m)$ appropriately.

Step 5. A node that realized m transmits *realize*(m) whenever it receives m or *K_pkt*(m); it transmits *real-*

ize_ack(m) whenever it receives *realize*(m).

Step 6. When a node that has not received m even once, receives a *realize(*m*)*, it transmits *realize_ack*(m) immediately and thereafter whenever it receives m, *K_pkt*(m) or *realize(*m*)*. (It executes no other protocol step.)

4.3 PKRM (Proactive Knowledge and Reactive Message) Protocol

This protocol combines the features of PDP and RDP, with no $Neigh$ and fewer data structures and control packets (as in PDP) and fewer transmissions of m (as in RDP). In PKRM, unrealized nodes transmit $K(m)$ (in a *K_pkt*), not m as in PDP, once every β time. When node that does not have m, receives $K(m)$, it is prompted to ransmit *req*(m) packet and thereby request m to be transmitted.

When a node that transmitted $K(m)$ receives a *req*(m), it has effectively evaluated the predicate $Neigh - \{K(m)\} \neq \{\}$ of the RDP to be true and transmits m. Thus, only on a *need to propagate* basis, m is transmitted. PKRM uses the pro-active transmissions of $K(m)$ by unrealized nodes to inform (infect) the nodes of realization if they have a realized neighbor around. This means that the additional knowledge vectors of the RDP are redundant. Finally, as in the other two protocols, β is fixed to be $2(\beta + \delta) \leq B$. The protocols steps are as follows.

Steps 1 and 2. As in Steps 1 and 2 of the PDP.

Step 3. A node that transmitted m once, thereafter checks once every β seconds:

3(a) the node has not realized m: If it has received *req*(m) in the past β seconds, it transmits m (with $m.K$ set to its $K(m)$), else it transmits *K_pkt(*m*)* (with no $KK(m)$).

3(b) the node has realized m: If it has received m, *K_pkt(*m*)* or *req*(m) in the past β seconds, it transmits *realize*(m) that contains only $m.id$; otherwise, it does nothing.

Step 4. When a node that has unrealized m, receives m or *K_pkt(*m*)*, it updates its $K(m)$ as per the contents of the received $m.K$. If the node has $(n - f)$ or more 1-bits in its $K(m)$ or received *realize(*m*)*, it realizes m.

Step 5. When a *realize(*m*)* is received after the realization of m, the received packet is ignored.

Step 6. When a node that has not received m even once, receives a

K_pkt(m*)*: it transmits *req*(m) after a random time interval distributed uniformly in $(0, \beta - 2\delta)$, or

realize(m*)*: it ignores the received packet.

4.4 Optimized PKRM Protocol

The PKRM protocol is optimized towards execution efficiency, minimizing collisions and bandwidth reduction.

Event driven execution. After m is realized, a thread carries out the instructions of step 3 and step 5 in response to receiving a PKRM related packet. Consequently, no periodic inspection of the messages is necessary and the thread goes dormant once the transmissions of PKRM related packets for m end.

Staggering the proactive disseminations. When m remains unrealized, the protocol checks the messages received in the recent past, not every β seconds (as in step 3), but after every $\widehat{\beta}$ seconds, where $\widehat{\beta}$ is an independent random duration distributed uniformly in (0, β).

Suppressing the proactive disseminations. At the end of the randomly chosen timeout in steps 2 and 3(a), if a transmission of m is due, only *K_pkt(m)* is transmitted if two copies of m were received during the timeout; if a transmission of *K_pkt(m)* is due, it is suppressed if two $m.K$ with more or the same knowledge as local $K(m)$ were received during the timeout. This is done on the basis that the neighborhood is dense, and the planned transmission would offer little additional information over what has been recently seen to have been disseminated.

4.5 A Consensus Protocol

We adopt the (fixed-network) protocol of [8] for the wireless context, run it as a broadcast 'application' using the optimized PKRM and study its performance. We here provide a brief sketch on the workings of the protocol of [8].

The protocol operates in *asynchronous rounds* with each round $r \geq 0$ having two phases. Any node can propose its initial 'value' by broadcasting it and thereby *initiate* a consensus execution. When a node that has not yet proposed any, receives another node's initial proposal, it can either accept the latter as its own or choose its own value, and then *participate* in the consensus run. Thus, each node i has some *initial* value (denoted as $V_i(0,1)$) to broadcast in round 0 phase 1.

In phase 1, node i broadcasts its value for round r ($V_i(r,1)$) and waits to receive ($\lceil (n+1)/2 \rceil$) values of $V(r,1)$ from distinct nodes including itself; if the received values are identical, it adapts that value as its value for phase_2 ($V_i(r,2)$); else, it sets $V_i(r,2)$ to a special value $\perp$ which no node will have as its initial value $V(0,1)$. Since ($\lceil (n+1)/2 \rceil$) is a majority in n, if nodes i and j choose non-$\perp$ value then $V_i(r,2)$ = $V_j(r,2)$.

In phase 2, node i broadcasts $V_i(r,2)$ and waits (again) to receive ($\lceil (n+1)/2 \rceil$) values of $V(r,2)$; if the received values of $V(r,2)$ are identical, it *irreversibly* decides on that value as the consensus outcome; otherwise, it executes round $r+1$ after doing one of the following: if a non-$\perp$ value $V(r,2)$ has been received, $V_i(r+1,1)$ is set to that value, else one of the $V(0,1)$ values it knows of is randomly chosen to be $V_i(r+1,1)$. The latter occurs when majority nodes had different initial values in round $r = 0$ or all had broadcast $\perp$ as $V(r, 1)$, $r > 0$.

If node i reaches consensus decision in phase 2 or receives the decision from another node (at any time), it broadcasts the decision and stops the execution. (**Expedited Decisions.**) Similarly, if node i receives $V(r', 1\ or\ 2)$ or $V(r', 2)$) while in waiting to receive enough $V(r, 1\ or\ 2)$ or $V(r', 1)$) values respectively, $r' > r$, it adopts the received value and starts executing the appropriate phase of round r'. (**Expedited Executions.**)

5 Simulations

The protocols' performance is studied through simulations and the main parameters used are shown in table 1. To remove the initial bias, each simulation was run for 1000 seconds before the nodes start broadcasting or initiating consensus. Each simulation was run 10 times with different random seeds and the average over these runs constitute a point in all the graphs shown.

Nodes that crash were randomly chosen. A chosen node crashes at an instance distributed uniformly between the time the first and the last m were broadcast. In consensus runs, it crashed at the beginning of round r and phase ph, chosen uniformly in [0, 2] and [1, 2] respectively.

Table 1.

Simulation Parameters	
Simulator	SWANS v1.0.1 [1]
Number of nodes	50
Number of crashes	10 (20%)
Area size	1000m x 1000m
Mobility model	Random Waypoint
Node speed [min, max]	[1m/s, variable]
Pause time	0s
Broadcast generation rate	1/s
Total number of broadcasts	100
Broadcast message size	512bytes
Nodes' buffer size	50 messages
Choice of broadcasters	Random
Choice of consensus initiators	Random
Fading model	Rayleigh
Pathloss model	Two-Ray

The parameters measured are *latency* and *overhead* (bandwidth). Latency for a broadcast protocol is the time elapsed between m being broadcast and the earliest instance when $(n - f)$ nodes receive m; consensus latency is the time elapsed between initiation and the first node deciding.

Broadcast overhead is the *total bytes* transmitted by a broadcast protocol per byte payload of m per node; it is measured as the ratio of the total bytes transmitted during an execution over (payload bytes of $m \times n$). (For example, the overhead estimate for simple flooding will be 1 if space for fields such as $m.id$ is ignored.) Note that the overhead estimate measures total bytes transmitted until transmissions of m and control packets for m end. The consensus overhead is the ratio of the total number of bytes transmitted during a run over the number of nodes.

We vary both density and maximum node speed. The former is the average number of nodes within a disc of radius equal to the nodes' wireless range and is varied by changing the wireless range from 100m to 300m in steps of 25m. The resulting density thus varies from $11/7$ to $99/7$ and the average size of immediate neighborhood from $4/7$ to $92/7$. The max. speed varies from 1m/s to 35m/s.

5.1 Relative Performance of Core Protocols

We first compared the performance of PDP, RDP and PKRM of Section 4 to study the impact of different design approaches. Figure 1 shows the broadcast overhead for β = 5 seconds and maximum speed = 10 m/s, with no crashes.

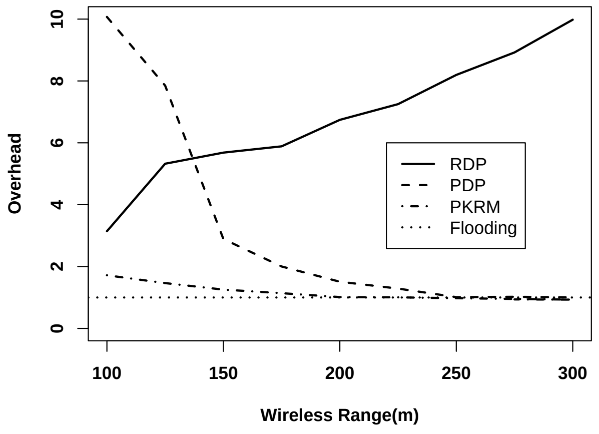

Figure 1. PDP, RDP and PKRM protocols.

The PDP fares best in denser networks while the RDP in sparser ones. It appears that pro-actively transmitting m, with careful use of (only) *realize* packets, yields low overhead in dense networks; similarly, transmitting m only on the *need_to_propagate* basis works well in sparser conditions. These benefits are offset by high overhead in networks of opposite nature. RDP expends too many control packets in dense networks, until neighbors are known to have the same $K(m)$, $KK(m)$ and $KR(m)$, and also often redundantly transmits m even though all neighbors do have m; PDP's periodic transmissions of m itself is unproductive in sparse networks. The overhead for the PKRM supports one's intuition that combining the features of the PDP and RDP, would save bandwidth over a range of densities. It approaches that for simple flooding once the network ceases to be very sparse, and becomes close to 1 in very dense networks.

5.2 Performance of PKRM$_o$

Figures 2 and 3 show how PKRM$_o$ performs for varying densities for three different values of β and with a max speed fixed at 10m/s. The overhead and the latency are very low beyond 150m and 200m wireless range respectively, even though 20% of nodes are allowed to crash during the simulation. (Note: the more the crashed nodes, the longer it takes for m to reach $(n - f)$ nodes.) Figures 4 shows how PKRM$_o$ performs in terms of

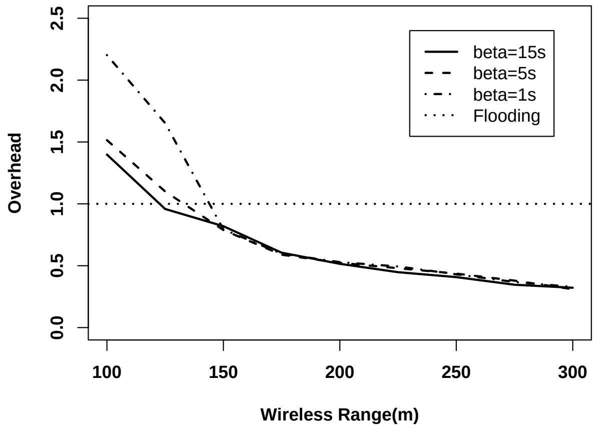

Figure 2. Overhead vs. density (max speed = 10m/s).

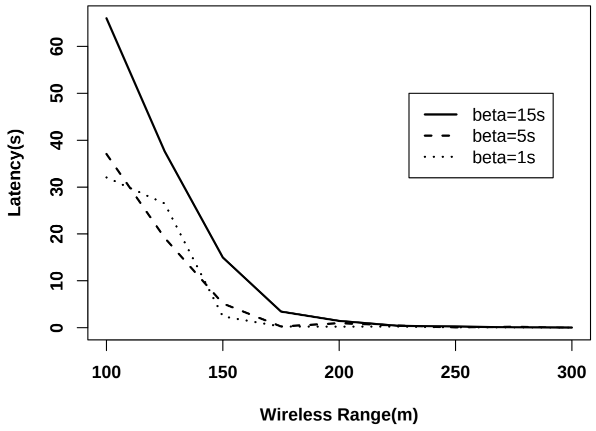

Figure 3. Latency vs. density for max speed = 10m/s

overhead for various speeds, with wireless ranges fixed at 100m and 200m. (The three graphs above the flooding line correspond to 100m.) The overhead at 200m and also above (not shown) seems almost unaffected by the increase in mobility, while at 100m, the mobility actually benefits PKRM$_o$ slightly, since an increased mobility tends to heal partitions quicker.

Figure 5 shows how the latency is affected by node speeds, with a wireless range again fixed at 100m and 200m. (The top three graphs again correspond to 100m.) The observations regarding range and speeds, hold here as well (as in Figure 4).

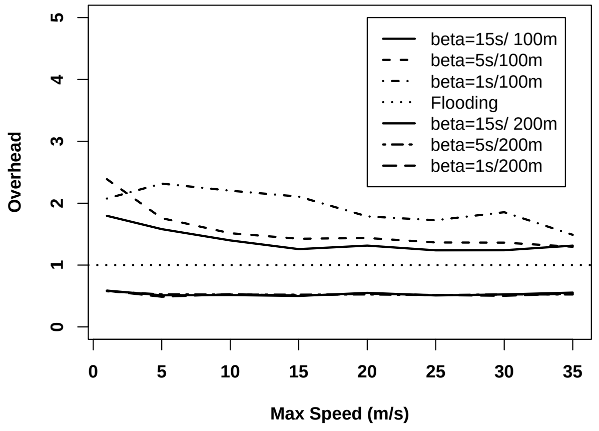

Figure 4. Overhead vs. speed for range = 100m/200m

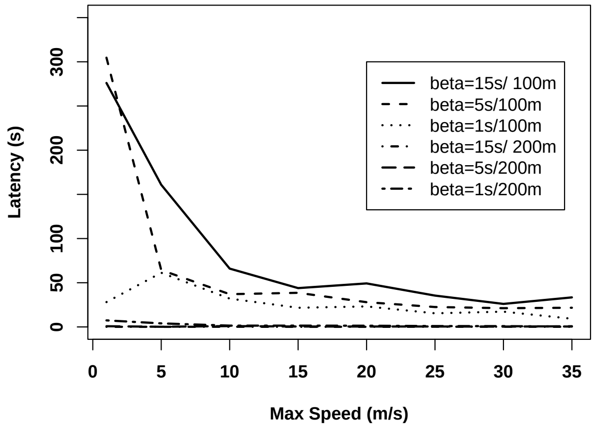

Figure 5. Latency vs. speed for range = 100m/200m

5.3 Performance of Consensus Protocol

The consensus protocol uses PKRM$_o$ (with $\beta = 5s$) for broadcasting. The study reported here is of focused in nature but has surprising results. We report how a consensus run is affected when nodes initiate at the same time but proposing *only* different initial values. (Fewer distinct proposals made at different instances do not tend to slow consensus down.) The number of nodes initiating a consensus varied between 1 and 40. Note that latency is the duration between the consensus initiation and the first decision; after the latter, the protocol behavior is the same irrespective of the number of initial proposals. What we found surprised us: the number of differing initial proposals, had an almost negligible effect on latency and overhead so long as it is more than one. Obviously, when only one proposal was made, the protocol terminated in exactly one round every time, but the difference when varying the number of initial proposals between 2 and 40 was limited. Figures 6 and 7 show how setting the number of different initial proposals to 1, 20 and 40 impacts the overhead and latency over a range of node speeds, with wireless range = 100m. These findings were in sharp contrast to the performance study we did in wired, local area network environments, where the number of different values proposed had a big impact on both latency and overhead.

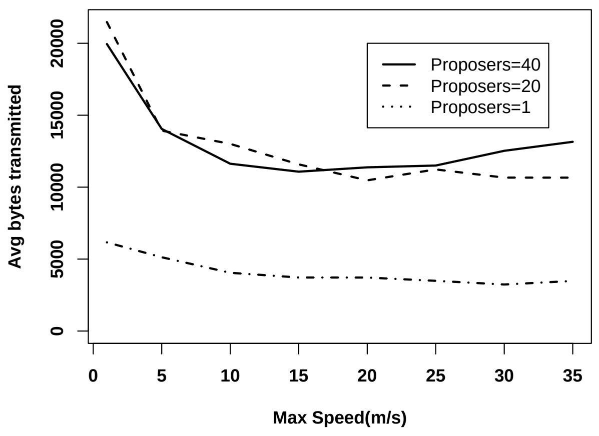

Figure 6. Consensus overhead vs. speed for wireless range = 100m with 1, 20 and 40 different initial values.

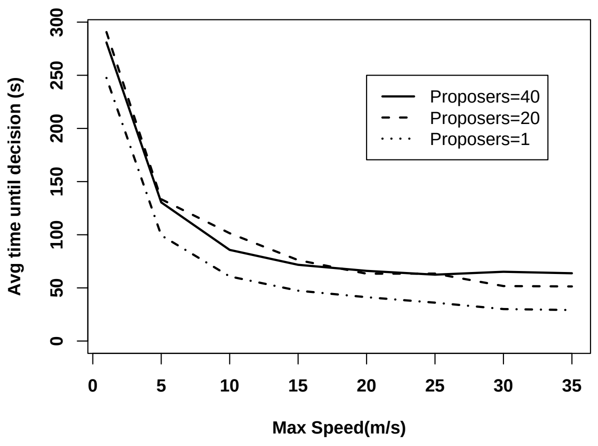

Figure 7. Consensus latency vs. speed for wireless range = 100m

The reason is due mainly to the *absence of LAN effect*. Recall that the nodes wait to receive $(\lceil (n+1)/2 \rceil)$ messages of a given phase. In MANETs, unlike in LANs, a broadcast is received at widely different times by various nodes (i.e., a node normally acts as a forwarder of m, and in PKRM$_o$ after a random delay following the reception). In a typical run of PKRM$_o$, with range = 100m, max speed = 5 m/s and $\beta = 5s$, the first reception of

m by 20, 26, 30 and 40 nodes occurred within 15.83, 36.31, 67.48, and 94.15 seconds respectively after m was broadcast. Thus, fewer nodes complete the waiting much earlier than others, choose a random value in phase_2 and force the stragglers to accept the chosen values as their 'choice'. (See expedited executions in 4.5). That is, the slow ones do not actually make a random choice. Further, the earliest of the earlier ones often manage to impose their choice on a majority of slow nodes. So, the protocol converges towards a decision faster.

As depicted in Figure 8, we seldom observed more than 2 rounds for the first decision to be made and never more than 3 rounds.

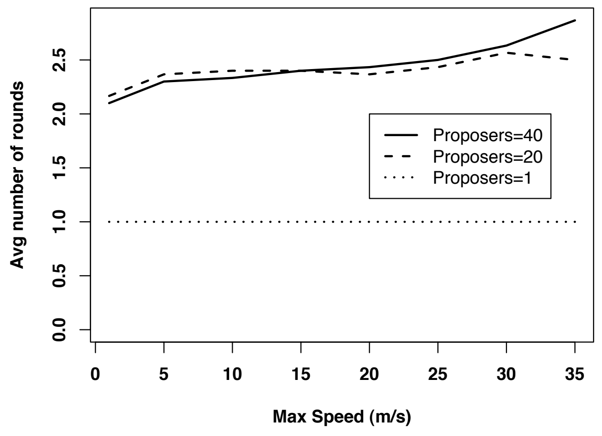

Figure 8. Rounds vs. Max Speed with wireless range=100m.

6 Conclusion

The broadcast protocols presented and studied here ensure maximum coverage that can be guaranteed. This feature, on one hand, helps applications, like consensus, to perform well and, on the other, requires m to be buffered until realization. The latter can cause buffer overflow under heavy message traffic and for very large values of I and n. In such occasions, the protocols can be made to operate for a lower assured coverage (c) by appropriately defining realization, and this will reduce message retention duration. The problem of buffering is addressed in various ways in the literature. In [4], a node 'realizes' once it entrusts m with another 'suitable' node. To identify the latter, it probes nearby nodes for, and collects, feasibility information, and then evaluates an application-tunable *utility* function. Probing involves broadcasting of small packets and is invoked judicially (to minimize overhead). PKRM$_o$ (possibly specified with lower c) can be an ideal candidate for it. Use of (probabilistic) deliverability predictability and of a family of *oracles* determines the suitable node in [15] and [14], respectively. Both assume that the communication opportunities are in general predictable from the nature of the application. Our protocols assume that only B and δ are predictable from the application settings: we have, in the terminology of [14], a *contact* oracle that outputs only B and *queueing* and *traffic demand* oracles for δ. While the above cited works focus storage issue in the context of unicasting, [19, 5] consider one-to-many dissemination. The latter's approach is similar to our RDP, but m is realized once it has been transmitted τ times; using Markovian analysis, τ is estimated to be $O(\ln(n))$

for maximum coverage.

We have assumed an initial configuration for $\mathcal{G}$ wherein each of the n nodes has a unique sequence number in $[0, \ldots, n-1]$ and knows the value of n. This is not easy to achieve and realizing this assumption is addressed as a topic in itself by [2]; interestingly, it is done using a consensus protocol, assuming a broadcast protocol, and disallowing crashes (so that $n > 2f$ holds) until the initial configuration is formed. Once is $\mathcal{G}$ initialized, the problem of managing join/departure requests and of assigning a unique sequence number to a joiner, can be solved in the presence of crashes by imposing a total order on the requests which is feasible with a consensus protocol [13].

We have pursued the approach of *partitionable group* in which any partition that occurs heals eventually. We note here that a partition can be permanent in the *partition-centric* paradigm (e.g., [16]) in which a node's *world-view* is confined to those nodes which are deemed to have connectivity with that node. Our experience and that of others [7, 6] indicate two problems in working with this paradigm: a partition may be falsely concluded (due to inappropriate timeouts used) even when connectivity does exist; this is acknowledged, for example, in [16]. Secondly, when healing of partitions is observed, the state reconciliation which must ensue between the merging components is a message-expensive operation even in fixed network systems. For these reasons, we chose to take the approach of partitionable group which does not allow partitioning between operative nodes to become permanent.

References

[1] R. Barr, Z. J. Haas, and R. van Renesse. JiST: An efficient approach to simulation using virtual machines. *Software Practice and Experience*, 2004.

[2] D. Cavin, Y. Sasson, and A. Schiper. Consensus with unknown participants or fundamental self-organization. In *Proceedings of the 3rd International Conference on ADHOC-NOW 2004*, pages 135–148, Vancouver, July, 2004.

[3] T. D. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. *JACM*, 43(4):685 – 722, July, 1996.

[4] X. Chen and A. L. Murphy. Enabling disconnected transitive communication in mobile adhoc networks. In *ACM Workshop on Principles of Mobile Computing*, pages 21–27, August, 2001.

[5] D. Cooper, P. Ezhilchelvan, and I. Mitrani. High coverage broadcasting for mobile ad-hoc networks. In *the proceedings of the Third IFIP-TC6 Networking Conference*, pages 100–111, 2004.

[6] D. Dolev and D. Malki. The transis approach to high availability cluster communication. *Communications of the ACM*, 39(4):64–74, April 1996.

[7] P. Ezhilchelvan, R. Macedo, and S. Shrivastava. Newtop: a fault-tolerant group communication protocol. In *the Proceedings of 15th IEEE Intl. Conf. on Distributed Computing Systems*, pages 296–306, 1995.

[8] P. Ezhilchelvan, A. Mostefaoui, and M. Raynal. Randomized multivalued consensus. In *Proceedings of the 4th International Symposium on Object-Oriented Real-Time Computing*, pages 195–200, 2001.

[9] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[10] R. Friedman and G. Tcharny. Evaluating failure detection in mobile ad-hoc networks. *Technical Report, Computer Science Department, Technion*, CS-2003(06):0–22, October, 2003.

[11] T. Gopalsamy, M. Singhal, D.Panda, and P. Sadayappan. A reliable multicast algorithm for mobile ad hoc networks. In *Proceedings of ICDCS*, 2002.

[12] R. Guerraoui, M. Hurfin, A. Mostefaoui, R. Oliveira, M. Raynal, and A. Schiper. Consensus in asynchronous distributed systems: A concise guided tour. (LNCS 1752):33–47, 2000.

[13] V. Hadzilacos and S. Toueg. Fault-Tolerant Broadcasts and Related Problems. In S. Mullender, editor, *Distributed Systems*, pages 97–146. Addison-Wesley, 1993.

[14] S. Jain, K. Fall, and R. Patra. Routing in delay tolerant network. In *ACM SIGCOMM*, pages 299–311, 2004.

[15] A. Lindgren, A. Doria, and O. Scheleén. Poster: Probabilistic routing in intermittently connected networks. In *ACM MobiHoc*, June, 2003.

[16] S. Nesargi and R. Prakash. Locating cache proxies in manets. In *the Proceedings of MobiHoc*, pages 175–186. ACM Press, 2002.

[17] R. Prakash, N. Shivaratri, and M. Singhal. Distributed dynamic channel allocation for mobile computing. In *the Proceedings of 14th Symposium on Principles of Distributed Computing*, pages 47–56. ACM Press, 1995.

[18] G. Tel. Robust algorithms (chapter 13.2). In *Introduction to Distributed Algorithms*, pages 429–434. Cambridge University Press, 2001.

[19] W. Vogels, R. van Renesse, and K. Birman. The power of epidemics: Robust communication for large-scale distributed systems. In *Proceedings of HotNets-I*, pages 131–135. ACM Press, 2002.

7 Appendix

7.1 Proposition 1

Suppose that nodes never crash (each node is operative) and have no knowledge about immediate neighborhood. A broadcast protocol with the SSP cannot guarantee $c \geq 1$, unless every node with m transmits m at least once every τ time, $\tau < (B + \delta)$, until transmissions of m are stopped for ever.

Proof: Let us hypothesize that there is a broadcast protocol which, in every execution, (1) preserves the SSP and ensures $c \geq 1$; and, (2) chooses a node that has m and makes it 'silent' (i.e., not transmit m) for at least $(B+\delta)$ time. Specifically, the protocol chooses a silent duration S, $S \geq B + \delta$, a timing instance s, $s + S < t_e$, and some node with m, and keeps the chosen node silent during $[s, s + S]$.

The proposition is proved by contradiction. Since I is unknown and $(t_e - t_b)$ is finite, it is possible to have executions in which I is larger than $(s + S) - t_b$ and also $(t_e + S) - s$. Consider one such execution in which m is broadcast when the broadcaster's immediate neighborhood is empty.

Let $\mathcal{P}$ be the set of nodes that have m at some $t \geq t_b$, and $\overline{\mathcal{P}}$ be the set of those that do not have m. $\mathcal{P}$ is a singleton set at t_b (containing only the broadcaster node). So, $\mid \mathcal{P} \mid \geq 1$ at any t and $c = \mid \mathcal{P} \mid -1$. Choose t such that $\overline{\mathcal{P}}$ at t is not empty. (This is possible as $\mid \overline{\mathcal{P}} \mid = (n - 1)$ at $t_b \leq t$.) Let the MANET, starting from t, keep the nodes of $\mathcal{P}$ disconnected from those of $\overline{\mathcal{P}}$, except for the period mandated by the liveness property.

$t \leq s$: Since $I > (s + S) - t_b$, the direct connectivity expected during $[t, t + I]$ can occur during $[s, s + S]$. Let the MANET choose a node from $\mathcal{P}$ which the protocol has made silent at s, and a direct connectivity period that starts at $s + \delta$ and ends at $s + B + \delta \leq s + S$.

$t > s$: Since I is larger than $(t_e + S) - s$, the direct connectivity expected during $[t, t + I]$, can occur during $[t_e, t_e + S]$, after all transmissions of m end at t_e.

No node in $\overline{\mathcal{P}}$ receives m during or outside the direct connectivity period and $\mid \mathcal{P} \mid$ does not increase after t. At t, $\mid \mathcal{P} \mid \geq 1$ and $c \geq 0$. That is, $c \geq 1$ is not ensured.

7.2 Proposition 2

Any crash-tolerant broadcast protocol that has the SSP, cannot guarantee that more than $(n - f - 1)$ nodes receive an operative node's broadcast, even if more than $(n - f)$ nodes, including the broadcaster, do not crash.

Proof (By Contradiction): Consider two executions of the protocol. Let t_{e_1} and t_{e_2} be the timing instances in these executions, after which nodes do not retain m for propagation. By the SSP, $(t_{e_1} - t_b)$ and $(t_{e_2} - t_b)$ are finite durations. Let $\mathcal{F}$ be any set of f nodes which does not include the broadcaster of m, and $\overline{\mathcal{F}}$ be its complementary subset.

Execution 1: All nodes of $\mathcal{F}$ have already crashed before t_b. Since there are only $(n - f)$ operative nodes (including the broadcaster), $c < (n - f)$ at t_{e_1}.

Execution 2: No node has crashed before t_b and all n nodes remain operative until t_{e_2}. However, the MANET keeps nodes of $\mathcal{F}$ outside the wireless range of every node in $\overline{\mathcal{F}}$ until t_{e_2}. This is possible if the unknown $I > (t_{e_2} + B - t_b)$, and the liveness property will be met when the MANET ensures direct connectivity between some nodes of $\mathcal{F}$ and $\overline{\mathcal{F}}$ just after t_{e_2}. Nodes of $\mathcal{F}$ thus neither receive m nor execute the protocol for m.

The protocol design system cannot distinguish these two executions for three reasons: (1) no node crashes *during* the executions, (2) nodes of $\mathcal{F}$ do not execute the protocol in both cases, and (3) there is no mechanism for nodes of $\overline{\mathcal{F}}$ to detect whether a node in $\mathcal{F}$ is crashed or operative. Therefore, $c < (n - f)$ in the second execution as in the first. This contradicts the hypothesis, since all n nodes are operative throughout the second execution.

7.3 Correctness Arguments for the Proactive Dissemination Protocol (PDP)

Consider an execution in which either the broadcaster is operative or an operative node receives m. Let $t \geq t_b$ a timing instance during this execution and $\overline{\mathcal{P}}$ denote the set of all those operative nodes not in $\mathcal{P}$ at t.

Delivery. Say, no operative node that has m has realized m at t. Choose $\mathcal{P}$ to be any non-empty subset of all those operative nodes with identical $K(m)$ at t. (By the nature of the execution considered, there is at least one such singleton $\mathcal{P}$.) $\overline{\mathcal{P}}$ cannot be empty. Otherwise, $\mathcal{P}$ has all operative nodes (which are at least $n - f$) and all of them have identical $K(m)$; since a node has 1 for its own bit in its $K(m)$, all nodes in $\mathcal{P}$ have realized m which is not the case at t.

When node i of $\mathcal{P}$ and node j of $\overline{\mathcal{P}}$ directly connect, either j receives m for the first time or the nodes exchange their different $K(m)$. Thus, as the execution progresses with no node realizing m, each occurrence of direct connectivity increases 1 bits in the $K(m)$s of some operative nodes. Since $(n - f)$ is finite, some operative node(s) must realize m within some finite duration.

Termination. Say, only some operative nodes have realized m at t. Let $\mathcal{P}$ be the set of those operative nodes that have m but not realized it at t. When node i of $\mathcal{P}$ and node j of $\overline{\mathcal{P}}$ directly connect, (1) node i will realize if node j has realized, or (2) node j receives m for the first time. Since the number of operative nodes is finite, case (2) will cease and all operative nodes that have m realize m in finite time. *realize(m.id)* will no longer be transmitted.

7.4 Correctness Arguments for the Reactive Dissemination Protocol (RDP)

Claim. Consider node i and node j with $K_i(m) \neq K_j(m)$ at some time t during an execution. It is not possible for both $KK_i(m)[j] = 1$ and $KK_j(m)[i] = 1$ at t.

With no loss of generality, suppose that $KK_j(m)[i] = 1$ at t. This is possible only if node j has known in the past that $K_i(m) = K_j(m)$. But at t, $K_i(m) \neq K_j(m)$. That is, node i has increased its $K_i(m)$ which must have caused its $KK_i(m)$ to be re-set. Further, node i could not have learnt that node j also increased its $K_j(m)$ in the same way. Therefore, $KK_i(m)[j]$ cannot be 1, and can only be 0.

The claim suggests that when unrealized nodes i and j with different $K(m)$ experience direct connectivity, at least one of them will transmit at step 3. Further, if $KK_i(m)[j]$ = 0 and $KK_j(m)[i] = 1$, then node j gains more 1-bits. The rest of the arguments can be constructed by choosing appropriate $\mathcal{P}$ as was done for PDP above, and is omitted.

7.5 Correctness Arguments for PKRM Protocol (PDP)

The Arguments follow from those for the PDP, with an observation that when $2(\beta + \delta) \leq B$, the PKRM uses direct connectivity between node i with unrealized m and a node j effectively: node j receives m if it has no m (steps 6 and 3a), or both nodes exchange their $K(m)$ if both are unrealized (step 3a).