RADcast: Enabling Reliability Guarantees for Content Dissemination in Ad Hoc Networks

Bo Xing, Sharad Mehrotra and Nalini Venkatasubramanian
School of Information and Computer Sciences, University of California, Irvine
{bxing, sharad, nalini}@ics.uci.edu

***Abstract*— This paper deals with the problem of reliable and fast broadcast of mission-critical data with rich content over ad hoc networks. Existing approaches to dissemination reliability often assume network size knowledge, or that receivers know about the dissemination in advance. Without making similar assumptions, we propose a distinct approach which accommodates the varying reliability needs of applications. We develop the RADcast (Reliable Application Data broadcast) protocol as an integration of two components: (a) Peddler, which ensures that receivers obtain the dissemination metadata, and (b) Pryer, which delivers the actual data to dissemination-aware receivers. We indicate how reliability guarantees/performance tradeoffs can be achieved by a careful instantiation of Peddler and Pryer. We implement RADcast on mobile devices inside a middleware and determine its feasibility. Furthermore, through extensive simulations, we show that RADcast achieves desired reliability in all cases, and performs consistently under varying network conditions and device mobilities. As compared to existing approaches, RADcast either incurs significantly lower latency/message overhead, or reduces latency by 50% with a tradeoff in message overhead.**

I. Introduction

In this paper, we study reliable broadcast of rich content in ad hoc networks. Much of the prior work on ad hoc network broadcasting (e.g., [1] [2]) has focused on control data dissemination, where the goal is to send small routing-related data to specific destination(s) with unknown location(s) (e.g., in AODV [3]). In contrast, our interest is to study application level broadcast, with the goal of delivering application-generated rich data (in the form of text, images, audio, video, etc.) from a source node to all other nodes. Our motivation stems from multiple scenarios in the crisis response domain. As a concrete instance of such a scenario, consider the following application being developed by the Orange County Fire Authority (illustrated in Fig. 1). During disaster relief, fire trucks, ambulances, police cars and helicopters gather at the rescue site. Any of them can generate maps annotated with hazard information (e.g., chemical leakage) or take snapshots of nearby regions, and share them with all others in real time. The rescue personnel thus keep their situational awareness in sync. These contents are securely transmitted via an ad hoc mode 802.11 variant protocol at the 4.9GHz public safety frequency band. In such content broadcast scenarios, in addition to speed, applications may require guarantees on the reliable delivery of mission-critical data. The reliability needs of applications might vary – some applications demand to cover at least a certain percentage of nodes; others desire to reach all possible recipients.

Reliably delivering rich data to all nodes in an ad hoc network is a big challenge. Due to the uncertainty of wireless transmissions, data can easily get lost in the air. Because of dynamic topology changes caused by nodes moving or powering on/off, accurate topological knowledge is hard to obtain. The disseminator may or may not know how many potential receivers there are (e.g., in our motivating use case scenario, receivers are from different organizations). Hence, achieving reliability often involves sacrificing transmission/time efficiency, or even making the heavy assumption that no receiver crashes or leaves the network during the dissemination.

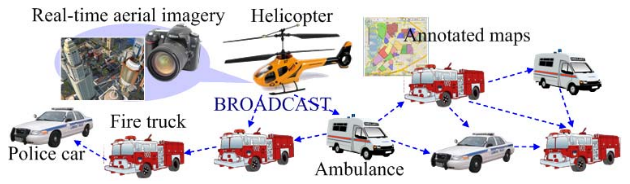


Fig. 1. Dissemination of Mission-Critical Rich Content: An Example Scenario

Existing reliable broadcast techniques usually rely on one of the following mechanisms. (i) Acknowledgements: the source node and/or the relaying nodes collect acknowledgements from the receivers [4] [5] (by assuming that they have some degree of topological knowledge). (ii) Continuous push: the source node and/or the relaying nodes repeatedly transmit the data until complete coverage is deduced [6] (a key assumption being that they know the network size). (iii) Continuous pull: receivers keep requesting data from the source node and/or the relaying nodes until receiving the whole data [7] [8] (by assuming that they are aware of the dissemination ahead of time).

In this paper, we propose a middleware approach to reliable content dissemination (it leaves lower layer mechanisms untouched, and thus can be deployed directly on off-the-shelf devices). We develop the *RADcast* (Reliable Application Data broadcast) protocol which addresses the varying needs of dissemination applications. The overriding goal is *reliability* – not just to maximize delivery ratios, but also to provide guarantees (i.e., assurance that desired level of reliability will be achieved). Meanwhile, *timeliness* (short latency being incurred before recipients receive the content) and *message efficiency* (small number of messages/bytes being transmitted) are pursued as well. The targeted environment is a connected wireless network formed by mobile nodes, which may not have infrastructure support but rather communicate through direct links. The network scale under consideration is tens to hundreds of nodes, as we believe in reality it would be rare that thousands of devices satisfy communication needs solely by ad hoc connectivity.

RADcast decomposes the reliable dissemination task into two concurrent subtasks, and thus integrates (i) an awareness assurance protocol (*Peddler*) and (ii) a data diffusion protocol (*Pryer*). Peddler is responsible for ensuring that all receivers obtain the dissemination metadata, while Pryer takes advantage of receivers' awareness to provide guaranteed content delivery. By tuning the two components and their parameters, RADcast offers three levels of reliability guarantees that applications can choose from. The design of RADcast accommodates the distinguishing characteristics of ad hoc networks: it does not rely on any routing or overlay construction protocols, nor does

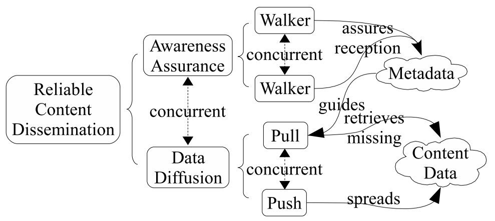


Fig. 2. RADcast: A Distinct Approach to Reliable Content Dissemination

it require topological information. Through extensive simulations, we show that RADcast always achieves desired reliability, while still outperforming existing approaches in either latency, or both latency and message overhead. It performs consistently well under all network conditions, and scales with both network size and content size. We also implement RADcast on mobile devices and determine its feasibility in real settings.

In the next section, we overview the design philosophy of RADcast. We present Peddler and Pryer in Section III and IV respectively. We then describe how RADcast intelligently integrates Peddler and Pryer (Section V). We sketch its implementation in Section VI and evaluate its performance in Section VII. Section VIII reviews related work and concludes the paper.

II. RADcast: Decomposing the Problem

Supporting reliability in large data dissemination is challenging. Solutions developed for homogeneous wired networks typically divide data into equal size chunks, followed by concurrent exchange of chunks amongst nodes [9]. Extending these solutions to dynamic heterogeneous networks is not straightforward and has been shown to be NP-hard [10]. RADcast is built on a similar premise – to bypass IP fragmentation which causes performance issues in wireless networks [11], RADcast pre-fragments contents into equal size units; each fragment fits into the MAC layer MTU (Maximum Transmission Unit). In the meantime, RADcast generates metadata which describes the dissemination (containing source address, file ID and number of fragments), and encapsulates it in every message.

The fragmented content along with metadata are propagated through the network. A receiver once hearing the first RADcast message obtains the metadata and thus becomes content-aware. It then autonomously retrieves missing fragments from neighbors according to the metadata. This concurrent push and pull process constitutes the *data diffusion* subtask of RADcast. While creating fast awareness, it however does not suffice to provide guarantees on how many nodes receive the metadata. Hence, in parallel with data diffusion, RADcast executes an additional subtask – *awareness assurance*. The goal is to ensure that all (connected) nodes are informed about the ongoing dissemination. *Rather than to greatly enhance delivery ratio, the awareness assurance subtask is intended to provide the disseminator with reliability guarantees.* This approach – decomposing the dissemination task – is depicted in Fig. 2.

While data diffusion is relatively straightforward, awareness assurance is hard. A lightweight mechanism is needed which quickly does the job without introducing much traffic. Considering that, RADcast adopts a network traversal approach: when dissemination starts, the source node emits a *walker* (a mobile-agent-like message that carries only the metadata) into the network. The walker hops between neighboring nodes, and stops traveling once it has visited all nodes. The nodes visited by the walker, as a result, are guaranteed to be dissemination-aware. For faster traversal and better scalability, RADcast generalizes the idea to the employment of multiple concurrent walkers – the source issues several walkers, which travel around to visit receivers in parallel and thus share the traversal workload.

In other words, RADcast reduces reliable dissemination to a fast network traversal problem. While complete traversal is required, the following metrics are to be optimized: (i) *cover moves/cover time* (the number of walker moves/the time elapsed before all nodes are visited once); (ii) *termination moves/termination time* (the number of walker moves/the time elapsed before walkers stop traveling); (iii) *message overhead* (the number of messages/bytes transmitted in supporting walker movements). This problem is NP-hard (it easily converts to the Hamiltonian path problem), and *is further harder if the network size is unknown to walkers*. In the next section, we develop Peddler as a solution to the fast network traversal problem.

III. Peddler: Awareness Assurance

Ad hoc network traversal has been studied for applications in various contexts. (i) Group communications: Dolev et al. employ randomly walking agents to collect and distribute group membership information [12]. The cover moves (for n nodes) is proved to be O(n^3) on average. (ii) Routing in sensor networks: a route is computed by an agent which traverses static sensors following the depth-first strategy [13]. Depth-first generates a spanning tree and backtracks once a leaf node is reached. (iii) Token circulation for ordered message delivery: the goal is to minimize nodes' waiting time before they get the token and talk [14]. It is shown that passing the token to the least recently visited node in each move yields good performance.

Peddler, the awareness assurance component of RADcast, adopts the idea of traveling walkers as well. A walker is in spirit the same as a mobile agent or a token in the aforementioned schemes. However, targeting a different context, Peddler aims to optimize not only the cover time, but also the termination time of walkers. For optimizing cover time, Peddler attempts to find the best walker traveling routes. To optimize termination time, Peddler needs to be able to tell whether walkers have visited all connected nodes. This is straightforward if initially the source node could let walkers know how many nodes there are (i.e., if the network size or its estimate is known to the source node); in this case, termination time simply equals cover time. On the other hand, if network size is unknown, deducing traversal completion is a nontrivial task. Optimistic deduction may leave some nodes unvisited, while pessimistic deduction leads to long termination time and high overhead. Hence, Peddler seeks to find the tradeoff point where traversal completion can be correctly inferred at the earliest time.

In the following, we first develop intuitions for the single walker scenario, and then generalize them for multiple walkers.

1) Single Walker: In the single walker case, the source node issues one walker when dissemination starts. The walker travels around, counts the number of distinct nodes it has visited, and stops when a termination condition is met (signifying that all nodes have been visited). We now describe how Peddler optimizes the cover time and termination time of the walker.

Optimizing Cover Time: Peddler employs the Least Recency of Latest Visit (LRLV) strategy to dictate walker traveling routes. LRLV marries the best of random walk and the recency-based token circulation scheme – in each move, if there are fresh (**Definition**: a node is *fresh* if it has never been visited by any walker) neighbors, the walker hops to one of them which is

Pseudo-Code 1 (a) DCTC(S), (b) DCTC(W)

```
(a)                                          (b)
1: $w$ visits $N_i$                          1: $w$ visits $N_i$
2: if $N_i \notin S_1$ then                  2: if $N_i \notin S_1$ then
3:     $S_1 \leftarrow S_1 \cup N_i$;        3:     $l_1 \leftarrow l_1 + l_2 + 1$;
4:     $S_2 \leftarrow \{N_i\}$;             4:     $l_2 \leftarrow 0$;
5: else if $N_i \in S_1$ then                5: else if $N_i \in S_1$ then
6:     $S_2' \leftarrow S_2$; $S_2 \leftarrow S_2 \cup N_i$;   6:     $l_2 \leftarrow l_2 + 1$;
7:     if $S_2' = S_2 = S_1$ then            7:     if $l_2 = \eta \times l_1$ then
8:         $w$ terminates;                   8:         $w$ terminates;
```

randomly picked; otherwise it hops to the neighbor whose latest visit by the walker is the earliest. The rationale behind LRLV is to check the already visited nodes to find any missed neighbors of them without forming infinite loops. Unlike depth-first, which performs backtracking along the reverse path, LRLV does not require persistent and symmetric links, and thus is applicable in dynamic networks as well.

Optimizing termination time: Peddler uses three termination conditions; a walker terminates once one of them is met, whichever comes first: (i) when the number of distinct nodes the walker has visited equals what the source node assigns; (ii) & (iii) when the *Double-Check Termination Conditions* (DCTC(S) and DCTC(W), S for strong, W for Weak) are met. DCTC(S) and DCTC(W) are responsible for determining walker termination when network size is unknown or the network size given by the source node is overestimate. DCTC(S) uses a deterministic approach and leads to short termination time, whereas DCTC(W) employs a probabilistic approach and is intended to deal with scenarios where some nodes move arbitrarily fast or crash/leave so that DCTC(S) cannot be met.

DCTC(S) originates from the following observation. If the walker visits a set of nodes in the first place and then revisits them one more time during which no fresh node is visited, chances are that no more nodes remain to be visited; this is because in LRLV the walker always moves to fresh neighbors if there is any. The rationale behind DCTC(S) is analogous to a two-round checking process: the walker visits a group of nodes in the first "round", and then in the second "round" it visits them again to double-check whether any neighbors of them were missed. We illustrate the intuition of DCTC(S) using Pseudo-Code 1(a), where the walker (w) is thought of as maintaining two node sets: first-round node set (S_1) and second-round node set (S_2). S_1 contains the nodes w has visited. S_2 contains the nodes w has revisited since the most recent time it visited a fresh node, plus that node. Both sets initially contain only the source node, and are updated in each move. S_2' is used to memorize the previous state of S_2.

Thus, the termination time dictated by DCTC(S) ($term_n$ for a network of n nodes) is a function of cover time: $term_n = cover_n^1 + cover_n^2 + 1$, where $cover_n^1$ is the cover time for visiting all the n nodes in the first "round", starting from the source node; $cover_n^2$ is the cover time for the second "round", starting from the nth visited node. Intuitively, DCTC(S) is conservative in that it restarts the second "round" every time it encounters a fresh node. In fact, this conservativeness is essential in adapting walker terminations to the dynamic network environments. In Section VII we will show its importance and effectiveness by comparing DCTC(S) to other termination conditions.

DCTC(W) bears similar idea of DCTC(S), but takes into account the possibility that a visited node may crash/leave before it is revisited, in which case DCTC(S) could not be met. As opposed to inspecting which nodes have been visited, DCTC(W) determines the end of the second "round" by examining the number of walker moves that have been made (or, "round" length). That is, a walker terminates when the length of its second "round" is η times that of its first "round" (η is a predefined protocol parameter, which approximates the ratio of the upper bound of cover moves to its lower bound). When DCTC(W) is met, it is highly probable that the second "round" has ended and no node that is still connected was missed. Similar to DCTC(S), the intuition of DCTC(W) can be described using Pseudo-Code 1(b). The walker (w) additionally maintains first-round length (l_1) and second-round length (l_2), which are initially 0 and are updated in each move.

2) Multiple Concurrent Walkers: Peddler gracefully extends the walker idea to multiple walkers, and thus expedites traversals by exploiting the concurrency among them. When a dissemination starts, the source node simultaneously issues several walkers. Each walker travels following a general version of the LRLV strategy – in each move, if no fresh neighbor exists but certain neighbors have been visited only by sibling walkers (**Definition**: the walkers that are issued concurrently are *sibling walkers* to each other), the walker hops to one of them which is randomly picked; if the walker has visited all the neighbors, it hops to the one which it visited least recently.

If the network size is known, the source node evenly divides the traversal task and assigns to each walker a relatively small number of nodes to visit. A walker stops traveling once it has visited assigned number of distinct nodes that had not been visited by sibling walkers. Hence, when all walkers terminate, all nodes have been visited. On the other hand, if network size knowledge is unavailable, each walker terminates independently according to DCTC(S) and DCTC(W) (to accommodate that, on line 2 of both sides in Pseudo-Code 1, the condition changes to "N_i is not in any walker's S_1"). Thus, single walker is just a special case of multiple walkers in Peddler.

3) Properties of DCTC(S): DCTC(S) is formally stated as follows. Suppose a walker w at its t_1th move, visits node N_v for the first time, and thus has visited v distinct nodes N_1, N_2, ..., N_v, each of which was fresh before being visited by w. Further, suppose at its t_2th move, w revisits $N_k (k \in \{1, 2, ..., v-1\})$ for the first time after its t_1th move. **DCTC(S):** w terminates at its (t_2+1)th move, if (i) from the (t_1+1)th to the t_2th move w revisits N_1, N_2, ..., N_{v-1}, but visits no fresh node, and (ii) at its (t_2+1)th move w does not visit a fresh node.

In any case, DCTC(S) assures that all nodes that do not "hide" from walkers are visited. Strictly speaking, **Theorem:** Given that walker movements follow LRLV, DCTC(S) guarantees complete traversal of all stably reachable nodes (**Definition**: a node is *stably reachable* if, during the short time period since walkers are issued till all of them terminate, there exists a fixed path from this node to the source node; two neighboring nodes are *stably connected* if they keep connected during this time period). This can be proved by contradiction on basis of the following **Lemma:** If a walker w terminates when DCTC(S) is met, a stably connected neighbor of a node that has been visited by w must have been visited by w or its sibling walkers. The complete proofs are provided in [15].

Another property of DCTC(S) is that, if a walker terminates before DCTC(S) is met, the traversal could be incomplete. This can be verified using a counter-example ([15]). In section VI we will describe how we implement DCTC(S) and DCTC(W) along with LRLV in a distributed and lightweight manner.

IV. Pryer: Data Diffusion

The data diffusion component of RADcast, Pryer is based on a mix of push and pull - while pull guarantees complete data reception, push controls the paces of data propagation. The push is through a form of scoped flooding. The pull happens only between neighboring nodes without the puller having to know the pullee. As opposed to gossiping with remote nodes [16] [17], this form of pull avoids heavy traffic and eliminates the need for topological knowledge.

1) Data Push: The source node sends out the fragments sequentially at a certain interval (called *source push interval*). A node N_i, when receiving a fragment f for the first time, schedules rebroadcasting f with a random delay. During this backoff period, if N_i hears a neighbor rebroadcasting f, it cancels its pending rebroadcast. Thus, rather than every node rebroadcasting, only a small portion of nodes do so. This saves a number of unnecessary transmissions, especially when the network is dense. The nodes that are not covered by the scoped flooding will retrieve the missing fragment through pull.

2) Data Pull: Data pull is realized through receivers advertising reception progress in all outgoing messages. A node N_i piggybacks its reception progress if it is transmitting a fragment (implicit ad). Otherwise N_i transmits its reception progress alone (explicit ad) when (i) N_i knows that a neighbor has certain fragments it is missing (by tracing neighbors' reception progress), or (ii) N_i has not transmitted any message for a long time (by running an idling timer). N_i's ad tells neighbors (i) which fragments N_i is missing and neighbors can retransmit for N_i, and (ii) which fragments N_i has received and can retransmit for neighbors. Upon hearing an ad from N_i, neighboring nodes selectively retransmit N_i's missing fragments which they have. N_i's reception process is represented using four sequence numbers ($l1_i$, $r1_i$, $l2_i$, $r2_i$). They delineate N_i's two lowest missing-fragment gaps. Thus, N_i retrieves fragments with smaller sequence numbers earlier, which is beneficial if the order of the fragments is important to the application.

To guarantee its reception of all fragments, N_i starts an idling timer as soon as it gets the metadata, and destroys it once the complete content is received. The timer counts the time since N_i's most recent transmission. Every time it expires, N_i transmits an explicit ad after a small delay, or piggybacks it if N_i has a data fragment to transmit during the backoff time.

3) Protocol Description: In the Pryer protocol, nodes' actions are driven by timer expirations and message receptions. Pseudo-Code 2 depicts how a node reacts when receiving a Pryer message. Other than that, Pryer optimizes on message efficiency, and hence time efficiency as well, to make itself potentially adaptive to network density and data size. First, every message is transmitted after a random delay to allow possible cancelation or piggybacking of it. Second, each node suppresses explicit ad transmissions and fragment retransmissions by enforcing lower bounds on transmission intervals, so as to prevent redundant messages from overwhelming the network. For the proof of the correctness of Pryer, please refer to [15].

Pseudo-Code 2 Reacting to Pryer Messages

```
1: if $N_i$ receives a Pryer message $M$ sent by $N_j$ then
2:     if $M$ contains a fragment $f$ then
3:         if $N_i$ receives $f$ for the first time then
4:             $N_i$ schedules rebroadcasting $f$
5:         else if $N_i$ has a pending transmission of $f$ then
6:             $N_i$ cancels the pending transmission
7:     if $N_i$ has $N_j$'s missing fragments then
8:         $N_i$ schedules retransmitting one randomly picked
9:     else if $N_j$ has $N_i$'s missing fragments then
10:        if $N_i$ has no pending transmission then
11:            $N_i$ schedules transmitting explicit ad
12:    else if $N_j$ and $N_i$ have same missing fragments then
13:        $N_i$ cancels pending explicit ad transmission if any
```

TABLE I
RADcast: Supporting Adaptive Reliability

	Pryer	Peddler	Net Size Knowledge
Max	✓	✓	×
Lower-Bounded	✓	✓	✓
Best-Effort	✓	×	N/A

V. RADcast: Supporting Adaptive Reliability

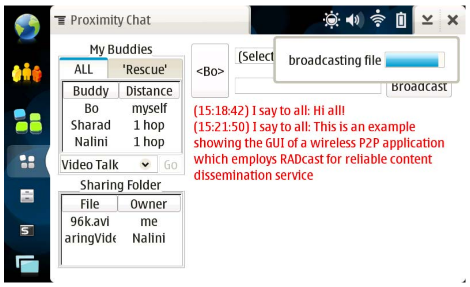
A naive integration of Peddler and Pryer in RADcast would assume uniform reliability needs and blindly utilize network size knowledge. In practice, however, dissemination applications vary widely in terms of the reliability guarantees they need. Network size knowledge can be inaccurate or outdated. Moreover, inherent tradeoffs exist between reliability and other aspects of performance. Hence, RADcast needs to leverage all these factors through an intelligent integration of Peddler and Pryer and careful calibrations of their parameters.

To support varying reliability needs, RADcast offers three levels of reliability that applications can choose from: (a) *Max Reliability Guarantee* (all reachable nodes are guaranteed to receive the content), (b) *Lower-Bounded Reliability Guarantee* (specific number of nodes are guaranteed to receive the content), and (c) *Best-Effort Reliability* (reach as many nodes as possible; note that this does not imply low reliability, just that no guarantees are offered). In each case, RADcast attempts to meet the desired reliability at the lowest cost and tailors its components for optimized performance (Table I).

Max Reliability Guarantee: Since the application demands to reach all reachable nodes in this case, RADcast takes full advantage of Peddler and Pryer. Meanwhile, it ignores any knowledge of network size (which cannot be verified to be accurate), and lets walkers explore all possible recipients within reach. In doing so, RADcast achieves the highest possible reliability at the cost of increased latency/message overhead.

Lower-Bounded Reliability Guarantee: In this case, RADcast executes both Peddler and Pryer, and passes network size knowledge (if available) to walkers. By doing that, RADcast reduces walker termination time/message overhead and lowers cost; meanwhile it guarantees delivery to specific number of recipients. Network size knowledge can come from several sources: either the application explicitly specifies the number of nodes to reach (e.g., the disseminator knows how many devices his organization has); or RADcast leverages knowledge from other layers if available: (a) routing layer (e.g., running a proactive routing protocol), (b) middleware layer, where Peddler uses itself as a tool to measure network size (walkers report to the source how many nodes they have visited when they terminate), which serves as input for its future executions.

Best-Effort Reliability: The application seeks to reach as many recipients as it can, but places emphasis on timeliness and efficiency. In this case, RADcast eliminates the initiation (and consequent cost) of Peddler and only executes Pryer. As we will show later, Pryer by itself can achieve high delivery ratios in most situations although no guarantees can be made.

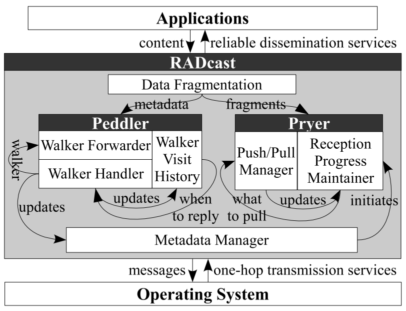


Buddy	Distance
Bo	myself
Sharad	1 hop
Nalini	1 hop

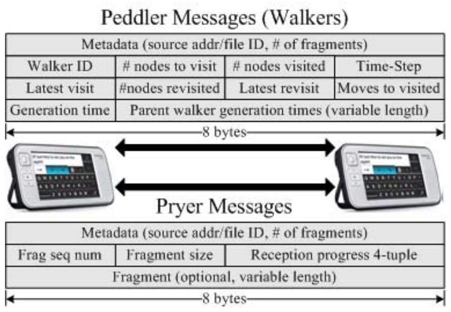
(15:18:42) I say to all: Hi all!
(15:21:50) I say to all: This is an example showing the GUI of a wireless P2P application which employs RADcast for reliable content dissemination service

File	Owner
96k.avi	me
aringVide	Nalini

(a) Example Application GUI



(b) Implementation Structure


Peddler Messages (Walkers)
Metadata (source addr/file ID, # of fragments)
Walker ID
# nodes to visit
# nodes visited
Time-Step
Latest visit
#nodes revisited
Latest revisit
Moves to visited
Generation time
Parent walker generation times (variable length)
8 bytes
Pryer Messages
Metadata (source addr/file ID, # of fragments)
Frag seq num
Fragment size
Reception progress 4-tuple
Fragment (optional, variable length)
8 bytes

(c) Message Transmissions and formats

Fig. 3. RADcast: Implementation on Mobile Devices

VI. Implementation

In order to determine its feasibility, we have implemented RADcast on Nokia N800 Internet Tablets [18] (the operating system is a modified version of Debian GNU/Linux). It is encapsulated in a middleware suite which offers a variety of wireless peer-to-peer communication services. An example application we have built on top of the middleware is a wireless proximity messenger (a snapshot shown in Fig. 3(a)). RADcast can be invoked programmatically via an API call that has the following format: *RADcast(char** filePaths, int numFiles, int level)*, where *filePaths* is a string array storing the paths of the files to be broadcast; *numFiles* indicates the number of files; *level* selects the desired reliability. Note that when multiple files are being disseminated, they are treated as one single file, and thus Peddler (if invoked) needs to be executed only once.

RADcast is implemented as a set of interactive modules (Fig. 3(b)). At the source node, a request for RADcast from the application triggers data fragmentation and appropriate metadata being generated. While the metadata is passed to both the Peddler and the Pryer modules, the fragments are manipulated by the Pryer module. A non-source node which runs a RADcast daemon program learns about the dissemination through hearing Peddler/Pryer messages. It grabs the metadata and stores it in the Metadata Manager, which then communicates with the Peddler and Pryer modules to provide/update dissemination metadata, as well as local paths where received files are stored.

The Peddler module chooses to adopt a lightweight implementation among several options; the goals are (i) to accommodate multiple concurrent walkers, and (ii) to have walkers carry as little state information as needed (field formats are depicted in Fig. 3(c)). Instead of each walker maintaining which nodes it has visited, every node memorizes at what *time-step* (a field of a walker, which initially is 1 and increments by 1 in each move) it was visited by each walker. When a walker arrives at a node, the Walker Handler updates the fields of the walker where necessary, and evaluates termination conditions for it. If termination has not been reached, the Walker Forwarder is triggered, and based on the LRLV strategy, a neighbor is selected to forward the walker to. The realization of the LRLV strategy is embedded in a four-way handshake walker-forwarding scheme (similar to RTS/CTS/DATA/ACK). An advertisement is first sent out, to which neighboring nodes schedule their responses with varying delays. Each node calculates its responding delay based on the time-step at which it was last visited by the walker – as a result, fresh neighbors respond first; among others, those least recently visited respond earlier (a node cancels its scheduled response if hearing others' responses). The walker is then forwarded to the neighbor whose response is received first. The neighbor sends back an acknowledgement after taking over the walker.

In the Pryer module, the Reception Progress Maintainer traces which fragments have been received during a dissemination. Based on that, the Push/Pull Manager controls what fragments to push or pull, schedules fragment rebroadcasting (push) or fragment requests (pull), and cancels unnecessary transmissions whenever possible. For more details about the implementation of RADcast, as well as experiment results from real deployments of RADcast, please refer to [15].

VII. Performance Evaluation

Although we test RADcast in real world settings, the experiments are constrained by the small number of devices we have and the limited topologies we can generate. In order to evaluate the performance of RADcast under a variety of network scales and conditions, we further examine it using simulations.

A. *Experiment Methodology*

We use QualNet v4.0 [19] as the simulation framework. Nodes randomly placed in a rectangular area employ the IEEE802.11b MAC protocol (2Mbps bandwidth) on top of the two-ray propagation path-loss model. UDP is the transport protocol. Table II summarizes the scope and dimensions of our simulation study. We experiment with a wide range of network densities and mobilities in networks of different scales. The default transmission range is selected as such, so as to generate relatively sparse topologies that are connected at least at some time points. Nodes' movements follow the random waypoint mobility model, with zero pause time and the minimum speed kept constant at 1m/s. Each simulation run has one node serve as the source node, and runs sufficiently long for the traffic to completely vanish. Every result reported in this section is averaged over simulation runs with *all* possible source nodes in 10 different topologies (e.g., a result for a 50-node network is an average from 500 simulation runs). While delivery ratio measurements are given a 100% confidence interval, other measurements are given a 95% confidence interval.

Our first set of experiments examine RADcast as a whole, and compare it against other dissemination techniques (the default content size is 32KB). We use the following metrics to capture reliability, timeliness and efficiency respectively: *delivery ratio* (the percentage of nodes that receive the complete data), *latency* (the time elapsed since the start of a dissemination till when all receivers receive the data) and *message overhead* (the ratio of the average number of bytes transmitted per node (both content data and metadata, including RADcast/UDP/IP/MAC headers) to the content data size).

In our second set of experiments, we take a closer look at how Peddler performs as a standalone protocol, as it is the key

TABLE II
EXPERIMENTAL NETWORK CONDITIONS

	Parameter	Values	Default
Network Size	number of nodes @ area	20 @ $(506m)^2$, 30 @ $(620m)^2$, 80 @ $(1012m)^2$	50 @ $(800m)^2$
Network Density	transmission range	140m, 160m, ..., 260m	160m
Node Mobility	maximum speed	0m/s, 5m/s, ..., 30m/s	10m/s (36mph)

to reliability in RADcast. Specifically, we measure its *cover time*, *termination time* and *messages per node* (average number of messages each node transmits) under various conditions.

B. Performance of RADcast

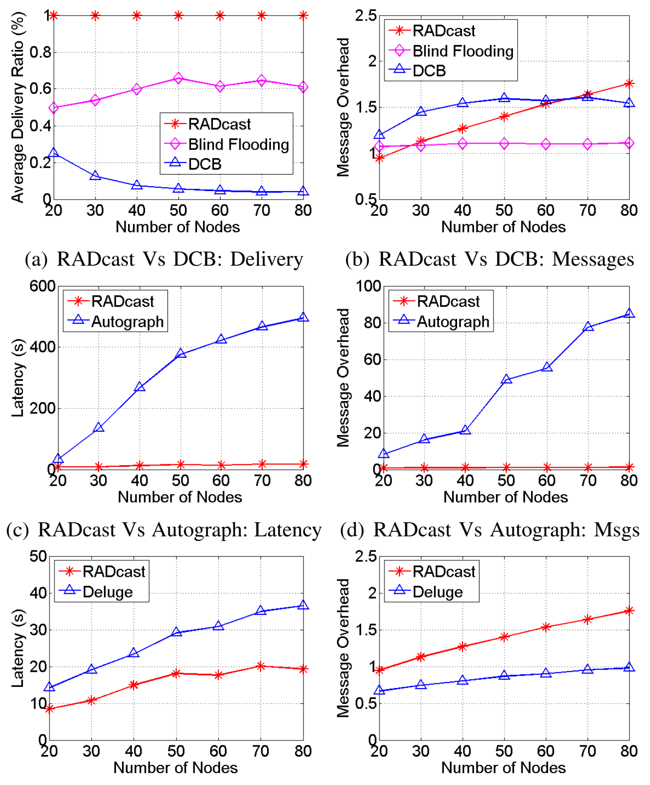
Here we first demonstrate RADcast's superior performance to existing approaches; we explore RADcast's efficiency under varying network conditions and determine its resilience; we then investigate the roles of its components and its dissemination dynamics. By default, one walker is issued (η, the ratio of cover times' upper bound to its lower bound, is 3, derived from experiments). The source push interval is set to 100ms, which yields good performance observed from extensive experiments.

As reliability is our foremost goal, we first confirm the reliability achieved by RADcast. We observe that RADcast (in its max reliability guarantee and lower-bounded reliability guarantee modes) retains 100% delivery ratio at all times. More importantly (not captured in quantitive results), when a node initiates a dissemination using RADcast, it is provided with strong confidence that its desired reliability will be achieved.

Comparisons against Other Approaches: The primary motivation of these comparisons is to examine whether RADcast, and hence its approach to reliable dissemination, can perform comparatively to or even outperform existing approaches (acknowledgement based, continuous push based and continuous pull based). For each of these categories, we select a technique as a "sample point" and test how RADcast compares to it. We pick DCB (Double-Covered Broadcast) [5], Autograph [6] and Deluge [7] as the representatives, respectively.

DCB is a broadcast protocol intended to achieve the balance between reliability and efficiency; it reduces broadcast redundancy through neighbor-designating techniques, and enhances reliability using implicit acknowledgements. Autograph achieves "deterministic" reliability, with the assumption that the network size is known to all. Nodes rotate in periodically retransmitting the data (every 100ms and up to 100 times in our experiments), while advertising their knowledge of who have received the data. Once a propagation route of the data has collected all nodes' "autographs" (last two bytes of link-local IP addresses), nodes deduce dissemination completion and stop retransmitting. Deluge is a reliable large-data dissemination protocol designed for sensor networks. Each node occasionally advertises the latest version of data it has. Upon learning that certain neighbors have newer data, a node sends requests to such a neighbor, which then transmits the requested data.

DCB is originally designed for small data; its performance severely degrades when data size increases [11]. Hence, when comparing RADcast to it, we test DCB as the scheme for spreading data fragments. As DCB offers no reliability guarantee, we measure the highest delivery ratio DCB can achieve given the same amount of time within which RADcast (without network size knowledge) delivers the data to all. We first find

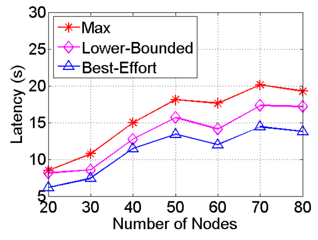


(a) RADcast Vs DCB: Delivery
(b) RADcast Vs DCB: Messages
(c) RADcast Vs Autograph: Latency
(d) RADcast Vs Autograph: Msgs
(e) RADcast Vs Deluge: Latency
(f) RADcast Vs Deluge: Messages

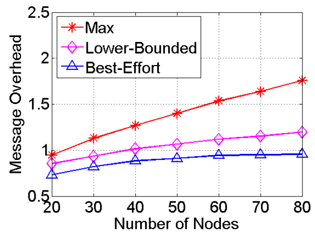
Fig. 4. Comparing RADcast against Other Techniques (32KB Data)

the largest fragment size (f) with which DCB makes 100% delivery in most cases. When DCB deals with data of size d, we set the source push interval to $L/(d/f)$ (L is RADcast's latency in disseminating d data under the same conditions). Meanwhile we measure the performance of Blind Flooding (each node rebroadcasts a fragment upon first receiving it) using the same method to provide a baseline reference. As shown in Fig. 4(a) and 4(b), RADcast significantly outperforms DCB – within the time needed for RADcast to achieve 100% delivery, DCB consumes higher message overhead (the declination in message overhead with increasing network size is due to the drop in delivery ratio), but only a very small portion of recipients receive the complete data. This is mainly because DCB is unable to guarantee the delivery of individual fragments when the network is heavily loaded. Interestingly, even Blind Flooding outperforms DCB due to its inherent redundancy. In comparison, RADcast achieves guaranteed reliability, while paying the cost of slightly higher message overhead.

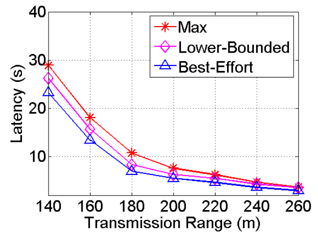
Autograph is only suited for small fragments as well, and may fail to achieve 100% delivery when there is heavy background traffic. Hence, we measure the least time Autograph needs for delivering all fragments to all receivers. We first find the largest fragment size with which Autograph always makes 100% delivery. Then in each dissemination, we use this fragment size and incrementally increase the source push interval; once the delivery ratio reaches 100%, we mark down the latency and message overhead. The average results (compared to RADcast with network size knowledge) are plotted in Fig. 4(c) and 4(d). RADcast is shown to be much more efficient than Autograph. The root cause for this is the substantially lower cost of its push/pull mixed approach. In contrast, in the push approach, nodes in all parts of the network repeatedly



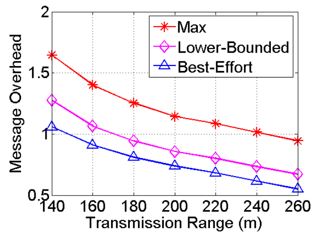
(a) Latency Vs Net Size



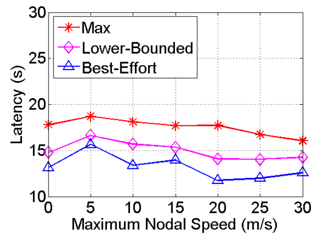
(b) Messages Vs Net Size



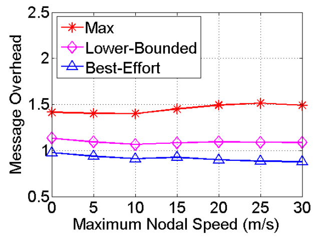
(c) Latency Vs Net Density



(d) Messages Vs Net Density



(e) Latency Vs Mobility



(f) Messages Vs Mobility

Fig. 5. RADcast: Performance under Varying Network Conditions (32KB)

retransmit, attempting to deduce fragment delivery completion; However, the resulting heavy traffic makes completion deduction even harder, and hinders the distribution of other fragments.

To compare RADcast to Deluge, we optimize Deluge and its parameters so that it yields the best performance in our context. All nodes are pre-aware of the dissemination. The data starts propagating when the source node begins advertising the new content. A receiver sends advertisements slowly before it receives the first fragment and after it receives all fragments, but advertises more frequently (every 100ms, same as the source push interval in RADcast) in between. Fig. 4(e) and 4(f) compare RADcast (without network size knowledge) and Deluge. RADcast incurs almost half the latency of Deluge while consuming higher message overhead. Its concurrent push and pull mechanism makes dissemination faster. The extra communication cost is introduced by data push and the walker mechanism, which however eliminates nodes' unnecessary advertisements even when disseminations are infrequent.

Resilience, Heterogeneity and Scalability: Fig. 5 depicts the performance of RADcast at its three reliability modes across varying network conditions. The reliability levels are achieved at different time/message cost, but the differences are insignificant. As network conditions and mobilities change, the latency and message overhead of RADcast demonstrate resilience. They grow proportionally with the increase in data size, and ascend slowly as the network size grows. They decrease when network becomes denser – more neighboring nodes collaborate in retransmitting fragments while suppressing their transmissions to a reasonable extent. To further test RADcast's adaptability to heterogeneity, we set each node's transmission range to a random value between 140m and 260m. The results show that RADcast is resilient to heterogeneity –

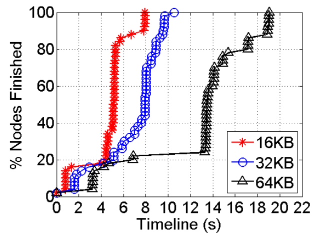


Fig. 6. RADcast: Typical Dissemination Progress along the Timeline

nodes	RW	DF	LRLV
20	119	28	25
30	229	46	41
40	326	64	63
50	398	83	82
60	524	99	111
70	572	118	140
80	655	137	155

Fig. 7. Comparing Cover Moves of LRLV and Other Traversal Strategies

both latency and message overhead are slightly smaller than when devices' transmission ranges are uniformly 200m.

Although RADcast is designed with tens of nodes in mind, we are also interested in seeing how it scales in larger networks, and additionally, with larger data. We examine RADcast handling data of 512KB, 1MB and 2MB in networks of the following dimensions: 100 nodes in $(1131m)^2$ area, 150 nodes in $(1386m)^2$ area, and 200 nodes in $(1600m)^2$ area. We observe that RADcast scales well with network size and data size. Disseminating a 1MB file to 200 nodes overall takes 6 minutes (results are not shown due to space limitation).

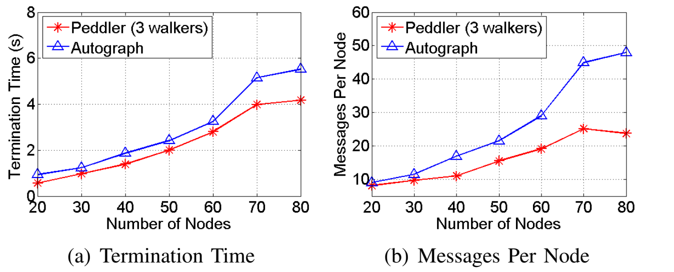
Protocol Specifics: First, to identify the role that Peddler plays in RADcast, we look into the performance of Pryer as a standalone protocol. Although Pryer achieves average delivery ratio over 98% in our experiments, it cannot make guarantees, and even experiences low delivery ratios in a few executions. In situations where (i) network is sparse, (ii) nodes move fast, and/or (iii) data size is small, chances are higher that some receivers miss the metadata and thus Pryer fails to deliver the content data to them. It is evident that the reliability guarantees RADcast offers is made possible by Peddler ensuring metadata receptions. The cost of that is slightly increased latency and higher message overhead (Fig. 5).

To characterize the dissemination behavior of RADcast, we pick a 50-node static topology and watch the propagation of the content. Fig. 6 plots the percentage of nodes that have finished receptions along the timeline in typical runs, and demonstrates an epidemic behavior. That is, a group of nearby receivers finish receptions at the same time; after a while, another group which is farther from the source node finishes. Because data push through scoped flooding is with low redundancy, nodes in the same region tend to miss similar fragments. While a receiver recovers some missing fragments, its neighbors recover them as well since all transmissions are MAC-broadcasts. This then enables farther nodes to retrieve these fragments.

C. Performance of Peddler

In this experiment set, we dig further deeper and explore the performance of Peddler. we first compare Peddler, the LRLV strategy and DCTC(S) with other approaches to exhibit their effectiveness and efficiency. We then test Peddler's behavior in different network situations. Finally we investigate the impact of multiple walkers and network size knowledge. We report that in all our experiment runs, Peddler covers all nodes.

Comparisons against Other Approaches: We begin with examining the efficiency of the LRLV strategy, the engine of Peddler which steers walkers traveling. We compare it against renowned traversal strategies, namely, Random Walk (RW) and Depth-First (DF). It is possible to come up with other strategies bearing similar ideas to LRLV, e.g., Least Recency of First Visit; they however suffer from forming infinite loops, and thus



(a) Termination Time (b) Messages Per Node

Fig. 8. Comparing Peddler and Autograph

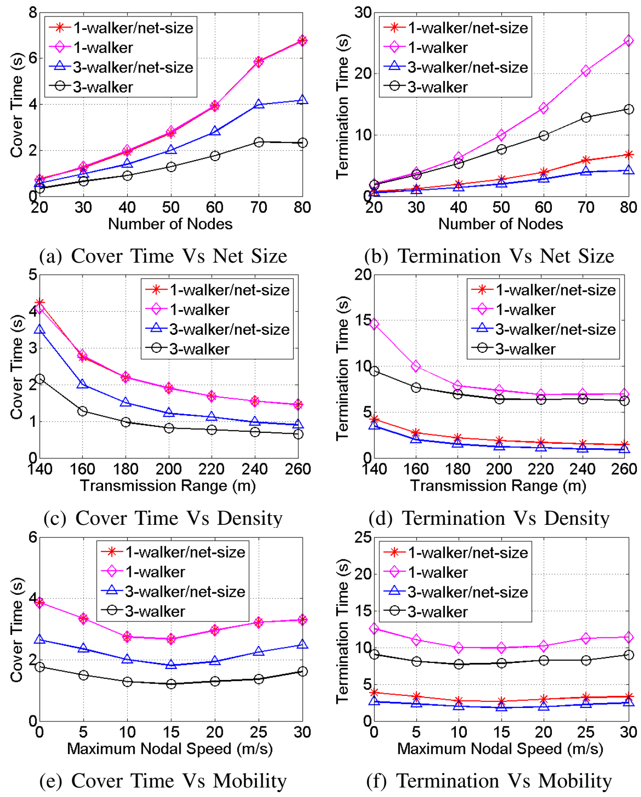
are not considered. Fig. 7 lists the average cover moves of the three traversal strategies in static networks of varying sizes. Although DF takes the least cover moves in some cases, LRLV has a very close performance in all cases. Moreover, unlike DF, LRLV does not require links to be static or symmetric.

DCTC(S) determines walker termination in most situations where network size is unknown. We are aware that there exist other termination conditions which appear more efficient. We make up such a condition and test how DCTC(S) compares to it. Think of the walker (w) as maintaining a visited node set (S_1) and a confirmed node set (S_2). When w visits a node N_i, N_i is added to S_1; it is added to S_2 only if the next node N_i will forward w to is not a fresh node. Termination happens when S_1 becomes and remains equal to S_2. Based on LRLV, this condition guarantees complete traversal in theory. However, in our experiments, overall 17.8% of the executions do not complete traversal. Moreover, this failure rate grows with the increase in network size (35.1% in 80-node networks). Because its aggressive strategy checks most nodes only once, it misses quite a few fresh neighbors. In contrast, DCTC(S) completes traversal in all our experiments. It double-checks all nodes, and thus is more effective in lossy and dynamic networks.

Next we compare Peddler with other awareness assurance techniques. Since metadata typically is small, awareness assurance can also be fulfilled by a broadcast protocol that guarantees delivery of small data. Although such protocols are rare, Autograph well fits in here. We hence test Autograph distributing the same metadata and measure its termination time and message overhead. As Autograph requires network size knowledge, we compare it to Peddler with network size knowledge. Shown in Fig. 8, Peddler incurs slightly shorter termination time, and also generates fewer messages (moreover, an Autograph message is typically larger in size with the "autographs" attached). This evidences that Peddler's network traversal approach not only meets the need of awareness assurance, but is efficient as well. Moreover, as opposed to Autograph, Peddler does not require network size knowledge to operate, which makes it more appealing.

Resilience to Varying Network Conditions: By looking at Peddler's performance across varying network characteristics (Fig. 9), we make the following observations. (i) Peddler has better performance in denser network topologies. Because walkers have higher chances to hop to fresh nodes in each move, cover time and termination time both decrease. Whereas, message overhead does not monotonically decrease because larger number of neighboring nodes communicate in each walker move. (ii) Peddler is resilient to node mobilities. This results from Peddler relying on neither static links nor neighbor knowledge from beaconing protocols.

Protocol Specifics: Finally we explore the impact of Peddler's inputs and parameters.



(a) Cover Time Vs Net Size (b) Termination Vs Net Size
(c) Cover Time Vs Density (d) Termination Vs Density
(e) Cover Time Vs Mobility (f) Termination Vs Mobility

Fig. 9. Peddler: Performance under Varying Network Conditions

Fig. 9 demonstrates the benefit of employing multiple walkers – it reduces cover time and termination time. The scalability of Peddler comes from that – e.g., in a network of 200 nodes, the cover time can be cut down by half (7.72s for 3 walkers, as opposed to 16.70s for 1 walker). The performance gain brought by multiple walkers stems from the concurrency among them, and is irrespective of whether network size is known. However, it could be offset by the extra network traffic it incurs (results not shown here due to space limitation); and this is why the saving on time becomes smaller with increasing number of walkers. Our experiments show that with 5 or more walkers, the cover/termination time stops decreasing but tends to go up; 3 walkers is optimal in balancing time and message efficiency.

Peddler benefits from network size knowledge. As shown in Fig. 9, when network size is given, Peddler's termination time is roughly the same as its cover time; otherwise, it is 2 to 3 times longer, and the message overhead multiplies accordingly (our experiments show that they further increase if nodes can crash). Interestingly, the impact of network size knowledge on cover time also depends on the number of walkers. With single walker, cover time is not affected by network size knowledge. Nonetheless, when multiple walkers are issued, cover time is shorter if network size knowledge is unavailable (Fig. 9(a)). Although surprising, this makes sense because in this case, instead of terminating right after visiting assigned number of nodes, each walker explore as many fresh nodes as possible, which leads to an even higher level of concurrency.

D. Performance Evaluation Summary

RADcast achieves reliable dissemination in a timely and efficient manner with minimum assumptions, which is not achieved by any compared techniques. RADcast persistently achieves guaranteed delivery when integrating Peddler and Pryer. It substantially outperforms acknowledgements- and push-based approaches in all aspects. As compared to pull-based techniques, without assuming receivers' pre-awareness, RADcast cuts down latency by almost 50%. RADcast works well across varying network density and mobility, and scales with both network size and data size. It naturally adapts to node joining, and is resilient to node leaving. As the key to reliability, Peddler completes the assurance awareness task quickly and efficiently in all situations with and without network size knowledge. When exploiting multiple concurrent walkers and network size knowledge, it achieves even better performance.

VIII. Related Work and Concluding Remarks

Since the broadcast storm problem [20] being identified, a large body of research work has been dedicated to reducing redundant transmissions in control data broadcast [21] [22] [23] [24] [2] [1] [25] [26]. Aside from efficiency, the reliability of broadcast has drawn considerable attention as well. Probabilistically reliable broadcast protocols enhance delivery ratio by increasing redundancy [27] or employing acknowledgements [4] [5]. Accomplishing full reliability is hard, and typically involves the deduction of dissemination completion. In Autograph [6], it is done by a propagation route of the data covering all receivers and collecting their "autographs". The authors of [28] propose to deduce successful delivery by message stability.

Multicast protocols would be overkill if used for broadcast – all nodes participate in overlay maintenance, which is expensive if nodes can move arbitrarily. However, the techniques employed for multicast reliability could be useful. For enhancing the reliability of best-effort multicast protocols, gossiping is a commonly adopted idea. In Anonymous Gossip [16], nodes gossip with randomly selected group members to recover lost packets. PIDIS [29] further enhances efficiency by intelligently choosing the best gossiping routes. RDG [17] uses gossiping for both packet delivery and lost packet recovery. Reliable multicast protocols usually utilize acknowledgements to ensure end-to-end delivery. They either require the source to collect ACKs from all receivers and retransmit packets (e.g., RMA [30]), or rely on intermediate nodes on a tree structure to perform local recovery (e.g., FAT [31] and ReAct [32]).

In sensor networks, reliable dissemination of application data has been explored for distributing codes, queries, etc. (e.g., Deluge [7]). The techniques are tailored for specific scenarios, i.e., frequent data update from fixed sinks to stationary sensors that are pre-prepared. An efficient code distribution scheme is to incrementally update the code running on sensors [33]. To provide a reliable dissemination service, Sprinkler [34] embeds a virtual grid over the network and locally computes connected dominating sets. Infuse [35] exploits the reliability offered by TDMA to recover missing data at the MAC layer.

Our work presented in this paper studies providing reliability guarantees for content dissemination in addition to achieving high delivery ratios. Our unique approach neither requires topological knowledge, nor does it need receivers to be pre-aware of disseminations or be stationary. We reduce the problem to a network traversal problem, to which Peddler is developed as a solution. The bulk of the dissemination task is carried out by Pryer, our data diffusion subprotocol. By integrating Peddler and Pryer, the RADcast protocol is able to offer adaptive reliability. Although designed with awareness assurance in mind, Peddler as a standalone protocol could be useful in other applications as well, to name a few, reliable dissemination of small data, network size estimation, etc..

References

[1] W. Peng and X.-C. Lu, "Ahbp: An efficient broadcast protocol for mobile ad hoc networks," *Journal of Computer Science and Technologies*, 2001.

[2] W. Lou and J. Wu, "On reducing broadcast redundancy in ad hoc wireless networks," *IEEE Trans. Mobile Computing*, 2002.

[3] C. E. Perkins and E. M. Royer, "Ad-hoc on-demand distance vector routing," in *IEEE WMCSA*, Feb. 1999, pp. 90 – 100.

[4] S. Alagar, S. Venkatesan, and J. Cleveland, "Reliable broadcast in mobile wirless networks," in *MILCOM*, Nov. 1995, pp. 236 – 240.

[5] W. Lou and J. Wu, "Double-covered broadcast (dcb): A simple reliable broadcast algorithm in manets," in *INFOCOM*, vol. 3, Mar. 2004.

[6] E. Vollset and P. Ezhilchelvan, "An efficient reliable broadcast protocol for mobile ad-hoc networks, Tech. Rep. CS-TR-822, 2003.

[7] J. W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *SenSys*, 2004.

[8] Z. Genc and O. Ozkasap, "Eramobile: Epidemic-based reliable and adaptive multicast for manets," in *WCNC*, 2007.

[9] A. M. Farley, "Broadcast time in communication networks," *SIAM Journal on Applied Mathematics*, vol. 39, 1980.

[10] S. Khuller and Y.-A. Kim, "On broadcasting in heterogeneous networks," *ACM-SIAM. Symposium on Discrete algorithms*, 2004.

[11] B. Xing, M. Deshpande, N. Venkatasubramanian, and S. Mehrotra, "Towards reliable application data broadcast in wireless ad hoc networks," in *WCNC*, 2007.

[12] S. Dolev, E. Schiller, and J. Welch, "Random walk for self-stabilizing group communication in ad hoc networks," *IEEE Trans on Mobile Computing*, 2006.

[13] S. R. D. Swapnil Patil and A. Nasipuri, "Serial data fusion using space-filling curves in wireless sensor networks," in *SECON*, 2004.

[14] N. Malpani, Y. Chen, N. H. Vaidya, and J. L. Welch, "Distributed token circulation in mobile ad hoc networks," *IEEE Trans on Mobile Computing*, 2005.

[15] B. Xing, S. Mehrotra, and N. Venkatasubramanian, "Radcast: Enabling reliable content dissemination in ad hoc networks," Tech. Rep., Aug 2008.

[16] R. Chandra, V. Ramasubramanian, and K. P. Birman, "Anonymous gossip: Improving multicast reliability in mobile ad-hoc networks," in *ICDCS*, 2001.

[17] J. Luo, P. T. Eugster, and J.-P. Hubaux, "Route driven gossip: Probabilistic reliable multicast in ad hoc networks," in *INFOCOM*, 2003.

[18] "Nokia internet tablets," `http://tableteer.nokia.com`.

[19] "Qualnet 4.0," `http://www.scalable-networks.com`.

[20] S.-Y. Ni, Y.-C. T. Y.-S. Chen, and J.-P. Sheu, "The broadcast storm problem in a mobile ad hoc network," in *Mobicom*, 1999.

[21] H. Lim and C. Kim, "Multicast tree construction and flooding in wireless ad hoc networks," in *ACM MSWIM*, 2000.

[22] W. Peng and X.-C. Lu, "On the reduction of broadcast redundancy in mobile ad hoc networks," in *IEEE Mobihoc*, 2000.

[23] J. Sucec and I. Marsic, "An efficient distributed network-wide broadcast algorithm for mobile ad hoc networks, Tech. Rep. 248, Sept. 2000.

[24] J. Wu and F. Dai, "Broadcasting in ad hoc networks based on self-pruning," in *INFOCOM*, vol. 3, Mar. 2003.

[25] ——, "A generic distributed broadcast scheme in ad hoc wireless networks," in *IEEE ICDCS*, May 2003.

[26] E. Pagani and G. P. Rossi, "Reliable broadcast in mobile multihop packet networks," in *ACM MOBICOM*, Sept. 1997.

[27] P. Rogers and N. Abu-Ghazaleh, "Towards reliable network wide broadcast in mobile ad hoc networks, Tech. Rep. tr-cs-02-04, Dec. 2004.

[28] K. Singh, A. Nedos, G. Gaertner, and S. Clarke, "Message stability and reliable broadcasts in mobile ad-hoc networks," in *ADHOC-NOW*, 2005.

[29] C.-C. Shen and S. Rajagopalan, "Protocol-independent packet delivery improvement service for mobile ad hoc networks," in *MASS*, 2004.

[30] T. Gopalsamy, M. Singhal, D. Panda, and P. Sadayappan, "A reliable multicast algorithm for mobile ad hoc networks," in *ICDCS*, 2002.

[31] W. Liao and M.-Y. Jiang, "Family ack tree (fat) supporting reliable multicast in mobile ad hoc networks," in *ICC*, 2002.

[32] V. Rajendran, Y. Yi, K. Obraczka, S.-J. Lee, K. Tang, and M. Gerla, "Reliable, adaptive, congestion-controlled adhoc multicast transport protocol," UCSC, Tech. Rep., 2003.

[33] N. Reijers and K. Langendoen, "Efficient code distribution in wireless sensor networks," in *WSNA*, 2003.

[34] V. Naik, A. Arora, and P. Sinha, "Sprinkler: A reliable and energy efficient data dissemination service for wireless embedded devices," in *RTSS*, 2005.

[35] S. S. Kulkarni and M. Arumugam, "Infuse: A tdma based data dissemination protocol for sensor networks," in *SenSys*, 2004.