

Where-Fi: A Dynamic Energy-Efficient Multimedia Distribution Framework for MANETs

Shivajit Mohapatra, Bogdan Carbutar, Michael Pearce, Rohit Chaudhri & Venu Vasudevan

Applications Research Center

Motorola Labs, Schaumburg, IL 60196

{mopy, carbutar, michael.pearce, rohit.chaudhri, venuv}@labs.mot.com

Abstract

Next generation mobile ad-hoc applications will revolve around users' need for sharing content/presence information with co-located devices. However, keeping such information fresh requires frequent meta-data exchanges, which could result in significant energy overheads. To address this issue, we propose distributed algorithms for energy efficient dissemination of presence and content usage information between nodes in mobile ad-hoc networks. First, we introduce a content dissemination protocol (called CPMP) for effectively distributing frequent small meta-data updates between co-located devices using multicast. We then develop two distributed algorithms that use the CPMP protocol to achieve "phase locked" wake up cycles for all the participating nodes in the network. The first algorithm is designed for fully-connected networks and then extended in the second to handle hidden terminals. The "phased locked" schedules are then exploited to adaptively transition the network interface to a deep sleep state for energy savings. We present two novel applications (called "Zeitgeist" & "MeCast") developed using our protocol that present compelling "social experiences" for users. We have implemented a prototype system (called "Where-Fi") on several Motorola Linux-based cell phone models. Our experimental results show that for all network topologies our algorithms were able to achieve "phase locking" between nodes even in the presence of hidden terminals. Moreover, we achieved battery lifetime extensions of as much as 28% for fully-connected networks and about 20% for partially-connected networks.

I. INTRODUCTION

Recent years have witnessed the emergence of significant entertainment markets around peer-to-peer (p2p) content sharing technologies [2]. These enabling technologies allow their users to search, locate, download and share content with friends and family, or even with loosely affiliated communities. Such user-to-user forms of sharing have addressed the commercially important issue of helping content owners market content directly to consumers [1]. With the global penetration of mobile phones that have rich media and wireless networking capabilities, we are beginning to see the above trend migrate from PCs to mobile handsets, and from wide-area communities of users to dense local-area social situations (e.g. coffee shops, train stations etc.) [27], [3], [9]. This shift presents an opportunity for designing proximity-aware systems that deliver novel "social experiences" by allowing users to not only discover and access "current" content being consumed in their immediate proximity but also build statistics of popular content around them in a purely opportunistic fashion. However, such systems must first address the difficult research challenge of providing scalable, energy efficient presence and content distribution between co-located devices. Moreover, to keep presence information fresh, the distribution mechanisms have to focus on frequent, unsynchronized small meta-data updates rather than large infrequent payloads which makes energy optimization difficult.

A popular asynchronous communication paradigm for capturing such updates is publish/subscribe [10], [33], [21]. In such systems, subscribers can express their interest in certain kinds of content to other devices (publishers) which are explicitly discovered in an earlier step. Subsequently when content is accessed on a device, a notification is sent to all devices whose registered interests are compatible with the item. For example, UPnP (Universal Plug and Play) [4] is a popular industry wide standard that uses the pub/sub paradigm to allow devices to dynamically advertise, discover, search and control services. While simple and elegant in the sense that it supports decoupling of publishers and subscribers, this approach has several limitations especially in a highly dynamic mobile environment. To receive asynchronous notifications, subscribers are required to permanently monitor their network interface. The energy inefficiency in having to keep network interfaces in always-on duty cycles would significantly reduce device lifetimes. Secondly, the sudden disconnection of devices imposes significant delay and communication overheads in order to keep the network topology up to date.

In this paper, we address the aforementioned challenge of energy efficient dissemination of presence and content usage information between nodes in mobile ad-hoc networks. First, we introduce a dissemination protocol called CPMP (Content Presence Multicast Protocol) that can be used on mobile systems for effectively disseminating presence updates to neighboring nodes. We then present two fully distributed algorithms that use the CPMP protocol to achieve a "scheduled rendezvous" type synchronization between nodes in the ad-hoc network. With such synchronization, we manage to get all nodes within a one-hop distance to "phase lock" their wake up schedules to receive and send updates. Such phase locking allows the switching of the wireless network interface to a "deep sleep" mode thereby saving battery energy. We elaborate on this later in the paper. Our first algorithm addresses a fully connected one-hop ad-hoc network where each node can hear every other node. In the second algorithm, we extend the first algorithm to handle hidden terminals. Note that we do not consider multi-hop networks (as in intermediate nodes do not route packets), but handle the case where a single node is a part of one or more networks. Specifically in our solution we do not exchange meta-data between nodes that are not within transmission range of each other. This assumption makes the problem of energy-efficient dissemination more tractable and obviates the need for handling obsolete/irrelevant context information or unprecedented delays associated with multi-hop ad-hoc networks.

Scheduled rendezvous wake-up based protocols [13], [25], [8] have been widely researched and are the simplest in theory to implement. These protocols require that all nodes synchronize their sleep/wake-up schedules to meet the potential demand for communication. The biggest advantage of this approach is that when a node is synchronized and awake, it is guaranteed that all other nodes are awake and listening, thereby ensuring that network broadcasts/multicasts and most underlying traditional routing functionality will work. The problem however is that these solutions assume strict clock synchronization among nodes. Clock synchronization [15] is a complex communication intensive operation, with overheads that might not be acceptable in highly dynamic environments. In our solution, we utilize relative times to synchronize transmissions thereby avoiding clock synchronization overheads. Furthermore we also do not require nodes to perform neighbor discovery. Errors in either clock synchronization or neighbor discovery can lead to network partitions which are hard to unify and re-synchronize. However, we assume that nodes know how often they want to exchange data (and are flexible about it) and exploit this information to achieve synchronized wake-up between nodes.

The primary contributions of this paper are as follows: (i) We introduce a presence and media usage distribution protocol (called CPMP) for effectively exchanging exchanging media usage meta-data between groups of collocated mobile devices (ii) Two decentralized, power aware algorithms for synchronizing the transmission of updates to collocated devices are proposed.(iii) A detailed evaluation of both the mechanisms is presented. (iv) Finally, we present two novel prototype applications (called “MeCast” and “MyZeitgeist”) that have been designed and implemented over the CPMP protocol and our proposed algorithms on Motorola’s Linux-based cell phones. We use the term “Where-Fi” to include the power aware CPMP module and the aforementioned applications along with the other components in our system.

II. SYSTEM OVERVIEW

The “Where-Fi” system architecture is shown in Figure 1 with the contributions of this work highlighted in grey. The power aware CPMP (PAC) module is implemented in middleware and is the primary component of the system. Both the CPMP protocol and our algorithms are implemented within this module. The middleware exports APIs that are used by applications using the CPMP protocol. Note that “CPMP” applications could refer to a class of applications that require dissemination of periodic updates to co-located devices (e.g. presence/context information, content consumption information, heartbeats, beacons etc.). Figure 1 shows two such applications that are presented in section IV. It is desirable that such applications are able to disseminate updates reliably to all neighboring devices with low energy overhead. The CPMP protocol makes provision for such optimization by requiring applications to include their relative next transmission time in each packet. Our algorithms (presented in Sec III) utilize these “transmission times” to achieve scheduled rendezvous type synchronization for all one-hop neighbors in the system.

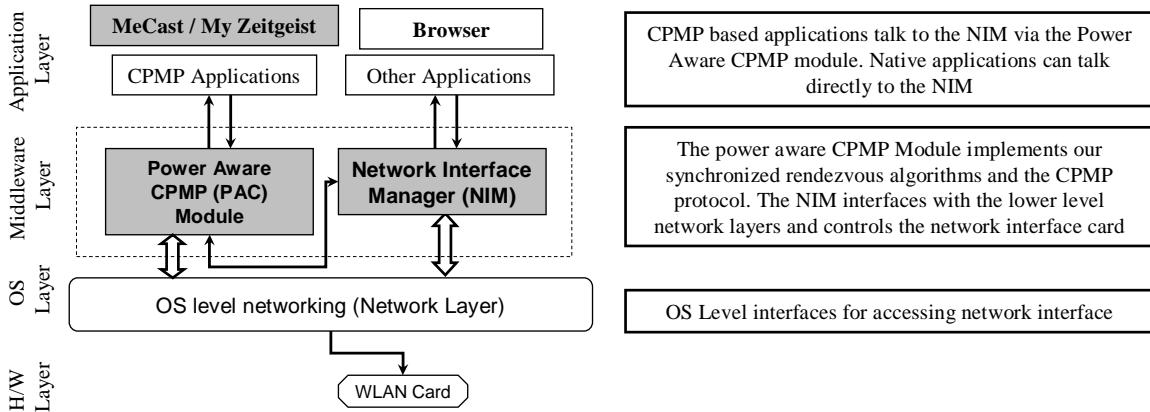


Fig. 1. Where-Fi System Architecture on a mobile device

The PAC module interfaces with the network interface manager (NIM) module which is a middleware abstraction used to control access to the wireless network interface hardware using OS layer API interfaces. The PAC module calculates and communicates the next expected transmission time for the “CPMP” based applications. Other native applications (e.g web browser, media player etc.) could directly input their network usage requirements to the NIM module. The NIM module then uses these inputs to determine a sleep/active schedule for the wireless interface card. Note that in our current approach a simplistic algorithm is employed by the NIM in case applications are unable to determine their usage requirements (e.g. browser). While it is relatively easy to optimize energy consumption (by switching network interface to a low duty cycle) on individual devices for CPMP applications (periodic), it is much harder in a highly dynamic and distributed setting with nodes constantly entering and leaving the system. The difficulty here is to achieve synchronized wakeup schedules for all devices (without synchronizing clocks or performing neighbor discovery) while still ensuring reliable delivery of updates.

Figure 2 illustrates the network connectivity models we consider in this paper. We also assume that the devices shown in the figure have networking capabilities and run the Where-Fi middleware. Figure 2(a) shows several heterogeneous devices

in a fully connected ad-hoc network (e.g. Starbucks coffee shop). In such a network each device is within one-hop range of every other device and can directly listen in to all their multicast/broadcast packets. The fact that all nodes are in range of each other makes it easier to “phase lock” their transmission times.

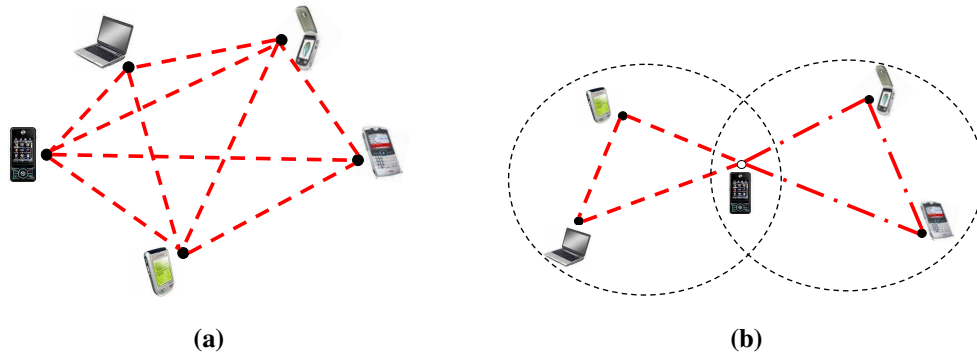


Fig. 2. Network Connectivity Model (a) Fully Connected One Hop Ad-hoc Network (b) Multiple separate One-Hop Ad-hoc Networks with hidden terminals

Figure 2(b) shows a more generalized case where not all nodes are within transmission range of each other (e.g. grand central station, train compartments, soccer stadium etc.). Such networks are generally susceptible to the “hidden terminal” effect as illustrated in the figure. The node in the center is in range of all the nodes. However, the nodes on the left and right are out of range of each other and therefore cannot communicate directly. In such a scenario, the node at the center needs to synchronize with both of the one-hop networks in order not to lose updates from either network. However, this incurs an additional cost of keeping the network interface (of the center node) in a high duty cycle for a longer duration resulting in lower energy savings. In this work, we do not consider multi-hop ad-hoc networks where intermediate nodes route packets to nodes outside the one-hop radio range of each other. In the following section, we present an overview of the CPMP (Content Presence Multicast Protocol) protocol which was designed to specifically distribute multimedia content consumption updates to co-located devices. However, the protocol can be easily extended to all types of presence/context information. Note that even though we call it a multicast protocol it also works in a broadcast environment.

A. CPMP Overview

We designed the Content Presence Multicast Protocol (CPMP) to support social content consumption experiences. It is intended to be deployed in ad-hoc wireless local area network environments. CPMP provides a framework for periodically communicating information about what content is currently being consumed and what content is being sought for future consumption at each participating node. Our goal was to make the protocol efficient and scalable while including features intended to support synchronization of presence message transmissions. CPMP messages are transmitted periodically to inform nearby devices of updated content presence information using IP multicast. Messages also include an HTTP port number on which (optionally) a host may be listening for HTTP requests to support related services (such as content transfer). Fig 3 shows the details of the messaging formats used in the protocol. CPMP messages are transmitted via multicast UDP/IP frames. Each message is encapsulated in a single UDP/IP message and transmitted to the CPMP multicast address. In order to avoid IP fragmentation, the protocol requires that implementations limit message sizes to the link layer maximum transmission unit.

Field	Type	Len	Val	Description
Version	INT	1	0x01	Protocol Version 1
MSG_ID	INT	2		Message Identifier
SENDER	STR	Var		Sending user or device name
HTTP_PORT	INT	2		HTTP services TCP port
T_X	INT	2		Time to next transmission (sec)
NumNow Playing	INT	1		# of now playing items
Now Playing	CI	Var		Now playing items
NumWanted	INT	1		# of wanted items
numNowWanted	CI	Var		Wanted Content items

(a)

Field	Type	Len	Val	Description
Content_Class	INT	1	0x01	Content Class (0x01-0x04)
Content_TYPE	STR	Var		MIME type of Content
NUM_GENRE	INT	1		# of GENRE strings
GENRE	STR			Genre of Content (repeated)
Class specific data		Var		Content Class specific fields

(b)

Fig. 3. CPMP Message Framing (a) CPMP Content Presence Messages (b) Content Item (CI) Records. Integer data is encoded in big-endian variable length fields and Integer field lengths must be 1, 2, 4, or 8 bytes long. String data is encoded using a 2-byte big-endian length prefix followed by the specified number of bytes of UTF8-encoded character data. Class specific data refers to data specific to a particular content type (e.g. ID3 tags for music).

Each CPMP message includes a field (T_X) that specifies the number of seconds in which to expect a new CPMP message from that node. Note that this does not include any redundant transmissions of the current message - if retransmission is

used to improve reliability, each retransmission must update the T_X field so that it accurately reflects the delay until the new message will be transmitted. By transmitting CPMP messages at approximately the same time CPMP messages are expected, an implementation can avoid powering on the wireless LAN radio for transmissions - essentially piggy-backing CPMP transmission with reception. Ideally, all participating nodes will use this technique simultaneously, resulting in synchronization of CPMP activity.

B. Problem Statement

In this work we attempt to solve the following problem. Given a set of nodes participating in an ad-hoc network and using a multicast presence distribution protocol (e.g. CPMP) for exchanging updates, design an algorithm to achieve a scheduled rendezvous type synchronization between all nodes (i.e. synchronized transmissions), considering nodes have arbitrary arrival/exit times, unsynchronized clocks and flexible T_X s. The challenge is how to design such an algorithm for an ad-hoc network where

- 1) all nodes are within radio range of each other (e.g. Fig. 2a)
- 2) where not all nodes are within radio range of each other (e.g. Fig. 2b).

C. Problem Characterization

Depending on the desired behavior for nodes participating in the ad-hoc ecosystem, various power aware CPMP algorithms can be designed. We assume the following behavior for nodes in our system: initially, each node keeps its network interface active for an interval of length t_a , followed by a sleep interval of length t_s (where $t_s = (R \times t_a)$; $R = 1, 2, 3, \dots$). Nodes hear CPMP multicasts made during the interval t_a and miss them when the network interface is transitioned to sleep mode. Each node wakes up at every interval ($= T_X$) and multicasts its own CPMP packet. This behavior is repeated for the entire life of the node. If a node is the only node in the system, this behavior saves energy. However, to prevent nodes from entering a cycle where they are perpetually out-of-sync with other nodes, we adjust the length of t_a , such that $t_a \geq T_X$. This ensures that when a node enters an active interval, it is guaranteed to hear transmissions from all nodes within radio range. The tradeoff here is that a higher $R = \frac{t_s}{t_a}$ value will save more power but it will incur the penalty of missing updates as well as increase the time to discover new neighbors. We now introduce a definition for node synchronization, followed by a definition for network stability. They will be used in Section III.

Definition II.1. (Synchronization) *Two nodes are said to be synchronized when their transmission times coincide. A node that is synchronized with at least one other node is said to be synchronized. A packet received by a synchronized node during a sleep interval is said to be out-of-sync.*

Definition II.2. (Stability) *A node is said to be stable if during an active or sleep interval it knows when to expect CPMP transmissions from all its neighbors. A stable network is a network where all nodes are stable.*

III. PAC ALGORITHMS

We propose two algorithms for achieving “phased locked” wake up times among nodes of a network. Our first algorithm, FC_PAC (fully-connected PAC) assumes a fully connected network where each multicast is heard by every node in the system. For such networks, FC_PAC effectively synchronizes the wake up schedules of *all* the nodes in the system. The second algorithm, PC_PAC (partially-connected PAC), extends FC_PAC to allow for a looser synchronization in larger scale networks, where hidden terminals may be present. PC_PAC does not “phase lock” the entire network; instead it updates the wake up schedule of each node to account for all neighboring CPMP transmissions.

Algorithm Overview: The general behavior of the PAC algorithms is as follows: Initially, nodes are assumed to be “not synchronized”. During listen intervals, each node listens to receive CPMP packets from other nodes in its vicinity. When not synchronized, a node will synchronize with the node(s) from which it receives the first CPMP packet(s). In a fully-connected network, where all nodes receive all transmitted packets, the procedure called FC_PAC (Algorithm 1) is sufficient to synchronize all the nodes in the network. However, a node A (say) may receive out-of-sync packets. These packets can be either sent by newly joining nodes that have missed previous CPMP packets, or by nodes that are out of range of other CPMP transmissions received by A (hidden terminals).

PC_PAC extends FC_PAC to handle a more generic case where each node cannot assume that transmissions it is hearing are also being heard by every other node (i.e. hidden terminals are present). Instead of “de-synchronizing” the receiver A , PC_PAC handles a node join by allowing the sender, B , of the out-of-sync packet to synchronize with A . This is done by making A wait for a full active plus sleep cycle, during which an un-synchronized B would become synchronized with A . If at the end of this cycle A receives another out-of-sync CPMP packet from B , it has clearly detected a hidden terminal. That is, B is also synchronized with other nodes and will not “de-synchronize” with them in order to synchronize with A . In such a case, A now has to adjust its wakeup schedule to account for the hidden terminal B . PC_PAC (Algorithm 2) handles hidden terminals by updating the wake up schedule of A (and B) to include the next out-of-sync transmission from B (A).

Algorithm 1 FC_PAC algorithm for synchronizing all the nodes fully connected ad-hoc network

```

1. Object implementation PAC;
2. inQ : InputQueue;           #packet recv queue
3. pktList : Pkt[];           #list of packets
4.  $T_X$  : int;                 #delta time to next transmission
5. nextSendCPMP : int;        #next time to send CPMPupdate
6.  $t_{curr}$  : int;             #current time
7. sync : bool;               #sync variable

8. Operation main()
9.   while (true) do
10.    pktList := inQ.getAllPackets();
11.     $t_{curr}$  := getCurrentTime();
12.    if (sync = false) then
13.      setTX( $t_{curr}$ , pktList); fi
14.   od
15. Operation setTX( $t_{curr}$  : int, pktList : Pkt[])
16.  $tx_{min}$  := getMinTX(pktList);
17. if ( $tx_{min} + t_{curr} = nextSendCPMP$ ) then
18.    $T_X := tx_{min}$ ;
19.   sync := true;
20. else if ( $tx_{min} + t_{curr} > nextSendCPMP$ ) then
21.    $T_X := \min(t_{curr} + tx_{min} - nextSendCPMP, T_X)$ ;
22.   sync := true;
23. fi
24. end

```

A. FC_PAC

Algorithm 1 illustrates the distributed algorithm (FC_PAC) modeling the behavior of an unsynchronized node (say A) in a fully connected ad-hoc network. FC_PAC achieves "phased lock" synchronization of all participating nodes in that network. The `main` procedure contains the infinite loop processing incoming packets and the `setTX` procedure handles packets received when the node is not yet synchronized. In Algorithm 1, the `nextSendCPMP` variable denotes the absolute time at which node A will send its next CPMP packet. While not shown in the algorithm, its value is incremented along with T_X immediately after A sends a CPMP packet (in a separate "send" thread). The `sync` variable (initialized to "false") denotes whether A has already synchronized with other nodes. By default, A is blocked on a receive system call, waiting for CPMP packets (line 10). It then retrieves the packet whose advertised T_X value is the smallest among the received packets. Let us denote the sender of that packet as B . The FC_PAC algorithm works as follows: If the next transmission time of B coincides with the next transmission of A (line 17), A synchronizes with B (i.e. A adjusts its T_X such that its next transmission is phase locked with B 's. A uses the T_X value from B 's CPMP packet to achieve this). If the next transmission of B will occur later than that of A (line 20), A sets its T_X value to the difference between the two transmission times, but only if the difference is smaller than its current T_X value. This is because at time `nextSendCPMP`, A will send a CPMP packet showing that it will synchronize with B after an interval of length $t_{curr} + tx_{min} - nextSendCPMP$. If this difference is larger than A 's T_X value, A uses the same T_X during its next CPMP transmission.

Theorem III.1. *In a fully-connected network where all the nodes have the same t_a and t_s values and where the latest node joins at time T , all nodes are synchronized by time $T + (t_a + t_s)$.*

Proof:

Let N be the number of nodes in the network. W.l.o.g we assume that nodes n_0, \dots, n_{N-1} join in this order. Using induction, we show that the network synchronizes. The base case is when the first two nodes n_0 and n_1 join and n_1 synchronizes with the CPMP transmission of n_0 . The induction step is when nodes n_0, \dots, n_{i-1} are synchronized and node n_i joins the network. In the worst case node n_i will have to wait for a full cycle ($t_a + t_s$) before becoming synchronized with the network. This is because it may have to wait for a full sleep interval, t_s followed by an entire active interval, t_a before receiving a CPMP packet from one of the other i nodes. The CPMP packet sent by n_i after its initial listen interval will be ignored by the already synchronized network. Note that if multiple nodes (n_i, \dots, n_{i+k}) join at similar times, they will *all* receive CPMP packets from n_0, \dots, n_{i-1} before any of n_{i+1}, \dots, n_{i+k} has time to send a first CPMP packet. ■

B. PC_PAC

The simplicity of the FC_PAC algorithm stems from the assumption of the existence of a broadcast channel (fully connected network). This is a reasonable assumption for networks formed in small closed environments (e.g. Starbucks shop). However in more generic larger scale networks (e.g. train compartment), hidden terminals may be present. In such environments, nodes may synchronize only with a subset of their neighbors, effectively missing updates sent by the rest of their neighbors (Figure 2(b)). In this section we propose PC_PAC, an extension to the FC_PAC algorithm, that solves the "phase lock" synchronization problem for networks with hidden terminals. Specifically, the PC_PAC algorithm 2 extends the first algorithm by relaxing the assumption

Algorithm 2 PC_PAC: extension to the FC_PAC algorithm to handle out-of-sync packets due to hidden terminals. Nodes loosely synchronize only with their neighbors.

```

1. Object implementation PAC;
2. activeNetwork : Network;      #the neighbor(s) transmitting next
3. OOSNode : Network[];         #list of out - of - sync neighbors
4. NW : Network[];              #list of sync neighbors
5. nextWakeUp : int;            #time when activeNetwork transmits
6. OOS : bool;                  #out - of - sync variable
7. OOSNodeSize : int;          #size of OOSNode

8. Operation main()
9.   while (true) do
10.    pktList := inQ.getAllPackets();
11.    tcurr := getCurrentTime();
12.    if (sync = false) then setTX(tcurr, pktList);
13.    else setWakeUp(tcurr, pktList); fi
14.  od
15. end

16. Operation setWakeUp(tcurr : int, pktList : Pkt[])
17. if (tcurr = nextWakeUp or nextSendCPMP) then
18.   handleActiveNetwork(tcurr, pktList);
19.   getNextActiveNetwork(tcurr);
20.   return;
21. fi
22. for each pkt in pktList do
23.   OOSNodeSize := OOSNode.size();
24.   if (tcurr < nextSendCPMP & OOS = false
25.    & contains(OOSNode, NW, pkt) = false) then
26.     OOS := true;
27.     Network nw := new Network(pkt.nodeId);
28.     nw.phase := tcurr - nextSendCPMP;
29.     OOSNode.add(nw); fi
30.   if (OOSNodeSize = OOSNode.size()
31.    & OOS = true) then
32.     NW.add(newNetwork(pkt));
33.     OOSNode.remove(pkt); fi
34.   if (OOSNode.isEmpty() = true) then
35.     OOS := false; fi
36. od
37.   getNextActiveNetwork(tcurr);
38. end

37. Operation getNextActiveNetwork(tcurr : int)
38. if (NW.size() > 1) then
39.   Network nw := getMinPhase(NW);
40.   if (tcurr < nextSendCPMP + nw.phase
41.    & nextSendCPMP + nw.phase < nextSendCPMP) then
42.     nextWakeUp := nextSendCPMP + nw.phase;
43.     activeNetwork := nw;
44.   fi endif end

46. Operation handleActiveNetwork(tcurr, pktList)
47. if (pktList.size() > 0) then
48.   activeNetwork.setCount(0);
49.   removeSyncNodes(OOSNode, pktList);
50. else
51.   activeNetwork.incCount();
52.   if (activeNetwork.count = THRESHOLD) then
53.     if (NW.isEmpty()) then sync := false;
54.     else activeNetwork := getNextNetwork(NW); fi
55.   fi
56. fi
57. end

```

that multicast packets are heard by all participating nodes in the network. We introduce a procedure called `setWakeUp` to implement the behavior exhibited when out-of-sync packets are received by a node. Note that a node can receive a out-of-sync packet if it has already phase locked itself with a set of nodes and is in its active period t_a . The `getNextActiveNetwork` and `handleActiveNetwork` procedures are called by `setWakeUp` in order to first determine the set of neighbors that will send a CPMP update next and second, to update the status of those neighbors when their expected CPMP transmissions occur or fail to take place.

We illustrate the behavior of the procedure for a node A . The procedure `setWakeUp` is only called when the value of the `sync` variable is true, that is, A has already synchronized with a subset of its neighbors (line 13). The purpose of this procedure is for A to update the value of its `nextWakeUp` variable, denoting the next time (in absolute value based on A 's internal clock) when A has to wake up to receive the transmission of CPMP packets by a subset of its neighbors. The procedure achieves this through the maintenance of a list of neighboring networks, `NW`, that are not synchronized with each other (hidden). The networks in `NW` represent disjoint subsets of A 's neighbors, where all the nodes in a network in `NW` transmit CPMP packets at the same time. Since each of the networks in `NW` has a different transmission time, at each time A also maintains an `activeNetwork` variable, denoting the network in `NW` that will transmit next.

Similar to the `setTX` procedure, `setWakeUp` is called when packets are received by the network interface. If the time when `setWakeUp` is called is different from the time when node A has to transmit a CPMP packet and when A expects a CPMP transmission (line 22), it means A has received out-of-sync packet(s). To handle these packets, A maintains an array of out-of-sync packets, `OOSNode`. The reason for the `OOSNode` array is to filter out-of-sync packets sent by newly joining neighbors, from out-of-sync packets sent by neighbors that are already synchronized with some of their other neighbors.

We use B to denote the sender of one of the out-of-sync packets. If this is the first time A receives an out-of-sync packet from B , it generates a new network in `OOSNode` and adds B to it (lines 25-28). Each such network has a `phase` field, that denotes the time interval between the `nextSendCPMP` value of A and of B (line 27). If this is the second time A receives an out-of-sync packet from B (line 29), it means that B is already synchronized with other nodes and will not synchronize with A . Then, A adds B (and any other nodes that have sent packets at the same time with it) to the `NW` list and removes B from the `OOSNode` list (lines 30,31). That is, from then on A has to monitor and wake up for the next CPMP transmission of B . If the time when `setWakeUp` is called coincides with the time when A has to transmit a CPMP packet or when A expects a CPMP transmission (line 17), A removes any of the nodes from `OOSNode` that have synchronized with it (line 49). That is, if for instance B was not synchronized when it sent the first out-of-sync packet, by its next CPMP transmission it would have received a CPMP packet from A , effectively updating its T_X value to synchronize with it. Then, its next transmission would take place at the same time with A , at which point A considers B to be synchronized with it. Then, A resets the counter of the active network (line 48). Finally, A sets its wake up time to coincide with the transmission time of the next active network. This is done in the procedure `getNextActiveNetwork` (lines 37-45).

Node failures or leaves: The counter of networks from the `NW` list of A is used to detect nodes that became disconnected, due to mobility or failures. That is, if A misses more than a `THRESHOLD` number of CPMP packets from one of the networks in `NW`, it removes the network from `NW` (lines 52-55). As explained above, the on-time receipt of a packet from the current active network prompts A to set the network's counter to zero (line 48).

Summary: To summarize the behavior this algorithm we refer the reader to Figure 2(b). When using the `PC_PAC` algorithm, the node at the center syncs with the two sets of hidden terminals and wakes up its network interface more frequently to receive their packets. Consequently, it will have lower energy savings but will not miss updates from either network. We can now prove the following result.

Theorem III.2. *In a network where the latest node joins at time T , the network reaches stable state by time $T + 2(t_s + t_a)$.*

Proof: (Sketch)

To prove that the network reaches stable state it suffices to show that any node reaches stable state. Let A be such a node. If A is the first node to join among its neighbors, it is easy to see that using `FC_PAC` and `PC_PAC` it will reach stable state. If one of its neighbors joins at time T , in the worst case A will receive its first out-of-sync packet after $t_s + t_a$ seconds and the second out-of-sync packet after another $t_s + t_a$ seconds.

If A is not the first to join among its neighbors, it will first synchronize with the sender of the first CPMP packet it receives. Then, symmetric to the case studied above, A will handle out-of-sync packets, setting wake up times for all its neighbors. If A joins at time T , it will receive all the CPMP packets of its neighbors during its first active interval. In the worst case, that takes $t_s + t_a$ seconds. A second round of CPMP packets, needed to handle the out-of-sync packets will then come after another $t_s + t_a$ seconds. It is important to note that two out-of-sync neighbors, A and B , do not adjust the times of their CPMP transmissions. Instead, both A and B set wake up times (using their `NW` lists) such that each is listening when the other transmits its CPMP updates. ■

C. Network Interface Manager

In our system, the network interface manager (NIM) module (Figure 1) interfaces with the operating system and determines when to transition the network card into the deep sleep mode. While the primary focus of our work is to synchronize wakeup times for “CPMP applications”, we realize that the algorithms implemented within the NIM module should not disrupt communication requirements for non-CPMP applications (e.g. browser). The problem is that it is impossible to anticipate when an event-driven application might need to use the network (mouse click) or when a remote device might initiate a network connection request. Therefore, power management for the wireless network interface is an active research area with particular focus on designing adaptive solutions for precisely this problem [12], [28], [11], [5].

In the Where-Fi system we implement a simple algorithm in the NIM module to handle non-CPMP applications. The NIM module exports simple API interfaces: `Schedule_NIC_On(time t)` and `Schedule_NIC_Off()`. The first call can be used by an application to specify (preferably ahead of time) when it requires the network interface to be active. The NIM module then returns a code to the application indicating whether the call can be successfully completed. If the indicated time is the current time, then the NIM boots up the network interface (if it is in deep sleep) which would introduce a delay, typically in the order or several hundred milliseconds. Using the above API, CPMP applications can specify the intervals for which they need the network interface to be active (calculated using the PAC algorithms). If an application is not cognizant of how long it needs the network (i.e. does not call `Schedule_NIC_Off()`), then the NIM module uses a simple heuristic based on network inactivity timeout to determine when to transition the network interface back to the sleep state. We have also implemented a more trivial algorithm where the NIM never turns off the network interface if a non-CPMP application is alive and using the network.

IV. WHERE-FI IMPLEMENTATION

We have designed and implemented two novel applications that present a unique “social experience” for co-located cell-phone users using the Where-Fi framework. Specifically, using our applications users can visualize current multimedia content being consumed around them and tune-in in real time to a participating user’s playlist.

The Where-Fi framework uses the CPMP protocol to update neighboring devices with metadata of currently “playing” content. This enables devices running Where-Fi to (i) visualize popular content around them (using an application called “Zeitgeist”) and (ii) follow a particular user in real-time by tuning in to his/her playlist (“MeCast”). The Zeitgeist application captures CPMP updates and builds statistics of the most popular items consumed in its vicinity. It then presents to the user a sorted list of the most popular items, allowing the user to obtain more information on items of interest, as well as to buy the rights to use them, over the currently available internet connectivity (Wi-Fi or GPRS). Using MeCast, a user (castee) can then follow any other user in the system. This is done automatically when the castee selects one of the displayed items currently consumed by a neighbor (caster). However, instead of having the castee manually select each item it wants to download, the MeCast application enables the castee to automatically transfer the selected caster’s items in the order in which they are consumed by the caster. This is done until the castee explicitly selects a different neighbor to follow or decides to consume its own items. Thus, each Where-Fi device effectively behaves like a radio station. Note that each device can be both a caster and a castee. In terms of access rights, we specify that for MeCast, a castee can access items transferred from a caster only as long as a direct Wi-Fi connectivity exists between them. When the caster and castee become disconnected, a DRM module in the castee erases all items transferred from the caster.

We have prototyped the Where-Fi ecosystem on two GSM Linux-based Motorola cell phone models (see *Appendix*): the Motorola A910 (X-Scale 300MHz processor, Wi-Fi 802.11b/g, UMA technology) and the ROKR E6 (X-Scale 300MHz processor, no WiFi but with Bluetooth PAN profile support). A super-peer system (Mac-Mini running Linux) was used as a gateway to enable the Bluetooth only phones to communicate with the WiFi phones. The PAC and the NIM modules are implemented using C++ and unix system calls. Each application consists of a native C++ module for performing lower level operations on the phone (networking, file access, CPMP interfacing) and a J2ME application for the user interface. The native and J2ME modules were interfaced using local sockets. The native modules are about 10000 lines of C++ code while the user interfaces were roughly 1000 lines of Java code.

V. PERFORMANCE EVALUATION

A. WhereFiSim

We designed and implemented a java based simulator called “WhereFiSim” for evaluating the performance of the PAC algorithms on larger scale networks that were hard to realize in practice. In WhereFiSim, nodes are distributed uniformly at random in a pre-defined rectangular space, specified for each experiment suite. The joining behavior of nodes is modeled by associating each node with a start time. Each simulation starts at time zero and lasts for a specified interval. For each experiment, we compared the FC_PAC and PC_PAC algorithms against a baseline algorithm, where the network card is kept always on. The battery consumption of nodes was modeled on the current drain characteristics of the Motorola A910 phone mentioned earlier. The average current drawn when the phone has the wireless card off is 2.63mA and the average current drawn when in active mode (GSM camped and Wi-Fi interface on) is 3.95mA. The total battery life of the A910 is 1100 milliamps hour (mAh). The WiFi card behavior is simulated on WhereFiSim by turning the card on one second before the scheduled transmission and is kept on for two seconds. This interval covers the latency of switching on/off the network card and allowing for collision avoidance algorithms at the MAC layer. This is a conservative behavior and is representative of our implementation on the actual phones.

In our simulations we used two different $R = t_s/t_a$ ratios. In the first setup, we set the length of the active intervals, t_a , to 32 seconds and the length of sleep intervals, t_s , to 64 seconds (i.e. $T_X = 32$, $R = 2$). In the second simulation, we set $t_a = 20$, $t_s = 80$ (i.e. $T_X = 20$, $R = 4$). For both simulations, the total duration of an active plus a sleep interval was roughly 100s. The following section reports the results of our evaluation based on the following criteria: time to sync, packet loss and energy consumption.

B. Performance Results

FC_PAC: In the first set of experiments we randomly placed between 2 and 50 nodes in a $100 \times 100 m^2$ area, that upper bounds the dimensions of a coffee shop, airport lounge or conference room. Due to the transmission range of nodes (115m), the networks formed are fully connected. We set the start time of each node to a random value between 0 and 100s, and performed each experiment for a duration of 25000s.

In Figure 4 we compare the synchronization time and the energy consumed by nodes running FC_PAC for the two simulations described in the Section V-A. We consider the synchronization time of a node to be the time when the node first “phase locks” its transmission times with the other nodes in the network. Figure 4(a) shows the average per-node synchronization time for networks between 3 and 50 nodes. The graph shows that a shorter T_X synchronizes nodes quicker even when they have larger sleep intervals thereby demonstrating that T_X has a more dominating effect than R in synchronizing nodes. Second, it shows that the time to synchronize decreases with the number of nodes in the network. The reason for this is that the first node to send a CPMP packet is the one to wait the longest until it becomes synchronized. As the number of nodes increases, the

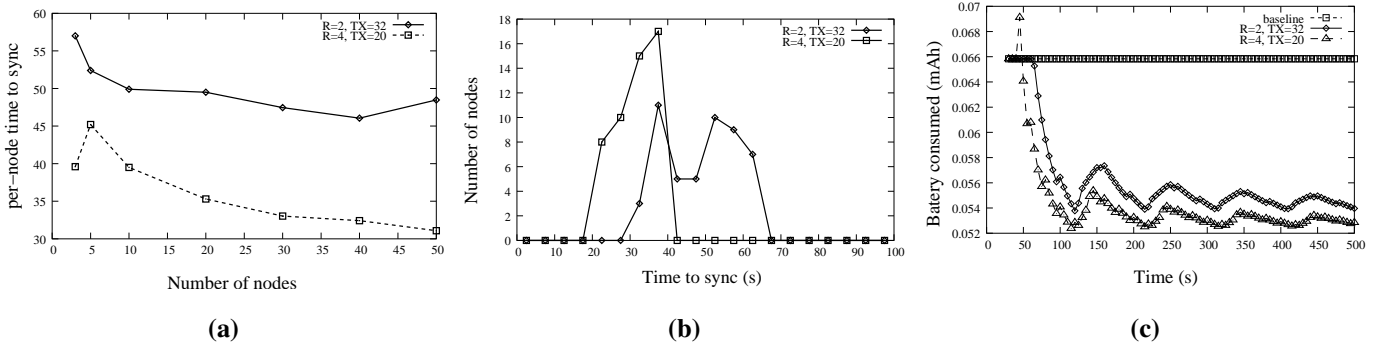


Fig. 4. FC.PAC for the two setups ($R = 2, TX = 32$ and $R = 4, TX = 20$). (a) Evolution of the average per-node time until synchronized, for networks between 2 and 50 nodes. (b) The distribution of the time required to synchronize for the nodes of a network with 50 members. (c) Evolution in time of the average battery consumed by a node using FC.PAC vs. the energy consumed by a node continuously keeping its network card active. The numbers are shown for one of the nodes of a 50 node network, for both setups, ($R = 2, TX = 32$ and $R = 4, TX = 20$).

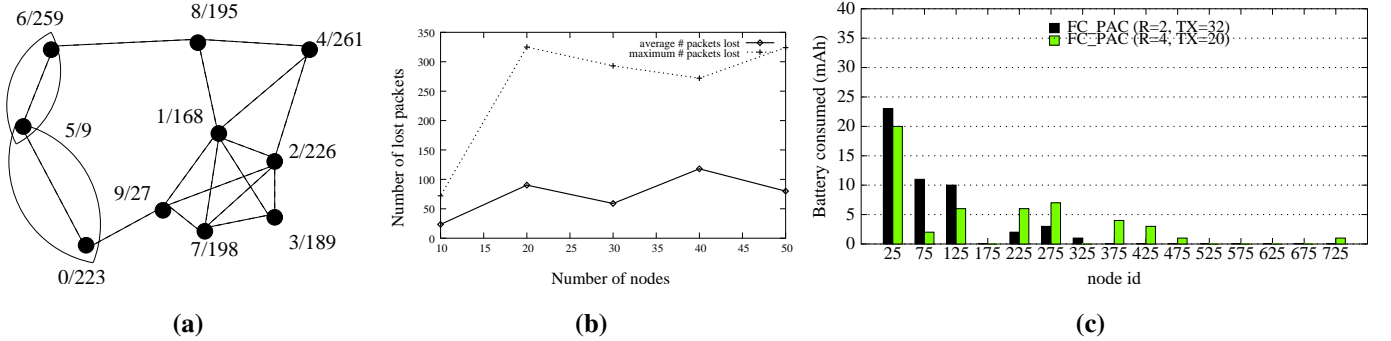


Fig. 5. FC.PAC: evaluation of hidden terminal effects. (a) Example topology of a 10 node network. Nodes are named with numbers from 0 to 9. Dotted lines represent connectivity. Ovals show some pairs of nodes that are synchronized by the end of a 2000s experiment run. Each node is labeled with its name and its start up time. For instance, 0/223 specifies that node 0 starts at time 223s. Node 0 synchronizes only with node 5, effectively missing all updates from neighbors 7 and 9. (b) Average and maximum per-node number of packets lost due to hidden terminals in networks of 10 to 50 nodes. (c) Distribution of the number of lost packets per node, in a 50 node network, for the two setups previously introduced.

effect of the sync time of the first node on the average sync-time reduces. Figure 4(b) illustrates the bell-shaped distributions of node synchronization times for a network of 50 nodes, for the two simulations. For the first setup, node synchronization times range between 30-35s (for 2 nodes) and 60-65s (for 8 nodes). For the second setup, node synchronization times have a much smaller range, with the lower bound in the range of 20-25s (8 nodes) and the upper bound of 35-40s (17 nodes).

The comparison of the average battery consumed in time by one node of a 50 node network, running FC.PAC for the two setups of Section V-A and the baseline algorithm is shown in Figure 4(c) for a 500 second interval. Each point is shown as the total battery consumed, measured in milliamps hour (mAh), averaged over the time since the node is active. The spike seen at the beginning of the experiment (time 40s) for the second setup, is due to a transmission that occurs at that time, which increases the average. However, the average value of the battery consumed is seen to be decreasing in time. Nodes running the baseline algorithm ran out of battery after an average of 16735 seconds, whereas nodes running FC.PAC in the first simulation ran out of battery after 20425s. This implies a battery lifetime extension of 22%. In the second simulation, nodes ran out of battery after 21575 seconds, which represents a 28% battery lifetime extension over the baseline algorithm.

Hidden Terminal Effects: The FC.PAC algorithm assumes that all the nodes of the network are within each other's communication range. We now show the effects of this assumption in an environment where hidden terminals occur. For this, we generated networks of 10 to 50 nodes and simulated deployment in areas of sizes ranging between $200 \times 200 m^2$ and $500 \times 500 m^2$. Figure 5(a) shows an example network of 10 nodes, deployed in a $200 \times 200 m^2$ square area. We ran each experiment for 2000 seconds. As expected we found that some nodes only synchronized with a subset of their neighbors and we out of phase with others resulting in missed updates. For instance, using the network from Figure 5(a), node 5 synchronizes with both its neighbors, 0 and 6, whereas node 0 does not synchronize with neighbors 7 and 9, missing all their updates. Figure 5(b) shows the average and the maximum per-node number of updates lost in our experiments. For example, for networks of 20 nodes, the average number of updates lost is 90, with a maximum of 325 packets lost for a single node. Figure 5(c) shows the distribution of the number of lost packets per node, in a 50 node network, for the two setups previously introduced. In the first setup, almost half of the nodes lost less than 50 packets, 21 nodes lost between 50 and 150, while 6 nodes lost between 200 and 350 packets. In the second setup, 20 nodes lost under 50 packets. However, 29 nodes lost between 50 and 500 packets and one nodes lost 728 packets. Nodes lost more packets for larger R values, when the value of $t_a + t_s$ remains constant. This is because for larger R values nodes will send more CPMP updates, resulting in higher losses. In the next subsection we report the performance of PC.PAC, which successfully handles hidden terminals.

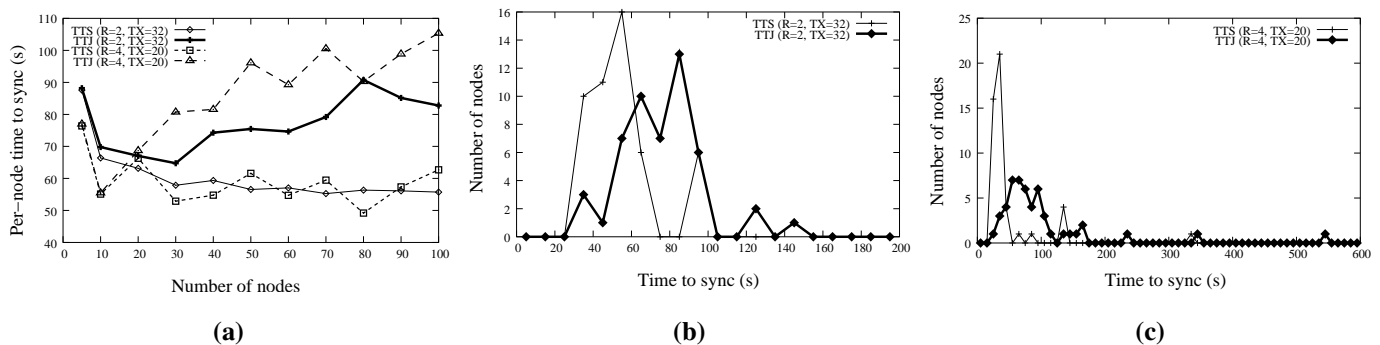


Fig. 6. PC.PAC: time to synchronize for networks of up to 50 nodes. (a) Evolution of the average per-node time-to-sync (TTS) and time-to-join (TTJ), for networks between 2 and 50 nodes. (b) First setup ($R = 2, TX = 32$): the distribution of TTS and TTJ for the nodes of a network with 50 members. The x axis represents time and the y axis shows the number of nodes whose TTS or TTJ takes that value. (c) Second setup ($R = 4, TX = 20$): the distribution of TTS and TTJ for the nodes of a network with 50 members.

PC.PAC: We evaluated PC.PAC by simulating networks of up to 100 nodes in square areas of up to $750 \times 750 m^2$. Similar to previous measurements, we experimented with two setups, one where $TX = 32$ and $R = t_s/t_a = 2$ and one where $TX = 20$ and $R = 4$. The starting times of nodes were set to be close to the beginning of the experiment (time 0), such that no two nodes have the same start up time. The reason for this is that we wanted to avoid having to account for the late start up times of nodes and also to avoid a situation where neighboring nodes start already synchronized.

We first measured the time required for nodes to synchronize with all their neighbors. For this, we used two metrics. The first metric is the time it takes a newly joining node to synchronize with another node, already part of the network. This is the same metric as the one we used when measuring the performance of FC.PAC. We call this metric *time-to-sync* (TTS). The second metric is the time it takes for a joining node to discover all its neighbors and set up its schedule to include their transmission times. We call this metric *time-to-join* (TTJ).

Figure 6(a) compares the average per-node TTS and TTJ values for networks having between 5 and 100 nodes. Similar to FC.PAC, the average value of TTS decreases (from 87s to around 55s) when the size of the graphs increases. For the same setup, the TTJ value is always larger than TTS, since TTS is set when a node receives a first CPMP packet, whereas TTJ is set when (or after) a node receives CPMP packets from all its neighbors. The highest difference between TTJ and TTS is of 43 seconds, for the first setup and for networks of 100 nodes. Overall, for both setups, the value of TTJ exhibits an increasing trend with larger networks. This is because the value of TTJ is determined by the network's degree as well as the latest start up time of neighbors.

Figures 6(b) and 6(c) depict the distribution of the TTS and TTJ values for the two setups, for a network of 50 nodes. The maximum value for the TTJ of a node for the first setup ($R = 2, TX = 32$) is under 150s, whereas for the second setup ($R = 4, TX = 20$), a few nodes have TTJ values in excess of 200s, with a maximum of 545s. This is because when nodes have shorter active intervals the probability of their active intervals to intersect is smaller.

For the same network settings, we also studied the battery consumed by a device when running our PAC algorithms. Figure 7(a) compares the per-node battery consumed by PC.PAC and FC.PAC with the battery consumed by the baseline algorithm for a network of 20 nodes, in 10000 seconds. As seen from the figure, the energy consumption of PC.PAC is comparable to that of FC.PAC which is good because we prevent lost updates when using PC.PAC. When using the first setup ($R = 2, TX = 32$) FC.PAC saves on average 20% and PC.PAC saves 18% of the battery consumed by the baseline algorithm. For the second setup, PC.PAC saves on average 19% of the energy consumed by the baseline algorithm. The difference between the two setups for PC.PAC is also illustrated in Figure 7(b), that shows the evolution in time (between times 1700s and 2000s of the experiment) of the average battery consumed by one of the nodes in a 20 node network.

In our final experiment, we measured the longest interval a node manages to turn off its wireless card when running PC.PAC, which provides insight into how to improve our NIM module. This value is influenced by the node's degree as well the number of unsynchronized neighbors. Figures 7(c) and 7(d) show the distribution of the LSI value for nodes in networks of 50, 70 and 100 nodes for the two setups. For each network, around half of the nodes have a LSI value close to T_X , the maximum possible. Such nodes have either a small number of neighbors (nodes placed on the boundary of the network) and/or are able to synchronize with all their neighbors. However, most of the other half of the network has an LSI of between 15s and 30s (Avg ≈ 25 s) for the first setup and between 5s and 15s (Avg ≈ 16 s) for the second setup. The variation of the average LSI among graphs of different sizes for the same setup was observed to be quite small.

Experiments Summary: The summary of our performance evaluation is as follows. For highly connected networks the use of FC.PAC is recommended. In such networks nodes have a good chance of synchronizing with all their neighbors, thus receiving all CPMP updates. Moreover, FC.PAC will save considerable battery power (e.g., 22-28% more battery life than the baseline algorithm). For disconnected networks, where synchronization clusters may form, the use of PC.PAC is recommended. While nodes using PC.PAC take a longer time to reach a stable state (between 1% and 69% more time than FC.PAC) and save less energy (around 3% less than FC.PAC), articulation nodes (nodes neighboring on several synchronization clusters) running PC.PAC will receive all CPMP updates of neighboring clusters.

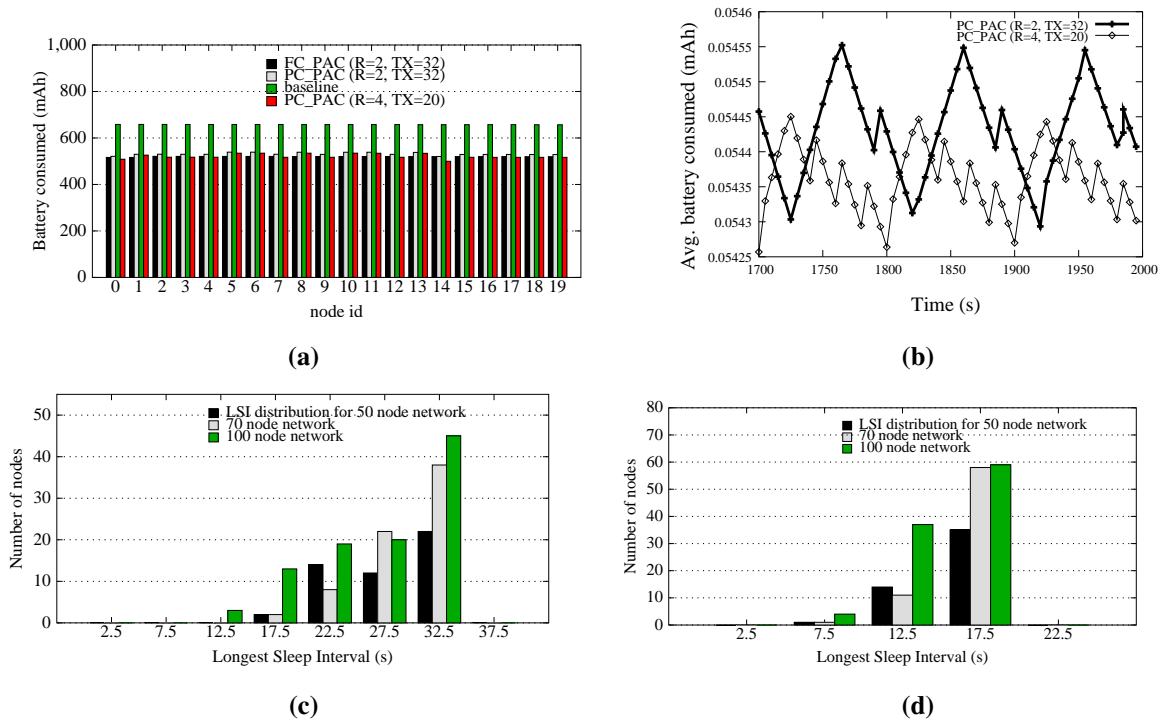


Fig. 7. (a) Comparison of per-node energy consumed in 10000 seconds by PC_PAC, FC_PAC and the baseline algorithms. (b) Comparison of the two setups in 300s (1700s-2000s): average battery consumed by PC_PAC in a network of 20 nodes. The battery consumed is measured in milliamps hour. The small spikes denote transmission intervals. The increasing slope intervals denote active intervals whereas decreasing slopes denote sleep intervals. (c) Per-node longest sleep interval (LSI) for the first setup ($R = 2, TX = 32$). The x axis shows the longest time to sleep in intervals of 5s denoted by their center value. The y axis shows the number of nodes whose LSI value is in the corresponding interval. (d) LSI distribution for the second setup ($R = 2, TX = 32$).

VI. RELATED WORK

State-of-the-art research on ad-hoc networks continues to search for ways to optimize energy while minimizing the penalties incurred due to latency, dropped packets and partitioned networks caused by induced low-duty cycles on the network interface. In [7], the author identifies three broad categories in which power management for ad-hoc networks can be classified: rendezvous based wakeup [8], [13], [30], [23], [20], [25], [36], [19] where all nodes are synchronized to listen to the medium around the same time, asynchronous wake-up [35], [16] where nodes are not synchronized but the wakeup cycles are designed to overlap, and booted wakeup [26], [29], [22] wherein a low-power alternate radio (e.g. bluetooth) is used to sense the medium and boot up the wireless interface when required. Interestingly, depending on the nature of the ad-hoc ecosystem (e.g. application behavior, network topology) under consideration, one of the aforementioned categories typically provide the best solution. For our application requirements and specific ecosystem which needed all nodes to receive updates frequently and reliably, the scheduled rendezvous based wakeup approach was the obvious choice.

Most broadcast/multicast based protocols generally assume knowledge about their location to optimize wakeup cycles. SPAN [13] for example exploits network redundancy by keeping active a subset of active nodes such that every destination is at most one hop away. It is a distributed protocol wherein nodes make local decisions as to whether they should sleep or join a forwarding backbone. In [19], network redundancy is exploited by using mobile nodes to ferry messages within the network. Such an approach can have unbounded latencies and requires neighbor discovery or location awareness to identify destination nodes. Firefly synchronization [23], [30] models the internal clock of a firefly as a pulse coupled oscillator whose phase is modified upon reception of an external message. The goal is to apply synchronization phenomena occurring in nature (e.g. fireflies) to ad-hoc systems. The Pulse protocol [8] was proposed for sensor networks to achieve synchronization using a gateway node. A pulse emanating from the gateway is used to generate a tree to provide multihop routing paths to the gateway. Here all traffic needs to be routed to the gateway. The birthday paradox approach [25] proposes that nodes that are not involved in communication can go to sleep for a period proportional to their neighbor density and residual battery power. S-MAC [32] is a medium access protocol that uses messages to synchronize nodes in a sensor network with nodes have constant sleep/awake periods. This approach is extended in T-MAC [14] which dynamically adjusts the periods based on snooped traffic patterns. Odds [20] presents a probabilistic approach for localized decision making on each node with respect to when and how long it needs to enter standby mode to conserve energy. While several of the above approaches are designed for multi-hop ad-hoc networks, ours is more specific to ad-hoc networks where the goal is to disseminate information and not forward packets. Unlike several of the above approaches we do not require nodes to be time synchronized or nodes to be aware of their neighbors, both of which are complex intensive processes and introduce inaccuracy in highly dynamic environments. While unrelated to this work, we direct interested users to explore other interesting solutions that have been explored in the

context of power saving in ad-hoc networks [16], [26], [29], [22], [35].

We also consider our approach to be a cross-layer optimization approach wherein our middleware component (NIM) directly interacts and controls the high/low duty cycles of the network interface. In this respect our work could significantly benefit from solutions presented in other cross layer systems [6], [34], [24]. Power consumption characteristics of network interfaces have been well studied for commercially available wireless interfaces. Interesting results on wireless power consumption for popular media formats are presented by Chandra et.al. in [11], [12]. Feeney et. al have also explored and simulated power saving mechanisms in wireless ad-hoc networks [18], [17], [31].

VII. CONCLUSION AND FUTURE WORK

In this paper we studied the problem of synchronizing the periodic transmissions of neighboring devices in a novel wireless framework, called Where-Fi. We proposed two synchronization algorithms, FC_PAC exclusively for highly connected networks and PC_PAC for more disconnected environments, where hidden transmissions may occur. Our results show that for all network topologies, devices running PC_PAC were able to receive all multicast packets from neighboring nodes (unlike FC_PAC). However, the PC_PAC algorithm exhibited higher stabilization times (≈ 1 minute longer than FC_PAC) but showed comparable battery savings (only 3% less than FC_PAC). Future work would include designing mechanisms for making the PAC algorithms resilient to Byzantine attacks, improving the NIM module to better handle non-CPMP applications and finally designing novel Digital Rights Management (DRM) techniques for location-based content sharing.

REFERENCES

- [1] "Consumer Taste Sharing Is Driving the Online Music Business and Democratizing Culture", <http://cyber.law.harvard.edu/home/uploads/511/11-consumertastesharing.pdf>.
- [2] <http://www.breakthru.com/index.php/weblog/press/>.
- [3] "The Nokia Sensor Project", <http://europe.nokia.com/a4144923>.
- [4] The UPnP Forum, UPnP Low Power Architecture v1.0. 2007.
- [5] M. Anand, E. B. Nightingale, and J. Flinn. "SelfTuning Wireless Network Power Management". In *MobiCom*, Sept 2003.
- [6] M. Anand, E. B. Nightingale, and J. Flinn. "Ghosts in the Machine: Interfaces for Better Power Management". In *The Second Annual International Conference on Mobile Systems, Applications and Service*, June 2004.
- [7] T. Armstrong. Wake-up based power management in multi-hop wireless networks. In *Term Survey Paper for Quality of Service Provisioning in Mobile Networks*, Univ. of Toronto, 2005.
- [8] B. Awerbuch, D. Holmer, and H. Rubens. "The Pulse Protocol: Energy efficient Infrastructure Access". In *IEEE INFOCOM*, 2004.
- [9] H. Caitiuro-Monge, K. Almeroth, and M. del Mar Alvarez-Rohena. "Friend Relay: A Resource Sharing Framework for Mobile Wireless Devices". In *ACM International Workshop on Wireless Mobile Applications and Services on WLAN Hotspots*, Sept 2006.
- [10] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. "Design and Evaluation of a Wide-Area Event Notification Service". *ACM Transactions on Computer Systems*, Vol 19, number 3, Oct 2001.
- [11] S. Chandra. "Wireless Network Interface Energy Consumption Implications of Popular Streaming Formats". In *MMCN*, 2002.
- [12] S. Chandra and A. Vahdat. "Application-specific Network Management for Energy-aware Streaming of Popular Multimedia Formats". In *Usenix Annual Technical Conference*, June 2002.
- [13] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. "Span: an Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks". In *ACM Wireless Networks Journal*, Volume 8, Number 5, Sept 2002.
- [14] T. V. Dam and K. Langendoen. "An adaptive energy-efficient MAC protocol for wireless sensor networks". In *ACM SenSys*, Nov 2003.
- [15] J. Elson and D. Estrin. "Time Synchronization for Wireless Sensor Networks". In *IEEE IPDPS*, 2001.
- [16] L. M. Feeney. "A QoS Aware Power Save Protocol for Wireless Ad Hoc Networks". In *Proceedings of the First Mediterranean Workshop on Ad Hoc Networks*, 2002.
- [17] L. M. Feeney, C. Rohner, and B. Ahlgren. "The impact of wakeup schedule distribution in asynchronous power save protocols on the performance of multihop wireless networks". In *IEEE Wireless Communications and Networking Conference (WCNC'07)*, Mar 2007.
- [18] P. Hurni, T. Braun, and L. M. Feeney. "Simulation and evaluation of unsynchronized power saving mechanisms in wireless ad hoc networks". In *4th International Conference on Wired/Wireless Internet Communications (WWIC 2006)*, MAY 2006.
- [19] H. Jun, W. Zhao, M. H. Ammar, E. W. Zegura, , and C. Lee. "Trading Latency for Energy in Wireless Ad Hoc Networks using Message Ferrying". In *Workshop on Pervasive Wireless Networking (PERCOM)*, 2005.
- [20] Z. Li and B. Li. "Probabilistic power management for wireless ad hoc networks". In *ACM/Kluwer Mobile Networks and Applications (MONET)*, Oct 2005.
- [21] R. Meier and V. Cahill. Steam: Event-based middleware for wireless ad hoc networks. In *Proceedings of the 22 nd International Conference on Distributed Computing Systems Workshops*, 2002.
- [22] M. J. Miller and N. H. Vaidya. "Power Save Mechanisms for Multi-Hop Wireless Networks". In *Proceedings of the First International Conference on Broadband Networks (BROADNETS)*, 2004.
- [23] R. E. Mirolo and S. H. Strogatz. "Synchronization of pulse-coupled biological oscillators". *SIAM Journal on Applied Mathematics*, Volume 50 , Issue 6, Dec 1990.
- [24] S. Mohapatra. "DYNAMO: Power Aware Middleware for Distributed Mobile Computing". PhD thesis, University of California, Irvine, Dec 2005.
- [25] S. PalChaudhuri and D. B. Johnson. "Birthday Paradox for Energy Conservation in Sensor Networks". In *USENIX Symposium of Operating Systems Design and Implementation*, 2002.
- [26] T. Pering, Y. Agarwal, R. Gupta, and R. Want. "CoolSpots: reducing the power consumption of wireless mobile devices with multiple radio interfaces". In *Proceedings of ACM/USENIX MobiSys*, 2006.
- [27] P. Persson, J. Blom, and Y. Jung. "DigiDress: A Field Trial of an Expressive Social Proximity Application". In *UbiComp*, 2005.
- [28] P. Shenoy and P. Radvok. "Proxy-Assisted Power-Friendly Streaming to Mobile Devices". In *MMCN*, 2003.
- [29] E. Shih, P. Bahl, and M. J. Sinclair. "Wake on Wireless: An Event Driven Energy Saving Strategy for Battery Operated Device". In *Proceedings of the 8th annual international conference on Mobile computing and networking*, 2002.
- [30] A. Tyrrell, G. Auer, and C. Bettstetter. "Fireflies as Role Models for Synchronization in Ad Hoc Networks". In *Intl. Conf. on Bio-Inspired Models of Network, Information, and Computing Systems (BIONETICS)*, Dec 2006.
- [31] H. Yan, D. Lowenthal, K. Li, R. Krishnan, and L. Peterson. "Client-Centered, Energy-Efficient Wireless Communication on IEEE 802.11b Network". In *IEEE Transaction on Mobile Computing*, 2006.
- [32] W. Ye, J. Heidemann, and D. Estrin. "An energy-efficient MAC protocol for wireless sensor networks". In *IEEE INFOCOM*, 2002.
- [33] E. Yoneki and J. Bacon. "Dynamic group communication in mobile peer-to-peer environments". In *ACM Symposium on Applied Computing*, 2005.
- [34] W. Yuan. "GRACE-OS: An Energy-Efficient Mobile Multimedia Operating System". PhD thesis, University of Illinois at Urbana-Champaign, October 2004.
- [35] R. Zheng, J. C. Hou, and L. Sha. "Asynchronous Wakeup for Ad Hoc Networks". In *ACM MobiHoc*, Jun 2003.
- [36] R. Zheng and R. Kravets. "On-demand Power Management for Ad Hoc Networks". In *IEEE INFOCOM*, 2003.

APPENDIX

This is a supplemental page that shows snapshots of the "Where-Fi" system in action.



Fig. 8. Where-Fi demo showing three Motorola A-910 phones with built in WiFi support, in ad-hoc connection mode. The label above each phone describes the type of application running on the phone.



Fig. 9. The rightmost phone (castee) receives a CPMP update from the middle phone (caster). The castee's UI shows the content currently accessed by the caster. The castee then chooses to follow the caster using the "Follow" menu button.



Fig. 10. The Zeitgeist phone shows the most popular items recently accessed locally (only 2 up to this point in the demo). The castee starts playing the same content the caster is playing.



Fig. 11. The Zeitgeist application shows details of a selected content item, with its popularity indicated by a counter (see the stylus).



Fig. 12. The Zeitgeist application also allows users to download additional information about the content item over the phone's GPRS connection.



Fig. 13. Bluetooth only phone (Motorola ROKR E6) participating in the Where-Fi ecosystem.