MULTIHOP SYNCHRONIZATION IN FRACTEL AND COMMUNICATION THROUGH ENERGY SENSING

A Thesis Submitted
in Partial Fulfillment of the
Requirements for the Degree of
Master of Technology

Ashutosh Dhekne
under the guidance of
Prof. Kameswari Chebrolu
and
Prof. Bhaskaran Raman

Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

Dissertation Approval Certificate

Department of Computer Science and Engineering

Indian Institute of Technology, Bombay

The dissertation entitled **"Multihop Synchronization in FRACTEL and Communication through Energy Sensing"**, submitted by **Ashutosh Dhekne** (Roll No: **07305016**) is approved for the degree of **Master of Technology** in **Computer Science and Engineering** from **Indian Institute of Technology, Bombay**.

Prof. Kameswari Chebrolu **CSE, IIT Bombay** **Supervisor**	**Prof. Bhaskaran Raman** **CSE, IIT Bombay** **Co-Supervisor**
Prof. Anirudha Sahoo **CSE, IIT Bombay** **Internal Examiner**	**External Examiner**

Chairperson

Place: IIT Bombay, Mumbai
Date: 24^{th} **June, 2009**

Abstract

Wireless technology provides many avenues for connecting remote inaccessible areas to the digital grid of the modern world. Long distance connectivity is possible using directional and sectorized antennas. Taking this further, we may support multiple hops multiplying the distance covered by such networks. In this thesis, we implement a multihop TDMA system with synchronized nodes and centralized scheduling using off-the-shelf wifi hardware and open source madwifi driver. We use Linux software timers and the high accuracy hardware clock for maintaining the slot structure at each node in a tree topology. The system can be used for providing VoIP and video conferencing abilities to remote villages and may also serve as a back-haul for GSM networks taking cellular connectivity to the remotest areas without incurring the high cost of laying fiber.

A second part of this thesis presents a novel technique of communication between wireless devices that share the same frequency spectrum but use different physical layer encodings. We use energy sensing for making this communication possible and hence call it Esense. We demonstrate that it is possible to perform meaningful data communication between such devices using the length of energy bursts as an alphabet. For our evaluation, we use a 802.11 wifi device and a 802.15.4 device and propose the relevance of such communication to three distinct research domains: (1) coexistence of wireless devices, (2) energy management and (3) interference mapping. We use packet lengths that are not frequently used by the 802.11 devices to build an alphabet for Esense and show that the 802.15.4 device can validly detect the presence of the Esense packet on air in most cases. We propose that this technique be incorporated in future wireless standards enabling a collaborative approach towards other standards.

Acknowledgements

I wish to extend my most sincere thanks to my advisors Prof. Kameswari Chebrolu and Prof. Bhaskaran Raman for providing me the opportunity of being a part of this project. Also, Prof. Purshottam Kulkarni was always accessible to help in my concerns and share my excitements. Immense help has been provided by the Department of Electrical Engineering at IIT Bombay in terms of availability of the antenna lab for my experimentations using the spectrum analyzer. Without their support, the Esense part of this project was not possible.

This work was jointly done with Nirav Uchat and I am deeply touched by his calm nature and benevolence. Our friendship grew through the sheer spirit of working together for hours on various aspects of this project. I wish to thank him for patiently listening me out on multiple occasions and always being supportive even when I was working on Esense.

I wish to thank all my friends who have constantly encouraged me and allowed me to bounce ideas off them. Nothing of this project would have been possible without their support. Finally, I wish to thank the Department of Computer Science and Engineering for providing state-of-the-art infrastructure that set the stage right for my project work.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The progress of wireless brings forth many prospects for linking remote, inaccessible areas to the better provisioned ones. Wireless medium is more suited for such regions than wired connectivity due to lower cost and relative ease of deployment of wireless networks. The main challenge in wireless is long distance transmission and reception of signals which has now been shown to be possible through the use of directional and sectorized antennas. Once this main hurdle is circumvented, we can investigate protocols for long distance connectivity over wireless links. In the first part of this thesis, we shall delve into one such protocol that we have designed. This work has been done together with Nirav Uchat. The synchronization and timer aspects of the implementation are presented in this work, whereas Nirav describes the protocol in complete detail and also walks through the implementation.

Another part of this thesis deals with communication through energy sensing on the wireless medium. When different devices use the same frequency spectrum, but use a different physical layer encoding, they cannot communicate with each other. This frequently leads to collision of signals or causes back offs leading to a decreased performance for all the devices. Typically in the Industry, Scientific and Medical (ISM) band, devices such as cordless phones, sensor devices, and wifi devices all share the same frequency band and cause interference with each other. Even though the devices cannot understand each other's signals, they can sense that some communication is on going because of an increased level of energy on air. In this part of the thesis, we investigate a novel technique of communication through sensing of energy and assigning meaning to the lengths of energy bursts.

Chapter 2

Introduction - Synchronization

Connectivity to the outside world is no longer a luxury even for the remotest parts of the world. It is a necessity for better living conditions, brighter livelihood and better knowledge of the world. However, constructing robust infrastructure for digital connectivity takes a long time and at times ceases to be economically viable because of the sparse population in such regions. Laying fiber optic lines to these areas is surely costly and time consuming, but if we have a way of communication without the need to lay lines, the cost of setup would reduce drastically. The FRACTEL project [11] is envisioned to provide internet connectivity to remote and inaccessible areas using long distance wireless links as the back-haul and shorter distance links for last mile access. In addition to data connectivity, we also envision to provide voice and video conferencing abilities using this infrastructure. This provisioning of quality of service and the use of long distance links make it necessary to modify the standard 802.11 protocol.

We use the license free 2.4GHz frequency band with commodity 802.11 WiFi hardware and the open source madwifi driver [6]. We have a *root* node connected to an ISP through some wired connection. This root node is most likely to be placed at a location in a nearby city or town. It is connected to other nodes placed about 25-30 km apart using long distance links established by directional antennas. These second-tier nodes will then be connected with other such nodes and so on, forming a tree topology. At the last level, user devices are connected with a local omni directional antenna connected to the tree at some node. Transmission and reception of data by these nodes in the tree topology is controlled by a schedule disseminated by the root node. Since the areas where this project is to be deployed are devoid of any

wifi activity, we do not need a contention based protocol. The root node is supposed to create the schedule such that nodes do not have to contend for the channel; it is supposed to create a TDMA schedule. Figure 2.1 shows an example of the FRACTEL system.

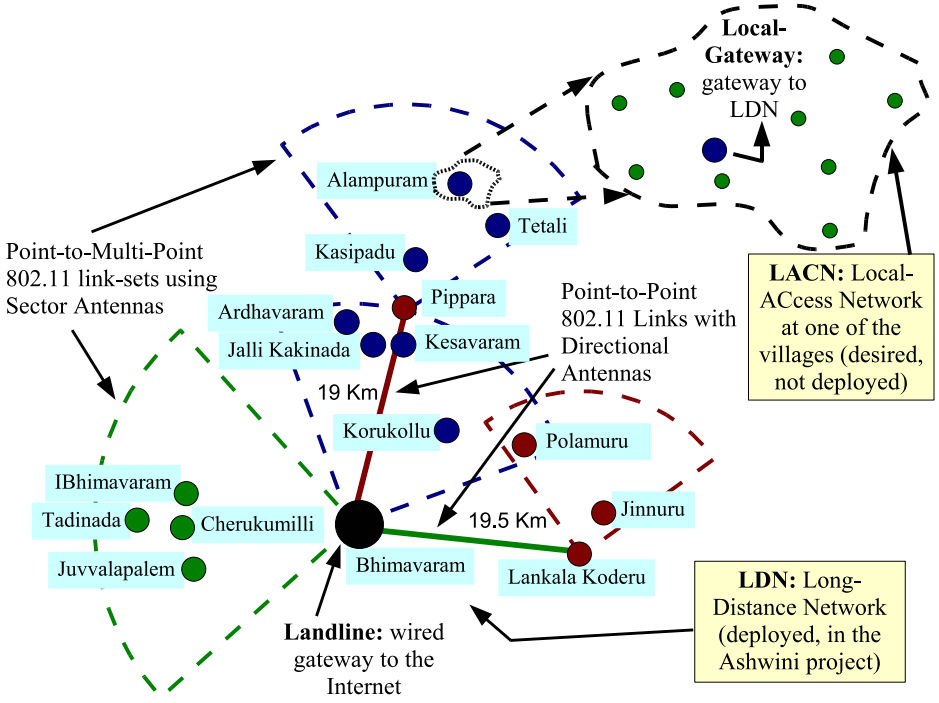


Figure 2.1: An example deployment

The project addresses many issues in the domains of scheduling, node synchronization, MAC layer modifications, routing, and experimentations with long distance links. This part of the thesis primarily deals with synchronization of nodes. In a tree topology, and when we wish to run a TDMA schedule over it, we need to synchronize the clocks of all the nodes. Clocks typically drift apart from each other and hence require periodic resynchronization. Also, a TDMA implementation depends heavily on perfect slotting of time. We discuss how we achieve this time slotting using Linux kernel timers. In contrast to previous work on synchronization, we perform *multihop* synchronization of nodes. In the rest of this report, we shall discuss the approach we are taking for time synchronization of nodes and maintaining the slotting structure.

We start by looking at what aspects have been dealt with in the related work. We then describe our approach for time synchronization. It involves quantification of the clock drifts between different cards, and the mechanism for synchronizing nodes in the topology. The TDMA MAC protocol has been designed, and we have a working

schedule structure in place and have tested the synchronization mechanism on a 5-node linear topology. Our implementation of the TDMA protocol shows very good results for running video conferencing and streaming media applications.

Chapter 3

Related Work

The synchronization problem we face in our network is very similar to that faced in wireless sensor networks. We wish to use the synchronized nodes to transmit and receive in accordance with a TDMA schedule instead of the usual CSMA. We therefore require synchronizing nodes at microsecond accuracy and maintaining this synchronization. We will first look at the uncertainties in the critical path and then discuss few synchronization protocols used in wireless sensor networks.

3.1 Uncertainties in synchronization

The reason that synchronization is so difficult is because of a number of uncertainties in the time required for activities in the critical path [24]. After we timestamp a packet in software, the packet queues in the hardware to get transmitted. When its turn comes, it requires some time to actually go on air called the transmit time. The packet then takes some time to travel from the transmitting node to the receiving node. This is called the propagation delay. If the nodes are fixed, the propagation delay is constant baring significant environmental changes. When the receiver receives the packet, it causes an interrupt and then the interrupt service routine (ISR) is invoked. The ISR is the earliest place in software on the receiver side where we may record the receive time. However, some hardware may provide the hardware-receive timestamps which are more accurate than the software timestamps.

With a timestamp exchange protocol, the clock skew between two clocks can be adjusted for. However, there is an additional uncertainty with clocks. Two clocks once adjusted for skew will not continue to run synchronously due to clock drifts.

Two clocks may run at slightly different speed depending on their manufacturing precision, crystal properties, and environmental conditions. The drift shown by a clock may itself keep changing and therefore usually requires sophisticated regression models for adjustments.

3.2 Related Synchronization Techniques

We will now briefly look at a few synchronization techniques for sensor networks, which are also applicable to our setting.

3.2.1 Reference Broadcast Synchronization

We may be able to increase the accuracy of time synchronization if we eliminate some of the uncertainties in the critical path. The RBS protocol [13] achieves this by using a novel technique. A reference node periodically broadcasts synchronization beacons and all the receivers who receive this beacon record their local time and then exchange the recorded time with each other. In this protocol, all the receivers of the beacon are synchronized with each other. The sender of the beacon is not synchronized with the receivers. Therefore, the beacon itself need not contain any timing information.

With the sender's uncertainties eliminated from the critical path, significant timing accuracy can be gained. However, every receiver may have to store an offset with respect to all other receivers. This can be a significant overhead particularly if the nodes are resource constrained. Moreover, we require a reference broadcaster to remain outside of the time synchronization since it cannot be synchronized in this mechanism. This also implies that its broadcast must be visible to each receiver present in the network. It is, therefore, not suitable for hierarchical topology and non-omni directional links.

3.2.2 Timing-sync Protocol for Sensor Networks

This is a sender–receiver synchronization protocol [14]. The working of the protocol is divided into two phases. The first phase essentially creates a spanning tree of the mesh network. This tree structure is used by the second phase for a pair-wise exchange of timing information. The child and the parent node know each other from the first phase. The parent sends a time_sync packet to the child node. The child

node responds with a time stamped packet (time = T1) and sends it to the parent. The parent node records its own local time when it received this packet (time = T2) and responds with another time stamped packet at time T3 which also includes the times T1 and T2. The child again records the time it received this response packet (time = T4).

The child node now knows the time difference between T1 and T2, which corresponds to the clock skew between the two plus the propagation delay. This protocol gives us the clock skew as well as the propagation delay. An advantage of this protocol over RBS is that it is inherently multi-hop. However, each pair of nodes in the tree structure must exchange two messages to be synchronized. Since it works over a tree topology, it may be suitable in our setting. However, the way we have designed our protocol, we do not support two-way message exchange for synchronization.

3.2.3 The Flooding Time Synchronization Protocol

This protocol achieves sender-receiver synchronization by broadcasting time stamped messages to all possible receivers [17]. It uses a preamble and few sync bytes which can be continuously used by the receiver to estimate the clock difference between the sender and the receivers. It reduces the jitter in interrupt handling time by averaging multiple timestamps taken at each byte boundary. Only the final average timestamp is embedded into the packet. This protocol also includes linear regression to predict the clock drift. The paper mentions that the clock drift in the Mica motes is about $40\mu s$. We observed a smaller clock drift in our hardware and may not need to compensate it, since the clocks are resynchronized periodically. This protocol also deals with changing the root node periodically. Due to our natural tree topology, we do not require this feature.

3.2.4 Clock Synchronization Protocol for 802.11 Networks

This protocol deals specifically with the problem of synchronizing 802.11 nodes in a multihop adhoc network. [26]. The paper describes how the number of stations sending the beacon can be reduced by designating some stations as more “important” for beaconing than others in a mesh network. This reduces beacon collisions. It requires creation of dominating sets of nodes which helps in deciding which nodes are more important. It also does frequency adjustments to get all clocks to have about

the same accuracy. It deals with general mesh ad-hoc networks and not with tree topology we have in our application. The paper, however, describes a very useful mechanism already present in 802.11. The beacon frames sent in ad-hoc networks themselves contain a timestamp which is that time of the local clock when the first bit of the timestamp hits the wireless interface. Added to propagation delay this gives an error of $\pm 5\mu s$. The finding that the beacon actually contains a very precise time can go a long way in performing synchronization. We may be able to send beacons whenever we want to synchronize a pair of nodes with more accurate sender's timing than mere MAC layer time stamping of packets. We use this mechanism to timestamp our own scheduling packets and make complete use of the hardware time stamping feature.

Chapter 4

Timers, Clocks and Synchronization

A TDMA based implementation depends heavily on the proper functioning of timers for its success. In this section, we describe the available choice of timers and then describe the mechanism of synchronization we follow.

4.1 Timers

Timers are an integral part of our TDMA implementation, since they maintain the slotting structure at each node. Linux kernel provides software timers that allow a minimum granularity of 1ms (we are not using real-time kernel). The Atheros chipset AR5212 uses a one-shot hardware timer and another periodic hardware timer for sending out periodic beacons. Both these timers are used by the Atheros proprietary Hardware Abstraction Layer for configuring beacons. After the HAL code was released [3], we could directly configure the hardware timers to cause interrupts at specific time intervals. The one-shot timer has a granularity of 128μs and the periodic timer has a granularity of 1ms.

With this given hardware, we have a choice of using either the hardware or software timer. Our decision depends on which timer provides us the best precision and accuracy at both low and high load conditions. In order to characterize the hardware timer, we setup a node to trigger periodically at every 100ms. The node was subjected to low-load conditions and then to very high number of RX interrupts (RX interrupt is caused when the node receives a packet). When the CPU has to process

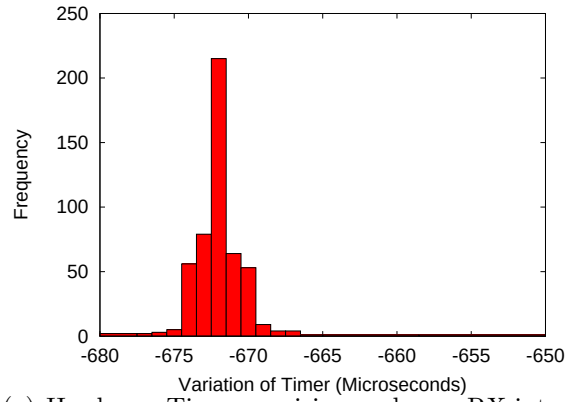
a large number of interrupts, it is possible that the hardware timer may not perform well, since the hardware timers also cause interrupts. We observe in Figure 4.1(a) that hardware timer is very precise with RX interrupts disabled, but is *unacceptable* when RX interrupts are enabled as seen in Figure 4.1(b). Note that the x-axis in the first figure shows difference between when the timer should ideally trigger and when it actually triggered; i.e., the *accuracy* of the timer. Therefore, the negative values indicate that the timer always triggered earlier than intended. The closer this value is to zero, the better. In Figure 4.1(b), the timer triggered 500 times periodically at every 100ms, and at each such instance, the difference between the trigger time and that of the previous trigger is recorded. The y-axis shows the variation of this recorded time. The x-axis shows each of the 500 instances.

The software timer on the other hand performs well in both high load and low load cases. As seen in Figure 4.2 the variation of the software timer is under 5μs. Also, the accuracy of these timers is better than the hardware timer being very close to zero. Hence we dropped the idea of using the hardware timer and continued with the 1ms software timer.

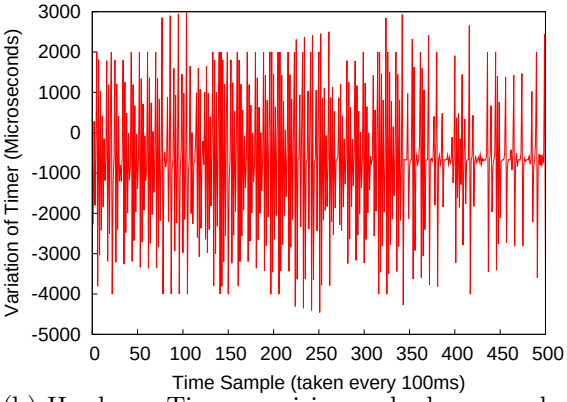
A slightly better performance may be possible by either using a real-time kernel or a hardware tuned for time slotting. In our implementation, we do not require a node to cause an interrupt at each slot. The interrupt is necessary only when it is the node's transmission slot. This optimization radically reduces the load on the timer system and improves the robustness of the implementation on our hardware.

4.2 Clock Drift

An important aspect in time synchronization is clock drift. Once two clocks are synchronized, they are expected to remain so. However, due to differences in their hardware caused by a multitude of factors such as environment, manufacturing processes and material purity, clocks tick at slightly different rates. Usually, this difference in rate is a few ppm (parts per million) and becomes important only when it interferes with the expected accuracy of the timers. In a TDMA protocol that depends on precise time slots, with each slot being only a few milliseconds wide, clock drift can quickly deteriorate performance. Clock drift calls for periodic resynchronization of nodes. How often to resynchronize depends on the worst case clock drifts observed on a particular hardware.



(a) Hardware Timer precision under no RX interrupts



(b) Hardware Timer precision under large number of RX interrupts

Figure 4.1: Performance of hardware timers under no load and under lot of RX interrupts.

The Atheros card has a 1μs granularity clock. The value is accessible through a call to the `ath_hal_gettsf64()` function. We setup two nodes as receivers and one node as a packet sender. Each receiver records its own time value when a packet from the sender is received. Comparing the difference between consecutive such packet receptions, we calculated the drift between the pair of cards. We observed clock drift to vary between different pairs of Atheros cards from a few microseconds to less than 25μs per second as shown in Table 4.1. The clock drift must be accommodated in the guard band in addition to the timer inaccuracies.

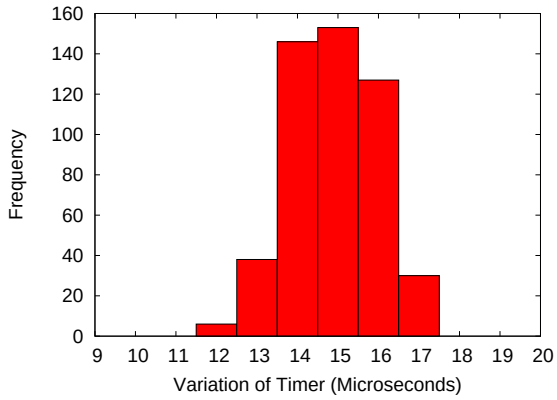


Figure 4.2: Software timer performance is not affected by increased RX interrupts.

Table 4.1: Clock drift for different pairs of cards in μs/s. Negative values indicate that the difference between the offset keeps on decreasing.

Card Pair	1-5	1-6	1-3	6-3	5-3
Average	-2.78	9.72	4.07	-1.41	13.24
Std Dev	0.42	0.47	1.03	0.5	1.51

4.3 Hardware Timestamp

When a packet is enabled for sending by the madwifi driver, it does not immediately leave the card. We need the exact time when a packet was sent for proper node synchronization. Details of node synchronization are given in Section 4.4. To get a packet time stamped by the hardware, we have to fool the hardware into believing that the packet being sent is a beacon packet. Beacons are time stamped with a 64-bit one microsecond granularity clock value at byte 24-31 of the beacon packet (counting starting from 0). This mechanism can be employed by calling the `ath_hal_setuptxdesc()` function with the value `HAL_PKT_TYPE_BEACON` ORed with the `flag` parameter.

We require only the schedule packets to be time stamped by the hardware. We create and send these special packets from the MAC layer itself, and hence can alter the flag without affecting non-schedule packets. All other packets coming from the network layer come through the function `ath_hardstart()` where we append a data header to these packets. The schedule packet is prepared at the MAC layer using a function `fractel_ath_mgtstart()` which is very similar to the `ath_mgtstart()` function. We append the schedule header before we reach the `ath_tx_startraw()`

function and thus, can always treat data packets differently from the schedule packets. The hardware also stamps a sequence number at byte 22-23. We have disabled hardware sequence numbering by carefully choosing the value of the second byte of the packet structure to set the `RETRY` flag. Only a few values of this byte disable the sequence numbering which otherwise modifies the bytes 22-23 of all packets.

4.4 Synchronization

All schedule packets are sent with the `HAL_PKT_TYPE_BEACON` flag set, as described in Section 4.3, and the hardware timestamp is used by the receiving nodes to accurately synchronize with the global time. In addition to the hardware timestamp (tx_hw_ts), the schedule also contains the sender's offset (tx_offset) with the global time and the global time when the current control slot ideally started (slot_start). Using this information along with the receive timestamp (rx_ts) and the slot interval (slot_interval) each node calculates its offset from the global time and the time of the next slot as per equation 4.1 and 4.2 respectively. Figure 4.3 shows the same equations graphically.

$$offset = rx\_ts - (hw\_ts - tx\_offset) \tag{4.1}$$

$$next\_slot\_time = slot\_start + slot\_interval \tag{4.2}$$

The packet is hardware time stamped *while* the packet is being sent out on air, whereas, the receiver timestamps the packet *after* the complete packet has been received. Hence, we miss out on the transmit time of the packet but since it can be calculated, it can be easily compensated.

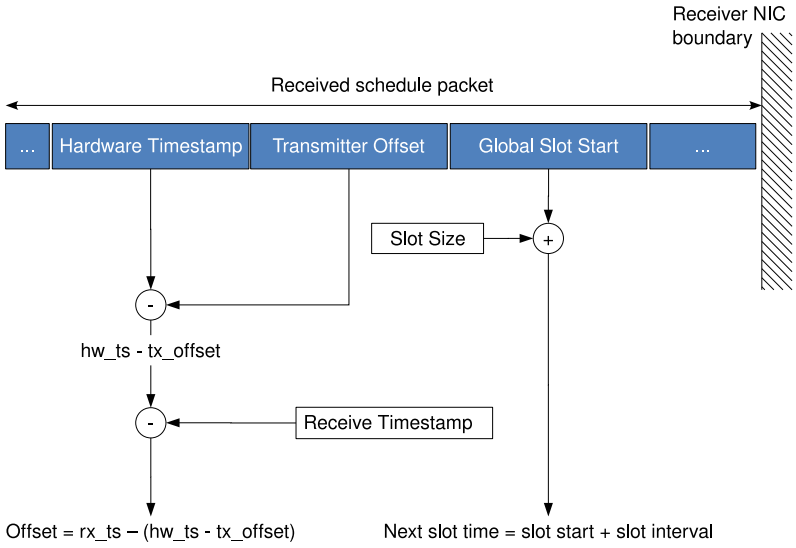


Figure 4.3: A node calculate its offset from the global time and the time of the next slot after receiving a schedule packet.

Chapter 5

TDMA Protocol and Synchronization

In this chapter, we briefly describe our TDMA protocol and how the synchronization described in Chapter 4 fits in the big picture. We note that the work on the TDMA protocol is jointly done with Nirav Uchat.

The randomness inherent in CSMA based protocols for wireless access makes it difficult to support QoS for real-time traffic on multihop wireless links. On the contrary, a centralized TDMA protocol, may ensure that all flows allowed in the network can be sustained in terms of QoS guarantees. In the subsequent sections we describe in detail our TDMA protocol for multihop communication.

5.1 Network Topology

We envision the network to be a tree topology with a fully provisioned root node that decides the schedule, permits flows and controls admission to the network. The root node also periodically publishes the routing tree. All the nodes in the network are synchronized with the root node and follow the schedule disseminated by it. They receive the schedule strictly from their own assigned parents and broadcast the schedule when their turn comes. Nodes join the network by first listening to the schedule frames, thus getting synchronized with the network, and then requesting the root node to allocate it a place in the routing tree. On receiving the routing tree, a new node seeing its ID in the tree understands that its *node join* request has been granted. Other nodes that are already present in the network update their routing

entries, if required. Each routing entry indicates a parent-child relationship. These entries together indicate the entire network topology that must be used by nodes to receive schedules from their own parents.

5.2 Time Slots

The smallest interval of addressable time is known as a slot. A fixed number of slots comprise a logical frame. The number of slots in a frame is fixed for a network, but, in general, is a configurable parameter. There are three types of slots in a frame and their positions and numbers are fixed for a network. Each frame contains a few control slots, a few contention slots and many more data slots. Figure 5.1 shows the three parts of a frame and how the slots are numbered.

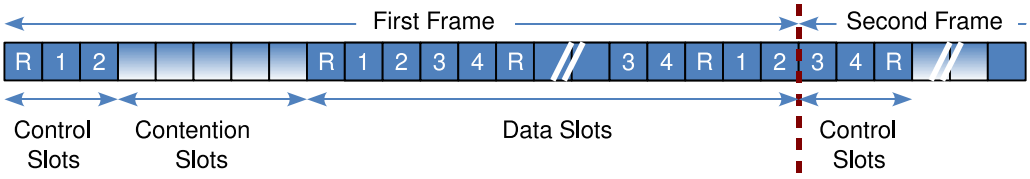


Figure 5.1: Components of a Frame

All nodes in the network are kept synchronized by sending special packets in the **control slots** in a systematic manner. Once synchronized, each node knows when the next slot starts as described in Section 4.4. The node will then start a software timer for triggering at the next slot start boundary. When the timer triggers, the actions taken by a node depend on whether the current slot is a control slot, a contention slot or a data slot. The accuracy of the timer is important for correct working of the slots.

In addition to synchronization, the control slots are used to send scheduling and routing tree information. The number of control slots in a frame, the frame length and the depth of the network determine the time required to propagate the schedule to all nodes. The number of control slots in each frame are constant, but they are repeatedly numbered from 0 (shown as R in Figure 5.1) to $n - 1$ where n is the number of nodes in the network and can span over many frames. Referring to the example in the Figure 5.1, given three control slots in each frame, and five nodes in the topology, first three nodes send their schedules in the first frame and the remaining two send their schedules in the consecutive frame. After the second slot in the second

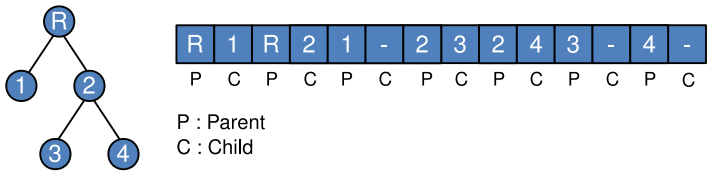


Figure 5.2: Structure of the routing tree communicated through a schedule packet.

frame, the schedule transmission opportunity rotates again to the root node. Each node transmits the schedule packet in a control slot determined by its position in the routing tree. A node transmits the schedule at a control slot number equal to the number of unique parents before it in the routing tree.

The **data slots** are similarly numbered from 0 to a maximum number given in the schedule. Nodes transmit data in the data slots according to the schedule sent in control slots. The **contention slots** are used for sending node join and bandwidth requests from non-root nodes to the root node. Nodes do not own these slots; they have to contend for air time.

5.3 Frame Format

All packets sent on air, have a custom header attached to them depending on the type of the packet. Schedule packets are constructed in the driver by the root node. It consists of a schedule header, a (possibly zero) number of scheduling elements and (possibly null) routing tree information. The schedule header has synchronization information and has the number of scheduling elements and routing tree elements contained in the packet. A scheduling element contains the transmitter, receiver and flowid for a data slot. Packets pertaining to different flows flowing through a node must be distinguished from each other so as to provide QoS guarantees. The flowid serves this purpose. All scheduling elements together describe the path of all data flows in the network. The routing tree information is simply a parent-child relationship described linearly. Each non-root node must appear at least once as a child node in this tree. A node may be a parent for multiple other nodes. Figure 5.2 shows an example topology and its routing tree. This centralized routing facilitates the root node to keep complete control over the bandwidth usage in the entire network enabling QoS guarantees.

Data packets are attached a header that help in routing the packets. In addition to the next hop and end-to-end source and destination fields, it also has the flowid field.

This field enables a relay node to keep the received packet under a separate queue for each flow-destination pair. The flowid field is particularly useful so that different data streams passing through the same node can be given different bandwidths, which helps in ensuring that we meet the QoS requirements of each flow.

5.4 Throughput Measurements

The synchronization module has facilitated the implementation of the above mentioned TDMA system. A testimony to the correct functioning of the synchronization framework can be obtained by checking the throughput obtained by the TDMA system built over this framework. Nirav has implemented the TDMA system and performed extensive experiments to check the obtainable TCP and UDP throughput. We present here the theoretical throughput calculations and then show that the practically observed throughput is very close to the theoretical value. Table 5.1 shows the transmit time for the various parts of a packet at 54Mbps.

Table 5.1: Time taken to transmit various portions of the packet at 54Mbps

Description	Bytes	Time (μs)
UDP Payload	1470	217.77
UDP Header	8	1.185
IP Header	20	2.962
Ethernet Header	14	2.074
CRC Trailer	4	0.592
Fractel Data Header	32	4.740
PLCP Header	-	20.444
Total	-	249.767

The calculations for theoretical throughput are shown for a 5-node linear topology. All nodes are set to transmit at 54Mbps and can transmit only in their own transmission slots. With five nodes transmitting, we use 87 of the 92 available data slots[1] in a round-robin fashion, so that each node gets $87/5 = 17.4$ slots per frame. The number of packets sent in each slot depends on the slot size and the size of the packet. Equation 5.1 calculates the theoretical throughput for the 4-hop case with the slot size of 2ms and a 100μs guard band giving 87 slots per second to each node.

[1]We have 3 control slots, 5 contention slots and 92 data slots. The last x data slots in a frame are not used so that the control slot timer for the next frame is triggered precisely. x is equal to the number of nodes in the network.

Similar calculations are performed to derive the theoretical maximum throughput for any number of hops.

$$\begin{aligned} Transmit\,time &= 1900\mu s\,(2000 - 100\mu s\,guard\,band) \\ Packets/slot &= 1900/249.767 = \lfloor 7.607 \rfloor \\ Packets/sec &= (\#ofslots/sec) * (packets/slot) \\ &= 87 * 7 = 609 \\ Throughput &= 609 * 1470 * 8/(10^6) \\ &= 7.16Mbps \end{aligned} \tag{5.1}$$

We now present the comparison between theoretical and practically observed values for 1 hop and 4 hop case for different slot sizes in Table 5.2. We note that the slightly lower UDP throughput for the 4 hop case may be due to wireless losses, which were about 0.5% during our experiments.

Table 5.2: Comparison of theoretical and practical UDP throughput

	1 Hop		4 Hop	
Slot Size	Theoretical	Observed	Theoretical	Observed
1	15.9	15.9	6.13	5.97
2	18.5	18.5	7.16	6.95
3	19.4	19.4	7.50	7.28
4	19.8	19.8	7.67	7.47
5	20.1	19.9	7.77	7.55
10	20.6	20.4	7.98	7.74

Any discrepancy in the synchronization module would have caused the UDP throughput to suffer due to misalignment of slots. The observation that the theoretical throughput is also obtainable practically bolsters our confidence in the synchronization module.

Chapter 6

Conclusion - Synchronization

The FRACTEL project as it stands currently provides a framework for implementing a complete TDMA protocol. Currently, we use software timers for maintaining the slotting structure crucial in the TDMA implementation. It is possible to have a custom made hardware that provides much more accurate timers. The slot size we use is limited by the granularity of the kernel timers. At 54Mbps, a 1500byte frame takes only about 228μs to transmit. With a slot size of approximately that granularity, we could ensure very good throughput for bidirectional TCP connections. We require microsecond granularity timers for unleashing the full potential of the TDMA implementation.

The working of the multihop schedule dissemination and synchronization has been proved to work as desired through extensive experimentation on a 5-hop linear topology. The details of these experiments are part of Nirav Uchat's thesis, and are not repeated here. We expect the same setup to extend to any number of hops. In fact, the setup can be directly used for providing a back-haul long distance link to remote areas. The QoS aspects and the flow management aspects are scheduler's concerns, and we have not delved into these. However, our platform is ready for outdoor experimentation as well as plugging in of various scheduling algorithms.

Chapter 7

Introduction - Esense

Devices of the same kind can communicate with each other because they can understand the bit encoding used by each other. However, many different standards have evolved which use radically different encodings in the same frequency spectrum, making cross-standard communication impossible. Nevertheless, since all these devices use the same frequency band they are capable of sensing change in energy pattern on the medium. An example is the ISM band which is used by 802.11 Wifi devices, 802.15.4 Zigbee devices, cordless phones and by Bluetooth devices. When such devices are used in each other's vicinity, they will disregard signals from other devices as noise. They do not derive any information from a communication between devices of any other standard. However, since all these devices can perform carrier-sensing, it may be possible for a sender to send a specific energy pattern on air and another device to decipher it.

In this part of the thesis, we investigate if such communication between nodes using fundamentally different physical layer encoding but the same frequency spectrum is indeed possible. Though the concept can be applied in a general setting, we use two devices–802.11 wifi card and 802.15.4 tmote sky–to demonstrate a mechanism of using energy patterns for meaningful communication between such devices. We first motivate why such communication may be required and then explain our approach for facilitating such communication.

7.1 Usage Scenarios

One may wonder why would such devices, completely diverse in their purpose, require to communicate with each other. We present below three scenarios where the energy sensing concept could be employed.

7.1.1 Facilitating Coexistence

With the high proliferation of digital devices, we may come across many places where more than one type of devices use the same frequency band. For example a home may easily have a wifi network for internet connectivity and a sensor network to control electronic gadgets. Both these networks work in the same frequency band and interfere with each other. If however, the wifi nodes could communicate with the sensor nodes and reserve a time slot for the sensor nodes, during which the wifi nodes will not transmit, we would have an interference free communication window for both type of devices. Wifi access points can define a contention free period that can be used by the access point to its discretion. This period may be advertised to the sensor nodes through transmission of a special energy signal.

7.1.2 Energy Management

In a wireless mesh network, many nodes may be used only sparingly. Whenever a wireless node is not in use, it should ideally enter into sleep states to conserve power. However, other nodes will need some mechanism to wake it up when packets arrive for it. The 802.11 PSM (Power Save Mode) allows nodes to sleep between two beacons. The nodes are expected to wake up at the beacon time, when the access point would broadcast a map of all nodes for which the access point has any pending data. Except for those nodes, others may go back to sleep. This protocol may cause excessive wakeups for devices that are rarely used.

We suggest using an out of band low power radio that stays awake while the main wifi node sleeps and saves power. Any node wishing to wake up a sleeping node sends the special energy pattern which the low power radio detects and wakes up the main node. The low power radio could be the motes used in sensor networks since their power consumption is extremely low compared to wifi devices.

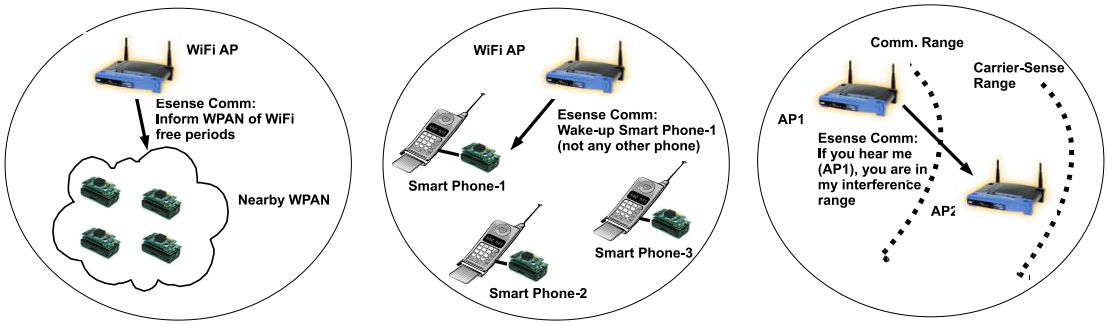


Figure 7.1: Esense application in three example scenarios

7.1.3 Interference Mapping

With large number of wifi access points being deployed in office and home setting, the problem of interference between two access points also aggravates. A common solution is to use different channels for adjoining access points. However, there are never sufficient number of orthogonal channels, and administrations may want campus wide networks to use only a single channel keeping other channels free for adhoc use. In any case, the interference range of wireless devices is much more than its transmission range. This means a far away access point may still be interfering with the network even if one cannot capture and decode any frames from that access point. In such cases, it becomes extremely difficult to pin-point the node causing the interference.

However, if the access points are made to transmit unique energy signatures along with their beacons, such energy pattern could be detected by interference mapper and the interfering node could be identified. Note that the contents of a packet can be deciphered only if the receiver is within the transmission range of the transmitter. However, energy patterns can be decoded as long as the signal is strong enough for the interference mapper to detect it.

These three problems and solution approaches are illustrated in Figure 7.1, and summarized in Table 7.1.

7.2 Challenges and Our Approach

We enable communication between devices by constructing an alphabet set based on different durations of energy bursts. If both the underlying networks are generally

Table 7.1: Esense communication details for the three example scenarios

Scenario	*Sender*	*Intended Receiver*	*Possible Message Content*
Coordinated Coexistence	WiFi Node	All nodes (or Master) of a given WPAN	Identity of the WPAN, next CFP start time (relative to reception of this message)
Energy Savings	WiFi Node	Secondary radio of the WiFi node to wake-up	Identity of the secondary radio (or receiver WiFi)
Interference Map	WiFi Node	Broadcast	Identity of the sender WiFi node

seeing traffic, we face the challenge of being able to distinguish the energy bursts we use for energy based sensing (henceforth referred to as Esense packets) from those normally exchanged (henceforth referred to as regular packets) in either networks. This is highly dependent on what the two underlying networks are. We have discussed this issue of distinguishing the packets in detail in this thesis. We have enabled Esense communication from an 802.11 device to an 802.15.4 device. This unidirectional communication is directly applicable to the first two of the scenarios we described where the sender is a 802.11 device and the receiver is a 802.15.4 device. Even in the third scenario, one could use the 802.15.4 device as a receiver for determining interference, particularly since the 802.11 radios do not easily expose the carrier sensing ability.

Even though we have enabled only unidirectional communication between 802.11 and 802.15.4, the idea is general enough and can be employed for any devices that use and expose the carrier sensing functionality. For all standards other than the two we use, one will have to perform experiments similar to what we demonstrate here and arrive at a usable set of Esense packets.

Distinguishing regular packets from Esense packets is simplified if we somehow use energy bursts for Esense packets that are not used by the regular packets on a network. We studied wifi traces from multiple sources and found that the packet distribution in wifi networks is bimodal. This means there are a lot of packet sizes between the few most frequent packet sizes that are not used much by the underlying network. Similar studies need to be done for other types of networks to find out usable packet bursts in those domains. Although this seems promising because there is a large alphabet set available due to the bimodal distribution, there are certain practical challenges in accurately detecting the duration of an energy burst.

The 802.15.4 device we use as the receiver uses a CC2420 chip and exposes the CCA pin. However, the timer available on the device and the resolution at which

the pin can be polled creates a practical limitation on the measured energy burst length. We show that the reported energy burst length is within a margin of about 90μs (corresponding to 3 ticks on the 802.15.4 device). We address this challenge and show that a reasonable alphabet set can still be used for Esense communication.

We have validated that the 802.15.4 device does indeed detect Esense packets from an ongoing wifi communication. We increase the robustness of this detection by sending an Esense packet multiple times within a small time window and reporting an event when at least a few of these reach are detected at the receiver.

In the rest of this report, we first briefly look at the related work done in the three domains where our solution is applicable. We then establish that the frequency of lengths used by wifi networks is bimodal which allows us to use some packet lengths as esense packets. We then investigate the granularity of length detection and gap separation presented by the tmote sky 802.15.4 device. We then construct a set of esense packets and validate that the mote can detect the occurrence of an esense packet in most of the cases.

Chapter 8

Related Work

To our knowledge, the framework of communication through energy sensing is quite novel. We are unaware of any prior work in this space. We have given three example research domains in which our framework is applicable; there has been considerable prior work with respect to these. We present the same in this section.

Many different wireless standards such as IEEE 802.11b/g [4], IEEE 802.15.4 [5] and Bluetooth [2] have been developed to work in the license-free Industry, Scientific and Medical (ISM) band. There are many situations where these devices may be operated at the same place at the same time. Coexistence study in [25] has shown that 802.11 signals can cause severe interference with 802.15.4 devices causing packet loss rates up to 90%. A solution to this is to limit the transmit power of interfering devices, but such approach is not always feasible given that both the networks could be spread over multiple hops. Another common approach [21, 15] is to monitor interference and switch to another channel having no or minimum interference. Also, WPANs can use some channels that do not overlap with the 802.11 channels. However, there is a limited number of channels and this solution works only as long as there is another channel with lesser interference. Since 802.11 has larger bandwidth channels (22MHz) (and hence only three orthogonal channels) than 802.15.4 (5MHz), it may be difficult to get away from 802.11 interference.

Therefore, merit exists in being able to share the same channel using time division multiplexing and coordinating transmissions in the same channel. The concept we propose in this work, where we can communicate between devices with different PHY encodings can be used to facilitate this coordination between WLANs and WPANs.

Since WiFi based devices are typically considered more energy constrained than

wired devices, considerable work has been done in the are of energy conservation in WiFi devices. The IEEE 802.11 standard [4] provides a power save mode (PSM) wherein WiFi devices may wake up during the access point's beacon transmissions and check if they are supposed to receive any data. The nodes may go back to sleep if no data is intended for them. Building on this basic idea, [16] proposes that the nodes may not wake up every beacon interval, and may progressively sleep longer durations without causing much performance setback. The bounded slowdown mechanism is effective, but still causes unnecessary wake ups. The ideal case would be if the access point can dynamically cause the WiFi device to wakeup only when it has data for that device.

This dynamic wakeup scheme is particularly effective because as shown in [10], the power consumption of WiFi interfaces is substantial even when idle and using these power save modes. Given this high idle power consumption of WiFi, the work in [23] has considered the approach of totally shutting down the WiFi-interface and using a low-power secondary radio. A WiFi sender wishing to wake-up a WiFi-receiver does so, by using its secondary radio to communicate with the secondary radio of the receiver. This solution however suffers from the disadvantage of range-mismatch: WiFi covers a much larger area than the secondary radio. The short range of the secondary radio makes it necessary to place multiple intermediate proxies and presence servers.

The dynamic wakeup we mention here is similar to the approach in Wake-on-WLAN [18] where the authors completely shutdown the router along with the WiFi interface and use a secondary radio with very low power listening capabilities. The authors of [23] use the primary radio (WiFi) of the sender for sending a wakeup signal. This signal is received by the low power radio on the receiver side and this secondary device wakes up the entire WiFi router. However, the secondary radio causes the wakeup when it observes *any* wireless activity. Such mechanism can only work if there is no other WiFi activity in the vicinity, which is true for the rural setting discussed in [18] or when the links are point-to-point. However, we wish to provide a solution in which a particular node is woken up amidst other ongoing communications.

Cell2notify [10] approaches the range mismatch problem by using both the WiFi interface and the cellular radio. It allows WiFi devices to sleep while their cellular interface is still awake. When a VoIP call is made to such a sleeping WiFi device,

a cell phone call is established to the device which activates the WiFi radio and the VoIP call is completed. This solution is specific to VoIP smart phones and requires cellular network. It needs custom infrastructure called the cell2notify server which maps VoIP identifications and cell numbers. Since the server makes cell phone calls using a cellular network, it is possible that the cell phone operators may block the server ID.

In contrast to the above solutions, our Esense-based solution has the following advantages. First, it can work in a variety of settings: enterprise WiFi or long-distance community networks; whether the end-device WiFi interface is off or the entire end-device is off. Second, it does not suffer from the range mismatch problem since the secondary radio directly senses the WiFi-energy. This is possible because the receiver sensitivity of the secondary radio (-95dBm) is better than the necessary WiFi received signal strength at the end device (typically -90dBm @1 Mbps). Third, our solution needs software changes at the WiFi sender side and integration of a very low cost ($10), low power (60mW) secondary radio[1] at the receiver side. Specifically, it does not need any intermediate proxies or servers on the infrastructure side.

The third scenario in which our framework is applicable is in constructing the interference map of an 802.11 network. Prior work [20, 12, 19] in this domain rely on building the map by conducting individual and pair-wise broadcast measurements of order O(N^2), where N is the number of nodes in the network. [22] reduces the number of measurements to O(N) by considering only individual broadcast measurements and relying on a physical layer model to derive the delivery ratio/throughput of a link when it operates in conjunction with others.

One of the major drawbacks of these approaches is that it requires network down-time to conduct the experiments ([20] reports 28 hrs are needed for these experiments on a 22 node test bed). In a dynamic environment where the interference profiles can change over time (due to environmental conditions) this is a serious limitation. Further it requires careful coordination among the different nodes to conduct the experiments apart from requiring a global view of the network.

Our solution to this problem, on the contrary does not require any downtime. The operating nodes need to just periodically broadcast their identity (and possibly the traffic load) and any receiver node R can process this identity information to figure out the number of interfering nodes and at what energy level they interfere with R.

[1] CC2430 is a system on a chip that comes for under 10$ that can be used for this purpose.

This information can then be used to perform appropriate transmit power/channel allocation, or TDMA-scheduling. The interference map may also be coupled with the traffic load information to derive the throughput achievable at this node based on an analytical model. These derivations can help with routing, call admission control and other such decisions that depend on the current traffic conditions at a node.

Chapter 9

The Esense Communication Framework

We now present the overall Esense framework. We first develop the basic idea, then present practical limitations, and subsequently our approach to Esense given the limitations.

9.1 Esense: overall approach

In this section, we describe the basis for building the Esense framework while being oblivious to practical limitations of the hardware used. Later, we consider the practical aspects and alter our approach accordingly.

9.1.1 Choices for Esense

If the encoding of a packet can not be deciphered, either because the received signal strength is very low or because the receiving device does not understand the encoding, then a receiver can sense only the energy pattern on the channel. The information that can be associated with such an energy burst can come from one of three parameters: the duration of the burst, the amplitude of the burst and the gap between two such bursts. The intensity or the amplitude of the energy burst varies with distance and a large number of environmental factors. Control on the received energy amplitude is very difficult to achieve. Similarly, in the existence of many transmitting devices in a distributed setting, it is difficult to achieve fixed separation between bursts. In CSMA, for example, any station may transmit a packet after sensing the channel free

for a DIFS interval. One station cannot, in general, dictate the gap between energy bursts. The only useful parameter then, is the duration of the energy burst.

The duration of an energy burst can be controlled by sending a packet which has a transmission time equal to the desired duration of the burst. In this work, we develop a framework for communication based on varying the energy duration.

9.1.2 Alphabet set

Having decided that the best way of communication using energy bursts is the duration of the burst, we need to find out which energy burst lengths can be used. The set of such useful energy bursts will be called an alphabet set. It is possible to build a vocabulary over this set and create an infinite number of words. The minimum cardinality of this alphabet set required is two. However, the larger the alphabet set, the framework can be as much more efficient (in terms of number of bursts required to send a message across and the time required for the complete information transfer).

The efficiency of the framework depends both on what duration of energy bursts we choose for the alphabet set and the cardinality of the alphabet set. For example, in the energy management scenario described in Section 7.1.2, each node to be woken up requires a different message from the AP. If the number of WiFi receivers is smaller than the Esense alphabet set, a single Esense packet is sufficient to convey the message. However, if the number of nodes in the network is larger than the alphabet set, we need combinations of Esense packets to denote each node differently. Thus, it may be beneficial to have a larger alphabet set.

The total time taken for transmitting a message depends on the length of bursts we have in our alphabet set. Suppose we have two possible alphabet sets: $\alpha = \{100, 200\}$ and set $\beta = \{1000, 1100, 1200, 1300\}$. Now, transmitting a 8-bit word using set α will require 8 symbols whereas transmitting the same word using set β will require only 4 symbols. However, since each element is the duration of burst, 8 symbols from α will cause the duration of the entire word to be only $150 * 8 = 1200$ time units on an average. On the contrary, the 4 symbols from β will cause the duration of the word to be $1150 * 4 = 4600$ time units. Here, β has larger cardinality than α, and still, the time taken for transmitting a message will be less if we use the set α. This shows that we have to be careful in choosing the alphabet set, favoring smaller bursts over larger ones.

In a typical 802.11 network, we expect normal packets to be transmitted constantly

at various transmission rates. This means that a receiver will continuously observe some energy pattern on the channel. Hence there is a possibility that a receiver may confuse a regular packet for an Esense message. The Esense alphabet set then must be chosen in such a way that shall minimize the false positive rate–we must use those packet lengths which are least used by the operational network.

9.1.3 Packet sizes in WiFi traces

As a proof of concept of the Esense framework, we enable uni-directional communication from an 802.11 radio to an 802.15.4 radio. As noted in Section 7.1, this hardware choice applies in the three scenarios listed. Hence, we attempt to construct the alphabet set based on the communication patterns predominant in WiFi networks. There are a variety of WiFi traces available in the public domain [1]. We identify five representative traces to study the communication patterns; these are traces from various WiFi infrastructure mode deployments [1]. The five traces are: Cafeteria (Powells), Library, PSU-CSE department (all from [9]), OSDI Conference [8] and Stanford CSE department [7].

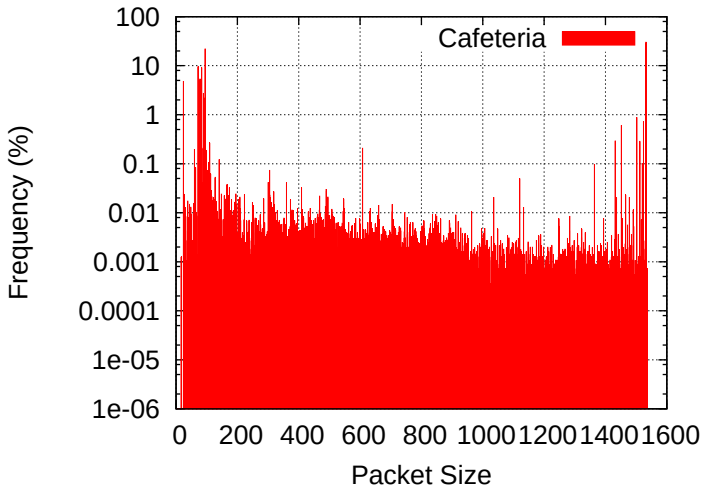


Figure 9.1: Packet size distribution of cafeteria trace

The packet size distribution for two of the five traces is depicted in Figure 9.1 and Figure 9.2. Note that the y-axis is log scale. Across all traces, the majority of the packets are either small packets ($<$ 140 bytes) corresponding to the 802.11 management packets such as ACKs and Beacons or are around 1500 byte packets

[1] We were not able to procure a trace of an operational mesh network since these networks are still mainly used for experimental purposes.

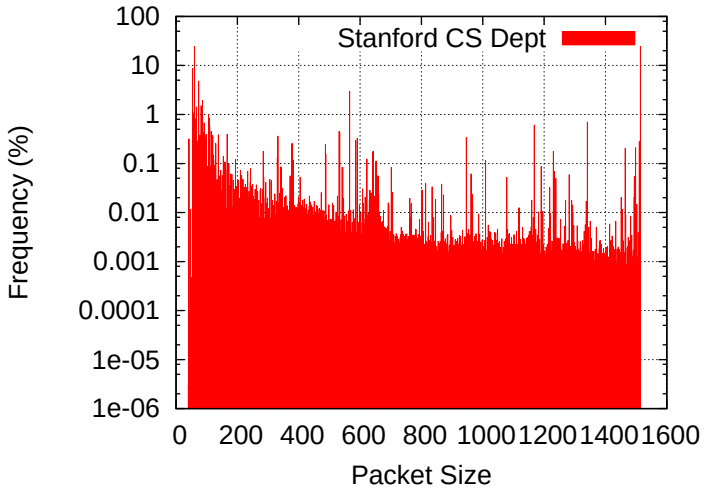


Figure 9.2: Packet size distribution of Stanford trace

corresponding to the Ethernet MTU. Such a bimodal distribution is a well know observation in the Internet; we confirm that the same applies in WiFi networks too.

We may thus use packet lengths in the central region that shows a low degree of occurrence for building the Esense alphabet. In addition, we may also use packet lengths larger than 1500 bytes since they are allowed in the WiFi MAC layer.

9.1.4 A possible approach for Esense

From the above observation, we can suggest a possible approach for Esense. The 802.15.4 receiver detects both regular as well as Esense packets. Hence, it needs a mechanism to distinguish between them and a mechanism to associate meaning to an Esense packet. Assuming that the 802.15.4 receiver can detect channel occupancy up to an accuracy of a byte, we can consider a simple solution. We may exclude all packet sizes that occur frequently and allocate the rest as instances of the alphabet set for Esense. We may fix a *threshold* percentage for exclusion of packets.

The higher we choose the threshold percentage, the larger is the alphabet set, since we would be excluding lesser number of packet sizes. However increasing the threshold percentage can result in high false positives, where some of the regular packets that were not excluded get interpreted as Esense packets. Also note that the alphabet set needs to be bounded at the maximum packet size in 802.11 which is 2304 bytes.

For an example threshold percentage of 1%, the number of packet sizes occurring more frequently than 1% is under 16 for all traces. This leaves a base alphabet set

of 2288 possible Esense packet sizes, which is quite large. We describe later how this large set is not available to us in practise.

9.1.5 Detecting Esense alphabets

With the ground laid, the receiving 802.15.4 node is expected to measure the duration of energy on the channel and map it to a corresponding packet size. If this packet size does not belong to the set of Esense alphabet, the packet is assumed to be a regular packet and rejected. Otherwise, the meaning of the packet length is interpreted as given in a correspondence chart. It is possible that some regular packets get detected as Esense packets because our threshold percentage is never 0%. We have characterized the number of false positives in our evaluation.

In this base scheme, we can have as many Esense messages as the size of the alphabet set. We can enhance this by building a vocabulary (sequence of alphabets) from the alphabet set; however, we leave this to future work.

9.2 Practical Limitations

Practically, the above described scheme faces two main challenges. (1) The 802.15.4 Esense receiver can only sense the energy burst duration and not the rate at which it is sent; mapping this to a packet length is not straightforward since the WiFi sender could be sending at any of the multiple possible data rates. (2) The accuracy of commodity IEEE 802.15.4 radio hardware in detecting channel occupancy is limited. Further, this detection of energy burst is made even more difficult because 802.11 supports multiple rates, with 802.11g mode supporting very high data rates, up to 54 Mbps. Also, the inter-packet gaps in an operational system are unpredictable due to the very nature of the CSMA/CA protocol. These gaps could be as small as 50μs for 802.11b and 28μs for 802.11g.

The first issue implies that we cannot use packet sizes as it is to encode Esense alphabets. The second issue, that of inaccuracy in channel occupancy estimation, may arise due to two reasons: (a) finite granularity of time measurement, and (b) finite sampling granularity. The inaccuracy issue manifests in many ways, as illustrated in Figure 9.3. First, when the channel occupancy is say x μs, the radio hardware may indicate a number greater or smaller than this. Second, the inherent inaccuracy combined with high data rate operation may mean that an entire packet is missed by

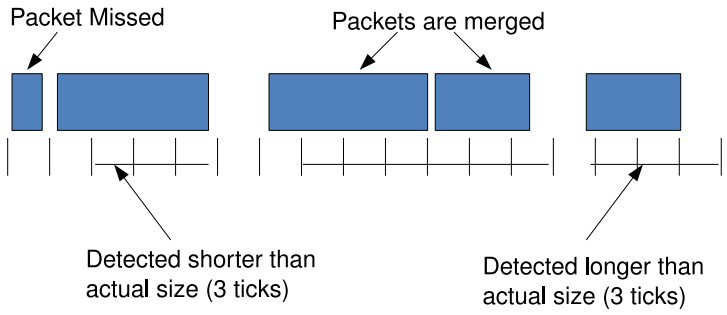


Figure 9.3: Illustration of 802.15.4 hardware limitations

the Esense receiver. For example, when operating at say 54 Mbps, a 40 byte packet's transmission time is roughly 31 μs (including PLCP overhead). The 802.15.4 radio may not even be able to detect this packet. Third, if the time separation between two adjacent packets is very small (this interval is dictated by the random back off of the 802.11 standard), the radio may club two or more packets together and interpret it as one large packet.

We characterize the accuracy of 802.15.4 radio in detail in Section 10. For now, we outline the procedure we follow to address the above limitations.

9.3 Practical Alphabet Extraction Algorithm

We address the issue of multiple 802.11 data rates, by using energy burst lengths directly as Esense alphabets, instead of using packet sizes as alphabets [2]. We take a measurement based approach to determining the base alphabet for Esense.

The hardware we employ is the CC2420-based Tmote Sky. It uses a relatively impoverished microcontroller, the MSP430, but it is good for low-power operation. The microcontroller is only 8 MHz, and has 40 KB of ROM, and 10 KB of RAM.

The time measurement granularity on this platform is 1/32 KHz, or approximately 30.5 μs, since it uses a 32 KHz oscillator. We shall refer to this as one clock *tick* henceforth. In the Esense prototype based on this platform, we poll the CCA pin of the CC2420 chip "continuously". We note that since the microcontroller is slow, the polling has a finite time granularity.

We factor in the limitations of the commodity 802.15.4 hardware by actually using

[2]The 802.11 sender needs to figure out the packet size and rate at which to send this packet, such that the resulting channel activity results in the correct energy burst length.

the hardware during the process of Esense alphabet construction itself. We measure the channel occupancy when a real WiFi node transmits packets of sizes as dictated by the traces and packet intervals as dictated by the standard. For a given packet, the 802.15.4 hardware measures the channel occupancy as a run-length of clock ticks. Much like what we have done earlier for packet sizes, we calculate the frequency of occurrence of a given run length. We extract our alphabets from the complement set of run lengths that exceed a threshold percentage of frequency. This process needs to address the following four issues.

1. What is the run length we should use to bound the complement set?
2. Do we consider all run lengths in the complement set as alphabets?
3. How do we handle false negatives resulting from merging of packets by the low resolution hardware?
4. How do we handle false positives resulting from regular packets being confused as Esense packets?

We address the above issues in the following manner.

9.3.1 Bounding the complement set

The maximum run length of energy burst available to us is the MTU of the standard. However, at different transmit rates, this length may vary considerably. For example, a 1500 byte packet transmitted at 54 Mbps will occupy about 8 ticks while a 2304 byte packet will occupy about 12 ticks which leaves us with a very limited set of 4 ticks that can be potentially be used as Esense packets assuming that all these packet sizes occur less than the threshold percentage times in the normal operations of 802.11. At the same time, at lower data rates, we may be left with a much larger set of available bursts.

We can overcome this limitation by considering the lowest rate at which an 802.11 node can operate. If an 802.11 node has been configured to operate in the g mode only, we bound the alphabet by the run length that occurs when a 2304 byte packet is sent at 6 Mbps (lowest data rate in the g mode). If the 802.11 node has been configured to operate in the b mode, we bound the alphabet by the run length that

occurs when a 2304 byte packet is sent at 1 Mbps (lowest data rate in the b mode) [3]. By doing this, we are effectively requiring that the Esense packets always be sent at the lowest available rate irrespective of the rate used for normal data transmission.

9.3.2 Building-in margins between alphabets

We would have liked to consider all the run lengths in the complement set as alphabet instances, but this will not account for the practical measurement limitations of the 802.15.4 hardware. A particular detected run-length could potentially correspond to a window of actual packet lengths. Conversely, it is also possible that the same packet length is measured as different run lengths at different times. To account for this, in our algorithms, we choose a *margin*, and ensure that for any run length chosen as an alphabet, there is a gap of at least the margin number of ticks between this alphabet and any adjacent alphabet. This gap is also maintained between an alphabet and a regular packet whose frequency exceeds the threshold percentage of frequency. The actual value of the margin is determined by 802.15.4 hardware characterization.

Running this algorithm on a part of a trace as seen by the 802.15.4 hardware, we choose only a subset of the complement set of run lengths. Once we determine the alphabet set, we map the run length to appropriate packet size considering the lowest data rate of operation possible in that setting (since we have stipulated that Esense packets are always sent at the lowest data rate).

9.3.3 Handling false positives and negatives

Problems 3 and 4 have a common solution. To reduce the occurrence of false positives and negatives, we consider the use of repetitions. We essentially send the Esense alphabet (packet) multiple times in a small time-window. And at the receiver, we deem that an Esense alphabet (packet) has been received only if it detects at least some threshold number of instances of the Esense alphabet within the chosen time window.

To summarize our design, a WiFi node wishing to send an Esense packet does so, by selecting a run length (and therefore a corresponding packet size at the lowest

[3] A 802.11 node in g mode can potentially transmit at 1 Mbps. Doing so can increase its alphabet space further but we make the worst case assumption that its configuration prevents it from doing the same.

rate of operation) in the alphabet that corresponds to the message it wishes to communicate. It then sends this packet multiple times at the lowest possible data rate. The 802.15.4 receiver concludes it has seen an Esense packet if it observes a channel occupancy run length corresponding to an alphabet that occurs at least a specified number of times within a given time window.

Chapter 10

Evaluation

We first present our experimental setup, in Section 10.1. We then describe the accuracy characterization of the 802.15.4 platform in Section 10.2. This then leads us to experimentally determining the Esense alphabet set for 802.11 to 802.15.4 communication in Section 10.3. We finally validate the chosen Esense alphabet set and evaluate the effectiveness of Esense communication in Section 10.4.

10.1 Experimental Methodology

In all of our experiments, we use a WiFi radio setup as the sender and an 802.15.4 radio (on the Tmote Sky platform) as the 802.15.4 receiver. For the 802.11 sender, we use a laptop equipped with a 802.11b/g WiFi card (with the Linux open-source madwifi driver) as the transmitter. We term the 802.15.4 receiver platform as a "mote", since this is the platform commonly used in many Wireless Sensor Network (WSN) applications.

The 802.15.4 receiver mote is connected to another laptop, which is used to log experimental data via the mote's serial interface. To reduce the memory requirement at the mote (mote has a limited memory of only 10 KB of RAM), we just log the time (clock tick) at which the channel state changed–either from free to busy or vice versa and send the log periodically to the laptop. The consolidated log can then be used to calculate the run length in clock ticks of the channel busy time.

For our experiments to characterize the accuracy of the mote in detecting channel busy time, we need very fine grained control over the spacing between adjacent packets. We achieve this control by modifying the madwifi driver. We disable back

off by ORing `HAL_TXQ_BACKOFF_DISABLE` with the `tqi_qflags` variable. We set the contention window to zero by setting `cwmin` and `cwmax` variables to zero. We then use the AIFS feature (part of the 802.11e standard) to control the spacing between packets. For a given AIFS, the spacing between two packets is given by the formula $(10 + AIFS \times 20)\mu s$, where AIFS can take on values starting from 0. Hence we can achieve spacing as low as 10μs between the packets. We verified that this mechanism works correctly by connecting the WiFi card to a spectrum analyzer. We observed values as given by the formula with an error under 2μs. We note that we found this behavior to work with the specific driver and the Atheros cards we used. This may not be a general way of spacing packets with some inter-packet gap.

For our other experiments, we need to emulate WiFi activity as specified by the different traces. Now, the timing interval between packets as captured by the traces is not a reflection of what actually happens on the channel since the timers are software based and not very accurate [1]. Also, the packets in the trace could be originally sent by a number of different nodes which is very difficult to emulate. So, we just use the packet size distribution from the traces and consider that all these packets are backlogged at a single WiFi sender. The spacing between the packets is then dictated by the random back off employed by the 802.11 standard. This is a worst case assumption as far as our setting is considered; in reality the spacing between the packets will be more than what we consider, and any extra inter-packet spacing can only help the mote detect the packets better.

In the madwifi driver we queue packets in bunches of 500. A user level program reads the required lengths of the packets from the trace and feeds this length to the driver using a proc file. The driver then constructs a packet and queues it. After 500 such packets are received by the driver, it starts transmitting the packets and instructs the user level program to wait. Doing so in bunch of 500 packets ensures that the packet buffer does not overflow at the driver. Also, this gives a chance for the mote to copy its log to the computer. The mote has limited memory, so it is better to keep periodically logging its data to the PC than to overflow the mote's memory.

[1] While some trace files do provide high accuracy timing, not all of our traces do. To be consistent, we do not consider the timing information provided by the traces

10.2 Mote Accuracy Characterization

In order to characterize the accuracy of the mote, we perform two experiments. In the first experiment, we try to capture what the mote reports when the channel is busy for a specified duration. For this experiment, we send a given sized packet 5000 times, with inter-packet spacing set to a sufficiently large value of 50ms. We measure the channel occupancy as reported by the mote and compare it with the actual value. The actual value is measured using a spectrum analyzer.

Table 10.1: Mote characterization: margin of error

Busy Time	*Value1*	*Value2*	*Value3*
123 μs	122 ; 8.84%	152.5 ; 74.5%	183 ; 16.62%
785 μs	762.5 ; 1.7%	793 ; 63.84%	823.5 ; 34.46%
4710 μs	4697 ; 26.6%	4727.5 ; 67.38%	4758 ; 5.94%

Table 10.1 shows the findings of the mote when the actual channel occupancy duration was set to 123, 785 and 4710μs. The columns represent the first, second and third highest frequency components of the mote measurements. Note that the mote reports its findings in clock ticks. We multiply this with 30.5μs to arrive at the numbers in the table.

As can be seen from the table, the mote often reports a value higher than the actual channel occupancy. And the maximum margin of error is about 2 clock ticks (i.e., 61μs). We attribute the inaccuracies to the coarse time granularity as well as the system delays involved in polling the CCA pin.

The goal of the second experiment in the mote characterization is to determine the spacing between packets at which a mote can successfully distinguish one packet from the other. For this experiment, we send a periodic stream of packets, all of the same size, separated by a given interval. We choose a packet size that results in channel occupancy time of 785μs. We consider different spacing: 10, 30, 50, 70 and 90 μs by appropriately varying the AIFS value. Figure 10.1, Figure 10.2 and Figure 10.3 present the frequency plot of various packet sizes detected by the mote for the 50μs, 70μs and 90μs inter-packet gaps respectively. Note that x-axis are different in the three cases.

When the inter-packet gap is small, the mote is unable to detect this gap and merges two or more packets to report one big packet. Note that whether it will be able to detect the gap or not depends on when the hardware actually polls the pin. So,

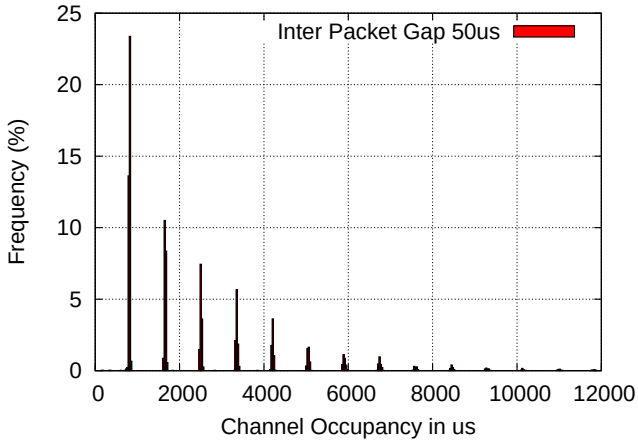


Figure 10.1: Channel occupancy as reported by the mote at 50μs inter-packet gap (actual channel occupancy: 785μs)

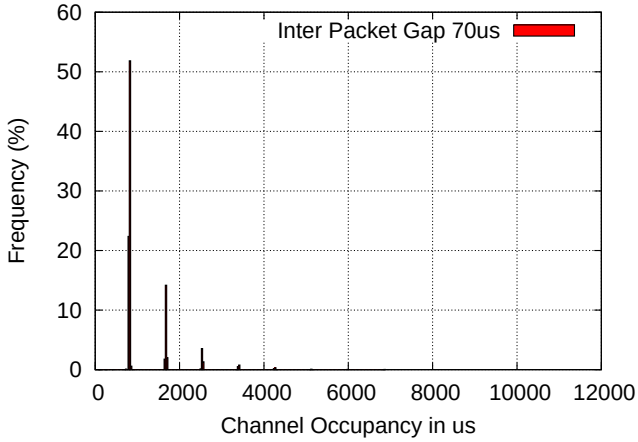


Figure 10.2: Channel occupancy as reported by the mote at 70μs inter-packet gap (actual channel occupancy: 785μs)

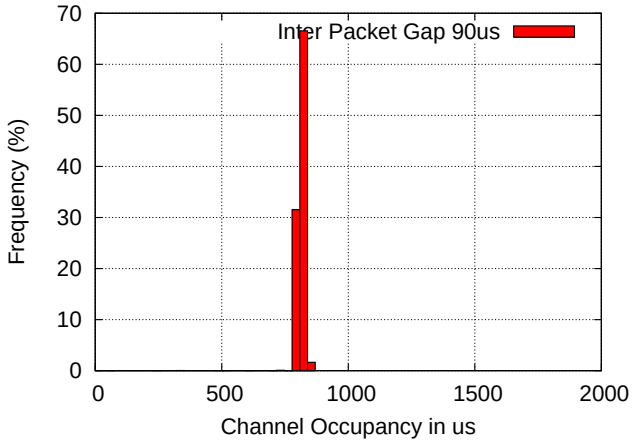


Figure 10.3: Channel occupancy as reported by the mote at 90μs inter-packet gap (actual channel occupancy: 785μs)

sometimes it can detect the gap and sometimes it may even merge multiple packets. The 50μs inter-packet gap is too small for the mote and it reports multiple packets as one. This is evident from the periodic spikes seen in the Figure 10.1. These spikes occur at multiples of the packet transmission time. The largest run length that we observe in this experiment corresponds to one where 15 packets got merged, though this occurs very rarely at 0.1%. At 70μs as shown in Figure 10.2, the largest run length we observe corresponds to 5 packets getting merged and this happens at a frequency of 0.5%. At 90μs the mote is able to clearly separate out the individual packet as is evident from the Figure 10.3.

According to the 802.11b standard, the minimum separation between the packets is at least 50μs. With backlogged queues, the average separation would be about $50 + 16 \times 20 = 370\mu s$ (assuming a CWMin of 32). So, in most cases the mote can clearly separate out the packets. With respect to 802.11g mode, if it has to support legacy 802.11b nodes, the same parameters as above apply. However it can also operate with a smaller slot time of 9μs (as opposed to 20μs in 802.11b). This then gives a minimum separation of 28μs and average separation of 172μs. So, the mote can still perform well in this setting. Of course, if we obtain or create a hardware that allows better precision at sensing energy, this shortcoming no longer exists. In essence, this does not hurt the concept. It may only hurt our experimentation with the given hardware limitations.

In summary, the mote reports channel occupancy time with a margin of error which is ± 2 clock ticks (i.e. $\pm 61\mu s$). And it can separate out the packets clearly only when they are separated by at least $90\mu s$.

10.3 Alphabet Extraction

The alphabet set has to be found out by running a part of the trace and acquiring the mote's readings of the energy burst duration. We consider snapshots of the traces, each snapshot consisting of 500 packets in the trace. For the CSE-PSU, Library, and Cafeteria traces (which were shorter), we took 20 different 500-packet snapshots from different positions in the trace, resulting in a total of 10,000 packets. And for the CSE-Stanford and OSDI traces (which were larger), we took 50 different snapshots, resulting in a total of 25,000 packets. We verified that the distribution of the extracted trace is similar to the original trace.

For a given extracted trace, the 802.11 sender generates packets of size as specified in the trace and sends them back-to-back. This is done using a proc file as explained in Section 10.1. On air, these packets get separated out by a gap as dictated by the random back off of 802.11 standard. Now, there is the question of what 802.11 data rate to use to send these packets. The traces unfortunately do not provide this information. Hence we consider a variety of data rates including random rates to emulate nodes that employ auto-rate adaptation. In fact, if the traces did provide this information, it would still be better to run the traces with different rates.

Figure 10.4 shows the findings of the mote for the Cafeteria trace at 4 different rates. Note that the y-axis is log scale and the x-axis range is different for the different rates.

We make three observations from the figure. One, the bimodal distribution which we saw in the packet lengths, is preserved in the run-length distributions, up to 18 Mbps and after that it starts becoming unimodal. This is because at high data rates, the shorter length packets (corresponding to beacons, acks, and management frames etc) are often not detected by the mote. Two, one can observe peaks at positions corresponding to multiples of the predominant run-lengths. This is due to the mote merging adjacent packets. This may not be as predominant in reality since we assume backlogged packets in our emulation.

Third, if we were to consider only run lengths with frequency greater than 1% for elimination, prior to alphabet extraction, we can see one chunk of run lengths available between the two predominant modes. This chunk starts to become smaller beyond 18 Mbps and disappears altogether at 54 Mbps. For the various data rates, there is another chunk of run lengths available to the right of the second highest frequency component. This chunk starts at 410 ticks @ 1 Mbps, 70 ticks @ 6 Mbps, 28 ticks @ 18 Mbps, and 19 ticks @ 54 Mbps. If we assume that all Esense alphabets are sent at 1 Mbps, the maximum packet size corresponds to a run length of about 610 ticks. So, there is considerable space available in all cases to construct alphabets in spite of the mote's limitations.

While we are presenting Figure 10.4 only for the Cafeteria trace, we plotted similar graphs for the other traces too, and observed similar findings.

Now, for the actual alphabet extraction, for each of the five traces, we considered six different data rates at which the packets were sent: 1, 11, 6, 18, 36, 54 Mbps. This gives a total of 30 run-length logs. For each log, we extract the alphabet based on the

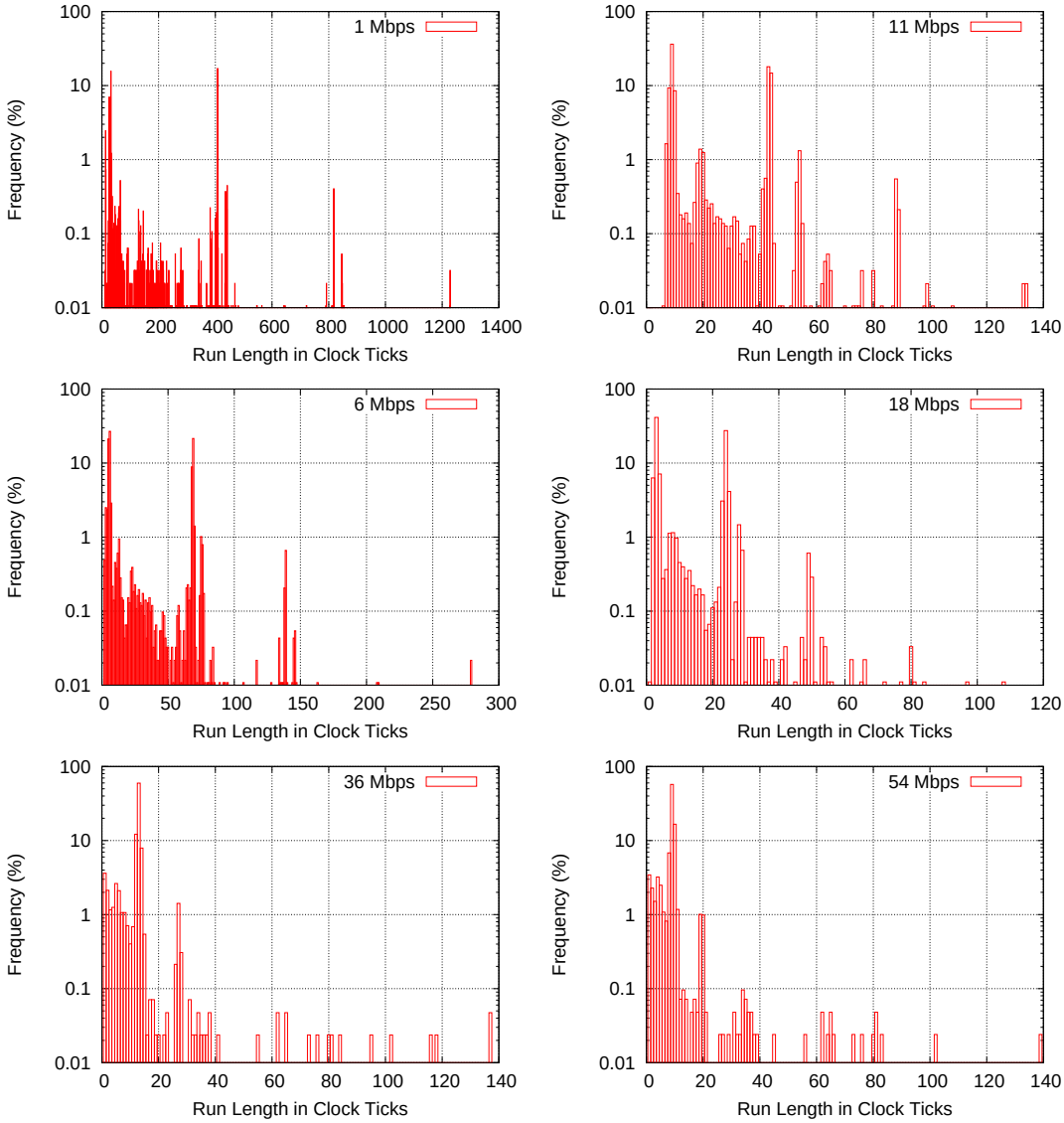


Figure 10.4: Cafeteria trace: Run Lengths at 1, 11, 6, 18, 36, 54 Mbps

algorithm explained in Section 9.3. The algorithm uses a margin of 2 ticks, obtained in our mote characterization experiments in Section 10.2. We use the value of 1% for the threshold frequency of occurrence, beyond which we eliminate a run-length from consideration for the Esense alphabet set. This threshold provides a good tradeoff between alphabet size and false-positive rate. From Figure 10.4, most run lengths below 1% occur at a very low frequency.

The algorithm takes as input another parameter: the mode of operation (b or g). This specifies the rate at which the Esense alphabet is sent. For the b and g mode, the alphabet is sent at 1 Mbps and 6 Mbps respectively. Note that the b mode should

not be interpreted to mean that the data rates are restricted to under 11 Mbps. It just means that it uses a data rate for *alphabets* that is available only in that mode.

Table 10.2 shows the size of the extracted alphabet set from the above-mentioned 30 logs: 5 traces along rows, 6 data rates along columns. For the 6, 18, 36, and 54 Mbps columns, the values without brackets correspond to the b mode of operation and values within correspond to the g mode of operation.

Table 10.2: Alphabet size

Trace	*1Mbps*	*11Mbps*	*6Mbps*	*18Mbps*	*36Mbps*	*54Mbps*	*All Rates*
CSE-PSU	109	115	116 (15)	118 (16)	119 (15)	118 (13)	107 (12)
Library	112	116	117 (15)	118 (14)	117 (15)	117 (14)	110 (12)
Cafeteria	115	115	118 (15)	118 (14)	117 (15)	118 (15)	110 (11)
CSE-Stanford	112	115	117 (14)	116 (14)	117 (15)	118 (14)	109 (12)
OSDI	110	117	113 (15)	118 (16)	119 (17)	119 (11)	110 (11)
All traces	104	113	112 (10)	115 (13)	116 (14)	117 (15)	100 (10)

In the table, we have a last row labeled "All traces", and a last column labeled "All rates". The last row corresponds to alphabet extraction for the case where we eliminated run-lengths which exceeded the threshold percentage of 1% in *any* of the traces (i.e. in at least one of the traces). Similarly, the last column corresponds to alphabet extraction where we eliminated run-lengths which exceeded the 1% frequency threshold in *any* of the six considered rates of operation. The last row and the last column thus correspond to conservative choices of the alphabet set; the last entry in the table thus represents the most conservative choice. (Conservative with respect to data rate implies: if we do not know the rate at which a high frequency regular packet size would be sent, eliminate all the run lengths corresponding to all the possible data rates).

We observe that the "All traces" row is only slightly different from the rows for the 5 traces. This implies that the run length distributions are more or less the same across the five traces. In fact, we observed in our algorithm output that the alphabet set itself is not very different across the various traces.

Similarly, we observe that the "All rates" column is only slightly different from the columns for the 6 different rates. This means that there are only a few run-lengths, which show a low frequency of occurrence at one rate, but show a high frequency of occurrence at another rate. This then means that rate adaptation will not significantly affect the size of the extracted alphabet set.

We observe in the table that across the six rates, as the data rate increases, the size of the alphabet set shows a slight increase. This is because at higher data rates, packet transmission times are much smaller and get clustered towards the shorter run length, leaving more empty space from which to extract alphabets.

As can be seen from the table, we get about 100 alphabets when alphabets are sent at 1 Mbps and about 10 when alphabets sent at 6 Mbps. This difference due to change of mode is because in g mode, the maximum run length that we can consider is only 100 ticks (corresponding to 2304 bytes sent at 6 Mbps), while in b mode this turns out to be 610 ticks.

We have performed the above experiments for different threshold percentages. At 10%, the resulting alphabet set was 108 for b mode and 13 for the g mode (corresponding to the "All traces" row and "All rates" column). At 0.1%, the alphabet set falls sharply to 60 and 3 respectively. This sharp decline is mainly due to the conservative operation across the traces. This is because quite a few run lengths in all the traces exceed the small threshold of 0.1%. And further very few of these "frequent" run lengths are common across the traces. This results in lot of elimination resulting in smaller alphabet set. The conservative operation across data rates (the "All rates" column) for any trace still yields high alphabet set: around 90 and 6 for b and g respectively.

In the above experiments, we have considered only a subset of the data rates supported by 802.11 b and g. For a given trace, there isn't much difference in the alphabet sets across the different rates. But for completeness we consider all data rates for the Cafeteria trace. Table 10.3 shows the results of this experiment. The results are very similar to what we obtained earlier. The conservative "All rates" entry now reduces the alphabet set size to 105 (6) since we are considering intersection across a much larger set.

The above results in terms of alphabet size could improve considerably if we worked with an 802.15.4 hardware that is more accurate[2].

10.4 Validation

We now validate our communication framework using the above extracted alphabet. We consider the alphabet set corresponding to the conservative calculation; i.e. the

[2]Note that in the process of building more accurate hardware, one should not lose the original low-cost and low-power advantages of 802.15.4.

Table 10.3: Cafeteria trace: all data rates

802.11b Data Rates			
1mbps	2Mbps	5.5Mbps	11Mbps
115	115	115	115
802.11g Data Rates			
6mbps	9Mbps	12Mbps	18Mbps
118(18)	118(15)	118(15)	118(14)
802.11g Data Rates			
24mbps	36Mbps	48Mbps	54Mbps
116(13)	117(15)	118(16)	118 (15)
All rates			
105 (6)			

set corresponding to the "All traces" row and "All rates" column. We consider both modes of operation i.e. the b and g compatible modes. For the b mode of operation, we consider two fixed data rates of 11 Mbps and 18 Mbps and one variable data rate. For the fixed data rates, all regular packets with the exception of management packets are sent at the respective rates. The management and the Esense packets are sent at 1 Mbps. The variable data rate is supposed to model nodes that employ rate adaptation. For this case, the regular packets (with the exception of management packets) gets sent at a rate randomly chosen from the set (1, 11, 6, 18, 36 and 54 Mbps). For the g mode of operation, we consider one fixed rate of 18 Mbps and one variable rate. The management and Esense packets in this mode are sent at 6 Mbps. The other regular packets are sent at 18 Mbps for the fixed rate case and randomly chosen from the set (6, 18, 36 and 54) for the variable data rate case.

Each of these traces is large enough so that we can take sixty 500-packet snapshots at different trace positions generating a total 30,000 packets trace. We choose the 60 snapshots such that they do not overlap with those we used for extraction of the alphabet in Section 10.3; our validation trace is distinct from the training trace. We then embed 250 random alphabets into this trace. While embedding an alphabet instance, we may repeat the same value multiple times within a small window of time. This repetition reduces the possibility of false positives due to regular packets seeming as if they are Esense packets. We consider four possible repetition values: 1, 3, 5, and 10. Different validation experiments are performed with these repetition counts. We refer to this repetition count as the *sender repetition count*, SRC. Practically in a CSMA network, it is possible that other regular packets squeeze between two instances of the Esense packet being transmitted. We emulate this by allowing a few

random number of packets (maximum of 5) to appear in between two instances of Esense packets.

For this validation, we make some changes in the way we supply packets to the madwifi driver. We still send packets on air in chunks, but each chunk ends at the transmission of an Esense alphabet SRC number of times. For example, if we have set the SRC to 3, a chunk ends after three instances of the current Esense alphabet are sent. We have modified the tinyOS code on the mote to report the packet lengths monitored if no packet is observed for a specific amount of time. Thus, after a chunk is transmitted by the wifi device, the mote reports back what it saw on air. We get 250 such reports pertaining to the 250 times that the Esense alphabet is sent. This modification makes it possible to check for false negatives. If the last few packets seen by the mote do not look like the Esense alphabet we expect to see, we report this as a false negative.

At the receiver, we measure the number of distinct alphabets detected. Since a given alphabet gets sent multiple times within a window, the receiver concludes that an alphabet is detected successfully if it observes the alphabet a specified number of times within the window. We term this expected repetition count as the receiver detection count, RDC. RDC can be equal to or less than the SRC.

Table 10.4 shows the number of false positives and false negatives that occur in the various cases. False positives correspond to the case where no alphabet was sent but the receiver detected one. False negatives correspond to the case where an alphabet was sent but was not detected. The first column in the table corresponds to the pair (SRC, RDC). We make the following observations from the table.

- For a given SRC, as the RDC is increased, it results in lesser number of false positives but higher number of false negatives. This trend is evident in the table. Higher the RDC, lesser the chance of a regular packet being mistaken for an Esense packet. A regular packet has to repeat in a small time window as many times as the RDC to be mistaken for an Esense packet. But this also means that we cannot properly detect alphabets since the mote can merge packets. Hence there is a fundamental trade-off in the choice of RDC.

- The overall false positives as well as negatives are quite small especially for values beyond (5,3) considering that there are 250 alphabets sent amidst 30,000 packets. Note that the false negatives should be compared relative to the alphabets sent and the false positives relative to the total packet count. The

Table 10.4: False positives and negatives: 250 Alphabets, 30,000 Regular Packets

	b Compatible Mode						g Compatible Mode			
Param	11Mbps		18Mbps		R-Adapt		18Mbps		R-Adapt	
(SRC, RDC)	FN	FP	FN	FP	FN	FP	FN	FP	FN	FP
Cafeteria Trace										
(1,1)	24	268	11	142	13	413	8	73	2	63
(3,1)	3	332	3	185	0	419	0	242	0	162
(3,2)	17	67	22	22	10	26	13	23	9	10
(5,2)	0	24	1	14	1	19	0	20	1	16
(5,3)	5	4	3	0	5	2	1	2	3	4
(10,4)	0	6	0	1	0	2	0	0	0	2
(10,5)	0	0	0	0	0	0	0	0	0	0
(10,6)	1	0	6	0	1	0	1	0	0	0
CSE-PSU Trace										
(1,1)	17	545	18	340	21	787	20	92	13	262
(3,1)	2	626	0	414	1	808	0	100	1	290
(3,2)	24	130	20	80	14	97	17	14	10	48
(5,2)	1	126	0	67	1	109	1	14	0	50
(5,3)	5	40	7	29	7	32	4	3	2	15
(10,4)	0	24	0	20	0	21	0	2	0	8
(10,5)	0	12	0	9	0	15	0	0	0	6
(10,6)	0	8	2	5	2	8	1	0	3	1

parameter set (10,5) or (10,4) seems to be a good operation point. It results in no false negatives and very small false positives.

- The fact that there are false positives even at (10,6) seems to indicate that some regular packets (which occur less than 1% of the time) repeat at least 6 times in a small window. We looked at the individual traces and confirmed that it is indeed the case. However a point to note in this context is that we have not used timing information from the traces. In reality if a regular packet repeats multiple times but the interval between repetitions is large say above 100 ms, then it will never result in a false positive. In our evaluation however, it will result in a false positive.

- A value of RDC below 2 can generate quite a few false positives. However, false positives can easily be tackled with a small change at the sender. Whenever the sender sees a regular packet that corresponds to an alphabet, it can either pad or truncate it to make it a non-alphabet. The false positive number can also be reduced if we choose a smaller threshold percentage when extracting the alphabet. This will however reduce the alphabet size.

- The g mode alphabet seems to be slightly more robust to false positives and negatives, at least for the PSU-CSE trace. Within a mode, the higher data rates seem to be more robust. This is to be expected since higher data rates and g mode operation gives a relatively uncluttered run length space from which to extract the alphabet.

In summary, the above results look encouraging. We are able to extract a sufficiently large sized alphabet set for communication. Our validation shows that communication can work effectively in practice with very few false positives and false negatives.

10.5 Discussion

10.5.1 Applicability in Other Contexts

Our evaluation of the ideas presented here has been using unidirectional communication from a 802.11b/g device to a 802.15.4 device. However, this choice of standard was driven more by the availability of such programmable devices and not by any limitations of the concept. All that this technique needs is ability to sense energy at good accuracy in the frequency band in which we are interested. Empowered with such capability, the ideas presented here can be used with other standards, for bi-directional communication and possibly also in many additional scenarios than listed here. For example, one could use this framework to achieve energy savings in 802.11a networks, construct interference maps of Wimax networks, and explore coexistence between Bluetooth and 802.11. The concepts presented here would work on out-of-the-box products if they expose such sensing functionality, or if one designs new hardware that can sense energy signals with sufficient accuracy.

10.5.2 Building Vocabulary

We have not looked at building vocabulary on top of the base alphabet. Such an approach may be necessary when the message space exceeds the base alphabet. For example if one were considering energy savings in 802.11g infrastructure mode, with the use of only the base alphabet set, it would be quite constraining. The alphabet size of 10 means that Esense-based wake-up via the secondary radio is possibly for only 10 clients at a time. Further, the false positive (negative) rate can be reduced

significantly by considering an extended vocabulary: a specific pattern of alphabets will be more difficult to generate (miss) than a single alphabet. However, delineating word boundaries in presence of insertions/deletions of alphabets on the channel is not a straightforward task. The insertions on the channel correspond to regular packets being misinterpreted as Esense packets. The deletions on the channel correspond to the case where the hardware is unable to detect the alphabet due to merging of packets. Such insertion/deletions on channels have been looked in the context of error correction codes. We believe this is an interesting avenue for future exploration.

10.5.3 Packet size distribution

One may argue that the idea of energy sensing falls apart if we do not have a packet size distribution that shows a modal behavior. Our observation, based on Wifi traces has shown that modes *do* exist in this domain. However, there may be many other application domains such as traffic dominated by video streams where the modal behavior may be absent. There are two possible solutions to this problem. First, we may use a much more accurate hardware for energy sensing which may be able to measure energy burst length to the extent of one byte. This will make any packet size that has occurred less frequently to be considered as an Esense packet. Second approach is to designate some packet lengths as Esense reserved and all participating devices may either fragment packets or pad them so that no regular packets are sent corresponding to the Esense packet lengths. Interestingly enough, this modification can be made even if the nature of the traces is modal. We can ensure a very low rate of false positives if regular packets never look like Esense packets. However, such an approach requires modification of the drivers for all devices that wish to be a part of this network.

10.5.4 Possible consideration for Esense in future standards

As newer wireless standards emerge, with the unlicensed frequency spectrum already scarce, more and more devices are likely to interfere with each other and the need for cross-standard communication or co-existence becomes acute. Future wireless standards may benefit from building a native support for Esense during their conception phase. A wireless standard, when it is designed, cannot predict what other future standard is going to operate in the same spectrum space. A standard could embed

support for Esense by reserving certain packet lengths or energy run lengths. Of course, all standards must agree on what run-lengths to reserve so that these can be used for intelligent Esense-based coordination with any other wireless device in the future.

Chapter 11

Conclusion - Esense

We considered a new method of communication between devices that cannot interpret the individual bits of the packet. However, if these devices share the same frequency spectrum, we facilitate their communication by sensing and interpreting energy patterns on the air. We argue that the most effective method for communication is to assign meaning to the length of an energy burst compared to the signal strength or the inter-packet gap.

We describe how this framework opens up new approaches to solving problems in at least three distinct research domains. Significant work has been done in these three domains, but our solution is novel in that it can simultaneously offer solutions to all of them. We validate our mode of communication on a hardware platform using realistic traces from WiFi deployments. Our evaluation shows that our approach can lead to sufficiently large alphabet set that can facilitate effective communication. We believe that this framework has implications in other areas beyond the three example scenarios we have presented and has a potential for consideration in future wireless standards.

Bibliography

[1] A Community Resource for Archiving Wireless Data At Dartmouth. http://crawdad.cs.dartmouth.edu/.

[2] Bluetooth SIG. https://www.bluetooth.org/.

[3] HAL source code released. http://madwifi-project.org/wiki/news/20080929/hal-source-code-released.

[4] IEEE 802.11. http://standards.ieee.org/getieee802/802.11.html.

[5] IEEE 802.15.4 WPAN Task Group 4 (TG4). http://www.ieee802.org/15/pub/TG4.html.

[6] Madwifi website. http://madwifi.org.

[7] Traces of the Stanford CS department's wireless network. http://crawdad.cs.dartmouth.edu/stanford/gates, 2003.

[8] Traces of network activity at OSDI 2006. http://crawdad.cs.dartmouth.edu/microsoft/osdi2006, 2006.

[9] Dataset of wireless LAN traffic around Portland, Oregon using a commercial sniffer VWave. http://crawdad.cs.dartmouth.edu/pdx/vwave, 2007.

[10] Yuvraj Agarwal, Ranveer Chandra, Alec Wolman, Paramvir Bahl, Kevin Chin, and Rajesh Gupta. Wireless wakeups revisited: energy management for voip over wi-fi smartphones. In *MobiSys '07: Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 179–191, New York, NY, USA, 2007. ACM.

[11] Kameswari Chebrolu and Bhaskaran Raman. Fractel: A fresh perspective on (rural) mesh networks. *ACM SIGCOMM Workshop on Networked Systems for*

Developing Regions (NSDR'07), A Workshop in SIGCOMM 2007, Aug 2007, Kyoto, Japan.

[12] Saumitra M. Das, Dimitrios Koutsonikolas, Y. Charlie Hu, and Dimitrios Peroulis. Characterizing multi-way interference in wireless mesh networks. In *WiNTECH '06: Proceedings of the 1st international workshop on Wireless network testbeds, experimental evaluation & characterization*, pages 57–64, New York, NY, USA, 2006. ACM.

[13] J. Elson, L. Girod, and D. Estrin. Fine-Grained Time Synchronization using Reference Broadcasts. *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002), Boston, MA, December 2002.*

[14] Saurabh Ganeriwal, Ram Kumar, and Mani B. Srivastava. Timing-sync Protocol for Sensor Networks. *SenSys '03.*

[15] Sangjin Han, Sungjin Lee, Sanghoon Lee, and Yeonsoo. Kim. Channel allocation algorithms for coexistence of lr-wpan with wlan. In *IEICE Transactions on communications*, May 2008.

[16] Ronny Krashinsky and Hari Balakrishnan. Minimizing energy for wireless web access with bounded slowdown. In *MobiCom*, 2002.

[17] Mikls Marti, Branislav Kusy, Gyula Simon, and kos Ldeczi. The Flooding Time Synchronization Protocol. *SenSys '04.*

[18] Nilesh Mishra, Kameswari Chebrolu, Bhaskaran Raman, and Abhinav Pathak. Wake-on-wlan. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 761–769, New York, NY, USA, 2006. ACM.

[19] Dragoş Niculescu. Interference map for 802.11 networks. In *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 339–350, New York, NY, USA, 2007. ACM.

[20] Jitendra Padhye, Sharad Agarwal, Venkata N. Padmanabhan, Lili Qiu, Ananth Rao, and Brian Zill. Estimation of link interference in static multi-hop wireless networks, 2005.

[21] S. Pollin, M. Ergen, A. Dejonghe, L. V. Perre, F. Catthoor, I. Moerman, and A. Bahai. Distributed cognitive coexistence of 802.15.4 with 802.11. In *CROWNCOM*, 2006.

[22] Charles Reis, Ratul Mahajan, Maya Rodrig, David Wetherall, and John Zahorjan. Measurement-based models of delivery and interference in static wireless networks. *SIGCOMM Comput. Commun. Rev.*, 36(4):51–62, 2006.

[23] Eugene Shih, Paramvir Bahl, and Michael J. Sinclair. Wake on wireless: an event driven energy saving strategy for battery operated devices. In *MobiCom '02: Proceedings of the 8th annual international conference on Mobile computing and networking*, pages 160–171, New York, NY, USA, 2002. ACM.

[24] Fikret Sivrikaya and Bulent Yener. Time Synchronization in Sensor Networks: A Survey. 2003.

[25] C. Won, J. H. Youn, H. A. Sharif, and J. Deogun. Adaptive radio channel allocation for supporting coexistence of 802.15.4 and 802.11b. In *IEEE Vehicular Technology Conference*, 2005.

[26] Dong Zhou and Ten H. Lai. An Accurate and Scalable Clock Synchronization Protocol for IEEE 802.11-based Multihop Ad Hoc Networks. *IEEE Transactions on Parallel and Distributer Systems*, 2007.