SAS: Self-Adaptive Spectrum Management for Wireless Sensor Networks

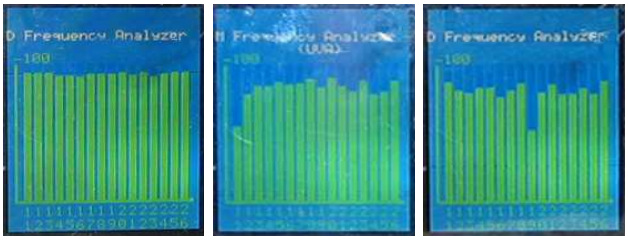
Gang Zhou†, Lei Lu†, Sudha Krishnamurthy‡, Matthew Keally†, Zhen Ren†
†Department of Computer Science, College of William and Mary, Williamsburg, VA
‡United Technologies Research Center, East Hartford, CT
†{*gzhou, llu, makeal, renzh*}*@cs.wm.edu*, ‡*krishnadots@gmail.com*

***Abstract*— Smart wireless sensor devices are becoming increasingly ubiquitous and are expected to be embedded in everyday objects in the near future. When these devices are deployed in overlapping or adjacent geographic areas, the unlicensed 2.4GHz ISM band will be crowded. To deal with the crowded spectrum, we propose SAS, a Self-Adaptive Spectrum management middleware for wireless sensor networks. SAS enables single-frequency MAC protocols with multi-frequency capability, so that an existing MAC protocol, like B-MAC, can automatically adapt to the least congested physical channel at runtime. We implemented SAS in TinyOS 2.1 with nesC and evaluated its performance with TelosB motes. Our performance results demonstrate that SAS improves the performance of existing single-frequency MAC protocols, like B-MAC. The use of SAS results in higher packet reception ratio and system throughput, and a lower packet delay and energy consumption.**

I. Introduction

With the maturity and wide application of low-power wireless protocols, such as IEEE 802.11b [1], 802.15.1 [2] and 802.15.4 [3], we envision a future in which wireless devices, such as wireless keyboards, power-point presenters, cell phone headsets, and health monitoring sensors [4] will be ubiquitous. Unfortunately, the pervasiveness of these devices leads to increased interference and congestion within as well as between networks, since they adopt overlapping physical frequencies. For example, 802.11b uses Direct Sequence Spread Spectrum (DSSS) and divides the 2.4 GHz ISM band into 14 channels (5 MHz distance apart). IEEE 802.15.1 divides the 2.4 GHz ISM band into 79 1-MHz channels and uses Frequency Hopping Spread Spectrum (FHSS). IEEE 802.15.4 divides the 2.4 GHz ISM band into 16 5-MHz channels and uses DSSS. When these wireless devices are used in the same office or home environment, there is a high likelihood that the 2.4 GHz ISM band will be overloaded and congested.

In addition to the interference from co-existing networks and wireless devices, communication in the 2.4 GHz ISM band may also be affected by existing electrical appliances. For example, Figure 1 displays the carrier-sensed power levels of the 16 channels, from 11 to 26, in IEEE 802.15.4, measured by the SeeMote [5] in a home environment. The x-axis is the channel index number and the y-axis denotes the sensed power level in dBm. Comparing Figure 1 (a) and (b), it is quite clear that an existing WSN congests channel 11 (default channel in Tinyos 2.1) and also generates interference on the other channels. By comparing Figure 1 (a) with (c), we observe that the presence of a microwave also greatly increases the sensed power level in channel 19, from -94 dBm when the microwave is off to -52 dBm when it is running. When the microwave is on, it also leads to higher noise levels for the other channels. By comparing Figure 1 (b) and (c), we surprisingly find that the interference from the microwave is even stronger than that arising from a co-existing WSN. One method to



(a) Quiet Environment (b) Co-existing WSN (c) Microwave Running

Fig. 1. 2.4 GHz Spectrum Usage in a Home Environment

address the problem arising from this crowded spectrum is to introduce more unlicensed frequency band. However, this usually needs approval from government agencies (such as the Federal Communications Commission (FCC) in the United States). Another method to address the crowded spectrum is to propose software solutions that can enable multiple nodes to conduct parallel communication at different frequencies (i.e. channels). In the low power wireless communication literature, several approaches have been developed to utilize multiple frequencies for parallel communication, including cognitive radios [6] and multi-frequency MACs [7], [8]. Even though these approaches have proven to be effective in improving system performance for networks in which each individual node has only a half-duplex radio transceiver, they are usually designed for a specific PHY layer or encapsulated within a specific MAC protocol. These solutions are not generic and hence, can not be used to easily enhance existing single-frequency MAC protocols with multi-frequency capability, in order to support parallel communication.

Therefore, the lack of a generic solution to leverage existing MAC protocols motivates us to design SAS: a Self-Adaptive Spectrum Management middleware for wireless sensor net-

works, which can be easily integrated with an existing single-frequency MAC protocol to take advantage of multiple frequencies for parallel communication. The main contributions of this work are: 1) We propose SAS as a middleware that can easily enhance existing MAC protocols with multi-frequency capability for system-wide parallel communication. 2) We implement SAS in TinyOS 2.1 with nesC and evaluate its performance using TelosB motes. The performance results demonstrate that SAS is capable of enhancing the performance of single-frequency MAC protocols (e.g. B-MAC [9]).

The rest of the paper is organized as follows. In Section II, we discuss related work. In Section III, we describe the design of SAS and its implementation using TinyOS and TelosB. We discuss performance evaluation results in Section IV, and finally, present our conclusions in Section V.

II. State of the Art

A number of multi-frequency MAC protocols have been proposed for wireless networks in which each node only has a single half-duplex radio transceiver. According to [10], these MAC protocols can be classified into two categories: those that adopt RTS/CTS control packets and those that do not. Some multi-frequency MAC protocols rely on RTS/CTS control ([11] [12] [13] [14]), because either their designs are based on IEEE 802.11b [1], or they adopt RTS/CTS control for frequency negotiation ([7] [15] [16] [17]). The work in [10] demonstrates that even though RTS/CTS control exhibits good performance in general wireless ad-hoc networks, it is not appropriate for WSNs, where the network bandwidth is very limited and the MAC layer packet size is very small (typically 30∼50 bytes), compared to a packet size of at least 512 bytes in general wireless ad-hoc networks.

Some other multi-frequency MAC protocols are specially designed for wireless sensor networks applications and do not use RTS/CTS controls [10] [18]. These MAC protocols demonstrate largely improved system performance due to the use of multiple frequencies for parallel communication, but they are not generally designed for enhancing existing single-channel MAC protocols (e.g. B-MAC [9]). The SAS protocol we propose in this paper targets the same wireless sensor network applications, but without incurring the overhead of RTS/CTS packets. Instead of proposing another special-purpose multi-frequency MAC protocol, we put forth a self-adaptive spectrum management middleware, which can be easily integrated with existing single-channel MACs, in order to dynamically share the spectrum for parallel communication.

III. SAS Design

There are three important challenges to be addressed in designing SAS. First, SAS is a middleware positioned between the MAC and PHY layers, as shown in Figure 2. So clear interface definitions are needed between SAS and MAC and between SAS and PHY. Second, while neighboring nodes in SAS are encouraged to use different frequencies for parallel unicast communication, broadcast needs to be supported from the root. SAS addresses this challenge through virtual transceiver design. Third, SAS needs to manage available physical frequencies during runtime and make the right decisions for switching frequencies, according to current spectrum usage. In the following subsections, we elaborate how SAS addresses these challenges.

A. SAS Interface

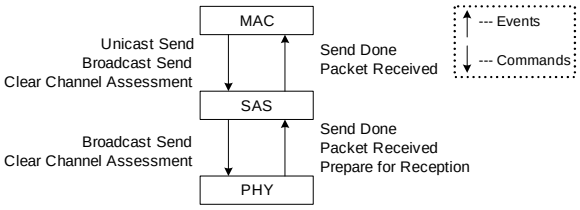


Fig. 2. SAS Middleware Architecture

SAS is a generic multi-frequency middleware that is accessible through the interfaces defined in Figure 2. The multi-frequency service in SAS can be invoked through three simple commands: `UnicastSend`, `BroadcastSend` and `ClearChannelAssessment(CCA)`. `UnicastSend` is used to send unicast packets, `BroadcastSend` is used to transmit broadcast packets, and `CCA` is used for carrier sensing. In response, SAS uses the `SendDone` event to inform the MAC layer whether the packet has been sent or dropped (transmission failure). The `PacketReceived` event is signaled to notify the MAC layer that a packet has been correctly received.

SAS builds its service upon functions provided by the PHY layer. As shown in Figure 2, SAS calls PHY's `BroadcastSend` command to send both unicast and broadcast packets, using different destination addresses for differentiation. The `CCA` command of PHY is used for carrier sensing. The `SendDone` and `PacketReceived` events generated by PHY are used to notify SAS about the result of packet transmission and reception, respectively. The PHY layer also generates the `PrepareForReception` event, which informs SAS that a valid packet has been detected in the air. Depending on the radio hardware at the PHY layer, this event can have different implementations.

The `CCA` command deserves further explanation. In a single-frequency MAC design, `CCA` means "Is *the* channel busy?", since the MAC assumes that there is only one physical channel. However, when multiple frequencies are supported, a `CCA` request implies "Is the channel that the *receiver* is using busy?" Since the MAC layer does not know what channel the receiver currently uses, it is not possible for the MAC layer to provide a frequency parameter within a `CCA` request. Instead, the `CCA` command is slightly modified to take the receiver ID as input, since the MAC layer knows the destination node that the packet is addressed to.

When SAS gets the receiver ID from the CCA request, it looks up its neighbor table to find out the channel number that the destination node currently uses. SAS conducts carrier sense on this specific channel and returns the results to MAC.

Thus, through this simple CCA interface reconfiguration, SAS is able to enhance single-frequency MAC protocols to take advantage of multiple frequencies in a transparent manner, without the need for redesigning MAC protocols to achieve multi-frequency capability.

B. SAS Virtual Transceiver

As mentioned in Section II, a typical sensor device, like MICAZ and TelosB [19], is usually equipped with a single half-duplex radio transceiver, to reduce product cost, form factor, and energy consumption. This limited radio transceiver needs to be enhanced by software, to support multi-frequency capability. The SAS virtual transceiver proposed in this section is designed to meet this requirement.

In the multi-frequency context, we assign neighboring nodes different frequencies for unicast packet reception, and the same default channel for broadcast reception. This design philosophy maximizes parallel unicast communication and at the same time seamlessly supports broadcast from the root. With the help of SAS, the limited radio transceiver is transformed to a virtual transceiver that is able to snoop on three frequencies: f_{BC}, f_{UNI}, and f_{CAN}, *simultaneously*. f_{BC} denotes the default frequency chosen for broadcast, and f_{UNI} indicates the frequency for unicast reception. This unicast frequency is selected from the currently least loaded frequencies and it may vary from time to time and from node to node, according to a node's current spectrum usage within its local geographic area. When a node is first assigned a unicast frequency or when it switches frequency, it notifies its neighbors with the updated f_{UNI} value through a broadcast (reliability issues [20] in broadcast can be incorporated in future). In addition to the broadcast and unicast frequencies, a candidate frequency, f_{CAN}, is guessed at runtime, in order to replace the current unicast frequency when it is congested. Details about how to pick the f_{CAN} frequency is explained later in Section III-C.

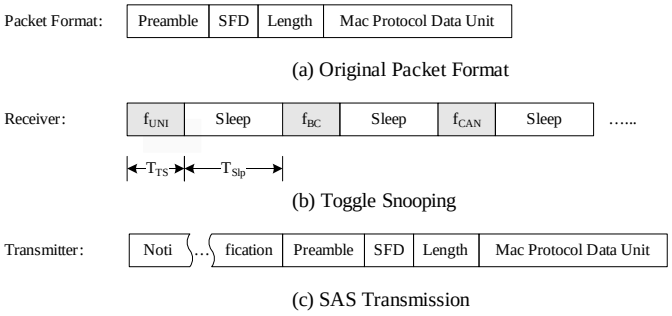


Fig. 3. SAS Virtual Transceiver

Figure 3 presents the detailed design of a SAS virtual transceiver. As shown in Figure 3(a), a packet normally starts with the preamble bytes, then follows with the Start-of-Frame Delimiter (SFD) field, the packet length field, and then the data bytes. Also, an optional CRC field is usually added at the end of the data bytes.

In a single-frequency context, when a node is deployed and turned on, it enters into the idle listening state. However, in SAS's multi-frequency context, the node enters the toggle snooping state, as shown in Figure 3(b). The transceiver listens to the unicast frequency f_{UNI}, the broadcast frequency f_{BC}, and the candidate frequency f_{CAN}, alternatingly (with optional sleep periods for possible duty cycling extensions in the future). After listening to each frequency, the transceiver goes into sleep state to save power. T_{TS} denotes the time that a node stays on any of the three frequencies and T_{Slp} denotes the time that a node stays in the sleep state. Whenever the transceiver stays in frequency f_{UNI} or f_{BC}, and also detects a radio signal in the air, it stops frequency toggling to prepare for data reception. As shown in Figure 3(c), each data packet in SAS is preceded by notification bytes, which inform a receiver in the toggling state to stop frequency toggling and instead, switch to the data reception state. The toggle snooping resumes when the packet reception ends.

The toggle snooping performed by the receiver has the same implementation in different radio hardware. However, the implementation of the notification bytes in the SAS transmission depends on the hardware platform. Since currently many wireless sensor devices, like MicaZ, Telos and IMote2, use the CC2420 radio, we present the details of the SAS virtual transceiver that we implemented over the CC2420 radio in TelosB motes. For the SAS virtual radio transceiver to operate correctly, the time to send the notification (T_{send}) must be greater than $2T_{TS} + 3T_{Slp}$ (see Figure 4). When this condition is violated, the transmitter's notification time is not long enough to ensure that the receiver will be able to pick up a valid packet signal, and that the sending packet will be received without interference.

CC2420 is a packet-level radio hardware. It does not provide software designers the ability to either configure a long preamble or read individual preamble bytes when a packet is in air. Instead, the radio hardware is in charge of adding and removing preamble bytes automatically. So, we can not use a long preamble to serve as the notification, as is possible in a byte-level radio hardware (like CC1000 in Mica2 motes). Hence, in our implementation, we use the data packet itself (multiple duplicates) as the notification, as shown in Figure 4.

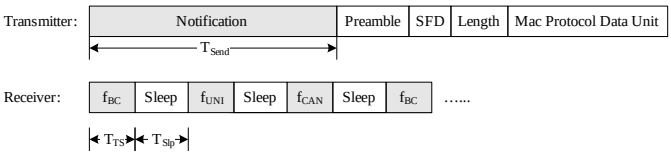


Fig. 4. SAS Virtual Transceiver Implementation in CC2420 Radio

C. Runtime Frequency Switching

We have explained the communication protocol when the SAS virtual transceiver operates on both broadcast and unicast frequencies. We now explain how SAS switches its unicast frequency f_{UNI} dynamically, according to the runtime spectrum usage. By switching unicast frequencies, it is possible

to take advantage of all available physical frequencies, in order to balance the traffic load in both spatial and temporal dimensions.

To achieve that, SAS needs to answer two questions at runtime: 1)Is the current frequency (i.e. channel) congested? and 2)What is the best frequency to switch to when the current one is congested? To answer these two questions, SAS carefully manages its available resources, which are the physical frequencies. The state of each frequency in each node varies according to the state machine depicted in Figure 5. A

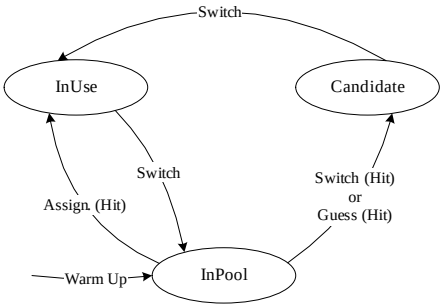


Fig. 5. The SAS State Machine

node uses one of the following three states to denote the usage status of each channel: `InUse, InPool` and `Candidate`. The frequency that is chosen for current unicast reception is in the `InUse` state. The currently guessed candidate frequency is assigned the `Candidate` state. All the other frequencies are placed in the `InPool` state. The state transition is triggered by one of the following four actions: warm up, frequency assignment, frequency guess, and frequency switch.

1) WarmUp and Frequency Assignment: When a sensor network is newly deployed and started, it is important to detect what frequencies are currently used and which of them suffer interference from existing electrical and electronic devices. An initial warmup phase serves this purpose, during which the load on each frequency is measured and recorded. During warmup, each node scans all physical frequencies in multiple rounds. In each round, the node stays on each frequency for a short time that is long enough for sampling the channel status. The status is a binary result: 0 for idle and 1 for busy. This value is assigned to $\bar{\phi}$, which is used to generate a moving average value ϕ (see Table III-C.1) with a decay of α:

$$\phi_{n+1} = \alpha \times \phi_n + (1 - \alpha) \times \bar{\phi} \quad (1)$$

After collecting the channel load information, each node knows the current spectrum usage of existing networks and devices. Each node avoids choosing the congested frequencies by checking corresponding ϕ values. Also, to avoid internal congestion within the newly deployed network, frequency assignment is needed after the short warm-up. During frequency assignment, each node backs off for a random time period before it chooses its frequency. During the backoff period, each node records the frequencies it overhears. When its backoff timer fires, a node chooses one of the least loaded frequencies for unicast reception.

TABLE I

TABLE ENTRY MAINTAINED FOR EACH PHYSICAL FREQUENCY

Frequency State	At any time, a frequency should be in one of the three states: InPool, InUse and Candidate.
ω	It indicates how many times a frequency/channel has been chosen for unicast packet reception.
ϕ	It denotes the channel load, calculated as a moving average with decay α. The moving average computation gets re-initialized each time a frequency enters the Candidate state again.
ψ	It represents the packet loss ratio, calculated as a moving average with decay η. The moving average gets re-initialized each time the frequency is used again.
ξ	This is the composite value that reflects how competitive a frequency is, and is used for frequency comparison. It is a balanced reflection of a frequency's most recently measured load ϕ, the frequency's credit history ω and its current usage by neighboring nodes φ.

2) Candidate Frequency Guess: In SAS, each node needs to be prepared for frequency switches when congested. So it is helpful for each node to know the traffic load of other physical frequencies, even though they are currently not in use. Considering scalability, it is energy and bandwidth intensive for each node to keep track of the loads of all frequencies all the time. However, it is not necessary to continuously carrier sense all the frequencies, since each node only needs one candidate frequency to switch to when congested.

An energy-efficient and scalable design is to let each node guess one frequency that may have the highest possibility of being the most lightly loaded frequency in the near future, and place that frequency in the `Candidate` state. A `Candidate` frequency may be discarded, when it is found to be likely overloaded and hence no longer promising. In that case, the frequency is placed in the `InPool` state again.

To assist in deciding when and how to guess a candidate frequency, we introduce the ξ parameter. As shown in Table III-C.1, ξ is a composite metric, which characterizes the frequency's recently measured load (ϕ), the frequency's credit history (ω), and the number of neighboring nodes (φ) that are currently using it. The frequency load ϕ is computed as in Equation (1). To compute a frequency's credit history, SAS uses the parameter ω to record how many times a frequency has been used, which reflects how many times this frequency has been discarded and replaced by a better one. The greater the ω value, the higher the probability that the frequency will be congested and discarded again in the near future, and hence, the less promising the frequency is. Also, each node should avoid competing for the same frequency with its neighbors. When the number of available physical frequencies is no less than the node density, SAS should give neighboring nodes different frequencies to avoid congestion. When the number of available physical frequencies is very limited compared to the node density, each node in SAS needs to give higher priority to frequencies that are shared among fewer neighbors. Hence, the

φ parameter is introduced to indicate the number of neighbors that share a frequency. Taking these three into consideration, parameter ξ can be calculated as in Equation (2), where β and γ are coefficients:

$$\xi = \beta \times \omega + \gamma \times \phi + (1 - \beta - \gamma) \times \varphi \quad (2)$$

The first frequency guess happens at the end of the warm-up, when the initial `Candidate` frequency needs to be decided. After that, frequency guess is triggered when the Candidate frequency's ξ value exceeds ξ_{Thr}. If the frequency guessing causes the Candidate frequency to enter into the `InPool` state again, the `Candidate` frequency is replaced with the frequency in the `InPool` state (see Figure 5) that has the smallest ξ value.

3) Frequency Switch: As illustrated in Figure 5, when a frequency switch happens, the current unicast frequency, f_{UNI}, switches its state from `InUse` to `InPool`. In addition, the `Candidate` frequency replaces the `InUse` frequency, and a new `Candidate` frequency is picked out from the pool. Since f_{UNI} is only used for unicast reception, the frequency switch is triggered when the monitored packet loss ratio of the application is above a threshold.

Typically, there are two ways for a receiver to obtain the packet loss ratio. One solution is to use a packet sequence number. By tracking the sequence numbers and calculating the gaps, a receiver is able to compute the packet loss ratio. The second scheme is to use transmitter-receiver handshakes, such as RTS-CTS-DATA-ACK in IEEE 802.11b [1]. However, both of them introduce extra overhead and neither of them takes advantage of the SAS design. In the first scheme, SAS is required to get access to and understand the upper layer's sequence numbers, which violates SAS's design goal of being a generic multi-frequency middleware that does not depend on support from the MAC protocol layered above it. The second scheme introduces significant communication overhead.

Hence, we propose a novel scheme to implicitly compute the packet loss ratio in SAS, by taking advantage of the toggle snooping design described in Section III-B. In SAS, toggle snooping is the default behavior of each receiver, which only gets interrupted when a valid data signal is detected in the air. If a packet is correctly received after an interruption, the receiver knows that the communication has succeeded, otherwise a communication failure is detected. This success/failure binary status is obtained for free, and is recorded in the packet loss variable, $\bar{\psi}$, and a moving average value ψ is computed with decay η as follows:

$$\psi_{n+1} = \eta \times \psi_n + (1 - \eta) \times \bar{\psi} \quad (3)$$

By comparing the ψ value with a threshold ψ_{Thr}, the frequency switch is automatically triggered, when $\psi \geq \psi_{Thr}$.

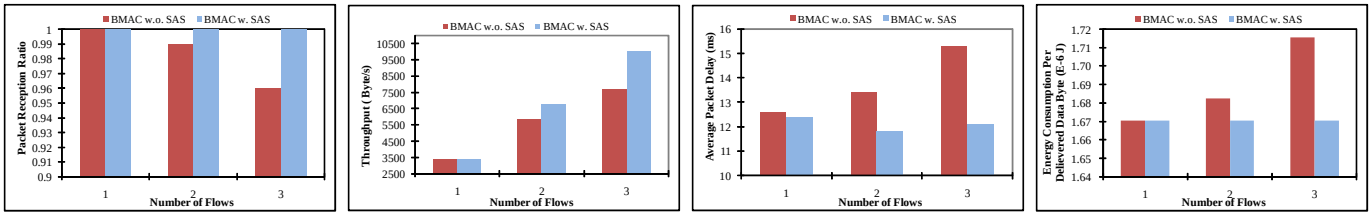
IV. Performance Evaluation

We integrated B-MAC with SAS in TinyOS 2.1 on TelosB motes and experimentally evaluated its performance and compared it with that of regular B-MAC protocol. The experiments were conducted in a campus building, where the primary source of interference was the wireless network in the building. As we mentioned in Section III-B, the data packets themselves are used for notification. So the duration of the continuous CCA check interval should be longer than the interval between the transmission of successive notification data packets. This check interval was set to be the maximum delivery delay in the network. During this interval, the CCA check was performed 400 times, which is also the default value in the B-MAC code. In order to deal with heavy traffic, when a node receives a data packet on a channel, the radio continues to stay on that channel for a certain short period. We set this period to 100 milliseconds, which is the default value used in the B-MAC code. We set the notification time (T_{send}) to 20 milliseconds for the transmitter and let the transmitter send data packets as fast as possible. For the parameters presented in Section III, we used the following values: $\alpha = 0.96$, $\beta = 0.45$, $\gamma = 0.35$, and $\eta = 0.96$. In each of the experiments, we varied the number of data flows from 1 to 3 and each sender transmitted $10,000$ data packets to a different receiver, to emulate an environmental monitoring application. When there was more than one flow, the receivers adaptively chose a different frequency during the warmup stage, when B-MAC with SAS was used.

Figure 6(a) shows the packet reception ratio (PRR) achieved by B-MAC with and without SAS, as the number of contending flows in the network increases. The PRR is calculated as the ratio of the total number of data packets successfully delivered to the total number of data packets transmitted. The PRR when SAS is used remains almost 100%, even when the number of flows increases. On the other hand, B-MAC without SAS delivers 100% of its given load when there is only one sender-receiver pair. However, the reception ratio drops to 99% when there are 2 simultaneous data flows, and further drops to 96% when there are 3 data flows. This degradation may either be due to collision or because the contention in the network may result in the transmitter queues being full, forcing some of the transmitters to drop some of the packets from their queues.

Figure 6(b) shows the measured system throughput. The throughput measures the performance gain and is calculated as the total amount of useful data successfully delivered in the network per unit time. The packet size we used is 54 bytes (data payload plus MAC header). With SAS, the total network throughput achieved when there are 3 contending flows, is almost 3 times that achieved when there is a single flow. The reason is that the multi-frequency management in SAS enables neighboring nodes to adaptively choose different physical frequencies. As a result, when the number of flows in the network increases, the MAC layer above SAS does not suffer as much radio interference as the regular B-MAC and more nodes are able to simultaneously transmit without collisions. In the case of B-MAC without SAS, the total throughput also increases when the number of flows increases. However, due to congested channels, each node has to backoff more number of times, in order to transmit one packet.

Figure 6(c) shows the average packet delay as the number of contending flows in the networks increases. The packet delay measures the one-way latency experienced by a data



(a) Packet Reception Ratio (b) System Throughput (c) Average Packet Delay (d) Energy Consumption

Fig. 6. Delay and Overhead of SAS

packet from the time it is transmitted by the MAC layer on the transmitter side to the time at which it is received by the MAC layer on the receiver side. B-MAC without SAS experiences a higher packet delay compared to B-MAC with SAS. The average delay of B-MAC with SAS remains close to 12 milliseconds as the number of flows increases from 1 to 3. In contrast, the packet delay for B-MAC without SAS increases from 12.6 milliseconds, when there is a single sender-receiver pair, to 15.3 milliseconds, when the number of flows increases to 3. The increased delay in the latter case is primarily because the the number of backoff in B-MAC increases as the number of flows increases. The delay when using SAS does not increase significantly, since the load is distributed across three frequencies.

The energy consumption per byte of successfully delivered data is shown in Figure 6(d). This is the sum of the energy consumed for sending a packet and for performing the CCA check. The power level used for transmissions is 0 dBm. With SAS, the energy consumed remains almost constant, as the number of flows increases from 1 to 3, whereas in the case of B-MAC without SAS, the energy consumed increases with the number of flows. The reason is that when the traffic is heavy, the packet loss is higher in the case of B-MAC without SAS (see Figure 6(a)) and hence, more energy is spent in sending data that was finally corrupted.

V. Conclusions

SAS is an adaptive spectrum management middleware for wireless sensor networks, which provides a way to deal with the problem of crowded spectrum. SAS is designed to enhance single-frequency MAC protocols with multi-frequency capability, so that an existing MAC protocol like B-MAC, can dynamically adapt to the least congested physical frequency at runtime. This dynamic choice of frequency is useful in order to address the performance degradation resulting from interference and time-varying traffic patterns. Our performance results, based on an implementation on a TelosB testbed, demonstrate that SAS improves the performance of single-frequency B-MAC, and results in higher packet reception ratio and system throughput, and a lower packet delay. As part of future work, we plan to perform a more extensive evaluation of SAS, especially in the presence of dynamic traffic patterns and with different radio platforms.

References

[1] "IEEE 802.11, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification," ANSI/IEEE Std. 802.11, 1999.

[2] "IEEE 802.15.1, Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs)," IEEE Std. 802.15.1, 2002.

[3] "IEEE 802.15.4, Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)," IEEE Std. 802.15.4, 2003.

[4] G. Zhou, J. A. Stankovic, and S. F. Son, "Crowded Spectrum in Wireless Sensor Networks," in *IEEE EmNets*, 2006.

[5] L. Selavo, G. Zhou, and J. A. Stankovic, "SeeMote: In-Site Visualization and Logging Device for Wireless Sensor Networks," in *IEEE BASENETS*, 2006.

[6] F. Wang, M. Krunz, and S. Cui, "Spectrum Sharing in Cognitive Radio Networks," Tech. Rep., University of Arizona, 2007.

[7] J. So and N. Vaidya, "Multi-Channel MAC for Ad-Hoc Networks: Handling Multi-Channel Hidden Terminal Using A Single Transceiver," in *ACM MobiHoc*, 2004.

[8] J. Mo, H. So, and J. Walrand, "Comparison of Multi-Channel MAC Protocols," in *Symposium on Modeling, Analysis, and Simulation of Wireless and Mobile Systems*, 2005.

[9] J. Polastre, J. Hill, and D. Culler, "Versatile Low Power Medium Access for Wireless Sensor Networks," in *ACM SenSys*, 2004.

[10] G. Zhou, C. Huang, T. Yan, T. He, J. A. Stankovic, and T. F. Abdelzaher, "MMSN: Multi-Frequency Media Access Control for Wireless Sensor Networks," in *IEEE INFOCOM*, 2006.

[11] P. Bahl, R. Chancre, and J. Dungeon, "SSCH: Slotted Seeded Channel Hopping for Capacity Improvement in IEEE 802.11 Ad-Hoc Wireless Networks," in *ACM MobiCom*, 2004.

[12] A. Raniwala and T. Chiueh, "Architecture and Algorithm for an IEEE 802.11-Based Multi-Channel Wireless Mesh Network," in *IEEE INFOCOM*, 2005.

[13] A. Adya, P. Bahl, J. Padhye, A. Wolman, and L. Zhou, "A Multi-radio unification protocol for IEEE 802.11 wireless networks," in *BroadNets*, 2004.

[14] J. Li, Z. J. Haas, M. Sheng, , and Y. Chen, "Performance Evaluation of Modified IEEE 802.11 MAC for Multi-Channel Multi-Hop Ad Hoc Network," in *IEEE AINA 2003*.

[15] N. Jain and S. R. Das, "A Multichannel CSMA MAC Protocol with Receiver-Based Channel Selection for Multihop Wireless Networks," in *IEEE IC3N*, 2001.

[16] A. Tzamaloukas and J.J. Garcia-Luna-Aceves, "A Receiver-Initiated Collision-Avoidance Protocol for Multi-Channel Networks," in *IEEE INFOCOM*, 2001.

[17] Z. Tang and J.J. Garcia-Luna-Aceves, "Hop-Reservation Multiple Access (HRMA) for Ad-Hoc Networks," in *IEEE INFOCOM*, 1999.

[18] Y. Wu, J. Stankovic, T. He, J. Lu, and S. Lin, "Realistic and Efficient Multi-Channel Communications in Wireless Sensor Networks," in *IEEE INFOCOM*, 2008.

[19] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling Ultra-Low Power Wireless Research," in *ACM/IEEE IPSN/SPOTS*, 2005.

[20] J. Chang and N. F. Maxemchuk, "Reliable Broadcast Protocols," in *ACM Transactions on Computer Systems*, 1984.