Defending Wireless Sensor Networks from Radio Interference through Channel Adaptation

WENYUAN XU
University of South Carolina
and
WADE TRAPPE and YANYONG ZHANG
WINLAB, Rutgers University

Radio interference, whether intentional or otherwise, represents a serious threat to assuring the availability of sensor network services. As such, techniques that enhance the reliability of sensor communications in the presence of radio interference are critical. In this article, we propose to cope with this threat through a technique called channel surfing, whereby the sensor nodes in the network adapt their channel assignments to restore network connectivity in the presence of interference. We explore two different approaches to channel surfing: coordinated channel switching, in which the entire sensor network adjusts its channel; and spectral multiplexing, in which nodes in a jammed region switch channels and nodes on the boundary of a jammed region act as radio relays between different spectral zones. For coordinated channel switching, we examine an autonomous strategy where each node detects the loss of its neighbors in order to initiate channel switching. To cope with latency issues in the autonomous strategy, we propose a broadcast-assisted channel switching strategy to more rapidly coordinate channel switching. For spectral multiplexing, we have devised both synchronous and asynchronous strategies to facilitate the scheduling of nodes in order to improve network fidelity when sensor nodes operate on multiple channels. In designing these algorithms, we have taken a system-oriented approach that has focused on exploring actual implementation issues under realistic network settings. We have implemented these proposed methods on a testbed of 30 Mica2 sensor nodes, and the experimental results show that channel surfing, in its various forms, is an effective technique for repairing network connectivity in the presence of radio interference, while not introducing significant performance-overhead.

Categories and Subject Descriptors: C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; C.5.2 [**Computer System Implementation**]: Minicomputers

General Terms: Security, Reliability, Experimentation

Additional Key Words and Phrases: Jamming, Radio Interference, Channel Surfing

Author's addresses: W. Xu, Computer Science and Engineering, University of South Carolina, Columbia, SC, 29208; email: wyxu@cse.sc.edu; W. Trappe, Y. Zhang, WINLAB, Rutgers University, North Brunswick, NJ, 08902; email: {trappe,yyzhang}@winlab.rutgers.edu.

ACM Reference Format:
Xu, W., Trappe, W., and Zhang, Y. 2008. Defending wireless sensor networks from radio interference through channel adaptation. ACM Trans. Sens. Netw., 4, 4, Article 18 (August 2008), 34 pages. DOI = 10.1145/1387663.1387664 http://doi.acm.org/10.1145/1387663.1387664

1. INTRODUCTION

The reliable operation of a wireless sensor network is closely tied to the ability of the sensor radios to successfully communicate with each other. Considerable research effort has been devoted to tuning the operation of a sensor node's network stack, for example, Woo et al. [2003]; Polastre et al. [2004]; and Wan et al. [2003]. Most of this literature focuses the medium access control (MAC) layer, the link layer, and the network layer with respect to the issues associated with performing multihop data delivery within the various resource constraints that are uniquely characteristic of sensor networks. Collectively, this base of research provides a means for wireless sensor networks to reliably deliver sensed data as long as the radio environment is relatively benign. As wireless sensor networks become increasingly pervasive, however, it is very unlikely that the radio environment will continue to be so favorable.

There are several reasons for this gloomy forecast. First, as an increasing number of wireless devices are deployed (sensor or otherwise) that use open spectrum bands (e.g., the 900MHz, 2.4GHz, and 5GHz bands), it will be inevitable that there will be problems of spectrum coexistence. This is a well known problem for wireless communications in dense urban environments where cordless phones, which share the same spectrum, suffer degraded performance when many devices operate simultaneously within a small distance of each other. Such interference problems can be projected for sensor networks as more sensor devices are deployed in these bands.

Whether intentional or not, interference and jamming will be a serious threat to the reliable communication of sensor messages. The traditional approach to coping with radio interference is to employ more sophisticated physical-layer technologies, such as spread spectrum. Such methods, however, imply more expensive transceivers and, with the exception of some military systems, most commodity sensor and wireless networks do not employ sufficiently strong spreading techniques to survive jamming or to achieve multiple access. Instead, systems like the Berkeley Mica2, Zigbee (e.g., MicaZ), and even 802.11 are based upon a carrier-sensing approach to multiple access. Because of their use of carrier sensing for medium access control, these systems are particularly susceptible to radio interference or jamming. Recent studies [Law et al. 2005; Xu et al. 2005] have revealed the relative ease with which jamming can be conducted on such sensor networks.

In the absence of advanced physical layer techniques, a natural question that arises is whether the techique of changing the operating frequency can be migrated up the stack to the link layer. In this article, we examine the ability of a sensor network to cope with radio interference or jamming through the use of *channel surfing*, whereby the sensor nodes adapt their link layer frequency allocations in order to avoid interference. Unlike frequency hopping techniques at the physical layer, however, continually changing the frequency assignments at the link layer is a poor strategy for maintaining network services and instead, in our work channels are changed as needed based on the detection of interference, using techniques such as those in Xu et al. [2005]. The challenging research question here is how to establish network connectivity between multiple frequency zones. The inherent diversity in network configuration, interference model, and platform setup suggests that no single solution is sufficient. We have proposed four strategies to restore network connectivity across multiple channels, each having unique characteristics and advantages. Although channel surfing may be applied to more general wireless networks (e.g. 802.11), in order to validate our strategies, we focused our study on a sensor network platform, and have implemented our methods on a 30-node Mica2 sensor network testbed. In the process of implementation, we have overcome a number of challenges and have demonstrated that all four strategies can effectively maintain network operations in the presence of jamming/interference, with minimal false alarms.

We begin the article in Section 2 by providing an overview of the sensor network and interference model used in our studies. We next introduce the channel surfing strategy in Section 3 and Section 4, where we detail a set of increasingly sophisticated and powerful channel surfing protocols. In Section 5 and Section 6, we describe our validation effort on our sensor testbed. We wrap up the article by discussing related work in Section 7, and provide concluding remarks in Section 8.

2. SYSTEM MODELS

The objective behind this study is to examine networking issues associated with adjusting channel assignments in a sensor network in order to avoid radio interference. In this section we outline the basic sensor communication and jamming model that we use throughout this article.

2.1 Our Sensor Communication Paradigm

There are many choices for sensor platforms and data dissemination models available to the sensor network designer. The broad range of choices implies that there are many different directions that one can take in order to tackle the problem of radio interference. Early on in our studies we found that it was impractical to devise a generic approach that worked across all varieties of sensor networks, and instead found that it was necessary to tailor the design of our solutions to a specific communication paradigm.

Channel surfing requires that sensor radios change their channel allocations. In order to commence with channel surfing, the radio devices employed must have a notion of a channel. Most sensor platforms have a natural form of channelization that is accomplished by changing the carrier frequency. In our validation efforts, for example, we use the 916.7MHz Mica2 platform and separate the channels by 800KHz, effectively giving us 32 channels. In the algorithm discussion, we assume that channelization exists.

Another important factor in the sensor communication paradigm is the choice of the data dissemination method and the associated routing protocol. From a sink's viewpoint, there are two main data dissemination paradigms: few-to-one data dissemination (whereby a sink is connected to one source node or a small number of source nodes), as in Directed Diffusion [Intanagonwiwat et al. 2000] and SPIN [Heinzelman et al. 1999]; and many-to-one (whereby a sink node is connected to a large portion of a network), as in Zebranet [Juang et al. 2002] and TAG [Madden et al. 2002]. In this work, we have chosen to focus on the many-to-one model, and we assume that there is only one sink that collects data. For the many-to-one model, our studies focus on tree-based routing schemes, a popular family of routing protocols whereby the network establishes and maintains a forwarding tree [Madden et al. 2002].

We briefly touch upon the salient features of the tree-based routing schemes needed for our discussion. In these schemes, a routing tree is formed with the sink node serving as the root of the tree. A node selects its routing parent as its best radio neighbor in the direction of the tree's root. In such a tree-based routing structure, a node usually has a *parent* (except the sink) and one or more children (except the leaf nodes), wherein it receives data packets from its children, and sends packets to its parent. A node's parent and children are considered its neighbors. Besides the parent and children, there may be other nodes that are within a node's communication range. These nodes, are not neighbors of the node. In this article, we use the term "neighbors" to refer to the topology-based relationship rather than to a physical location-based relationship.

2.2 Our Interference Model

In considering the potential sources of radio interference, there is a very broad range of capabilities that one might assume for the interferer, ranging from whether the interference is incidental or intentional, powerful or resource-constrained, narrowband or broadband, or static or adaptive.

We note that if the jammer is a high-powered, non-coherent, broadband source of interference (e.g. capable of occupying all channels simultaneously), then there is no hope for building a resilient sensor network short of choosing a different PHY-layer transceiver with a powerful anti-jam margin [Proakis 2000; Schleher 1999]. Further, it should also be noted that an aggressive adversary may jam a single channel at a time (e.g., by reprogramming another sensor) and rapidly switch between channels to effectively disrupt network services across all channels. Both cases represent powerful, aggressive broadband jamming adversaries that represent adversarial cases.

Instead of considering powerful *attacker* interference models for which the only viable defense might be powerful physical layer techniques (or to localize the interferer and remove it from the environment), in this work, we consider an unintentional or a relatively *benign jammer* in which the interferer blocks one (or even a few) of the channels at a time. This model has been considered elsewhere in the literature [Xu et al. 2004; Xu et al. 2005; Law et al. 2005; Wood and Stankovic 2002], and it can be accomplished by employing a narrowband RF source (such as a waveform generator) or by another sensor node that has been reprogrammed to jam a particular channel. Further, even if the interferer hops to different channels, we assume that the interferer stays on one channel for a brief period of time before switching to another channel. Such an interference model could correspond to cases where protocol adaptation causes an interferer to accidentally follow the network to a new channel. Additionally, we assume that the network is able to exist for a period of interference-free existence in order to allow the network to stabilize.

Finally, we generally consider traditional security threats (such as authentication, communication confidentiality, and coping with node compromise) to be orthogonal to the issues discussed in this article. However, issues related to maintaining the randomness of the channel assignments used in channel surfing, and the authentication of channel switching commands, will be described where appropriate.

Even a relatively benign interferer that blocks one (or even a few) channels at a time can have a deleterious impact on the communication crossing the network. As we shall point out in the next section, our channel surfing algorithms operate on demand—when interference is detected, channel adaptation is used. Hence, as the starting point for coping with jamming, it is necessary to detect jamming. Since spectrum utilization is a local phenomenon, detecting the presence of a jammer must be done by each individual sensor node—that is, a node can only detect whether it is jammed, not whether other nodes are jammed. Several jamming detection approaches have been proposed, ranging from measuring simple properties such as ambient signal strength and packet delivery ratio [Wood et al. 2003; Xu et al. 2004], to performing more complicated consistency checks. In this article, we utilize our own detection scheme [Xu et al. 2005], which involves a consistency checking process on each node to ensure reliable identification of jamming attacks. We note that even if an interferer only partially degrades the network link quality (e.g., by blocking half of the packets), the classification of such a situation is the responsibility of the detection algorithm, and, of the policy the network operator uses to define "interference." The implication of this fact is that in the channel surfing algorithms we describe, we would like for the jamming/interference detection process to be a separate module that is utilized by the channel surfing algorithm. As long as a node is declared "jammed", channel surfing will kick in.

3. CHANNEL SURFING OVERVIEW

Typically, when radio devices communicate, they operate on a single channel. Here, the concept of a channel may be a single operating frequency or more generally may be any division of the operating spectrum. For the sake of discussion, we shall assume that the notion of a channel is associated with a single carrier frequency. Channel surfing is motivated by frequency hopping, a physical layer technique used in spread spectrum communication. As noted earlier, we assume that the interference blasts on only one channel (or at most a few channels) at a time.

The basic idea behind channel surfing is that the link layer channel assignments should be changed in order to avoid interference. When thinking of how to achieve this, a natural idea is to directly apply the philosophy of constantly changing the channels (as is done in frequency hopping spread spectrum). Employing such a strategy at the link layer, however, can be detrimental and costly to providing network services. First, if one rapidly changes the channel assignment, then it is necessary to have a fine granularity of synchronization across the entire network. Even if channels are changed less rapidly, for example on the order of a hundred milliseconds, constantly changing the channel incurs switching overhead. For example, routes that exist on one channel may not be guaranteed to exist on other channels, and frequently changing channels may cause the network to become unstable, thereby necessitating frequent route maintenance and discovery each time a channel is changed. Further, such penalties are incurred even if there is no interference presence.

Based on these arguments, a better strategy would be to adapt channel allocations only when needed, that is, when interference is detected. In order to achieve an interference-resistant sensor network, we propose a collection of distributed on-demand channel surfing algorithms. As the starting point for these algorithms, each node runs a jamming detection process to determine whether it is jammed. In channel surfing, those nodes that detect themselves as *jammed* nodes should immediately switch to another orthogonal channel and wait for opportunities to reconnect to the rest of the network. After the jammed nodes lose connectivity, their neighbors, which we refer to as *boundary* nodes, will follow Algorithm 1 to discover the disappearance of their jammed neighbor nodes (e.g. via a drop in link quality) and temporarily switch to the new channel to search for them. If the lost neighbors are found on the new channel, the boundary nodes will participate in rebuilding the connectivity of the entire network.

Before we move onto our specific algorithms, we first discuss a few challenging issues in the channel surfing framework. The first challenge concerns the potential boundary nodes. The basic scheme specifies that a boundary node

```
Algorithm: Channel Surfing Framework
while (1) do
    if (NeighborsLost() == TRUE) then
        working_channel = next_channel;
        for ($i = 0; i < m; i++$) do
            SendInquiryPacket();
            if (ReceiveReplyWithinT() == TRUE) then
                break;
            end
        end
        if ($i == m$) then
            working_channel = original_channel;
        else
            Use a Channel Surfing Strategy;
        end
    end
end
```

Algorithm 1. The channel surfing framework.

should switch to a new channel after its neighbors are jammed and have escaped to the new channel. However, if a node immediately probes the next channel whenever it experiences a poor link quality with any of its neighbors, the system will enter a non-stable state because wireless sensor networks inherently experience frequent link quality degradations or even topological changes [Zhao and Govindan 2003; Woo et al. 2003]. Fortunately, after carefully studying the working of the underlying system, we have found that it is possible for boundary nodes to correctly differentiate jammed neighbors from those neighbors that just had a poor link with the boundary node. This can be explained as follows. For tree-based routing, a node has precisely two types of neighbors/links: one link with its parent, and possibly several links with its children). Thus there are two possibilities for broken links and we can address each of these separately. First, if a node witnesses degradation in the link with its parent, then the underlying routing protocol will first attempt to find another, suitable parent node. If the node finds a replacement parent, then it will announce its parent selection in a routing update. Only if there is no suitable replacement parent will the node probe the next channel to find its parent. In this way, the node does not need to switch channels if it can still maintain its normal network operations. Next, we cover the case of a node losing one of its children. If the lost child node was not jammed, but just connected to a new parent, the node should hear its former child's routing announcement. If it does not witness the child's routing announcement within a specified window of time, then it will probe the next channel looking for its lost child. Thus, a node will become a boundary node either because it is a parent of jammed nodes, or because it is a jammed node's child that cannot find a new parent.

After detecting the loss of a neighbor, the boundary node should not switch to the new channel too quickly. If it switches too soon, it may arrive at the new channel before the jammed nodes. To understand this, consider a scenario where the jammer starts interference at time t_0. At that time, the jammed nodes will not be able to send out packets. However, since it takes less time for a node to detect the absence of a neighbor than it does for a node to decide it is jammed, the boundary nodes will detect the absence of a jammed node at time $t_0 + \delta_1$, while the jammed node will declare itself jammed at $t_0 + \delta_2$, where $\delta_2 > \delta_1$. If the boundary nodes switch to the new channel immediately after $t_0 + \delta_1$, they will not find the jammed nodes there. Rather than have the boundary node wait on the next channel, which would prevent it from conducting its primary objective of relaying messages to the sink, or having the node constantly flip-flop between channels looking for its children, we should make the boundary nodes wait an additional amount of time of at least $\delta_2 - \delta_1$ before switching to the next channel. The values of δ_1 and δ_2 are characteristic of the particular routing protocol, and of the jamming detection scheme.

In addition to the switch timing for a boundary node, it is important to have a discovery protocol by which a boundary node can find its neighbors on the new channel. After a node switches to the new channel to search for its neighbors, it sends out an inquirys message, such as "Is my neighbor X here?" If it receives a reply from X, it starts working on repairing the connectivity between X and the sink. Otherwise, it waits for time δ to send another message. If the node does not hear from X after a few trials, it assumes the child is not jammed, returns to the original channel and resumes its original operation in the network. In total, the time spent probing the next channel should be less than δ_2 to avoid cascading channel probing.

Finally, it is desirable to choose the next channel so that the interferer cannot predict what channel the nodes will surf to, thereby lessening the chance that an interferer could accidentally (or intentionally) follow the network to the next channel. We may choose to chain the channel selections using a keyed pseudo-random generator. If the nth channel assignment is $C(n)$, then we take $C(n+1) = E_K(C(n), r)$, where K is a key shared by all nodes in the network and used exclusively for channel assignment, and r is a *round* number that changes after 32 channel changes. Once the network has changed its channels 32 times, for example, spanning the 32 orthogonal channels of the Mica2 platform the network channel allocation starts another round. By doing this, the next round of the channel switching sequence will be different from the previous round (the previous 32 times). If ever $C(n + 1) = C(n)$, then the channel assignment proceeds to $C(n + 2)$ and so on until a different channel is selected. Finally, if the jammer can block several channels, then after a jammed node escapes to a new channel, it should first detect whether the channel is jammed before it starts working on that channel. In practice, one typically has to check at most a few iterations in order to find a new channel that is not jammed.

4. CHANNEL SURFING STRATEGIES

After the boundary nodes discover that their neighbors are jammed and have escaped to another channel, they will attempt to reconnect the jammed nodes with the rest of the network. In this article, we propose two different classes of techniques that the boundary nodes can use to repair network connectivity: (1) *coordinated channel switching*, in which the boundary nodes participate in transitioning the entire network to the new channel, thereby reestablishing the network on the new channel; and (2) *spectral multiplexing*, where the boundary nodes multiplex between the old channel and the new channel, serving as a bridge that connects nodes operating on different channels. In this section, we discuss these two strategies, outline their challenges, and highlight their advantages and disadvantages. Further practical issues are discussed in Section 5.

4.1 Coordinated Channel Switching

The idea behind coordinated channel switching is rather simple: the entire network must coordinate its evasion of the interference by switching to the next channel and resuming network operation there. These strategies are characterized by a transition phase during which an increasing number of nodes switch to the next channel. Following the transition, the entire network resumes stable operation on the next channel. Within the family of coordinated channel switching protocols, the strategies are characterized according to the coordination strategy governing the transition. In this article, we examine two different strategies: first, a technique whereby each node autonomously follows its neighbors to the next channel; second, a strategy whereby nodes are

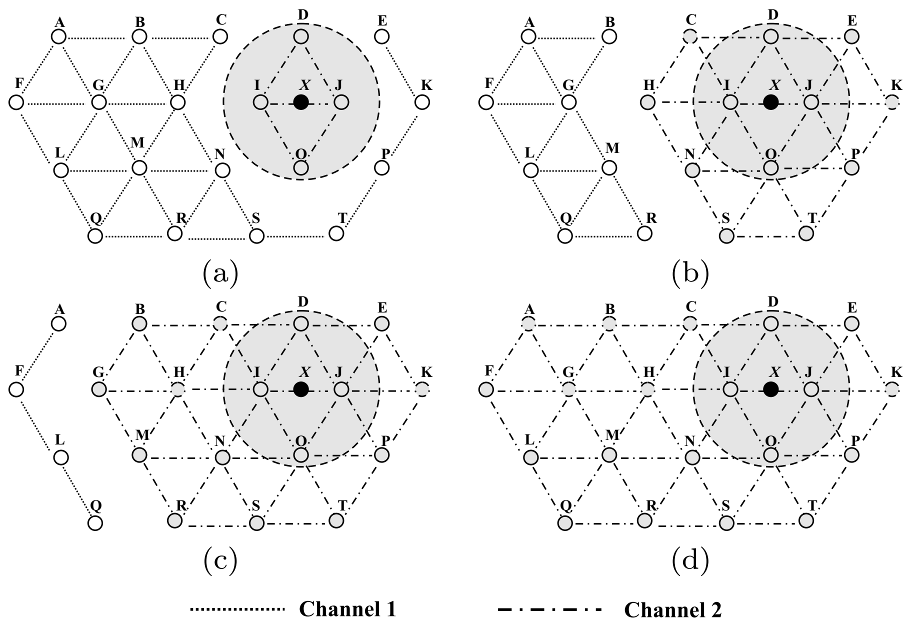


Fig. 1. A walk-through of autonomous channel switching. The shaded circle illustrates the jammed area, with the jammer X at the center. The entire network switches to channel 2 within four rounds, each round shown in one plot.

accelerated through the transition phase through the broadcasting of channel changing commands by the boundary nodes.

4.1.1 *Autonomous Channel Switching.* In the autonomous channel switching scheme, a node will autonomously switch to the new channel if it detects that some of its neighbors have moved to the new channel. In particular, each node is responsible for determining the next channel $C(n+1)$ it should switch to by using $C(n+1) = E_K(C(n), r)$, where r is the round number. The scheme begins with the jammed nodes detecting they are jammed. After a set amount of time, the boundary nodes will notice that they have not received messages from their jammed neighbors. The boundary nodes will then probe the new channel, searching for their lost neighbors. If the boundary node finds a neighbor residing on the new channel, it will stay in the new channel, extending the zone of nodes into the new channel. This process repeats until the entire network reconnects on the new channel. Overall, a network with n hops from the jammer to the boundary of the network, will take n rounds to complete the channel switching.

Algorithm walk-through. In order to illustrate the autonomous algorithm, let us walk through the example depicted in Figures 1(a)–(d). [1] Let channel 1

[1]We note that this example is merely for illustrative purposes, and thus we have depicted a grid-like topology to support discussion. In reality, the topology may be generic and, in our validation efforts later, we focus our experiments and implementation on a routing scheme that employs a tree topology.

be the old channel and channel 2 be the new channel. Here, the jammer X affects nodes $\{D, I, J, O\}$. Upon detecting that they are jammed, these four nodes switch to channel 2, as shown in Figure 1(a). The dashed-dot lines indicate links that exist in channel 2, while the dotted lines correspond to links in the first channel. The boundary nodes $\{C, E, H, K, N, P, S, T\}$ will notice that their jammed neighbors are no longer on channel 1, and will probe channel 2. When they discover their neighbors on the new channel, they will remain on channel 2 and form a network on channel 2, extending the size of the channel 2 subnetwork. In the next round, the channel 1 neighbors of $\{C, E, H, K, N, P, S, T\}$ will notice that some of their neighbors are missing and will probe channel 2. Upon finding their neighbors in channel 2, they will remain, and the channel 2 subnetwork will grow. This process repeats until the entire network has moved to the new channel. Including the jammed nodes, there are four hops from the jammer to the boundary of the network. Thus, after the jammed nodes evade to the new channel, we need three more rounds for the rest of the network to convene on channel 2, as shown in Figure 1(b), (c), and (d).

Algorithm challenges. This algorithm relies primarily upon the ability of each node to detect the absence of its neighbors, and to differentiate between whether a node is missing due to fluctuations in link connectivity, or from jamming. Hence, when implementing this algorithm, the issues identified in Section 3 become essential to the performance and operation of this scheme.

Discussion. One of the main advantages of this scheme is the simplicity of its description. Further, this scheme introduces minimal additional communication overhead. The nodes in the network merely detect the absence of their neighbors, and probe subsequent channels. Unfortunately, this scheme requires a long latency for the entire network to switch channels. If there are n hops from the jammer to the boundary of the network, then it will take a duration of at least $n\delta_2$ for the network to stabilize (recall from the discussion in Section 3 that we require boundary nodes to wait at least an additional $\delta_2 - \delta_1$). To differentiate between jamming and natural causes of poor link quality, it is necessary for δ_2 to be large. Thus, since the protocol latency is linear in the number of hops in the network (or roughly the square root of the number of nodes in the network), the issue of scaling becomes a dominant factor for this protocol. As the number of sensors in the network increases, the latency associated with the transition phase can become prohibitive and a large fraction of the sensor data will not be delivered.

4.1.2 *Broadcast-Assist Channel Switching.* Switching channels autonomously incurs significant latency because (i) a node can only switch after its neighbors have done so, and (ii) it must wait for some time even after its neighbors have switched. Broadcast-assist channel switching addresses both problems. Instead of requiring that every node detects whether its neighbors have switched, a boundary node that has found a jammed neighbor residing on the next channel will facilitate a more rapid phase transition by broadcasting a command. In particular, a boundary node switches back to the old channel, broadcasts a *channel switch command*, and returns to the new channel. Once

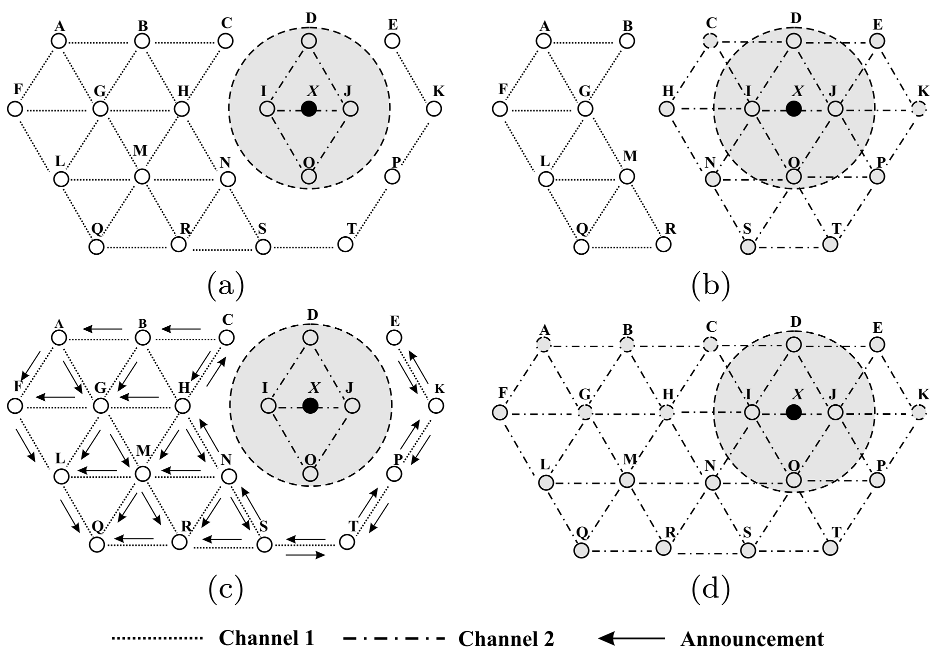


Fig. 2. A walk-through of broadcast-assist channel switching. The shaded area depicts the jammed area, with the jammer X at the center.

a node receives this notice, it rebroadcasts the command and switches to the channel specified. Overall, broadcast-assist channel switching facilitates parallel channel switching and, just as parallel execution is usually faster than sequential execution, it can significantly reduce the network switching latency.

Algorithm walk-through. We now examine the broadcast-assist channel switch algorithm using the same example as used to describe the autonomous scheme. Nodes $\{D, I, J, O\}$ are jammed by the jammer X and consequently switch to the new channel, for example, channel 2, as shown in Figure 2(a). As a result, nodes $\{D, I, J, O\}$ form the channel 2 subnetwork, and the rest of nodes form the channel 1 subnetwork. After time δ_2, the boundary nodes, $\{C, H, N, S, T, P, K, E\}$ notice that their jammed neighbors are no longer on channel 1, and probe them on channel 2 (Figure 2(b)). After finding the jammed nodes on the new channel, the boundary nodes return to the original channel temporarily and broadcast a switching notice $(n_{id}, C(n+1))$, where n_{id} is the ID of the sender node, and $C(n+1)$ is the new channel to switch to. The switching notice is sent to the rest of the network through the channel 1 subnetwork, as shown in Figure 2(c). The boundary nodes join the jammed nodes on the new channel after broadcasting the notice. Shortly thereafter, the rest of the nodes switch to the new channel after receiving the switching notice, and reestablish the network on channel 2, as shown in Figure 2(d).

Algorithm challenges. The major challenge facing this scheme is the fact that unreliable/variable links can cause some nodes to miss a channel switch notice. However, the channel switching command is typically broadcasted independently by multiple boundary nodes. Thus a node is very likely to receive at least one notice, and be able to switch to the new channel. Should a case arise where a node does not receive a switch notice, it will still autonomously move to the next channel as it will detect that it cannot receive messages from neighbors that have already switched to the new channel.

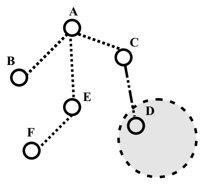
Discussion. Compared to autonomous switching, broadcast-assist channel switching incurs much less switching latency. After jamming is detected, the boundary nodes switch their channel within time δ_2, and then broadcast the switching notice. Suppose the network has n hops between the jammer and the boundary of the network, and that μ is the delay for one-hop transmission ($\mu \ll \delta_2$). Then the overall latency is roughly $\delta_2 + (n-1)\mu$, which is much less than the $n\delta_2$ latency required by autonomous channel switching.

Additionally, the success of broadcast-assist channel switching doesn't depend on the likelihood that each individual node will detect the loss of its neighbors. Rather, as long as one of the boundary nodes finds its lost neighbors in the new channel and informs the rest of network, the network will resume its connectivity in the new channel in spite of the radio interference. Finally, we note that the broadcasted channel switch command should be authenticated [Perrig et al. 2002] to prevent malicious message injection by an adversary. Thus, in practice, the channel switch notice has the format of $E_{K_A}(n_{id}, C(n+1), nonce)$, where K_A is a network-wide authentication key that is distinct from the key used to generate $C(n+1)$, and *nonce* is included to prevent replay attacks.

4.2 Spectral Multiplexing

Performing a coordinated channel switch requires the entire network to reestablish the routing tree since the link connectivity will not be the same on the new channel. The global nature of coordinated channel switching can be a source of significant network cost, and a natural alternative is to employ a local response where only jammed nodes switch channels, while non-jammed nodes remain on the original channel. To guarantee the communication between these two frequency zones, boundary nodes have to work on both channels by repeatedly switching back and forth between two channels to relay packets, a process we call *spectral multiplexing*.

Initially, for spectral multiplexing, jammed nodes that were originally on channel $C(n)$ will switch to $C(n+1) = E_K(C(n), r)$, for the round number r. However, for spectral multiplexing, the primary challenge lies in the fact that the boundary nodes must carefully decide on which channel they should stay, when, and for how long, so that they can minimize the number of packets that cannot be delivered due to the frequency mismatch between the sender and the receiver. If the boundary nodes are configured with dual radios, this scheduling is unnecessary. However, commercial sensor platforms including Berkeley motes only have one radio interface and, as a result, a node can work on only one channel at a time. It is therefore crucial to make sure that the sender and the receiver are able to work on the same channel when they want to exchange messages. Toward this end, boundary nodes must transmit an



(a) the network topology

slot 1	A, B, C, E, F
slot 2	C, D

(b) the global schedule

Fig. 3. Illustration of the synchronous spectral multiplexing algorithm.

announcement to their neighbors that they are operating in a “dual-mode” on two channels. Further, these boundary nodes must employ a synchronization mechanism to coordinate the spectral schedules of the sender and the receiver when one party needs to work in “dual-mode.” The overall scheduling objective is to ensure that dual-mode nodes are present on the correct channel when the neighbors on that channel are ready to transmit.[2]

In general, there are two ways of coordinating schedules from different entities: one is to have all the entities adopt synchronous schedules, and the other is to operate in an asynchronous fashion. We thus propose the corresponding methods to coordinate the frequency schedules for neighboring nodes: (1) Synchronous Multiplexing, in which all the nodes share the same schedule by dividing the global time axis into different slots and assigning one slot to a channel; and (2) Asynchronous Multiplexing, in which a node operates on a local schedule, and the boundary nodes make local decisions about when to switch channels.

4.2.1 *Synchronous Spectral Multiplexing.*

In synchronous spectral multiplexing, the entire network is governed by one global clock. The global time axis is divided into slots, and multiple slots form a round. The number of slots in a round is determined by the number of channels the network has to operate on at any specific time. (In this article, we limit our discussions to situations where the network operates on 2 channels simultaneously, but the discussion can be easily extended to cover situations with more than 2 channels.) Each slot is assigned to a single channel, and during that time slot, the nodes in the network may only use the corresponding channel, regardless of whether they are jammed nodes, boundary nodes, or other nodes. At the end of a time slot, the entire network utilizes the next channel and, again, the nodes that are not using the next channel do not transmit, nor must they switch channels unless they are dual-mode boundary nodes. By following this global schedule, we can avoid frequency mismatch between a pair of communicators.

Algorithm walk-through. Figure 3(a) presents an example network scenario in which D is jammed, and switches to channel 2. Its parent node, C, thus becomes a boundary node, and has to multiplex between two channels. The rest of the nodes continue to work on channel 1. The global schedule for this case is shown in Figure 3(b), which has two slots for each round, with slot 1 allocated

[2]The need for scheduling transmissions has also been considered in the context of duty cycling, as in S-MAC[Ye et al. 2002], in order to preserve energy.

to channel 1, and slot 2 to channel 2. Following this schedule, during slot 1, nodes $\{A, B, E, F\}$ work as normal. Node C sends out packets to its parent A, but does not receive any packets from D. At the end of slot 1, these nodes stop their activities on channel 1, and node C switches to channel 2. During slot 2, the only transmitting node is D, and C buffers all the packets it receives from D. At the end of slot 2, D ends its transmissions and C switches to channel 1. These two slots keep alternating in this fashion until the radio interference ends, or for the lifetime of the network.

Algorithm challenges. There are several challenging issues associated with this scheme: (1) How to synchronize the schedules of every node, (2) how to start multiplexing, and (3) how to determine the slot duration?

Synchronization. One natural synchronization approach is to have the entire network work under a global synchronized clock, wherein each node maintains a unique and global timescale. Many protocols can be used to establish a global timescale across the entire network, for example, TPSN (Timing-sync Protocol for Sensor Networks) [Ganeriwal et al. 2003]. A closer investigation, however, reveals that perfect synchronization across the entire network is not only inefficient, but also unnecessary. Instead, since communication takes place locally among neighboring nodes, we can focus on achieving a fine synchrony within any local region. Additionally, instead of employing traditional pairwise synchronization, we let the root initiate the synchronization process by broadcasting SYNC packets to its children.

Nodes use a timer to start/end a slot; at the beginning of a slot, a node sets its timer to the duration of a slot, and when the timer expires, a node switches to the next slot. Without periodic synchronization, the timers on different nodes may drift significantly. To avoid this, *SYNC* packets are sent periodically at an interval much larger than the slot duration. Upon receiving a *SYNC* packet, a node will immediately terminate the current slot and start a new slot by resetting the timer to the slot duration. This simple protocol can effectively minimize the synchronization error between a pair of neighbors. The resulting synchronization error, $\Delta\tau$, only includes delays involved in sending, propagating, and receiving *SYNC* packets.

Building a global synchronization from a local synchronization protocol requires a starting reference point that initiates the synchronization process. In a tree-based forwarding structure, it is natural to choose the root of the tree as the reference point. Therefore, in the first round, the root sends out *SYNC* packets to its children, whose depth is 1, and whose clocks will thus be synchronized within $\Delta\tau$ of the root's clock. Similarly, in the $(i+1)^{th}$ stage, the nodes with depth i send *SYNC* packets to their children. Finally, after every node receives a *SYNC* packet, for a tree with depth of n, some leaf nodes will be $n\Delta\tau$ behind the root. This synchronization procedure is presented in Figure 4. After synchronizing, each node will start a new slot upon the expiration of its timer. However, in order to compensate for the synchronization error between its time and its neighbor's time, we require that a node wait for a short, random time period prior to transmission.

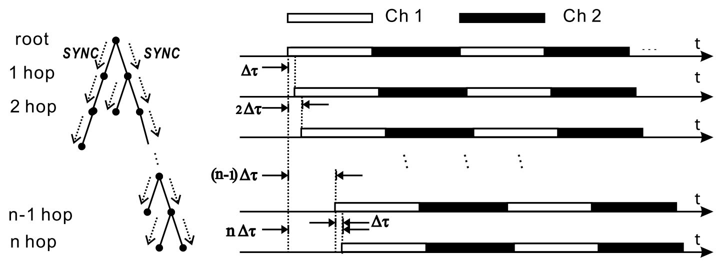


Fig. 4. Illustration of the synchronization mechanism.

Further complicating synchronization is the fact it must be maintained when nodes work on different channels (coping with drift while multiplexing). For example, in the case where the parent node is on channel 1, while the child is on channel 2, the *SYNC* packet from the parent will be lost. To address this complication, the *SYNC* packet should specify which channel the current slot is associated with. In order to guarantee that nodes on both channels are synchronized, the node should send these *SYNC* packets in rapid succession across both channels. This mechanism ensures that the *SYNC* packets will reach the destination.

Initiation. As soon as a boundary node discovers that jammed nodes have evaded to the new channel, it will send a message to the entire network on the original channel that contains a list of the channels it will be working on. After a node receives this message, it will compare its own channel with the channel list included in the message, and append its channel if it is not already in the list. In this way, when the message reaches the root of the routing tree, it will contain all the channels the network has to operate on, and the root will create a slotted channel schedule based on this list, and broadcast the schedule down the tree, along with the clock synchronization packets.

Slot duration. Slot duration is an important parameter in the synchronous spectral multiplexing algorithm. At first glance, it seems intuitive that a shorter slot duration is more desirable, because if a node stays on one channel briefly enough, the required buffer space will be smaller and, more importantly, less latency will be incurred. However, we have found that a smaller slot can be problematic as well, mainly due to the overhead associated with switching channels. Before a node switches to a new channel, it has to finish receiving all the packets that are in transmission. In order to guarantee this, after the timer expires, we let each node wait for a small amount of time for all the possible transmissions to complete. In our implementation, this translates into the parent node waiting a little longer because the receiving side is usually the parent node in a tree-based routing. Another problem with short slot durations is that the synchronization errors will be proportionately large with respect to the duration of a slot, thereby affecting this scheme's efficiency. Finally, we note that there is a radio startup cost associated with switching channels (e.g., 250msec for the CC1000 radio chip in the Mica2 mote). Overall, a good slot duration should be determined based on several factors: the available buffer space, the traffic rate, the required message latency, and the synchronization error.

In this article, we choose to adopt the largest slot durations that can satisfy the available buffer space constraints, and we consider our underlying sensing application model to have a periodic traffic pattern. Further, we have empirically witnessed that a boundary node is more likely to be a parent than a child of a jammed node (children of jammed nodes typically look for alternate parents on the original channel), and thus boundary nodes will merely receive on channel 2, but will both receive and send on channel 1. Specifically, based on the traffic rate from each channel and the buffer size, each boundary node calculates the longest stay time it can have on each channel (usually a node should stay on each channel for the same amount of time). In order to understand the calculation, let us look at an example. Suppose a boundary node A has a buffer that can support 10 slots, and it has one child on channel 1 that produces 20 packets per second, and two children on channel 2 each of which produces 10 packets per second. Node A can at most stay on each channel for 250 msec. By spending 250 msec on each channel, it receives 10 packets in a round (5 from each channel), which fills up its buffer. After each boundary node independently calculates a slot duration, the sink collects all the information, chooses the smallest one as the global slot duration, and announces this.

Discussion. Synchronous multiplexing adopts a deterministic global schedule that governs the channel assignment of every node in the network. The deterministic nature of this algorithm guarantees that it can work well even under complex scenarios where multiple nodes need to work on multiple channels and these nodes are neighbors of each other. However, in order to achieve this, every node in the network must pay the extra overhead needed to maintain synchrony between nodes.

4.2.2 *Asynchronous Multiplexing.* In the asynchronous multiplexing algorithm, a node is only aware of its neighbors' channel information, but not the channel information of a remote node. The simplest spectral scheduling method is to have a boundary node flip its radio frequency between two channels in a round-robin fashion. However, a completely random round-robin multiplexing strategy ignores the schedules of the communicating parties, and would thus fare poorly. For example, suppose a jammed node, working on the new channel, sends packets at times 10, 20, 30, 40, and 50. If the corresponding boundary node stays on the new channel during time windows [1, 6], [13, 18], [25, 30], [37, 42], [49, 54], then it will miss the packets sent at 10, 20, and 30. The resulting packet loss ratio for the jammed node would then be as high as 60%. This example illustrates the limitation of a random round-robin scheme and highlights the need for some level of coordination between the boundary node and its neighbors for the asynchronous multiplexing scheme.

Algorithm walk-through. Figure 5 illustrates the idea behind the asynchronous multiplexing scheme. In this example, the boundary node A has to receive packets from three nodes, B, C, and D, with the first two working on channel 1 and the last one working on channel 2. Suppose all three nodes send

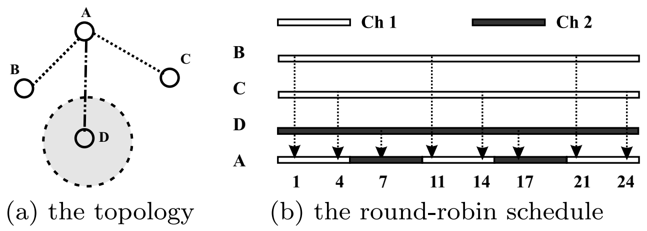


Fig. 5. Illustration of the round-robin asynchronous spectral multiplexing algorithm.

packets every 10 seconds, starting at time 1, 4, and 7 respectively. In this case, starting from time 0, A decides to stay on one channel for 5 seconds and then switches to the next channel for 5 seconds. In this way, A can receive every packet from its neighbors.

Algorithm challenges. The challenges associated with these schemes include synchronization and slot duration.

Synchronization. To coordinate the schedules of a boundary node and its children, we have adopted a simple protocol that involves the boundary node announcing its schedule (the duration it will stay in each channel) by notifying its children just after it switches to a new channel. In the example in Figure 5(a), A notifies nodes B and C of its schedule as soon as it switches to channel 1, so that they can start transmissions when A enters channel 1, and stop transmissions after A leaves channel 1. Similarly, it also notifies D whenever it is on channel 2. We note that a child node must buffer both its own packets as well as packets coming from its own children while waiting for the dual-mode parent to return to the channel it is working on. To counteract the possibility that the notifications could be lost, a child should start to send its buffered packets immediately after it hears from its parent.

Slot duration. Determining the slot duration in asynchronous spectral multiplexing is easier than in the case of synchronous spectral multiplexing, because in the former situation, not only can different boundary nodes employ different schedules, but they can also stay for a different amount time on each channel. For example, considering that the boundary node is usually the parent of jammed nodes, the boundary node should stay longer on channel 1 than on channel 2, as it has to forward all the packets received in both channels to its parent via channel 1.

Due to the nature of asynchronous spectral multiplexing, nodes can determine their slot durations in a more flexible fashion. Suppose a boundary node decides to stay on channel 1 for t_1 time and channel 2 for t_2 time ($t_1 \geq t_2$), where t_1 and t_2 are chosen according to the traffic volume on each channel and its buffer size. For example, it can choose to have $\frac{t_1}{r_1} + \frac{t_2}{r_2} = B$, where r_1 and r_2 are the traffic rates on the two channels respectively, and B is the buffer size. After setting this baseline schedule, the boundary node can adapt its switching rate as a response to varying network conditions (e.g., topology change, traffic rate change, etc).

Discussion. Compared to synchronous multiplexing, asynchronous multiplexing does not maintain a global schedule, and thus incurs less synchronization overhead. The advantage of asynchronous multiplexing, however, is more pronounced when the jammed region is small and regular. For larger jammed areas, we will have more boundary nodes that work on multiple channels. In this case, the overhead gap between synchronous and asynchronous techniques lessens. A final advantage of the asynchronous method is its ability to adapt to local traffic and buffer conditions.

5. SENSOR TESTBED AND METRICS

We now focus our discussion on our experimental validation efforts. We have built a 30-node mote testbed, and have conducted numerous experiments with the testbed to evaluate the effectiveness of the four channel surfing strategies in providing interference-resistance.

5.1 Testbed Configuration

We have built our sensor network testbed using 30 Mica2 sensor motes. These devices each have a 902–928 MHz Chipcon CC1000 radio. We used 916.7MHz as the original channel and separated our channels by 800KHz, effectively giving us 32 channels. The justification for using 800KHz channel separation is to ensure orthogonality of the medium access control layer. In fact, as noted in Xu et al. [2004], although the specifications for the radios suggest that much smaller frequency separation is needed for orthogonality, practical experience indicates that a larger separation is needed to ensure minimal interference between radios on different channels. The operating system running on each mote was TinyOS version 1.1.7 [TinyOS]. We attached one of the motes to a MIB510CA programming board in order to act as the network sink. In order to conduct experiments that exhibit repeatable characteristics, we chose an indoor laboratory area where we could fix the deployment across the experiments, as illustrated in Figure 6. However, due to our space limitations (there are walls just beyond the boundary of the picture), we were forced to reduce the radio range of each mote: in the depicted configuration, we have a node separation of roughly 2.5 feet. We tuned the transmission power of each mote down to −5dBm in order to restrict the radio range of each sensor node and to increase the network hop count.

The primary objective of this article is building a jamming-resistant sensor network, and we have focused our efforts on exploring how a networked system can maintain connectivity in the presence of jamming/interference, regardless of the type of the application data. As a result, we have not attached any specific sensors to the motes. Rather, we modified the Surge application, which comes with TinyOS, to convey experimental statistics. By default, Surge uses a tree-based routing algorithm (which we shall call $SP(t)$) for a single network sink, as detailed in the `MultiHopRouter.nc` file in the TinyOS 1.1.7 release. Since our focus was on networking issues associated with jamming resistance, we did not employ message acknowledgements or retransmissions. In addition to this basic communication model, we note that each sensor message contains

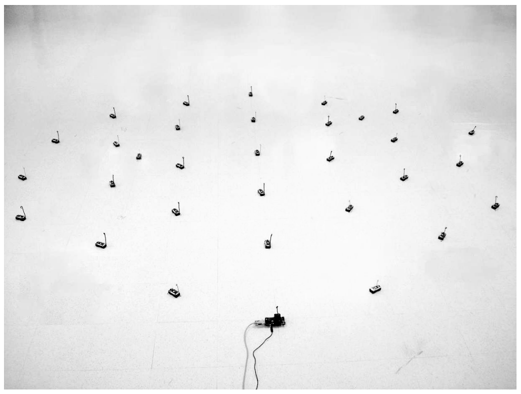


Fig. 6. Our Mica2 testbed consists of 30 motes that are placed on the floor. Nodes are roughly separated from each other by 2.5 feet. The sink is located at the bottom of the figure, with a programming board attached to it.

a sequencing field (in the routing header) that can be used to estimate performance statistics, such as link quality. Finally, we note that our packet size was 32 bytes, and that a node can buffer at most 24 packets (across all channels).

5.2 Implementation of a Basic Sensor Network

Before we could develop the proposed channel surfing strategies on the testbed, we had to modify the existing TinyOS code to address a number of implementation-related issues. In this section, we list a few of the more relevant issues.

The first challenge is devising a reliable link quality estimation technique since this directly affects the efficiency of the routing process. A good estimation technique must be both accurate and resilient to fluctuations. For example, our routing protocol maintains the tree-based topology based on the measured link quality: a node chooses the neighbor that has the best link quality as its parent. Similarly, if a node observes low link quality from the current parent, it will attempt to choose a new parent. On the other hand, due to the low-power and low-fidelity nature of the Mica2 radio, the resulting wireless link quality usually fluctuates greatly, which makes measuring link quality a challenging task.

The existing estimation method utilizes the windowed mean with exponentially weighted moving average estimator, where link quality is defined as:

$$\theta_{new} = \frac{N_{recv}(t)}{\max(N_{exp}(t), N_{recv}(t))} \tag{1}$$

$$\theta = \theta_{old} \times \alpha + \theta_{new} \times (1 - \alpha). \tag{2}$$

Here θ_{new} is the currently estimated value, $N_{recv}(t)$ is the number of received packets within time t, and $N_{exp}(t)$ is the number of expected packets within time t. In Equation 2, the value of α governs how much the past estimation affects the current estimation. The default value for α is set to 0.25 in TinyOS 1.1.7, but we found that this value is too low and the estimated link quality was overly variable. Therefore, the routing topology changed frequently and was hard to stabilize. Through our experiments, we found that $\alpha = 0.75$ yields much better estimates, which incidentally agrees with the value recommended in Woo et al. [2003].

After adopting suitable parameters for estimating link quality, we faced the challenge of choosing a parent node. A node chooses its parent based on two criteria: the hop count from the sink, and the estimated link quality. Ideally, the parent node should have the smallest hop count and the best link quality. However, in reality, nodes that satisfy both criteria may not exist, so we have to prioritize them. In our implementation, we give more importance to link quality than to hop count. For example, we specify that a parent node must have a link quality better than 75% (compared to 10% in the original $SP(t)$). At the same time, we avoid frequent parent switching by adopting the rule that a node can only choose a new parent when the resulting link quality is at least 20% better than the quality of the current link.

In the TinyOS 1.1.7 release of Surge, the application attempts to push packets to the network queue at a fixed rate. However, if the send operation of the previous Surge packet is incomplete, that is, the `SendDone` has not been received, then the latest packet is discarded. This posed a problem for us as it did not guarantee that a fixed number of traffic packets would reach the network buffers. In order to guarantee a nominal data rate, we modified Surge to push data regardless of whether the `SendDone` callback was received.

5.3 Implementation of the Channel Surfing Framework

Beyond building a basic sensor network, we changed components in TinyOS to support channel surfing. The addition of components was carried out in a manner to be generic and support all four of the strategies described in this article. Specifically, in adapting TinyOS to support channel surfing, there were many implementation issues that had to be addressed.

We began by modifying the buffering mechanism in `QueuedSendM` to address the fact that buffered packets may need to be sent on different channels. We associated each buffered packet with an identifier indicating the channel on which that packet needed to be transmitted.

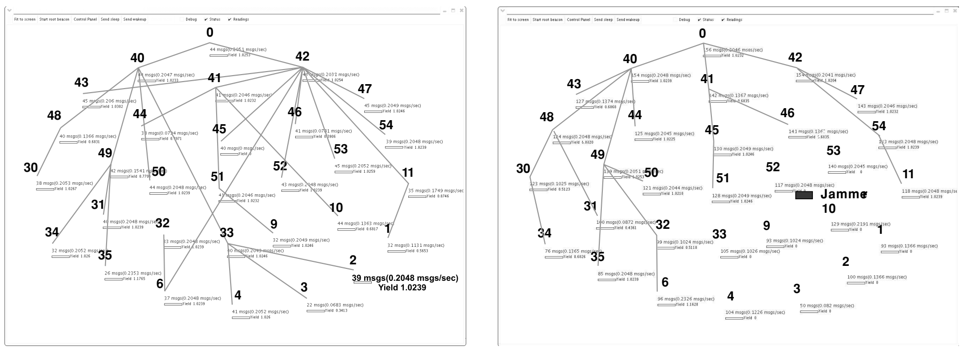
We next altered the replacement policy of the mote neighbor table. Each Mica2 mote maintains a neighbor table recording the link quality between each node in its radio range (e.g., *radio neighbors*) and itself. Though a node may have many radio neighbors, especially in a dense deployment, the motes in our testbed only have 16 entries in their neighbor table. As a result, an appropriate buffer replacement policy must be employed to sort a node's radio neighbors. The default replacement policy in TinyOS sorts the radio neighbors based on the link quality between each radio neighbor and the considered node. This policy, however, must be modified to implement channel surfing strategies. To understand this, suppose node A's child, B, is jammed. A is supposed to find that the link quality between B and itself has degraded, and then probe the next channel to search for B. However, since B is jammed, the estimated link quality between A and B may be so low that B is evicted from A's neighbor table. In this case, A loses awareness of the existence of B, and will not look for it on the new channel. In order to address this problem, we modified the replacement policy so that it always sorts a node's topological neighbors ahead of non-topological radio neighbors. Hence, we can ensure that a jammed node stays in its former parent's neighbor table until the former parent finds out that it is jammed.

The third issue is that it is necessary to revisit the problem of link quality estimation, and make some further modifications. Estimating the link quality involves comparing the number of packets a node receives with the number of packets it expects, by using the sequence numbers of the packets. However, when we operate on multiple channels, this estimation mechanism may cause problems because a dual-mode node will have roughly a 50% link quality, and thus nodes on either channel will avoid choosing the dual-mode nodes as their parents, hindering the realization of spectral multiplexing. We address this by assigning independent sequence numbers on different channels, so that the estimated link quality for a dual-mode node on both channels is sufficiently high.

One additional challenge we faced had to do with the formation of loops in the channel surfing prototypes. Although it is possible for loops to form in the original shortest path routing scheme, it is highly unlikely that they will form, since parents are selected using a combination of link quality and hop counts. Loops can form in channel surfing, however, because a node's ancestor might be in a different channel, thus invalidating the prior hop count measurement used in selecting a new parent. This situation can be easily addressed by requiring nodes that move to the next channel to reinitialize their hop counter, and by placing an upper bound on the hop count allowed for network nodes.

5.4 Performance Metrics for Channel Surfing

The overall impact of channel surfing strategies on a network can be examined in two aspects: the additional overhead it may introduce when the network is in its normal state (i.e., no jamming or radio interference), and the benefit it may have when the network experiences radio interference. To minimize the protocol overhead, we designed the channel surfing strategies so that the protocols do not rely on extra beacons or messages. Instead, we reuse the beacons and the link estimation methods employed in the routing layer. Thus, if the network is in a normal state, it will have comparable performance regardless of whether it employs channel surfing or not. Meanwhile, if the network undergos some form of radio interference, channel surfing can immediately take effect to repair the network connection. Finally, a gray area in which channel surfing may impose unnecessary overhead is caused by possible false positives, when channel surfing reacts to neighbor losses due to non-interference reasons. Nonetheless, as pointed out in Section 3, we minimize the likelihood of having false positives by only checking a lost neighbor in the new channel under very few circumstances, and by adopting a large time threshold. As a result, we believe the proposed channel surfing strategies are both effective in interfered situations, and unintrusive in normal situations. To demonstrate these points,



(a) (b)

Fig. 7. The network topology for jamming experiments. (a) Prior to introducing the jammer, (b) shortly after the jammer is introduced.

we choose to use the following two performance metrics:

—*Network recovery:* The main objective of channel surfing is to repair normal network functionalities in the presence of jamming or radio interference. Therefore, we measure their merit by comparing network characteristics in the following three scenarios: (1) under normal conditions, (2) under jamming but without channel surfing, and (3) after applying channel surfing strategies. Specifically, we focus on two network characteristics: network performance (number of packets delivered to the sink), and the latency required to recover network performance.

—*Protocol overhead:* Another class of metrics are related to the overhead introduced by the channel surfing strategies. One obvious concern is that nodes may unnecessarily switch channels. As a result, we have measured the number of channel switches every node experiences throughout the duration of an experiment.

We recorded the packet delivery statistics at the sink by using the sequencing field to maintain a running record of the packets that the sink receives from each sensor. Additionally, as noted earlier, we utilized the sensor messages themselves as a means to measure experimental statistics, such as the operating channel for each sensor node.

6. EXPERIMENTAL RESULTS

We deployed the testbed as illustrated in Figure 6, and the resulting tree-shaped routing topology (Figure 7(a)) was captured using the Surge Network Viewer. In the routing tree, the root node (node 0) corresponds to the network sink. Across a different set of experiments, we placed the jammer in different locations to demonstrate the generality of our channel surfing solutions. We note that we observed comparable performance across all nodes, but due to space limitations we present results for six nodes representing specific, interesting scenarios that reveal relevant protocol behaviors.

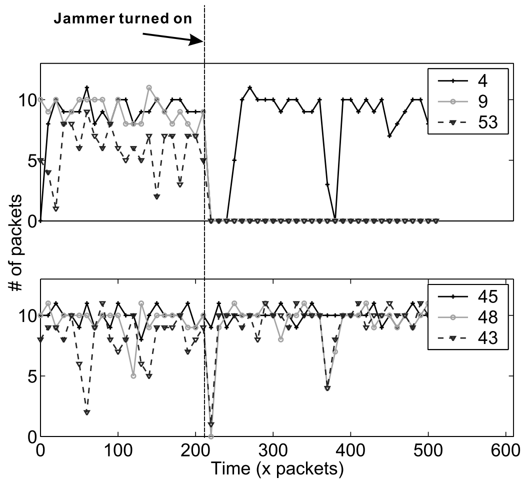


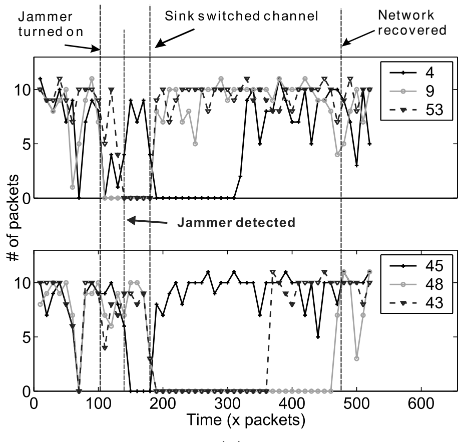
Fig. 8. Packet delivery time series for a 50-second time window under first normal and then jammed network conditions.

6.1 The Impact of Jamming/Interference

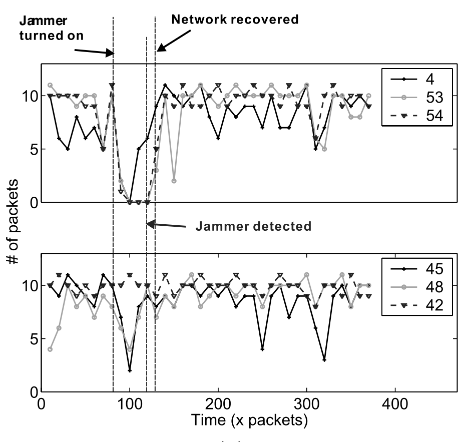
We first conducted an experiment to study the network behavior under normal indoor conditions as well as the impact that a single jammer can have on the network. In this experiment, we first let the network run for more than 20 minutes prior to introducing a jammer. For our jammer, we used the same device as legitimate nodes, thus the jammer can only jam one channel at a time. In particular, we used the constant jammer of Xu et al. [2005], which is a mote that bypasses the MAC-layer to continually transmit random bits into the network. Figure 7(b) shows the location of the jammer, and the fact that the jammer destroyed the connections between several nodes and the sink. Specifically, in this example, nodes $\{9, 10, 33, 52, 53\}$ lost their connections because they were directly affected by the jammer, while nodes $\{1, 2, 3, 4\}$ lost their connections because their parents were jammed.

We next take a closer look at how the packet delivery quality between each node and the sink evolved with time, both with and without jamming. We present time series for the number of packets delivered to the sink in a window of 10 packet from six nodes in Figure 8. In these results, each node generated and sent packets at a rate of 1 packet every 5 seconds. Under perfect network conditions, we expect each node to deliver 10 packets in a 10-packet window, but even under normal network conditions prior to jamming, the traces exhibit short term temporal fluctuations.

We note that, after introducing the jammer, nodes 9 and 53 were not able to deliver packets to the sink. As discussed earlier, that is either because these nodes were jammed, or because all their possible parent nodes were jammed. As a result, the packet delivery ratio became 0. We note that although node 4 lost its former parent due to jamming, it later found a replacement parent on channel 1 and hence its packet delivery became normal after a short interruption.



(a)



(b)

Fig. 9. (a) Packet delivery time series for a 50-second time window for the autonomous channel surfing strategy. (b) Packet delivery time series for the broadcast-assist strategy.

6.2 Coordinated Channel Switching Results

The first set of channel surfing results that we provide are for the two coordinated channel switching protocols described in Section 4.1, in which the entire network changes its operating channel to a new channel.

6.2.1 *Autonomous Channel Switching.* We conducted an experiment to study how autonomous channel switching repairs a jammed network. In this experiment, we introduced the jammer after the network had run for 10 minutes. As soon as jamming was detected, the autonomous channel switching strategy worked to repair the network topology so that all of the nodes switched channels, forming a new routing tree on channel 2 (not illustrated due to space limitations).

The packet delivery time series for the experiment are presented in Figure 9. This plot presents the time series of the number of packets delivered to the sink in a 10-packet window, from nodes $\{4, 9, 53, 45, 48, 43\}$. These six nodes represent some jammed (9 and 53) and some unaffected nodes. The autonomous algorithm has different impacts on different nodes, depending on their locations in the topology:

(1) Before the sink switches to channel 2, those nodes that are not affected by the jammer will still continue delivering packets on channel 1, thus maintaining a good delivery ratio. Please refer to the traces for nodes 4, 48, and 43 between the 100th interval (when the jammer was turned on) and the 174th (when the sink switched channels).
(2) Nodes that switch channels before the sink, either because they are jammed (i.e., node 9) or because they detect the loss of their neighbors before the sink (i.e., node 45), are able to resume packet delivery as soon as the sink switches. Please refer to the traces for nodes 9, 45, and 53 after the 174th interval, when the sink switched channels. Of course, these nodes usually suffer disrupted network services before the sink switches.

(3) After the sink switches to channel 2, those nodes that were working on channel 1 suffer from low delivery ratios until they detect that some of their neighbors have evaded, and they then decide to switch. For example, nodes 4, 48, and 43 take a long time to reconnect to the sink. Among these three nodes, node 4 switches to channel 2 faster than the other two because it is closer to the nodes using channel 2.

Looking at traces from different nodes in the network, we can conclude that how fast a node adopts a new channel in a jammed scenario depends on several factors: (1) whether the node is jammed or not; (2) whether the node switches channels before the sink; and (3) if the node was unaffected by the jammer, its topological distance from the sink. We note that, if the percentage of nodes that are directly affected by the jammer is larger, then the network can switch to a new channel faster. Otherwise, it may take a long latency for some nodes to join the rest on the new channel. In this particular example, node 48 required 290 packet intervals to reconnect after jamming occured.

Another important statistic is how many times a node switches channels throughout the experiment. A good channel surfing scheme should try to minimize this amount. Autonomous channel switching incurs only 1 channel switch for every node. This is due to its simplistic nature and to the assurance that a node does not probe the next channel too soon. The latter is guaranteed by the following two facts: (1) in the case of a disappearing child, the underlying routing infrastructure can help a node differentiate a child that has evaded to the next channel, from a child that has chosen another parent node; and (2) in the case of a disappearing parent, a node simply chooses another parent without probing the next channel, and only probes when there is no legitimate parent candidate.

6.2.2 *Broadcast-Assist Channel Switching.* Next, we conducted experiments to study the effectiveness of the broadcast-assist channel switching method. Just as in autonomous channel switching, we observed that the broadcast-assist channel switching method completely restores connectivity for every sensor to the network sink on the new channel. As expected, compared to autonomous channel switching, broadcast-assist channel switching can significantly reduce the transition latency for the entire network to escape to the new channel, as indicated by the packets-delivered time series shown in Figure 9. Here we selected a sampling of six nodes and observed that, regardless of a node's position, they can resume operations on the new channel quickly and almost at the same time. The switching latency is roughly the sum of jamming detection latency, probing time, and broadcast latency. Specifically, the transition phase only took 46 packet intervals, and 39 out of the 46 intervals were used for the jammed nodes to detect that they are jammed, as well as for the boundary nodes to discover that their child nodes are missing. We would like to emphasize that we purposefully let a node wait for a rather long time period (i.e., 39 packet intervals) before probing the next channel after it detects a poor link quality between itself and its children. We take the viewpoint that sensor networks will experience many more temporary topological changes than longer-term jamming/interference, and that we therefore must reduce the false positives of channel probing to achieve more stable network operations. Also, we would like to point out that even with such a conservative approach, we could improve the recovery process by having jammed nodes buffer packets during the network transition periods.

We also measured the total number of channel switches for these six nodes versus time. As expected, we saw that, prior to introducing the jammer, no nodes switched channels because our algorithms were tuned to have very few false positives when determining whether to probe the next channel. As soon as the jammer started, since nodes 53 and 54 were directly affected, they switched channels, and stayed there afterwards, thus switching only once. Node 42, however, reported three channel switches because it was a boundary node that first switched to channel 2 to probe its child, then switched back to the original channel to broadcast the switch notice, and finally switched back to the new channel to resume network operations. Other boundary nodes exhibited similar behavior, while more distant nodes only switched once as a response to the switch notice. Overall, broadcast-assist channel switching incurs very few channel switches for each node. Finally, we note that we conducted experiments with multiple simultaneous jammers in different positions, and observed that the broadcast-assist method was able to repair the network in these cases with latencies on the order of 50 packet intervals.

6.3 Spectral Multiplexing Results

We now examine results for spectral multiplexing. In these methods, we only switch channels for the jammed nodes while a subset of nodes multiplex between the original and new channels. We note that it is unfair to compare coordinated channel switching with spectral multiplexing under the same network configuration, because these two strategies are complimentary to each other, each designed to deal with different situations. Specifically, the coordinated strategy is suitable for cases where a large region of the network is jammed, while the multiplexing strategy is suitable for cases with much smaller jammed regions. Thus, for spectral multiplexing, we block fewer nodes in order to have a smaller jammed region. One further difference between these two types of strategies is that spectral multiplexing typically requires more buffer space for storing packets during multiplexing. Given the limited buffer space on the motes, we chose to adopt a lower data rate (1 packet every 10 seconds) than the rate used in earlier experiments.

6.3.1 *Synchronous Spectral Multiplexing.* In the synchronous spectral multiplexing experiment, we used the same general layout as shown in Figure 7(a). After the network had run for 40 packet intervals, the jamming/interference process began. The jammed region consisted of nodes 52, 53, and 10, and these three nodes promptly switched to channel 2. After the jammed nodes evaded to the new channel, their former parents detected their disappearance, and became the boundary nodes. Nodes 41 and 45 were such examples because they used to be the parents for nodes 10 and 52, respectively. The

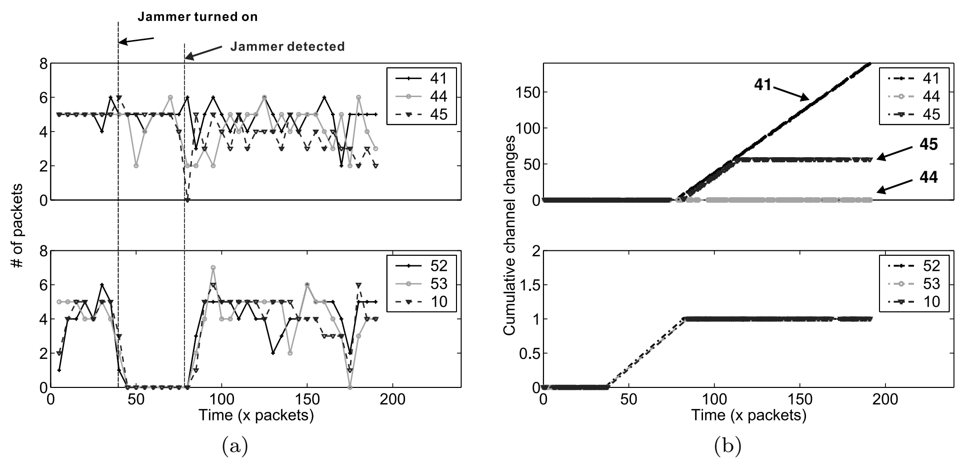


Fig. 10. Statistics for the synchronous spectral multiplexing strategy: (a) number of packets delivered to the sink in a 50-second window vs. time, (b) total number of channel switches vs. time.

boundary nodes then announced themselves on the new channel, and waited to be selected as parents by the nodes on channel 2. In this experiment, all three jammed nodes chose node 41 as their parent. Thus, node 41 started working on two channels, while node 45 went back to work on channel 1. Overall, node 41 had four child nodes, three on channel 2 (nodes 52, 53, and 10), and one on channel 1 (node 44). In this experiment, the slot duration was 6.1 seconds, so that the number of packets buffered per channel was roughly 3 to 6 packets.

Figure 10(a) presents the time series of the number of packets delivered to the sink from six nodes in a 5-packet window. These six nodes include the three jammed nodes (nodes 52, 53, and 10), one boundary node (node 41), one potential boundary node (node 45), and one child of the boundary node that worked on channel 1 (node 44). The plot shows that the jammed nodes resumed their normal packet delivery performance. None of the other nodes were much affected by the jammer. The interval during which the jammed nodes had disrupted services was roughly 48 packet intervals, which is similar to the recovery time in the coordinated channel surfing strategy. Again, during the total recovery latency of 50 packet intervals, the boundary nodes waited for around 39 packet intervals before switching to the new channel to search for their children. After the boundary nodes found their children on the new channel, it only took them 11 packet intervals to start working on dual channels.

We report the total number of channel switches for these six nodes in Figure 10(b). These numbers agree with the previous discussion regarding how different nodes responded to the emulated jamming in the experiment (e.g., node 41 continually switches channels). Interestingly, as noted earlier, node 45 started out as a dual-mode node and it initially switched channels frequently. However, after a short period of time, it was not selected as a dual-mode parent for any nodes on channel 2. It then returned to channel 1 and no longer switched channels.

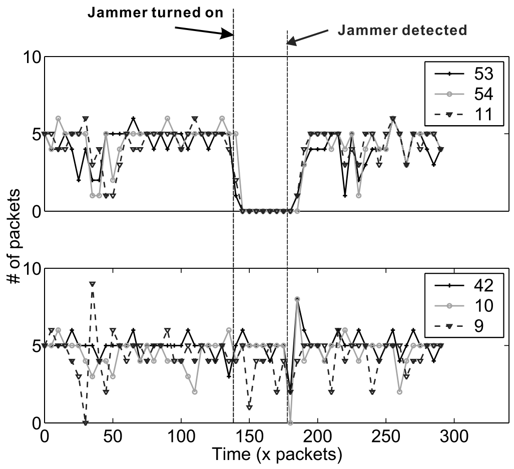


Fig. 11. Packet delivery time series for asynchronous spectral multiplexing.

6.3.2 *Asynchronous Spectral Multiplexing.* Our asynchronous experiment used a similar setup as the synchronous multiplexing experiment. The jammed region consisted of nodes 53, 54, and 11. After the jammed nodes evaded to channel 2, their former parents, nodes 42 and 10, also switched to channel 2, and announced their willingness to work on channel 2. Node 42 was chosen to be the parent by all three jammed nodes, and started flipping between both channels in a round-robin manner, while node 10 continued on channel 1. From then on, node 42 had three children (the three jammed nodes on channel 2).

Figure 11(a) presents the time series for the number of packets delivered to the sink from the previously mentioned nodes in a 5-packet window: the three jammed nodes (nodes 53, 54, and 11), one boundary node (node 42), and one potential boundary node (node 10). It is clear from this figure that the asynchronous multiplexing scheme can quickly recover the connections between the jammed nodes and the sink without affecting other nodes in the network. As in the case of synchronous multiplexing, this scheme also incurred roughly a 50-packet service disruption period for the jammed nodes. We also recorded the total number of channel switches for these six nodes and observed trends analogous to those reported for synchronous multiplexing.

6.4 Channel Surfing Discussion

As we noted, we designed the channel surfing strategies so that the protocols do not rely on extra beacons or messages. Therefore, there is no additional overhead needed if the network is in a normal state.

Further, though these results have shown that the four channel surfing strategies fare comparably, we emphasize that it is better to evaluate these strategies according to the network and interference scenarios for which they are most appropriate. The autonomous channel switching strategy relies on each individual node to detect whether its lost neighbors have been jammed and have escaped to the next channel. It does not introduce any additional protocol overhead, which can help keep the protocol design of a sensor network simple and clean. Its less proactive nature, however, leads to a longer delay in switching the entire network.

Broadcast-assist channel switching, which requires all network nodes to switch their channel, is most suitable for cases where a large region is jammed, and where the jamming occurs on a longer time scale (e.g. from a long-duration unintentional interference source).

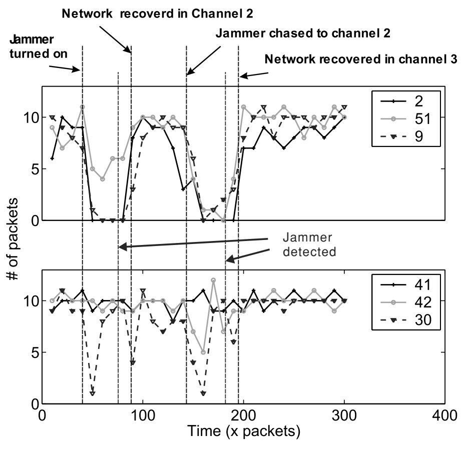
Spectrum multiplexing, however, is more effective for transient jamming where a few nodes are affected for a short duration. Further, we note that we have implemented a simple protocol stop criteria, whereby the boundary nodes stop multiplexing when they find that all their neighbors are working in one channel. Additionally, spectrum multiplexing is effective in scenarios where different regions of the network have interferers operating on different channels. This is in contrast to channel surfing strategies, which seek to establish a single operating frequency across the entire environment.

As a last comment, we note that we should determine whether to adopt synchronous or asynchronous spectral multiplexing based on the underlying traffic model. For instance, synchronous spectral multiplexing is more suitable for regular traffic patterns, while asynchronous spectral multiplexing can better cope with irregular (e.g., bursty) traffic.

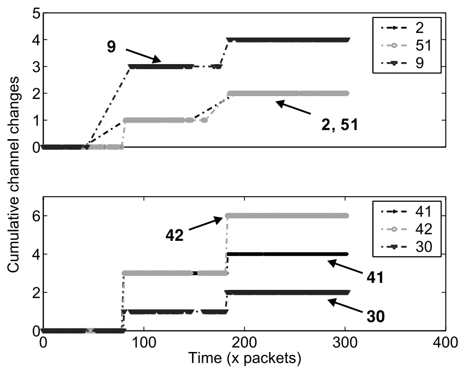
6.5 Channel Following Jammers

We were also interested in more challenging interference scenarios, such as the scenario in which the jammer follows the network as it channel surfs. We conducted experiments with this new jammer model, and found that all four of the proposed schemes could restore network connectivity in such cases. Due to space limits, we do not provide results for all four schemes here, but rather we show the experimental results for the broadcast-assist channel switch scheme.

The network setup was the same as in Figure 7(a). We started the network on channel 1, and then introduced the jammer approximately at the 40th packet into the experiment, as depicted in the time series in Figure 12(a). Shortly thereafter, the network adapted, with all nodes switching to the second channel after an overall latency of approximately 47 packet intervals. We allowed the network to run in the new channel for 50 packet intervals to find the new channel to which the network moved, and thus the jammer switched to channel 2 at the 140th packet interval. The network again adapted to the interference, switching to the third channel after a total latency of roughly 51 packet intervals. Examining the packet delivery time series for node 51 illustrates an interesting phenomenon regarding the effectiveness of a jammer: when the jammer was on channel 1, it was not entirely effective in disrupting the operation of node 51 while when the jammer was on channel 2, it was more effective at disrupting the communications from node 51. We believe this is due to the irregularity of the Mica2 radio. In addition to packet delivery traces, we recorded the number of times different nodes switched channels during the



(a)



(b)

Fig. 12. (a) Packet delivery time series for the broadcast-assist strategy when the jammer follows the network's channel surfing. (b) Time series illustrating the amount of times a node has changed its channel during the network's operation.

experiment, as illustrated in Figure 12(b). This curve points to the fact that the protocol does not require an excessive amount of channel switching, typically requiring either one or three channel change attempts prior to settling in on the new channel. We draw the reader's attention to the channel change trace for nodes 41 and node 42. During the first channel change, these nodes switched back to the original channel to broadcast the change channel command, thus requiring a total of three channel changes. However, after the jammer follows them to the new channel, both the radio dynamics and the underlying network topology have changed, and in this case only node 42 was involved in switching back to channel 2 to announce the change channel command.

7. RELATED WORK

Coping with jamming and interference is usually a topic that is addressed through conventional PHY-layer communication techniques. In these systems, spreading techniques are commonly used to provide resilience to interference [Proakis 2000; Schleher 1999]. Although such PHY-layer techniques can address the challenges of an RF interferer, they require more advanced transceivers and consequently have not found widespread deployment in commercial sensor networks. Instead, most sensor platforms use simpler radios with carrier sensing. Consequently, any form of radio interference that sufficiently elevates the energy in a channel can prevent a sensor from accessing the channel, effectively disrupting communications. We note, though, that the PHY in platforms like MicaZ or 802.11 employ a minimal amount of spreading in order to have multipath resistance. Additionally, we note that our frequency scheduling and synchronization methods are similar to those used in physical layer frequency hopping and TDMA, Rappaport and Grieco [1984] though our techniques operate on a much coarser time scale.

The issue of jamming detection for sensor networks was studied by Wood and Stankovic in Wood et al. [2003]. This study employed a measure of the channel's utility to detect jamming, and primarily focused on the issue of mapping the jammed region. The problem of jamming detection was further studied by Xu et al. [2005], where the authors presented several jamming models and explored the need for more advanced forms of detection algorithms to identify jamming. Additional jamming strategies were studied by Law et al. [2005], and the efficiency of these methods was quantified in terms of the amount of resources needed to conduct an attack. Further work on jamming has studied MAC-layer jamming attacks on reservation-based medium access control schemes [Rajeswaran and Negi 2005].

Countermeasures for coping with jammed regions in wireless networks, in studied in Noubir and Lin [2003]; Xu et al. [2004]; and Navda et al. [2007]. In Noubir and Lin [2003], the use of low density parity check (LDPC) codes is proposed to cope with jamming. Further, an anti-jamming technique is proposed for 802.11b that involves the use of Reed-Solomon codes. In Xu et al. [2004], two countermeasures are presented for coping with jamming. The first method, channel surfing, serves as the motivation for this article. The second method, spatial retreats, was studied in more detail in Ma et al. [2005] and involves mobile sensor nodes physically moving away from the interference source to reestablish connections. In Cagalj et al. [2007], wormhole-based anti-jamming techniques are studied analytically. The proposed approaches try to ensure the successful delivery of at least one alarm message for each event, instead of resuming continuous network connectivity. Our work builds on these efforts by providing system validation. In Navda et al. [2007], channel hopping technique was studied in 802.11 networks.

Although not precisely a jamming attack, one may exploit the MAC layer to achieve increased network resources [Bellardo and Savage 2003; Kyasanur and Vaidya 2003]. The issue of detecting non-MAC compliancy was recently studied in Raya et al. [2004]. This work showed that a greedy user can increase his share of bandwidth by corrupting the RTS and CTS of other users to prevent packet transmission, or may corrupt ACKs to cause the ACK contention window to increase, leading to larger backoff. They proposed DOMINO, a system for detection of such greedy behavior in the MAC layer of IEEE 802.11 public networks.

The use of multiple channels has been proposed as a means to enhance the throughput and performance of wireless mesh networks [Raniwala et al. 2004; So and Vaidya 2003; R.Garces and Aceves 2000], and of cellular networks. In this area, the radio devices are distinctly resource-rich compared to sensors, and typically have multiple-radio interfaces. The challenge underlying this field is assigning the channels across multiple nodes and time slots in order to avoid collisions and congestion. These works, however, are primarily intended to enhance performance in normal radio conditions, not to cope with external radio interference, as is described in this article. Generally, channel allocation for these scenarios is achieved with the aid of a centralized entity, or via a common control channel. In our work, we have not resorted to centralized entities or control channels in order to achieve jamming resistance.

8. CONCLUDING REMARKS

It is foreseeable that, as wireless sensor networks become increasingly deployed, they will be subjected to increased levels of radio interference. Such interference may be intentional, such as might arise from a jammer, or may be incidental, as may occur due to the presence of other wireless networks. In either case, most commercial wireless sensor networks will be susceptible to radio interference and, as a result, the ability of the sensor system to feed data to monitoring applications may be undermined. It is therefore critical to develop methods that can make sensor networks coexist with each other and even survive external interference. These defense mechanisms, however, must be distributed, easy to scale, and have low false positives. We have tackled this challenge in this article by presenting a family of channel surfing strategies that may be used to restore connectivity in the presence of radio interference. We presented two families of channel surfing strategies: the first, which we refer to as channel switching, consists of techniques whereby the entire sensor network changes its operating frequency; the second family, which we refer to as spectral multiplexing, aims to change the operating frequency in a neighborhood local to the interference, with boundary nodes acting as a radio-bridge across different channels. We implemented our strategies on a testbed of Mica2 motes, and have reported their performance for several interference scenarios. We have found that our broadcast-assist strategy, as well as our multiplexing schemes, can effectively repair the network with short latency.

Building large scale, interference-resistant sensor networks will require a long term effort by the entire community. This study is intended to serve only as a starting point towards this goal, and there are several interesting possibilities for future work. One important direction we are investigating involves coping with more advanced jamming/interference scenarios, such as multiple jammers conducting rapid channel chasing. Finally, we note that other defense strategies besides channel surfing warrant systems-level investigations.

REFERENCES

Bellardo, J. and Savage, S. 2003. 802.11 denial-of-service attacks: Real vulnerabilities and practical solutions. In *Proceedings of the USENIX Security Symposium*. 15–28.

Cagalj, M., Capkun, S., and Hubaux, J. 2007. Wormhole-based anti-jamming techniques in sensor networks. *To IEEE Trans. Mob. Comput.* To appear.

Ganeriwal, S., Kumar, R., and Srivastava, M. 2003. Timing-sync protocol for sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys'03)*. 138–149.

Garces, R. and Aceves, J. G. L. 2000. Collision avoidance and resolution multiple access for multi-channel wireless networks. In *Proceedings of IEEE INFOCOM*. 595–602.

Heinzelman, W., Kulik, J., and Balakrishnan, H. 1999. Adaptive protocols for information dissemination in wireless sensor networks. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*. 174–185.

Intanagonwiwat, C., Govindan, R., and Estrin, D. 2000. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proceedings of the 6th Annual ACM/IEEE International Conference on Mobile Computing and Networks (MobiCOM)*.

Juang, P., Oki, H., Wang, Y., Martonosi, M., Peh, L., and Rubenstein, D. 2002. Energy-efficient computing for wildlife tracking: design and tradeoffs and early experiences with zebranet. In

Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems. 96–107.

Kyasanur, P. and Vaidya, N. 2003. Detection and handling of MAC layer misbehavior in wireless networks. In *Proceedings of the IEEE International Conference on Dependable Systems and Networks*. 173–182.

Law, Y., Hartel, P., den Hartog, J., and Havinga, P. 2005. Link-layer jamming attacks on S-MAC. In *Proceedings of the 2nd European Workshop on Wireless Sensor Networks (EWSN'05)*. 217–225.

Ma, K., Zhang, Y., and Trappe, W. 2005. Mobile network management and robust spatial retreats via network dynamics. In *Proceedings of the The 1st International Workshop on Resource Provisioning and Management in Sensor Networks (RPMSN05)*.

Madden, S., Franklin, M., Hellerstein, J., and Hong, W. 2002. TAG: a Tiny aggregation service for ad-hoc sensor networks. In *Proceedings of the Usenix Symposium on Operating Systems Design and Implementation*.

Navda, V., Bohra, A., Ganguly, S., and Rubenstein. 2007. Using channel hopping to Increase 802.11 resilience to jamming attacks. In *IEEE Infocom Minisymposium*. Anchorage, AK.

Noubir, G. and Lin, G. 2003. Low-power DoS attacks in data wireless lans and countermeasures. *SIGMOBILE Mob. Comput. Comm. Rev. 7,* 3, 29–30.

Perrig, A., Szewczyk, R., Tygar, D., Wen, V., and Culler, D. 2002. SPINS: security protocols for sensor networks. *Wirel. Netw. 8,* 5, 521–534.

Polastre, J., Hill, J., and Culler, D. 2004. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys'04)*. 95–107.

Proakis, J. G. 2000. *Digital Communications*, 4th Ed. McGraw-Hill.

Rajeswaran, A. and Negi, R. 2005. DoS analysis of reservation based MAC protocols. In *Proceedings of the IEEE International Conference on Communications*.

Raniwala, A., Gopalan, K., and Chiueh, T. 2004. Centralized channel assignment and routing algorithms for multi-channel wireless mesh networks. *ACM Mob. Comput. Comm. Rev. 8,* 2, 50–65.

Rappaport, S. S. and Grieco, D. M. 1984. Spread-spectrum signal acquisition—Methods and technology. *IEEE Comm. Mag. 22*, 6–21.

Raya, M., Hubaux, J., and Aad, I. 2004. Domino: a system to detect greedy behavior in IEEE 802.11 hotspots. In *Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services (MobiSYS'04)*. ACM Press, 84–97.

Schleher, C. 1999. *Electronic Warfare in the Information Age*. MArtech House.

So, J. and Vaidya, N. 2003. Multi-channel MAC for ad hoc network: handling multi-channel hidden terminals using a single transceiver. In *Proceedings of ACM MobiHoc*. 222–233.

TinyOS. Tinyos homepage. http://webs.cs.berkeley.edu/tos/.

Wan, C.-Y., Eisenman, S., and Campbell, A. 2003. Coda: congestion detection and avoidance in sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys'03)*. 266–279.

Woo, A., Tong, T., and Culler, D. 2003. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys'03)*. 14–27.

Wood, A. and Stankovic, J. 2002. Denial of service in sensor networks. *IEEE Comput. 35,* 10, 54–62.

Wood, A., Stankovic, J., and Son, S. 2003. JAM: A jammed-area mapping service for sensor networks. In *Proceedings of the 24th IEEE Real-Time Systems Symposium*. 286–297.

Xu, W., Trappe, W., Zhang, Y., and Wood, T. 2005. The feasibility of launching and detecting jamming attacks in wireless networks. In *Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'05)*. 46–57.

Xu, W., Wood, T., Trappe, W., and Zhang, Y. 2004. Channel surfing and spatial retreats: defenses against wireless denial of service. In *Proceedings of the ACM Workshop on Wireless Security*. 80–89.

Ye, W., Heidemann, J., and Estrin, D. 2002. An energy-efficient MAC protocol for wireless sensor networks. In *Proceedings of the IEEE INFOCOM*. Vol. 3. 1567–1576.

Zhao, J. and Govindan, R. 2003. Understanding packet delivery performance in dense wireless sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys'03)*. 1–13.

Received April 2007; revised October 2007; accepted October 2007