Evaluation of ad hoc routing over a channel switching MAC protocol

Ethan Phelps-Goodman and Lillian Kittredge

December 10, 2004

Abstract

Several recent works on wireless ad hoc networks have looked at increasing bandwidth by utilizing multiple radio channels. One such scheme is SSCH, which uses a distributed coordination mechanism to dynamically distribute overlapping flows on separate channels. While the simulation data from the original SSCH paper show substantial performance gains through channel hopping, they also suggest an adverse effect on ad hoc routing when running over SSCH. In this paper we investigate further the interaction between routing and SSCH. We argue that the potential problems are significant, and build a simulation to quantify these effects. Unfortunately, due to problems with the simulation, we were for the most part unable to collect meaningful data on our question.

1 Introduction

An ad hoc wireless network is a set of wireless mobile nodes forming a network without the aid of a centralized control structure. As their name implies, ad hoc networks are unstructured and highly dynamic. They networks have been an active area of research for over a decade. They were originally investigated for military applications because of their fault tolerance and infrastructureless deployment, but research into civilian applications has overtaken military applications [4]. It should be mentioned that the IEEE 802.11 specification defines an ad hoc network as an intfrastructureless network in which the nodes are within range of one another [5]. We use the more common definition that includes larger networks where traffic must be routed along multiple hops.

Improving the capacity of wireless ad hoc networks is an open area of research. One recent approach to this problem is to exploit the ability of wireless technologies to communicate on multiple orthogonal channels. 802.11, for instance, supports 13 independent channels. The use of multiple channels was originally intended for infrastructured networks where central coordination of channels is possible. Recent work by Bahl et al. extends the benefits of multiple channels to ad hoc networks [1]. Their scheme, Slotted Seed Channel Hopping (SSCH), uses a randomized distributed algorithm for coordinating channel hopping among nodes. The protocol attempts to schedule nodes so that those with traffic for each other will rendezvous on the same channel, while avoiding unrelated traffic on other channels.

While SSCH can give dramatic increases in throughput, there is reason to believe that channel hopping interacts poorly with ad hoc routing protocols. These protocols are fundamental to the operation of an ad hoc network, so any negative impact on the routing protocols will have a significant effect on network performance. This problem of the interaction between channel hopping and routing is touched on in [1], where they show a moderate degradation in the performance of the Dynamic Source Routing (DSR) protocol [2]. In this paper we give a further investigation into the interaction of these protocols, and use simulation to quantify the performance loss.

The rest of the paper is organized as follows. Section 2 describes SSCH in more detail. Section 3 de-

scribes DSR and discusses the possible negative interactions with SSCH. Section 4 presents our simulation and preliminary results. In section 6 we discuss issues brought up by our simulation and future work. Section 7 concludes the paper.

2 SSCH

SSCH is a link-layer protocol for distributed coordination of channel hopping. The objective is to arrange for multiple flows within radio range to be sent over separate channels. This allows bandwidth to scale linearly with the number of flows, up to the point when all 13 channels are saturated. While there have been other channel hopping protocols for 802.11, SSCH is, to our knowledge, unique in that there is no dedicated channel for control information, and so none of the congestion and delay issues associated with this. SSCH is also advantageous in that it runs on top of IEEE 802.11 compliant hardware, and requires no changes to the 802.11 protocol. The rest of this section will describe the scheduling mechanisms that control channel hopping, and the coordination mechanisms that determine schedule changes.

2.1 Time multiplexing

SSCH divides time into a repeating series of four 10ms slots. Each node maintains a schedule of channels encoded in four sub-schedules – one for each slot. Nodes are assumed to have synchronized clocks, so that slot boundaries roughly align. (Bahl et al test their protocol for robustness to clock skew, and find that it fares well.) The division into four separately scheduled slots allows each node to take on multiple roles, such as forwarding for one flow in the first slot, acting as a data source in the secod slot, and so on. The four slots are repeatedly cycled through in order, so that two nodes synchronized on a slot don't have to wait more than 30ms before their schedules align again. We say that two nodes are synchronized on a slot if they share the same schedule for that slot, and so are visiting the same channels in the same order, allowing them to share data during every instance of that slot (1/4 of the time).

The schedule for each slot consists of an order for visiting the 13 channels. The total time for each of the four slots to iterate through the 13 channels is known as a cycle. A cycle also includes one additional slot at the end known as the *parity slot*, which is needed to prevent logical partition. The schedules are constructed in such a way that unsynchronized nodes are guaranteed to overlap on some channel at least once per cycle. This ensures that nodes learn about one another's presence, and avoids partition of the network. Once per slot each node broadcasts its complete schedule. Nodes use the data about their neighbors' schedules to make decisions about how to change thier own schedules.

Packets are kept in per-neighbor FIFO queues. These queues are prioritized according to perceived reachability. Reachability is determined by cached schedules for that node, and whether past transmissions based on that schedule have been successful. A node will only attempt to send packets to neighbors it perceives are reachable (unless none are perceived to be reachable, in which case it sends all the packets it has).

Channel hopping introduces a problem for applications that use broadcast packets. Unlike in single-channel networks, a broadcast packet will not reach all of the neighbors in range, since they are likely to be on spread across different channels. SSCH deals with this by retransmitting broadcast packets 6 times in successive slots, making it more likely, but not guaranteed, that neighbors will hear the broadcast. In section 3.1, we discuss how this issue affect the interaction of SSCH with DSR.

2.2 Changing schedules

Nodes change their schedules with the goals of converging on channels with nodes they wish to communicate with, and moving off congested channels. In a well synchronized network, nodes keep their schedules synchronized with nodes that they have traffic for, and only occasionally overlap with other nodes. During each slot, a node may change its schedule for the next slot based on four fairly simple rules. The first and most straightforward rule is that nodes alter their schedules to synchronize them with the sched-

ules of nodes they want to send to. To take into account the nodes' desire to receive as well as send packets, the node observes how many packets it receives in each slot. This way, for each sub-schedule, it knows how many packets it received in the last instance of this slot (the last time it used this sub-schedule). If the node received more than 10 packets the last time this sub-schedule was used, then that sub-schedule is labeled receiving, and is not allowed to change.

Nodes also explicitly avoid congestion. They accomplish this by observing which other nodes have synchronized their schedules in this slot. If the number of neighbors sharing a schedule is more than twice as many as the number of neighbors this node communicated with previously in this slot,the channel is considered congested. The node de-synchronizes from the others on the congested channel, simply by creating a new random schedule to replace the current one. This is necessary to prevent all nodes converging to one schedule. Finally, in order to avoid partition, nodes are only allowed to update the schedule for the first slot during the parity slot.

3 The DSR Protocol

We use DSR as a representative protocol for testing channel switching and ad hoc routing. There are many other protocols that could have been used (see GPSR [10] and AODV [11] for example), and looking at other routing protocols may be interesting future work. For this study we chose DSR primarily so that our results would be directly comparable to those in [1]. DSR is one of the oldest and most well-studied ad hoc routing protocols, and is moving towards adoption as a standard [3]. This makes DSR a good choice for evaluation because it is one of the more relevant protocols, and there is plentiful existing performance data. From a practical perspective, it has the advantage that there are several DSR simulation modules are freely available. In this section we give a brief introduction to how DSR works, with emphasis on the parts that we believe interact poorly with SSCH.

DSR is a reactive protocol, meaning that nodes waits until a route is needed before attempting to determine the route. This is in contrast to active protocols, such as link state and distance vector in wired networks, which send background traffic to keep their routing tables constantly up to date. In DSR, when a sender needs a new route, it floods a route request across the entire network. Flooding has obvious scaling problems, and DSR is only intended to be used on networks with small diameter (less than 10 hops according to [2].) Extensive caching is used to reduce the amount of flooding needed.

There are three main components to DSR: route discovery, forwarding, and route maintenance. Each of these is potentially affected by SSCH. Route discovery is accomplished by flooding, as discussed above. Each time the request is re-broadcast, additional path information is appended, so that by the time the request reaches the destination, a list of intermediate hops used to get to the destination has been built up. This path information is then sent back to the requester along the reverse route. Note that the efficacy of flooding hinges upon request broadcasts reaching all neighbors.

Forwarding is accomplished by including the full route of every packet in the packet header. Each intermediate node simply forwards to the next address in the path. The forwarding node is responsible for retransmission to its next hop in the event of a failure. If the next hop is not reached by a certain timeout, the forwarding node returns a route error message back along the path to the original sender.

Route maintenance consists of the caching mechanisms used to keep the routes up to date. Essentially, every piece of routing information heard is recorded. This includes the paths in routing request broadcasts and the paths in data packets begin forwarded. In order to gather as much information as possible, DSR is supposed to be run with the network interface in promiscuous mode. (Promiscuous mode allows the transport layer to see all packets on the medium, not just those destined for that machine.) If a node overhears a packet being sent along a circuitous route, it sends a route reply back to the original sender with the shorter path. Also, if a sender learns of a broken link, it floods the information along with its next route request so that other nodes can remove affected routes from their tables. In this way, long routes are

eventually shortened and broken routes are repaired.

3.1 Interaction with SSCH

The most obvious problem when running DSR over SSCH is that the efficacy of the flooding mechanism depends on broadcast packets propagating quickly to the entire network. Since at any time a node will only share a channel with a fraction of its neighbors, the normal operation of broadcast may be significantly impaired. As previously mentioned, SSCH attempts to remedy this by retransmitting broadcast packets over several slots. This is partially effective, but Bahl et al. still find that SSCH increases route discovery time and route length.

There are also other potential problems not considered in[1]. One is that DSR relies on promiscuous mode operation to optimize routes and maintain cached information. Though an SSCH node promiscuously listens to all of the traffic on its current channel, it is also purposefully trying to avoid encountering traffic not destined for it, by avoiding congested channels. This could lead to less path optimization over time, since news of better routes is not heard. Having less cached information from promiscuous receiving could also lead to more routes having to be discovered by flooding. Finally, forwarding may be less efficient, as a forwarding node may have to wait until its schedule is aligned with the next hop, potentially increasing transmission delay by several tens of milliseconds per hop.

To summarize, when using DSR over SSCH we expect to see:

- longer route discovery times
- longer initial path lengths
- more delay from forwarding
- less path optimization over time
- more route requests over time

4 Evaluation

We build off of *ns-2* [6], a widely used open-source network simulator. *ns-2* is a packet level simulator with built in functionality for all levels of the network stack, from radio propagation models to FTP transfers. Although originally designed for wired networks, *ns-2* was extended by the CMU Monarch Project to handle wireless ad hoc networks [2]. The included DSR module we use is the same one used in [2] to test DSR against other ad hoc routing protocols. Although SSCH itself is only moderately complicated, integration with *ns-2* turned out to be substantially more difficult than we initially expected, requiring changes to existing code, and a detailed a understanding of the low level workings of a significant portion of the code base.

Logically, SSCH sits at the link layer, above the network interface. While Bahl et al. refer to their SSCH implementation as a MAC layer module, we chose to change the existing 802.11 module as little as possible, only inserting hooks to allow it to directly alert our SSCH module to RTS transmission failures, and to deal with the SSCH schedule-advertisement packets. The bulk of our SSCH implementation is a module between the `LL` (Link Layer) object of *ns-2* and the MAC layer. The very existence of such a separate module indicates the odd design decisions in the structure of the *ns-2* stack. At the higher levels of the network stack, the tracing mechanism for data collection were modified to make data collection manageable. Also, although *ns-2* supports multiple channels, there was no existing mechanism to allow nodes to switch channels, so changes were required in the node implementation.

Other differences between our SSCH implementation and Bahl's arise from the differences between the simulation environments we use. They use the proprietary simulator QualNet [7], which simulates the PHY layer more cleanly and realistically, including physical-layer buffers of packets and a delay for channel-switching [1]. *ns-2* lacks these elements. We suspect that the structure of QualNet is in general more coherent and well-designed than that of *ns-2*, which has evolved over time and at the hands of multiple developers with differing goals.

One important caveat about our results with *ns-2* is that even before making any SSCH-related changes, we were unable to increase the bandwidth of 802.11 beyond 3.2Mb/s. This is about an order of

magnitude short of what 54Mbs WiFi is capable of. We were not able to explain this bandwidth limitation other than it seems to be coming from a variety of delays throughout the network stack rather than a single limiting bottleneck. This is a problem for SSCH, because at this rate, a 10ms slot (which Bahl et al. assumed to fit about 35 full length packets) only fits 3 packets. To get around this, we increased the slot length to 100ms. Because of these workarounds, the bandwidth results given here should be used only as a relative comparison between results in this paper and not with other work or real-world data.

4.1 Experiments

Our first test is a simple sanity-check, confirming that our SSCH implementation still shows the basic advantage over 802.11 that Bahl et a. found. A group of stationary nodes are placed all within range of each other. (The radio range in all our simulations is set to approximately 250m.) The nodes are divided into sender/receiver pairs. All communication is single hop, and each node is either sending one flow or receiving one flow. The flows are UDP with the maximum rate set to saturate a single 802.11 channel.

Figure 4.1 shows the total data throughput summed over all flows. With 802.11, a single flow saturates the bandwidth, so increasing the number of flows does not increase the total bandwidth. With SSCH, on the other hand, we see a nearly linear increase in bandwidth, as additional flows simply move to uncongested channel schedules. The bandwidth peaks around 13 flows as expected, and then starts to degrade as contention and overhead from control traffic lower the utilized bandwidth.

The second graph, Figure 4.1, shows the average delay in different-sized networks. Though SSCH starts off with a higher delay due to its operation overhead, the delay in 802.11 increases linearly with number of flows, as each flow has to contend for the channel. In SSCH, communicating nodes mostly have their own channels, so delay remains constant until the number of flows exceeds the number of channels, at which point they once again contend for the medium.

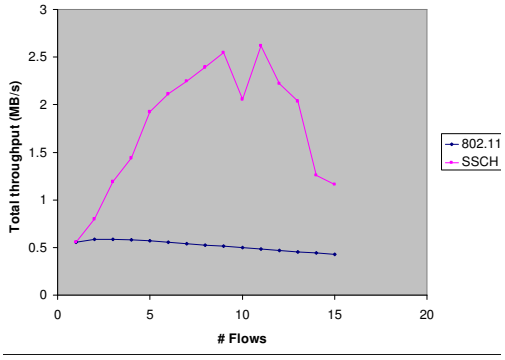

Figure 1: Throughput rate of disjoint flows on SSCH versus 802.11

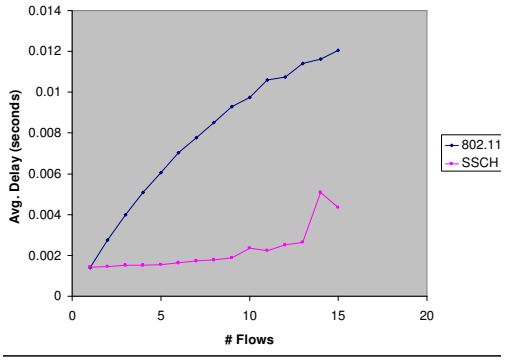

Figure 2: Delay of disjoint flows on SSCH versus 802.11

4.2 SSCH and DSR over a multi-hop network

The experiments presented in this section are meant to address the main questions of this paper: How does SSCH impact the time to find routes, and the quality of routes found, the maintenance of routes over time, and the delay and total throughput of the network? For reasons that will be examined in the next section, we do not have reliable data to address these questions. This section will describe the experiments we ran and the (faulty) results we obtained.

The simulations in this section evaluate SSCH when operating on a multi-hop ad hoc network. The topology used is a simple evenly spaced square grid of nodes. Each node is 150m from each of its neighbors on the grid. This puts each node in range of its adjacent and diagonal neighbors, so that each node has eight neighbors total. In all tests, 10 maximum rate UDP flows are established from nodes on one side of the grid to nodes on the opposite side. In this way, the dimension of the grid directly relates to the minimum number of hops in any path. The flows start within the first half second and the simulation runs for 30 seconds.

We experimented with a variety of network topologies and traffic patterns. In particular, we tried a

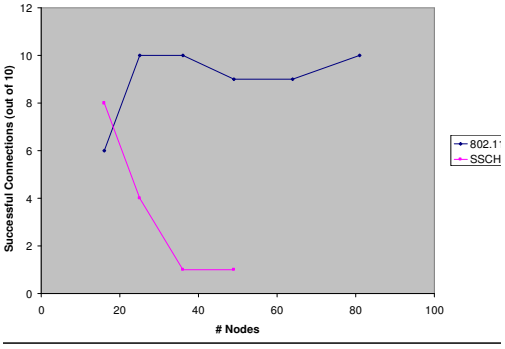

Figure 3: The number of flows sucessfully established

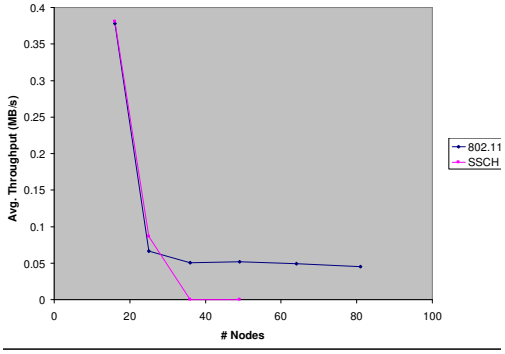

Figure 4: Throughput, averaged over 25 seconds over all active flows

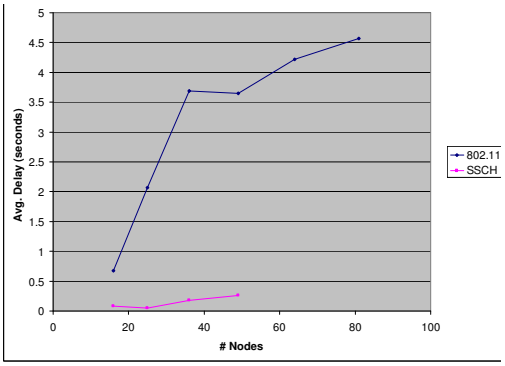

Figure 5: Average delay

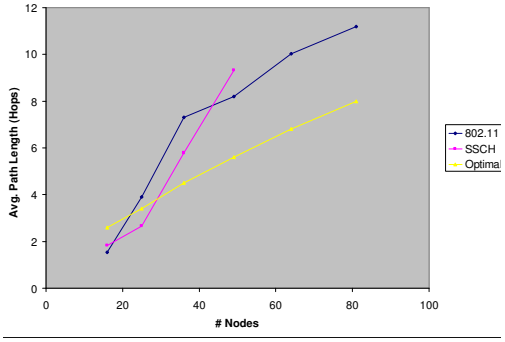

Figure 6: Average number of hops between source and sink

number of randomized node layouts, but it was difficult to reliably create large random networks that were not partitioned. The regular grid setup was chosen for our final tests because, though artificial, it gives DSR plenty of choice in paths, but keeps the guarantee about minimum path lengths. We also measured networks with moving nodes, but did not get results consistent enough to report.

Figure 4.1 show the number of flows that were successfully established. Except for an anomaly in the 16-node network, in the experiments without SSCH, the network established connections most of the time, only failing to find routes for two flows, each in one of the larger networks. With SSCH it performs much worse: the 64- and 81-node networks weren't able to establish even a single connection. The need even to include this graph is a surprise–even with SSCH's potential performance problems, we were not expecting any source to go the full 30 seconds without finding a route to its destination. This common failure to establish connections means that the sample size of the following data is very small.

Figure 4.1 shows the throughput, averaged over the last 25 seconds of the simulation and over all active flows. Similarly, Figure 4.1 shows the average delay and Figure 4.1 shows the average number of hops between source and sink. We do not believe the data to be reliable, and so do not discuss the results further. We present the data mainly to show that we were capable of collecting interesting data, had the underlying simulation been working. We also collected data on how path length varied over time as DSR attempted to optimize its routes, but did not feel it was significant enough to include here.

4.3 Discussion of Problems with DSR and *ns-2*

While the dismal performance of SSCH in our tests could conceivably be due to faulty implementation, we were surprised to see surprisingly bad performance in DSR running on 802.11, essentially unchanged from the release version. It has trouble discovering routes, sustains a throughput of about 1/10th the

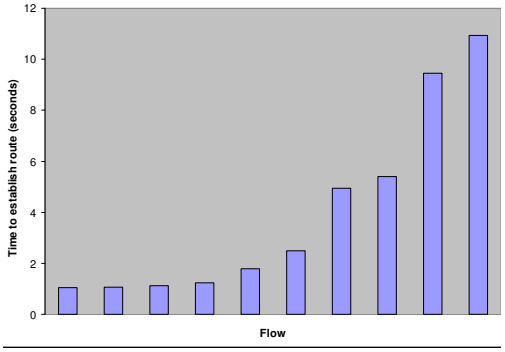

Figure 7: Amount of time to find a route in DSR over 802.11

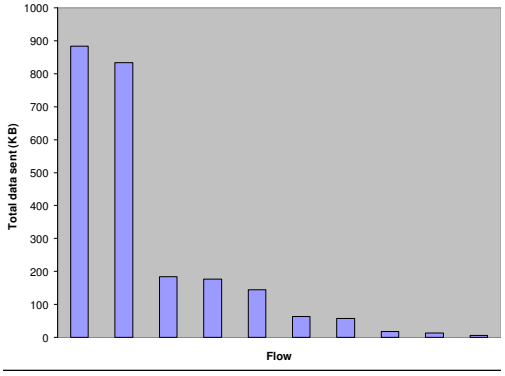

Figure 8: The same flows from the prvious figure, sorted by total amount of data sent by the flow in the entire run

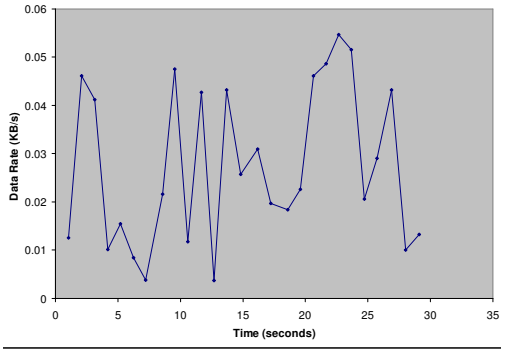

Figure 9: Rate of transmission of one particular flow over time

available bandwidth, and has a completely unreasonable delay of 4 seconds over 8 hops. In this section, we set aside SSCH and discuss why we were unable to get reasonable data with *ns-2*.

To investigate the source of these problems, we looked closer at the data from one typical run of the simulation on a network of 25 nodes. Figure 4.2 shows the amount of time between issuing a route request and sending the first data packet over the discovered route. Each vertical bar is a flow; the flows have been sorted by increasing time. Five of the flows established a route in under 2 seconds, which sounds high, but is within reasonable limits. The other half of the flows took longer, with the last taking nearly 11 seconds to find a route in a network of 25 nodes! This could explain why in some runs, not all the flows were established: perhaps even 30 seconds still isn't long enough.

Figure 4.2 shows the same 10 flows, this time sorted by total amount of data sent by the flow in the entire run. Two of the flows achieve a poor but plausible total of nearly a megabyte of data in 30 seconds. The rest barely send data, with the worst flows managing to transmit only a handful of packets successfully. It is probably not coincidental that the two high rate flows were the two pairs of nodes that were only 2 hops apart, while the rest of the nodes were 4 hops apart. Looking closer at the first of the two high rate flows, Figure 4.2 shows the rate of transmission of one flow over time. The source node is sending UDP packets at a constant rate, but the receiving end is only getting packets in small bursts. The transmission rate varies by an order of magnitude from second to second.

Taken together, these observations lead us to conclude that the codebase that we built off of is not functioning well enough to be considered usable. However, we find it highly unlikely that the basic *ns-2* or the DSR protocol itself are at fault. A few of the possible explanations for the problems:

- the current release of *ns-2* may be buggy. *ns-2* is a large open source project, and is bound to contain bugs. In fact, we identified one in `channel.cc`, and the current release's "stable" release doesn't even compile without manually fixing some lines.
- we may have inadvertently made changes which crippled DSR.
- something about our simulation setup may be incorrect and causing poor performance. *ns-2* is configurable with literally hundreds of simulation parameters, and the documentation explicitly makes no promises that the default settings are realistic.

Without more investigation we can't speculate on which of these, if any, is the ultimate cause.

5 Related Work

Most of the previous work on using frequency diversity for ad hoc capacity improvement has been with hardware using multiple radios. Dynamic Channel Assignment (DCA) [8] and Multi-radio Unification Protocol (MUP) [9] are two of these. In DCA, one of the radios is used on a fixed control channel. A weakness of this is in the dedicated control channel - not only is that channel lost for purposes of data transmission, it is also susceptible to interference. SSCH's distribution of control traffic across channels makes it robust to such interference. MUP uses both radios for both data and control traffic, but does not switch channels, and so can only utilize as many channels as the device has radios. Furthermore the fact that both of these require a whole extra radio is something of a drain, as the cost of an extra radio is generally considered high, since it currently requires a separate NIC.

There is little other work which addresses multichannel operation in ad-hoc wireless networks for devices with one radio. The only work other than SSCH of which we are aware is the Multichannel MAC (MMAC) [12] . It uses a common control channel to coordinate channel rendezvous between nodes. Its coordination system is based on the 802.11 power save model. Unfortunately, this requires good clock synchronization, which is difficult in ad hoc networks in general, and particularly so in situations where broadcast is nontrivial. Also, they use a single control channel, which becomes a performance bottleneck.

6 Future work

The foremost piece of future work is to complete the evaluation that was attempted in this paper. We have demonstrated a cause for concern, but without solid data it is impossible to know how serious the issue is.

A major improvement for the evaluation of SSCH and its interaction with routing protocols would be a deployment on a physical testbed. Especially given the inaccuracies of simulators like *ns-2* when simulating channel hopping (discussed in 4), we feel that a physical deployment would provide much more meaningful data. Furthermore, this would allow a comparison with the testbed evaluation of DSR [2]. An issue which is likely to come up in any physical simulation of SSCH is that the 13 "orthogonal" channels provided by IEEE 802.11 will interfere with each other when the radios are at close range [13]. SSCH does not currently make provisions to deal with this co-channel interference.

Because SSCH is one of the most promising protocols of its kind, we would be interested to evaluate it in the context of other routing protocols. However, in addition to examining its performance with other routing protocols, we feel that the greatest gain to be had would be in rethinking current protocols with SSCH in mind. In the longer term, we believe that designing channel hopping and routing protocols to share information and work together has the potential to provide better performance than protocols designed separately. A channel hopping protocol with knowledge of routing and traffic patterns may be able to make more informed decisions about schedule changes. Similarly, a routing protocol that is aware of the channel switching can get a more accurate view of the its neighbors and the capacity of its flows.

7 Conclusion

In this paper we have explored the interaction between channel hopping with the SSCH protocol and routing with the DSR protocol. We argue that the design of SSCH leads to unavoidable problems in routing. We describe our implementation of SSCH in a simulation environment, and discuss our results when running DSR over our SSCH implementation. Unfortunately, for reasons about which we can only speculate, the underlying simulation was unable to give realistic data. Thus we cannot come to any conclusion, but urge further testing of DSR over SSCH.

References

[1] P. Bahl, R. Chandra, and J. Dunagan. SSCH: Slotted Seeded Channel Hopping for Capacity Improvement in IE EE 802.11 Ad-Hoc Wireless Networks. In MobiCom '04.

[2] D. Johnson, D Maltz, and J. Broch. DSR: The Dynamic Source Routing Protocol for Mulithop Wireless Ad Hoc Ne tworks. In Ad Hoc Networking, Addison-Wesley, 2001

[3] D. Johnson, D. Maltz, and Y. Hu. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR). IETF MANET Working Group INTERNET-DRAFT, 2004.

[4] C. Elliot and B. Heile. Self-Organizing, Self-Healing Wireless Networks. IEEE 2000.

[5] IEEE 802.11b/D3.0, Wireless LAN Medium Access Control(MAC) and Physical (PHY) Layer Specification: High Speed Physical Layer Extensions in the 2.4 GHz Band, 1999.

[6] S. McCanne and S. Floyd. UCM/LBNL/VINT Network Simulator - ns (version 2). (URL: `http://www.isi.edu/nsnam/ns/`)

[7] QualNet, `http://www.qualnet.com/.`

[8] S.-L. Wu, C.-Y. Lin, Y.-C. Tseng, and J.-P. Sheu. A New Multi-Channel MAC Protocol with On-Demand Channel Assignment for Mobile Ad Hoc Networks. In International Symposium on Parallel Architectures, Algorithms and Networks (I-SPAN) 2000.

[9] A. Adya, P. Bahl, J. Padhye, A.Wolman, and L. Zhou. A Multi-Radio Unification Protocol for IEEE 802.11 Wireless Networks. In IEEE International Conference on Broadband Networks (Broadnets) 2004.

[10] B. Karp and H. Kung. GPSR: Greedy Perimeter Statelss Routing for Wireless Networks. In ACM MobiCom, 2000.

[11] C. E. Perkins. Ad-hoc on-demand distance vector routing, in MILCOM '97 panel on Ad Hoc Networks, Nov. 1997.

[12] J. So and N. H. Vaidya. Multi-Channel MAC for Ad Hoc Networks: Handling Multi-Channel Hidden Terminals Using a Single Transceiver. In ACM MobiHoc 2004.

[13] Texas Instruments. The Effects of Adjacent Channel Rejection and Adjacent Channel Interference on 802.11 WLAN Performance. http://whitepapers.zdnet.co.uk/0,39025945,60070499p-39000680q,00.htm.