Implementation of Dynamic Channel Switching on IEEE 802.11-Based Wireless Mesh Networks

Gang Wu Sathyanarayana Singh Tzi-cker Chiueh

Stony Brook University

ABSTRACT

Interference makes it difficult for a wireless network to provide robust performance and could sometimes lead to transient failures. Dynamic channel switching (DCS) allows a wireless network interface (NIC) to operate in different frequency channels during different time periods without disrupting network connections that traverse the NIC. DCS enables a wireless mesh network to avoid frequency channels with serious interference in certain parts of its coverage area by switching to more idle channels. Although DCS provides additional radio agility, it significantly increases the complexity of the wireless mesh network's routing protocol. This paper describes the design, implementation and evaluation of a wireless mesh network system called *Carlsbad*, which supports both DCS and load-balancing/fault-tolerant routing, and successfully runs on low-cost commodity IEEE 802.11-based access points. Performance experiments on the first *Carlsbad* prototype show that despite the additional overhead it introduces, DCS can indeed improve the overall throughput of an IEEE 802.11-based wireless mesh network, sometimes by a factor of more than 2, for both TCP and UDP connections.

1. INTRODUCTION

Communication quality of wireless network links is heavily dependent on various external factors such as physical object geometry and radio signal interference. As a result, real-world radio channel quality tends to exhibit small-scale and large-scale temporal variation that is in general difficult to model and predict. These temporal quality fluctuations can be considered as transient failures because they oftentimes lead to serious performance degradation. How to architect wireless networks so that they can deliver robust performance is thus an important issue, especially for enterprise-grade wireless network deployment. This paper describes the design and implementation of an adaptive channel assignment scheme called *Dynamic Channel Switching*(DCS) on an IEEE 802.11-based wireless mesh network, and shows that it indeed provides more robust performance in the presence of radio signal interferences, the major cause of wireless link quality fluctuation.

DCS virtualizes a wireless network interface (NIC) by allowing it to operate in different frequency channels during different time periods without disrupting network connections that traverse the NIC. DCS enables a wireless NIC to physically interact with multiple wireless NICs each of which operates in a different frequency channel. This additional flexibility provided by DCS offers two distinct advantages. First, it greatly increases a WMN node's effective connectivity beyond the number of physical NICs it has and thus potentially provides more room for optimization to the routing protocol. Second and more importantly, DCS enables each pair of interacting wireless NICs to communicate over the least loaded channel, thus making the best use of the allocated frequency spectrum.

There are several non-trivial challenges in implementing DCS on a standard-based wireless network. First of all, the performance overhead of switching an wireless NIC between channels must be sufficiently small to be practical. Second, switching wireless NICs between channels requires substantial changes to the routing and transport protocols running on the wireless networks. Finally, a sophisticated control mechanism, comparable in complexity to those used in multi-channel wireless mesh networks [8], is needed for optimal channel assignment and channel time allocation. Because of space constraints, this paper focuses only on the first two challenges in the context of WiFi-based wireless mesh networks.

A wireless mesh network (WMN) consists of a set of nodes each of which is equipped with one or multiple wireless NICs and plays the roles of access point or intermediate router or both. Typically a WMN is connected to the wired internet through one or multiple gateways, and most of the packets traversing a WMN go through one of these gateways. Because WMN nodes are most stationary, the routing protocol used in WMNs is more similar to those used in wired networks than to those used in mobile ad hoc networks (MANET). More specifically, because the maximal number of hops in real-world WMNs is bounded to a small number (approximately 5), many WMN routing protocols actually borrow heavily in design from the routing protocols used in wired local-area networks, which are pre-dominantly based on the concept of *spanning tree*, for example, IEEE 802.1D [1]. The WiFi-based WMN system described in this paper, *Carlsbad*, is no exception.

When a WMN has multiple gateways, *Carlsbad* constructs multiple spanning trees over the WMN, each rooted at a distinct gateway, and ensures that each node in the WMN be able to interact with the wired internet through one and only one gateway. To balance the loads of the gateways, *Carlsbad* associates WMN nodes with spanning trees in such a way that the entire traffic load is approximately divided among these gateways. To quickly recover from a link or node failure (including gateway failure), *Carlsbad* assigns to each WMN node a list of back-up parent nodes in addition to

WICON'08, Nov. 17-19, 2008, Maui, Hawaii, USA.

the primary parent node. In general, a WMN node's backup parent nodes may or may not be in the same spanning tree as the WMN node, and they offer an alternative path for the WMN node to reach the wired internet when its primary path fails.

Unfortunately, because WMN routing algorithms that support both load balancing and fault tolerance rely heavily on broadcasting and DCS prevents a WMN node from reaching all its immediate neighbors using a single physical broadcast frame, integrating DCS with these WMN routing algorithms poses a major technical challenge. The major focus of this paper is the design and implementation of the first known single-radio IEEE 802.11-based WMN system that supports both DCS and load-balancing & fault-tolerant routing. Despite the additional performance overhead due to periodic channel switching, empirical measurements on the first operational *Carlsbad* prototype show that the additional flexibility of DCS can indeed improve a WMN's throughput by more than a factor of 2, because DCS allows the WMN to switch to least loaded frequency channels in parts of its coverage area with heavy interference. In addition, this prototype demonstrates that the channel switching latency can be reduced to under 2 msec on commodity wireless LAN NICs and that hop-by-hop TCP could effectively eliminate the adverse impact of DCS's increased latency on transport-layer performance.

2. RELATED WORK

Several research efforts have attempted to leverage multiple NICs to support multi-channel networks. In [14] and [13], the authors proposed to use a dedicated control channel to assign channels to wireless NICs, and operate the NICs accordingly. Draves et al. [10] assumed each NIC operates in one channel throughout. and the number of NICs per node is equal to the number of available channels. It proposed a metric that considered the bandwidth and loss ratio of each link to find a high-throughput path between a source and a destination. Some proposals [8, 9, 16, 18] explored how to best leverage a limited number of NICs on each node. Raniwala et al. proposed both the centralized [9] and distributed [8] approaches to assign channel to each NIC and balance the load among NICs. *Carlsbad* focused on reaping the performance benefits of multiple radio channels on a single NIC.

So et al.[12, 17] and Liu et al.[11] assumed that it is possible to apply channel switching on a packet-by-packet basis and proposed techniques to find an optimal channel for each packet transmission. The slotted seeded channel hopping (SSCH) scheme [15] is a virtual MAC protocol on top of IEEE 802.11 MAC. Each node is assigned a pseudo-random channel hopping sequence which ensures any of two neighboring nodes overlap periodically, and thus multiple communications can take place at the same time but at different channels. SSCH assumes the channel switching delay is approximately 80 μs. In contrast, *Carlsbad* is built upon commodity WLAN access points with considerable channel switching overhead. Its channel switching frequency is lower (i.e. once every 500 msec). So et al. [17] proposed a routing protocol for multi-channel networks that each node has only one interface, which is similar to our architecture. The paper provides results based on their simulation. None of the previous works on single-interface multi-channel produced actual working systems, let alone systems that worked on commodity IEEE 802.11 NICs. As a result, all the performance results in the associated papers were based on simulations, rather than empirical measurements collected on a fully operational prototype working in a real testbed, as in the case of *Carlsbad*.

Much research [3, 4, 5] has gone into on-demand routing for mobile ad-hoc networks. In addition, there has been much research [8, 9, 10, 16, 17, 18, 19] on throughput optimization techniques for multi-channels wireless mesh networks. They used such wireless link characteristics as link quality, packet loss ratio, gateway load, local interference, and the combination of the above to achieve load balance and identify higher-throughput paths. *Carlsbad* assumes that its wireless nodes do not move and takes a similar approach to routing algorithm design by tailoring it to spanning tree topologies.

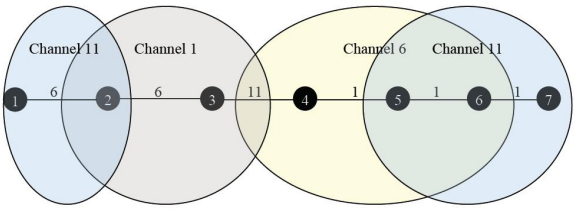

Figure 1: *An example of how* Carlsbad *performs dynamic channel switching to avoid the adverse performance impact due to interference from neighboring wireless LANs, each of whose coverage is represented as an ellipse marked with its operating channel.*

Carlsbad's spanning tree construction algorithm is similar in design to the spanning tree protocol of IEEE 802.1D [1] where in all the bridges and switches that are interconnected run an algorithm to find a spanning tree rooted at the root bridge. In [1], each switch or bridge is involved in periodic advertisement message exchange for tree maintenance. The major difference is that 802.1D does not support dynamic load balancing whereas *Carlsbad* does. In addition, *Carlsbad* automates IP address assignment as well as routing tree construction. Finally, *Carlsbad* supports spanning tree-based load-balancing and fault-tolerant routing under dynamic channel switching, in which a node cannot always directly communicate its neighbors using physical broadcast.

3. DYNAMIC CHANNEL SWITCHING

3.1 Architectural Overview

DCS allows a wireless NIC to connect to multiple wireless NICs that operate in different frequency channels during different time periods, a capability particularly useful for WMNs whose nodes have a small number of wireless NICs. Moreover, DCS allows a single-interface WMN to use different frequency channels at different parts of its coverage area so as to avoid heavily interfered channels already used by neighboring networks. For example, Figure 1 shows a linear-topology WMN traversing through multiple WLANs, each of whose coverage area is represented as an ellipse and labeled with its corresponding operating channel. To minimize interference between this WMN and the background WLANs, the radio channel time is divided into cycles, each of which in turn consists of two slots, and each wireless NIC operates in a distinct frequency channel during each of the two time slots. Assuming the underlying wireless technology is IEEE 802.11b, which has three non-overlapped channels (Channel 1, 6 and 11), Node 3 communicates with Node 2 over Channel 6 because the background wireless LANs operate in Channel 1 and 11; similarly Node 3 communicates with Node 4 over Channel 11 because the background wireless LAN operates in Channel 1 and 6.

Dynamic channel switching entails the following four design issues. First, how is the radio channel time divided into cycles and slots? Second, which frequency channel does each wireless NIC operate during each channel time slot? Third,

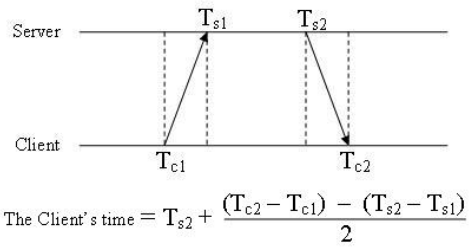

Figure 2: *Hop-by-hop clock synchronization mechanism in Carlsbad . The Client's time corresponds to the server's clock value corresponding to T_{c2}.*

how do communicating nodes synchronize with each other so that they can switch to the same channel at the right time? Finally, how is DCS integrated with wireless routing algorithms? Owing to lack of space, we will focus on the last two issues, and make the following assumptions about the first two issue:

- The radio channel time is divided into fixed-sized cycles, each of which consists of two slots of equal length.
- There is a separate radio resource manager that is capable of discovering the load on each usable channel in each local area, and informing the WMN nodes of these per-channel load information.

Assume each WMN node is equipped with only one wireless NIC, and the WMN nodes are organized into one or more spanning trees, each of which is rooted at a gateway node connected to the wired internet. Under DCS, each wireless NIC uses one of the two slots (called the *parent slot*) to communicate with its parent, and the other slot (called the *children slot*) to communicate with all of its children. A parent slot is typically of the same size as its corresponding children slot because *Carlsbad* assumes the aggregate traffic volume on a node's children links should be largely the same as that on its parent link. If a WMN node itself can also generate traffic, this assumption no longer holds, and the parent slot probably should be larger than the sum of the children slots because most traffic is to or from the wired internet, rather than among WMN nodes.

3.2 Hop-by-hop Synchronization

Dynamic channel switching requires communicating nodes to synchronize their clocks at millisecond-level accuracy so that they can switch to the same channel at the same time. NTP (Network Time Protocol) [2] is a standard protocol for synchronizing clocks of Internet nodes. Unfortunately, NTP requires *approximately symmetric* packet transmission delay, which does not always hold on multi-hop wireless links, especially when dynamic channel switching is enabled.

To minimize the impact of asymmetric packet transmission delay, *Carlsbad* adopts a *hop-by-hop clock synchronization* protocol to allow each *Carlsbad* node to tightly synchronize with its parent. Each *Carlsbad* node acts both as a synchronization server servicing synchronization requests from its children, and a synchronization client requesting to synchronize its clock with its parent. More concretely, a synchronization client records the current time T_{c1}, puts it in the payload of a synchronization request, and sends this request to the server, which immediately responds with two more time stamp T_{s1} and T_{s2}. T_{s1} is the time at which the server receives the client's request, and T_{s2} is the time immediately before the response is sent. Upon receiving the server's response, the client records the fourth time stamp, T_{c2}, and then immediately adjusts its clock using the formula listed in Figure 2. The second term of the right-hand side of the formula represents the propagation delay, which is assumed to be the same for both transmitting and receiving a packet.

To improve the accuracy of this hop-by-hop synchronization scheme, these four time stamps are taken inside the kernel right above the WLAN NIC driver. As a result, uncertainties due to context switching and process scheduling are eliminated. However, interrupt latency including interrupt mask delay still cannot be separated out. Finally, a *Carlsbad* node sends a synchronization request only when it is in the same channel as its parent, and a synchronization response is valid only when it is returned in the same channel time slot as the corresponding request. To enforce the latter, a synchronization response that returns more than 2 msec later is ignored, because empirically the round-trip time of a synchronization request is less than 2 msec.

3.3 Minimizing Channel Switching Latency

Although DCS provides frequency agility that could help avoid heavily interfered channels, it incurs additional performance overhead because a wireless NIC cannot send/receive packets when it switches from one channel to another. How to reduce the channel switching overhead to a level that does not offset the performance gain from DCS's ability to minimize interference is a major implementation challenge.

An IEEE 802.11 WLAN interface can communicate with peers in a multi-hop network in two modes: *ad-hoc* mode and *Wireless Distribution System* (WDS) mode. When a WLAN interface operates in the ad hoc mode, it always invokes a *channel scanning* procedure whenever it switches between channels. Because this channel scanning procedure is meant to locate other ad hoc networks, it may scan all available channels by sending out probe messages and waiting for responses and therefore is very time-consuming. If a node cannot find any other nodes, it forms its own ad-hoc network. The ad-hoc mode is not appropriate for DCS for the following two reasons. First, each channel switching incurs a delay of 400 to 500 milliseconds, which mainly comes from transmission of probe messages and waiting for their associated responses or time-outs. Second, an ad-hoc network may be partitioned into different islands with different network IDs (BSSIDs), even though they may share the same SSID. This is unacceptable for DCS.

The WDS mode in the IEEE 802.11 standard was originally designed to bridge traffic between two WLAN access points over a wireless link. When a WLAN NIC operates in the WDS mode, it does not need to scan channels when switching from one channel to another. As a result, the channel switching latency can be reduced to less than 2 millisecond. Because minimal channel switching latency is absolutely essential, *Carlsbad* is designed to run only in the WDS mode.

When two nodes communicate with each other in the WDS mode, each creates a WDS interface for the other. Multiple WDS interfaces, each being a pseudo interface, could be bound to one physical interface. Each *Carlsbad* node is configured to run in the *lazy WDS* mode, which allows a node to automatically create a WDS interface for a peer whenever it receive the first packet in the WDS format from that peer. In addition, each *Carlsbad* node is configured with a broadcast WDS interface, which is used to broadcast messages. Nodes in the receiving range of a broadcast WDS message automatically create a WDS interface for the broadcast message's sender. In summary, on each *Carlsbad* node, one WDS interface is created for each of its neighbor nodes.

3.4 Implementation Issues

We implement *Carlsbad* on a commodity WLAN access point, Linksys WRT54G, which comes with a 200-MHz RISC CPU, 32MB DRAM, 16MB flash memory, five 100-Mbps Ethernet ports, one 802.11g WLAN port, and a generic serial port or the Ethernet port. Moreover, WRT54G uses a stripped-down version of Linux as the control firmware. There are several third-party open-source Linux-based firmware providers on WRT54G, such as OpenWrt[6] and Sveasoft[7]. As a result, we can customize WRT54G with our modification either at the user level or the kernel level.

The implementation can be divided into two parts, switching the operating channel and queuing packets destined to non-active WDS interfaces. Switching channel involve two actions, switching the channel and then setting the access point's SSID. Both actions are implemented on top of the wireless driver's ioctl calls.

When a WMN node operates in the DCS mode, it cannot always immediately forward packets received from one neighbor to their next-hop neighbor because these two neighbors may be reachable in different channel time slots, e.g., traffic from parent nodes to children nodes or vice versa. Therefore, a *Carlsbad* node needs to buffer incoming packets and forward them only during the channel time slots in which the corresponding next-hop neighbors are reachable.

The Linux kernel in WRT54G maintains a packet queue for each WDS interface, which is a link list containing pointers to SK buffers that actually hold packets. To send a packet over a WDS interface, the Linux kernel dequeues a packet from the interface's packet queue and inserts it into the ring buffer of the WDS driver, which in turn issues a DMA request to copy the packet into the WLAN NIC, and releases the associated SK buffer after the DMA is completed. To buffer packets destined to a WDS interface, *Carlsbad* modifies the kernel function `netif_queue_stopped()` to disable dequeuing packets from that interface's kernel queue. When a WDS interface is ready for transmission because a *Carlsbad* node switches to its associated channel, *Carlsbad* sends out all the packets in the interface's queue by calling `netif_schedule()` to re-enable dequeuing of packets from the interface's kernel queue.

Packets may be lost when two DCS nodes communicate with each other but they are not active in the same channel simultaneously. This arises for two reasons. First, the clocks of two communicating DCS nodes are not perfectly synchronized. Second, after a packet is copied to a WLAN NIC, it may take a variable amount of time to get successfully transmitted, owing to back-off delay and retransmission. Because software has no control over a packet once it is copied to a WLAN NIC, it is possible that a packet is copied to a NIC before the end of a channel time slot and only gets transmitted in the next channel time slot. Because channels assigned to two adjacent time slots are different, this packet is transmitted in the wrong channel and is thus lost.

To minimize packet losses due to these two reasons, *Carlsbad* divides each channel time slot into three parts, as shown in Figure 3: *head margin*, *operating period*, and *tail margin*. A *Carlsbad* node DMAs packets to its WLAN NIC only during the operating period, but not during head and tail margins. However, it receives packets in the entire channel time slot. The head margin is meant to compensate for imperfect clock synchronization, whereas the tail margin is for imperfect clock synchronization and variable physical packet transmission delay.

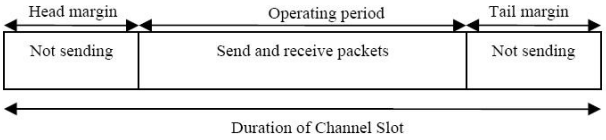

Figure 3: *A channel time slot is divided into three parts, the head margin, operating period, and tail margin, whose lengths are empirically determined.*

4. SPANNING TREE-BASED ROUTING

4.1 Overview

Although DCS is particularly useful for single-interface WMNs, the same frequency agility advantage also applies to multi-interface WMNs. However, we focus only on single-interface WMNs in this paper. In the following, we assume a WMN in which every node is equipped with only one IEEE 802.11-based wireless NIC, and one or more of these nodes (called *gateways*) are directly connected to the wired internet. *Carlsbad* supports two modes of operation: *single-channel* mode, in which every NIC operates only in one channel, and *multi-channel* mode, in which a NIC may operate in different channels during different time periods.

Carlsbad chooses a spanning tree topology because most of the traffic in a WMN is between WMN nodes and the wired internet and there is not much peer-to-peer traffic within the WMN. To build up these spanning trees in a distributed fashion, after associating itself with a spanning tree, a *Carlsbad* node *periodically* broadcasts an advertisement message consisting of the following information:

- *Routing metric:* the estimated load going through the current gateway with which the node is associated.
- *Path information:* the list of nodes on the path between the node and its gateway.
- *Sequence number:* a unique identifier for each advertisement to avoid duplicated messages and to calculate message loss ratio.

For a gateway node, it broadcasts an advertisement message periodically after it boots up to recruit other non-gateway nodes to join its tree. For a non-gateway node, after boot-up, it keeps silent, waits for advertisement messages from all of its live neighbors, and builds a *local neighborhood map* for those nodes that are reachable in one hop. Each non-gateway node compares the routing metrics in the received advertisement messages, and picks one of the neighbors as its parent and sends a join request to the chosen neighbor to be a member of the associated spanning tree. If the join request is approved, the parent node acknowledges the request, updates its local routing table, and informs its ancestor nodes of the newly joined node; otherwise, the non-gateway node tries to join the next best neighbor until it exhausts all its neighbors.

Each *Carlsbad* node maintains a routing table entry for each of its descendant nodes. For example, a gateway node keeps a routing table entry for every node in its associated spanning tree. However, a *Carlsbad* node does not need to maintain routing table entries for its ancestor nodes. Instead, each *Carlsbad* node uses its parent as the default router, that is, packets whose destination is not one of its descendent nodes are simply forwarded to its parent node. As a result, a *Carlsbad* node can reach its ancestors through its default router, and its sibling nodes using a triangle route via their common ancestor node.

When a node joins a spanning tree, its parent serves as a DHCP proxy and requests for a new IP address from the wired DHCP server on behalf of the newly joined node. For gateway nodes, they acquire their IP addresses through the standard DHCP protocol from the DHCP servers on the wired subsets with which they are associated.

In summary, given a set of wireless nodes, *Carlsbad*'s single-channel routing algorithm allows each node to obtain its

local physical connectivity information, acquire a unique IP address, and participate in a routing spanning tree, all without any human intervention.

4.2 Load Balance and Fault Tolerance

To maximize the utilization efficiency of precious radio resources, it is essential that the loads on a WMN's be balanced. In addition, a WMN should be able to quickly reroute traffic around failed links or nodes so as to maintain robust network connectivity in the presence of failures. In *Carlsbad*, each non-gateway node maintains a list of backup parents, which are its neighbors that are neither current parent nor current descendants, and updates their routing metrics when receiving advertisements from them. When a *Carlsbad* node detects a back-up parent with a better routing metric, it brings itself and its descendents to join the new parent and disconnects itself and its descendents from the old parent. Then both the new parent node and old parent node inform their ancestor nodes of this route change event, and these ancestor nodes update their routing table entries with respect to the *Carlsbad* node and its descendents. Because advertisements are sent out periodically, each *Carlsbad* node is *constantly* comparing potential parents and choosing the best one to associate itself with. Through this mechanism, a *Carlsbad* WMN can continuously re-arrange itself to balance the loads on the gateways.

To perform load-balancing routing, each *Carlsbad* node maintains a windowed running average of the upstream and downstream traffic load on the path to each of its parents, and uses the following criterion: $PathLoad_{CurrentParent} - LocalLoad > PathLoad_{PotentialParent} + LocalLoad + \alpha$ to determine if switching from its current parent to a backup parent will result in a more balanced load among the gateways, where $LocalLoad$ represents the node's locally generated traffic load and α is an empirical parameter designed to prevent route oscillation. In addition, to prevent all nodes from switching their parents simultaneously, *Carlsbad* requires each node that plans to switch parent to pick a random back-off timer value, and to physically switch parent only when the timer expires and the gateways' loads are still not balanced at that point.

To perform fault-tolerant routing, each *Carlsbad* node constantly monitors the advertisements from its parent to determine if it is unreachable because of a link or node failure. Whenever a *Carlsbad* node, say N, detects that its parent is unreachable, it quickly tries to associate itself and its descendents with one of N's back-up parents. If N can successfully associate itself and its descendents with one of the back-up parents, it does not need to inform its descendents of the failure. However, if none of the back-up parents are accessible, then N has to involve its children in failure recovery. In particular, N will send a message to its children to trigger their failure recovery logic, as if the link between N and its children is dead. After this message, N's children will attempt to connect with their back-up parents, and N itself will enter the silent state to wait until a new parent candidate (most likely one of its former children) shows up. If necessary, the same distributed failure recovery procedure is applied recursively throughput the entire subtree under N, until either finding a link to other spanning trees or concluding that N's subtree is isolated from the rest of the WMN.

4.3 Integration with DCS

Under dynamic channel switching, a WMN node can no longer assume that it can always receive advertisement messages from its immediate neighboring nodes, because a neighbor node may use a different channel when broadcasting its advertisement messages. To overcome this problem, *Carlsbad* requires each WMN node to send its advertisement messages, which include the channel slot schedule and the channel used in each slot, to the root of its spanning tree, which periodically generates an aggregate advertisement from per-node advertisements and broadcasts it to all WMN nodes, including those in other spanning trees. From each aggregate advertisement message received, each *Carlsbad* node can keep track of each neighbor's status, including whether it is alive, its associated path load to its root, its channel slot schedule and the channels it uses. Given that each *Carlsbad* node is able to keep track of the advertisements of its neighbors through the aggregate advertisement mechanism, it can now apply the same load-balancing and fault-tolerant routing algorithm described previously, with one exception. When a node A detects that its current parent is unreachable and tries to associate itself with a new parent node B, it needs to take two additional steps (Here we assume each cycle consists of a parent slot and a children slot, and the children slot is not further subdivided.):

- Identify the channel time slot that B uses to communicate with its child nodes, assign that channel time slot as A's parent slot, and adjust the channel time slots and their channel assignments for A's descendants recursively.
- Set the channel of A's parent slot to that of B's children slot and let A send a join request to B.

When a *Carlsbad* network starts, it starts in the single-channel mode, and after the spanning trees stabilize, each root issues a DCS mode switching command to request every node under its tree to start running in the multi-channel mode. This command is delivered reliably, and a parent node won't turn into the multi-channel node until it is sure that all its child nodes have received this command. A root node can also request its descendants to switch back to the single-channel mode when a *Carlsbad* network encounters concurrent failures and the network starts the reconfiguration process.

In the non-DCS mode, a newly joining node keeps silent and waits for advertisement messages from its immediate neighbors. However, in the DCS mode, a newly joining node actively scans each available channel to identify its physical neighbors and probes them. More concretely, whenever a new node, say A, scans a channel, it broadcasts multiple probe messages to check if any nodes operating in that channel; if a node, say B, receives a probe message from A, it adds A into its local neighborhood map, and responds to A with its latest advertisement message; When A receives B's response, it adds B to its local neighborhood map. A new node stays in each scanned channel long enough (currently 3 seconds) to make sure its neighboring nodes have a chance to receive and respond to its probe messages, and then moves on to the next channel. After scanning all the channels, a new node selects the neighbor with the best routing metric as its parent. When a new *Carlsbad* node boots up, it starts in the single-channel mode for a period of time to check if the rest of the network is in the single-channel mode. If not, the new node switches to the multi-channel mode and starts channel scanning.

5. PERFORMANCE EVALUATION

5.1 Dynamic Channel Switching

5.1.1 Testbed Setup

The first *Carlsbad* prototype is implemented on a set of commercial IEEE 802.11b wireless LAN access points, Linksys's WRT54G, which runs on Linux and costs about $40 USD each. We used a simple wireless testbed shown in Figure 4

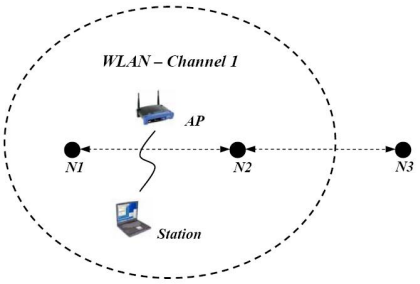

Figure 4: *The wireless testbed used in this study consists of a 2-node background WLAN, an AP and a station, and a 3-node 2-hop linear network with nodes N_1, N_2, and N_3.*

to evaluate the effectiveness of the *Carlsbad* prototype's dynamic channel switching mechanism. The testbed contains a 3-node *Carlsbad* network (N_1, N_2 and N_3) with a linear topology and a 2-node background WLAN. Each linear network node's signal can only reach its immediate neighbor. That is, N_3 is a hidden node for N_1. The background WLAN consists of an AP and a station, and their radio signals can reach N_1 and N_2, but not N_3. All of them transmit packets at the physical rate of 11 Mbps. When sending traffic on both networks, we used UDP packets with IP packet size of 1500 bytes, set the contention window CW_{min} to 15 and CW_{max} to 1023, and transmitted them as IEEE 802.11 frames with long preamble and PLCP header. The channel time slot size is set to 500 msecs by default unless specified otherwise. We measured a network's throughput based on IP packet size without including IEEE 802.11 header.

5.1.2 Gain from Congested Channel Avoidance

We first measured the baseline throughput of the *Carlsbad* network and the background WLAN when they are isolated from each other and sent traffic at full speed over Channel 1. The results are shown in the isolated columns of the non-DCS row of Table 1. The maximum throughputs of the *Carlsbad* network and the WLAN are 3.37 Mbps and 6.72 Mbps, respectively. The *Carlsbad* network's throughput is about half of that of the WLAN because the former has twice as many hops as the latter. Then we measured their throughputs when both are active simultaneously: The AP was pumping traffic to the station and N_1 was transmitting packets to N_3 via N_2 at full speed. The throughputs of the *Carlsbad* network and WLAN are reduced to 2.09 Mbps and 2.50 Mbps, respectively. In this test, there are three senders in the same collision domain, N_1, N_2 and the AP. The sum of the transmission rates of these three senders is 6.78 Mbps, which is almost the same as the isolated WLAN's throughput.

To understand how effectively dynamic channel switching could mitigate the performance degradation due to interference, we performed two experiments. In the first experiment, the *Carlsbad* network shared only one channel with the WLAN: the link $N_1 - N_2$ used Channel 1 and the link $N_2 - N_3$ used Channel 11. In the second experiment, the *Carlsbad* network did not share any channel with the WLAN: the link $N_1 - N_2$ used Channel 6 and the link $N_2 - N_3$ used Channel 11.

As shown in the Carlsbad/Isolated column of the DCS rows, the isolated throughput of the *Carlsbad* network when DCS is turned on is 3.08 Mbps, which is independent of the number of shared channels and is about 93% of the theoretical optimum (3.31 Mbps), which is one half of a one-hop WDS link's measured throughput (6.62 Mbps) because each WLAN NIC is half duplex and thus can either send or receive but not both. The missing 7% results from the channel switching overhead. The isolated throughput of the background WLAN is unaffected by whether the *Carlsbad* network turns on DCS or not.

Configuration	*Carlsbad*		WLAN	
	ISOL	SIMU	ISOL	SIMU
non-DCS	3.37	2.09	6.72	2.50
DCS (shared=1)	3.08	1.67	6.72	5.05
DCS (shared=0)	3.08	3.08	6.72	6.70

Table 1: *The effectiveness of dynamic channel switching in minimizing the interference between the* Carlsbad *network and the background WLAN in the testbed. Both networks try to send packets at full speed. ISOL and SIMU are abbreviations for Isolated and Simultaneous, respectively. Throughput is measured in Mbps.*

When DCS was turned on and the *Carlsbad* network and WLAN sent traffic simultaneously over one shared channel, the throughput of the *Carlsbad* network is 1.67 Mbps and that of the WLAN is 5.05 Mbps. The *Carlsbad* network's throughput decreases from 2.09 Mbps (non-DCS) to 1.67 Mbps because N_1 only spends half of its time competing for Channel 1's resource and the other half being silent as N_2 was transmitting packets to N_3 over Channel 11. For the same reason, the WLAN's throughput increases from 2.50 Mbps (non-DCS) to 5.05 Mbps.

When DCS is turned on and the *Carlsbad* network and WLAN do not share any channels, these two networks are essentially isolated from each other. As a result, their "simultaneous" throughputs (6.70 Mbps) are almost identical to their isolated throughputs (6.72 Mbps). The combined throughput of the two under DCS without channel sharing is 2.35 times as high as that under non-DCS.

To quantify the impact of intra-channel interference, we fixed the injected traffic rate of the *Carlsbad* network at 1.5 Mbps, and varied the number of links that shared the same channel as the background WLAN among 0, 1 and 2. The measured throughputs of the background WLAN are 6.72 Mbps, 5.25 Mbps and 3.41 Mbps, respectively. This result shows that by carefully avoiding already used channels, DCS can improve the throughput of neighboring WLANs by a factor of close to two.

5.1.3 Impact of Head/Tail Margin

To avoid packet loss in the transition from one channel to another, *Carlsbad* stops copying packets into the NIC during head and tail margins. To empirically determine the best size for head and tail margins, we set up a 4-node linear network ($N_1 - N_2 - N_3 - N_4$), each of whose links ran over a non-overlapped channel. Each node is able to communicate with its immediate neighboring nodes, but is outside the sensing range of all other nodes. For example, N_2 can communicate with N_1 and N_3 directly, but cannot communicate with N_4. N_1 sent 1500-byte UDP packets to N_4. All other parameters were set in the same way as in previous subsections. By tagging each UDP packet with a unique ID, we could measure the UDP packet loss ratio at N_4.

The channel slot time was set to 500 milliseconds, that is, each node operated in one channel for 500 milliseconds and then switched to the next. We fixed the head margin at 30 milliseconds and varied the tail margin from 0 to 30 milliseconds, and measured the impact of tail margin size on the packet loss ratio. The results are shown in Figure 5, whose X axis is the tail margin size and the Y axis is the measured packet loss ratio. When the tail margin size is 0, the perceived packet loss ratio is the highest at 1.43%. As the tail margin size increases, the packet loss ratio decreases because more synchronization errors and transmission delay variations can be accommodated. 10 milliseconds is a turning point, as the packet loss ratio remains flat at 0.02% after

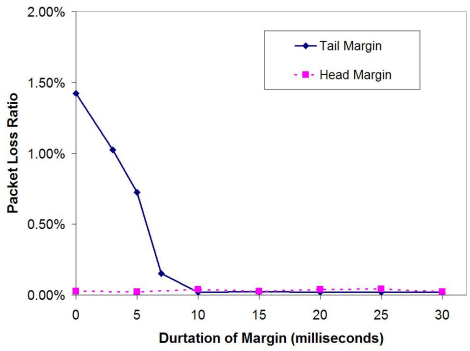

Figure 5: *The impact of varying the tail (head) margin size on the packet loss ratio, while fixing the head (tail) margin size at 30 (10) milliseconds.*

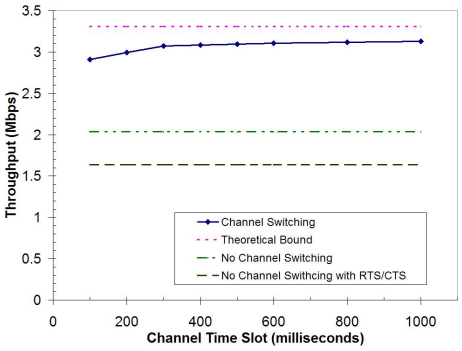

Figure 6: *UDP throughput of a 4-node linear network running DCS when the channel slot time is varied from 100 milliseconds to 1000 milliseconds.*

the tail margin size becomes greater than 10 milliseconds. Moreover, most of these packet losses result from random bit corruption rather than dynamic channel switching, because these losses are uniformly distributed over time rather than clustered during the channel transition periods.

Next we examined the impact of the head margin size on the packet loss ratio while setting the tail margin to 10 milliseconds. Results in Figure 5 suggested that head margin may not be necessary because with the default retransmission count of 4 and a 1500-byte packet taking more than 10 msec to transmit at 11 Mbps, the amount of time required to retransmit an IEEE 802.11 frame four time is more than sufficient to accommodate most cases in which the a receiver node switches to the common channel a little bit later than the sender.

From the above experiments, we recommend setting the head margin to 0 and the tail margin to 10 milliseconds for IEEE 802.11b WLAN NICs. However, they may need to be re-tuned for different physical transmission rates or different retransmission counts.

5.1.4 Impact of Channel Time Slot Size

To understand the impact of the channel time slot size on a *Carlsbad* network's UDP throughput, we used the same 4-node linear network and experiment set-up as before. The head margin was set to 0 milliseconds and the tail margin was set to 10 milliseconds. Larger channel time slots lower the relative performance impact of head/tail margins and channel switching latency, but increase the end-to-end delay and potentially degrade the throughput of TCP connections. Figure 6 shows the perceived UDP throughput at

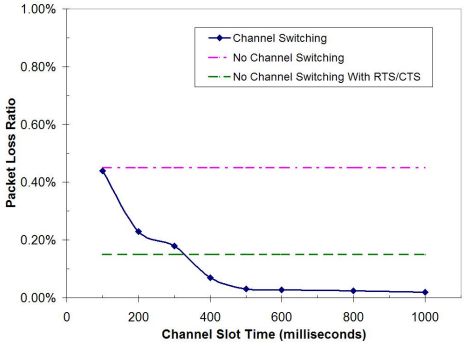

Figure 7: *Packet loss ratio of a 4-node linear network running DCS when the channel slot time is varied from 100 milliseconds to 1000 milliseconds.*

N_4 when the channel time slot is varied from 100 milliseconds to 1000 milliseconds. In addition to DCS's throughput curve, there are three other straight lines, which correspond to the theoretical throughput bound of a 4-node linear network, the measured throughput of a non-DCS configuration with RTS/CTS turned off, and the measured throughput of a non-DCS configuration with RTS/CTS turned on. These three configurations have nothing to do with DCS and therefore their throughputs are independent of the channel time slot size.

The theoretical throughput is half of a one-hop WDS link's measured throughput ($6.62/2 = 3.31$ Mbps). The measured throughput of the 4-node network is 2.03 Mbps when both DCS and RTS/CTS are turned off, and is 1.64 Mbps when DCS is turned on and RTS/CTS is turned off. RTS/CTS could mitigate the hidden node problem, but also introduces additional performance overheads due to extra control frames. The DCS configuration's throughput is significantly better than the non-DCS configurations, by a factor of more than 2, because the former forces a TDMA-like behavior on media access and thus significantly reduces contention delays. As the channel time slot size increases, the gap between the DCS and non-DCS configuration increases because the relative overhead of head/tail margins diminishes.

As the channel slot time increases, the packet loss ratio decreases and eventually stabilizes, as shown in Figure 7. When DCS is disabled, the packet loss ratio is 0.45%, which is mainly attributed to the hidden node problem. Turning on RTS/CTS reduces the packet loss ratio to 0.15%. Enabling DCS further reduces the packet loss ratio to 0.02% because it completely eliminates the hidden node problem. Most of these remaining packet losses result from channel errors. RTS/CTS cannot solve the starvation problem due to hidden nodes because it is designed to avoid transmission collision from the hidden node, but does nothing to stop a node from dropping packets because its RTS never gets acknowledged, which is the main reason of packet loss in this experiment.

5.1.5 TCP Throughput

A major weakness of dynamic channel switching is its increase in the end-to-end delay, which could have an adverse effect on TCP connections' throughput. To a first approximation, a TCP connection's throughput is the congestion window size divided by RTT. In the 4-node linear network, the RTT is roughly 3 seconds with a channel slot time of 500 milliseconds. Assuming a congestion window size of 64 Kbytes, the throughput of this network tops out at about 174 Kbps. In addition long round-trip delay (RTT) slows down the growth rate of the congestion window in the presence of packet losses.

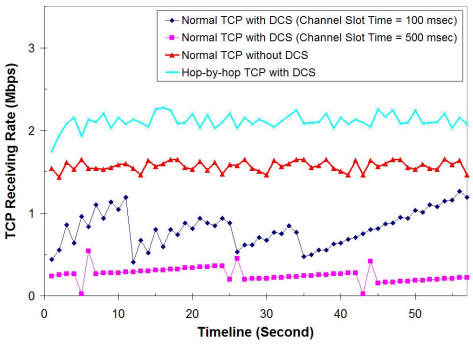

Figure 8: *TCP throughput of a 4-node linear network running DCS when varying the channel time slot from 100 msec to 500 msec, and applying TCP on an end-to-end basis or a hop-by-hop basis.*

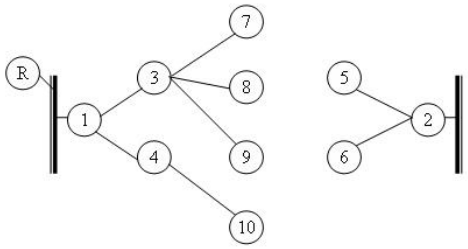

Figure 9: *An 10-node wireless mesh network testbed used in the performance study of load-balancing routing*

In this experiment, N_1 sent 4000 1500-byte TCP packets to N_4 over a 4-node linear network running DCS using TCP Reno, with the following TCP features turned on, TCP timestamp, SACK, DSACK, and FACK. In addition we turned on the TCP window scaling option to make sure that the congestion window can grow to a sufficiently large value to fully utilize the network bandwidth. We measured the perceived throughput at N_4 when the channel slot time is 100 milliseconds and 500 milliseconds.

As shown in Figure 8, the 100-msec configuration performs better than the 500-msec configuration because the former can recover the congestion window faster than the latter, as indicated by the slope of the throughput curves when they move upwards. However, neither configuration's maximal

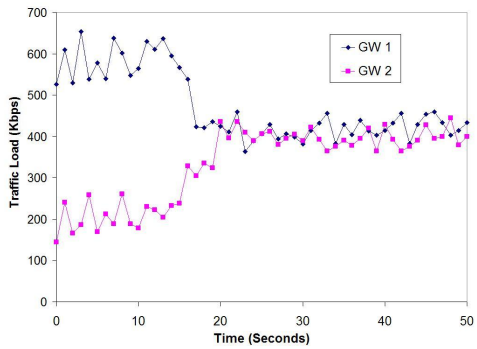

Figure 10: *The evolution of the traffic loads on the two gateways of the testbed under load-balancing routing in the DCS mode*

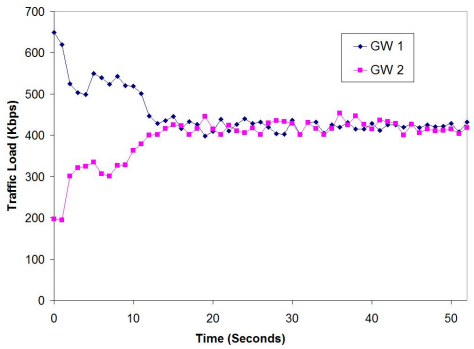

Figure 11: *The evolution of the traffic loads on the two gateways of the testbed under load-balancing routing in the non-DCS mode*

Configuration	Detection	*Recovery*	Total
Case 1, non-DCS	3.70	0.25	3.95
Case 1, DCS	6.51	0.45	6.96
Case 2, non-DCS	3.95	1.40	5.35
Case 2, DCS	6.72	11.72	18.44

Table 2: *Breakdown of the total failure recovery time for the two test cases into the failure detection and failure recovery components*

throughput exceeds 1.3 Mbps, which is much lower than the measured UDP throughput of 3 Mbps in Section 5.1.4.

To demonstrate that it is possible to provide similar TCP throughput to UDP throughput on a DCS network, we implemented a hop-by-hop TCP mechanism into *Carlsbad*: A M-hop TCP connection is is broken into M 1-hop TCP connections. Because TCP's retransmission works within a single hop, whenever a packet loss occurs, the congestion window could recover within the same channel time slot in most cases. As a result, the throughput of DCS combined with hop-by-hop TCP when the channel slot time is 100 msec performs even better than non-DCS combined with end-to-end TCP, as shown in Figure 8. On the average, the sustained throughput of the DCS plus hop-by-hop TCP configuration is 2.1 Mbps, whereas that of the non-DCS configuration is only 1.56 Mbps.

5.2 Load Balance and Fault Tolerance

To evaluate the effectiveness of *Carlsbad*'s load-balancing routing, we ran *Carlsbad* on a 10-node 2-gateway testbed whose topology is shown in Figure 9, where R is a control server on the wired network. Each node in the testbed sent a UDP stream of 100 Kbps to R. Initially, the loads on the two gateways, GW1 and GW2, are very different because the initial topology is skewed, but after 20.5 seconds, two of the nodes in the middle column switch to GW2 and the gateways' loads are balanced, as shown in Figure 10. We repeated the same experiment but turned off DCS, and the time taken to balance the loads of GW1 and GW2 is reduced to 14.5 seconds, as shown in Figure 11. The difference between the two mainly comes from the fact that under DCS a *Carlsbad* node can communicate with a physical neighbor only in its channel time slot, and therefore it takes longer to probe back-up parents and re-associate with a new parent.

To assess the effectiveness of *Carlsbad*'s fault-tolerant routing, we ran *Carlsbad* on a 10-node 2-gateway testbed whose topology is shown in Figure 12. We tried two failure cases: (1) Node 3 and 4 die, and (2) Node 1 dies. In both cases, the affected nodes successfully switched to the alternative gate-

way. As shown in table 2, the total failure recovery times for Case (1) and (2) under the DCS mode are 6.96 sec and 18.44 sec, respectively. For Case (1), more than 92% of the failure recovery time is spent on failure detection, whereas for Case (2), more than 60% of the failure recovery time is spent on failure recovery, because Node 3 and 4 in this case probed all possible parents only to give up and ask their children (Node 7 and 8) to take over the failure recovery operation. In both test cases, the total failure recovery time in the DCS mode is much longer than that in the non-DCS mode, because the fact that DCS prevents a node from immediately communicating with its physical neighbors is an even more serious restriction during failure recovery. How to overcome this restriction to speed up failure recovery is a topic for future research.

6. CONCLUSION

Because of interferences, application-perceived throughput fluctuates more and thus is less robust on wireless networks than on wired networks. The most promising way to mitigate interference-induced performance degradation on wireless networks is to make the most of available radio spectrum. Traditionally, a wireless interface operates in one frequency channel at a time, because it is generally considered expensive to switch between channels on a granular basis, especially on standards-based wireless networks. This paper presents a fully working dynamic channel switching prototype that demonstrates, for the first time, not only it is technically feasible to implement DCS on IEEE 802.11-based wireless networks, but also DCS can deliver substantial performance improvements despite its additional channel switching overhead.

Although dynamic channel switching has been explored before, it has never been fully implemented, tested and shown to work on multi-hop wireless networks that are built from commodity hardware. In addition, none of previous works investigated how to integrate DCS with upper-layer routing and transport protocols. The *Carlsbad* system described in this paper successfully integrates dynamic channel switching with spanning tree-based fault-tolerant and load-balancing routing, and shows that it is indeed practical to apply dynamic channel switching to off-the-shelf wireless LAN NICs. The research contributions of this paper include

- A low-latency channel switching implementation that reduces the channel switching latency to below 2 msec on commodity IEEE 802.11 WLAN NICs,
- A spanning tree-based load-balancing and fault-tolerant routing algorithms that works with dynamic channel switching, and
- A fully operational prototype that empirically demonstrates that by leveraging DCS to avoid congested channels, *Carlsbad* could deliver a performance improvement as high as a factor of 2 for multi-hop UDP/TCP connections.

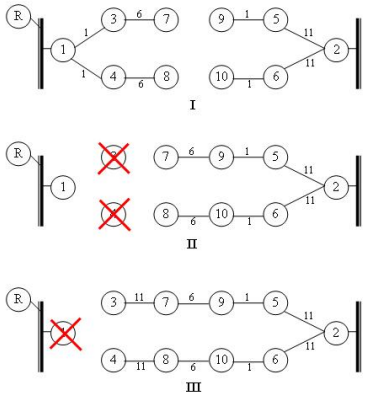

Figure 12: *The original topology used in the fault-tolerant routing study and the resulting topologies when some of the nodes fail.*

7. REFERENCES

[1] IEEE 802 Working Group; "IEEE 802.1D: IEEE MAC Bridges standard draft 4"; Nov. 2003.

[2] D. Mills; "Network Time Protocol (Version 3) Specification, Implementation and Analysis"; IETF RFC 1305, Mar. 1992.

[3] C. Perkins, E. Belding-Royer, and S. Das; "Ad Hoc On Demand Distance Vector (AODV) Routing"; IETF RFC 3561, July 2003.

[4] T. Clausen, P. Jacquet; "Optimized Link State Routing Protocol (OLSR)"; IETF RFC 3626, Oct. 2003.

[5] D. Johnson, Y. Hu, and D. Maltz; "The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4"; IETF RFC 4728, Feb. 2007.

[6] OpenWRT Team; "OpenWRT: A Linux distribution for WRT54G"; http://openwrt.org

[7] Sveasoft Inc. http://www.sveasoft.com

[8] A. Raniwala, T. Chiueh; "Architecture and Algorithms for an IEEE 802.11-based Multi-channel Wireless Mesh Network"; in Proc. of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'05), Miami, FL, Mar. 2005.

[9] A. Raniwala, K. Gopalan, and T. Chiueh; "Centralized channel assignment and routing algorithms for multi-channel wireless mesh networks"; ACM Mobile Computing & Comm Review (MC2R), April 2004.

[10] R. Draves, J. Padhye, and B. Zill; "Routing in Multi-radio, Multi-hop Wireless Mesh Networks"; in Proc. of the 10th annual international conference on Mobile computing and networking (MobiCom'04), Philadelphia, PA, Sep. 2004.

[11] Y. Liu, E. Knightly; "Opportunistic Fair Scheduling over Multiple Wireless Channels"; in Proc. of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'03), San Francisco, CA, April 2003.

[12] J. So, N. Vaidya; "Multi-Channel MAC for Ad Hoc Networks: Handling Multi-Channel Hidden Terminals Using A Single Transceiver"; in Proc. of the 5th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC'04), Japan, May 2004.

[13] S. Wu, C. Lin, Y. Tseng, and J. Sheu; "A new multi-channel mac protocol with on-demand channel assignment for multi-hop mobile ad hoc networks"; in Proc. of the 5th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'00), Dallas/Richardson, TX, Dec. 2000.

[14] J. Deng and Z. Haas; "Dual Busy Tone Multiple Access (DBTMA): A New Medium Access Control for Packet Radio Networks"; in Proc. of IEEE 1998 International Conference on Universal Personal Communications (ICUPC'98), Florence, Italy, Oct. 1998.

[15] P. Bahl, R. Chandra, J. Dunagan; "SSCH: slotted seeded channel hopping for capacity improvement in IEEE 802.11 ad-hoc wireless networks"; in Proc. of the 10th annual international conference on Mobile computing and networking (MobiCom'04), Philadelphia, PA, Sep. 2004.

[16] M. Alicherry, R. Bhatia, and L. Li; "Joint channel assignment and routing for throughput optimization in multi-radio wireless mesh networks"; in Proc. of the 11th annual international conference on Mobile computing and networking (MobiCom'05), Cologne, Germany, Sep. 2005.

[17] J. So and Nitin H. Vaidya; "A Routing Protocol for Utilizing Multiple Channels in Multi-Hop Wireless Networks with a Single Transceiver", Tech. Rep., University of Illinois at Urbana-Champaign, Oct. 2004.

[18] P. Kyasanur and N. Vaidya; "Routing and interface assignment in multi-channel multi-interface wireless networks"; In Proc. of Wireless Communications and Networking Conference (WCNC'05), New Orleans, LA, Mar. 2005.

[19] J. Tang, G. Xue, and W. Zhang; "Interference-Aware Topology Control and QoS Routing in Multi-Channel Wireless Mesh Networks"; In Proc. of the 6th ACM international symposium on Mobile ad hoc networking and computing, Urbana-Champaign, IL, May 2005

[20] Steven Ashley, "Cognitive Radio," Scientific American, February 20, 2006.