Dear Author,

Any queries or remarks that have arisen during the processing of your manuscript are listed below and highlighted by flags in the proof. Please check your proof carefully and mark all corrections at the appropriate place in the proof (e.g., by using on-screen annotation in the PDF file) or compile them in a separate list.

For correction or revision of any artwork, please consult [http://www.elsevier.com/artworkinstructions](http://www.elsevier.com/artworkinstructions).

**Articles in Special Issues**: Please ensure that the words 'this issue' are added (in the list and text) to any references to other articles in this Special Issue.

<table>
<thead>
<tr>
<th>Location in article</th>
<th>Query / remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>Please provide authors’ biography and photographs.</td>
</tr>
<tr>
<td>Q2</td>
<td>Please provide complete postal address for affiliations 'a' and 'b'.</td>
</tr>
<tr>
<td>Q3</td>
<td>Please check the insertion of the country name in affiliation 'b'.</td>
</tr>
<tr>
<td>Q4</td>
<td>Please check Telecommunication data.</td>
</tr>
<tr>
<td>Q5</td>
<td>Please provide complete details for Ref. [17].</td>
</tr>
<tr>
<td>Q6</td>
<td>Please provide complete details for Ref. [21].</td>
</tr>
</tbody>
</table>

**Uncited references**: References that occur in the reference list but not in the text – please position each reference in the text or delete it from the list.

**Missing references**: References listed below were noted in the text but are missing from the reference list – please make the list complete or remove the references from the text.

Electronic file usage

Sometimes we are unable to process the electronic file of your article and/or artwork. If this is the case, we have proceeded by:

- [ ] Scanning (parts of) your article
- [ ] Rekeying (parts of) your article
- [ ] Scanning the artwork

Thank you for your assistance.
Implementation and experimental evaluation of multi-channel MAC protocols for 802.11 networks

Long Le\textsuperscript{a,}\textsuperscript{*} and Injong Rhee\textsuperscript{b}

\textsuperscript{a}NEC Laboratories Europe, 69115 Heidelberg, Germany
\textsuperscript{b}Department of Computer Science, North Carolina State University, United States

ABSTRACT

Multi-channel MAC protocols have recently obtained considerable attention in wireless networking research because they promise to increase capacity of wireless networks significantly by exploiting multiple frequency bands. However, most of these protocols remain as pure academic interest since they only exist on paper and in simulation code but have no practical implementation. In this paper, we report lessons learned from our endeavor in which we implement three representative multi-channel MAC protocols: Asynchronous Multi-channel Coordination Protocol (AMCP), Multi-channel MAC (MMAC), and Slotted Seeded Channel Hopping (SSCH) on off-the-shelf IEEE 802.11 hardware. We explore practical impacts of these multi-channel MAC protocols and present results of our experimental performance evaluation. The major findings of our performance evaluation are: (1) all multi-channel MAC protocols underperform the original 802.11 MAC at low load, (2) all multi-channel MAC protocols give better performance than the original 802.11 MAC at medium and high load, (3) AMCP performs worst among all multi-channel MACs in one-hop and multi-hop 802.11b scenario but delivers the best performance in multi-hop 802.11a scenario, and (4) SSCH attains the best results in one-hop scenarios or at low loads but loses its effectiveness at high loads in multi-hop scenarios.

© 2010 Elsevier B.V. All rights reserved.

1. Introduction and motivation

As wireless technologies continue to mature, they have become an integral part of the Internet. However, limited bandwidth resources of the wireless medium and interference between nodes pose serious obstacles to the deployment of high-performance applications in wireless networks. This problem has been tackled by limiting interference and obtaining efficient usage of bandwidth resources. The IEEE 802.11 standard divides the wireless spectrum into different spectral bands called "channels". This allows simultaneous communications and limits interference between nodes. Besides allowing the coexistence of multiple wireless networks on different channels, frequency division increases capacity of 802.11 wireless networks in infrastructure mode because access points can avoid interference by operating on different channels. However, ad hoc networks cannot utilize frequency diversity because all nodes in an ad hoc network generally tune to a common channel. Thus, capacity of ad hoc networks is generally restricted to the bandwidth of a single channel.

Recent work has addressed the capacity limitation of ad hoc networks in multi-channel environments either at the MAC layer [3,26,32,35] or at the routing layer [8,11,20,21,29]. Solutions at the routing layer generally require multiple radio interfaces per node and perform either static or dynamic channel assignments such that radio interfaces of neighboring nodes have one or more common channels. On the other hand, solutions at the...
MAC layer generally do not require multiple radio interfaces. This paper focuses on the implementation and experimental evaluation of multi-channel solutions at the MAC layer.

Many multi-channel MAC extensions for IEEE 802.11 have been proposed in recent years [3,24,38,32,34,35]. However, all these protocols remain as pure academic interest since they only exist in simulation code but have no practical implementation. In this paper, we present our implementation of three representative multi-channel MAC protocols: Multi-channel MAC (MMAC), Slotted Seeded Channel Hopping (SSCH), and Asynchronous Multi-channel Coordination Protocol (AMCP) on off-the-shelf IEEE 802.11 hardware. To our best knowledge, our work is the first practical implementation and performance study of multi-channel MAC protocols for IEEE 802.11 hardware. We discuss lessons learned from our endeavor.

Our motivations for choosing IEEE 802.11 as our hardware platform are threefold: First, since IEEE 802.11 is already widely deployed, an implementation of multi-channel MAC protocols based on IEEE 802.11 hardware would be ready for deployment at present time. Second, since its hardware is nowadays universally available, an implementation on off-the-shelf IEEE 802.11 hardware would encourage experimental research to achieve practical solutions that can have an impact on real-world problems. Third, the wireless spectrum for IEEE 802.11 is getting increasingly crowded since private wireless networks are currently being deployed at a fast rate [2,6]. This situation demands practical solutions that can make efficient usage of bandwidth resources.

The main contributions of our paper are as follows. First, we present our implementation of three representative multi-channel MAC protocols: MMAC, SSCH, and AMCP on off-the-shelf IEEE 802.11 hardware. To our best knowledge, our work is the first practical performance study of multi-channel MAC on IEEE 802.11 hardware. Second, we identify a number of problems of these multi-channel MAC protocols that were not considered in their original designs. We present our solutions to these problems. Third, we explore practical impacts of these multi-channel MAC protocols and present results of an experimental performance evaluation conducted in our wireless testbed. The rest of our paper is organized as follows. Section 2 reviews related work and covers background in multi-channel MAC protocols. Section 3 presents details of our implementation of AMCP, MMAC, and SSCH. Section 4 provides details of our wireless testbed and our experimental methodology. Section 5 presents results of our experimental performance evaluation for AMCP, MMAC, and SSCH. Section 6 discusses lessons learned from our endeavor. Section 7 concludes our paper.

2. Background and related work

In this section, we discuss background information for multi-channel MAC. In particular, we provide details of the three multi-channel MAC protocols that we implement and evaluate in this work: AMCP, MMAC, and SSCH. We also review practical implementations of existing solutions for multi-channel wireless networks.

2.1. Multi-channel MAC protocols

2.1.1. Dynamic channel assignment

The Dynamic Channel Assignment (DCA) protocol uses two radio interfaces per node to operate on one control channel and multiple data channels [38]. In DCA, one radio interface is tuned to the control channel and is called the control interface. The other radio interface is called the data interface and is dynamically switched to one of the data channels to transmit packets and acknowledgments. Each node in DCA maintains a channel usage list CUL for all data channels. Nodes update their CUL based on the control frames\(^2\) that they overhear on the control channel. Further, each node also maintains a free channel list FCL that it dynamically computes from its CUL. DCA uses three types of control frames on the control channel for channel reservation: request-to-send (RTS), clear-to-send (CTS), and reservation (RES). When host A wants to send data to host B, host A inserts its FCL into a RTS frame and sends the frame to host B. Host B matches host A’s FCL with its own FCL to select an available data channel (if any) and sends a CTS frame back to host A. Host A then sends a RES frame to prevent its neighbors from using the selected data channel for a specified interval. Neighbors of host A and host B can update their CUL based on the CTS and RES frame they overhear on the control channel.

2.1.2. MMAC

The Multi-channel MAC (MMAC) protocol uses only one radio interface to operate on multiple data channels [35]. MMAC requires time synchronization among nodes and divides time into beacon intervals. Each beacon interval starts with a small window called ATIM window. MMAC borrows the term “ATIM” (Ad hoc Traffic Indication Message) from IEEE 802.11 Power Saving but uses ATIM for a different purpose. Each node in MMAC maintains a data structure called Preferable Channel List (PCL) that keeps track of channel usage within a node’s neighborhood. In MMAC, a channel can have three states at a node: high, medium, and low preference. High preference indicates that the node has already selected this channel for the current beacon interval and must continue to choose this channel until the next beacon interval. Medium preference means that this channel has not been taken by any neighbors within the node’s transmission range. Low preference indicates that this channel is already taken by at least one neighbor within the node’s transmission range. Nodes also maintain per-channel counters to record the usage of each channel in a beacon interval. These counters allow nodes to balance channel load.

During an ATIM window, all nodes switch to a default channel to exchange beacon and ATIM frames (the default

---

\(^1\) With the exception of Dynamic Channel Assignment (DCA), all other multi-channel MAC protocols reviewed in Section 2 do not require multiple radio interfaces.

\(^2\) In strict terminology, a package of data or control information is called a “frame” at layer 2 and a “packet” at layer 3. We use frame and packet interchangeably in this paper.
channel can also be used for data exchange outside of ATIM windows). When host A wants to send data to host B, host A inserts its PCL into an ATIM frame and sends it to host B. Host B compares host A’s PCL with its own PCL to select a channel and transmits an ATIM-ACK frame back to host A. If host A cannot select the channel specified by host B because it has already chosen another channel, it must wait until the next beacon interval. Otherwise, host A sends an ATIM-RES frame with the selected channel so that neighbors within its transmission range can update their PCL. After the ATIM window, host A and host B switch to the selected channel to exchange data.

2.1.3. SSCH

Slotted Seeded Channel Hopping (SSCH) uses only one radio interface to operate on multiple data channels [3]. SSCH requires time synchronization among nodes and divides time into slots. In SSCH, each node maintains a channel schedule containing a list of channels that the node plans to switch to in subsequent slots.

In SSCH, a node’s channel schedule is compactly represented as a current channel and a set of rules for updating channel. SSCH suggests that the set of rules is represented as a set of \((\text{channel, seed})\) pairs. Each node iterates through its set of \((\text{channel, seed})\) pairs in each slot and performs the following channel update:

\[
\text{channel}_i = (\text{channel}_i + \text{seed}_i) \mod N,
\]

where \(i\) is between 0 and 3, and \(N\) is the number of available orthogonal channels, e.g., \(N = 3\) for IEEE 802.11b and 13 for IEEE 802.11a. In SSCH, each node frequently broadcasts its channel schedule and keeps track of other nodes’ channel schedules. When host A wants to send data to host B, host A can follow host B by adopting host B’s channel schedule.

2.1.4. AMCP

The Asynchronous Multi-channel Coordination Protocol (AMCP) requires only one radio interface per node and no time synchronization among nodes to operate on \(N\) data channels [32]. However, AMCP uses a dedicated control channel on which nodes exchange control frames to negotiate channel reservation. In AMCP, each node maintains a channel table with \(N\) entries corresponding to \(N\) data channels. Each entry indicates whether a channel is available or for how long the channel is being used by other nodes within a transmission range. Further, each node maintains a variable prefer that can take values from 0 to \(N\). If nonzero, prefer contains the index to a data channel that the node prefers. If zero, prefer indicates that the node has no preference.

When host A wants to send a data packet to host B, it first selects a data channel to compete for. If host A’s variable prefer is nonzero and this channel is available, host A will select this channel. Otherwise, host A randomly selects one of its available data channels. Host A inserts the index to the selected data channel into an RTS frame and sends it to host B. When host B receives the RTS frame, host B can send a Confirming CTS with the index to the selected data channel if that channel is available for host B. Otherwise, host B sends a Rejecting CTS with a list of its available data channels. Upon receiving a Confirming CTS, host A switches to the selected channel and transmits a data packet to host B. Otherwise, host A selects a channel available for both host A and host B and sends a new RTS frame to host B. Other hosts within the transmission range of host A and host B can overhear the RTS and CTS frames and update their channel tables for channel availability accordingly.

2.2. Multi-channel MAC for Wireless Sensor Networks (WSNs)

Kim et al. [19] proposed Y-MAC, a TDMA-based multi-channel MAC for WSNs. Time synchronization in Y-MAC is achieved by a sink that periodically broadcasts time reference for other nodes. Y-MAC divides time into fixed-length frames, each consisting of a broadcast period and a unicast period. Nodes in Y-MAC initially operate on a predefined base channel. When nodes detect overload on the base channel, they hop to other channels to exchange data. Each node determines its hopping pattern locally as follows. Each node periodically broadcasts its channel selection. Each node also collects the channel selections from its neighbors and uses this information to choose the least used channel for the next cycle. Y-MAC is implemented on a sensor node platform.

Le et al. [22] proposed a practical multi-channel MAC for WSNs that assigns a home channel to each node. All nodes initially operate on the same channel. When overload ensues on this channel, nodes begin to migrate to other channels to distribute the load across multiple channels. When a node needs to send data to another node on a different channel, it switches to that channel. This multi-channel MAC is implemented on a sensor node platform.

So et al. [33] demonstrated that synchronization for multi-channel MAC protocols is a nontrivial problem. They focused on the design and implementation of practical synchronization techniques rather than the implementation of multi-channel MAC protocols. They investigated and implemented practical synchronization techniques specifically for multi-channel MAC protocols on sensor node platforms running TinyOS [28].

While practical multi-channel MAC for WSNs bears resemblance to our work, there are several important differences. First, WSN implementations are based on sensor node platforms equipped with an IEEE 802.15.4 radio interface. These platforms have different characteristics than off-the-shelf IEEE 802.11 hardware. Second, since WSNs do not support TCP/IP applications, the impact of multi-channel MAC on TCP/IP applications has not been investigated. Third, while WSN implementations have low-level control over the radio interface, access to off-the-shelf IEEE 802.11 hardware is restricted to an abstract interface. This has important implications for the implementation. Fourth, WSN implementation and ours address different application scenarios. In WSNs, nodes usually form a tree topology to forward data from multiple sources to a sink. On the other hand, our work targets mesh networks where connection patterns between nodes can be arbitrary. This subtle difference has important impact on the level of multiplexing at the MAC layer.
2.3. Practical implementations for 802.11 multi-channel networks

Existing research in multi-channel MAC for IEEE 802.11 has been mainly of theoretical interest. We are aware of the following practical work.

Chereddi et al. [8] implemented a channel abstraction module for 802.11-based multi-channel multi-interface networks. The module implements supports for frequent channel switching and allows to group multiple wireless interfaces into a single virtual interface. This work focused on channel abstraction to support routing solutions in multi-channel multi-interface networks [20,21]. They did not implement and investigate the effects of multi-channel MAC protocols.

Raniwala and Chiueh implemented a distributed algorithm for channel assignment in 802.11-based multi-channel multi-interface networks [29]. They focused on the aspects of channel assignment and routing to adapt to dynamic traffic loads and achieve load balancing where interfaces rarely switch channels. They did not consider practical implementations of multi-channel MAC protocols.

In summary, practical work in multi-channel wireless networks focused on evaluation of routing protocols [8,13] and existing work on multi-channel MAC has remained pure academic interest. The lack of practical implementation and performance evaluation of multi-channel MAC for IEEE 802.11 is the main motivation for our work.

3. Implementation

In this section, we first describe our system architecture and protocol utilities that facilitate the implementation of multi-channel MAC protocols. We then discuss our implementation of AMCP, MMAC, and SSCH.

3.1. System architecture

Our implementation uses off-the-shelf IEEE 802.11 hardware with a chipset manufactured by Atheros Communications [10]. Fig. 1 depicts the system architecture of our implementation. The Atheros chipset sits on a wireless NIC and is connected to a host system via a PCI bus. Atheros Communications does not provide any public documents for its chipsets. However, the MadWifi project uses a library of binary code called "hardware abstraction layer" (HAL) that allows the host’s operating system to interact with the Atheros chipset [23].

The MadWifi Linux driver consists of four main components: hardware layer abstraction, Atheros driver, 802.11 implementation, and rate adaptation. The hardware layer abstraction hides hardware details and provides an interface to the Atheros chipsets. The Atheros driver deals with low-level hardware details on the host system, including interrupts and interacts with the PCI bus to get packets from and to NIC. The 802.11 implementation handles the high-level functionality of IEEE 802.11 such as scanning, authentication, association, encryption and decryption (the low-level functionality of IEEE 802.11 such as performing CSMA/CA and generating RTS, CTS and ACK frames is implemented in the Atheros chipsets to achieve fine and accurate timing). Rate adaptation adjusts the bit rate of NIC based on measured packet loss rate [4].

Our implementation for multi-channel MAC protocols is integrated into the MadWifi driver and has two main components: protocol utilities and multi-channel MAC protocols. The protocol utilities interact with the rest of the MadWifi driver and create a framework for implementing multi-channel MAC protocols. This allows protocol implementers to concentrate on protocol development without dealing with much details of the MadWifi driver.

3.2. Protocol utilities

Our protocol utilities implement the following features: ioctl interactions, channel switching, packet buffering, per-neighbor packet queue maintenance, and control frame generation. Two additional features that could be implemented in our protocol utilities are time synchronization and retransmission of control frames. However, these two features are optionally implemented by a multi-channel MAC protocol to address its specific needs. The implementation of our protocol utilities is discussed below.

3.2.1. ioctl interactions

Our protocol utilities receive ioctl commands from the user space and load/unload the implementation of
multi-channel MAC protocols at run time. They also parse the ioctl parameters and pass them on to the multi-channel MAC protocols.

3.2.2. Channel switching

A crucial requirement of multi-channel MAC protocols is the ability to switch channels. However, the Atheros chips do not offer this functionality directly via the hardware layer abstraction. We implement this feature via the MadWifi driver in the following manner. First, DMA engines for the transmit and receive queues on the Atheros chips are stopped. Next, all pending packets in the transmit queue are purged. Finally, a reset command with the new channel as a parameter is issued to the chips.

3.2.3. Packet buffering

Frequent channel switches required by multi-channel MAC protocols can cause significant packet losses. This is because all pending packets in the transmit and driver queues are dropped due to a reset of the Atheros chipset during a channel switch. We minimize packet losses by adding a packet buffer between the Atheros driver and the implementation of high-level 802.11 functionality. Our implementation counts the number of pending packets in the transmit queue and passes down a new packet to the Atheros driver every time it receives an interrupt signal notifying the completion of a packet transmission. The maximum number of packets in the transmit queue is limited to four.

3.2.4. Per-neighbor queues maintenance

In multi-channel environments, a neighboring node can be temporarily unreachable when it is on a different channel. Our implementation maintains per-neighbor queues to avoid head-of-line blocking when the recipients of some packets in the transmit queue are on a different channel. We implement a simple variant of stochastic and round-robin scheduling to achieve a level of fairness among flows destined for different nodes.

3.2.5. Control frame generation

As reviewed in Section 2, AMCP, MMAC, and SSCH require additional control frames undefined in the IEEE 802.11 standard. Our protocol utilities assist multi-channel MAC protocols in generating control frames by performing buffer allocation, protocol header processing, and control frame delivery between the Atheros driver and multi-channel MAC. In general, control frames are broadcast because nodes need to be able to overhear their neighbors’ channel schedules in SSCH or their neighbors’ control frames for channel reservation in MMAC and AMCP.

3.3. Implementation of MMAC

Our MMAC implementation is guided by technical details found in [35]. An important task in implementing MMAC that is not mentioned in [35] is how to achieve time synchronization among nodes. Time synchronization in ad hoc networks is a challenging problem and has been studied extensively [12,18,31,33]. The Timing Synchronization Function (TSF) in IEEE 802.11 achieves time synchronization in ad hoc mode by having nodes adjust their local time to the fastest running clock in the following manner. All nodes set a timer for transmitting a beacon frame for time synchronization. The timer interval is the sum of a constant interval called the beacon interval and a small random delay. The first node that has its timer expired broadcasts a beacon frame that contains a time stamp as a reference time. Other nodes update their local clocks if their values are later than the reference time. All nodes then reset their timer for the next beacon transmission. Huang and Lai discovered that time synchronization based on broadcast reference time tends to render the node with the fastest clock out of sync because clocks in TSF only move forward and never backward in [18]. They proposed an adaptive mechanism for each node to adjust its random delay before it broadcasts a beacon. A node’s adjusted random delay depends on how fast its clock is in comparison to that of other nodes.

Since clock synchronization requires accurate timing, beacon generation is implemented in the hardware of the wireless NICs. The MadWifi driver and the host’s operating system remain oblivious of the beacon generating task. Our MMAC implementation aligns the ATIM window with the beacon interval and relies on the time synchronization provided by the wireless NICs as follows. When a wireless NIC receives a beacon frame, it updates its local clock as discussed above and passes the frame to the MadWifi driver for further processing. Upon receiving a beacon frame, the MadWifi driver triggers MMAC to start a new ATIM interval. Since beacon generation is handled by the hardware of the wireless NIC, this mechanism works for all nodes except for the node that generates the beacon frame. We solve this problem by having nodes that receive a beacon frame wait for a small random delay before broadcasting an MMAC sync frame. Upon receiving an MMAC sync frame, nodes that have not recently received a beacon frame start a new ATIM interval; other nodes refrain from sending another MMAC sync frame. Since beacon and sync frames can be lost in the wireless medium, our MMAC implementation also uses a kernel timer as an additional trigger to start a new ATIM window. This timer is reset upon the arrival of a beacon or a sync frame.

To evaluate the synchronization accuracy of our MMAC implementation, we perform an experiment in which 10 nodes broadcast a frame at the beginning of their ATIM windows. Nodes measure the time difference between their own and other nodes’ ATIM windows. All nodes are within the transmission range of each other. We collect data for 10 million samples. Our data shows that more than 95% of the samples have a time difference less than 2 ms and over 98.5% of the samples have a time difference less than 10 ms.

Our implementation divides a beacon interval into three phases: ATIM, data exchange, and switch back. During an ATIM phase, each node chooses a neighbor among all nodes that it has pending packets for (we use a variant of stochastic and round-robin scheduling policy for neighbor selection). Each node sends an ATIM frame with its channel list to its selected neighbor. Upon receiving the ATIM frame, the neighbor chooses a channel as discussed.
in Section 2) and sends back an ATIM-ACK frame. The sender acknowledges the ATIM-ACK frame with an ATIM-RES frame. All nodes wait until the ATIM phase is over and then switch to their selected channels to exchange data. When the data exchange phase is over, all nodes enter the switch back phase where they tune their wireless NIC to the default channel and wait for a new ATIM phase. The switch back phase does not exist in the original MMAC protocol [35] but is introduced in our implementation due to considerable switching delay as will be discussed in Section 4.

3.3.1. Improvement for MMAC

As discussed in Section 3.2, ATIM frames have to be broadcast so that nodes can overhear their neighbors’ control frames for channel reservation. Unlike unicast frames, broadcast frames are not acknowledged. Our MMAC implementation uses timers and retransmissions to increase the reliability of ATIM exchanges as follows. Each ATIM control frame has an additional information element that contains the MAC address of the intended recipient that is supposed to acknowledge the frame. If the sender does not receive an acknowledgment within a certain time interval, it retransmits the control frame and resets its retransmission timer.

Even with the obtained synchronization accuracy presented in the previous subsection, a sender can still be slightly faster than its respective receiver. In this case, the sender switches to a data channel before the receiver and starts transmitting data packets after the ATIM phase. However, all these packets are lost since the receiver is not yet on the data channel to receive them. In IEEE 802.11, a sender retransmits an unacknowledged unicast frame for a few times. If the receiver arrives at the new channel after the retransmissions, the data frame is lost. We remedy this “late receiver problem” by having the sender send a sync request and wait for a sync response from the receiver. If the sender does not receive a sync response within a certain time interval, it retransmits its sync request. The sender only starts its packet transmission after the handshake is established.

We also address the “early leaving receiver problem” which is the counterpart of the “late receiver problem”. This problem occurs when the receiver’s clock runs faster than the sender’s clock and the receiver returns to the control channel while the sender is still transmitting data packets on the data channel. We address this problem by having the receiver send a terminate request to the sender before switching back to the control channel. Since a terminate request frame can be lost on the wireless channel, we increase its reliability by having the receiver retransmit this frame three times. The retransmission interval is set to 1 ms. When receiving a terminate request, the sender responds with a terminate confirm and returns to the control channel. When receiving a terminate confirm, the receiver also returns to the control channel.

3.4. Implementation of SSCH

Our SSCH implementation follows technical details found in Bahl et al.’s paper [3]. In SSCH, each node randomly selects its channel schedule as a set of 4 (channel, seed) pairs. Each node cycles through all its (channel, seed) pairs and determines its channel for each time slot as described in Eq. (1). Each node frequently broadcasts its channel schedule and an offset that informs other nodes how far it has progressed in its current cycle. Like MMAC, SSCH also requires that nodes are synchronized and align their cycles with each other. We implemented a similar time synchronization function for SSCH like for MMAC.

3.4.1. Improvement for SSCH

In SSCH, a sender can reach a receiver by adopting the receiver’s (channel, seed) pairs and following the receiver’s channel hopping pattern. Changing channel schedule is implicit in SSCH. The sender does not inform the receiver that it wants to follow the receiver’s channel hopping pattern in order to send packet to the receiver. Further, the receiver does not signify the sender when it leaves its current channel hopping pattern and adopts a new one. A problematic situation can occur when host A wants to send packets to host B while host B wants to send packets to host C. In this case, host B takes host C’s channel schedule and no longer follows the channel hopping pattern that host A has just adopted. Thus, all packets that host A sends to host B are lost. SSCH alleviates this problem by having the sender count the number of acknowledgments received for its transmitted packets at the end of each time slot. If this number is below a certain threshold, the sender marks the receiver’s channel schedule as unknown and no longer follows the receiver’s channel hopping pattern. We find in our testing that this solution is not effective in limiting packet losses. The reason is that this solution only reacts after the sender has lost an entire slot worth of packets when the receiver changes its channel schedule.

We implement a simple but effective solution for limiting packet losses due to the aforementioned “missing receiver problem” as follows. At the beginning of each slot, the sender sends a SSCH sync request to the receiver. The sender sets a timer and keeps retransmitting its SSCH sync request every time the timer expires until one of the two following events occurs: (1) if the sender receives an SSCH sync response from the receiver, the sender deletes the timer and starts sending data packets until the end of the current slot, and (2) if the number of retransmissions of SSCH sync requests reaches a predefined threshold, the sender declares the receiver’s channel schedule unknown and stops sending packets to the receiver until it receives an update for the receiver’s channel schedule. We set this threshold to 5 in our experiments. Like for MMAC, we also implement our remedy for the “late receiver problem” and the “early leaving receiver problem” for SSCH.

3.5. Implementation of AMCP

Our AMCP implementation is guided by technical details found in Shi et al.’s paper [32]. In AMCP, nodes use a dedicated control channel to negotiate a data channel and then switch to the data channel to exchange data. AMCP does not require time synchronization among nodes so that implementing AMCP is somewhat easier than MMAC and SSCH. Nevertheless, we face a number of practical problems in implementing AMCP.
3.5.1. Improvement for AMCP

A practical problem with channel switching on current off-the-shelf IEEE 802.11 hardware is that if a packet is passed down to the hardware immediately before a channel switch, that packet may be lost. The reason is that the HAL command for channel switching essentially results in a reset of the Atheros chipset. In particular, a problematic situation arises when a receiver has to respond to a sender’s RTS frame with a CTS frame and then immediately switches to the reserved data channel. In this case, the CTS frame will be lost. We address this problem by having the receiver defer switching channels for a small amount of time so that the wireless NIC can send out the CTS frame before it is reset. Since the receiver defers switching to the reserved data channel, it is possible that the sender arrives at the data channel upon receiving the receiver’s confirming CTS before the receiver. This “late receiver problem” can cause considerable packet losses because the sender already starts transmitting data packets on the new channel before the receiver arrives. We remedy this problem by having the sender delays sending packets for the same amount as the receiver waits on the control channel before switching to the reserved data channel. We also implement our remedy for the “early leaving receiver problem” for AMCP.

AMCP assumes that the channel switching delay is relatively low (224 μs) so that nodes can exchange control frames and perform channel switching on a per-packet basis. While channel switching delay of the hardware alone might be low, the total switching delay for both hardware and software on current off-the-shelf IEEE 802.11 NIC is rather high (this issue will be elaborated in Section 4.1). We amortize the overhead of channel switching delay by having nodes run AMCP on a per-interval basis.

As discussed in Section 2, AMCP control frames are broadcast on the control channel so that each node is aware of channel reservations by other nodes within its transmission range. Since broadcast frames are transmitted unreliably (without acknowledgment), we use timers and retransmissions to increase the reliability of exchanges of AMCP control frames. The sender of an AMCP control frame inserts the MAC address of its intended recipient into an additional field of the control frame and sets a timer. If the timer expires, the sender retransmits the control frame and resets the timer. If the sender receives an acknowledgment for the control frame, it deletes the timer.

We observe another problem for AMCP when multiple nodes are involved. A typical scenario of this problem is depicted in Fig. 2. In this scenario, A and C both attempt to send data packets to B. A and B successfully establish an RTS/CTS handshake on the control channel and switch to another channel to exchange data. After an interval for data exchange, A and B switch back to the control channel. They pause for a short interval, establish an RTS/CTS handshake, and switch to a data channel again. In this scenario, C keeps retransmitting RTS but cannot reach A because A spends most of its time on a data channel. This “lockout problem” can be alleviated as follows: After returning from a data channel, each node pauses for an interval that is randomly chosen between 0 and max RTS/CTS delay before attempting to establish RTS/CTS handshake (max RTS/CTS delay is considerably larger than the retransmission interval for RTS).

3.6. Handling broadcast frames

A common problem for all multi-channel MAC protocols is that since nodes are distributed on different channels, a broadcast data frame transmitted by a node may not reach all its neighbors. Broadcast data frames are actually quite important because they are used by ARP and routing messages. We alleviate this problem by implementing a cache for broadcast data frames. Every time a node performs a channel switch, it retransmits a few of its cached broadcast data frames. Cached broadcast data frames are discarded after a number of retransmissions. We remark that the effectiveness of this solution differs for different multi-channel MAC protocols. For example, it depends on whether all nodes have a control channel (MMAC and AMCP) and whether they operate synchronously (MMAC and SSCH).

4. Experimental methodology

In this section, we explain our experimental methodology. We first present measurement results that quantify the overhead of implementation and the channel switching latency. We describe our wireless testbed and discuss how we generate realistic traffic. Finally, we explain our method for calibrating our traffic generation. Results of the calibration is used to tune the network load in our actual experiments presented in Section 5.

4.1. Quantifying overhead

4.1.1. Overhead of protocol utilities

Since the protocol utilities described in Section 3.2 add additional code to the MadWifi driver, we need to verify that they do not incur any non-negligible performance overhead. This is important because the packet buffer introduced by the protocol utilities could interfere with the operational pipeline between the MadWifi driver and the wireless NICs, and have adverse effects on the performance. We conduct experiments where a sender uses Iperf to transmit UDP packets to a receiver at a transmission rate varied between 0 and 54 Mbps. We observe that the CPU usage increases by 5% with the modified MadWifi driver. However, there is no difference between throughput results obtained by the unmodified and modified MadWifi driver.

4.1.2. Channel switching delay

We perform three experiments to measure the channel switching delay of current off-the-shelf wireless NICs.

Please cite this article in press as: L. Le, I. Rhee, Implementation and experimental evaluation of multi-channel MAC protocols for 802.11 networks, Ad Hoc Netw. (2010), doi:10.1016/j.adhoc.2009.12.004
These experiments measure the latency between a channel switch and the instant when a wireless NIC can either transmit or receive a packet.

In the first experiment, we instrument code into MadWifi’s function that is used to switch the channel for a wireless NIC. We record the timestamps when the function is entered and exited to estimate the time it takes a wireless NIC to tune to a new channel. Since the wireless NIC might not be ready to transmit and receive packets immediately after a channel switch, we also conduct two further experiments.

In the second experiment, we equip a sender with one wireless NIC and a receiver with two wireless Nics. The sender’s wireless NIC is first tuned to the channel of the receiver’s first wireless NIC. The sender sends a packet to the receiver and immediately tunes its wireless NIC to the channel of the receiver’s second wireless NIC. After the channel switch, the sender sends a second packet to the receiver on the new channel. The elapsed time between two packets arriving at the receiver on its two different wireless NICs measure the channel switching delay on the transmit side.

In the third experiment, we equip a sender with two wireless NICs and a receiver with one wireless NIC. The sender continuously sends packets to the receiver on both NICs. The external antenna of transmitting NICs are placed 1 m apart of each other to reduce interference between them [1,29]. We obtain additional shielding between the transmitting NICs by wrapping them in aluminum foil [30]. The receiver first tunes its wireless NIC to the channel of the sender’s first NIC and then switches to the channel of the sender’s second NIC. The time difference between two packets arriving at the receiver on different channels measure the channel switching delay on the receive side.

We repeat each of the three experiments a million times and find that they obtain similar results. The experiments indicate that the average values of switching latency on both transmit and receive side are 4.50 ms. In general, we find that the switching latency on the transmit and receive side obtains similar values. Over 92% of our 1 million samples achieve a switching delay between 4 and 5 ms. About 6% of our samples obtain a switching delay between 5 and 6 ms. This result shows a stark contrast between theoretical assumptions and practical limitations (the switching delay was assumed to be 80 μs by SSCH [3] and 224 μs by both MMAC [35] and AMCP [32]). We note that while channel switching delay of a RF transceiver alone can be reduced to 40–80 μs [17,25], the total switching delay for the entire system (hardware and software together) of current off-the-shelf wireless NIC is significantly higher.

4.2. Testbed

We set up a wireless testbed consisting of 12 nodes that are placed on the same floor. Our nodes are commodity PCs that are equipped with wireless NICs based on Atheros chipsets. The wireless cards support IEEE 802.11 a/b/g. Our nodes run Linux kernel 2.6.23. AMCP, MMAC, and SSCH are implemented as extensions of the MadWifi release 0.9.4. We fix the transmit power and transmission rate of the wireless NICs at 17 dBm and 12 Mbps. The rationale behind this is that we want to focus on the performance evaluation of multi-channel MAC protocols and eliminate any potential side effects of rate and power adaptation of the wireless NICs. Wireless NICs are configured to run in ad hoc mode. In order to prevent network partitioning [5], we fix the BSSID at all nodes to a value that is computed as a function of the ESSID. The wireless NICs are configured to operate in the 5-GHz frequency band and provide 19 orthogonal channels. We choose the 5-GHz frequency band over the 2.4-GHz frequency band because it is currently significantly less occupied than the heavily used 2.4-GHz band. Indeed, the 5-GHz frequency band is pristine in the building of our wireless testbed. Thus, our experiments do not suffer interference with currently deployed 802.11 wireless networks.

4.3. Traffic generation

Generation of realistic traffic plays an important role in the performance evaluation of network protocols. In general, this is done in a three-step process: (1) collect data packets generated by users in operational networks, (2) derive a traffic model from captured packets, and (3) synthesize traffic based on the derived traffic model.

For the first step of realistic traffic generation, we obtain a large collection (tens of gigabytes) of packet traces from the archive of wireless data at Dartmouth [14]. These packet traces that we obtain were generated by wireless users at the campus of Dartmouth College between 2003 and 2004 [15].

We are aware of three software packages that can do the last two steps of generating realistic traffic: tmix [16], Swing [37], and Harpoon [36]. We choose Harpoon because of its availability. Given a packet trace, Harpoon derives a number of empirical distributions such as flow sizes and inter-connection time. These distributions are shown in Fig. 3A and B. Synthetic traffic can then be generated based on these distributions.

4.4. Calibration

We calibrate the load of synthetic traffic generated by Harpoon by varying the number of active sessions. In our calibration, the wireless NICs of two PCs are configured to 54 Mbps (the highest data rate). The two PCs are placed approximately 4 ft apart. One PC runs as a server. The other PC is used as a client. A third PC placed between the two PCs is used as a packet sniffer to measure the load on the wireless link.

The experiments run for an hour but results of the first 20 min are discarded to eliminate any transient effects at the beginning of the experiments. We repeat each experiment 10 times and average the obtained results. Fig. 3C shows a linear relationship between the number of active sessions and the synthetic traffic load. Results of the calibration experiments also ascertain that our PCs are not the bottleneck in our actual experiments because they can generate a data throughput higher than 12 Mbps (the data rate used by the wireless NICs in experiments presented in Section 5).
5. Experimental results

We report results of our performance evaluation in this section. The parameters used for AMCP, MMAC, and SSCH are shown in Table 1. Most of these parameters have the default values that are recommended by the inventors of AMCP, MMAC, and SSCH. There are two notable exceptions where we deviate from the inventors’ recommended parameters: (1) a time slot of 10 ms for SSCH is too short due to the high channel switching delay and (2) channel switching on a per-packet basis for AMCP is not feasible on current off-the-shelf IEEE 802.11 hardware.

We use Harpoon to evaluate the performance of AMCP, MMAC, and SSCH. We use results from our calibration experiments in Section 4.4 to tune the number of Harpoon’s active sessions so that the generated traffic load is 3.6, 7.2, and 10.8 Mbps on an uncongested wireless link. Since the transmission rate of the wireless NICs is fixed at 12 Mbps, the chosen Harpoon’s active sessions represent light, medium, and heavy load in our experiments. For shorthand notations, we term these loads 30%, 60%, and 90%. We note that these terms are only used as shorthand notations. When the wireless link is congested, the actual load in our experiments will be different due to two reasons: collisions at the MAC layer and TCP’s closed-loop behavior (Harpoon uses TCP as the underlying transport protocol).

Two kinds of network topology are used in our experiments: one-hop and multi-hop. In experiments with one-hop topology, a Harpoon client is configured to exchange data only with Harpoon servers that are within its one-hop neighborhood. This restriction is removed in experiments with multi-hop topology. For routing data in multi-hop topology, we use olsrd [27], an implementation of the Optimized Link State Routing Protocol (OLSR) [9].

We conduct experiments for two different deployment scenarios: 802.11b and 802.11a. In an 802.11b deployment scenario, AMCP, MMAC, and SSCH use three orthogonal channels chosen randomly from the 19 available channels. In an 802.11a deployment scenario, AMCP, MMAC, and SSCH operate on 13 orthogonal channels.

The key performance metrics in our performance evaluation are the end-to-end response times for data exchange between Harpoon clients and servers and the number of completed data exchanges (Tables 2 and 3). We run our experiments for an hour but discard the first 20 min to eliminate any transient effects at the beginning of the experiments. We repeat each experiment 10 times and take an average of the results.

5.1. Quantifying our improvements for AMCP, MMAC, and SSCH

We conduct experiments for AMCP, MMAC, and SSCH with and without our improvements discussed in Section 3. We depict our results for one-hop topology in an 802.11b deployment scenario in Figs. 4–6. To explore the
parameter space, we also present experimental results when AMCP, MMAC, and SSCH operate with a 200-ms data interval. Our improvements for AMCP, MMAC, and SSCH appear effective and outperform their original counterpart. The performance gain is remarkable for small flows that can complete within a few hundreds of milliseconds. For example, our improvements reduce the median completion time from 294 to 182 ms for AMCP, from 351 to 176 ms for MMAC, and from 185 to 124 ms for SSCH at 60% load. Given the inferiority of the original multi-channel MACs, we only show results of improved AMCP, MMAC, and SSCH in the remaining of this paper. Our results also indicate that a data interval of 100 ms appears to be a good trade-off. While a long data interval ameliorates the protocol and switching overhead, this gain can only be realized for large-size flows. On the other hand, if multiple senders want to send data to a receiver at the same time, they will have to wait longer when a long data interval is used.3

5.2. Comparison of Multi-channel MAC protocols in one-hop topology

We show our results for one-hop 802.11b experiments in Fig. 7. For comparison purpose, we also show results of the standard 802.11 MAC. The median response times for 802.11 MAC, AMCP, MMAC, and SSCH are 85, 91, 150, and 88 ms at 30% load. The number of completed data exchanges for these protocols are 29,013, 26,904, 26,584, and 27619. The results are quite surprising: all multi-channel MAC protocols outperform the standard 802.11 MAC protocol at this load. The reason for this unexpected phenomenon is the channel switching overhead of AMCP, MMAC, and SSCH. At this low load, SSCH slightly outperforms MMAC and AMCP. MMAC obtains the worst performance at this load. We conjecture MMAC's poor performance as follows. When data packets are passed down to the MAC layer, a sender in SSCH and AMCP can attempt to establish a handshake with a receiver immediately. On the other hand, a sender in MMAC has to wait until a new ATIM phase starts. This behavior of MMAC increases end-to-end response times.

At 60% load, 802.11 MAC, AMCP, MMAC, and SSCH yield a median response time of 318, 182, 176, and 124 ms. The number of completed exchanges obtained by these protocols are 47,601, 51,730, 52,189, and 53,476. We also observe in Fig. 7B that SSCH gives the best overall performance at 60% load. AMCP delivers the worst performance among all multi-channel MACs at this load due to the fact that it uses one of the three available channels as a control channel and is hence less effective than the other multi-channel MACs. We note in Fig. 7B that while AMCP, MMAC, and SSCH outperform 802.11 MAC for approximately 80% of the flows, they actually deliver worse performance than 802.11 MAC for the remaining 20% flows. We suspect that these flows are large and take long time to complete. For these flows, the channel

---

3 We also perform experiments with a 50-ms data interval. However, these results are inferior to those obtained with a 100-ms data interval and are not reported here.
938 switches cause a considerable performance penalty for AMCP, MMAC, and SSCH. The results at 90% load are similar to those at 60% load but even more pronounced. The median response times for 802.11 MAC, AMCP, MMAC, and SSCH are 426, 186, 183, and 129 ms at 90% load. The number of completed data exchanges for these protocols are 54,798, 66,371, 67,538, and 68,095. AMCP increases its performance in comparison to the 802.11b scenario since its overhead of a control channel becomes negligible in the 802.11a scenario. SSCH performs best and AMCP slightly outperforms MMAC in this scenario. As the load increases from 60% to 90%, we observe that time synchronization among nodes becomes less accurate because it relies on beacons that are susceptible to collisions at a high load (broadcast beacons cannot be protected by ACK). This phenomenon has an adverse impact on MMAC and SSCH. Nevertheless, the results obtained at 90% load exhibit a similar trend as those obtained at 60% load. 802.11 MAC, AMCP, MMAC, and SSCH

Fig. 6. Quantifying our improvement for SSCH.

Fig. 7. One-hop experiments in 802.11b scenario.

Fig. 8. One-hop experiments in 802.11a scenario.
obtain a median response time of 426, 138, 178, and 126 ms at 90% load. The number of completed data exchanges obtained by these protocols are 54,798, 71,683, 70,269, and 72,091.

5.3. Comparison of multi-channel MAC protocols in multi-hop topology

We show our results for multi-hop 802.11b experiments in Fig. 9. The performance of all MAC protocols deteriorates markedly in comparison to results obtained with one-hop topology. There are three main reasons for this phenomenon. (1) Since a flow can now traverse multiple hops, it takes longer to complete a request/response exchange. (2) Due to the half-duplex nature of IEEE 802.11 NICs, end-to-end response times for multi-hop flows increase. (3) The actual channel load increases in comparison to one-hop experiments because a data packet has to be transmitted at least once by each node that it traverses.

At 30% load, AMCP, MMAC, and SSCH underperform the standard 802.11 MAC protocol like in the one-hop scenarios. At 60% load, the median response times for 802.11 MAC, AMCP, MMAC, and SSCH are 431, 226, 364, and 253 ms. In terms of response times, the multi-channel MACs outperform 802.11 MAC for approximately 60% of the flows but deliver worse performance than 802.11 MAC for the remaining flows. The number of completed data exchanges for 802.11 MAC, AMCP, MMAC, and SSCH at 60% load are 30,169, 32,086, 30,794, and 31,896. At this load, all multi-channel MACs do not provide a clear advantage over 802.11 MAC in multi-hop 802.11b scenario.

There are different reasons for this phenomenon for each multi-channel MAC. MMAC causes nodes to align their handshakes with the ATIM phase and induces additional delay in the multi-hop scenario. AMCP incurs the overhead of a control channel, which becomes a major performance bottleneck in multi-hop 802.11b scenarios. SSCH relies on time synchronization among nodes which becomes less accurate in a multi-hop topology. As load increases to 90%, the median response times for 802.11 MAC, AMCP, MMAC, and SSCH increase to 1027, 566, 586, and 658 ms. The results indicate that the network is saturated for 802.11 MAC at this load. The number of completed data exchanges for the MAC protocols are 34,518, 42,357, 41,365, and 40,725 at 90% load. All multi-channel MACs outperform 802.11 MAC at this load.

We briefly discuss the difficulty of synchronization in multi-hop topology. Consider a chain topology A–B–C–D where each node can only reach its immediate neighbor. If A follows B’s clock and D follows C’s clock, then synchronization can be achieved in the entire network. However, if B follows A’s clock or C follows D’s clock, then B and C are not synchronized. Multi-hop synchronization is exacerbated under heavy load because a node can miss its neighbors’ beacons more frequently.

We show our results for multi-hop 802.11a experiments in Fig. 10. At 30% load, AMCP, MMAC, and SSCH obtain similar results like in the multi-hop 802.11b scenario because the additional channels do not provide any benefits for this low load. At both 60% and 90% loads, AMCP, MMAC, and SSCH obtain considerable performance improvement as compared to the multi-hop 802.11b...
scenario thanks to the additional channels. For example, AMCP, MMAC, and SSCH obtain a median response time of 179, 284, and 216 ms at 90% load in the 802.11a multi-hop scenario (these values are 566, 580, and 658 ms for AMCP, MMAC, and SSCH in the multi-hop 802.11b scenario). We observe in Fig. 10 that AMCP provides the best performance at these loads since its overhead of a control channel now becomes negligible. In comparison to one-hop topology, SSCH loses its advantage because time synchronization becomes less accurate in multi-hop topology. In particular, SSCH underperforms AMCP and MMAC at 90% load.

SSCH represents an interesting point in the design space of multi-channel MAC that is worth a discussion. SSCH has a lower overhead than AMCP and MMAC because it does not require a control channel (AMCP) or a handshake phase (MMAC). Further, SSCH requires only a channel switch per data interval. On the other hand, AMCP and MMAC incur two channel switches for a data interval. However, SSCH’s low overhead demands a fine-grained synchronization among nodes because SSCH does not have a fallback mechanism such as a control channel (AMCP) or a handshake phase (MMAC). For this reason, SSCH outperforms AMCP and MMAC in one-hop scenarios or at low loads where synchronization can be attained easily. However, SSCH fal ters at high loads in multi-hop scenarios where fine-grained synchronization is difficult to achieve.

6. Lessons learned

Implementing AMCP, MMAC, and SSCH on off-the-shelf IEEE 802.11 hardware is an ambitious endeavor. The implementation of each protocol takes an experienced kernel developer several months to complete. We discuss our learned lessons and provide recommendations for further research and development.

First, common techniques for clock synchronization rely on the exchange of timestamps or broadcast references [12,18,31]. However, because broadcast frames are not protected by RTS/CTS and are susceptible to collisions, the transmission of timestamps or broadcast references is unreliable under heavy load. Further, it is very difficult to achieve synchronization accuracy finer than 1 ms using standard hardware and software implementations. We also note that time synchronization incurs additional implementation complexity and prolongs the implementation process. Thus, unless time synchronization is available via an external signal like GPS, a multi-channel MAC that does not rely on clock synchronization appears attractive.

Second, since channel switching delay on off-the-shelf IEEE 802.11 hardware is rather large (several milliseconds) [7,29], it is important to design a multi-channel MAC protocol that avoids frequent channel switching. Otherwise, the performance gain realized by multi-channel MAC can be offset by the performance penalty incurred by frequent channel switching. On the other hand, it is desirable if hardware vendors can provide IEEE 802.11 hardware with low switching delay.

Third, due to imperfect synchronization and possibly different channel switching delay, it is possible that a sender arrives at a channel earlier than a receiver. For this reason, it is recommended that the sender establishes a handshake with the receiver before starting packet transmission. Similarly, when the receiver decides to leave a channel, it is recommended that it informs the sender.

Fourth, when devising a multi-channel MAC, a protocol designer should consider that a node can be simultaneously involved in multiple data exchanges with other nodes. Special attention is required to avoid “lockout” and to provide multiplexing between different flows at a node. This mindset is crucial in a multi-hop topology.

Fifth, a multi-channel MAC generally requires that a node can overhear its neighbors’ channel selection. For this reason, control frames are usually transmitted as broadcast frames. However, since broadcast frames are not acknowledged and also not protected by RTS/CTS, a protocol design should be as simple as possible to avoid undesired consequences due to possible loss of control frames. Further, additional reliable mechanisms need to be provided for the exchange of control frames.

7. Summary and conclusions

Multi-channel MAC protocols have been an active research area in recent years but most of these protocols remain as pure academic interest. In this paper, we report an implementation study and performance evaluation of three prominent multi-channel MAC protocols: AMCP, MMAC, and SSCH. Our main contributions are as follows. First, we present an implementation AMCP, MMAC, and SSCH on off-the-shelf IEEE 802.11 hardware. Our work is the first practical implementation of multi-channel MAC for IEEE 802.11. We discuss lessons learned from our endeavor. Second, we identify a number of problems of these multi-channel MAC protocols that were not considered in the original design of these protocols. We present our techniques to remedy these problems. Third, we explore practical impacts of these multi-channel MAC protocols and present results of an experimental performance evaluation in our wireless testbed. The most important findings of our performance evaluation are: (1) all multi-channel MAC protocols deliver worse performance than the original 802.11 MAC at low load due to their channel switching overhead, (2) all multi-channel MAC protocols outperform the original 802.11 MAC at medium and high load, (3) AMCP gives the worst performance among all multi-channel MAC protocols in one-hop and multi-hop 802.11b scenario due to the overhead of its control channel but obtains the best performance in multi-hop 802.11a scenario because of its independence on time synchronization, and (4) SSCH outperforms AMCP and MMAC in one-hop scenarios or at low loads but fal ters at high loads in multi-hop scenarios due to its strong dependence on a fine-grained synchronization among nodes.

Acknowledgments

The authors are indebted to the reviewers for their insightful comments.
References


