Multi-Channel Mesh Networks: Challenges and Protocols[*]

Pradeep Kyasanur, Jungmin So, Chandrakanth Chereddi, and Nitin H. Vaidya
University of Illinois at Urbana-Champaign

Abstract— **Supporting high throughput is an important challenge in multi-hop mesh networks. Popular wireless LAN standards, such as IEEE 802.11, provision for multiple channels. In this article, we consider the use of multiple wireless channels to improve network throughput. Commercially available wireless network interfaces can typically operate over only one channel at a time. Due to cost and complexity constraints, total number of interfaces at each host is expected to be fewer than the total channels available in the network. Under this scenario, several challenges need to be addressed before all the available channels can be fully utilized. In this article, we highlight the main challenges, and present two link-layer protocols for utilizing multiple channels. We also present a new abstraction layer that simplifies the implementation of new multi-channel protocols in existing operating systems. This article demonstrates the feasibility of utilizing multiple channels, even if each host has fewer interfaces than the number of available channels.**

I. Introduction

In recent years, multi-hop mesh networks have been advocated as a cost-effective approach for providing high-speed last mile connectivity, supporting community networks, etc. Mesh networking architecture may spur the growth of bandwidth-intensive applications, such as video sharing among community members. For building high-speed, yet cost-effective mesh networks, it is desirable to build systems based on standard wireless technologies. Widely used wireless LAN standards, such as IEEE 802.11 [1], provision for multiple channels (e.g., IEEE 802.11a has provisioned for up to 12 channels in the US). Utilizing multiple channels is one approach for increasing the network capacity. The focus of this article is the design of mesh network protocols that utilize multiple channels provisioned for in IEEE 802.11.

Currently available off-the-shelf IEEE 802.11 wireless interfaces (e.g., [2]) can use only one channel at a time, although over time an interface can switch among different channels. Therefore, the number of channels on which a host may simultaneously transmit is limited by the number of interfaces at the host. Reduction in interface costs in recent years has made it feasible to equip hosts with multiple wireless interfaces. Nevertheless, it is expected that the number of interfaces at a host (say, 1-3) will be fewer than the number of channels available in the network (say, 12 channels in IEEE 802.11a networks).

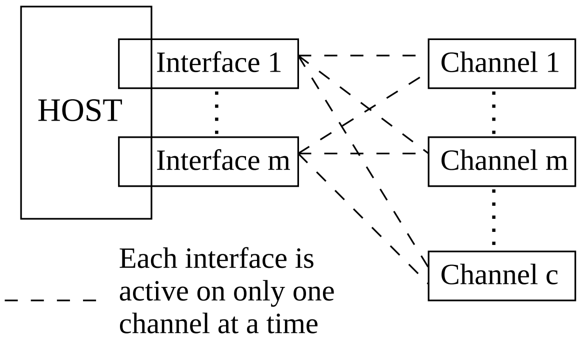


Fig. 1. Channel and interface model.

The available channels may either overlap, such that a channel partially shares its spectrum with adjacent channels, or may be completely non-overlapping. When channels overlap, transmissions on a channel may cause interference on adjacent channels. Therefore, protocols designed for overlapping channels have to account for possible interference on adjacent channels. For simplicity, in this article we restrict our presentation to protocols that require non-overlapping channels. We model all hosts in the network to have m interfaces per host, and each interface can send or receive data over one channel at a time. We assume that there are c non-overlapping channels available in the network. Figure 1 presents our channel and interface model. In this article, we present practical solutions for using all the available channels, even when the number of interfaces at each node is fewer

[*]This research was supported in part by US Army Research Office grant W911NF-05-1-0246, NSF grant ANI-0125859, and Vodafone Graduate Fellowship.

than the channels available in the network (i.e, $m < c$).

The effective use of multiple channels in a multi-hop wireless network requires several challenges to be addressed. For example, consider a network where all nodes have one interface, and there are two channels available. Suppose each node keeps its interface fixed on a specific channel. Under this scenario, if all nodes keep their interfaces fixed on a common channel, then the second channel is wasted. On the other hand, if some nodes tune into channel one, while other nodes tune into channel two, nodes on channel one cannot communicate with nodes on channel two, thereby partitioning the network. If interfaces are allowed to switch, this problem is alleviated, but added complexity is introduced in co-ordinating switching decision between neighboring nodes.

The rest of the article is organized as follows. In Section II, we discuss the challenges, and approaches for handling these challenges. We present two link layer protocols for utilizing multiple channels in Section III. Section IV presents a new abstraction layer in the operating system kernel for implementing multi-channel protocols. We conclude in Section V.

II. Motivation and Challenges

An important challenge in building mesh networks is to provide sufficient network capacity to meet the demands of high-bandwidth applications. The traffic in mesh networks is expected to be directed between (Internet) gateway hosts and mesh hosts. In addition, we expect that in many scenarios, such as community mesh networks, there will be traffic between mesh hosts themselves. A question of interest under such traffic constraints is whether a large number of channels can be utilized even if there are few interfaces per node.

We illustrate the benefits of using multiple channels with a simple example scenario shown in Figure 2. All nodes in the figure are within transmission range of each other. Suppose that the network has two channels, and each node has one interface. In this scenario, if node A is sending data to node B on one channel, then node D can simultaneously send data to node C on another channel. On the other hand, a specific node can receive data from only one other node at any time (under the restriction of one interface per node). Therefore, two channels can be fully utilized *only if we carefully schedule the transmissions*. This simple example illustrates the need for new multi-channel protocols that effectively schedule transmissions to utilize all the available channels, even if each node has fewer interfaces than available channels.

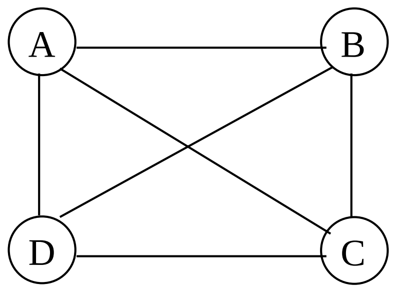


Fig. 2. Example scenario: Each node has one interface and two channels are available.

We have conducted an asymptotic analysis [3] to study the possibility of benefiting from multiple channels in large networks, under the constraint of few interfaces per node. Our results show that in a large network having n nodes (see [3] for full details of models and assumptions), even if every node has a single interface, up to $O(\log n)$ channels can be fully utilized. This result implies that in many practical scenarios, a small number of interfaces per node suffices to utilize a large number of channels. However, the theoretical capacity is achieved with centralized algorithms, and assumptions of steady traffic. Although the theoretical results further substantiate the possibility of utilizing all the available channels with few interfaces, development of distributed protocols is challenging. We next highlight some of the main challenges.

A. Distribute load across channels

One architecture used in the past, when there are c channels, and $m < c$ interfaces per node, is to use only m channels in the network. Every node permanently fixes its m interfaces to a common set of m channels. Although this approach does not require new multi-channel protocols (other than an initial channel assignment), many channels are left unused, especially when the number of interfaces per node is significantly smaller than the number of available channels. Network performance can be improved by using all the available channels, and this requirement implies that traffic should be distributed over all channels.

There are several approaches for distributing traffic load across all channels. One possibility is to fix interfaces of different nodes on different channels [4]. With this approach, a node sends out (or receives) data over only m channels, but by tuning interfaces of different nodes on possibly non-disjoint sets of m channels that together include all c channels, load is distributed across all channels. For example, in Figure 2, nodes A and B can fix their interface to channel 1, and nodes C and D can fix their interface on channel 2, thereby distributing traffic along flows A-B and C-D across both channels. The benefit of this approach is that fixing interfaces on a channel simplifies protocol implementation. However, naively keeping interfaces fixed on different channels may affect network connectivity. For example, in Figure 2, if node A has to send data to node D as well, node A and node D cannot communicate because their interfaces are on different channels. Keeping interfaces of nodes fixed (forever) on different channels results in a network partition if each node has a single interface. Even if each node has multiple interfaces, network partitions may arise if interface assignment is not carefully done.

A second possibility is to frequently switch the interfaces of a node among different channels [5], [6]. Using this approach, over time, a node can potentially distribute its load over all c channels. While this approach is more flexible, the sender and receiver nodes have to co-ordinate with each other before transmission, as described in the next sub-section. Furthermore, switching an interface from one channel to another incurs a delay, and frequent switching may adversely affect performance. We will describe one protocol in Section III that uses careful interface switching to distribute traffic across multiple channels, and is well-suited for single interface networks.

When each node has at least two interfaces, a third possibility is to use a hybrid approach [7] that keeps some interfaces of each node fixed, while other interfaces can switch among channels. We will describe another protocol in Section III that uses a hybrid approach.

B. Channel Co-ordination

In a single channel network, a sender node may initiate a communication to a neighboring node whenever the channel is free. Repeated failures in communicating with a receiver is commonly used as an indication that the receiver is not directly reachable. In a multi-channel network, with fewer interfaces per node than channels, a sender node may be within the communication range of a receiver node, but the interfaces of the two nodes may not be assigned to any common channel, thereby precluding communication. Furthermore, when interfaces switch frequently, a receiver that was previously reachable on a channel may have switched its interface to another channel, and is no longer reachable.

For example, in Figure 2, node A may initially send data to node B over channel 1. Subsequently, node B may switch over to channel 2 and begin data transmission to node C. While node B is on channel 2, if node A again attempts to send data to node B over channel 1, the attempt will fail. Such failures may be mistaken as link breakage, and can adversely affect the performance of higher layer protocols. Therefore, reachability between a pair of nodes, at a given time, depends on the channels assigned to the interfaces of the two nodes at that time, in addition to the channel conditions.

From the above discussion, it is clear that a sender node has to be aware of at least one channel on which the receiver is listening, before communication can be accomplished. There are several approaches to ensure the sender and the receiver share a common channel. When nodes have only one interface, the interface at a node may have to be switched to utilize different channels. Under such a scenario, one possibility is for all nodes to periodically rendezvous on a common channel [5], and during the rendezvous time publish the channels on which they will be on till the next rendezvous time. We present a protocol based on this technique in Section III.

A second possibility is for each node to follow a well-known sequence of switching through channels [6]. Since the sequence is well-known, a sender can predict when it will share a channel with the intended receiver, and send the data only when the nodes share a common channel.

The above two possibilities can be used even if each node has a single interface, but require fairly tight clock synchronization among nodes. When each node has at least two interfaces, other techniques that work with loose time synchronization can be used. One technique is to require each node to fix one interface on a common control channel [8] shared throughout the network. A sender node can negotiate with a receiver node, over the common channel, a channel to use for later data communication. Both nodes then switch one of their other interfaces onto the negotiated channel and complete data transfer. A drawback of this approach is that the common control channel can become a bottleneck to performance when the total number of available channels is large.

A second technique is to allow each node to fix one interface on a channel [7], with different nodes possibly using different fixed channels. Each node announces its fixed channel information to all other neighboring nodes. Since a sender node is aware of the fixed channel of a receiver node, it can switch one of its non-fixed interfaces to the fixed channel of the receiver for sending data. We present a protocol based on this technique in Section III.

C. *Support for broadcast*

Wireless channel is a broadcast medium. In a single channel network, a packet transmitted by a node can potentially be received by all nodes in the transmission range of the node. We define this capability as *local broadcast*. Local broadcast is used by many protocols, such as routing protocols, to efficiently disseminate information in the network. A key challenge in multi-channel networks is to continue to provide efficient local broadcast. If all nodes in the network have an interface on some common channel, then local broadcast packets can be sent over that channel. Otherwise, supporting local broadcast requires explicit support from multi-channel protocols.

For example, suppose node A in Figure 2 intends to broadcast a packet. Ideally, the packet should be received by nodes B, C, and D, which are neighbors of node A. If nodes B, C, and D are on a different channel than the channel used by A for transmitting the packet, then local broadcast is not achieved. If nodes are frequently switching channels, then there may never be a time when all neighbors of node A have an interface on a common channel.

The approach used to support local broadcast depends on the techniques in use for channel co-ordination. For example, when channel co-ordination is achieved through periodic rendezvous on a common channel, broadcast packets can be sent out during the rendezvous interval. This approach does not increase the cost of broadcast when compared to a single channel network, but increases the transmission delay of broadcast packets. In techniques where different nodes use different fixed channels, a node may explicitly transmit a copy of the broadcast packet on all channels. This technique incurs higher overhead for supporting broadcast than in a single channel network.

In general, the appropriate multi-channel protocol to use may also depend on the frequency of broadcasts. If broadcast traffic is more prevalent, approaches that use a common channel for the whole network may be better (although these approaches may not effectively utilize multiple channels).

III. Protocols

Protocols for exploiting multiple channels can be designed at several layers of the protocol stack. For example, the notion of multiple channels can be incorporated into a medium access control (MAC) protocol, or into a link layer solution above the MAC, or into a routing protocol. Higher benefits may be achievable by using cross-layer designs. In this article, we restrict our presentation to two link layer protocols. The link layer protocols are designed to operate over a single channel MAC protocol, such as IEEE 802.11. The link layer protocols are situated between the network layer and the single channel MAC, and present the abstraction of a single channel to higher layers. This allows existing higher layer protocols to operate unmodified.

The first link layer protocol, called MMAC, is intended for the scenario where each node may have only one interface. The second link layer protocol, called Hybrid Multi-Channel Protocol (HMCP), is optimized for the scenario where each node has at least two interfaces.

A. *MMAC*

MMAC [5] is a link-layer multi-channel protocol, designed primarily for nodes with a single network interface. A node equipped with a single interface can only listen to one channel at a time. Therefore, in order to use multiple channels, the interface has to be switched between channels. As discussed earlier, when nodes are allowed to switch channels, a channel coordination method is necessary because a pair of nodes should be listening on the same channel at the time of communication.

MMAC coordinates channels before communication by having nodes negotiate channels using control messages, before exchanging data traffic. In MMAC, time is divided into beacon intervals. At the beginning of each beacon interval, there is a duration of time called “ATIM window”, when all nodes in the network are forced to rendezvous (listen) on a common channel, say channel 1. During the ATIM window, a node which has traffic to send negotiates the channel to use, for subsequent data transfer with the receiver, using control messages. The process of channel negotiation is illustrated in Figure 3.

In this scenario, there are 4 nodes placed in chain-like topology (ordered as A-B-C-D). Node A has packets for B and node D has packets for C. When a new beacon interval starts, every node switches to channel 1 and this starts the ATIM window. Since node A has

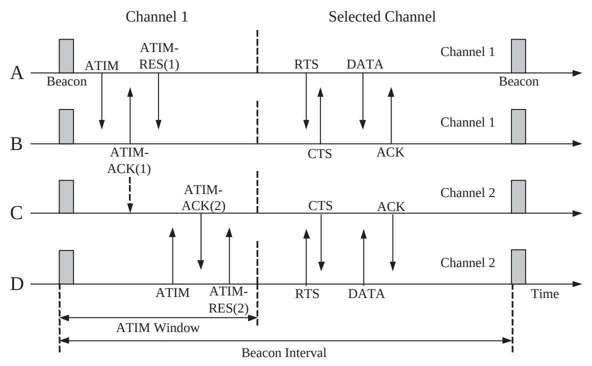


Fig. 3. Process of channel negotiation and data exchange in MMAC.

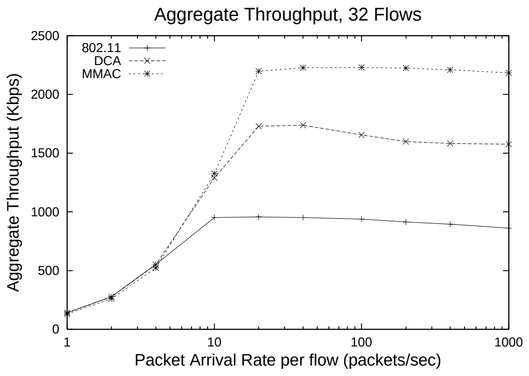


Fig. 4. Aggregate throughput in a scenario with 64 nodes and 32 flows. The number of channels is 3.

packets to send, it waits for a random delay (to avoid collisions), and sends an ATIM packet to B. In the ATIM packet, node A includes a *preferable channel list* (PCL), which specifies the channel usage in its neighborhood. When B receives the ATIM packet, it selects a channel considering sender's PCL and its own PCL. The most preferred channel is the one that is used by minimum number of nodes in the vicinity of the sender and the receiver. Suppose node B chooses channel 1. Then, node B sends an ATIM-ACK packet back to A, including the selected channel. On receiving the ATIM-ACK, node A confirms the negotiation by sending an ATIM-RES packet to B. From ATIM-ACK and ATIM-RES packets, the neighbors of node A and B know that A and B will be on channel 1 for the rest of the beacon interval. This information updates the PCL of the neighbors, so that they can also choose best channels for their use. When the ATIM window is over, nodes switch to the selected channel and communicate on the channel for the rest of the beacon interval. Since every node must start the ATIM window at the same time, MMAC requires clock synchronization among nodes. More detailed description of the protocol is in [5].

We have performed simulations with ns-2 simulator to evaluate the performance of MMAC. We have compared MMAC with IEEE 802.11 DCF which uses a single channel, and DCA [8] which is another multi-channel protocol. DCA requires at least two interfaces per node, so that one interface can be dedicated to stay on a fixed channel and exchange control messages. Figure 4 is a plot of aggregate throughput in a network of 64 nodes and 32 flows. There are 3 orthogonal channels, and the bit rate of each channel is 2 Mbps. As the traffic load increases, MMAC achieves higher throughput compared to IEEE 802.11 DCF and DCA.

In summary, MMAC addresses the channel coordination problem by having all nodes periodically rendezvous on a common channel and negotiate channels with each other. A pair of nodes select a channel which is used by minimum number of nodes in the neighborhood, and the channel usage information is obtained by overhearing control messages in the negotiation period. This technique ensures that the traffic load is distributed across channels. MMAC can support efficient broadcast by exchanging broadcast messages during the ATIM window when all nodes are on a common channel. With these techniques, MMAC achieves high performance by utilizing multiple channels, even when nodes are equipped with a single interface.

B. HMCP: Hybrid Multi-channel Protocol

We now describe HMCP [7], [9], a link-layer protocol that assumes each node has at least two interfaces. Every node divides its available interfaces into two groups. The interfaces in the first group are designated as "fixed interfaces", and are fixed (for long intervals relative to packet transmission times) on specified channels, called "fixed channels". Different nodes are free to choose a possibly different set of fixed channels. The interfaces in the second group are called "switchable interfaces", and these interfaces can frequently switch among the remaining non-fixed channels. To simplify rest of the discussion, we assume each node has exactly two interfaces, one of which is fixed and the other is switchable.

The fixed channels can be explicitly advertised to neighbors by broadcasting "Hello" messages. Whenever a sender needs to send packets to a receiver, it can switch its channel to the receiver's fixed channel and send packets. Thus, once the fixed channel of a node is discovered through the reception of a "Hello" message, explicit channel synchronization is not needed.

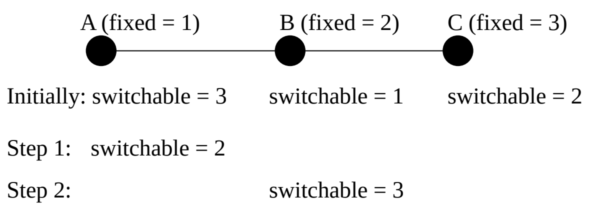


Fig. 5. Example of link layer protocol operation with 3 channels, 2 interfaces.

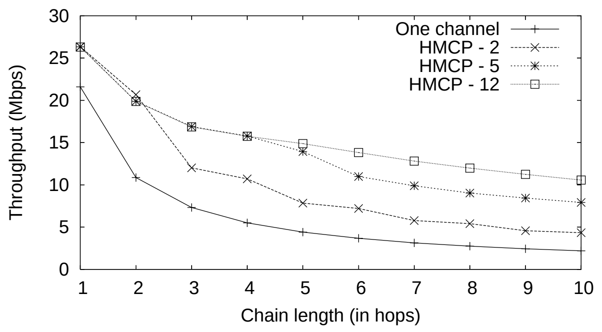


Fig. 6. Performance of single FTP flow.

The selection of fixed channel among nodes can be done in a distributed fashion. Each node maintains a *NeighborTable* containing the fixed channels being used by its neighbors. A node periodically checks the number of other nodes also using the same channel as itself, for the fixed channel. If the estimated number is significantly larger than average, then the node changes its fixed channel to a less used channel, and advertises this information using a "Hello" message. More details are in [9].

Figure 5 depicts a simple data transfer example using HMCP. Each node has two interfaces - one fixed and one switchable. Assume that node A has packets to send to node C via node B. Suppose nodes A, B, and C have their fixed interfaces on channels 1, 2, and 3 respectively. Assume that initially nodes A, B, and C have their switchable interfaces on channels 3, 1, and 2 respectively. In the first step, node A *switches* its switchable interface from channel 3 to channel 2, before transmitting the packet, because channel 2 is the fixed channel of node B. Node B can receive the packet since its fixed interface is always listening to channel 2. In the next step, node B switches its switchable interface to channel 3 and forwards the packet, which is received by node C using its fixed interface. Once the switchable interfaces are correctly set up during a flow initiation, there is no need to switch the interfaces for subsequent packets of the flow (unless a switchable interface has to switch to another channel for sending packets of a different flow).

In the proposed protocol, nodes in a neighborhood may be listening to different channels, and therefore a single broadcast transmission does not reach all neighbors. Therefore, local broadcast is implemented by sending a copy of the broadcast packet on all channels. Consequently, the total overhead of local broadcast is larger than in a single channel network. However, if the overhead is measured in terms of packets sent per channel, then this approach continues to have the same overhead as in a single channel network.

We have evaluated the performance of HMCP using simulations in Qualnet. Here, we will present a sample result to illustrate the benefits obtained by using multiple channels and multiple interfaces. In Figure 6, we evaluate the performance of HMCP in simple chain topologies. The length of a chain is varied from 1 to 10 hops. A FTP flow is setup from the first node to the last node of the chain. Figure 6 compares the flow throughput with a 1 channel network (labeled "One channel"), and the flow throughput of HMCP with varying number of channels (labeled "HMCP - x" where x is the number of channels available). The experiment assumes that our protocol uses two interfaces. Each channel is assumed to support a data rate of 54 Mbps, which is the highest data rate specified in IEEE 802.11a. As we can see from the figure, the FTP throughput in single channel networks rapidly degrades when the number of hops along a chain increases (this behavior is well-known). However, the FTP throughput degradation is less severe when multiple channels are used.

When multiple channels are available, HMCP assigns the fixed channel of successive nodes along the chain to different channels. Also, when an intermediate node is receiving data using one interface, it can simultaneously forward data to the next node using the second interface. Consequently, HMCP offers higher throughput by *using different channels on successive hops*, and by *using the two interfaces to receive and send data in parallel*.

The key observation from Figure 6 is that multiple channels can significantly improve throughput of a flow in multi-hop scenarios. Furthermore, even with only a few interfaces (2 in this example), having large number of channels (up to 12 channels in this example) is beneficial.

IV. Implementing multi-channel protocols

In the previous sections, we have identified the challenges, and presented simple protocols for multi-channel wireless networks. Implementing multi-channel protocols that require interfaces to switch frequently is non-trivial. In this section, we identify the additional features that have to be added to operating systems, with Linux as an example, to support protocols that require interfaces to switch. We then present the design of a new abstraction layer to manage interface switching across multiple channels. The abstraction layer also offers the benefit of presenting a single virtual interface to higher layers, independent of the number of physical interfaces present. This allows existing higher layer protocols to operate unmodified, even if the number of physical interfaces or channels is changed.

A. Need for extra support in current operating systems

Some of the difficulties in using existing operating systems, such as Linux, to support multiple channels are as follows.

1) Specifying channel to use for reaching a neighbor: An implicit assumption made in many operating systems is that *there is an one-to-one mapping between channels and interfaces*. This assumption is satisfied in a single channel network because an interface is fixed on the single channel used in the network. This assumption continues to be met in a network where each node has m interfaces, and the interfaces of a node are always fixed on some m channels. However, as we discussed in earlier sections, it is possible that the number of interfaces per node is significantly fewer than the number of channels. Therefore, protocols may require an interface to send data over multiple channels, by switching across channels. When interfaces have to switch across channels, the assumption that there is an one-to-one mapping between channels and interfaces is broken.

The one-to-one mapping assumption is reflected in kernel routing tables only specifying the interface to use for reaching a neighbor. Consider the scenario where a node has a single interface, but has to send data to one neighbor over channel 1, and another neighbor over channel 2. Now, associating the routing table entry of each node with only the interface information does not specify the need to use different channels for reaching the neighbors. Without this information in kernel tables, user space applications cannot transparently send data over multiple channels.

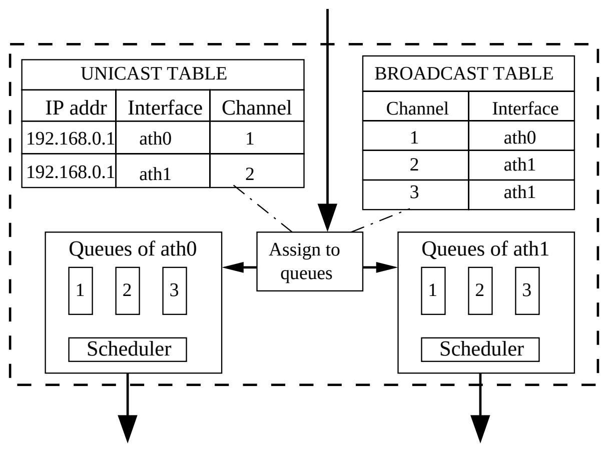
2) Specifying channels to use for broadcast packets: Certain multi-channel protocols (e.g., HMCP) require a copy of the broadcast packet to be sent out over multiple channels. Other protocols may require a broadcast packet to be sent on a specified common channel. Therefore, support is required to specify on what channels (one or more) broadcast packets have to be sent out. Without such support, higher layer applications that require broadcast may have to be modified to explicitly specify channels to use for broadcast.

3) Managing interface switching: As we discussed above, an interface may have to be switched between different channels to enable communication with neighboring nodes on different channels, and to support broadcasts. When an interface is currently sending a packet on some channel c_1, and a new packet is received that has to be sent over some other channel c_2, then the interface has to be scheduled to switch to channel c_2 only after communication on channel c_1 completes. If higher layer applications explicitly control when an interface has to be switched, then the applications have to be modified to carefully consider the current interface status before initiating a switch. Furthermore, consistency may have to be maintained between requests from different applications.

Therefore, there is a need to provide support in the kernel for maintaining consistency between different switch requests. Furthermore, when a packet cannot be sent out immediately because the interface is on a different channel at that time, then the packet has to be buffered for later transmission. Therefore, there is a need to support buffering of packets, and subsequent scheduling and transmission of buffered packets.

B. Proposed abstraction layer

The above challenges motivate the design of a new abstraction layer that is located between the network layer (and ARP) and the interface device drivers. The abstraction layer provides support for managing multiple interfaces, interface switching, and packet buffering. The layer exposes a *single virtual interface* to the higher layers. In addition, the abstraction layer provides an interface for higher layers to specify the channel and interface to be used for each destination node. This information is stored in a table, called the "Unicast table", within the layer. The abstraction layer maintains a second table, called the "Broadcast table", which contains a list of channels (and interfaces) on which a broadcast packet has to be sent out. The layer provides an interface to the higher layers for setting up the broadcast table.



UNICAST TABLE

IP addr	Interface	Channel
192.168.0.1	ath0	1
192.168.0.1	ath1	2

BROADCAST TABLE

Channel	Interface
1	ath0
2	ath1
3	ath1

Fig. 7. Components of the abstraction layer: Example assumes two interfaces and three channels are available.

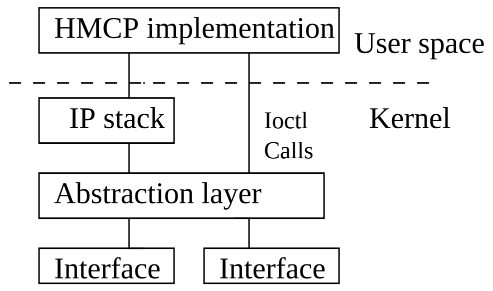


Fig. 8. Implementation of HMCP protocol using abstraction layer support.

The abstraction layer associates each interface with a separate queue for all the available channels. When a unicast packet is handed down to the layer, the channel and interface to be used for that packet is looked up in the unicast table. Based on the lookup, the packet is inserted into the appropriate channel queue. Similarly, when a broadcast packet is handed down to the layer, a copy of the packet is inserted into queues corresponding to every channel on which the packet has to be sent out.

The abstraction layer currently uses a simple round robin scheduler that services the channel queues of every interface. An interface is switched to another channel if the interface has been on the current channel for at least T_{max} time, and another channel queue associated with the interface has packets pending. It is possible to implement more sophisticated schedulers that support different channel switching policies.

Figure 7 represents the components of the abstraction layer. The table values are filled for an example scenario having two interfaces (called “ath0” and “ath1”) and three channels. In this example, we assume that local broadcast is supported by sending a copy of the broadcast packet on all three channels. Hence, the broadcast table has entries to send out the broadcast packet on three channels.

In summary, the main benefits of our proposed abstraction layer is to completely hide the complexity of managing multiple channels and interfaces from higher layers. Therefore, existing routing protocols, ARP mechanisms, etc. can be used without any modifications. The abstraction layer is generic enough to support different multi-channel protocols, and simplifies the implementation of protocols that require frequent interface switching.

C. *Prototype implementation*

We have implemented a prototype of the HMCP protocol in Linux. The abstraction layer was implemented as a kernel module. The main components of the HMCP protocol - exchanging “Hello” messages with neighbors, and deciding fixed channel assignments, were implemented as a user-space daemon. The user space daemon sets up the unicast and broadcast tables in the abstraction layer through “ioctl” calls exported by the kernel module. All other user applications run unmodified. Figure 8 depicts the prototype architecture.

The implementation runs on low-cost hardware from Soekris that is equipped with two wireless interfaces. We have validated the prototype with simple experiments that use 4 channels and two interfaces. Prototype development has demonstrated the feasibility of utilizing multiple wireless channels with commodity hardware, even if number of interfaces per node is fewer than the number of channels.

V. CONCLUSIONS

In this article, we have considered multi-channel protocols for the scenario where each node has fewer interfaces than the number of available channels. We argued that this scenario is likely in practice, and identified several challenges that have to be addressed before multiple channels can improve network performance. We presented two link layer protocols, MMAC and HMCP, for utilizing multiple channels. MMAC was designed for networks having a single interface per node, while HMCP was optimized for networks having at least two interfaces per node. Performance evaluations suggested that the proposed protocols can substantially improve network throughput by leveraging all the available channels.

Implementing multi-channel protocols is complicated by the lack of sufficient support in existing operating systems. Therefore, we presented a new abstraction layer that simplifies the implementation of new multi-channel protocols, and described the implementation of HMCP over the abstraction layer as an example. Our work has demonstrated that large number of channels, provisioned for in current wireless technologies, can be successfully utilized in practical systems with carefully designed multi-channel protocols.

REFERENCES

[1] *IEEE Standard for Wireless LAN-Medium Access Control and Physical Layer Specification, P802.11*, 1999.

[2] "Atheros inc," http://www.atheros.com.

[3] P. Kyasanur and N. H. Vaidya, "Capacity of Multi-Channel Wireless Networks: Impact of Number of Channels and Interfaces," in *ACM Mobicom*, 2005.

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

[5] J. So and N. H. Vaidya, "Multi-channel MAC for Ad Hoc Networks: Handling Multi-Channel Hidden Terminals using a Single Transceiver," in *Mobihoc*, 2004.

[6] P. Bahl, R. Chandra, and J. Dunagan, "SSCH: Slotted Seeded Channel Hopping for Capacity Improvement in IEEE 802.11 Ad-Hoc Wireless Networks," in *ACM Mobicom*, 2004.

[7] P. Kyasanur and N. H. Vaidya, "Routing and Interface Assignment in Multi-Channel Multi-Interface Wireless Networks," in *WCNC*, 2005.

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

[9] P. Kyasanur and N. H. Vaidya, "Routing and Link-layer Protocols for Multi-Channel Multi-Interface Ad hoc Wireless Networks," Tech. Rep., University of Illinois at Urbana-Champaign, May 2005.