Dichotomy: A Practical Architecture for Multi-channel IEEE 802.11 Multi-hop Networks

Kun Tan, Haitao Wu
Microsoft Research Asia
Beijing China

Li (Errar) Li
Bell Labs, Lucent
USA

Qian Zhang
HKUST
Hong Kong, China

Yongguang Zhang
Microsoft Research Asia
Beijing China

***Abstract* -- In this paper, we present Dichotomy, a novel practical architecture that exploits channel diversity to improve wireless multi-hop network throughput with a single transceiver. Unlike previous link-layer multichannel work that requires complex coordination and is difficult to implement, Dichotomy operates without clock synchronization, and with greatly reduced switching overhead. The core idea is to strategically select a subset of nodes as *anchors* whose channels seldom change; while other nodes are *hoppers* that dynamically switch its radio among channels of neighboring anchors. Communications are enforced between anchor-hopper pairs or same-channel anchor-anchor pairs. It uses localized distributed algorithm to select anchors and assign channels to anchors, while maximizing the channel diversity and keeping the network well connected. We implement Dichotomy as a software shim layer between IP and MAC, and evaluate it in a real test-bed. Experimental results show that Dichotomy effectively boosts the network throughput up to 100%. We further conduct extensive simulations and demonstrate that the Dichotomy architecture is scalable and achieves better or comparable performance than other previous work.**

Keywords -- Channel diversity, multi-channel, wireless multi-hop network

I. INTRODUCTION

This paper focuses on exploiting channel diversity to mitigate the interference, so as to improve the overall throughput of wireless multi-hop networks (WMN). We specifically consider the case in which most of nodes in a WMN are equipped with single transceiver. This assumption has much practical value since current devices are typically equipped with only one single half-duplex 802.11 radio interface.

To make use of multiple channels in a single-radio network, nodes need to dynamically configure their radios with different channel based on the traffic requirement. Recently, many approaches have been proposed in the literature that schedule wireless transmissions across different channels with different time granularities: 1) *packet level*. The schedule is performed in a packet-by-packet manner. Therefore, a transmission of a subsequent packet may be on a different channel from the previous one, e.g. RDT [15] and McMAC [4]. 2) *Link level (or Super-frame level).* The schedule is performed for a link between two nodes based on a small duration of time, typically within one hundred milliseconds. Within this duration, a group of packets to the selected wireless link can be transmitted. MMAC [1] and SSCH [2] are examples in this category. 3) *Session level*. In this category, the channel of a node's radio keeps unchanged during the entire communication session. Component-based Channel Assignment (CBCA) [8] is an example of session-level approaches.

Ideally, it would be better to perform channel scheduling at smaller granularity, as it would have higher flexibility to exploiting channel diversity. However, in practice, there is a tradeoff among the potential performance gain, the practical overhead and the system complexity. Packet-level scheduling suffers a high overhead, since it may switch a channel after each packet transmission. Switching channel has a delay that is comparable to packet transmission time for current NIC hardware. Link level approaches are designed to amortize this channel switching overhead by scheduling groups of packets together. However, existing designs require complex coordination among networked nodes, and therefore are difficult to implement in multi-hop networks. Session level approaches are proposed to avoid some issues of previous two types of solutions. However, these approaches has the least ability to exploit channel diversity.

In this paper, we present a new practical multi-channel architecture for single radio WMNs that performs channel scheduling at link level, while eliminateing two most important constraints existed in previous solutions [1][2]: tight clock synchronization and large channel switching overhead. Moreover, many of these solutions requires changing the underlying MAC behavior, but our scheme can be easily implemented as software modules on top of existing IEEE 802.11 MAC over commercial off-the-shelf hardware.

Our proposed architecture is called *Dichotomy*. The core idea is to strategically divide the nodes in the network into two subsets. One subset of nodes (named *anchors*) is assigned fixed working channels, while the other nodes (named *hoppers*) keep hopping among channels of neighboring anchors. Communications are enforced to occur only between anchor-hopper pairs or anchor-anchor pairs if they are assigned on the same channel. Since only one node between two communication parties is allowed to dynamically change channels, it does not require clock synchronization among neighboring nodes. We further develop a localized and distributed algorithm that dynamically selects anchors and coordinates the working channels of anchors in such a way that the interference in the network is minimized. Our evaluation demonstrates that although Dichotomy restricts anchors work on almost fixed channels, it is able to exploit multi-channel capability effectively.

Dichotomy differs significantly from previous multichannel MAC designs that are based on quiescent channel, which also do not require time-synchronization, e.g.

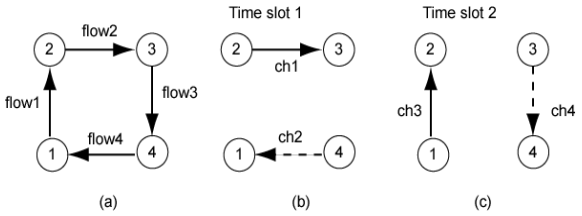


Figure 1. The illustration of the need of dynamic channel switching. (a) a four-node wireless network with four flows. (b) and (c) all nodes are scheduled to dynamically switch their channels. Then, simultaneous transmissions can happen on different channels.

RDT [15] and its later extension [16]. First, RDT require packet-level channel scheduling, so that after transmission, each node could return to a pre-assigned quiescent channel for receiving. Second, RDT allows all nodes in the network to change channels and packets are sent to neighbors in an opportunistic way. This may cause the *deafness problem* [16], which arises because an intended receiver may currently be transmitting in the quiescent channel of a third node, and thereby cause transmission failures and backoff. As we will show later, this deafness problem also exists in link-level multi-channel solutions that exploit opportunistic synchronization, which may cause performance loss. However, Dichotomy solves this problem by fixing anchors in stable channels, and therefore the neighboring hoppers and anchors can always ensure they are on the same channel before initiating transmissions.

We have implemented the Dichotomy architecture on Windows platform with commercial 802.11 wireless interface cards. We deploy and evaluate our implementation in a 10-node wireless multi-hop test-bed in our building. We further implement Dichotomy in NS2 and conduct large-scale simulation studies and compare our solution with other link-level solutions e.g. SSCH [2]. Our experience shows that Dichotomy architecture yields great improvement in network throughput compared to single channel 802.11, and achieves better or comparable performance compared to those solutions that require complex global time synchronization in the network.

The primary research contributions of our paper are summarized as follows:

1. We propose a novel Dichotomy architecture that exploits channel diversity at link level for single radio WMN. Our solution does not need clock synchronization among neighboring nodes.
2. We propose a localized and distributed anchor selection and channel assignment algorithm to minimize the interference in the network. We prove the proposed algorithm converges.
3. We implement our architecture in a real system and demonstrate that it is feasible and practical to perform channel schedule at link level. To the best of our knowledge, we are not aware of other implementation of multi-channel schemes for single radio 802.11-based multi-hop wireless network.
4. We evaluate our prototype in both an indoor test-bed and large-scale NS2 simulations. Our experimental results suggest that our architecture handles practical overheads well and greatly improves the network throughput by exploiting multichannel capability.

The rest of paper is organized as follows. Section II describes the background and the motivation of our work. We present the Dichotomy architecture in detailed in Section III. We present the system design and implementation in Section IV. We evaluate the performance of Dichotomy in our indoor test-bed and in NS2 simulator in Section V. Related work is discussed in Section VI. Section VII concludes the paper.

II. BACKGROUND AND MOTIVATION

Wireless multi-hop networks are typically implemented using IEEE 802.11 radios. Each node in a WMN is typically equipped with a *single* 802.11 radio, which can operate on a *single* 2.4G or 5G channel. Transmissions in the same channel may interfere with one another if they are within the interference range. Using multiple channels, this interference can be mitigated. As an example, Figure 1 shows a simple wireless network with four nodes. If all nodes are configured in the same channel, these flows interfere with one another and only one flow can transmit at any given time, and thus share the same channel capacity. However, if nodes are allowed to dynamically switch between two channels, i.e. channel 1 and 2, the transmissions can be scheduled on different channel simultaneously, and therefore double the overall network throughput, as shown in Figure 1b and c.

Ideally, these flows should be scheduled on different channels at small time granularity to ensure fairness among competing flows. For example, if we schedule the flow at session-level (i.e. first schedule the flow 2 and 4, and then schedule flow 1 and 3 after the completion of the previous two flows), this will cause a large latency for some flows and therefore is unacceptable, since users may have already aborted their transmissions[1]. In contrast, scheduling at packet or link level will provide users the illusion of concurrent services on different channels. However, performing scheduling at low level faces two important challenges: 1) practical switching overhead; and 2) tight clock synchronization among nodes. The first issue arises because a node needs to frequently switch from one channel to another, and this switching overhead can be significant if the scheduling is made at very fine granularity. The second issue occurs because when two communicating nodes are scheduled to transmit/receive on different channels, the link between them is broken and if they want to resume the communication, they have to agree to return to the same channel at the same time. In the following, we elaborate these challenges and explain why they are difficult to resolve in practice. We believe these are major obstacles for implementing existing approach in real systems.

- *Practical Switching Overhead:* Although current wireless hardware does support channel switching capability, there is an overhead for doing so. Firstly, it takes a delay for the radio hardware to reset its Voltage Control Oscillator (VCO) to provide stable frequency output. This *switching latency* ranges from 80us to a few hundred microseconds. In our work, we operate on a commercial off-the-shelf 802.11 NIC based on Atheros AR5212 chipset, which has a switching latency of 300us

[1] Current session-level approaches cannot handle such cases and will fall back to use only single channel for all nodes.

in 802.11a mode. Such latency is already comparable to a full-sized packet transmission time (1500Bytes) on 802.11a, and therefore per-packet switching is unrealistic. Secondly, when two nodes agree to switch to a common channel at the same time, even if they have perfectly synchronized clock, there still needs a guard-time to ensure that these two nodes are on the same channel before actual transmission could happen. This is because a NIC cannot reset a channel at arbitrary time point. If a transmission is going on, reset channel will cause packet drops. As a consequence, the switching action may be delayed by a maximal packet transmission time. This time can be from 385us to 2.16ms depending on the rate selected. In practice, this guard-time should be several milliseconds in practice and actually dominates the overall switching overhead.

- *Need for Tight Clock Synchronization:* In order for two neighboring nodes on different channels to rendezvous on a common channel at the same time, clock synchronization is needed and synchronized nodes need to re-synchronize periodically, or otherwise their clocks would eventually drift away. However, clock synchronization is particular difficult in multi-channel settings since traditional techniques using broadcasts work poorly [4]. Many existing link-level approaches require all nodes in the neighborhood to perform synchronized switching, with a desired precision at a few hundred microseconds [1][2]. This implies that all nodes in the network need to agree on a common clock, which is particular difficult in a dynamic environment like multi-channel multi-hop networks. As far as we know, we are not aware any solution on this and it remains an open problem [5].

We argue that the aforementioned two practical issues are actually coming from a common source. That is, *both two nodes in a communication link are allowed to dynamical switch their channels*. This creates the first issue: when a node switches to a channel, it has to wait for the other node to switch to the same channel before start communicating (guard-time); and it also creates the second issue as two nodes hop among channels, they have to follow precisely the pre-scheduled rendezvous time that cannot be changed independently. However, for any communication pairs, if only one node is allowed to switch, while the other remains on a fixed channel, these two issues are largely mitigated: 1) it does not need a guard-time, since the other node is guaranteed to be on that channel; and 2) no clock synchronization is needed. Although two nodes still need to synchronize their actions, this can be done by message-passing, instead of a pre-scheduled timer. Surprisingly, even if we limit the number of nodes that can dynamically switch their channels during normal operation, it may still be able to utilize multi-channel capability effectively. For example, in the four nodes network in Figure 1, it is indeed unnecessary for all four nodes to dynamically change channels. If two nodes, say 1, 3, stay on two different channels all the time, while node 2 and 4 performs dynamic channel switching, it actually achieves the same performance gain. This motivates us the design of Dichotomy.

III. DICHOTOMY

Dichotomy strategetically divides nodes into two types: *anchors* and *hoppers*. Anchors configure their radios to stable working channels that seldom change. Hoppers dynamically hop their channels to communicate with neighboring anchors. Communications are enforced to happen between anchor-hopper pairs or between two anchors if they are on a same channel. As illustrated in last section, Dichotomy effectively eliminates the overhead of large guard-time and the complexity of global clock synchronization, but without scarifying much flexibility in utilizing multi-channel capability. In the following, we present the distributed algorithm that dynamically selects anchor nodes and coordinates the working channels among them. We will present the system architecture of Dichotomy in the next section.

We refer a *Dichotomy assignment* as an assignment that selects a node to be either a hopper or an anchor and assigns a working channel to each anchor. We would like to find an assignment that maximally exploits channel diversity while at the same time retains the *similar connectivity* of the wireless network when all nodes are configured in a same channel, which we term as the *single-channel assignment*. Hereby, we term a Dichotomy assignment is valid if *any one-hop neighboring nodes in the single-channel assignment would be at most two hops away in any Dichotomy assignment.*

A. The Dichotomy Assignment Problem

We formulize the Dichotomy assignment problem in this section. We assume all transmissions use a fixed power P. We assume there are K channels available numbered $1,2,\ldots,K$. Let $G_0=(V,E_0)$ be the communication graph of a given wireless multi-hop networks. That is, suppose u, v are tuned to the same channel; an edge (u,v) exists in E_0 iff transmissions from u to v are received with a reception power greater than a pre-determined threshold.

We would like to assign a node as either a hopper or an anchor. For anchors, we would like to assign channels in such a way to maximally exploit channel diversity. Denote λ as the assignment. For a node u, if it is a hopper, $\lambda(u) = 0$; if it is an anchor and assigned channel i, $1 \le i \le K$, then $\lambda(u) = i$. Given an assignment, it induces a graph $G = (V, E)$. Assuming G_0 is connected, G should be connected as well under the given assignment. Further, to ensure the assignment is valid, the following requirement should be satisfied. That is, for $(u, v) \in E_0$, either $(u, v) \in E$ or there exists w such that, $(u, w) \in E$ and $(v, w) \in E$. Note that, this requirement implies G is connected.

Ideally, we would like to compute an assignment to maximize the network throughput. However, the problem is NP-hard even with known traffic patterns.

Theorem 1. *Given a multi-hop network $G = (V, E)$ and a known traffic pattern, computing a Dichotomy assignment λ that maximizes the network throughput is NP-hard.*

We outline the proof in Appendix.

Note that, our problem is unique, although it looks similar to a few known problems. It is different from traditional channel assignment or graph coloring problem. Our problem has a connectivity constraint due to the different roles of anchor and hopper, and communication only happens

between anchor and hopper or anchors with the same channel. Further, anchors form a dominating set. However, we do not want to compute a minimal dominating set nor we want to compute a connected dominating set. One can think of our goal as minimizing total interference with the connectivity constraint through channel assignment and anchor selection.

B. *The Localized Distributed Dichotomy Assignment Algorithm*

Given the NP-hard nature of the problem, in this subsection we present a localized distributed algorithm that computes a valid assignment. Our proposed algorithm jointly applies two greedy heuristics that select valid assignment to maximize the links between anchors and hoppers, and choose channels for anchors to minimize the interference.

We assume messages will be delivered reliably in bounded time. This can be achieved through retransmissions. We assume nodes exchange two-hop topology information through beaconing. Each node has three states related to assignment: *unassigned*, *declared* and *committed*. A node initializes in the *unassigned* state. When a node declares it wants to be an anchor or hopper, it will be in the *declared* state. When a node receives all ACKs from its neighbors, it goes into the *committed* state regarding its chosen role. For a given assignment λ, we say invariant ℓ is satisfied at node u if u can reach each neighbor in G_0 either directly (one node is a hopper, the other is an anchor) or indirectly through another neighbor (both nodes are hoppers or anchors not assigned the same channel). We say an anchor u is *pined* if there are two hoppers v, w which are neighbors of u in G_0,and $(v, w) \in E_0$. In our algorithm, a node u may be forced to change to a channel i that is assigned to node w. If this happens and i has been assigned by w to itself, we denote the channel i assigned to a node u as $i_{ID(w)}$. If u further forces v to take channel i, then v will get $i_{ID(w)}$ as well.

The algorithm has the following simple operations:

1. If a node is unassigned, and has a committed anchor, it declares as a hopper.
2. If a node receives ACKs from all neighbors after declared, it will enter into committed state. It sends a committed message to all neighbors.
3. if a node u and one of its neighbors v in G_0 are both NOT anchors (can be unassigned); however, they do not share a common anchor, then the one with a smaller ID will declare as an anchor; it will choose the channel that *conflicts* with the least anchors in its two-hop neighborhood.
4. if two committed anchors u, v are neighbors in G_0 , and do not have a hopper in common, further suppose u has channel $i_{ID(w)}$, and v has channel $j_{ID(t)}$, then u will change the channel to $j_{ID(t)}$ if $ID(t) < ID(w)$. Otherwise, v will change to $i_{ID(w)}$.
5. if a committed hopper u satisfies the following conditions: (1) the number of its neighbor anchors is less than half of its neighbors; (2) the last message u received from each neighbor in G_0 is its commit message; (3) u switches to anchor, invariant ℓ still holds at u; then u will declare as a anchor.
6. if a committed anchor u satisfies the following conditions: (1) has a neighbor anchor v in G_0 which is also committed; (2) both u and v share the same channel; (3) the last message u received from each neighbor in G_0 is its commit message; (4) u is not pined; (5) u switches to hopper, invariant ℓ still holds at u; (6) after u changes to hopper, for each node v of $N(u) \cup \{u\}$, the number of v's anchor neighbor is still more than half of v's neighbors; then u will declare as a hopper.
7. If u receives a role change from a neighbor w before its role change message gets received by w, and if w's change makes invariant ℓ invalid, then only one of the two nodes w, u can commit; the one with lower node ID has high priority.

A node marks a neighbor's role changed only if it receives its commit message. A node will assume its previous role if its declare message fails to commit.

Rule 1-5 achieves invariant ℓ at each node u while trying to maximize the anchor-hopper links. The idea for Rule 6 is to remove unnecessary anchors. Rule 7 handles the simultaneous role changes in a distributed system. Note that the protocol is adaptive and deals with topology changes automatically. As elaborated later, we implement this protocol through beaconing across channels. Nodes report the roles of its neighbors inside beacons. Only new nodes need to scan all channels to receive beacons in every channel.

We now illustrate the protocol operation using an example, which is a partial topology in our test-bed, as shown in Figure 4. Let's consider only nodes 1,6,7,5,8,9. Assume node 6 declares as an anchor and chooses channel R; then node 1 and 7 receiving its declare message; node 1 and 7, each replies with an ACK. In the mean time, each declares as a hopper. Suppose node 5 hears 7's declare message first, then 5 will declare as an anchor due to Rule 3. Suppose node 5 knows its two-hop topology through beacon messages. It will choose a channel B that does not conflict node 1's assignment. Further suppose, node 8 and 9 declares as an anchor simultaneously, node 8 chooses channel Y and the other G. Apply Rule 4, then 9 will change to channel Y.

We show the protocol outlined above converges and when it converges, the assignment is a valid Dichotomy assignment.

Theorem 2*. The localized distributed Dichotomy assignment algorithm converges.*

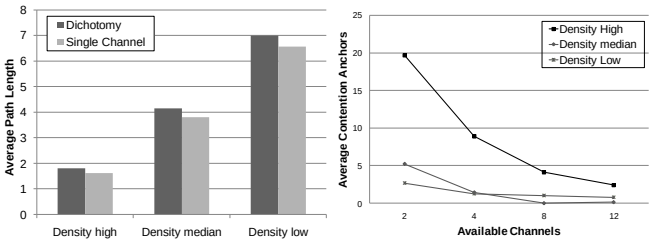
Theorem 3*.The converged assignment induces a graph $G = (V, E)$ that is connected; furthermore, the maximum stretch for any path is at most 2.*

We outline the proof in Appendix.

C. *Evaluation*

We evaluate the algorithm proposed earlier via simulations. We consider three scenarios with different node density. We fix 100 wireless nodes with transmission range of 100 meters, and put them into areas of 200x200 (density high), 500x500 (density median) and 800x800 m^2 (density low), respectively. In our simulation, we want to verify: 1) the impact of Dichotomy on the path length; and 2) the quality of resulted channel assignment. Figure 2 shows the average path length of Dichotomy as well as that of the single channel assignment. Theorem 3 states the worst case path length penalty is twice, but Dichotomy actually only increase the path length very slightly in general. This is reasonable since wireless multi-hop networks have many redundant links, removing some links (e.g. hopper-to-hopper links) would not affect network connectivity too much. Figure 2b shows the

number of average contention anchors for an anchor with different number available channels under different network densities. We call an anchor is contending with another anchor if they are within two-hop range and on the same channel. We can see that Dichotomy indeed allocates channels to each anchor fairly well. With the increase of available channels, the number of contending neighbor anchors decreases proportionally. Given the 12 channels defined in 802.11a, even in the densest network setting, the number of contending neighboring anchors becomes low (<3).



(a) (b)

Figure 2. Evaluation of Dichotomy assignment algorithm. (a) The average path length of Dichotomy comparing with the single channel assigment under random networks with different densities. (b) The average number of contention anchors with different number of available channels.

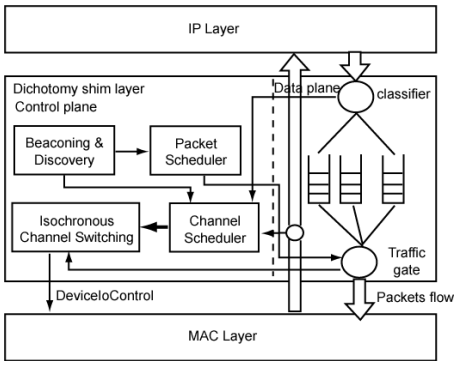


Figure 3. The system architecture of Dichotomy.

IV. System Architecture and Implementation

A. System Architecture

Dichotomy is implemented as a thin shim layer between IP layer and traditional MAC layer. It provides a virtual network interface to the IP layer as a normal Ethernet interface while hiding detailed operation on multiple channels. It adds functionalities on both data plane and control plane.

In the data plane, it contains two functional modules:

1. Per-neighbor FIFO queues and priority queue. Each neighbor is identified with its unique MAC address. Since neighbors of a node may locate on different channels, packets to each neighbor should be scheduled individually. The priority queue prioritizes the transmission of signaling packets, as detailed in subsection E.
2. Traffic gate. This module regulates the outgoing packets. It actually controls how many packets are allowed to be buffered at the MAC layer. As explained in subsection B, this module is critical to implement fast channel switch in COTS 802.11 NIC, such as those based on Atheros chipsets.

In the control plane, it contains four major modules:

1. Isochronous Channel Switching. This module cooperates with the Traffic Gate to implement fast channel switching with a delay of 570us. See details in subsection B.
2. Beaconing and Discovery. This module handles the task of learning about one-hop neighbors of a node. It employs a beaconing mechanism that allows nodes to discover one another. We explain the details in subsection C.
3. Channel scheduler. This module determines the channel on which a hopper would work on in the next time slot. It also coordinates anchors and hoppers by exchanging small signaling packets. We explain in the details in subsection D.
4. Packet scheduler. It defines how packets are scheduled to be sent when a hopper switches to the channel assigned to an anchor. How packets scheduled is discussed in subsection E.

B. Fast Channel Switching

Common drivers, e.g. MadWifi, expose programming interfaces that only switch a channel with several milliseconds delay (ranging from 2.8ms~5.8ms) depending on the system type and the load on the system. This is because as a common purpose driver, it is desirable to have a generic system call that can be used by many tasks, e.g. a) do radio modes switching as well; and b) be called at any time even if a transmission/receiving is ongoing on the hardware. Changing radio mode requires the hardware to reset its whole radio front-end circuit which takes time (for example, 1.8ms in Atheros AR5212 chipset); while changing channel when there is ongoing communication would require the whole reset of MAC controller as well as the DMA controller, which taking even longer time (up to 4ms in our experiments).

For a multi-channel protocol, we only need to reset VCO to provide a new frequency, and actually we do not want to reset channel during the ongoing communication, since it will cause packet drops. In this work, we customize a new AR5212 driver in Windows platform that provides a new system call which can directly reset VCO while not the whole radio front-end, which reduce much overhead mentioned earlier. But this system call cannot be invoked if the hardware is transmitting or receiving frames. We use a technique called *traffic gating* to prevent channel switching from happening during a packet transmission. The basic idea is that we keep counting packets we have sent to the MAC layer. A system call to channel switching is only performed when the count is zero. Note that traffic gating can only guarantee that there are no packets sending out, and with current hardware, by no means can we know that whether or not the hardware is receiving a packet. Therefore, before each reconfiguration on VCO, we still need to reset the receiving unit. It takes 300us to reset VCO. As we also need to reset the receiving unit, this adds another 270us overhead. This results 570us delay for performing a channel switching.

C. Beaconing and Discovery

Dichotomy employs a beaconing mechanism to discover one-hop neighbors. The beacon packet contains the MAC address, the role of the node, and the assigned channel if the

node is an anchor. The beacon also contains a list of one-hop neighbor information of the node. Therefore, by receiving beacons, a node can actually discover its two-hop neighbors. Beacons also contain fields to negotiate the roles among neighboring nodes. Both anchors and hoppers are beaconing across multiple channels. However, the difference lies in that anchors will return to its assigned channel immediately after sending a beacon; while hoppers may stay on the channel just beaconed for a short delay, hopefully receiving a response from an anchor on that channel. An anchor may reply to the hopper if it receives a beacon from its assigned channel. Note that beaconing is infrequent. Allowing anchors to leave their assigned channels for a short period of time does not have significant impact on the ongoing traffic. In current implementation, the beacon interval is set to 3 seconds, and a random channel is picked up to send a beacon when the interval expires. A neighbor is considered to be dead if for a significant long time no packets or beacons are received from it. The neighbor timeout time is set to 2 minutes in current implementation.

D. Channel Schedule

Dichotomy schedules channel time in *slot*. Note that unlike previous work, the concept of *slot* is only a time unit. It does not imply that every slot has the same length or nodes need to have their slots' boundaries aligned. We assume all nodes in the network have the same set of available channels.

A hopper classifies each channel in the channel set into one of three states:

1. *Empty*. A channel is classified as empty if no anchor is found to be on that channel.
2. *Inactive*. A channel is in inactive state if at least one anchor is found but no active traffic is ongoing on that channel.
3. *Active*. If there is active traffic, either sending or receiving on that channel, it is classified as active.

A hopper generates its hopping sequence in a randomized way. That is, after a slot ends (as specified later), a hopper randomly chooses an active channel and switch the radio to that channel. If there is no active channel, the hopper will choose an inactive channel to jump into. Note that hoppers are required to visit an inactive channel periodically in order to poll an incoming traffic that initiated from an anchor. However, this is done less frequently.

Since only a hopper knows when it switches to or leaves a channel, it needs to notify its presence to nearby anchors and it is also needed to ensure that an anchor does not send packets to it after it leaves the channel. To address this issue, a light-weighted signaling protocol is defined to coordinate the action between hoppers and anchors. When a hopper comes to a new channel, it will broadcast a *prob* packet on that channel. When receiving the *prob* packet, an anchor on the channel will learn a hopper is visiting the channel. Then, the anchor will reply a *prob-ack* to the hopper and start to transmit packets to the hopper. When the anchor finishes its transmission (by finding the queue to the hopper is idle for a while), the anchor will explicitly close the link from its side by sending a *no-more* signal. Upon receiving *no-more*, the hopper knows that the anchor has finished sending. If at the same time, the hopper also has an idle queue to the anchor, then the hopper ends the current slot and schedules for a new slot on different channel. We give a maximal length of a slot to be 30ms. That is, even if the queue is always full, the anchor and hopper have to end a slot after 30ms to yield the time to serve other flows. Note that the length of slot is actually determined by the traffic. This design has two advantages compared to fixed size slots: 1) when the traffic is light, the hopper may switch channel more frequency (with shorter slots). When the load is high, the hopper switch channel less frequently and thus trades delay for capacity; and 2) when a node serves multiple heterogeneous data-rate traffic, it could divide the time with different slot size and give more transmission time to high data-rate traffic. This, however, would reduce the quantization overhead if fixed time slots are used for all traffic. In our current implementation, all these signaling packets can be piggybacked in a tiny 8-byte header attached to data packets to further reduce the overhead.

E. Packet Schedule

Packet transmissions are scheduled based on the channel schedules. As mentioned earlier, each node maintains a per-neighbor FIFO queue. Once an anchor learns a hopper's presence, it will schedule the queue that targets at that hopper. If there are more than one hopper come to the same channel and communicate with the anchor, the anchor schedules the corresponding queues in a round robin manner. When a hopper switches to a new channel, it can start to schedule the queue to the corresponding anchor immediately after sending out the broadcast *prob* packet. Packets in priority queue are always scheduled before any data packets. Since nodes in Dichotomy are located in different channels, traditional broadcast is not supported. In current implementation, we simple use multiple transmissions to emulate a broadcast.

V. EVALUATIONS

A. Test-bed Experiments

We have a 10-nodes wireless test-bed that spans on the 4th floor of our office building. These nodes are mainly located in stuff cubicles with two of them located in office and a conference room. Paths between nodes are between 1 and 5 hops in length. The data rate between each one-hop link is between 3M to 20Mbps. The experiments in this section are run on 802.11a with the channel set containing 8 channels from 36 to 64. The layout of our test-bed is illustrated in Figure 4. Each node in the test-bed is a small VIA EPIA mini-ITX box with 1.2G processor. Each node attaches a NetGear 2.4/5GHz 802.11a/g card which is based on Atheros AR5212 chipset. The NICs are operating in a modified ad hoc mode with RTS/CTS disabled by default. In this mode, the 802.11 beacon is disabled. We do so to prevent the driver to merge the nodes with same SSID into one channel. Although Dichotomy can support different routing protocols, in our test-bed, we implement a very simple source routing, which selects a shortest-path based on a pre-installed database of measured link data rate.

1) Evaluations on Chain Topologies

Figure 5 shows the throughput with and without Dichotomy over a multi-hop chain from 1 hop to 6 hops. We add a greedy UDP traffic from the same sender and measure the achieved throughput with different hops. When there are only one or two hops, Dichotomy may decrease the throughput a bit due to the system overhead. This is because

the single radio becomes the bottleneck. If the path has more than three hops, with Dichotomy, the throughput of the flow stays steadily at around 8Mbps. This is because different links of the multi-hop chain can be scheduled over different channels; thereby it mitigates interference among nearby links. However, if all nodes along a multi-hop path work on a common channel, the throughput continues to drop as the path gets longer before the spatial reuse takes place. This demonstrates that, by exploiting channel diversity, Dichotomy can effectively mitigate the interference of multi-hop path.

2) Evaluations with Multiple Flows

We test Dichotomy with multiple flows. We add four flows in the test-bed, as shown in Figure 4. One flow is along a four-hop path, two flows are one-hop and one two-hop flow crosses all three flows. The throughput of each flow is shown in Figure 6, and the right most bars in the figure show the aggregated throughput, both with/without Dichotomy respectively. The results are measured in average over 60 seconds. We see that with more flows, Dichotomy can exploit multiple channels better since there are more opportunities to schedule transmissions on parallel channels even in randomized manner. In this four flow case, Dichotomy yields a 100% throughput gain compared to the case when all nodes have to work in the same channel. Note that another benefit can be shown is that Dichotomy also improves the network fairness. This is a side-effect of spreading transmissions in different channels. By doing so, the interference on each channel is reduced. It is well understood that 802.11 MAC can show severe unfairness in congested channel in a multi-hop network [13].

B. Simulations

In the previous subsection we evaluated our actual Dichotomy implementation in an indoor test-bed environment. However, given the limited resource we have, we are not able to evaluate our system in a very large scale setting. Moreover, we could not compare our system to some previous multi-channel designs, which may be difficult to be implemented with existing software/hardware. We address these issues by conducting packet-level simulation on NS2.

In our simulations, we randomly place 100 nodes in different sized area (200x200m, 500x500m and 800x800m) to generate networks with different densities (dense, median, and sparse) respectively. The transmission range is set to 100m. We randomly place 30 flows in the network and we vary the load of each flow dynamically. The packet size of each flow is 1024 bytes. In our evaluation, we mainly compare Dichotomy with SSCH [2]. This is because SSCH is a sort of Parallel Rendezvous approaches, which are known to perform best among a wide range of multichannel solutions [14]. Secondly, SSCH shares similar design principles with Dichotomy that use localized and randomized algorithms in channel and packet scheduling. So it is fair to compare them. In SSCH, each node can change its slot schedule independently, and it applies opportunistic synchronization with its neighboring nodes. Note that SSCH relies on strict global time synchronization of all nodes, while Dichotomy does not require any time synchronization.

In our simulation, we set the channel switching overhead to be 570us as stated in Section IV.B and the link speed is 54Mbps. For both SSCH and Dichotomy, the available channel number is 11. We also compare single channel 802.11 as a baseline. For fair comparison, we set the slot time of both SSCH and Dichotomy to be 10ms. In the simulation, we simply synchronize each SSCH node in the network via a "god", thereby we do not count the potential time synchronization overhead.

Figure 7 shows the network throughput with the increase of load, which is an average over 5 runs in 3 random topologies. As expected, both SSCH and Dichotomy yield significant throughput gain compared to single channel 802.11. When the network is dense, SSCH slightly outperforms Dichotomy. This is because that Dichotomy does increase the length of some path (from one hop to two hops). In a dense network, most of paths are very short. Therefore, increasing path length by one may introduce considerable overhead. However, such impact becomes less with the increase of path length, such as in the median and sparse network. SSCH generally has low performance when the offered load is low on multi-hop paths as shown in Figure 7 (a) and (b). This is because of two reasons. First, the slot in SSCH has fixed time. So when the traffic is low, it wastes transmission opportunities if the packets buffered are not enough to sustain the slot time. Secondly, the opportunistic synchronization could let some nodes out-of-sync with its neighbors, which also wastes transmission opportunities when the network is not highly multiplexed. Dichotomy does not have these issues as anchors usually do not change their channels. The slot time of Dichotomy is adaptive and a hopper can always communicate to a neighboring anchor anytime when it switches to the corresponding channel. When the network has been highly overloaded, these two issues

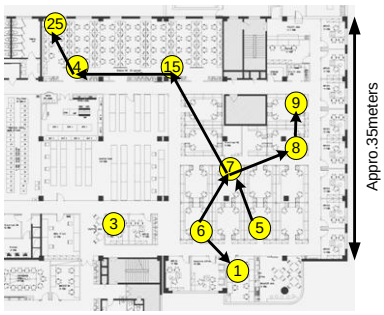


Figure 4. Layout of the test-bed.

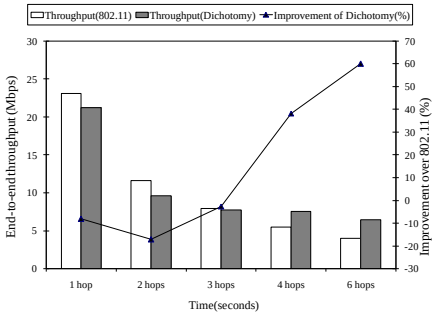


Figure 5 The performance over a multi-hop chain. We show both the throughput and the gain.

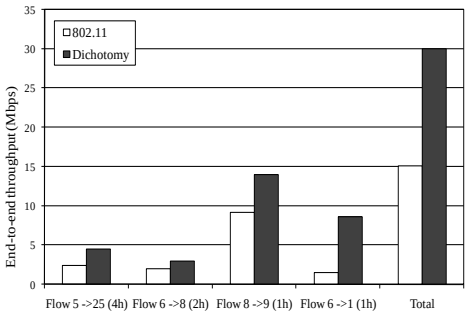


Figure 6. Throughput of multiple flows

become less significant for SSCH, where each node along a path has eventually got many packets buffered. When load is high, SSCH and Dichotomy yields similar throughput gain compared to 802.11. Note that the throughput gain in our tests does not increase proportionally to the channel used. This is because we measure the end-to-end throughput of multi-hop flows and these flows are not disjointed.

VI. RELATED WORK

There is a rich literature on using channel diversity to improve the capacity of wireless multi-hop networks. Many research focus on using multiple radios to exploit multi-channel capability [9][10][11]. But Dichotomy is designed to utilize multiple channels with a single radio.

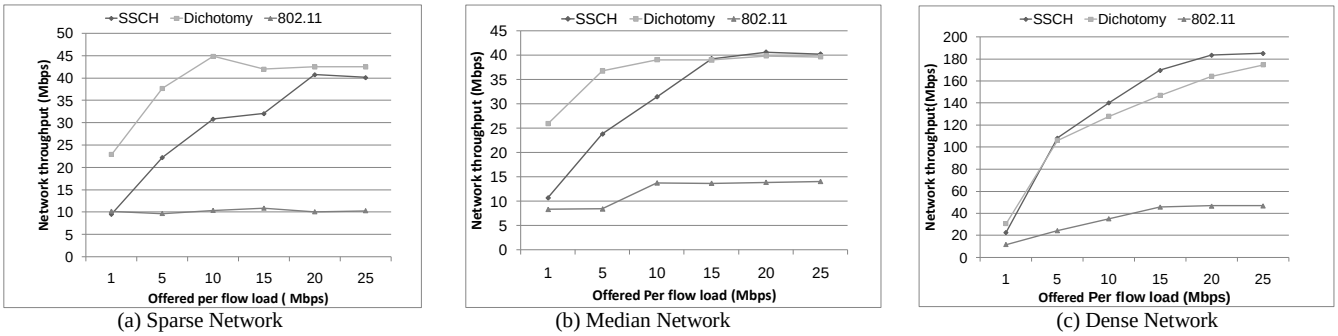
Several approaches for single radio WMNs in previous work propose enhanced MAC assuming fast channel switching is available in hardware. In [15], the authors proposed a *Receiver Directed Transmission Protocol* (RDT) in which each node returns to a quiescent channel after each transmission. If a node wants to send a frame to a neighbor, it will switch to the quiescent channel of its neighbor and transmit the frame. As mentioned earlier, RDT may cause Multichannel Hidden Terminal as well as the Deafness problem. To mitigate these problems, an extended RDT is proposed that exploits a second tone radio [16]. McMAC [5] further extends the idea with parallel rendezvous. Instead of maintaining a quiescent channel, McMAC assigns each node a home random hopping sequence. If a node wants send data to a neighbor, it will synchronize to the home hopping sequence of that neighbor. McMAC introduces random channel assignment thereby reliefs the burden of selecting the quiescent channels. But it needs pair-wise clock synchronization. Similarly, McMAC still suffers from the deafness problem. All these three schemes require packet-level channel switching. Given the practical switching overhead, these schemes may only be suitable for low speed communications. In contrast, Dichotomy schedules channel at link level. Thereby it amortizes the switching overhead even in high data rate wireless networks like IEEE 802.11a. Further, Dichotomy assigns anchors on almost fixed channels, which effectively prevents the deafness problem.

MMAC [1] and SSCH [2] are link level multichannel solutions based on IEEE 802.11. MMAC extends the 802.11 Power Saving Mode (PSM) and allocates fixed predefined time frame for control and data communication respectively. It leverages the default 100ms 802.11 super-frame as a slot. It requires tight clock synchronization among the networked nodes as communication parties need to return to control channel simultaneously in the control time to negotiate, and simultaneously jump to data channel to transmit. MMAC proposes to use IEEE 802.11 beacons to achieve the clock synchronization. However, it is well known that IEEE 802.11 TSF does not scale well with large multi-hop networks [12]. Using 802.11 TSF, MMAC schedules the channel in a coarse unit, say 100ms, which cause a few drawbacks: 1) it is possible to under utilize the channel if traffics are heterogeneous; 2) the scheduling delay is large when a node serves multiple flows; 3) it uses one channel for control, and the control channel could be a potential bottleneck.

SSCH [2], on the other hand, uses randomized channel hopping and optimistic synchronization. SSCH designs a scheduling time unit of 10ms. It assumes an 80us switching overhead, which seems to be too optimistic after considering the practical overhead like guard-time. SSCH divides time into slots and each slot has a separate pseudo-random hopping sequence, with which SSCH can synchronize to different nodes simultaneously. SSCH requires clock synchronization among all nodes and every node needs to align its slot edges with other nodes. It is not specified how to achieve so in the original paper of SSCH, and it is recently shown that achieving such global synchronization is very difficult [5].

Both MMAC an SSCH have not been implemented. But our goal is to design a practical multi-channel link-level protocol that can be implemented easily. The beauty of Dichotomy is that it removes the complexity of global clock synchronization and at the same time largely mitigates the practical switching overhead by removing the guard-time. This property actually makes Dichotomy implementable with software modifications on COTS hardware. Dichotomy schedules channels in an adaptive manner with a maximal length of 30ms, with which Dichotomy can flexibly allocate transmission time to heterogeneous traffic.

There is also a class of approaches that schedule channel at session-level, such as MCRP [7] and CBCA [8], which assign the nodes along one path to a same channel. The difference between MCRP and CBCA lies in that MCRP still allows nodes to switching channels if they are at the intersecting points of two flows; while CBCA will force all nodes to be in one channel which belong to a component formed by intersecting routes. Such approaches miss the opportunity to utilize the multi-channel within a path or a component, and therefore have least flexibility to exploit channel diversity. Moreover, although session-level approaches ease the design in link layer, they actually



(a) Sparse Network (b) Median Network (c) Dense Network

Figure 7. Network throughput vs offered load in ns2 simulation with 54Mbps channel in networks with different density.

complicate the design in routing layer. To the best of our knowledge, we are not aware of any implementation of session-based approaches in real systems.

VII. CONCLUSIONS

In single-radio 802.11 wireless mesh networks, capacity is limited if all nodes have to communicate using the same channel. To enable nodes to exploit multi-channel diversity, we design Dichotomy, which is a practical architecture that operates at link-layer and performs channel scheduling at fine granularity. Dichotomy is designed to operate without clock synchronization, using commercial-off-the-shelf hardware. We implement Dichotomy as a software shim layer lying between IP and MAC layer. Dichotomy avoids clock synchronization by strategically selecting nodes as anchors whose channels seldom change. We design a localized assignment algorithm that maximizes channel diversity.

We deploy and evaluate Dichotomy both on a real test-bed and in NS2 simulator. Our results demonstrate that Dichotomy handles practical overhead well and effectively boosts network throughput. Our experience suggests that performing multi-channel scheduling at link-level is actually practical in existing computer systems.

REFERENCES

[1] Jungmin So and Nitin H. Vaidya. A multi-channel MAC protocol for ad hoc wireless networks. Technical report, UIUC, 2003.

[2] P. Bahl, R. Chandra, and J. Dunagan. Ssch: Slotted seeded channel hopping for capacity improvement in ieee 802.11ad-hoc wireless networks. In MobiCom, 2004.

[3] S.-L. Wu, C. Y. Lin, Y. C. Tseng and J. P. Sheu. A New Multi-Channel MAC Protocol with On-Demand Channel Assignment for Mobile Ad Hoc Networks. I-SPAN 2000.

[4] H. So, G. Nguyen, and J. Walrand. Practical Synchronization Techniques for Multi-Channel MAC. In MobiCom 2006.

[5] H. So. Design of a Multi-Channel Medium Access Control Protocol for Ad-Hoc Wireless Networks. UCB/EECS-2006-54, 2006

[6] A. Tzamaloukas and J.J. Garcia-Luna-Aceves. Channel-Hopping Multiple Acess. In IEEE ICC 2000.

[7] P. Kyasanur and N. Vaidya. A routing protocol for utilizing multiple channels in multi-hop wireless networks with a single transceiver. In *International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks (QSHINE).* Aug 2005.

[8] R. Vedantham, S. Kakumanu, S. Lakshmanan and R. Sivakumar. Component Based Channel Assignment in Single Radio, Multi-channel Ad Hoc Networks. In Mobicom 2006.

[9] A. Raniwala T. Chiueh. Architecture and Algorithms for an IEEE 802.11-Based Multi-Channel Wireless Mesh Network. Infocom 2005.

[10] K. N. Ramachandran, E. M. Belding, K. C. almeroth, and M. Buddhikot. Interference-aware Channel Assignment in Multi-radio Wireless Mesh Networks. IEEE Infocom 2006.

[11] P. Kyasanur and N. Vaidya. Routing and interface assignment in multi-channel multi-interface wireless networks. Technical Report, 2004.

[12] J. Elson and K. Romer. Wireless Sensor Networks: A New Regime for Time synchronization. ACM SIGCOMM CCR, 2002.

[13] S. Xu and T. Saadawi. Does the IEEE 802.11 MAC Protocol Work Well in Multihop Wireless Ad hoc Networks? IEEE Communications Magazine, 2001.

[14] J. Mo, H. So, and J. Walrand. Comparison of multi-channel mac protocols. International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems, October, 2005.

[15] N. Shacham and P. King. Architectures and performance of multichannel multihop packet radio networks. IEEE Journal on Selected Areas of Communication, SAC-5(6):1013–1025, 1987.

[16] R. Maheshwari, H. Gupta and S. Das. Multichannel MAC Protocols for Wireless Networks. IEEE SECON 2006.

[17] K. Tan, H. Wu, L, Li, Q. Zhang, Y. Zhang. Dichotomy: A Practical Architecture for Multi-channel IEEE 802.11 Multi-hop Networks. http://www.research.microsoft.com/~kuntan/index_files/dichotomy-tr.pdf

APPENDIX

A. Proof of Theorem 1

The reduction is from vertices coloring, which is to decide whether a given Graph can be K-colorable. We construct a graph $G' = (V', E')$ as follows. For each node u, we add u, p_u to V'. We add (u, p_u) to E'. For each edge $(u, v) \in E$, we add a node n_{uv} to V', and add edge (u, n_{uv}), (v, n_{uv}) to E'. Assume each edge e has a capacity $c(e)$. For each node u, we create a flow destined to its p_u with data rate equals the edge capacity. We assume two edges interfere with each other iff an edge exists between the two transmitters. Since there is only one path between u and p_u, traffic will be sent directly. In order for all flows to be active simultaneously at all times, for all $(u, v) \in E$, they must communicate with p_u and p_v using different channels respectively. Such an edge channel assignment corresponds to a valid Dichotomy assignment. For example, for all $u \in V$, assign all p_u as hoppers, and assign all u as anchors which takes the channel used in (u, p_u) communication. Further assign all n_{uv} nodes to hoppers. It is easy to see that the two properties are satisfied. Thus, a Dichotomy assignment that achieves the maximum throughput $c(e)$ for each flow on edge $e = (u, p_u)$ exists iff G is K colorable. ■

B. Proof of Theorem 2

Due to the space limitation, we only sketch the proof here. We need to show the following three cases cannot happen.

1. Suppose hopper and anchor assignment converges. However, there exists a node whose channel assignment keeps changing;
2. Suppose there is a node whose role keeps changing, i.e. from hopper to anchor to hopper.
3. Suppose there is no assignment changes committed. However, there exists at least one node who keeps declare a role change, then aborts.

We proof by contradiction. The intuition for the convergence is that, once invariant ℓ is satisfied at a node. The only changes might happen is Rule 5 and Rule 6 which tries to prevent either too many anchors or hoppers. As we maintain the invariant and impose ordering on which node should change first, Rule 5 and 6 does not result in loops. For a complete proof, please refer [17] ■

C. Proof of Theorem 3

We look at each edge $(u, v) \in E_0$. There are the following cases:
(1) one is a hopper, the other is an anchor; (2) both are hoppers; (3) both are anchors with the same channel; (4) both are anchors with different channels. For the first and third case, $(u, v) \in E$. For case (2), there must be a common neighbor w such that w is an anchor. Otherwise, according to Rule 3, u or v will change to an anchor. This contradicts to the fact that the assignment is converged. Thus, for case (2), u, v is connected by a 2-hop path (u, w) and (v, w) in E. For case (4), they must share a common hopper w. Otherwise, according to Rule 4, u or v will change channels. This contradicts to the fact that the assignment is converged. Thus, u, v is connected by a 2-hop path (u, w) and (v, w) in E. Since each edge in G_0 is connected either directly in G or through a 2-hop path, this completes the proof. ■