Bringing Cross-Layer MIMO to Today's Wireless LANs

Swarun Kumar[†] Diego Cifuentes[†] Shyamnath Gollakota[⋆] Dina Katabi[†]
[†]Massachusetts Institute of Technology [⋆]University of Washington
{swarun, diegcif, dk}@mit.edu gshyam@cs.washington.edu

ABSTRACT

Recent years have seen major innovations in cross-layer wireless designs. Despite demonstrating significant throughput gains, hardly any of these technologies have made it into real networks. Deploying cross-layer innovations requires adoption from Wi-Fi chip manufacturers. Yet, manufacturers hesitate to undertake major investments without a better understanding of how these designs interact with real networks and applications.

This paper presents the first step towards breaking this stalemate, by enabling the adoption of cross-layer designs in today's networks with commodity Wi-Fi cards and actual applications. We present OpenRF, a cross-layer architecture for managing MIMO signal processing. OpenRF enables access points on the same channel to cancel their interference at each other's clients, while beamforming their signal to their own clients. OpenRF is self-configuring, so that network administrators need not understand MIMO or physical layer techniques.

We patch the iwlwifi driver to support OpenRF on off-the-shelf Intel cards. We deploy OpenRF on a 20-node network, showing how it manages the complex interaction of cross-layer design with a real network stack, TCP, bursty traffic, and real applications. Our results demonstrate an average gain of 1.6× for TCP traffic and a significant reduction in response time for real-time applications, like remote desktop.

Categories and Subject Descriptors C.2.2 [**Computer Systems Organization**]: Computer-Communications Networks

Keywords MIMO, Cross-Layer, Wireless, SDN

1. INTRODUCTION

Recent years have witnessed the boom of cross-layer wireless designs like SAM [28], Jello [32], MegaMIMO [23], WhiteRate [22], TIMO [10], SoftPHY [31], and many others [16, 29, 6, 11]. Instead of treating the physical layer as a black box, these systems jointly optimize network protocols and physical-layer signal processing. Collectively, they have created a rich literature of cross-layer designs that are implemented in software radios and have shown large throughput gains. Unfortunately, hardly any of these ideas have made it into Wi-Fi chips or real networks. Indeed, a form of stalemate exists: The research community is waiting for cross-layer innovations to be implemented in Wi-Fi hardware so that they may be used in operational networks. Yet, Wi-Fi chip manufacturers cannot make expensive hardware investments without better understanding how these designs interact with real applications and real networks. The situation evokes a similar picture from 10 years ago, when Internet protocols were developed and demonstrated in small testbeds and the ns simulator, but were not adopted by switch manufacturers. Wired networks broke this cycle by building innovations in commodity hardware and directly deploying them in operational networks. OpenFlow, Ethane, and the body of work that led eventually to software defined networks (SDNs) have pioneered this path. We believe that, similarly, cross-layer research needs to start targeting commodity Wi-Fi cards, actual applications, and today's networks.

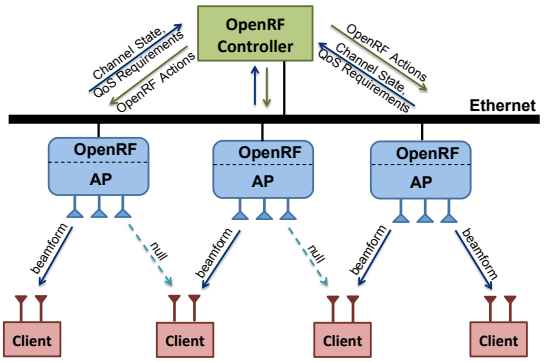


Figure 1—Architecture of OpenRF. OpenRF provides an interface to physical-layer MIMO signal processing, e.g., interference nulling, interference alignment, and beamforming. The OpenRF Controller coordinates network devices through this interface.

This paper takes the first step towards this goal. It presents OpenRF, a cross-layer architecture for MIMO interference management. OpenRF resides on Access Points (APs) and enables them to control MIMO signal processing at the physical layer. Specifically, OpenRF provides the following capabilities:

- It enables commodity Wi-Fi APs to perform three MIMO interference management techniques: interference nulling [11], coherent beamforming[23] and interference alignment[16].
- It is self-configuring. Network administrators need not understand MIMO signal processing and when to apply a particular interference management technique. OpenRF automatically infers the interference layout of the network, and dynamically applies the right combination of interference nulling, alignment, or beamforming, wherever they are beneficial.
- It translates high-level quality of service requirements of downlink traffic to low-level physical-layer MIMO techniques. For instance, an administrator may request OpenRF to guarantee a min-

SIGCOMM'13, August 12–16, 2013, Hong Kong, China.

imum rate to real-time applications in the network (e.g., remote desktop or VOIP). OpenRF employs MIMO signal processing to control interference across APs and deliver the desired rate.
- Finally, OpenRF is fully compatible with commodity 802.11n cards that implement transmit beamforming, an 802.11n optional feature. We have built OpenRF as a patch to the iwlwifi driver for Intel 5300 cards, providing researchers and network administrators the ability to deploy it in their networks to experience the benefits of cross-layer MIMO, or experiment with new physical-layer MIMO designs.

Architecturally, OpenRF borrows from the SDN design, in that it separates the control plane from the data plane and exposes functions that have traditionally been deeply hidden in the network stack, to higher layers. As illustrated in Fig. 1, the data plane is controlled by the *OpenRF interface*, which resides on access points and shields other components from how signal processing techniques are implemented at the device level. Analogous to the OpenFlow interface, the OpenRF interface operates over a table of (FlowID, Actions) tuples. In contrast to OpenFlow entries, where an action may identify which port to transmit the flow on, here, an action specifies the relative power used to transmit the flow on each of the AP's antennas. This is typically referred to in MIMO terms as the pre-coding vector of the flow. Just as forwarding steers a flow's packets toward a particular route in the wired network, pre-coding steers the PHY signal and creates a beam that propagates along a particular spatial direction, allowing the system to null interference at an unwanted receiver and focus the power on the desired receiver.

The control plane in our design is managed by the *OpenRF Controller.* It is configured with per-flow quality of service (QoS) requirements (e.g., a desired rate), as well as which PHY actions are supported by the OpenRF interface. It also periodically takes as input channel measurements from the APs. Using this information, the controller maps the QoS requirements into PHY actions like nulling, beamforming, or alignment. The controller then fills up the OpenRF table with these actions mapped to various flows, so that the needs of each flow can be satisfied.

A key challenge in OpenRF is the need to ensure the whole network stack operates reliably, while performing MIMO interference management. In other words, OpenRF has to account for TCP and application burstiness and the resulting dynamism in the interference patterns. Another challenge stems from the interaction between physical layer techniques and the 802.11 protocol. In particular, PHY techniques achieve gains by enabling concurrent transmissions using intelligent interference management. However, the 802.11 protocol is designed under the assumption that transmitters in the same interference region should not transmit concurrently. This assumption manifests itself in its handling of end-to-end reliability and related functions such as carrier sense, acknowledgments and retransmissions. OpenRF has to ensure concurrent transmissions at the PHY-layer without compromising 802.11's reliability. In §4 and §5 we explain these challenges in detail and we describe how OpenRF addresses them.

We have built a prototype of OpenRF on Intel 5300 Wi-Fi adapters. We patched the iwlwifi driver to enable nulling, alignment, and beamforming, separately and combined. We deployed a 20-node network, where six of the nodes act as APs and the rest are clients. We compared the performance of the network with and without OpenRF. Our results show the following.

- In a network of 20-nodes, and for a random setup of TCP and UDP flows, the aggregate performance gain in terms of throughput of UDP and TCP flows are 1.7× and 1.6× respectively.
- We also evaluate the impact of OpenRF on the quality of real-time applications. In particular, we evaluate rich applications (multimedia) over Remote Desktop, which is increasingly common in the enterprise. Our results show that OpenRF reduces the percentage of screen glitches in a VNC Remote Desktop client by 6×. It also reduces the 90^{th} percentile delay for VNC by 4×.
- Commodity 802.11n cards can perform beamforming to improve the average SNR at the receiver by 3 dB. Beamforming also leads to a flatter receiver SNR profile across OFDM bins. These reasons cause the bit-rate adaptation algorithm to jump to the next rate 80% of the time, thereby enhancing throughput. The cards can also perform interference nulling and interference alignment to reduce the average interference-to-noise ratio (INR) at the receiver by 12 dB and 11 dB respectively. This enables concurrent transmissions provided the interference is below 10-15 dB.

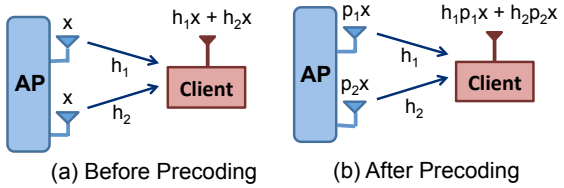


Figure 2—Example Topology. We show how a 2-antenna AP can precode its signal to a 1-antenna client.

Contributions: OpenRF is the first system that deploys physical-layer MIMO techniques (i.e., nulling, alignment, beamforming) on commodity Wi-Fi cards. OpenRF dynamically applies the right set of these MIMO techniques to suit any topology or traffic pattern. It is also the first cross-layer design that is demonstrated using a fully operational network stack with real applications. As such, it takes cross-layer design all the way from the physical layer to the application layer, and opens up an opportunity for these technologies to make impact on today's networks.

2. MIMO PRIMER

In this section, we provide a brief introduction to basic MIMO interference management techniques. To understand these techniques, it is important to know the following fundamental properties of MIMO transmissions [16, 35]:

- *An n-antenna node receives signals in n-dimensional space.* For example, a 1-antenna client receives a signal at only one antenna; so it receives signals in one-dimension. Similarly, a client with two antennas receives a signal on both of its antennas. So, the received signal is a vector in a 2-D space.
- *An n-antenna node transmits signals in n-dimensional space.* For e.g., a 3-antenna AP transmits a 3-D vector.
- *n-antenna receivers can decode up to n concurrent signals.*
- *A transmitter can use precoding to change how its signal is received at a particular node.* To do so, it multiplies the transmitted signal by a pre-coding matrix P. The pre-coding matrix can be chosen to null (i.e., cancel) the signal at a particular receiver, beamform the signal, or align it along some space. Below, we explain how these three MIMO techniques leverage precoding.

2.1 Interference Nulling

Interference nulling allows a transmitter to completely cancel (i.e., null) its signal at a receiver. For example, suppose a 2-antenna AP wants to null its signal x at a 1-antenna receiver as shown in Fig. 2(a). Say the AP sends x on both of its antennas. Let the channels from the two transmitting antennas to the receiver's antenna be h_1 and h_2. The receiver obtains the signal $h_1x + h_2x$. So, how can the AP precode its signal so that the received signal is zero?

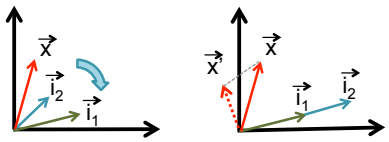


Figure 3—Interference Alignment at a 2-Antenna Receiver. The AP uses interference alignment to rotate the interfering signal i_2 along i_1, making the two interferers seem as if there were one.

Say, the AP pre-multiplies the signal at its first antenna by a constant p_1 and second antenna by another constant p_2. The new received signal is now $h_1p_1x + h_2p_2x$, which needs to be 0. Clearly, this can be solved easily, for example, by setting $p_1 = -h_2$ and $p_2 = h_1$.[1] Thus, the AP can now safely transmit without causing any interference at this client.

We can generalize nulling for a multi-antenna client by using a precoding matrix P such that $HP = 0$, where H is the channel matrix from transmitter to receiver [16, 11].

2.2 Coherent Beamforming

Coherent beamforming helps an AP maximize its signal at a receiver, i.e., increase the SNR. Let us revisit the example in Fig. 2, where this time, the 2-antenna AP wants to beamform its signal at the 1-antenna receiver. As before, the received signal is $h_1x + h_2x$. So, can the AP further increase the power of the received signal?

Fortunately, the AP can indeed do so by applying precoding once again, this time so that the signals from the two antennas add up constructively. In other words, received signal, $h_1p_1x+h_2p_2x$, needs to be maximized. Since h_1p_1 and h_2p_2 are complex numbers, to maximize their sum we need to choose p_1 and p_2 to align the two complex numbers so that they have identical phases. This can be done by choosing $(p_1, p_2) = (h_1^*, h_2^*)$.

In general, a multi-antenna AP can beamform its streams coherently at a single-antenna client using a precoding matrix $H^*/||H^*||$, where H^* is the conjugate transpose of the channel matrix. [23, 35].

2.3 Interference Alignment

Suppose a 2-antenna receiver receives two signals - a desired signal along $\vec{x}$ and an interfering signal along $\vec{i}_1$. As mentioned before, these signals can be represented as vectors in a 2-D space, as in Fig. 3. Since a 2-antenna receiver can decode two concurrent signals, it can discard the interfering signal, to obtain its desired signal. However, if another interferer joins the network, the receiver can no longer decode, since the 2-D space has a third vector $\vec{i}_2$, as in Fig. 3. How can the transmitter of $\vec{i}_2$ avoid interfering at the receiver?

Interestingly, the transmitter of $\vec{i}_2$ can leverage *interference alignment* and precode its transmission to rotate the vector $\vec{i}_2$ and align it along the same direction as $\vec{i}_1$. In effect, the receiver now obtains only two vectors, the desired signal along vector $\vec{x}$, and the sum of interferences $\vec{i}_1 + \vec{i}_2$ along a different direction, as shown in Fig. 3. Thus, it can easily decode by treating the unwanted interference $\vec{i}_1 + \vec{i}_2$ as one signal from a single antenna, and projecting $\vec{x}$ orthogonal to the interference.

In general, a multi-antenna AP can align its signals along the vector space V using a precoding matrix P, such that, $V^{\perp}HP = 0$, where $(.)^{\perp}$ denotes the orthogonal vector space [16, 11].

Note that since all the above techniques leverage MIMO precoding, they can be combined in different ways by a transmitter that has a sufficient number of antennas.

[1]One still needs to normalize to ensure the power after pre-coding sums up to the transmit power budget. For clarity however we ignore normalization, assuming that the transmitter always normalizes its signal before transmission.

VLAN ID	Ethernet			IP			TCP		Precoding Space	
	Src	Dest	Type	Src	Dest	Proto	Src	Dest	Coherence	Interference

Figure 4—OpenRF Flow Table. OpenRF defines flows based on packet headers, similar to OpenFlow.

3. OpenRF's Design Principles

OpenRF provides a general framework for applying MIMO PHY techniques in an Enterprise WLAN. It resides at the APs and can perform interference nulling and coherent beamforming without modifications to the clients. It can also perform interference alignment by patching the client's driver as explained in §5.3.

OpenRF is compatible with commodity 802.11n cards that implement transmit beamforming, an 802.11n optional feature.[2].

OpenRF's design is based on the realization that cross-layer interference management techniques can be decoupled into two classes: *coherence* techniques and *interference* techniques. Coherence techniques aim to maximize the signal strength at a receiver, e.g., coherent beamforming (See §2.2). In contrast, interference techniques allow APs to transmit additional concurrent streams, provided these streams do not interfere with existing parallel transmissions. Interference techniques include nulling, and alignment (See §2.1 and §2.3). Interestingly, coherence techniques can be performed completely locally, as they only involve a single AP transmitting to its own clients. In contrast, interference techniques need APs to synchronize with other APs, so that they do not interfere at each other's clients, while they transmit concurrently on the shared wireless medium. Thus, OpenRF's first policy is: *only interference techniques need to be coordinated (i.e., scheduled) by the central controller.* This rule reduces coordination complexity.

Second, just like OpenFlow steers a flow's packets toward a particular route, OpenRF steers the signal of each flow along a particular spatial direction. This is done by precoding the signal before transmission. However, unlike OpenFlow, OpenRF does not directly assign a precoding vector (or precoding matrix) to each flow. This is because MIMO APs need to combine transmitting flows *coherently* to their clients, along with canceling *interference* at other APs' clients. Fixing the precoding vector for a particular flow would also fix the set of flows that can be concurrently transmitted with that flow. However, due to traffic burstiness, the set of flows which need to be combined together changes frequently depending on which flows have pending packets in the queue. Thus, instead of assigning a precoding vector per flow, OpenRF assigns to each flow an *interference vector* and a *coherence vector*. Now suppose the APs consider a set of flows for concurrent transmission depending on which flows currently have pending packets in the queue. At each AP, OpenRF can now combine, in real time, the relevant coherence and interference vectors, to produce a precoding vector that enables transmitting the desired flow coherently, while nulling any interference that may affect other flows. Hence, OpenRF's second policy is to perform *late binding of interference and coherence decisions*.

In the following section, we explain the above policies and their implementation in greater detail.

4. OpenRF's Architecture

OpenRF is architecturally similar to SDN in that it separates the control and data planes, and exposes functions that have traditionally been deeply hidden in the network stack to higher layers. The data plane is controlled by an open interface that is managed in soft-

[2]The Transmit Beamforming feature allows OpenRF to set precoding vectors (or matrices) on 802.11n cards, with certain restrictions, as explained in §5.3.

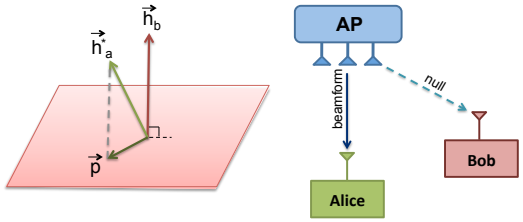


Figure 5—Computing the precoding vector. Suppose an AP needs to beamform its signal to Alice, while nulling interference at Bob. Then the precoding vector $\vec{p}$ is the projection of coherence vector $\vec{h}_a^*$ onto the plane orthogonal to interference vector $\vec{h}_b$.

ware by a controller. In this section, we describe the architecture of both the OpenRF Interface and the OpenRF controller.

4.1 OpenRF Interface

The OpenRF Interface operates over a table indexed by flows, which describes how the AP handles different flows in the network. Flows are identified based on fields in the packet header, such as destination IP address, port number etc. (Fig. 4). Additionally, OpenRF maintains for each flow, a coherence vector, which specifies the direction along which the signal is received coherently at the client, and an interference vector which specifies the direction that we need to be orthogonal to in order to avoid interference at this client.

The concept of an interference vector and a coherence vector (or more generally an interference matrix and a coherence matrix) is best explained via an example. Consider the scenario in Fig. 5, where a 3-antenna AP wishes to beamform its signal to its client, Alice, while nulling interference at Bob, the client of a different nearby AP. Let the channels to Alice be h_{a1}, h_{a2}, and h_{a3}, and the channel to Bob be h_{b1}, h_{b2}, and h_{b3}, as shown in the figure. We can express these channels as two 3-dimensional vectors, $\vec{h}_a$ and $\vec{h}_b$.

Now, any pre-coding vector that the AP computes must satisfy the following two conditions:

- **Interference Condition:** To null to Bob, the AP has to pre-code its signal such that it falls in the pink-colored plane orthogonal to Bob's channel vector, as shown in Fig. 5. Specifically, let $\vec{p}$ be the pre-coding vector that the AP applies to its signal. Then to null to Bob, we need $\vec{h}_b \cdot \vec{p} = 0$.
- **Coherence Condition:** The AP also wants to beamform its signal to Alice. In the absence of the nulling constraint, beamforming requires the AP to align its transmission with Alice's channels, i.e., $\vec{p} = \vec{h}_a^*$. However, this pre-coding vector creates interference at Bob as it does not satisfy the nulling condition $\vec{h}_b \cdot \vec{p} = 0$, thereby making concurrent transmissions to Alice and Bob infeasible. Thus, the AP's best option is to pick a precoding vector that satisfies the nulling condition but is as close as possible to $\vec{h}_a^*$. To do so, the AP projects $\vec{h}_a^*$ on the plane that nulls the signal to Bob, as shown in Fig. 5.

In the above example, Bob's interference vector is his channel $\vec{h}_b$, whereas Alice's coherence vector is the conjugate transpose of her channel $\vec{h}_a^*$. The AP can keep these vectors in the OpenRF table along with the entries of Alice and Bob. Whenever it wants to null to Bob and beamform to Alice, it uses these vectors to compute the required precoding vector $\vec{p}$ and applies it to the transmitted signal. The AP may also combine nulling to Bob with beamforming to a client other than Alice, say, Charlie. To do so, it only needs to combine Bob's interference vector with Charlie's coherence vector. These decisions can be made in real-time depending on which clients have packets pending at the AP.

While the above example deals with single-antenna receivers, it can readily be generalized to typical multi-antenna MIMO systems that leverage MIMO multiplexing. In the following section, we formalize our definitions of the coherence and interference vectors and the computation of the precoding vector. Note that as we generalize to multi-antenna MIMO clients, the vectors become matrices.

4.2 Formalizing the Precoding Computation

Suppose an n-antenna AP needs to transmit k independent MIMO streams $X_{k\times 1}$ to an m-antenna client, where $k \leq m$. Let $H_{m\times n}$ be the channel matrix, where $m \leq n$. The AP applies a precoding matrix $P_{n\times k}$ so that the received signal: $Y = HPX$.

The AP can choose the precoding matrix P to satisfy certain interference and coherence conditions. We define:

- **Coherence Matrix:** The coherence matrix C_i for flow i identifies the space along which the signal should be transmitted to increase the SNR at the client. For coherent beamforming to a single antenna client, we define $C_i = H_i^*/||H||$, where H is the channel matrix to the client. More generally, we define $C_i = Eig_k(H^*H)$, where $Eig_k(M)$ denotes the k eigen vectors of M with the largest eigen values.
- **Interference Matrix:** The interference matrix I_i for flow i identifies the space to which the signal should be orthogonal in order to avoid interference at the client. For interference nulling $I_i = H_i$. For interference alignment where the client is aligned along the direction (or, more generally, space) specified by the vector (or, matrix) V_i, $I_i = V_i^{\perp} H_i$.

Note that the above matrices allow for beamforming, and nulling/alignment to MIMO clients that receive multiple streams.

Computing the Precoding Matrix: Now that we have defined coherence and interference matrices, we need to explain how OpenRF computes the precoding matrix, in real-time, for an arbitrary set of concurrent flows. Assume that the OpenRF controller (described in §4.3) has already populated the interference and coherence matrices for a set of flows in the OpenRF table. Suppose the controller decides that the AP needs to transmit a flow 1, while concurrently canceling any interference caused to concurrent flows: $2, 3, \ldots, n$.

Let C_1 be the coherence matrix of flow 1 in the OpenRF table, and $I_2, \ldots, I_n$ be the interference matrices of flows $2, \ldots, n$. OpenRF first computes the combined interference matrix I by concatenating these interference matrices, i.e. $I = [I_2 \ \ldots \ I_n]^T$. The matrix I denotes the space that the signal should be orthogonal, to avoid interference at all of flows: 2, ..., n. Intuitively, OpenRF now needs to project its coherence space C_1, orthogonal to the interference space I. This can be interpreted as a standard geometric problem, similar to Fig. 5, but generalized to n-dimensional space. OpenRF computes the solution as the precoding matrix: $P = I^{\perp}(I^{\perp})^* C_1$, where $I^{\perp}$ denotes the null-space of matrix I.

4.3 OpenRF Controller

The OpenRF controller has two components: a central component that coordinates all APs, and a local component residing on each AP that adapts to real-time changes to channels and traffic patterns. We refer to this local component as the *local agent*. As argued in §3, OpenRF's key design principle is that the central controller manages only interference spaces across APs, and delegates managing the coherence space to the local component at each AP.

To schedule concurrent transmissions across APs, we divide time into slots. Having short slots corresponding to the size of a packet can cause excessive overhead. Hence, OpenRF leverages 802.11n's aggregate frames, which combine multiple MAC-layer packets into a single PHY-layer frame from the perspective of medium access.

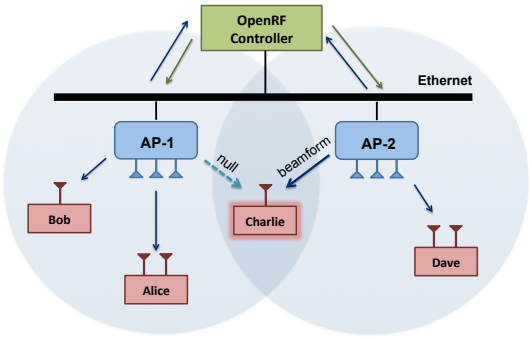


Figure 6—Illustrative Example. In this example, there are two 3-antenna APs: AP-1 and AP-2, and four clients.

This allows us to set the slot size corresponding to the size of an aggregate frame, which in our system defaults to 5 ms.

The central controller designates some slots for scheduling edge clients where two or more APs interfere.[3] Other slots are scheduled locally, by the local agent at each AP. This limits coordination overhead across APs to only these centrally scheduled slots, which we call the *interference slots*.

Once the central controller assigns the interference slots, the local agents on the APs abide by the following contract: (1) Any flow that suffers interference from other APs (i.e. flows to edge clients) can only be transmitted in its own interference slot. (2) Interfering APs that wish to concurrently transmit during this slot *must* perform the interference management technique recommended by the central controller, i.e., interference alignment or nulling. The local agents, however continue to have the flexibility to schedule any flow locally based on the dynamic traffic patterns, in any slot, provided the flow does not suffer from interference.

To illustrate the above rules, let us consider the simple example in Fig. 6. Here, AP-1 has two clients Alice and Bob, AP-2 has two clients Charlie and Dave. The pale blue circles show the interference range of each AP. For simplicity, we assume that all channels support the same rate and that there are no QoS requirements. Suppose the controller assigns every alternate slot as interference slots for edge client Charlie. During such a slot AP-1 and AP-2 follow these policies: (1) AP-1's contract is to null its signal to Charlie during all of his interference slots. However AP-1 is free to transmit concurrently to either Alice or Bob, depending on who has traffic. (2) AP-2's contract is that it can transmit to Charlie concurrently to AP-1 only in its designated slots (We explain how to selectively enable concurrent transmissions in 802.11 in §5). However, AP-2 is free to transmit to David, in either slot, concurrently with AP-1.

In the following paragraphs, we explain how the central controller and local agent function.

Local Agent. The goal of the local agent is to dynamically schedule which flow an AP must transmit to during each time slot. The local agent takes as input the list of interference slots, and the QoS requirements of different flows. The controller then employs the following algorithm described in Alg. 1, inspired by deficit round robin scheduling. At a high level the algorithm maintains a *credit counter*, c_i, for each flow i. In general, the credit counter of a flow is large, if it has not been scheduled for an extended period. Now at any time slot, the local agent picks the flow f whose credit counter is the highest. It then measures the number of bytes b it could send for this flow in this slot. Finally, it updates the credit counters, by reducing the credit of this flow, based on b, and redistributing this equally among all other flows. In effect, this ensures that the medium is shared fairly between all flows.

[3] Edge clients can be identified by checking if the SNR based on the channel matrices of APs is above a nominal threshold.

1 Pseudo-code for the Local Agent Algorithm

1: **function** LOCALAGENT(t, *qos*, *slots*)
2: ▷ t: Slot, *qos*: QoS needs, *slots*: Interference Slots
3: **if** $t \in$ *slots* **then** ▷ t an interference slot
4: f = Flow for slot t (if flow has packets to send)
5: **else** $f = arg[max_i(d_i)]$ ▷ f: Flow with most deficit
6: **end if**
7: $d_f = d_f - b$ ▷ d_f: deficit, b: no. of bytes sent
8: $b_f = b_f + b$ ▷ b_f: no. of bytes sent so far by f
9: **for** each flow i **do**
10: ▷ e_i: Expected no. of bytes for QoS or BE flows
11: $e_i = qos_i * t$, if QoS (or) $avg_{k \in BE}(b_k)$, if BE
12: $w_i = e_i / b_i$ ▷ w_i: weight for flow i
13: $d_i = d_i + b * w_i / \sum_k w_k$ ▷ Update deficits
14: **end for**
15: **return** f
16: **end function**

However, to support OpenRF, we need the following additional modifications: First, consider an interference slot, where other APs null interference for flow f, belonging to this AP. In such a slot, the AP always picks flow f, if f has pending packets in the queue.

Second, the local agent enforces two kind of QoS requirements: a proportional allocation of throughput, and fixed reservation of throughput. These requirements are configured by the network manager, for flows identified by their packet headers. Note that we do not consider delays in our current system. Proportional allocation requires the flows of an AP to achieve throughputs proportional to some set of weights: $\{w_i,$ for all flows $i\}$. This can naturally be incorporated into the local agent's algorithm. Specifically, instead of redistributing credit equally between flows, we redistribute it proportionally based on weights $\{w_i\}$ (Alg. 1, Line 13). This ensures that flows are allocated precisely according to these weights.

Now that we know how to allocate throughput proportionally, how do we enforce fixed reservations? In this scenario, we have two kinds of flows: Quality of Service (QoS) flows, which have fixed throughput requirements, and Best Effort (BE) flows, must have equal (or more generally, proportional) throughput between them to ensure fairness. Interestingly, the local agent can translate the problem of fixed reservations to proportional allocation with dynamic weights. In particular, the agent computes e_i, the expected number of bytes that it hopes to send during time span t for a flow. For QoS flows, e_i is simply $qos_i * t$, where qos_i is the throughput requirement. In contrast, all BE flows must achieve equal throughput in this time, so e_i is the mean of the throughput achieved for all BE flows during time span t. (Alg. 1, Line 11) The agent now applies proportional allocation by resetting the weights w_i, in every time slot, to e_i/b_i, where b_i is the number of bytes sent so far for flow i. In effect, this biases the weights to ensure that b_i is as close as possible to e_i, hence satisfying the QoS requirements.

Central Controller. The central controller assigns interference slots that serve flows to edge clients in the network. The controller takes as input the network state, i.e. channel state information, the list of clients and their flows. Similar to the local agent above, the central controller uses credit counters to decide the number and list of interference slots, while accounting for QoS, BE or proportional throughput requirements for different flows in the network.

The controller applies the following heuristic algorithm for each AP i in the network: First, it uses credit counters, similar to the local agents, to pick flow f_i for AP i. After choosing f_i, the controller

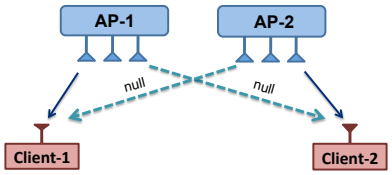


Figure 7—Interference Nulling. Here, AP-1 and AP-2 transmit concurrently to their respective clients: client-1, and client-2, by nulling interference at each other's clients.

iterates over all other APs (besides AP i), and checks if any of these APs may potentially interfere with f_i. For such APs, the controller decides how they should manage interference so that they can concurrently transmit with f_i. In particular, it prescribes interference nulling if f_i is to a single-antenna client, and interference alignment, otherwise.[4] Next, these APs are informed that the present slot is an interference slot allotted to f_i.

At this point, the controller must update the credit counters based on the throughput for the set of flows $\{f_i\}$. But recall that unlike the local agent, the central controller cannot obtain the number of bytes that have been sent in a slot. In other words, the controller needs to determine the throughput that each flow will achieve. Fortunately, this can be done fairly accurately by calculating the effective channel of each client, which is the product of the client's channel and the AP's precoding matrix, and then estimate the throughput using the ESNR algorithm [13].

Finally, the Central controller, unlike the local agent, can leverage its global view of the network to ascertain whether a flow's fixed throughput requirement can be satisfied. Whenever a new QoS flow joins the network, the controller performs the following *admission control algorithm*: It runs the above central controller algorithm with the new throughput requirement factored in. It then admits this flow as a QoS flow only if the AP achieves the requisite throughput requirement of this flow, at the end of the algorithm.

5. Integration with 802.11 Protocol

To implement OpenRF on commodity Wi-Fi cards, it must be integrated with the 802.11 protocol. Unfortunately, the 802.11 protocol was not designed with MIMO interference management techniques in mind. This causes subtle interactions between PHY-layer interference management and 802.11's MAC, as described below.

5.1 Ensuring Reliability

End to end reliability is essential for the correct operation of TCP and most applications. WLANs achieve reliability using 802.11's carrier sense, acknowledgments and retransmissions, which together provide an abstraction of a reliable communication channel to higher layers of the network stack. Unfortunately, these three reliability mechanisms do not lend themselves naturally to concurrent transmissions in the same interference region, which are essential for most PHY-aware techniques to achieve throughput gains.

(a) Carrier Sense: Consider the example in Fig. 7, where we have two APs, each transmitting streams to their own clients, while nulling interference at each other's clients. If carrier sense is disabled on the two APs, OpenRF will enable these APs to transmit concurrently in the same interference region and correctly deliver packets to their own clients (as described in the previous section). However, in the presence of carrier sense, one of the APs will begin transmitting before the other, causing the other AP to carrier sense

[4]Sometimes, the AP may not have enough antennas to simultaneously null/align to multiple flows. In such cases, this AP is not permitted to transmit to any of its clients during this interference slot.

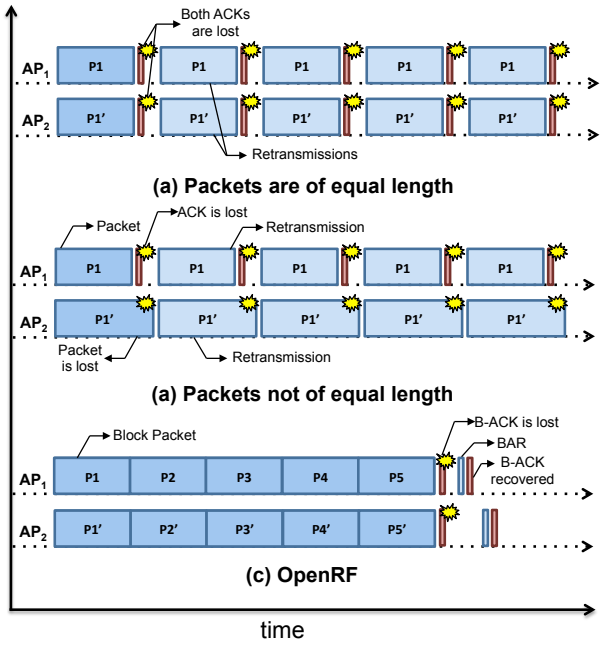


Figure 8—Collision of MAC-Layer ACKs. In Case (a), packets are of equal length, causing ACKs to repeatedly collide with each other. In Case (b), packets are of unequal lengths causing ACKs repeatedly to collide with packets. In contrast, OpenRF ensures that the system recovers from any MAC-layer ACK collisions.

the signal and abstain from concurrently transmitting. Thus, to provide concurrent transmissions at the physical layer, we need to make the two APs carrier sense only when appropriate. Of course, one option is to deactivate carrier sense altogether; but that would lead to severe collisions with uplink traffic.

To ensure that the two APs in Fig 7 transmit concurrently, despite carrier sense, OpenRF synchronizes their packets so that they start precisely at the same point and therefore effectively do not carrier sense each other. But how do we ensure that the two APs start exactly at the same time? Recall that 802.11 nodes decide at what time to transmit as follows: If the node senses the medium as idle, it picks a random slot between 0 and CW_{min}, and transmits starting from that slot. Hence, if we convince two nodes to pick exactly the same slot, they will never sense each other and will transmit concurrently starting from that slot. One way to achieve this is to set CW_{min} to zero, causing both APs to always transmit concurrently at the 0-slot. However, this alone is not enough, as always picking the zero slot gives the APs unfair access to the medium compared to other nodes in the network.

To recover fair medium access, we leverage the AIFS parameter in 802.11e/n standards. Instead of applying the same DIFS period to all types of traffic, 802.11e/n allows different traffic classes to replace the standard DIFS waiting period by a different waiting time referred to as the arbitrary inter frame spacing (AIFS). This enables different classes of traffic to have varying levels of priority. In particular, we leverage AIFS as follows: Whenever two APs have to transmit concurrently, we set their AIFS period to a fixed value of $DIFS + CW_{min}^{def}/2$, where CW_{min}^{def} is the default value for CW_{min}. In addition, we set CW_{min} to zero, so that both APs always send concurrently, precisely after AIFS. Together, these two mechanisms ensure that the two APs transmit concurrently in slot zero, but as their AIFS is longer than DIFS, it will look to other nodes as if they picked the middle slot $CW_{min}^{def}/2$, which is precisely what we need for average fairness. Note that APs can send flows that are not

meant to be transmitted concurrently with other APs, independently using standard AIFS and contention windows.

(b) MAC ACKs and Transmission: The above solution allows us to synchronize the two access points and make them transmit their data packets concurrently. However, since 802.11's acknowledgments are transmitted immediately after a packet, they will invariably cause collisions. Specifically there are two scenarios as shown in Fig. 8: (1) The concurrent packets span the same length on the wireless medium, in which case, their ACKs will collide as shown in Fig. 8(a). (2) The concurrent packets span different lengths on the medium, which causes ACKs to collide with data packets as shown in Fig. 8(b). In either case, APs do not receive acknowledgments for their data packets, and therefore they will end up retransmitting the packets again and again, eliminating any potential gain.

There are a variety of solutions to mitigate this, which are not compatible with the 802.11 standard. For example, one may propose that since the APs have multiple antennas, they may jointly decode the colliding ACKs. Unfortunately, joint decoding is not possible with the current 802.11 standard. Alternatively, one may propose applying interference nulling to ACKs on the uplink, similar to the downlink data packets. However, coordination between disparate clients is relatively difficult compared to the APs, since the clients are not connected to a wired backend.

To address the above problem, we leverage 802.11n block aggregation. Specifically, block aggregation allows you to bond multiple MAC-layer packets (MPDUs) to the same destination that span equal time durations of up to 5 ms on the medium as shown in Fig. 8(c). Note that packets may have different lengths and may be sent at different rates leading to blocks of sizes below 5 ms. So, we ensure blocks span precisely 5 ms by suffixing dummy MPDUs at the end of the block if necessary. Using block aggregation greatly reduces the number of ACKs in the system, since the client issues only a single *block acknowledgment* for all MPDUs in a block. However, the two block ACKs of concurrently transmitted blocks will continue to collide. Fortunately, when a block ACK is lost, the access point does not retransmit the entire block once again. Instead, the 802.11 standard specifies that the AP issues a short block acknowledgment request for the block ACK[1]. By sending this block acknowledgment request using standard best effort with default AIFS and contention windows, OpenRF can ensure that their is no concurrency of the block requests, therefore the block acknowledgments do not collide as shown in Fig. 8(c).

In summary, the above mechanisms ensures that packets that are transmitted concurrently are still acknowledged correctly, providing the expected level of reliability to TCP and higher-layer applications. Furthermore, the only overhead we incur to ensure reliability is the loss of up to one block ACK at most every 5 ms, which is relatively low, compared to the gains from concurrency.

5.2 Working with Unmodified Clients

PHY-aware techniques such as interference nulling, interference alignment and beamforming, all require knowledge of the channels from APs to their clients. To this end, one possible approach is to require clients to measure their channels and provide this information as feedback to the APs. However, this approach has two problems: First, it incurs high overhead. Second, many clients may not have the capability to measure these channels in the first place. For example, a single antenna client cannot measure the channels simultaneously from a 2-antenna AP. One might wonder if it suffices for the AP to transmit two individual packets: the first packet on its first antenna and the second on its second antenna. While this would allow the client to measure the channel from each transmit antenna separately, such measurements will be separated by at least

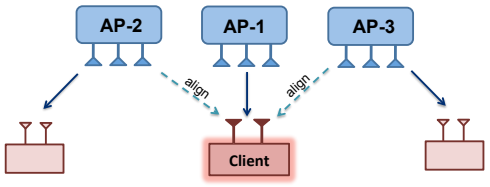


Figure 9—Interference Alignment. In this example, we have three AP-client pairs. AP-1 and AP-3 align their interfering signal to enable the 2-antenna client to decode AP-2's signal.

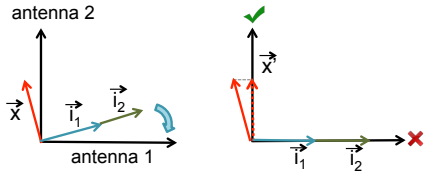


Figure 10—Enabling Alignment for Today's clients. OpenRF implements alignment for the client by aligning all interfering signals: i_1 and i_2, along antenna 1. The client disables antenna 1 to decode desired signal x on antenna 2.

hundreds of microseconds. However channels required for performing nulling or beamforming must be measured at nearly the same time, and can at most be separated by a few OFDM symbols [23].[5] So how can we perform nulling, alignment or beamforming at APs, without requiring clients to send their channel state information?

To address this problem we use the principle of channel reciprocity[11]. Reciprocity states that the channel in the forward direction is the same as the channel in the reverse direction, up to some calibration constant. In effect, APs can now use packets sent from the clients to measure the reverse channel, and then use those measurements to infer the forward channel needed for interference management. However, the AP still needs to know the calibration constant to derive the forward channel from the reverse channel.

Fortunately, this calibration constant only depends on the transmitter. It can be calibrated a priori by the having each AP measure the forward and reverse channels to any other AP and thereby infer the calibration constant.[6]

5.3 Overcoming Limitations of the 802.11n standard

Interference Alignment with today's MIMO clients. Implementing interference alignment with today's 802.11 nodes is quite challenging. In particular, consider the example in Fig. 9, where we have a 2-antenna client receiving its desired signal from AP-1. Two other access points, AP-2 and AP-3 align their transmissions together at this client to collapse their interfering signals into one stream. In principle, the client in Fig. 9 can decode his desired signal by projecting the signal it received from AP-1 orthogonal to the common direction of the two interferers (See Fig. 3). This is the standard approach for decoding multiple streams, and, in particular, interference alignment. The client should now ignore the interfering signal, since it does not need it, and in fact, cannot decode it. This mechanism has been used in several past designs implementing alignment on software radios [11, 16]. Unfortunately, current 802.11 standards enforce fate sharing between the different MIMO streams received by a MIMO receiver. Specifically, the

[5] If not, the channels would be severely distorted by an accumulated phase difference owing to the frequency offset between the clocks of the AP and client, as well as phase noise [23].

[6] The reason the calibration constant depends only on the transmitter, is that all interference management techniques rely only on the ratio of the channels from the transmit antennas, and not their absolute values [10]. As a result constant factors that depend only on the receivers are canceled out.

client in Fig. 9 is not designed to ignore the interfering stream and simply accept the desired stream that it was able to decode correctly (802.11 applies a single convolutional code across all streams. So, errors in one stream percolate to all streams). So how can OpenRF support interference alignment despite this major limitation?

Our solution to this challenge is counter-intuitive. The only way to implement interference alignment on off-the-shelf MIMO nodes is to convince the receiver that one of his antennas is not functional! Specifically, the driver can issue a command to deactivate one of the client's antennas. Suppose we now align the two interfering streams along antenna 1, by setting the direction of alignment V_i (see §4.2) as the unit vector along the axis corresponding to antenna 1 . We then ask the client to deactivate antenna 1 (Fig. 10).[7] Then, effectively, the client will not see the interfering stream altogether, and it will only observe its desired stream, which can now be decoded. This effectively tricks the client into performing interference alignment by ignoring the fate sharing between MIMO streams. Therefore, our approach allows AP-2 to transmit to its own client, even while AP-1 and AP-3 are concurrently transmitting to their own clients, thereby increasing concurrency in the network.

It has to be Unitary! Past literature, as well as our discussions so far, assumes that we can set the precoding vectors to arbitrary values according to the desired PHY-technique. However, the 802.11n compressed transmit beamforming feature requires the matrix of precoding vectors to be unitary. For example, an AP with 3 antennas, applies 3 precoding vectors which forms a 3×3 matrix. This matrix has to be unitary. To ensure this, after computing precoding vectors, OpenRF uses geometric transforms to express the ultimate precoding matrix, used by the device, as a unitary matrix.

6. IMPLEMENTATION

We implemented OpenRF by modifying the iwlwifi driver for Intel Wi-Fi cards on Ubuntu 10.04 LTS. We built our solution over the University of Washington's 802.11 CSI tool [12]. Our APs also use the transmit beamforming feature of the Intel 5300 Wi-Fi cards.[8]

To support the OpenRF Interface, we made the following modifications to the driver: First, we implemented the OpenRF table (Sec. §4.1) as a debugfs file that interfaces between user and kernel modes. Second, OpenRF uses the 802.11 CSI tool to calculate the reciprocal channels from the clients, which are used to maintain the coherence and interference spaces in the OpenRF table for various flows at this AP. Finally, we employ the transmit beamforming feature to set precoding vectors for various flows in real-time, using the coherence and interference spaces as in §4.2.

To support the OpenRF Controller at each AP, we first employ block aggregation to implement time slots spanning 5 ms. We use NTP over the wired Ethernet backbone to synchronize the block aggregation slots across APs in the network. We observed that this was sufficient to synchronize blocks spanning 5 ms each, within tens of microseconds. We intercept packets from the higher layers into per-flow queues at the driver and apply the local controller algorithm as described in §4.3. Our controller and scheduler algorithms are implemented using high resolution timers to minimize waste of airtime. The central controller is implemented on a Linux workstation, and coordinates APs in the network over the Ethernet backbone using the algorithm described in §4.3.

Finally, our implementation fully complies with 802.11n and handles concurrent transmissions, carrier sense, and ACKs as in §5.

[7]More generally, for better performance, the client dynamically deactivates the antenna with poorest channel to the desired AP.

[8]While we chose to implement OpenRF on Intel cards, OpenRF's design is compatible with any 802.11n card that supports the transmit beamforming feature[1].

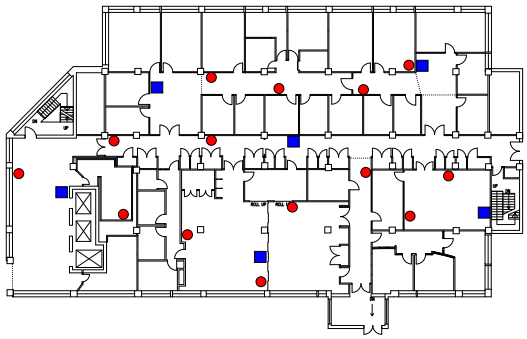


Figure 11—Floor map for our testbed. Our 20-node testbed includes 6 APs (blue squares) and 14 clients (red circles).

We deployed OpenRF on a network of 20 nodes, each containing an Intel Wireless-N series card connected to a PC or laptop. Our network has six 3-antenna APs, spread out on a single floor of a large building as shown in Fig. 11. Each AP is a Linux PC running hostapd, and implementing our patched iwlwifi driver. We deploy 14 clients, including seven single-antenna, five 2-antenna and two 3-antenna wireless nodes. We randomize the locations of the clients on the floor across different experiments (a candidate set of client locations is shown in Fig. 11). All APs in the network share the same wireless channel, potentially causing interference at some clients, that are at the edge of their AP's communication range.

7. RESULTS

We compare OpenRF in various settings with a standard 802.11n baseline that uses block-aggregation but does not perform interference nulling, coherent beamforming or interference alignment.

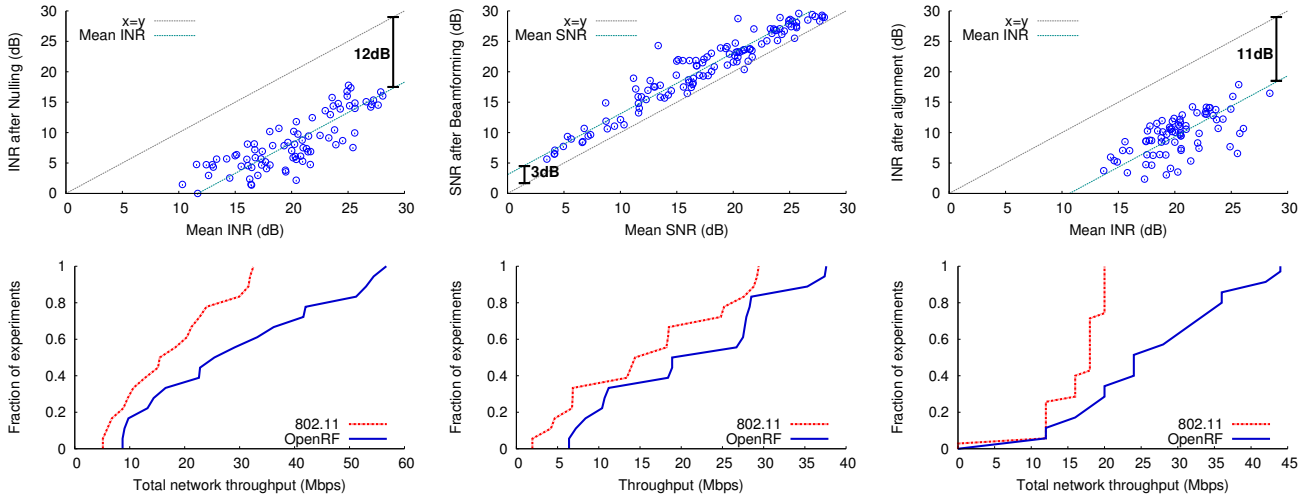
7.1 PHY-aware techniques on Commodity Cards

First, we evaluate how effectively commodity Wi-Fi cards can perform common PHY-aware techniques, like, interference nulling, coherent beamforming and interference alignment. We evaluate how these techniques impact TCP flows generated using iperf [30] in our 20-node network in Fig. 11. We use a mix of both long and short lived TCP flows. Each client has one long lived flow. Additionally, short flows are generated according to a Poisson arrival process, with mean inter-arrival time of 1s, and have a Pareto file size with mean of 125KB and shape parameter of 1.5 [21, 7].

We use the OpenRF Controller to identify scenarios corresponding to different MIMO interference management techniques, applied in this network. We repeat the experiment across randomly chosen client locations. In each case, we compare our system's performance with 802.11, by replaying identical traffic patterns.

Experiment 1: (Interference Nulling) We consider scenarios reported by the controller where two single-antenna clients obtain signals from two APs as shown in Fig. 7. To mitigate interference, the controller applies interference nulling on the APs, so that they null any unwanted interference at neighboring clients. We measure the interference-to-noise ratio (INR) at each client from its unwanted AP, with and without OpenRF's nulling, across experiments. We also measure the total throughput of the TCP flows per client, with OpenRF's interference nulling and standard 802.11.

Results 1: The plot in Fig. 12(a) shows that the INR reduction from Interference Nulling is an average of 12 dB. This is a substantial reduction in INR, and it is sufficient for enabling concurrent transmissions in common wireless networks. This is because the operational SNR of 802.11 is in the range of 5 - 25 dB [23]. Furthermore, clients that experience interference are often near the boundaries of



(a) Interference Nulling **(b) Coherent Beamforming** **(c) Interference Alignment**

Figure 12—MIMO Interference techniques. We plot the SNR or INR of the client and CDF of total throughput for TCP flows with and without (a) Interference Nulling. (b) Coherent Beamforming. (c) Interference Alignment.

the communication range of two Access Points. Such clients receive low SNR from their interfering APs, which still severely disrupts signals from their own AP. In fact, our results in Fig. 12(a), demonstrate that performing Interference nulling for such clients leads to a considerable gain in total throughput of TCP flows per client by a factor of 1.7×.

One might wonder why the reduction in INR in commodity cards using Interference nulling is below that of typical software radios [16, 11]. This is because, the channel state information reported by 802.11 is only provided for 30 OFDM sub-carriers, and each 3 × 3 transmit beamforming matrix is allotted only 24 bits [1]. While these settings reduce the overhead from explicit feedback, they also limit the INR reduction possible. However, as shown above, this reduction suffices to enable big gains for real WLANs.

Experiment 2: (Coherent Beamforming) In this experiment, we consider scenarios where the controller performs coherent beamforming from an AP to a single-antenna client. For each set of client locations, we measure the increase in SNR at the client, with 802.11 and OpenRF's beamforming. We also measure the total throughput of TCP flows to this client with OpenRF and standard 802.11.

Results 2: The plot in Fig. 12(b) shows the SNR from AP to client, with and without coherent beamforming. We observe that coherent beamforming provides a mean increase of 3 dB in the SNR of the desired signal. This is very significant, since we observed this was sufficient to enable 802.11's rate adaptation algorithm[13] to adapt to a higher rate compared to standard 802.11, under identical circumstances, in 80% of our experiments. In fact, our results for TCP flows, shown in Fig. 12(b) demonstrate a mean gain of 1.4× in the throughput of OpenRF's beamforming, compared to 802.11.

Besides the gain in SNR, the gain in throughput is facilitated by another interesting phenomenon. Quite often, some OFDM sub-carriers in the 802.11 channel profile of a single antenna experience low SNR due to fading. However, coherent beamforming from all three transmit antennas across sub-carriers ensures that fading of any single OFDM bin is extremely unlikely. Thus, channel profiles using diversity, are significantly more flat, when compared to standard 802.11. This is especially beneficial, since 802.11 uses the same modulation and coding scheme across OFDM bins.

Experiment 3: (Interference Alignment) In this experiment, we consider scenarios reported by the controller where a 2-antenna client is receiving its desired signal from its AP. However, the client also receives undesired interference from two other 3-antenna APs (Fig. 9). To mitigate this, we perform interference alignment, so that the two interfering APs align their signals along one of the client's antennas. This ensures that any residual interference on the client's other antenna is minimized. The client now decodes its desired signal, interference-free, from its other antenna (See §5.3).

For each client, we then measure the total INR received from both antennas, as well as how much residual INR leaked into the second antenna meant for the desired signal. We also evaluate the gain in the total throughput of the concurrent TCP flows made possible due to OpenRF's alignment, compared to standard 802.11.

Results 3: Fig. 12(c) plots the residual INR leaking into the direction of the desired signal, against the total INR. Our results show that the residual INR is 11 dB below the total INR. This is a significant reduction, as clients receiving signals at the boundary of three APs are likely to obtain low INRs from the interfering APs. Yet, these INRs are still comparable to the SNR of the desired signal. In fact, the total throughput of TCP flows for the client under OpenRF's Interference Alignment experiences a 1.5× gain compared to standard 802.11.

Interestingly, OpenRF's controller algorithm does not apply Interference Alignment for multi-antenna clients in all scenarios resembling Fig. 9. In some cases, the controller may choose not to apply alignment, as the channel matrix from an interfering AP to the client was ill-conditioned. In fact, for any MIMO system to provide multiplexing gains, it is imperative that the channel matrix is well-conditioned [35]. This constraint naturally extends to interference alignment. Specifically, Fig. 13 plots the ratio of total and residual INR for different condition numbers of the channel. Note that when the log of the condition number exceeds 1.2, the total and residual INRs are nearly equal, providing virtually no gain. As OpenRF's controller has channel state information, it accounts for this by applying alignment only when channel matrices are well-conditioned, i.e. the log of the condition numbers are below 0.6.

7.2 Application performance

In this experiment, we evaluate the impact of OpenRF on application performance. Specifically, we consider the problem of ac-

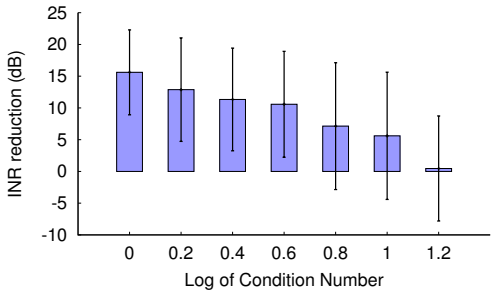


Figure 13—Effect of Channel on Interference Alignment. The gain of interference alignment is highly dependent on the condition number of the channel matrix. OpenRF only employs interference alignment when the log of the condition number is below 0.6.

cessing rich multimedia applications over remote desktop, which is increasingly common in the enterprise.

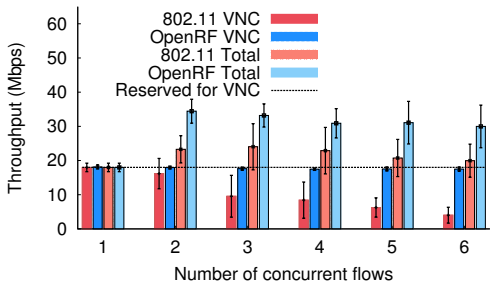
Experiment. Once again, we consider scenarios reported by the controller where two single-antenna clients obtain signals from two APs as shown in Fig. 7. In this experiment, our clients run a remote desktop session over their wireless links, using VNC [15], a commonly used open-source platform for remote desktop. In particular we play a 1080p HD-video over remote desktop from the remote AP. We implement the VNC server on the access points to eliminate any potential delays or loss of throughput over the wired link. Our experiment proceeds as follows: We start a VNC session from client-1 to AP-1 at $t = 0$. We then start a parallel VNC session from client-2 to AP-2 at $t = 30$ seconds, and gather traces for both sessions until $t = 60$ seconds. We repeat this experiment under identical conditions for 802.11 and OpenRF. We measure the following quantities:

- *Updates per second*: The VNC protocol works by clients requesting updates for a certain number of pixels for a frame. The server responds with any changes to these pixels. In our experiment, we play a clip of Psy's Gangnam style, a rich and dynamic HD-video requiring frequent updates. We note that updates at a frequency of 15-20 per second are sufficient to avoid perceivable glitches.
- *Response delay*: Measures the mean delay between a request and its corresponding response. If the response time is high (over 200 ms), the video has visible outages.

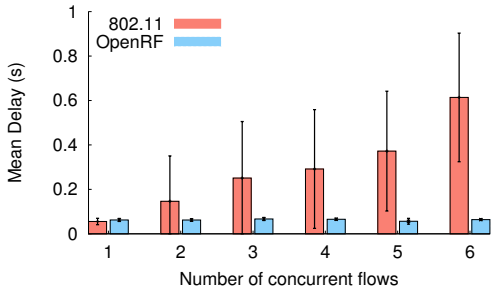
Results. Fig. 14 plots the traces for the two clients for both OpenRF and 802.11. First, we notice that with 802.11, the VNC flow ceases to provide a comfortable remote desktop experience once the two APs transmit concurrently. In particular, due to heavy contention between the two large VNC flows, we observe that the number of updates per second, varies dramatically in the range of 2-22 updates per second for each flow, with a mean of 12.2. This means that the user experiences several glitches in the video. More importantly, we observe on several occasions that the response delay is over 200 ms, with the 90th percentile response delay being 270 ms. Thus, the user experiences frequent and visible outages in the remote desktop application.

In contrast, OpenRF provides updates for both clients in the range of 14-22 updates per second, with a mean of 17.6 for the clients, even when they are transmitting concurrently. This is because OpenRF benefits from PHY-aware cross-layer techniques which enable concurrent transmissions and therefore eliminate the need for contention for medium-sharing between the access points. As a consequence, OpenRF enhances user experience by ensuring fewer glitches[9] in the video by a factor of 6×, compared to standard 802.11. In addition, the 90th percentile delay of OpenRF is 67 ms, which is well below the required 200 ms, and is 4× lower than that of standard 802.11.

[9]We quantify glitches based on the number of times there were fewer than a fixed threshold of 14 updates per second.



(a) Throughput Reservation



(b) Response Delay

Figure 15—Reservation with VNC. Plots the mean and standard deviation of the following quantities, for OpenRF and 802.11, across number of concurrent flows: (a) Throughput of VNC flow and total network throughput (b) Response Delay of the VNC flow.

7.3 Reservation

We study how OpenRF's reservation policy enhances application-level quality of experience.

Experiment. We consider scenarios reported by the OpenRF controller where a single-antenna client is at the edge of the communication range of the two APs, and therefore requires interference nulling from one of the APs. We reserve a throughput of 17 Mbps for a large VNC flow destined to this client from its APs. We then begin transmitting concurrent long-lived TCP flows, from either AP to up to five other clients in the network and report the VNC performance as a function of the number of concurrent flows.

Results. The plot in Fig. 15(a) shows the throughput of the VNC flows, as well as the total network throughput, across the number of concurrent flows for both 802.11 and OpenRF. We observe that OpenRF reserves throughput at a mean value of 17.2 Mbps for the VNC flow, as required by the flow. More importantly, it does this while continuing to provide the same gains in total network throughput by exploiting PHY-aware cross layer techniques to enable concurrent transmissions of flows between the two APs.

In contrast, with 802.11, the performance of the VNC flow decays with an increasing number of contending flows. Furthermore, the network throughput is lower, as the APs do not manage interference, and therefore must share the medium between them.

In addition, Fig. 15(b) depicts the delay of the VNC flows, with 802.11 and OpenRF. As expected, OpenRF maintains the required user quality of experience by keeping VNC response delays at a low mean of 63 ms, despite concurrent transmissions, while with 802.11, the delays progressively deteriorate with an increasing number of concurrent flows.

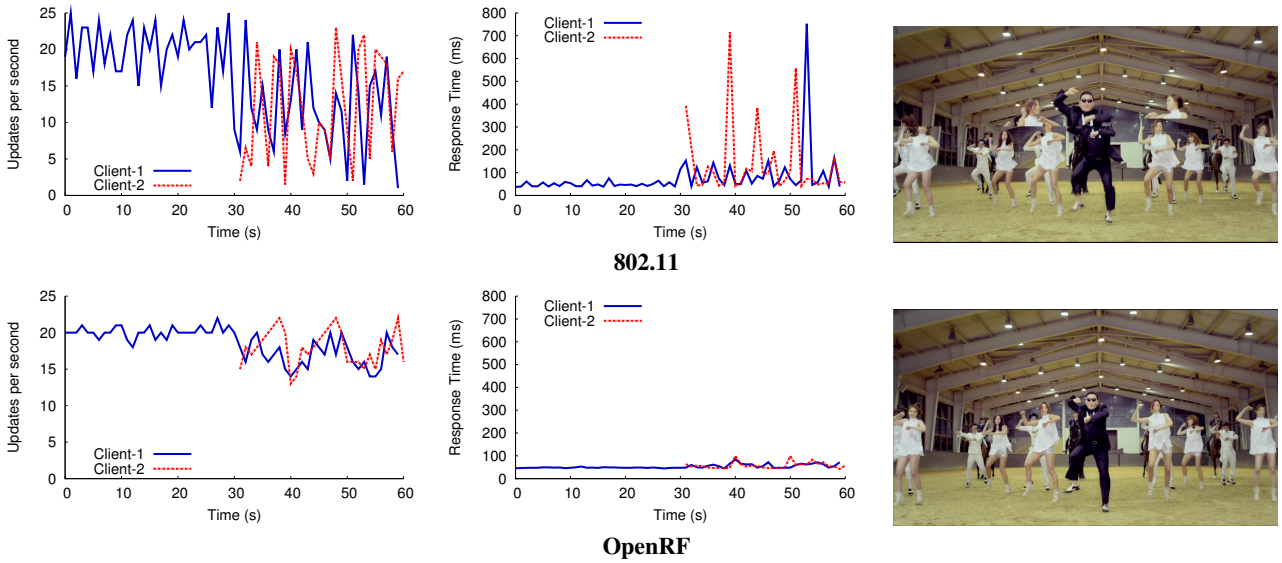


Figure 14—Performance of Video over Remote Desktop. Trace for 802.11 and OpenRF of: Column (1) - number of updates per second; Column (2) - response time(ms); Column (3) - snapshot of the HD video at the same frame.

7.4 Large scale measurement

Finally, we study how OpenRF performs under dynamic traffic patterns, dynamic channels, flow arrivals and departures for the full 20-node network.

Experiment. We place our AP and client nodes in randomly chosen locations (Fig. 11). Our clients are a combination of seven single-antenna, five 2-antenna and two 3-antenna nodes. We conduct our experiments with a variety of TCP and UDP best effort flows generated using iperf. In particular, our network allows a combination of TCP ($\approx$ 60 %) and UDP ($\approx$ 20 %) flows as well as uplink TCP and UDP traffic ($\approx$ 20 %).[10] UDP flows are long lived flows. TCP flows are generated using a mix of both long and short lived TCP flows. Each client has one long lived flow. Additionally, short flows are generated according to a Poisson arrival process, with mean inter-arrival time of 1s, and have a Pareto file size with mean of 125KB and shape parameter of 1.5. We assign TCP flows between uplink and downlink, according to the desired traffic ratios. We compare our system with standard 802.11 by replaying identical traffic patterns.

Results. Figure 16 plots the total throughput of all TCP and UDP flows per client, for 802.11 and OpenRF. In particular, we obtain a mean gain of 1.6× for all TCP flows and 1.7× for all UDP flows in the network. More importantly, we note that every client in our system achieves higher throughput on average, compared to 802.11. In fact, our results demonstrate that the OpenRF controller is truly self-configuring and correctly employs the right combination of PHY-aware techniques to suit dynamic network topologies and traffic patterns in real-time.

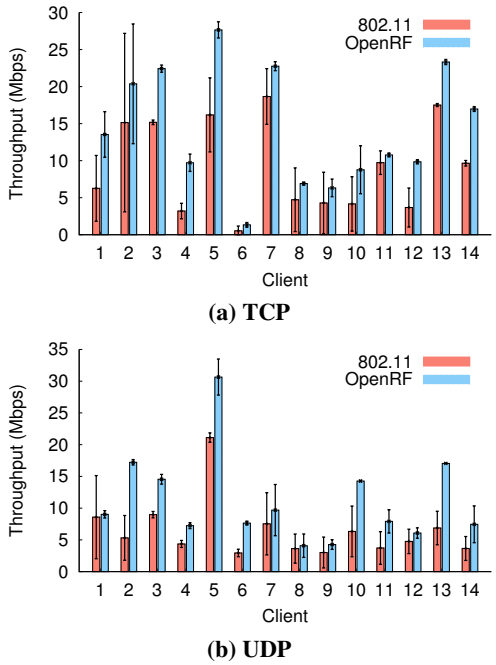


Figure 16—Performance. Plots the mean and standard deviation of throughput of TCP and UDP flows for all clients in the network

8. Related Work

Related work falls into three categories:

Cross-Layer Designs: OpenRF is related to past work on the cross-layer wireless designs [32, 31, 22], particularly to systems that perform MIMO interference management, e.g., Interference Alignment and Cancellation [11], Beamforming [2], SAM [28], and others [23, 10, 16, 6]. However, these systems use techniques that focus on specific topologies or traffic patterns. In contrast, OpenRF is a general architecture that applies the right set of MIMO techniques to any topology or traffic pattern. Furthermore, unlike past work implemented on software radios, OpenRF leverages a modified 802.11n MAC, which employs carrier sense over interference and coherence slots to support MIMO techniques in today's wireless LANs. Thus, OpenRF tackles new challenges such as interference management in the presence of bursty TCP traffic, cooperative MIMO techniques across APs in an enterprise, compliance and integration with 802.11 standards, and implementation using commodity Wi-Fi cards.

[10]These ratios are based on the traffic mix on our local network.

Software Defined Networks: Our work is inspired by the large body of work designing software defined networks for wired LANs, including OpenFlow [18], and Ethane [4]. While we borrow the design principle of separating the data and control plane, unlike OpenFlow which remains at Layer-2, OpenRF provides MIMO interference management techniques that impact the physical layer. Such techniques cannot be built simply using OpenFlow.

Recently, there have been several proposals bringing software defined networking to wireless LANs. For e.g., OpenRoads [33], OpenWiFi [14] and Odin [27] abstract the higher layers of the network stack to provide applications such as seamless mobility, load balancing, etc. Our work is complementary to this work, and further enhances the control plane through MIMO interference management techniques.

Perhaps the closest related work to our system is OpenRadio[3], which provides a programmable data plane for wireless base stations supporting multiple technologies such as LTE, WiFi, or WiMAX. OpenRadio deals purely with managing the data plane on a single wireless device, by re-using physical layer blocks to implement different technologies. Our system complements OpenRadio by coordinating multiple wireless APs in a network to manage interference from the control plane.

Enterprise WLAN Architectures: DenseAP [19], Dyson [20], CENTAUR [25], and DIRAC [34] have proposed architectures to better manage enterprise wireless LANs. OpenRF complements this work by adding new MIMO cross-layer techniques to the toolbox available for managing enterprise WLANs.

Recent systems have proposed improving network management by budgeting frequency channels [24, 5], time [9], or leveraging antenna arrays [17]. In contrast, our system enables multiplexing shared spectrum across interfering APs through MIMO interference nulling and alignment. This complements the above schemes, enabling more efficient use of spectrum per channel. In addition, OpenRF can accommodate networks moving towards channel bonding [8] leading to larger chunks of channels shared across APs.

Finally, Trinity [26] proposes profiling users based on mobility and channel coherence for enabling distributed MIMO and diversity. However, unlike OpenRF, it uses custom hardware (WARP boards) and does not study how these PHY layer techniques interact with real-life traffic patterns and application-layer requirements.

9. CONCLUSION

This paper introduces OpenRF, the first system that enables the deployment of physical-layer MIMO techniques on commodity Wi-Fi cards. It is also the first cross-layer design that is demonstrated using a fully operational network stack with real applications. While our current implementation of OpenRF demonstrates interference nulling, interference alignment and beamforming, we believe OpenRF can be extended to support other PHY-aware techniques such as multi-user MIMO and frequency diversity schemes [29], when future wireless cards support them.

Acknowledgments: We are grateful to Daniel Halperin for helping us with the 802.11 CSI tool [12] and Astrit Zhushi for reporting important fixes to the tool. We thank Anirudh Sivaraman, Srinivas N.G., Jonathan Perry, Arthur Berger, the NETMIT group and our reviewers for their insightful comments. We are grateful to our shepherd Peter Steenkiste for his detailed feedback. This work is funded by NSF. We thank members of the MIT Center for Wireless Networks and Mobile Computing: Amazon.com, Cisco, Google, Intel, Mediatek, Microsoft, ST Microelectronics and Telefonica for their interest and support.

10. REFERENCES

[1] IEEE 802.11n-2009 (Section 9.10). http://standards.ieee.org/findstds/standard/802.11n-2009.html.

[2] E. Aryafar, N. Anand, T. Salonidis, and E. Knightly. Design and experimental evaluation of multi-user beamforming in wireless LANs. In *MOBICOM*, 2010.

[3] M. Bansal et al. Openradio: A programmable wireless dataplane. In *HOTSDN*, 2012.

[4] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: taking control of the enterprise. SIGCOMM, 2007.

[5] R. Chandra et al. A case for adapting channel width in wireless networks. SIGCOMM, 2008.

[6] J. Choi et al. Achieving single channel, full duplex wireless communication. MobiCom, 2010.

[7] M. Crovella et al. Heavy-tailed probability distributions in the world wide web. *A practical guide to heavy tails*, 1:3–26, 1998.

[8] L. Deek et al. The impact of channel bonding on 802.11n network management. CoNEXT, 2011.

[9] P. Djukic and P. Mohapatra. Soft-tdmac: A software tdma-based mac over commodity 802.11 hardware. In *INFOCOM*, 2009.

[10] S. Gollakota, F. Adib, D. Katabi, and S. Seshan. Clearing the rf smog: making 802.11n robust to cross-technology interference. In *SIGCOMM*, 2011.

[11] S. Gollakota, S. Perli, and D. Katabi. Interference alignment and cancellation. *SIGCOMM*, 2009.

[12] D. Halperin, W. Hu, A. Sheth, and D. Wetherall. Tool release: Gathering 802.11n traces with channel state information. *ACM SIGCOMM CCR*, 2011.

[13] D. Halperin et al. Predictable 802.11 packet delivery from wireless channel measurements. In *CCR*, 2010.

[14] H. Hu. Openwifi : a consumer wifi sharing system. MEng. Thesis, MIT, 2007.

[15] C. Kaplinsky. Tightvnc related software. april 2008.

[16] K. Lin, S. Gollakota, and D. Katabi. Random access heterogeneous mimo networks. In *SIGCOMM*, 2011.

[17] X. Liu et al. Pushing the envelope of indoor wireless spatial reuse using directional APs and clients. MobiCom, 2010.

[18] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *CCR*, 2008.

[19] R. Murty, J. Padhye, R. Chandra, A. Wolman, and B. Zill. Designing high performance enterprise wi-fi networks. NSDI, 2008.

[20] R. Murty, J. Padhye, A. Wolman, and M. Welsh. Dyson: an architecture for extensible wireless lans. USENIXATC, 2010.

[21] V. Paxson and S. Floyd. Wide area traffic: the failure of poisson modeling. *IEEE/ACM ToN*, 1995.

[22] V. Pejovic and E. Belding. A context-aware approach to wireless transmission adaptation. In *SECON*, 2011.

[23] H. Rahul, S. S. Kumar, and D. Katabi. Megamimo: Scaling wireless capacity with user demand. In *SIGCOMM*, 2012.

[24] E. Rozner, Y. Mehta, A. Akella, and L. Qiu. Traffic-aware channel assignment in enterprise wireless lans. In *ICNP'07*.

[25] V. Shrivastava et al. CENTAUR: Ralizing the full potential of centralized wlans through hybrid data path. MobiCom, 2009.

[26] S. Singh et al. One strategy does not serve all: tailoring wireless transmission strategies to user profiles. HotNets, 2012.

[27] L. Suresh et al. Towards programmable enterprise wlans with odin. HotSDN, 2012.

[28] K. Tan, H. Liu, J. Fang, W. Wang, J. Zhang, M. Chen, and G. M. Voelker. SAM: enabling practical spatial multiple access in wireless LAN. In *MobiCom*, 2009.

[29] K. Tan et al. Fine-grained channel access in wireless lan. *CCR*, 2011.

[30] A. Tirumala et al. Iperf: The tcp/udp bandwidth measurement tool. *http://dast.nlanr.net/Projects*, 2005.

[31] M. Vutukuru, H. Balakrishnan, and K. Jamieson. Cross-layer wireless bit rate adaptation. *CCR*, 2009.

[32] L. Yang, W. Hou, L. Cao, B. Y. Zhao, and H. Zheng. Supporting demanding wireless applications with frequency-agile radios. NSDI, 2010.

[33] K. Yap et al. The stanford openroads deployment. In *WINTECH*, 2009.

[34] P. Zerfos et al. Dirac: a software-based wireless router system. MobiCom, 2003.

[35] L. Zheng and D. Tse. Diversity and multiplexing: A fundamental tradeoff in multiple-antenna channels. *IEEE Transactions on Information Theory*, 2003.