SourceSync: A Distributed Wireless Architecture for Exploiting Sender Diversity

Hariharan Rahul, Haitham Hassanieh, and Dina Katabi
Massachusetts Institute of Technology

ABSTRACT

Diversity is an intrinsic property of wireless networks. Recent years have witnessed the emergence of many distributed protocols like ExOR, MORE, SOAR, SOFT, and MIXIT that exploit receiver diversity in 802.11-like networks. In contrast, the dual of receiver diversity, sender diversity, has remained largely elusive to such networks.

This paper presents SourceSync, a distributed architecture for harnessing sender diversity. SourceSync enables concurrent senders to synchronize their transmissions to symbol boundaries, and cooperate to forward packets at higher data rates than they could have achieved by transmitting separately. The paper shows that SourceSync improves the performance of opportunistic routing protocols. Specifically, SourceSync allows all nodes that overhear a packet in a wireless mesh to simultaneously transmit it to their nexthops, in contrast to existing opportunistic routing protocols that are forced to pick a single forwarder from among the overhearing nodes. Such simultaneous transmission reduces bit errors and improves throughput. The paper also shows that SourceSync increases the throughput of 802.11 last hop diversity protocols by allowing multiple APs to transmit simultaneously to a client, thereby harnessing sender diversity. We have implemented SourceSync on the FPGA of an 802.11-like radio platform. We have also evaluated our system in an indoor wireless testbed, empirically showing its benefits.

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

General Terms Algorithms, Design, Performance

1 Introduction

Diversity across nodes is an intrinsic property of wireless networks. The wireless environment exhibits both receiver diversity and sender diversity. Receiver diversity is the property that a single transmitted packet traverses different channels to different receivers, and hence is unlikely to suffer fading at all receivers at the same time. Sender diversity, on the other hand, is the property that a packet transmitted *simultaneously* from multiple senders traverses different channels to the same receiver, and hence is unlikely to suffer fading from all senders at the same time. In the context of 802.11 networks, the ability to have multiple transmitters simultaneously forward a packet to a receiver can harness both frequency diversity and power gains. Specifically, 802.11 channels span a relatively wide bandwidth (20–40 MHz), where different senders experience deep fading in different frequencies. Enabling multiple transmitters to simultaneously forward a packet to a receiver ensures that no frequency is deeply faded at the receiver, and reduces the overall bit error rate for a particular transmission power. Second, simultaneously forwarding a packet enables senders to combine their transmission power and thereby deliver a higher SNR to the receiver, as compared to a single sender.[1]

Despite the benefits of simultaneous forwarding from multiple transmitters, existing approaches for sender diversity in 802.11 networks restrict themselves to only one sender transmitting at a time, using mechanisms like picking the sender with the best channel [26]. This is in sharp contrast to receiver diversity where many practical systems like ExOR, MORE, SOAR, SOFT, and MIXIT [4, 5, 31, 44, 19] leverage simultaneous reception across multiple receivers.

Simultaneous transmission from multiple senders has challenged 802.11 for three main reasons.

- First, senders need to be synchronized to the symbol level in order that their signals combine on the medium in a manner that reduces the overall packet error rate. Such fine-grained transmitter synchronization is difficult to achieve in a distributed manner, as has been observed by past research [9, 18, 13, 30]. The difficulty arises because the different transmitters need to time their transmissions so that they are synchronized accurately (to within tens of *ns*) [9] at the receiver. In the absence of a shared clock or a central controller, the only mechanism for synchronization is for senders to use packet reception as a reference. However, such a mechanism requires transmitters to compensate for differences in propagation delays, and hardware turnaround times from reception to transmission. These measurements are challenging because a node does not detect packet reception at the exact instant when the signal arrives at its antenna, but rather incurs a random delay depending on the noise in the environment and the receiver hardware. This variability is usually on the order of hundreds of *ns* [42], which is too high for accurate symbol-level synchronization.
- Second, the received signal is a combination of signals from multiple senders. Each of these signals has traversed a different path, and has hence experienced a different channel. One might think that the receiver could compensate for the channel distortion of the composite signal in the same manner as it would compensate for the channel distortion of a signal from a single sender. Unfortunately, this approach does not work since the composite channel has fundamentally different characteristics from single sender-receiver channels. Specifically, unlike single sender-receiver channels, which have a constant attenuation throughout a packet, the attenuation of the composite channel varies even within a single packet. This is because the oscillators of different senders naturally

[1] The FCC limits the maximum transmission power of a single sender, and combining transmissions therefore increases the maximum received power.

SIGCOMM 2010, August 30–September 3, 2010, New Delhi, India.

have slightly different operating frequencies, and hence the signals from different senders continuously rotate relative to each other.

- Finally, transmitted signals are complex numbers which have phases. Unless these signals are carefully orchestrated at the senders, they can add up constructively, enhancing each other, or destructively, weakening each other.

This paper introduces SourceSync, a practical architecture for harnessing sender diversity. SourceSync is designed for OFDM, which is the transmission scheme for most modern wireless networks, including 802.11 a/g/n, WiMax, LTE etc. SourceSync has three components that harness sender diversity in a distributed manner:

Symbol Level Synchronizer (SLS). SourceSync has a distributed synchronization algorithm that leverages packet reception as a time reference, computes robust estimates of the propagation delays from all senders to the receiver, as well as hardware turnaround times at each of the senders, and compensates for these delays at the senders prior to transmission, in order to ensure that the packets arrive synchronized at symbol boundaries at the receiver. The key feature that allows SourceSync to achieve tight synchronization is that it can prevent the inherent variability in packet detection from inducing variability in its propagation delay and turnaround time estimates. SourceSync has a mechanism that allows it to accurately measure the delay between the first sample of a packet and when the receiver detects that packet, and account for the delay when computing its estimates. Further, SourceSync can leverage data packets to track changes in propagation delay over time, and hence keep senders synchronized without the need for active measurements.

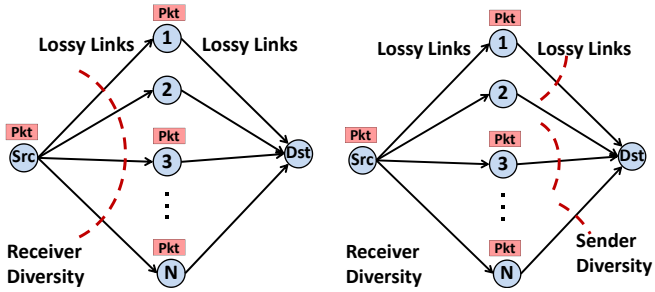
Joint Channel Estimator (JCE). A SourceSync receiver decodes the combined signal from multiple synchronized senders. However, SourceSync differs from prior schemes, where transmitted signals interfere, and hence decoding the signals either requires multiple transmissions from each sender, as in ZigZag [13], or a large difference in power (or code rate) between them, as in Successive Interference Cancellation [14, 40]. In contrast, SourceSync does not need to treat senders as interfering, and can decode a single simultaneous transmission from multiple senders, even when they have comparable powers. It estimates the individual channels from each sender, computes how they interact to create the composite channel, and tracks the variations of the composite channel through the combined packet.

Smart Combiner (SC). Since signals from multiple senders rotate continuously relative to each other, naively transmitting the same packet from all senders will cause the signals to combine destructively at some points within the packet. Therefore, senders need to have a joint strategy for manipulating the phase of the signal prior to transmission to ensure that their transmitted codewords do not combine destructively. SourceSync leverages the rich body of research on space-time block codes [39, 2, 16], which are typically used in MIMO systems to control how signals from different antennas on a single transmit node combine at a receiver. In contrast to MIMO systems, however, SourceSync uses these codes in a distributed manner across multiple transmit nodes.

We use SourceSync to develop the following two protocols.

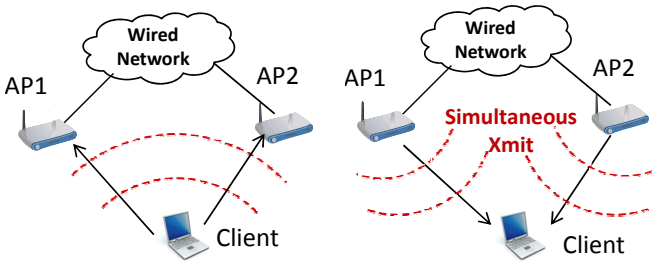
1.1 Combining Sender Diversity with Opportunistic Routing

Opportunistic routing protocols leverage receiver diversity; they exploit the fact that since wireless receptions are probabilistic, it is unlikely that all nodes closer to the destination are unable to receive a packet, as shown in Fig. 1(a). Protocols like ExOR, MORE, SOAR,



(a) First-hop receiver diversity (b) Second hop sender diversity

Figure 1: Opportunistic routing with sender diversity. SourceSync enables multiple forwarders to transmit jointly to the destination.



(a) Uplink receiver diversity (b) Downlink sender diversity

Figure 2: Last-hop with sender diversity. SourceSync enables multiple APs to transmit jointly on the downlink.

and MIXIT therefore allow any downstream node that receives a packet to forward it to the destination. However, none of these schemes take advantage of the analogous opportunity of sender diversity presented by the fact that multiple nodes often receive the same packet. SourceSync complements the opportunistic receptions exploited by current protocols with opportunistic synchronous transmissions by multiple forwarders. Specifically, since multiple forwarders are likely to receive a packet, they can transmit it simultaneously as shown in Fig. 1(b). This provides two types of gains. First, since different forwarders experience fades in different frequencies [30], joint transmission reduces the likelihood that a frequency experiences a deep fade at the receiver, and hence decreases the overall bit error rate. Second, since joint transmission allows forwarders to combine their power, it improves the receiver SNR, and thereby its bit rate.

1.2 Combining Sender Diversity with Last-hop Receiver Diversity

Protocols like MRD, SOFT and Link-Alike [25, 44, 17] all exploit different aspects of the same concept: last-hop diversity. Consider, for example, a sender that has poor connectivity to multiple nearby APs. A transmitted packet is unlikely to reach any specified AP, but is likely to be received by at least one AP. All the above protocols exploit this receiver diversity by allowing APs to combine received bits or packets over the wired network, and hence can increase *uplink* reliability without any retransmissions, as shown in Fig. 2(a). However, none of these schemes can similarly address a lossy *downlink* without expending medium time on retransmissions. SourceSync complements all these protocols by harnessing sender diversity to increase downlink reliability without any retransmissions, analogous to existing receiver diversity mechanisms on the uplink. Specifically, instead of requiring that a client receive packets from only one AP at a time, in SourceSync, multiple neighboring APs can transmit simultaneously to the client as in Fig. 2(b), and increase throughput.

1.3 Results

We implemented SourceSync on the FPGA of the WiGLAN radio platform [10]. We also implemented proofs of concept of both last-hop diversity, and opportunistic routing with sender diversity. Results from an indoor wireless testbed reveal the following:

- SourceSync's symbol level synchronization is accurate. Testbed evaluations show that two randomly chosen transmitters using SourceSync have a 95^{th} percentile synchronization error of at most 20 *ns* across the range of operational SNRs of 802.11.
- SourceSync increases the gains of opportunistic routing by exploiting sender diversity. Evaluating across multiple deployments with different bitrates and link loss rates, we show that the combination of SourceSync and ExOR achieves a median throughput gain of up to 45% over ExOR alone, and up to 2× over single-path routing.
- SourceSync is effective in harnessing last-hop sender diversity. Specifically, by having two APs transmit simultaneously to a client, SourceSync provides a median throughput gain of 57%. This is because the higher power resulting from simultaneous transmission from APs allows the combined transmission to use a higher 802.11 rate than a transmission by either AP alone.

1.4 Contributions

This paper makes the following contributions:

- It demonstrates via a design, implementation and testbed evaluation the practicality and benefits of simultaneous transmission in 802.11 networks.
- It presents a distributed algorithm for symbol level synchronization and an empirical study of its accuracy.
- It reveals the synergy between opportunistic routing and sender diversity by showing that opportunistic receptions can be further used to enable concurrent forwarding to downstream nodes.

2 Related Work

Sender diversity was pioneered by Laneman and Wornell's work on cooperative diversity, which theoretically demonstrated the gains of spatially diverse senders cooperating to relay information [21, 32]. Since then, many papers have analyzed aspects of sender spatial diversity focusing on signal processing and coding algorithms at the relays [35, 20, 33]. These papers focus on theoretical gains, ignore practical issues such as transmitter synchronization and oscillator offsets, and do not present a practical working system. Cellular networks today attempt to exploit sender diversity using Distributed Antenna Systems (DAS) [6]. DAS do not allow separate transmitters to send simultaneously; rather, they consist of a single transmitter with geographically distributed antennas connected using long, low attenuation cables. These systems are expensive and inflexible [7], and hence there is increasing interest in exploiting simultaneous transmissions from multiple senders in future cellular networks. The most recent WiMax multihop relay standard [38] includes simultaneous transmissions from multiple relays as an optional feature, and cooperative relays are also being considered for the future 3GPP LTE-Advanced standards [36]. However, there is no published work currently demonstrating a practical design and implementation of simultaneous transmissions for cellular systems, and further these systems operate under different constraints as they have the benefit of a centralized scheduler and a shared GPS clocking mechanism. 802.11 networks have also shown interest in exploiting sender diversity; however they still restrict themselves to only one sender transmitting at a time, using mechanisms like picking the sender with the best channel [26], which can neither exploit frequency diversity across senders, nor the power gain from combining multiple senders. Concurrent with our work, Zhang *et al.* [45] have demonstrated an implementation of cooperative diversity with nodes connected to a single shared clock. In contrast, our approach requires no shared clocks and applies to practical wireless networks, and also demonstrates the synergy of sender diversity with opportunistic routing.

Additionally, there has been recent work on systems that exploit concurrent transmissions from multiple senders, but cannot provide any sender diversity gains since they do not synchronize transmissions at the symbol level. These include systems like SMACK [9] for group acknowledgments, Message-in-Message [23] for exposed terminals, interference cancellation [14] and ZigZag [13] for hidden terminals, and ANC [18] for network coding.

Finally, SourceSync builds on past work on space-time block codes. These codes are used by different antennas on a single MIMO transmitter and do not extend to different transmitters due to lack of synchronization [39, 1, 2, 16], or because of oscillator frequency offsets [22]. SourceSync addresses synchronization and oscillator offset issues, showing that these codes can be implemented in a distributed manner to collect the gains of sender diversity in practice.

3 SourceSync Overview

SourceSync enables multiple senders to concurrently forward a packet to one or more receivers in order to collect diversity and power gains. It does so via a fully distributed joint PHY-MAC architecture.

(a) MAC: Medium access for concurrent transmissions is done by one of the senders, which we call the lead sender. Any node in the network can be a lead sender for a transmission. The lead sender accesses the medium via carrier sense, just as in 802.11. When the lead sender acquires the medium, it transmits a synchronization header. Other nodes that hear the synchronization header, and have the packet being transmitted, can then join the lead sender's transmission.

(b) PHY: The PHY layer ensures that concurrent transmissions are decodable at their intended receiver(s). It does so using three components: (a) a Symbol Level Synchronizer that ensures that transmissions from multiple nodes are synchronized, and can be decoded jointly at the receiver, (b) a Joint Channel Estimator which estimates the composite channel from the concurrent senders, and compensates for the resulting distortions, and (c) a Smart Combiner that encodes the concurrent transmissions to ensure that they combine on the channel in a manner that reduces the error rate at the receiver.

The next few sections describe the PHY in detail. The MAC is a simple extension of 802.11 carrier sense, and is described in the specific context of WLANs (§7.1) and opportunistic routing (§7.2).

4 Symbol Level Synchronization

4.1 Why do we synchronize transmitters?

To understand why one needs to synchronize, let us start by explaining what happens with a single sender-receiver pair. When a sender transmits to a receiver, the wireless signal bounces off walls, obstacles etc. and traverses multiple paths to the receiver. This phenomenon, known as the multipath effect, is a common distortion in wireless channels such as 802.11. As a result of the multipath effect, different copies of the same signal arrive at the receiver delayed with respect to each other. This means that the energy from one symbol bleeds into the next symbol, and corrupts its signal as shown in Fig. 3. Because

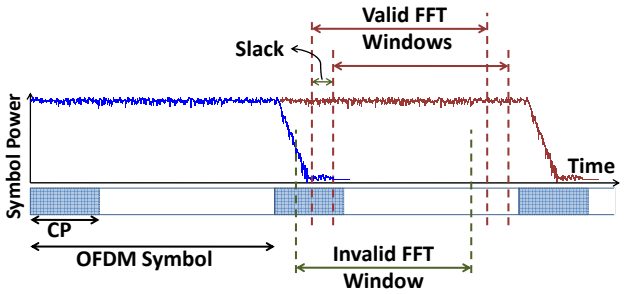
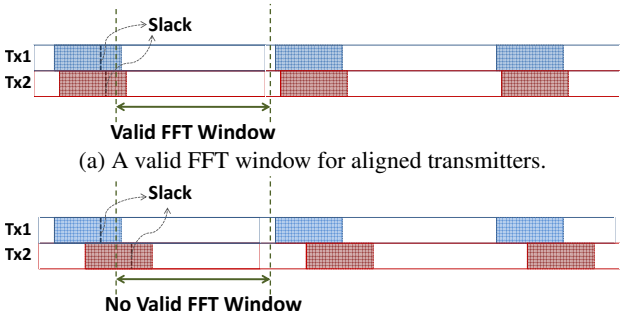


Figure 3: FFT windows at a receiver for a single transmitter. Any FFT window within the slack is valid. Any other FFT window would include energy from the previous symbol and hence is invalid.



(a) A valid FFT window for aligned transmitters.

(b) No valid FFT for these misaligned transmitters.

Figure 4: FFT windows at a receiver for two transmitters. In order to decode both transmissions, the symbols from the transmitters must arrive at the receiver aligned within the slack of the CP.

of this effect, OFDM symbols typically have a guard interval between them, called the cyclic prefix (CP). In a typical network, the value of the CP is chosen to be as small as possible while still accounting for the maximum multipath delay spread of the network, *i.e.*, the maximum delay difference between delayed copies of the signal.

OFDM data is encoded in the frequency domain. An OFDM receiver, in order to decode, converts the received symbol to a frequency representation by taking an FFT of the symbol. In order to do so while ensuring that the symbol is not corrupted by multipath noise from the previous symbol, the receiver should skip the samples in the CP, and take the FFT of the remaining samples.[2] In a typical network, the CP has a small amount of slack to allow for packet detection errors [42]. This means that the receiver has a corresponding amount of slack in the choice of where to align the receiver FFT window in a symbol. Thus, as shown in Fig. 3, any FFT window within the slack is valid. Any other FFT window would include energy from the previous symbol and hence lead to erroneous results.

Now, consider two senders transmitting the same symbol to a receiver. If the copies of this symbol from the two transmitters arrive at the receiver aligned within the existing slack of the CP, the receiver can take the FFT as before while still receiving energy only from this symbol, as shown in Fig. 4(a). If not, as before, any FFT window that the sender uses would include energy from the previous symbol, as shown in Fig. 4(b), and hence would yield incorrect results.

Of course, it is possible to increase tolerance to misalignment and provide more slack by increasing the CP. This approach, however, is problematic for two reasons. First, without sender synchronization, as in existing 802.11 networks, the amount of misalignment between senders can take any value depending on the differences in propagation delays and hardware processing times on different senders. While propagation delays may be bounded in certain environments based on the network diameter, the hardware processing times can be significantly different across senders. In fact, 802.11 standards [37, 11, 27] impose only very loose bounds on hardware turnaround times (10 μs in 802.11 a/g/n), and these are far longer than the 802.11 OFDM symbol time (4 μs). The second problem with increasing the CP is that the CP is overhead that is incurred for every OFDM symbol. Hence, the general trend has been to decrease the CP (for example, 802.11n negotiates down the CP if the network topology permits it [27]). Thus, even if one can exactly determine the required increase in the CP, such an approach will increase overhead and may significantly reduce, or even negate, the gains.

[2]Since the CP is a cyclic permutation of the symbol, and since FFT is periodic, the FFT yields correct results as long as it is within the symbol.

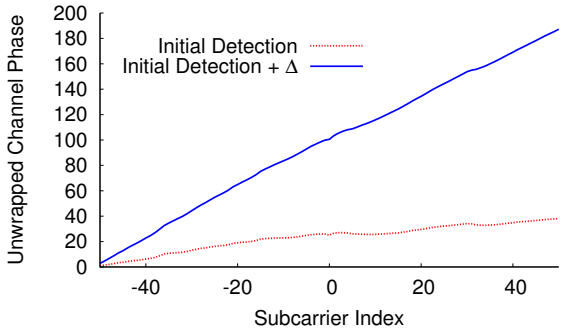


Figure 5: Unwrapped channel phase of OFDM subcarriers in a flat fading channel. The slope is a function of the detection delay.

4.2 Delay Measurements for Accurate Synchronization

At a high level, our synchronization algorithm is simple. One of the senders, called the **lead sender**, acquires the medium and transmits the packet. Upon hearing this signal, other nodes, which we refer to as **co-senders**, join the transmission. The choice of lead sender for a transmission depends on context and is explained in §7.

The key, however, is that transmissions from the lead sender and co-senders arrive aligned at the receiver. The challenge is that co-senders need to accumulate several samples before detecting the lead sender's transmission, and hence do not detect the transmission at the first sample. Further, different co-senders may take different times to turn around from receiving the lead sender's transmission to transmitting with the lead sender. Finally, signals from different senders traverse different paths and therefore incur different propagation delays. The co-senders therefore need to measure these different delays, and compensate for them to ensure synchronization at the receiver. In this section, we focus on how to accurately measure the delays, and describe how we compensate for the delays in the next section.

(a) Packet Detection Delay: This is the offset between the arrival of the first sample of the packet at a node, and the instant at which the receiver detects the packet. Estimating packet detection delay is a challenging task as it varies from packet to packet, and depends on the SNR, as well as the multipath characteristics of the channel.

SourceSync exploits a fundamental property of FFTs; a delay in the time domain manifests itself as a phase shift in the frequency domain [29]. To understand how we can leverage this property, let us look at the channel of an OFDM packet whose arrival the receiver detected at a few samples away from the first sample. For clarity, we discuss the case of a flat fading channel. The channel is a complex number, and we will focus on the phase of the channel in each OFDM subcarrier since that is the quantity affected by shifts in time. The dotted curve in Fig. 5 shows the receiver channel phases per subcarrier. As can be seen from the figure, the phases increase by a fixed slope. If we artificially induce an additional delay offset and process the

packet as if it were detected Δ samples after its actual detection time, the dotted slope of the graph changes to the solid slope as shown in Fig. 5. Thus, a delay offset in packet detection has introduced a shift in the phase of each OFDM subcarrier proportional to the index of that subcarrier.

In fact, one can show as a direct consequence of the definition of the FFT [29] that the change in phase of subcarrier i is $\frac{2\pi i\Delta}{N_s}$, where N_s is the number of samples in a symbol. Hence, in the graph in Fig. 5, the induced offset Δ introduces an additional slope of [3]

$$\zeta = \frac{2\pi\Delta}{N_s} \tag{1}$$

Now, what would the phase slope be if the receiver detects the packet exactly at the first sample? In the case of a flat fading channel (*i.e.* coherence bandwidth larger than channel bandwidth), the different OFDM subcarriers will experience similar channels. Hence, the phase of the subcarriers at different channels will be constant, and the slope will be zero. On the other hand, if the coherence bandwidth is very small, then the different OFDM subcarriers will experience uncorrelated channels. Since the phases of these channels are equally likely to be positive or negative, the slope will be close to zero in this case too. So, how about the intermediate case where the coherence bandwidth is neither too large nor too small? We can treat this case similar to the flat fading case by computing the slope over a small window of consecutive subcarriers that spans a width smaller than the coherence bandwidth, and averaging over several such windows. In fact, we do not need to differentiate between the cases; the solution proposed for intermediate channels works for the other cases too. Hence, in SourceSync, we compute the slope over windows of consecutive OFDM subcarriers that span 3 MHz, which is less than the coherence bandwidth of indoor channels [12], and average multiple such windows to estimate the overall slope. Since the slope should be zero in the absence of detection delay, we can substitute the average slope as ζ in Eq. 1, and compute the detection delay offset, Δ.

(b) Hardware Turnaround Delay: The turnaround delay is the time required for a co-sender to switch from reception of the lead sender's transmission to transmission of its concurrent signal. This time is dependent both on the speed of the baseband pipeline and the switching time of the radio frontend from reception to transmission. The turnaround time is constant for a particular node and can be measured by locally counting the hardware clock ticks from detection of the lead sender's packet to the beginning of the co-sender's transmission.

(c) Propagation Delay: This is the time of flight of the signal between the nodes. Given a transmitter-receiver pair, one can easily obtain an estimate of the total round trip delay between the nodes by having the sender send a probe and count the number of hardware clock cycles till it gets a response from the receiver. The round trip time elapsed between the transmission of the probe and the processing of the response has multiple components as follows:[4]

$$\begin{aligned}
\text{Delay}_{\text{Probe}\rightarrow\text{Response}} &= \text{Probe Propagation Delay from Tx to Rx} \\
&+ \text{Probe Packet Detection Delay at Rx} \\
&+ \text{Hardware Turnaround Time at Rx} \\
&+ \text{Response Propagation Delay from Rx to Tx} \\
&+ \text{Response Packet Detection Delay at Tx} \qquad (2)
\end{aligned}$$

Both sender and receiver can estimate their packet detection delays for the probe and response packets, as well as their hardware turnaround delays as described above. The receiver includes its delay values in the response packet. The transmitter knows the total round trip delay and its own packet detection delay, and can substitute these delays, as well as the delays in the receiver response packet in Eq. 2 to obtain the two-way propagation delays. The one-way propagation delay is computed as half the two-way propagation delay.

4.3 Compensating for Different Delays

SourceSync uses its measured delays to estimate how long co-senders must wait to ensure that their transmissions arrive synchronized with the lead sender's transmission at the receiver. At a high level, the lead sender initiates transmission by sending a synchronization header. The co-senders hear the synchronization header, switch from reception to transmission, and then begin transmitting their data.

Let d_i be the one-way propagation delay from the lead sender to co-sender i, h_i the hardware turnaround delay of co-sender i, and Δ_i the detection delay for the synchronization header at co-sender i. Co-sender i will not be ready to transmit until after a delay of $d_i+\Delta_i+h_i$. Hence, the lead sender cannot transmit data immediately after the synchronization header, but has to wait for all co-senders to be ready for data transmission. What is the least time necessary to ensure that all co-senders are ready? The 802.11 specification requires that a node should be able to transmit a response within a SIFS after another node transmits a packet to it [37, 11, 27]. Hence, it is sufficient that the lead sender waits for a SIFS (10 μs in 802.11 g/n) after the synchronization header. We will refer to this time, when all co-senders are ready to transmit, as the **global time reference**. Since co-sender i is ready to transmit $d_i+\Delta_i+h_i$ units after the synchronization header, it therefore needs to wait an additional time of $SIFS-(d_i+\Delta_i+h_i)$ to align itself with the global time reference.

Co-senders however should not begin transmission exactly at the global time reference, since different senders have different propagation delays to the receiver. Specifically, if the co-sender is further away from the receiver than the lead sender, it needs to transmit earlier than the global time reference, and if it is closer to the receiver, it needs to transmit after the global time reference. Exactly how much before or after depends on the one-way propagation delays. Let T_0 be the one-way delay from the lead sender to the receiver, and let t_i be the one-way delay from co-sender i to the receiver. Then, co-sender i simply waits for a time of $w_i = T_0 - t_i$ relative to the global time reference to determine when it should transmit.

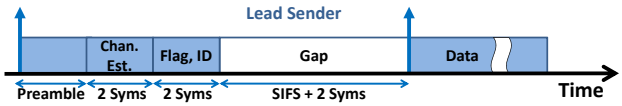
The above algorithm requires the co-senders to know the propagation delay from the lead sender to themselves, and the propagation delay from themselves and the lead sender to the receiver. SourceSync computes these delays by having nodes exchange periodic probes. The packet detection delay and hardware turnaround delays are both computed and compensated for locally at co-senders.

4.4 SourceSync's Synchronization Protocol

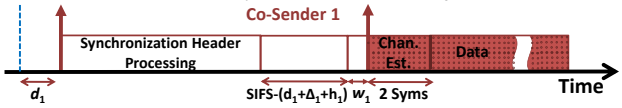
We now describe SourceSync's synchronization protocol, assuming that all co-senders have computed their wait times. For clarity, we fo-

[3] Note that the contribution of detection offset to channel slope is different from carrier frequency offset (CFO) and sampling offset (SFO) estimation. Specifically, the contribution of detection offset to slope is constant across symbols, unlike CFO which does not change the slope, but only shifts the intercept of the line in Fig. 5 from symbol to symbol, and SFO which creates a relative slope between two consecutive symbols [15].

[4] Eq. 2 assumes that the hardware turnaround delay at the transmitter is less than the sum of propagation delays and hardware turnaround delay at the receiver. Note that we can always ensure that this condition holds by adding a constant wait time at the receiver, whose value is known to the transmitter. We drop this detail from the equation for clarity.



(a) Transmission by lead sender for the joint frame.



(b) Transmission by co-sender for the joint frame.

Figure 6: Joint frame from the perspective of the senders. Symbols in solid blue are transmitted by the lead sender, symbols in dotted red by the co-sender, and symbols in white reflect silence periods. The co-sender hears the lead sender's transmission after a delay of d_1, waits for SIFS$-(d_1+\Delta_1+h_1)$ after processing the synchronization header, followed by a wait of w_1, and then begins its transmission.

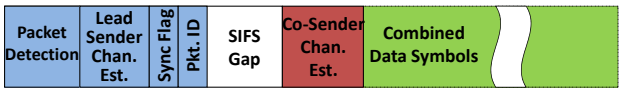


Figure 7: Format of joint frame seen by the receiver.

cus on two concurrent senders. The extension to multiple concurrent senders is straightforward.

The lead sender triggers the joint transmission by transmitting a **synchronization header**. The header contains a standard preamble for packet detection and channel estimation, followed by the lead sender identifier, a flag indicating that this is a joint frame, and a packet identifier (16-bit hash of the IP source address, IP destination address, and the IP identifier). After transmitting the synchronization header, the lead sender goes silent for a duration of SIFS to allow the co-sender to switch from reception to transmission. The lead sender stays silent for an additional duration of two symbols to allow the co-sender to transmit its channel estimation symbols, and then begins transmitting data. The co-sender, on its part, starts by listening on the medium. Once it receives the synchronization header, it continues listening till it has received the packet identifier and then switches from reception to transmission mode. The co-sender then waits for its wait time, w_1, computed as above, and transmits its channel estimation symbols, followed immediately by data. Figs. 6(a) and (b) show the transmission timeline of the joint frame from the perspective of the lead sender and co-sender respectively. As a result of this procedure, the receiver sees a single joint frame as shown in Fig. 7.

Two points are worth noting.

- SourceSync extends directly to more than two senders. In this case, after sending the synchronization header, the lead sender stays silent for the duration of a SIFS to allow all co-senders to switch, followed by two channel estimation symbols for each co-sender.
- The overhead of synchronization is low. In particular, it consists of a SIFS for switching and wait time, and 2 symbols per co-sender channel estimation. For example, in the case of 802.11 using 1460 byte packets, and 12 Mbps transmission rate, the overhead is 1.7% for two concurrent senders, and 2.8% for five concurrent senders.

4.5 Delay Tracking and Mobility

The algorithm described so far ensures that senders can transmit synchronized with each other. But what happens when nodes move? It might seem that the changes in propagation delays resulting from node mobility will necessitate constant probe-response exchanges to recompute these delays, and maintain synchronization. However, SourceSync can deal with mobility without additional probes. Instead, it simply uses data transmissions to continuously adjust wait times at co-senders and keep transmitters synchronized.

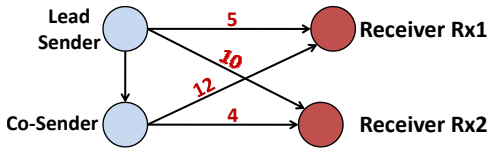


Figure 8: Synchronization at two receivers. One-way delays are shown. No choice of wait time allows perfect alignment at both receivers.

Specifically, for each received joint frame, a SourceSync receiver detects the start of the synchronization header, and computes the channels of the lead sender and the co-sender. It then measures the slopes of both these channels, and translates the measured slopes to symbol offsets using the technique described in §4.2. If the lead sender and co-sender are perfectly synchronized, their symbol boundaries will be aligned, and therefore their computed symbol offsets will also be equal. Otherwise, the difference of the offsets corresponds exactly to the misalignment between the senders. The receiver includes the measured misalignment in its ACK, and the co-sender uses this update to appropriately change its wait time for the following transmission.

4.6 Synchronization at Multiple Receivers

So far, we have focused only on synchronization at a single receiver. However, applications such as opportunistic routing would benefit from synchronization at multiple receivers.

In contrast to synchronization at a single receiver, where an appropriate choice of wait times at co-senders can achieve perfect alignment at the receiver, propagation delays may prevent us from achieving perfect synchronization simultaneously at multiple receivers. Consider the senders in Fig. 8 with one-way delays as shown. To synchronize at Rx1, the co-sender has to start data transmission before the lead sender. But to synchronize at Rx2, the co-sender has to start data transmission after the lead sender. Thus, it is not always feasible to synchronize senders simultaneously at multiple receivers. However, one can still leverage sender diversity gains from joint transmissions by increasing the CP to account for the residual misalignment. The objective of SourceSync in this case is to pick wait times at co-senders so as to minimize the maximum misalignment at all receivers.

SourceSync formulates this problem as a linear program that estimates the optimal wait time, w_i, for co-sender i. Define t_{ij} as the one-way delay from co-sender i to receiver j, and T_j as the one-way delay from the lead sender to receiver j. These values are estimated as in the single-receiver case described in §4.2. The pair-wise misalignment at receiver k of co-sender i with the lead sender can be written as $|(w_i + t_{ik}) - T_k|$, and similarly the pair-wise misalignment with another co-sender j can be written as $|(w_i + t_{ik}) - (w_j + t_{jk})|$. The linear program then chooses the w_i's so as to minimize the maximum pair-wise misalignment across the lead sender and all co-senders. This optimization is a linear program, and can be solved efficiently, especially since the number of co-senders and receivers is usually small, say, < 5. Note that to determine the potential receivers to synchronize at, we use the ETX metric as described in §7.2.

The lead sender performs this optimization and computes the necessary increase in CP as the maximum misalignment across all senders. In order to ensure that all senders in a joint transmission are synchronized throughout the joint frame, it communicates the new CP to co-senders as a field in the synchronization header. Co-senders use this increased CP for the concurrently transmitted data symbols.

5 Joint Channel Estimation

Now that senders are synchronized, the next step is to decode the joint frame at each receiver.We focus on a single OFDM subcarrier since OFDM subcarriers can be decoded independently. For simplicity of exposition, we consider two concurrent senders for the rest of this section. Our technique generalizes to multiple concurrent senders.

Say the two senders are already synchronized, and they both transmit the same symbol x_i in subcarrier i. After the FFT, the receiver receives a symbol y_i in subcarrier i, which is related to the transmitted symbol x_i as $y_i = H_i x_i + n$, where H_i is the composite channel experienced by x_i and n is noise. If the receiver knows the composite channel H_i, it can extract x_i from its received signal as $x_i = \frac{y_i}{H_i}$.

The composite channel, however, is affected by two factors. The first is the individual channels traversed by symbol x_i from each of the senders. The second factor is that each sender has a different oscillator crystal. It is unlikely that different crystals have exactly the same carrier frequency [24], and therefore, each sender has a different frequency offset with respect to the receiver. Hence, the composite channel can be written as:

$$H_i(t) = H_{i,1} e^{j2\pi\Delta f_1 t} + H_{i,2} e^{j2\pi\Delta f_2 t}$$

where $H_i(t)$ is the composite channel in subcarrier i at time t, $H_{i,j}$ are the individual channels in subcarrier i from sender j, $(j = 1, 2)$, and Δf_j is the frequency offset of sender j relative to the receiver.[5]

Since different senders have different frequency offsets, the two components of the composite channel will keep rotating relative to each other. SourceSync addresses this issue by leveraging the observation that the frequency offset is relatively stable over long periods of time. Therefore it can be computed at the same time as the initial pair-wise propagation delay estimation and communicated to each sender, which can then correct for the offset before transmitting by multiplying its transmitted symbol at time t by $e^{-j2\pi\Delta f_i t}$.

Once the transmitter corrects the offset, the receiver can estimate each sender's channel by using the corresponding channel estimation symbols in the joint frame. It can then add the individual channels to estimate the combined channel.

However, this is not sufficient. One can never correct completely for the frequency offset because, even if the estimate is relatively accurate, a small residual error in frequency accumulates over time leading to large phase errors and unrecoverable decoding errors throughout the packet. This is why, even for a single sender-receiver pair, OFDM decoders have to perform phase tracking to correct for residual errors in frequency offset throughout the packet. SourceSync performs phase tracking for the same reason. The difference, however, is that it has to perform independent phase tracking for each of the senders.

To do so, we augment the traditional OFDM algorithm for phase tracking. Specifically, OFDM allocates some subcarriers known as pilots in every data symbol for phase tracking. The exact algorithm for phase tracking is in [15], but the important point here is that the algorithm is designed to correct the residual frequency offset from a single sender. Hence, this algorithm cannot work as such for concurrent senders, since each sender has a different residual frequency offset. We address this issue by sharing the pilots between the concurrent senders across symbols. This is feasible since senders are synchronized and have a common understanding of symbol boundaries. For example, the lead sender can use pilot subcarriers in odd symbols, and the co-sender can use pilot subcarriers in even symbols. The receiver now maintains two residual frequency offset estimates which it applies to the individual channels of the corresponding senders before summing them to compute the composite channel.

[5]The frequency offset is normalized in units of the subcarrier width.

6 Smart Combiner

As stated earlier, even when the senders correct for the frequency offset, there is always a residual frequency error that, over time, causes the channel from each sender to rotate relative to the other. Further, the initial phase of the channel for the two senders at the beginning of a joint frame is random. The consequence of these two behaviors is that the signals from the concurrent senders can combine constructively or destructively depending on the random initial phase and the rotation of the two channels, and the senders cannot know how the signals are going to combine *a priori*. Thus, if the two senders naively send the same signal, some unlucky symbols will observe a deeply faded channel due to destructive combining and the receiver will be unable to decode those symbols.

Let us consider a scenario where the channels from the two senders happen to cancel each other, *i.e.*, $H_{i,1} = -H_{i,2}$. In this case, if the transmitters sent the same data symbol, x_i, the receiver receives $H_{i,1}x_i + H_{i,2}x_i$, which equals 0. Of course, one way to address the problem would be for one transmitter to transmit x_i and the other to transmit $-x_i$. This transformation would transform the destructive composite channel to a channel where the two signals reinforce each other at the receiver. But such a strategy does not always work; if the channels were originally aligned with each other, sending x_i and $-x_i$ would result in a 0 signal at the receiver, transforming the constructive channel into a destructive one! Since the transmitters cannot track the individual channels and their phases ahead of transmission, they need a coding strategy that will provide high throughput irrespective of the relative orientations and magnitudes of the channels.

SourceSync addresses this issue by leveraging space time block codes [39] that cleverly code data across symbols to eliminate deep fades due to destructive combination of signals. Specifically, in the case of two senders, SourceSync uses the Alamouti code [2], which is known to provide the optimal throughput in such a scenario, and has low encoding and decoding complexity. In the case of more than two senders, SourceSync uses a quasi-orthogonal space-time block code [16] that is a simple extension of the Alamouti coding scheme, and retains its simplicity of encoding and decoding. Given a sequence of data symbols, a SourceSync lead sender uses codeword 1 from the replicated Alamouti codebook specified by [16], and co-sender i uses the $(i+1)^{th}$ codeword from this codebook. This sequence of codewords also has the property that the receiver can decode the received frame even if only a subset of intended senders participate in the concurrent transmission. Note that a receiver can determine whether an intended co-sender participates in a transmission based on the presence of energy in the time slots corresponding to the channel estimation symbols of that co-sender.

7 Using SourceSync to harness sender diversity

Now that we have described the components of SourceSync, we explain how SourceSync can be used to harness sender diversity for opportunistic routing and wireless LANs. As we do so, we also explain how SourceSync integrates with the MAC for both scenarios.

7.1 Combining SourceSync with Last Hop Diversity

Consider a client that is in the neighborhood of multiple APs, but has poor connectivity to them. Uplink receiver diversity schemes like MRD, SOFT, and Link-Alike [25, 17, 44] exploit the fact that, while a transmitted packet has low probability of being received correctly by a specific AP, it is likely to be received by at least one AP, and all such APs can combine received packets or bits over the wired

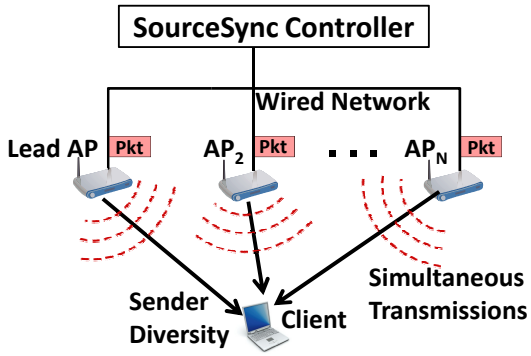


Figure 9: SourceSync for the last hop. SourceSync can harness sender diversity using concurrent transmissions from many APs.

network. SourceSync complements these schemes by enabling sender diversity on the downlink, *i.e.*, instead of a client receiving packets from only one AP at a time, multiple neighboring APs can transmit simultaneously to the client and increase downlink reliability.

SourceSync exploits last-hop diversity using the architecture shown in Fig. 9. We leverage the high bandwidth of the wired network connecting the access points. A SourceSync controller resides on the wired network, and uses it to forward packets arriving from the wired uplink to all the APs in a neighborhood. This enables multiple APs to transmit the same data to a wireless client. Further, the APs have a static ordering that decides which codeword of the space-time block they will utilize for their transmission.

MAC and Association: When a client first joins the wireless network, it associates with multiple, say K, APs in its neighborhood, where K is a tunable parameter. One of these APs, say the one with the best link to the client, is chosen as the lead AP for this client and this information is disseminated to all other APs. All the APs estimate the propagation delays to their associated client. Additionally, the APs can offline estimate their hardware turnaround delays and propagation delays to each other. Each AP then uses this information to calculate its delay compensation, as described in §4.4.

The APs use a contention-based MAC similar to 802.11. The only difference is that when there is a downlink packet destined to a client, only the lead AP contends for the medium. Once the lead AP acquires the medium, it transmits its synchronization header followed by the data. Upon hearing the synchronization header, all other APs join the transmission as described in §4.4.

Similarly to 802.11, a client acknowledges successful receptions. Note that since the ACK is on the uplink, APs can use standard receiver diversity techniques like SOFT [44] or MRD [25] to increase the reliability of ACK reception. Received ACKs are communicated to the lead AP over the wired network. The lead AP initiates retransmissions when it does not receive an ACK, and these retransmissions are joined by the other APs, similarly to the original transmission.

Rate Adaptation: The APs coordinate rate adaptation since all simultaneously transmitted packets must have the same set of data symbols. Rate adaptation in SourceSync is controlled by the lead AP. Specifically, the lead AP runs a standard rate adaptation algorithm such as SampleRate, RRAA or SoftRate [3, 43, 41] which makes rate decisions based on the feedback from the receiver (acknowledgment, soft rate hint *etc.*). The lead AP then includes the chosen rate for the packet in the synchronization header when it initiates transmission. Other APs use this information to pick the right transmission rate. Note that, since SourceSync can leverage power and diversity gains across APs, the combined transmission across APs might be able to use a rate that cannot be used by any individual transmissions.

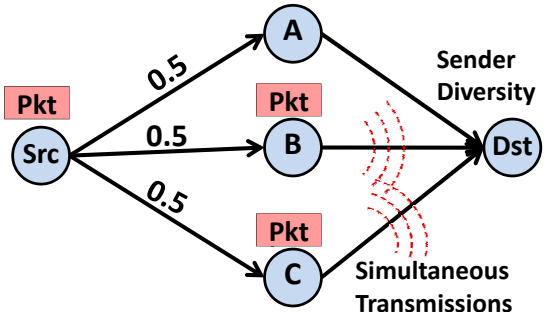


Figure 10: SourceSync with Opportunistic Routing. SourceSync exploits the fact that many relays hear a packet to improve throughput.

7.2 Combining SourceSync with Opportunistic Routing

In this section, we show how to extend opportunistic routing, particularly ExOR [4], to exploit sender diversity. Opportunistic routing has been proposed to deal with lossy links in wireless mesh networks. Consider the example in Fig. 10. Since all links have a loss rate of 0.5, a traditional single-path routing protocol will require an average of two transmissions to deliver a packet from the source to its nexthop router. However, when a source broadcasts its packet, the probability that at least one of these routers will receive it is $1-(0.5)^3$, and hence the expected number of transmissions to deliver a packet is reduced to 1.14. Opportunistic routing protocols exploit this property to decrease loss rates and increase mesh throughput.

However, the same property means that, half the time, multiple routers will receive the packet from the source. Further, the probability of such an event, *i.e.*, multiple routers hearing the same packet increases with the size and density of the network. Existing protocols cannot exploit this property. In contrast, SourceSync can leverage the fact that multiple routers in a mesh overhear the same packet to have these routers transmit the packet simultaneously towards the destination. This form of cooperative forwarding increases the effective transmission power, enabling the packet to make longer jumps towards its destination. Additionally, since the channels from the concurrent transmitters to a downstream node router are unlikely to experience simultaneous deep fading, overall loss rate is reduced.

In the rest of this section, we will describe how to integrate SourceSync with ExOR to provide an opportunistic routing protocol that exploits both sender diversity and receiver diversity. At a high level, ExOR works as follows. Given the link loss probabilities, ExOR computes the ETX metric [8] of each link, and then arranges the nodes in decreasing order of ETX distance from the destination. ExOR is designed for bulk transport. The source operates in batches, and starts by broadcasting all packets in the batch. Any node that overhears the packet can potentially forward it towards its destination. ExOR has a priority scheduler that ensures that each packet is forwarded by the node closest to the destination that has the packet. We refer the reader to [4] for the details of the scheduling algorithm.

MAC: SourceSync retains ExOR's MAC and extends it to allow simultaneous transmission from multiple forwarders. Similar to ExOR, the potential forwarders for a transmission are determined based on ETX measurements, and included in the packet header of a transmission. However, unlike ExOR, SourceSync ensures that when an ExOR forwarder transmits a packet, other nearby forwarders who happen to have overheard this packet join the transmission. This is similar to how neighboring APs join the transmission of a lead AP to provide lasthop diversity as described in §7.1. There is one key difference, however. Unlike in the last-hop scenario where AP transmissions need to be aligned at one receiver, in opportunistic

routing, transmissions from multiple forwarders need to be aligned at multiple receivers. Hence, SourceSync uses the SLS described in §4 to determine both the wait compensation at the forwarders, and the minimum necessary increase in the CP to compensate for misalignment between the receivers. This computation requires forwarders to know the delay differences between various nodes in their neighborhood, and the set of concurrent forwarders and potential receivers for each transmission.

SourceSync computes the delay differences between nodes by running periodic measurements, similar to existing loss rate measurements by mesh routing protocols. SourceSync however does not need to perform delay measurements between all node pairs. A node needs to compute delay differences only to nodes that are potential co-forwarders or potential nexthops. The size of this set dictates the measurement overhead. So, in SourceSync, only nodes that are connected by links with loss probability below a threshold perform pairwise delay measurements. Further, SourceSync leverages data packets from concurrent forwarders to keep updating its estimates of delay differences as described in §4.5.

What happens when all forwarders do not hear a transmission? It is likely that not all forwarders selected during the measurement phase hear all of their intended transmissions. Exchanging information for every packet about exactly which forwarders heard that packet in order to determine the increase in CP, as well as the transmission codeword and wait time to be used by each forwarder will introduce high overhead. SourceSync eliminates the need for such exchanges by leveraging the measurement phase to pick the required wait time and additional CP assuming all forwarders hear a transmission, and also determines the ordering (and therefore codeword) of the forwarders. After this assignment, whenever the lead forwarder transmits, other forwarders hear the synchronization header, which contains the additional CP and identifier of the packet to the transmitted. If a node is in the set of co-forwarders and has the transmitted packet, it joins the transmission using the appropriate wait-time compensation. The node also knows exactly which codeword to use for its transmission based on the precomputed ordering of co-forwarders. For example, say the lead forwarder is node i, and the size of the co-forwarder set is k. The lead forwarder then uses the first codeword, node $i-1$ uses the second codeword, and so on. Of course, not all nodes in the set of potential co-forwarders might hear the packet, or the transmission of the lead forwarder. Note that this does not affect the correctness of SourceSync; a receiver can still decode the concurrent transmission, and garner the benefits of sender diversity from co-forwarders that actually join the transmission.

8 Performance

We have implemented a prototype of SourceSync in FPGA using the WiGLAN radio platform [10] and evaluated it in a wireless testbed.

(a) Hardware: The radio board of our transceiver platform connects to the PC via the PCI bus, and acts like a regular network card. The radio operates in the 802.11a spectrum, has a maximum operating bandwidth of 128 MHz and a symbol time of 1 μs. We configure the radio to use 20 MHz of bandwidth, which is the bandwidth of 802.11 channels. The FPGA is clocked at 128 MHz, and the implementation supports standard 802.11 transmit and receive chains.

(b) Implemented Infrastructure: We implement the components of SourceSync and an infrastructure to evaluate it for last-hop diversity and opportunistic routing. Since symbol-level synchronization requires fine-grained sample level timing, we implement SourceSync in the FPGA, using a combination of Verilog and Simulink. In order to evaluate last-hop and opportunistic diversity, we also implement the following additional components:

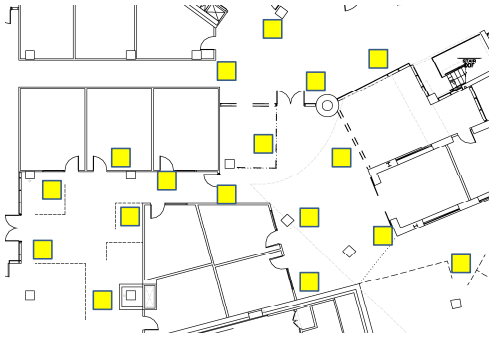


Figure 11: Testbed map. Node locations are highlighted.

(a) SampleRate: We implement SampleRate in our driver, using MadWifi as a reference. We modify SampleRate for SourceSync last-hop diversity to perform rate adaptation only on the lead AP.

(b) ExOR: We use the reference ExOR code and implement a simplified version for our topology, including ETX measurement, forwarder computation, and a priority scheduler.

We evaluate SourceSync in an indoor testbed. Fig. 11 shows the node locations in the experimental environment, which exhibits high diversity due to the presence of walls, metal cabinets, desks, and various combinations of line-of-sight and non-line-of-sight configurations. The exact evaluation methodology and topologies used for each experiment are described below.

8.1 Symbol Level Synchronization

In this section, we show that SourceSync can provide tight symbol level synchronization across nodes, and that without such tight synchronization the system may suffer significant reduction in SNR.

8.1.1 SourceSync provides tight synchronization

First, we investigate whether SourceSync provides accurate symbol-level synchronization across transmitters.

Method. In this experiment, we place a pair of SourceSync nodes acting as lead sender and co-sender, and one node acting as a SourceSync receiver at three randomly chosen locations in our testbed. We synchronize the two transmitters at the receiver using SourceSync, as described in §4.4 and §4.5. Next, we want to measure the resulting synchronization error (*i.e.*, the time difference between transmitters' symbol boundaries). Recall, however, that SourceSync works by measuring synchronization errors and feeding them back to the transmitters in the ACK so they can synchronize their next transmissions, as explained in §4.5. Thus, to measure SourceSync's synchronization error, we need an algorithm that is more accurate than SourceSync in measuring synchronization errors. How do we find such an algorithm? And if such an accurate algorithm exists, why don't we use it in SourceSync?

We can obtain such a highly accurate algorithm if we incur very large overhead. Specifically, instead of computing synchronization errors using only a few symbols at the beginning of each packet, as in SourceSync, we can replace all the data in the packet with known symbols and use the full packet to compute synchronization errors. A SourceSync packet starts with an initial header consisting of the lead sender's synchronization header followed by the co-sender's channel estimation symbols, after which the two senders jointly transmit their data. The regular SourceSync algorithm obtains an estimate

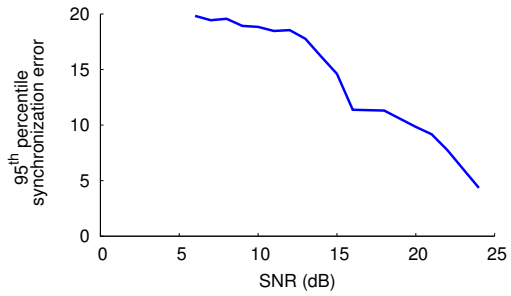


Figure 12: 95^{th} percentile synchronization error. SourceSync ensures that the synchronization error is less than 20 *ns* across the operational range of 802.11 SNRs.

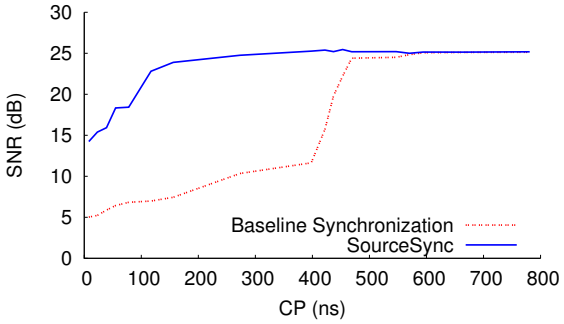


Figure 13: CP reduction with SourceSync. SourceSync enables concurrent transmissions to achieve high SNR with a significantly lower CP than an unsynchronized baseline that does not compensate for delay differences.

of the synchronization error using only the lead sender's synchronization header and the co-sender's channel estimation symbols, as described in §4.4 and §4.5. The error estimation algorithm, on the other hand, replaces the data in each packet with 200 repetitions of the initial header (*i.e.*, the lead sender's synchronization header and the co-sender channel estimation symbols). Since the synchronization error does not change within a packet, the new algorithm can obtain 200 estimates of the synchronization error for each estimate of SourceSync. By taking the average of these 200 estimates, the new algorithm dramatically reduces the estimation noise, and hence obtains an almost error free estimate of synchronization error for that packet. Such an algorithm is fine to evaluate the extent of synchronization error, but its overhead precludes its use in a practical system. For every set of locations, we transmit 2000 such packets and measure the average SNR from the two transmitters, as well as the transmitters' synchronization errors using both SourceSync and the new algorithm. We consider the new algorithm as the ground truth and compute SourceSync's synchronization errors with respect to the new algorithm. We repeat the experiment with multiple randomly chosen location triplets in our testbed.

Results. Fig. 12 shows the synchronization error between the two transmitters when using SourceSync, as a function of the average SNR. The graph shows that SourceSync's synchronization algorithm is robust across a wide range of SNRs. Specifically, the 95^{th} percentile of the synchronization error is less than 20 *ns* for the operational range of 802.11 SNRs. Thus, SourceSync's estimates can be used to perform highly accurate symbol level synchronization.

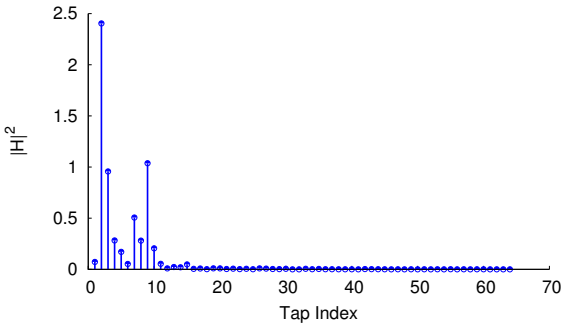


Figure 14: Delay spread of a single sender. The OFDM channel in the time domain has 15 significant taps, which corresponds to the CP length required with synchronization.

8.1.2 The need for accurate synchronization

SourceSync's compensates for delays at senders to synchronize symbols at the receiver, and so that the multipath tolerance of the joint transmission is as good as with a single transmitter. In this section, we evaluate the consequences of loose vs. tight synchronization.

Method. We place two transmitters and the receiver in a random line-of-sight configuration in our testbed. We label one transmitter a lead sender, and the other a co-sender. Both transmitters have identical hardware, and hence the same hardware turnaround delay. The only difference in delays between the transmitters is due to propagation. We compare two schemes: a baseline scheme where the lead sender transmits a synchronization header, and the co-sender joins the transmission without compensating for delay differences, and SourceSync's symbol level synchronization scheme where the co-sender joins the transmission after an appropriate wait time as described in §4.4. For both schemes, we calculate the average receiver SNR of a joint transmission, and perform this calculation for various values of the cyclic prefix (CP).

Results. Fig. 13 plots the SNR of the joint transmission as a function of CP, for SourceSync, and for the baseline. We see that SourceSync requires a far lower CP to achieve the peak SNR of the combined transmission, in comparison with the baseline. In particular, SourceSync requires only a CP of 117 ns (15 samples in our system) to achieve an SNR within 95% of the maximum, whereas the baseline requires a CP of 469 *ns* (60 samples in our system). Two points are worth noting. First, even when the transmitters have identical turnaround times, the baseline increases the required CP by 352 *ns* (45 samples) over what is required by SourceSync. By compensating for delay differences, SourceSync can operate with a much smaller CP, thus significantly increasing the benefits of sender diversity. Second, the baseline has no mechanism to identify the required increase in CP. Without this knowledge, one may pick a CP that is too small, in which case the communication system stops working. To prevent this scenario from occurring, one cannot simply set the CP to 469 *ns* since this value may not work for a different set of senders and receivers. One has to pick a conservative CP that works for any network, and hence incur a large overhead.

Finally, it might seem that SourceSync's SNR decreases at a CP lower than 15 samples due to residual synchronization error. However, this is not the case. The SNR reduction is due to the multipath delays in the channel. One can see this by looking at the time domain representation of the channel from one of the transmitters. Fig. 14 shows the magnitude of the time domain channel as a function of tap index. We see that the channel has around 15 significant taps. Reducing the CP below 15 samples causes symbols to leak into each other, and hence reduces the maximum achievable SNR of the system.

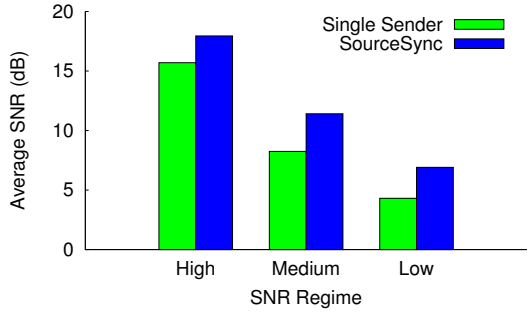


Figure 15: Power gains. SourceSync achieves a 2–3 dB gain over a single sender across the range of SNRs.

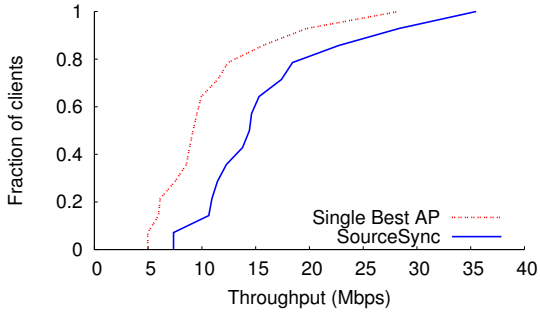


Figure 17: SourceSync at the last hop. The red dotted line is the CDF of throughput using selective diversity (*i.e.* single best AP). The blue solid line is the CDF of throughput using sender diversity across both APs with SourceSync. The CDFs show that sender diversity produces a median gain of 1.57× over selective diversity.

8.2 Power and Diversity Gains

As explained earlier, allowing multiple senders to transmit simultaneously provides both power gains from the addition of the senders' powers, and frequency diversity gains because it is unlikely that the same frequency experiences a fade from all senders to the receiver. In this section, we verify that SourceSync actually provides these gains.

Method. We place the receiver and two transmitters at various random locations in our testbed. For each set of locations, we measure the average SNR across subcarriers, as well as the SNR per subcarrier when each sender transmits separately, and when the two senders transmit in combination using SourceSync. We group the locations into three categories based on the SNRs of the senders transmitting separately: low (<6dB), medium (6–12dB), and high (>12dB).

Results. Fig. 15 plots the average SNR across subcarriers, both for senders transmitting separately, and for joint transmission using SourceSync. As we can see, SourceSync improves the average SNR by 2–3 dB for all SNR ranges. The increase in SNR is due to the addition of power from both senders to the receiver. In particular, simultaneous transmission from two senders whose signals arrive at the receiver with equal power results in an SNR increase of 3 dB.

To understand the gains further, we plot the SNR per subcarrier for all three SNR ranges. We see from Figs. 16(a)-(c) that SourceSync not only improves the average SNR, but has a flatter SNR profile than that of either sender transmitting separately. This shows that SourceSync is able to exploit sender diversity on a per-subcarrier basis. These gains are due to SourceSync's smart combiner (§6) that uses space time block codes at a subcarrier granularity to enable signals from multiple transmitters to combine constructively. The flatter SNR profile is important in channels like 802.11, which exhibit frequency selective fading and different SNRs across subcarriers. Since it is unlikely that both senders will simultaneously experience a fade in the same subcarrier, SourceSync has a flatter SNR curve. 802.11 convolutional codes can be affected by even a few bad subcarriers, and hence, a flatter profile allows the system to achieve significantly higher bitrates with SourceSync than without SourceSync.

8.3 Last Hop Diversity

We now examine the gains from using SourceSync in a last-hop scenario to harness sender diversity gains.

Method. We place the two transmitters, acting as APs, and the receiver, acting as a client, in random testbed locations. For each set of positions, we compute the throughput with each AP acting alone, as well as the throughput of the combined system with SourceSync, using SampleRate [3] for rate adaptation. We repeat the experiment with different sets of random locations.

Results. Fig. 17 shows the CDF of the throughputs obtained for each set of positions using the best AP for the client in that configuration, as well as the throughputs when leveraging diversity across APs using SourceSync. As can be seen, SourceSync provides benefits over selective diversity (*i.e.* using the single best AP) at all client throughputs, with a median throughput gain of 1.57×.

8.4 Opportunistic Routing with SourceSync

We evaluate the gains of SourceSync with opportunistic routing.

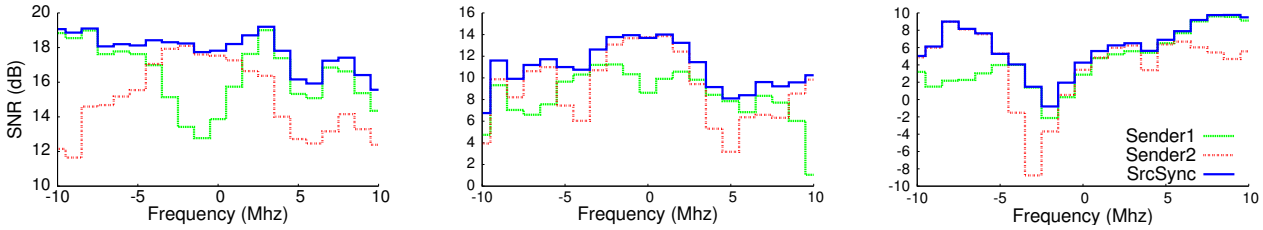
Method. We create a five node topology as follows. We place two nodes, acting as source and destination, at random locations in our testbed. For each choice of source and destination, we place nodes acting as relays in three other random locations between the source and destination location. We measure pairwise loss rates between the nodes, compute the ETX metric for each link, and evaluate three schemes: (a) a single path routing scheme that picks the best relay to route the packets from source to destination, (b) ExOR, which opportunistically uses any of the three relays as forwarders, and (c) a combination of ExOR and SourceSync which also exploits sender diversity to forward from relays to the destination. Since rate adaptation for opportunistic routing protocols is still an open area, we configure the entire network to run at 6 Mbps, and at 12 Mbps, and pick the configuration that provides the highest throughput. We repeat the experiment for 20 different topologies at each rate.

Results. Figs. 18(a) and (b) show the CDF of the throughputs with single-path routing, ExOR, and the combination of ExOR and SourceSync. As would be expected, ExOR can harness gains from receiver diversity from the source to the relays, and provide a median throughput gain of 1.26–1.4× over single path routing. SourceSync can provide additional gains of 1.35–1.45× over the receiver diversity in ExOR by exploiting sender diversity from the relays to the destination. Further, SourceSync and ExOR work in tandem and provide a median throughput gain of 1.7–2× over single path routing.

9 Conclusion

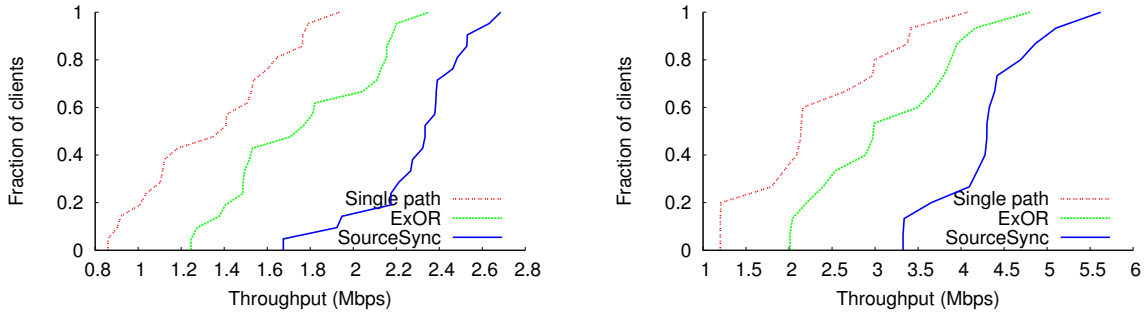
This paper introduces SourceSync, a distributed wireless architecture that exploits sender diversity and demonstrates its practicality via implementation and testbed evaluation. It integrates sender diversity with last-hop diversity and opportunistic routing, showing that this synergy can significantly improve throughput.

We believe that SourceSync has wider implications for wireless design than explored here. Techniques such as distributed beamforming [34] and lattice codes [28] promise significant throughput improvements in theory. However, these techniques have hitherto not



(a) High SNR (b) Medium SNR (c) Low SNR

Figure 16: Frequency diversity gains. SourceSync improves the SNR in each sub-carrier and creates a flatter SNR profile.



(a) Bitrate of 6 Mbps (b) Bitrate of 12 Mbps

Figure 18: SourceSync with opportunistic routing. SourceSync together with ExOR provides gains both over ExOR alone, and over traditional single path routing. The median gains are 1.26-1.4× over single path routing, and 1.35-1.45× over ExOR, depending on the bitrate.

been used in practice because they require some form of symbol synchronization. The synchronization mechanisms in this paper provide a first step toward practical implementations of these techniques.

Acknowledgments: We thank the reviewers, and our shepherd, Matt Welsh, for their comments. This work is funded by DARPA IT-MANET and an NSF award CNS-0831660.

References

[1] D. Agrawal, V. Tarokh, A. Naguib, and N. Seshadri. Space-time coded OFDM for high data-rate wireless communication over wideband channels. In *IEEE VTC*, volume 3, 1998.

[2] S. Alamouti. A simple transmit diversity technique for wireless communications. *IEEE Journal on selected areas in communications*, 16(8):1451–1458, 1998.

[3] J. Bicket. Bit-rate selection in wireless networks. Master's thesis, MIT, 2005.

[4] S. Biswas and R. Morris. Opportunistic routing in multi-hop wireless networks. In *ACM SIGCOMM*, Philadelphia, PA, 2005.

[5] S. Chachulski, M. Jennings, S. Katti, and D. Katabi. Trading structure for randomness in wireless opportunistic routing. In *ACM SIGCOMM*, Kyoto, Japan, 2007.

[6] W. Choi and J. G. Andrews. Downlink performance and capacity of distributed antenna systems in a multicell environment. *IEEE Trans. on Wireless Comms.*, 6(1):69–73, January 2007.

[7] Distributed Antenna Systems - No Replacement for Wireless Strategy. `http://medicalconnectivity.com/2008/02/05/distributed-antenna-systems-no-replacement-for-wireless-strategy/`.

[8] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. In *ACM MobiCom*, San Diego, California, September 2003.

[9] A. Dutta, D. Saha, D. Grunwald, and D. Sicker. SMACK: a SMart ACKnowledgment scheme for broadcast messages in wireless networks. In *ACM SIGCOMM*, Barcelona, Spain, 2009.

[10] F. Edalat. *Real-time Sub-carrier Adaptive Modulation and Coding in Wideband OFDM Wireless Systems*. PhD thesis, Massachusetts Institute of Technology, 2008.

[11] Local and metropolitan area networks–specific requirements part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. *IEEE Std 802.11g*, June 2003.

[12] A. Goldsmith. *Wireless Communications*. Cambridge University Press, 2005.

[13] S. Gollakota and D. Katabi. ZigZag Decoding: Combating Hidden Terminals in Wireless Networks. In *ACM SIGCOMM*, Seattle, WA, 2008.

[14] D. Halperin, T. Anderson, and D. Wetherall. Taking the sting out of carrier sense: Interference Cancellation for wireless LANs. In *ACM Mobicom*, San Francisco, CA, 2008.

[15] J. Heiskala and J. Terry. *OFDM Wireless LANs: A Theoretical and Practical Guide*. Sams Publishing, 2001.

[16] H. Jafarkhani. A quasi-orthogonal space-time block code. *IEEE Trans. on Comms.*, 2001.

[17] S. Jakubczak, D. Andersen, M. Kaminsky, K. Papagiannaki, and S. Seshan. Link-alike: Using Wireless to Share Network Resources in a Neighborhood. In *MC2R*, 2008.

[18] S. Katti, S. Gollakota, and D. Katabi. Embracing Wireless Interference: Analog Network Coding. In *ACM SIGCOMM*, Kyoto, Japan, 2007.

[19] S. Katti, D. Katabi, H. Balakrishnan, and M. Medard. Symbol-level Network Coding for Wireless Mesh Networks. In *ACM SIGCOMM*, Seattle, WA, 2008.

[20] G. Kramer, I. Marić, and R. Yates. Cooperative communications. *Found. in Networking*, 1(3), 2006.

[21] J. N. Laneman, D. N. C. Tse, and G. W. Wornell. Cooperative diversity in wireless networks: Efficient protocols and outage behavior. *IEEE Trans. on Inform. Theory, Dec 2004*.

[22] X. Li. Space-time coded multi-transmission among distributed transmitters without perfect synchronization. *Signal Processing Letters, IEEE*, 11(12):948 – 951, Dec. 2004.

[23] J. Manweiler, N. Santhapuri, S. Sen, R. Roy Choudhury, S. Nelakuditi, and K. Munagala. Order matters: transmission reordering in wireless networks. In *ACM MobiCom*, Beijing, 2009.

[24] H. Meyr, M. Moeneclaey, and S. A. Fechtel. *Digital Communication Receivers: Synchronization, Channel Estimation, and Signal Processing*. John Wiley, 1998.

[25] A. Miu, H. Balakrishnan, and C. E. Koksal. Improving loss resilience with multi-radio diversity in wireless networks. In *ACM MobiCom*, Cologne, Germany, 2005.

[26] A. K. L. Miu, G. Tan, H. Balakrishnan, and J. G. Apostolopoulos. Divert: Fine-grained path selection for wireless lans. In *ACM MobiSys*, Boston, MA, 2004.

[27] Local and metropolitan area networks–specific requirements part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. *IEEE Std 802.11n*, Oct. 2009.

[28] B. Nazer and M. Gastpar. The case for structured random codes in network capacity theorems. *European Transactions on Telecommunications*, 19(4):455–474, 2008.

[29] A. V. Oppenheim, A. S. Willsky, and S. H. Nawab. *Signals & systems*. Prentice-Hall, 1996.

[30] H. Rahul, F. Edalat, D. Katabi, and C. Sodini. Frequency-Aware Rate Adaptation and MAC Protocols. In *ACM MobiCom*, Beijing, China, September 2009.

[31] E. Rozner, J. Seshadri, Y. Mehta, and L. Qiu. SOAR: Simple Opportunistic Adaptive Routing Protocol for Wireless Mesh Networks. In *IEEE TMC* , 2009.

[32] A. Sendonaris, E. Erkip, and B. Aazhang. User cooperation diversity. Part I. System description. *IEEE Trans. on Comms.*, 51(11):1927–1938, 2003.

[33] A. Sendonaris, E. Erkip, and B. Aazhang. User cooperation diversity. Part II. Implementation aspects and performance analysis. *IEEE Trans. on Comms.*, 51(11):1939–1948, Nov. 2003.

[34] S. Shamai, O. Somekh, and B. M. Zaidel. Multi-cell communications: An information theoretic perspective. In *Workshop on Communications and coding*, 2004.

[35] O. Shin, A. Chan, H. Kung, V. Tarokh, et al. Design of an OFDM cooperative space-time diversity system. *IEEE Transactions on Vehicular Technology*, 56(4):2203, 2007.

[36] Further advancements for E-UTRA: Physical layer aspects, rel. 9, June 2009. Tech Specification Group Radio Access Network.

[37] Local and metropolitan area networks–specific requirements part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. *IEEE Std 802.11a*, June 2003.

[38] Local and metropolitan area networks, part 16: Air interface for fixed broadband wireless access systems: Amendment 1: Multihop relay specification, May 2009. IEEE.

[39] V. Tarokh, N. Seshadri, and A. Calderbank. Space-time codes for high data rate wireless communication: Performance criterion and code construction. *IEEE Trans. on Inform. Theory*, 44(2), 1998.

[40] S. Verdu. *Multiuser Detection*. Cambridge University, 1998.

[41] M. Vutukuru, H. Balakrishnan, and K. Jamieson. Cross-Layer Wireless Bit Rate Adaptation. In *ACM SIGCOMM*, Barcelona, Spain, August 2009.

[42] C. Williams, S. McLaughlin, and M. Beach. Robust OFDM timing synchronisation in multipath channels. *EURASIP Jrnl on Wireless Comm. and Networking*, 2008:7, 2008.

[43] S. H. Y. Wong, H. Yang, S. Lu, and V. Bharghavan. Robust rate adaptation for 802.11 wireless networks. In *ACM MobiCom*, Los Angeles, CA, 2006.

[44] G. Woo, P. Kheradpour, and D. Katabi. Beyond the bits: Cooperative packet recovery using PHY information. In *ACM MobiCom*, Montreal, QC, 2007.

[45] J. Zhang, J. Jia, Q. Zhang, and E. M. K. Lo. Implementation and Evaluation of Cooperative Communications in Software-Defined Radio Testbed. In *INFOCOM*, San Diego, CA, 2010.