Strider: Automatic Rate Adaptation and Collision Handling

Aditya Gudipati, Sachin Katti
Stanford University
{adityag1,skatti}@stanford.edu

Abstract

This paper presents the design, implementation and evaluation of Strider, a system that automatically achieves almost the optimal rate adaptation without incurring any overhead. The key component in Strider is a novel code that has two important properties: it is *rateless and collision-resilient*. First, in time-varying wireless channels, Strider's rateless code allows a sender to effectively achieve almost the optimal bitrate, without knowing how the channel state varies. Second, Strider's collision-resilient code allows a receiver to decode both packets from collisions, and achieves the same throughput as the collision-free scheduler. We show via theoretical analysis that Strider achieves Shannon capacity for Gaussian channels, and our empirical evaluation shows that Strider outperforms SoftRate, a state of the art rate adaptation technique by 70% in mobile scenarios and by upto $2.8\times$ in contention scenarios.

Categories and Subject Descriptors

C.2 [**Computer Systems Organization**]: Computer-Communication Networks

General Terms

Algorithms, Performance, Design

1. INTRODUCTION

Rate adaptation techniques face two challenging scenarios in wireless networks: time varying wireless channels and contention. To pick the right bitrate in time-varying wireless channels, nodes have to continuously estimate channel quality either via probing [2, 13, 21, 15] or by requiring channel state feedback from the receiver [4, 33]. However, probing is inaccurate since packet loss is a coarse measure of channel strength. Channel state feedback from the receiver is more accurate but incurs larger overhead, and can still be inaccurate in mobile scenarios when the channel varies every packet. The bitrate adaptation decision in contention is the opposite of time-varying wireless channels, i.e. do not adapt the bitrate due to contention related losses. Hence in such scenarios, nodes have to use probe packets such as RTS/CTS [1, 35] or require explicit notification from the receiver [33, 26] to discern the cause of packet loss and avoid making an incorrect bitrate change. Both techniques again incur overhead and reduce throughput.

Prior work has made considerable progress in reducing the overhead and improving accuracy of bitrate adaptation, but the conventional wisdom is that there is a fundamental undesirable tradeoff between accuracy and overhead that cannot be avoided. Higher overhead lowers network goodput, but inaccurate bitrate adaptation also significantly affects network performance. The performance impact is especially bad in mobile or high contention scenarios.

In this paper we present **Strider** (for **Stri**pping Deco**der**, our decoding algorithm), a system that eliminates the undesirable tradeoff between overhead and accuracy for rate adaptation. Strider designs a novel coding technique that allows a node to achieve almost the optimal bitrate adaptation possible in any scenario without incurring any overhead. Strider's code design has two important characteristics:

- Strider's code is *rateless*. Hence, senders do not have to perform any probing or require any channel state feedback or adjust their bitrates, they simply create a continuous stream of encoded packets using Strider's algorithm until the receiver decodes and ACKs. We show that Strider's technique achieves the same effective throughput as the omniscient conventional scheme which knows the channel state exactly in advance and always picks the right bitrate to transmit at.
- Strider's code is *collision-resilient*, i.e. it can take collided packets and decode the individual packets from them. Hence there is no need for the senders to discern the cause of packet losses and take measures to avoid collisions. We show that Strider's collision-resilient code achieves at least the same effective throughput as the omniscient collision-free scheduler, i.e. a scheme which knows exactly what nodes are contending in advance and schedules them in a collision-free manner.

The key intuition behind Strider is the concept of a *minimum distance transformer* (MDT). The MDT technique works by transmitting linear combinations of a batch of conventionally encoded symbols (e.g. QPSK symbols encoding bits that have been passed through a $1/5$ rate convolutional code). The intuition is that when we take a batch of L conventional symbols, and transmit M linear combinations of them, in essence we are mapping points from a L dimensional space (the conventional symbols) to points in a M dimensional space. Depending on the relative values of M and L, the minimum distance in this new space can be controlled. Since every channel code has a threshold minimum distance above which it can be decoded correctly, in Strider a sender can transmit linear combinations until the minimum distance in the new space goes above the required threshold and the packets are decoded correctly. Moreover, the minimum distance adjustment happens without any feedback from the receiver or probing by the sender. Hence the technique automatically

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGCOMM'11, August 15–19, 2011, Toronto, Ontario, Canada.
Copyright 2011 ACM 978-1-4503-0797-0/11/08 ...$10.00.

achieves the best bitrate, since in effect it is finding the densest constellation possible that still allows correct decoding, which is what conventional rate adaptation protocols are attempting to accomplish.

The other important component of Strider is collision resilience. Currently, collided packets are thrown away and receivers wait for retransmissions, hoping they wont collide again. Instead, Strider decodes both (or more) packets from a collision. The key reason for our collision resilience is Strider's rateless code: it allows the receiver to treat the packets from the sender with the weaker channel as noise, and due to its rateless property, after the receiver has accumulated sufficient transmissions it can decode the packet from the sender with the stronger channel. After decoding the first packet, we can subtract its contribution from the received signal and decode the packets from the other sender. Hence, Strider's code has the nice benefit of completely eliminating hidden terminals.

We show theoretically that Strider's code asymptotically achieves Shannon capacity for AWGN channels. Further, Strider's algorithm has linear-time computational complexity and is efficient to implement. We have prototyped Strider in the GNURadio [6] SDR platform and evaluated it in an indoor testbed via experiments using USRP2s, as well as trace drive simulations. We compare Strider to the omniscient scheme and SoftRate [33], a state of the art conventional rate adaptation scheme. The omniscient scheme has perfect channel knowledge, and always picks the optimal bitrate and schedules nodes in a collision-free manner. Our evaluation shows that:

- Strider achieves a performance that is within 5% of the performance of the omniscient scheme across a wide range of SNRs (5-25dB).
- When collisions happen, our results show that Strider does at least as well as the omniscient collision free scheduler, and surprisingly in many cases Strider does better! The reason as we discuss later is that collision free scheduling is actually sub-optimal and in many cases concurrently transmitting and applying our technique can deliver even higher throughput. Strider thus completely eliminates hidden terminals in our testbed.
- In comparison with SoftRate [33], we show that Strider achieves nearly 70% throughput improvement in mobile scenarios. Further in networks with contention/hidden terminals, Strider provides a $2.8\times$ increase over SoftRate.

Strider is related in spirit to prior work on rateless codes such as fountain codes [19]. However, these rateless codes work only when the packets are correctly decoded, and cannot handle wireless distortions such as noise and interference. Similarly, prior work on incremental redundancy and hybrid ARQ [29, 18, 7] provides a limited form of rate adaptation by adaptively providing the right amount of redundancy needed to enable decoding of partially correct packets. However, these techniques still have to pick the right modulation, and further do not work in the presence of collisions or interference. Strider provides complete rate adaptation, and handles collisions and interference in a single framework.

2. RELATED WORK

There is a large body of prior work on rate adaptation. Most techniques use one of two approaches: estimate channel strength via direct channel state feedback (in the form of SNR or BER measurements) from the receiver, or infer channel strength based on packet delivery success/failures [4, 33, 2, 13, 21, 15, 27]. Channel state feedback in fast changing mobile channels can be expensive, and worse yet, inaccurate since by the time the transmitter uses the feedback, the channel might have changed. Inference based on packet delivery success can be highly inaccurate, since packet delivery is a very coarse measure of channel strength. Further, none of these techniques work when there are collisions, and therefore often need to augment the rate adaptation protocol with extra overhead in the form of probing or feedback from the receiver to discern whether the packet loss was caused by a collision.

Strider is related to prior work in rateless codes and hybrid ARQ. Rateless codes such as LT [19] and Raptor codes [28] allow one to automatically achieve the capacity of an erasure channel without knowing the packet loss probability in advance. However these techniques require whatever packets are received to be correctly decoded, and do not work in wireless channels where packets are corrupted [28]. Second, hybrid ARQ schemes used in 4G wireless systems based on punctured turbo codes [18, 29] can be used to selectively provide extra redundancy in the form of coded bits, to help the receiver decode an erroneous frame. However, these techniques still need to pick the correct modulation and further do not work when there are collisions or external interference.

Strider's collision resilience component is related to prior work on interference cancellation [10, 9, 16, 17]. However, all prior techniques require that the colliding packets be encoded at the correct bitrate to enable them to decode collisions. For example in SIC, if the colliding packets have been encoded at a bitrate corresponding to the idle channel (which will happen because the colliding hidden terminal senders cannot know in advance that they will collide), SIC will fail to work [9]. Zigzag has a similar but less acute problem, since it also needs correct decoding of its interference free chunks, which requires the packets to be encoded at the correct bitrate. Further Zigzag needs the same set of packets to collide across successive collisions. Strider does not have any of these problems, since its rateless property automatically adjusts the effective bitrate to enable its stripping decoder to decode collisions, and it can decode even if collisions are between different sets of packets.

Strider's design is inspired by recent work on uplink power control in cellular CDMA systems, as well as theoretical work on rateless code design [3, 34, 23, 5, 24]. As we describe later, Strider treats each packet transmission as a set of virtual collisions among independent blocks, very similar to how multiple packet transmissions in a CDMA uplink wireless system collide. Hence decoding algorithms that are used in CDMA basestations have a similar structure to Strider's algorithm. Specifically, they need to control how power is allocated to each uplink transmitter to enable successful decoding, and we borrow from such algorithms to design Strider's encoding algorithm. The key contribution of Strider is the application of these techniques to design a rateless code for wireless channels, as well as an implementation and detailed evaluation of the technique using practical software radio experiments. Further, we also design a novel technique that extends the code design to handle packet collisions.

3. INTUITION

Senders have to adapt bitrates because of the threshold behavior of conventional techniques, i.e. they decode only at or above a particular SNR threshold depending on the coding rate and modulation choice. Even though it is fairly introductory material, we first discuss the reasons for this thresholding behavior since it provides insight into our eventual design.

In current schemes, data bits are first channel coded to add protection against noise. The level of protection is parameterized by the coding rate (e.g a $1/2$ rate code implies that every data bit is protected with one extra bit of redundancy). Coded bits are then modulated, i.e. they are mapped to points in a complex constellation and transmitted on the wireless channel. For example in BPSK, bits are mapped to two points on the real line $(\sqrt{P}, -\sqrt{P})$ (P is the transmission power) and transmitted. Due to attenuation and additive noise the re-

ceiver gets $y = x + n$, where n is Gaussian noise with variance σ^2. When decoding, the receiver first demodulates the received symbol, i.e. maps it to the nearest constellation point and infers what coded bit was transmitted. Hence if the Gaussian noise value is greater/-less than $(\sqrt{P}/-\sqrt{P})$ the receiver makes a bit error. However, the channel code decoder can correct a certain number of errors (depending on the amount of redundancy added) and decode the final data. Thus as long as the number of bit flips at the demodulation (BPSK) stage are less than the correcting power of the channel code, the data eventually gets decoded correctly.

Assuming the channel code rate is fixed, the key to ensuring decoding success is to make sure that the demodulation stage does not make more bit errors than the channel code can handle. This error rate is dictated by the *minimum distance* between any two constellation points (e.g. for BPSK it is $2\sqrt{P}$) and how it compares with the noise power (σ^2). To get good performance, the minimum distance has to be sufficiently large so that no more than the tolerable number of bit errors occur. If its too small, the channel code cannot correct, if its too large, the extra redundancy in the channel code is wasteful. Modulation schemes have different minimum distances (e.g. BPSK, QPSK, 16-QAM, 64-QAM have successively decreasing minimum distances), and depending on the channel SNR, the rate adaptation module's job is to pick the combination of modulation and channel coding that correctly decodes and maximizes throughput. Hence, different modulation and channel coding schemes have different SNR thresholds above which they begin to decode.

3.1 Our Approach

Strider takes a conventional fixed channel code and constellation which works only above a particular threshold SNR, and makes it ***rateless***. In other words, it enables the fixed channel code and constellation to decode at any SNR. We refer to this fixed channel code and constellation as the *static code* in the rest of the paper. For exposition simplicity, we assume in this section that the static code is using BPSK, but the actual implementation uses QPSK.

The key idea behind Strider's rateless transformation is the concept of minimum distance transformation (MDT). Intuitively, MDT takes a batch of symbols from the static code and maps them to a different space where the minimum distance between the two closest points can be tuned to meet the static code's requirements. To understand how MDT works, we begin with a simple (but suboptimal) approach that demonstrates the basic idea. Assume we have a BPSK symbol x from the static code. A simple approach to amplify the minimum distance is to take the symbol x, and transmit it multiple (M) times, but multiply each transmission by a complex number of unit magnitude but random phase $r_i = e^{j\theta_i}$ (so transmission power does not change). The receiver therefore gets the following symbols after noise gets added

$$\vec{y} = \vec{r}x + \vec{n} \tag{1}$$

where $\vec{r}$ is the M length vector of random complex numbers formed by the coefficients of each transmission, and $\vec{n}$ is the noise vector for the M transmissions.

The transmitter in essence has mapped a simple BPSK symbol x to a random point $\vec{y}$ in a M-dimensional space. To see why this amplifies minimum distance, lets compute the Euclidean distance in this new space between the original two BPSK constellation points $\sqrt{P}, -\sqrt{P}$. The new distance is $||2\sqrt{P}\vec{r}|| = 2\sqrt{MP}$, which is $\sqrt{M}$ times the original minimum distance, providing much higher resilience to noise. At some value of M (i.e. after a certain M number of transmissions), the static code will meet its minimum distance threshold and be able to decode.

As the reader can tell, the above naive approach is quite inefficient. It increases the minimum distance in large increments, whereas the static code itself might need a much smaller increment to decode. Our key observation is that instead of operating over single symbols as above, we can spread the transmission power over a batch of K symbols belonging to K parallel blocks (each block is generated by passing data through the static code). Specifically, instead of transmitting the K symbols separately one by one, Strider transmits random linear combinations of the K symbols

$$\sqrt{(1/K)}\left(\sum_{i=1}^{i=K} r_i x_i\right) \tag{2}$$

where the $\sqrt{1/K}$ factor is needed to ensure that every transmitted symbol has a power of P.

The transmitter picks separate random coefficients for each linear combination. Assuming the transmitter has to send M such linear combinations, the receiver receives the following system of linear equations distorted by noise.

$$\vec{y} = \sqrt{(1/K)}\mathbf{R}\vec{x} + \vec{n} \tag{3}$$

where $\vec{x}$ is the K length vector corresponding to the batch of K static code symbols, $\mathbf{R}$ is the $M \times K$ matrix consisting of the random phase coefficients r_i defined above, and all the other definitions are the same.

To understand how this technique achieves minimum distance transformation, we can use the following visualization. Intuitively, this operation is taking K dimensional vectors $\vec{x}$ and mapping it to random points in a M dimensional space. As M increases, the minimum distance between the two closest points in this new space increases. When $M = 1$ the minimum distance is $2\sqrt{P/K}$. For any value M, the minimum distance between points in the M-dimensional space corresponding to the closest constellation points for the static code symbols x_i (assuming BPSK) is $||2\mathbf{R}(\mathbf{i})\sqrt{P/K}|| = 2\sqrt{MP/K}$, where $\mathbf{R}(\mathbf{i})$ is the i'th column of matrix $\mathbf{R}$. Thus the minimum distance increases monotonically with M. Hence, by controlling the value of M (i.e. by controlling the number of transmissions), we can control the minimum distance until the static code's requirements for each of the K blocks are met and they can decode. Thus, we can keep on transmitting linear combinations until all the K blocks can be decoded.

Stepping back, Strider's technique has taken a static code that used to operate at or above a fixed SNR threshold, and converted it using MDT to work at any SNR by adjusting the minimum distance. In other words, we have converted the static code to be *rateless*. To decode, the receiver estimates what are the likely symbols $\vec{x}$ given $\vec{y}$ and the matrix $\mathbf{R}$, and then passes the K symbols through the decoder for the static code. In the following section, we describe the design of an efficient algorithm that realizes this insight, as well as extend it to decode collided packets.

4. DESIGN

First, we describe the two main design goals for Strider, and discuss them in the context of how these goals fit into the larger picture of code design:

- **Complexity of the decoding algorithm**: The efficiency of a code (defined as how close it's achieved throughput is to the Shannon capacity at any SNR) is typically proportional to the computational complexity of the decoding algorithm. For example, Shannon himself used a random codebook construction that achieves channel capacity, but incurs exponential decoding computational complexity. Algorithms such as sphere decoding and maximum likelihood (ML) [14] decoders try to mimic the random decoding structure of Shannon's design and hence

perform quite well, but still require at least cubic complexity, if not more. Recent code designs such as LDPC codes are the one exception to the rule, since they come close to achieving capacity yet only have linear decoding computational complexity. For practical implementations, low complexity algorithms are of course highly desirable. Our goal is to design an efficient code with ***linear decoding computational complexity***.

- **Feedback from the receiver**: In conventional code design, feedback from the receiver is often quite helpful in improving performance. For example, HARQ systems [29] use feedback from the receiver to determine how many extra parity bits to transmit and minimize wasteful transmission. Note that this is not channel-state feedback, but rather feedback about what data the receiver has already decoded. However, even such feedback can be expensive in wireless since spectrum is scarce, and can complicate protocol design since these feedback packets need to be scheduled and reliably delivered by the MAC protocol. More importantly, such feedback goes against the grain of rateless code designs [28, 19] which strive to operate so that the receiver has to only send a single ACK packet when *all* packets have been successfully decoded. Our goal is to design a code that requires the minimum possible feedback, i.e. it requires only ***one bit*** **of feedback from the receiver** when it has successfully decoded everything that was transmitted. The negligible feedback requirement simplifies protocol design.

Strider's encoding and decoding algorithms meet the above two design goals. Before describing the algorithm however, we summarize the operational algorithm in Strider to give the reader an overview of the end-to-end protocol, and also to harmonize notation. When a node has data to transmit, it uses the following four simple steps:

1. Data is divided into chunks of size 6KB. In each chunk we have $K = 33$ data blocks of length $L = 1500$bits each.
2. Each of the K data blocks is passed through the *static code* (currently we use a $1/5$ rate channel code and a QPSK constellation as the static code), to produce K blocks with $5L/2$ complex symbols each.
3. The K blocks are passed through Strider to create a packet for transmission.
4. Use the standard carrier sense mechanism to check if the medium is idle, and if it is transmit the packet. After transmission, wait for an ACK, which the receiver sends if it has successfully decoded the entire chunk consisting of K blocks. If no ACK is received, go to step 3 and repeat. Move to step 1 when an ACK is received.

We expand on Step 3 which is the core encoding step in Strider. To produce a packet for transmission, Strider linearly combines the K coded blocks from the static code to create one packet. For example to create the first symbol in the transmitted packet we would do the following computation

$$s_1 = r_1 x_{11} + r_2 x_{21} + \ldots + r_K x_{K1} \quad (4)$$

where x_{i1} is the first complex symbol in the i'th block, and r_i is the i'th complex coefficient used to create the linear combination. The computation is repeated with the same coefficients for all L symbols in each block. We assume that the random coefficients have been normalized so that the energy of the symbols is P, the transmission power budget. The above technique produces one packet for transmission. The header of each packet includes the coefficients used to create the linear combination of the blocks (i.e. the symbols $r_1 \ldots r_K$).

The sender creates packets using different linear combinations for each packet, and transmits them until the receiver can decode all K blocks and ACKs. Lets assume the receiver requires M packets before it can decode all K blocks. We can express the i'th symbol in each of the M packets received as:

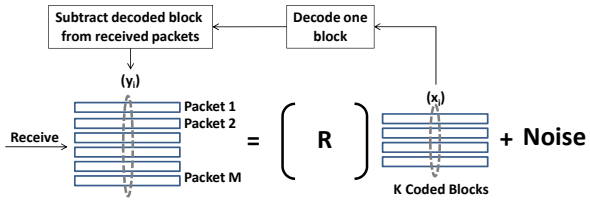


Figure 1: Strider's decoding algorithm

$$\vec{y_i} = \mathbf{R}\vec{x_i} + \vec{n} \quad (5)$$

where $\vec{y_i}$ is the M length vector consisting of the received symbols at the i'th position from the M received packets, and $\vec{x_i}$ is the K length input vector consisting of the i'th input symbols from the K coded blocks. $\mathbf{R}$ is the $M \times K$ matrix consisting of the coefficients used in creating the linear combinations for the M transmitted packets, with each row corresponding to one received packet. Finally $\vec{n}$ is the noise vector. Note that we did not include the channel attenuation in the above equation, we assume that the noise power has been scaled appropriately to account for channel attenuation.

4.1 Decoding Algorithm

As discussed before, we can visualize Strider as mapping K dimensional vectors ($\vec{x_i}$) to M dimensional vectors that are distorted by noise after they pass through the wireless channel to produce $\vec{y_i}$. Since the components of $\vec{x_i}$ can only take four discrete values (the four points of a QPSK constellation), the vector $\vec{x_i}$ can take at most 4^K different values. Since $\mathbf{R}$ is known to the receiver, the receiver can exactly estimate what 4^K possible points could have been transmitted. Hence one method to decode would be to calculate the closest constellation point among the 4^K possible points and then lookup the corresponding input. From that point, we can apply the traditional decoder for the static code to the K coded blocks and recover the original data.

However, this naive technique quickly gets complicated. For example, if $K = 10$ then the number of possible constellation points is $4^{10} = 1048576$! To compute the closest point, the decoder would require exponential memory and compute resources, which rules out this naive method.

4.1.1 Stripping Decoder

Strider's key insight is that *instead of trying to decode the entire vector $\vec{x_i}$ at once which incurs exponential complexity, we can try to decode it one component at a time*. Since each component in $\vec{x_i}$ can at most take 4 discrete values (QPSK), the computational complexity is significantly lower. Hence, Strider first decodes the first coded block's components, and passes them through the decoder for the static code to recover the original symbols. If decoding is successful, we can re-encode the first block and subtract it from the received vectors ($\vec{y_i}$) to remove the effect of the first coded block. Next, we can proceed to the second block and repeat the above process.

One way to visualize Strider's decoder is as follows: remember the received packets are each a linear combination of blocks belonging to one chunk. Strider's decoder is in effect trying to decode one block at a time, *strip* it from the received signals, and then decode the next block and strip it, and so on. Hence, we christen the scheme Strider for *stripping decoder*.

Operationally, the above intuition implies that Strider attempts to decode the first block while treating the other $K - 1$ blocks as interference. The algorithm would work as follows for the first block:

1. Take $\vec{R_1}$ (the first column of matrix $\mathbf{R}$) and form its complex transposed conjugate $\vec{R_1^*}$.
2. Take the dot product of $\vec{y_j}$ with $\vec{R_1^*}$ to obtain one symbol. Repeat for all $j = 1 \ldots L$ to obtain L complex symbols.
3. Attempt to decode the L symbols obtained in the previous step using the decoder for the static code.
4. If decoding is successful [1], block 1 is obtained. Subtract the symbols corresponding to block 1 from the received symbols, i.e. subtract $(x_{1j}\vec{R_1})$ from $\vec{y_j}$ to obtain a new vector $\vec{y_j'}$, and remove the first column from $\mathbf{R}$ to obtain a new matrix $\mathbf{R}'$. Go to step 1 and attempt to decode the second coded block using the same steps but with the new $\vec{y_j'}$ and $\mathbf{R}'$. Repeat until all blocks are decoded.

To see whats going on, consider what we have after carrying out the second step in the above algorithm:

$$\begin{aligned} \vec{R_1^*}\vec{y_i} &= \vec{R_1^*}\vec{R_1}x_{1i} + \vec{R_1^*}\vec{R_2}x_{2i} + \ldots + \vec{R_1^*}\vec{R_K}x_{Ki} + \vec{R_1^*}\vec{n} \\ &= |R_1|^2 x_{1i} + I \end{aligned} \quad (6)$$

where $\vec{R_i}$ is the i'th column vector in matrix $\mathbf{R}$ and I is collapsing all the other terms except the contribution from the i'th symbol of the first block. This computation is performed for all $i = 1, \ldots L$ symbols.

In Step 3, we collect these L symbols from the computation above and attempt to decode the first block. We can show that [31] the decoding will be successful only if the minimum distance for block 1 ($MD(1)$), is above a threshold $C * (I + N)$, where C is a constant dependent on the static code, while I and N are the interference and noise powers respectively. The minimum distance for block 1 after M transmissions and the decoding condition while treating all other blocks as interference can therefore be expressed as:

$$MD(1, M) = \sqrt{2}\sum_{i=1}^{M}|R_{i1}|^2 \geq C*\Big(\sum_{j=2}^{K}\Big|\sum_{i=1}^{M}R_{i1}^*R_{ij}\Big|^2 + \sum_{i=1}^{M}|R_{i1}|^2 n_i^2\Big) \quad (7)$$

The right half of the inequality thus consists of terms from the other blocks which are treated as interference and the noise.

There are two key takeaways from the equation above. First, as the receiver gets more packets (i.e. with increasing M), the minimum distance improves. Intuitively this makes sense, we expect our ability to decode to improve with every successive reception. Second, since the entries of $\mathbf{R}$ are picked randomly, any two columns in the matrix will be uncorrelated. The magnitude of the dot product of two uncorrelated complex vectors of equal magnitude will be less than the squared magnitude of either vector [31]. Hence, the right half of the inequality above grows relatively slower than the first block's minimum distance with M. Hence, with increasing M, the minimum distance for block 1 monotonically increases, until it exceeds the above threshold at which the static code can decode.

If block 1 is successfully decoded, we can subtract it and repeat the process for block 2. However, for this block, the interference will be only from blocks 3 to K, lesser than for the first block. Thus by stripping block 1 after decoding it, we reduce the minimum distance required for block 2 to decode, and as long as it is greater than the required threshold, the block will be decoded and the algorithm proceeds. All blocks will be decoded when the minimum distance for each is greater than the corresponding required threshold.

4.2 Encoding Algorithm

Our key deduction from the above analysis is that the *optimal design will have the property that all K blocks get decoded at once*. To see why, note that the receiver ACKs only when all K blocks are decoded, and that senders keep transmitting until an ACK is received. If at any point only a subset of the K blocks are decoded, then the next transmission will be wasted since it will contain components from the already decoded blocks. However this is a contradiction, since an optimal design by definition would not waste any transmission. Hence, the optimal design would guarantee that all K blocks get decoded at once.

[1]each block has a CRC at the end to check decoding success

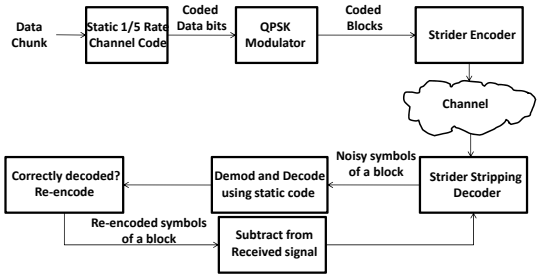


Figure 2: Strider's end-to-end design

The above insight has the following important consequence: *the minimum distance of all the K blocks should be greater than the required threshold for each block (defined in Eq. 8 below), when even any one of the blocks can be decoded*. This ensures, that if any block gets decoded, then all blocks get decoded. However the required minimum distance depends on the actual noise power in the channel, which the sender of course does not know. Hence the encoder just ensures the following condition: after every transmission it estimates the maximum possible noise power it can tolerate that still ensures that the minimum distance of each block is higher than the required threshold for each block. If the actual noise power is higher, then the sender will have to transmit more encoded packets. If the noise power is lesser than the maximum tolerable, then the receiver will decode and ACK.

To determine the entries of $\mathbf{R}$, we thus need to solve the following set of equations where the left hand side of the first equation represents the minimum distance $MD(b, m)$ that block b would need after m transmissions to decode,

$$\begin{aligned} &\sqrt{2}\sum_{i=1}^{m}|R_{ib}|^2 \geq C * \Big(\sum_{j=b+1}^{K}\Big|\sum_{i=1}^{m}R_{ib}^*R_{ij}\Big|^2 + \sum_{i=1}^{m}|R_{ib}|^2 n'^2\Big) \\ &\sum_{i=1}^{K}|R_{c,i}|^2 \leq P \quad \forall c = 1, \ldots, m \end{aligned} \quad (8)$$

and n' is an unknown noise variable representing the maximum tolerable noise at the m'th transmission. The second equation ensures that the total power of any transmission cannot exceed P. We have to solve the above for every value of $b = 1, \ldots, K$ and $m = 1, \ldots, M$.

The above set of equations constitute a non-linear optimization problem that can be solved numerically [3], we omit the specifics of the solution. However, we make two comments:

- First, the solver only provides the magnitudes of the entries in the matrix $\mathbf{R}$, while the phases of the complex entries are completely free. Strider picks these phases at random for each entry.
- Second, note that we have to compute $\mathbf{R}$ *only once*, and the computation is performed *offline*. After that $\mathbf{R}$ is essentially a codebook which all nodes know in advance. Hence the computation above is not on the critical path.

To summarize, each row in $\mathbf{R}$ provides the coefficients for creating a separate packet. So the above computation is run to create a sufficiently large matrix of size $P \times K$, such that we can create

upto P packets. In practice we will likely require much less than P transmissions and hence the receiver will only see a submatrix of $\mathbf{R}$. The sender picks the rows of $\mathbf{R}$ one after the other and uses them to linearly combine the K blocks to produce packets for transmission. The receiver decodes using the stripping decoder method described before.

4.2.1 Why is the above design rateless?

Strider started out with the premise that it converts a conventional static code that operates at a fixed SNR into a rateless one that operates at any SNR. Remember that to decode the static code, the minimum distance of each block needs to be above a threshold. As discussed earlier, the minimum distance depends on two factors. First, with increasing number of transmissions it monotonically increases. Second, with increasing noise strength (i.e. with a weaker channel) the required minimum distance increases. After the right number of transmissions, the minimum distance for each block exceeds the corresponding required threshold, ensuring that the block decodes. Thus the above design converts a fixed static code into a rateless one. Further, we show theoretically in Section 6 that the rateless conversion is efficient, i.e. if the static code achieves Shannon capacity at its decoding threshold SNR, Strider's rateless conversion asymptotically achieves Shannon capacity for Gaussian wireless channels across a wide SNR range.

5. DECODING COLLISIONS

Strider also transforms the static code to be collision-resilient, i.e. it enables us to decode all the component packets from collided signals. To see why, consider how the Strider decoding algorithm works even when there are no collisions. The stripping decoder initially attempts to decode the first block, while treating all other blocks which have been added to it as interference. If decoding is successful, it subtracts the first block and attempts to decode the second block while treating all other blocks as interference and so on. As we can see, Strider's stripping decoder is intrinsically treating every received packet as a set of collisions, where the collisions are between the blocks in a chunk. And the decoding works by decoding one block at a time, or in effect one component of the collision at a time. Hence intuitively, we can model a collision from two senders as a collision between blocks of both senders, and apply the same stripping decoder algorithm as above. We expand on this insight below.

Lets assume we have a scenario where two nodes Alice and Bob are hidden terminals and their transmissions collide at the AP. Since they do not receive ACKs after their first transmission, they re-encode using Strider's algorithm and transmit a new packet again, which will likely collide. Lets assume the AP gets M collisions, we can represent the i'th received symbols in the M collisions as:

$$\vec{y_i} = h_{Al}\mathbf{R}\vec{x_i^{Al}} + h_{Bob}\mathbf{R}\vec{x_i^{Bob}} + \vec{n} \tag{9}$$

where h_{Al} and h_{Bob} represent the channels, and x_i^{Al} and x_i^{Bob} represent the blocks for Alice and Bob respectively. For exposition simplicity we assume that the channel does not change through the M transmissions, but the results hold even if it does. We can rearrange the above equation into the following:

$$\vec{y_i} = \begin{bmatrix} h_{Al}\mathbf{R} & h_{Bob}\mathbf{R} \end{bmatrix} \begin{bmatrix} \vec{x_i^{Al}} \\ \vec{x_i^{Bob}} \end{bmatrix} + \vec{n} \tag{10}$$

In effect, the new set of equations is quite similar to Equation 5 for the single sender case discussed in the previous section, except for the fact that the size of the encoding matrix $\mathbf{R}$ as well as the data symbol vector has doubled. Hence, we can use the same stripping decoder method as above. Specifically, Strider uses the following algorithm:

1. Estimate h_{Al} and h_{Bob} from the packet preambles. We discuss how to estimate these quantities in detail in Sec. 5.0.2.
2. Calculate which node has the stronger channel, i.e. calculate $max(|h_{Al}|^2, |h_{Bob}|^2)$. Next use Strider's stripping decoder algorithm on Eq. 10 to try and decode the blocks of the node with the stronger channel.
3. If the previous step is successful, the signal after the contribution from the decoded blocks have been stripped will only consist of blocks from the weaker node's blocks. The resulting equation will be exactly like a single sender case with no collisions, hence we can use the standard stripping decoder algorithm to decode.

Intuitively, assuming Alice has the stronger channel, the stripping decoder is treating the packets from Bob as noise, and attempting to decode Alice's blocks. If successful, it subtracts the contributions of all of Alice's blocks and moves on to decode Bob's blocks. The steps above are reminiscent of successive interference cancellation (SIC) [10]. However, there is one critical difference. Unlike SIC, Strider does not need the colliding packets to be encoded at the right bitrate. Traditional SIC requires that the bitrates of the colliding packets be picked correctly so that the packet with higher power can be decoded while treating the other as interference [9]. However, if the nodes do not know that their packets are going to collide, they will not pick the correct bitrates required for SIC to work. Consequently SIC will fail to decode. Strider does not have this issue, since due to its rateless property it ensures that after an appropriate number of transmissions, the bitrate is sufficient to kickstart the decoding of the first block, which then starts a chain reaction for all the other blocks.

5.0.2 Practical Challenges in Decoding Collisions

1) Asynchrony: The above description assumed that nodes were synchronized across collisions i.e. transmitted packets collide exactly at the same offset across collisions. However, in practice due to random backoffs nodes will not be synchronized, and different collisions will begin at different offsets.

2) Collisions between different senders, or different chunks from the same senders: In practice, successive collisions could be between packets from different senders. Second, in Strider the packets from the node with the stronger channel will get decoded first, and the node will move on to transmit the next chunk of blocks. Hence successive collisions can be between different chunks from the same senders.

Strider is actually invariant to both of these problems because of its stripping decoder structure. Specifically, Strider attempts to decode each block separately, while treating everything else as interference. So lets say we are trying to decode the first block of Alice from the collisions. We collect all of the collisions, and can express the decoding problem for Alice's first block as follows:

$$\vec{y_i'} = \vec{R_1}x_{1i}^{Al} + \vec{R_2}x_{2i}^{Al} + \ldots + \vec{R_K}x_{Ki}^{Al} + \vec{I} \tag{11}$$

where the term $\vec{R_i}$ is the M length i'th column vector of $\mathbf{R}$, and $\vec{I}$ subsumes all the contributions from Bob's packets or from some other senders.

The above equation is collecting all the terms that have collided with the i'th symbol of Alice's first block across the M transmissions, and is just rearranging the terms in Eq. 10. To decode this block, we use the same stripping decoder technique. If successful, we re-encode it and subtract its contributions from all other symbols where it had a contribution. Thus, it does not matter what the identity of the terms in $\vec{I}$ is, since we do not use that knowledge in decoding Alice's blocks.

However, Strider does need to estimate the offsets where the collisions begin, so it knows where which symbol is. To do so we leverage

the preamble and postamble trick used in prior work [33, 16, 9]. We include a pseudorandom sequence in the preamble/postamble of each packet, and the receiver correlates the received samples against this known sequence. Since the pseudorandom sequence is uncorrelated with any other sequence except itself, the correlation will spike exactly when a packet starts, even if there is a collision. The location of the spike gives us the offset where the collision begins.

3) Compensating for Frequency Offsets: Different senders will have different carrier frequency offsets (CFO) w.r.t the receiver. When we decode a block and subtract, we have to compute and compensate for this frequency offset. Strider's current implementation is on top of a WiFi style OFDM PHY implemented with USRP2s and GNURadio. Hence, we use the standard Schmidl-Cox algorithm [25] for OFDM carrier synchronization and offset estimation. The algorithm is based on exploiting a repeating preamble by computing the cross correlation of a signal with a delayed version of itself, and computing the phase offset across the correlation values at different delays to compute the CFO. However, the algorithm needs to be modified for collisions, since we wont have a clean copy of the repeating preamble for the packet that starts second. Like prior work [33], we use the postamble to get a clean copy of the repeating preamble. The Schmidl-Cox algorithm is run on the postamble for the second packet. The algorithm also estimates the symbol timing and subcarrier spacing offsets apart from the CFO, which are then used in OFDM demod. We refer the reader to [25] for a detailed description of the standard Schmidl-Cox implementation.

4) Channel Estimation: The receiver needs to estimate the channels at the receiver for decoding the collisions. We use the pilot tones in the OFDM subcarriers (e.g. WiFi uses 4 pilot tones) to estimate the channel using the Least Squares algorithm [32]. Strider uses 4 pilot tones that are inserted in the packet header. Suppose $p_1, \ldots, p_4$ and $\vec{y} = y_1, \ldots, y_4$ are the 4 pilot symbols sent and received respectively. The LS channel estimate is given by:

$$\vec{h} = \mathbf{P}^{-1}\vec{y} \tag{12}$$

where $\mathbf{P}$ is diag$(p_1, \ldots, p_4)$, i.e. the 4×4 diagonal matrix of the 8 known pilot symbols. To estimate the channel at the other 48 data subcarriers we use linear interpolation at every subcarrier between two pilot subcarriers.

5) Collisions between more than two transmissions: Strider in principle can handle collisions between more than two packets. Specifically, Strider depends on the header of the packet being correctly decoded to handle collisions. Hence, similar to prior work [12], Strider appends the header to the end of the packet so that it can be recovered even under a collision. However if more than 2 packets collide, a receiver may not initially be able to decode all packet headers. But as decoding proceeds, one of the batches will get decoded after sufficient transmissions, and the decoded symbols are then subtracted from all collisions. After subtraction, a hidden header will be revealed at which point Strider can recover it and incorporate the new batch into the decoding process. We note however that in our experiments collisions between more than two nodes were quite rare, carrier sense works well enough that collisions happen only between hidden terminals, and configurations that involved three hidden terminals were very uncommon.

6. THEORETICAL ANALYSIS

Strider asymptotically achieves Shannon capacity for Gaussian channels. However, Strider's practical performance depends on how efficient the static code is at its decoding threshold. For example, a $1/2$ rate convolutional code with QPSK (used in the 12Mbps WiFi bitrate) has a decoding threshold of around 6dB [11] and achieves a rate of 1b/s/Hz at that threshold. But the Shannon capacity at that SNR is actually 2.3b/s/Hz. Hence convolutional codes are off from capacity, but we use them because they can be efficiently implemented.

Strider is orthogonal to the choice of the static code, and provides a technique for converting any static code into a rateless code that works at any SNR. Hence, what we wish to prove is that Strider's rateless conversion happens *without any loss in coding efficiency*, i.e., we would not achieve a higher rate than Strider by using a correctly picked conventional channel code and constellation at any SNR from the same class of static codes (e.g. convolutional codes in WiFi). Hence, we will assume that the rate $R(T)$ our static code achieves at its decoding threshold T is equal to the Shannon capacity at T, and intuitively show that after going through Strider's conversion it can achieve Shannon capacity across a larger SNR range.

When the sender uses Strider's algorithm, he is in effect dividing up the power among multiple blocks. Specifically, when he computes the entries of matrix $\mathbf{R}$, the magnitude of the column vectors corresponds to the powers that are allocated to the blocks. Lets assume we have K blocks and require M transmissions to decode. Hence, when the receiver manages to decode, the following condition is asymptotically true due to the way the matrix $\mathbf{R}$ is computed

$$\frac{P_1}{\sum_{i=2}^{K} P_i + N} = \ldots = \frac{P_j}{\sum_{i=j+1}^{K} P_i + N} = T \tag{13}$$

Here P_i is total power allocated to i'th block across the M transmissions, and N is the unknown noise power. Thus the total power used by the sender is $P = \sum_{i=1}^{K} P_i$ and the actual SNR of the channel is $10\log(P/N)$dB and is unknown to the sender. This equation is just restating the condition we developed in Eq. 8 that ensured that the minimum distance for each block is guaranteed to be greater than the required threshold for each block to decode.

The Strider decoder is a stripping decoder, i.e. it decodes the first block treating the second as noise, strips it after decoding and then decodes the second block. When the receiver decodes the K blocks, So the effective rate achieved by Strider at this point is:

$$\begin{aligned} R_{Strider} &= \sum_{i=1}^{K} R(T) = \sum_{i=1}^{K} \log(1+T) && (14)\\ &= \sum_{j=1}^{K} \log(1 + \frac{P_j}{\sum_{i=j+1}^{K} P_i + N}) && (15)\\ &= \log\left(\prod_{j=1}^{K} \frac{\sum_{i=j}^{K} P_j + N}{\sum_{i=j+1}^{K} P_i + N}\right) && (16)\\ &= \log(1 + \frac{\sum_{i=1}^{K} P_i}{N}) = \log(1+\frac{P}{N}) && (17) \end{aligned}$$

Thus the effective rate is the same as the Shannon capacity at power P and noise N. In other words, Strider achieves the same throughput as if the user had used the full power P to transmit with a capacity achieving code at the unknown SNR.

For our practical implemented algorithm, we use a convolutional code at a fixed rate as the static code. Hence the practical performance of our scheme will be dictated by how good the fixed static code is at its decoding threshold. However, we stress that *Strider is orthogonal to the choice of the static code*. Hence, if in the future efficient codes (e.g. LDPC codes [8]) that achieve capacity at their decoding SNR threshold become practically available in hardware, we can immediately use Strider to convert them into a rateless capacity achieving code that works at every SNR.

7. IMPLEMENTATION

Strider is designed to work on top of a WiFi-style OFDM PHY, with a 64 length FFT out of which 48 subcarriers are used for data, 4

for pilot tones and the rest are padding. In Strider the data stream is first divided into chunks of $K = 33$ parallel blocks of size 1500 bits each. Each block is passed through the static code encoder, which in our current implementation is a $1/5$ rate channel code based on convolutional codes and a QPSK constellation. Next, these K coded blocks are linearly combined to create a single packet. The symbols in this packet are striped across the 48 data OFDM bins, which are then passed through an IFFT to obtain the time-domain signal. At the receiver, the process is reversed.

Strider's current implementation builds on top of a 802.11 style OFDM PHY implementation in GNURadio from MIT [33]. However, our frontends are USRP2/RFX2400s whose interconnects cannot support the full 20MHz width required in Wifi, and are currently configured to use 6.25MHz (interpolation and decimation rates of 16) due to PC processing constraints. Hence the subcarrier width in our current implementation is 97.6KHz.

Static Code: Strider's current implementation uses a static code that consists of a fixed $1/5$ rate channel code. However, implementing a convolutional code with such a large constraint length is infeasible in practice. Strider adopts a standard communication theory trick, concatenate a $1/2$ and $1/3$ rate code to together create a $1/2 * 1/3 = 1/6$ rate code, and then puncture it to make a rate $1/5$ code. Both $1/2$ and $1/3$ rate codes are widely available and implemented in hardware. We refer the reader to [20] for a description of this technique.

Packet Header: Similar to traditional WiFi, the Strider header has a known preamble. After the preamble, the packet header includes the following packet parameters: Sender MAC address, destination MAC address, frame no, chunk no, the index of the row in $\mathbf{R}$ that is used to create the linear combination and packet length. The header is repeated at the end of the packet to protect it from collisions.

Complexity: The computational complexity of Strider is linear in the number of input data symbols. Compared to traditional WiFi, Strider employs a stripping decoder in addition to the decoder for the static code. Since we use convolutional style coding for the static code (the same as WiFi), the only extra complexity in Strider is from the initial stripping decoder component. The stripping decoder algorithm requires $K \times L$ complex multiplications for every packet received. If a block is decoded, it is subtracted from the received signal, which requires another L complex subtractions. Thus the two extra operations are both linear in the length of the data block. Strider's current implementation is bottlenecked by the decoding complexity of the static code, the extra overhead of Strider's stripping decoder is only around 20% in terms of wallclock time. However, the static codes we use are widely implemented in conventional wireless hardware for very high data rates, hence we believe Strider can be easily ported to a realtime hardware implementation.

8. EVALUATION

We evaluate Strider on an indoor testbed of 15 USRP2s and trace driven simulations. We compare Strider with the following:

- **Omniscient Scheme**: This scheme has perfect advance knowledge of the channel strength, and picks the maximum possible bitrate that can be decoded error free. The bitrate choices are from the 9 different bitrates available in the 802.11 standard, listed in Table 1. We augment the above rates with a 16-QAM, 2/3 code rate that achieves a rate of 2.66b/s/Hz to give the omniscient scheme more fidelity in picking the right bitrate. The omniscient scheme also guarantees that concurrent transmissions are scheduled in a collision-free manner.
- **SoftRate**: This is a state of the art rate rate adaptation protocol that uses soft information at the receiver to estimate the BER of a packet. The BER information is sent back to the sender via control packets, which uses it to make rate adaptation decisions. SoftRate's evaluation [33] shows that it outperforms almost all conventional rate adaptation techniques, so we compare against it as a representative of the best possible practical rate adaptation technique.

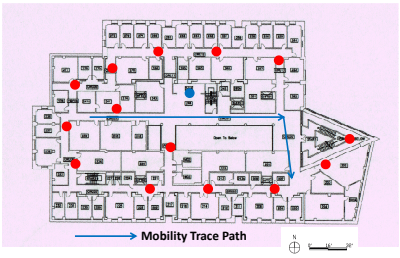


Figure 3: Strider Indoor Testbed Layout

Table 1: WiFi Bitrates

BitRate	Channel Code/Modulation	b/s/Hz
6	1/2, BPSK	0.5
9	3/4, BPSK	0.75
12	1/2, QPSK	1.0
18	3/4, QPSK	1.5
24	1/2, 16-QAM	2.0
32*	2/3, 16-QAM	2.66
36	3/4, 16-QAM	3.0
48	2/3, 64-QAM	4.0
54	3/4, 64-QAM	4.5

Before describing the experiments in detail, we briefly summarize our findings:

- In our testbed experiments, Strider achieves a throughput that is within 5% of the omniscient scheme across a wide range of SNRs (5-25dB). Note that Strider has no knowledge of the channel SNR, while the omniscient scheme has perfect advance knowledge.
- Strider eliminates hidden terminals in our testbed. Further, Strider achieves at least as good a throughput as the omniscient scheme which uses a collision free scheduler in most scenarios, and in the majority of the cases outperforms it.
- In comparison with SoftRate [33], a state of the art rate adaptation technique, we show that Strider outperforms by nearly 70% in mobile scenarios.
- In networks with contention and hidden terminals, Strider provides a throughput gain of 2.8× over SoftRate and 60% over the omniscient scheme.

9. INDOOR TESTBED EXPERIMENTS

In this set of experiments, we evaluate Strider using experiments in our indoor testbed of USRP2s. We compare with the omniscient scheme, since current USRP2s do not meet the timing requirements needed to implement dynamic rate adaptation techniques such as SoftRate. However note that the omniscient scheme is an upper bound on the performance of any conventional rate adaptation technique.

9.1 Strider's Rateless Conversion

Method: In this experiment, we randomly place two USRP2 nodes in our testbed and measure the SNR of the link. We then transmit 1000 packets between the two nodes. For omniscient scheme, we transmit using all the different bitrates, and pick the one which achieves the maximum throughput. For Strider, we use Strider's encoding and decoding algorithm. We repeat this experiment 10 times

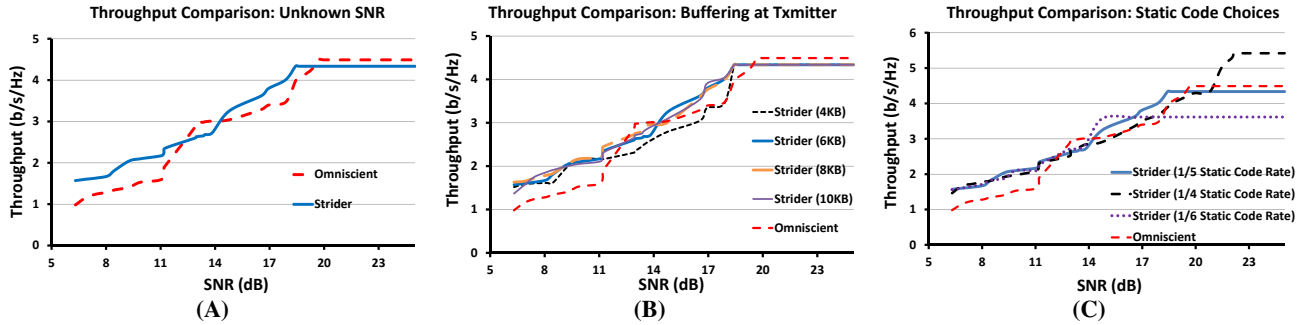


Figure 4: A) Strider performs almost as well as the omniscient scheme at all SNRs. B) Strider works fairly well even when the transmitter has a small amount of data to transmit. C) Strider's performance at high SNRs can be improved by selecting higher rate static codes.

for the same location of the nodes and take the average throughput for either scheme, expressed in terms of bits/second/Hz. We then change the locations of the two nodes to get a different SNR and repeat the above procedure. We plot the average throughput achieved by the two schemes vs SNR in Fig. 4.

Analysis: As Fig. 4 shows, Strider achieves a throughput that is atleast within 5% of the omniscient scheme at all SNRs between $4-24$dB. We comment on two regions of the graph. First, at medium to low SNRs ($4-16$dB), Strider often outperforms the omniscient scheme. The reason is that Strider has more granular steps, in fact it can achieve $K=33$ different effective bitrates. The omniscient scheme is choosing within a relatively smaller set, the 10 different channel coding and modulation choices listed in Table 1. Hence at certain SNRs, the omniscient scheme is limited by the choices it has. However, note that Strider is close to the omniscient scheme at every SNR, implying that even if the omniscient scheme had more choices, it could not have done better than Strider.

On the other hand, Strider is around 5% worse at high SNRs greater than 18dB. The same granularity that helped Strider at the medium and lower SNRs slightly hurts Strider in the high SNR region. Remember that Strider's effective throughput drops as $2*33/5M$ where M is the number of transmissions needed. Hence, when M is small (around $2-5$), then the effective bitrate exhibits jumps. In high SNR regions, Strider decodes using $2-5$ transmissions, and does not have the high fidelity to achieve close to the omniscient in this region. However, Strider is still only 5% off omniscient.

How sensitive is Strider to buffering?: Strider buffers data so that it has enough to form a batch of blocks that it can code over. However, in practice some applications might not generate enough traffic to fill the buffer, and hence Strider might need to work with smaller buffers. Strider can handle these by changing two parameters: the size of a block, as well as the number of blocks in a batch. We conduct an experiment where the sender has different amounts of buffered data available, and picks the best block and batch size for that buffer size. We plot the average throughput vs SNR for buffer size in Fig. 4(b).

Analysis: Fig. 4(b) shows that Strider works fairly well even when the buffer size is as small as 4KB. There is a slight underperformance at medium SNRs (12-16dB). The reason is that at small buffer sizes, Strider has to use a smaller batch size than the normal value of 33. The smaller batch size impacts the granularity of the effective bitrates Strider can achieve, and leads to slightly lower effective throughputs. But overall, Strider works fairly well even when there is only a small amount of data (4KB) to transmit. In the extreme case where the amount of data queued up at the transmitter is smaller than 4KB, Strider might be overkill. In such cases, the sender can simply use a fixed low rate code to transmit the packet, and switch to Strider only when the outstanding buffer is greater than 4KB.

On the other hand larger buffers (i.e. larger batch sizes) slightly improve performance, especially at high SNRs. However, larger batch sizes come with the obvious tradeoff of needing more buffering at the transmitter. We chose $K=33$ as the default since it gives good performance across our target SNR range, however the designer is free to choose a higher batch size if he wishes to target higher SNRs.

Impact of Static Code Choice: Strider's parameters, the 1/5 static code rate and the QPSK modulation, were picked to obtain the best performance in our target SNR range of $3-25$dB that is commonly found in deployed wireless networks. In the following experiment we vary the static code rate to check if Strider is sensitive to that choice. We note that varying the modulation is not necessary, since as we discussed in Sec. 6 what really matters for Strider's performance is the rate at which information is encoded in a block, because that parameter dictates the minimum distance required to decode a block. Changing the static code while keeping the QPSK modulation is sufficient to control the encoding rate of a block. We plot the average throughput vs SNR for different static code choices in Fig. 4(c).

Analysis: Fig. 4(c) plots the relative performance of different static code choices in Strider. As we can see, at most SNRs the different static code rates among the convolutional family do not make a big difference. The differences again are at high SNRs, and is mostly due to the granularity of the effective bitrates achieved for different static code rates. Higher static code rates (e.g. 1/4 code rate) in fact perform better at higher SNRs, achieving nearly 5.5b/s/Hz at SNRs$>22dB$. Thus changing the static code rate provides the designer another lever if he wishes to optimize Strider for higher SNRs, outside our current target range of $3-25$dB.

9.2 Strider's Collision Decoding

To evaluate Strider with collisions, we set up hidden terminal scenarios in our testbed using USRP2 nodes. To evaluate if a particular node configuration is a hidden terminal scenario, we implement a simple threshold based carrier sense on the USRP2 nodes and check if they can carrier sense each other. The two hidden terminal nodes transmit to a fixed third USRP2 node, which acts as the receiver.

Method: We compare against the omniscient collision-free scheme where the two senders take turns transmitting 1000 packets to the receiver, and use the maximum error free bitrate for their channels during their transmissions. For Strider, the two senders transmit concurrently and the collided packets are decoded at the receiver using the Strider collision decoding algorithm. We compute the average throughput achieved by the omniscient scheme and Strider over 10 consecutive runs. We plot the CDF in Fig. 5(A).

Analysis: Fig. 5(A) shows that Strider surprisingly outperforms the omniscient collision free scheduler in most of the scenarios! The median throughput gain over the collision-free scheduler is nearly 30%. The reason is that in a hidden terminal scenario, if one node has a

stronger channel than the other, then collision-free scheduling is actually suboptimal. The collision-free scheduler allows both nodes to transmit an equal number of packets, however the node with the weaker channel will take longer to transmit the same number of packets. Consequently, even though the node with the stronger channel can achieve higher rates when he is given the chance to transmit, he is limited due to the weaker channel node. Hence overall network throughput drops.

Strider on the other hand lets both nodes transmit concurrently and decodes from collisions. When the two senders have equal channels, it achieves the same throughput as the omniscient scheme. When the channels are unequal, the node with the stronger channel gets his packets decoded first, and moves on to the next chunk. Hence, unlike the collision-free omniscient scheme it does not have to wait for the weaker node to finish. Consequently, the medium is better utilized and leads to higher overall network throughput.

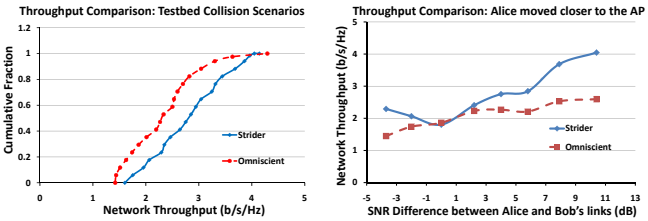
Impact of Relative SNRs: To better understand the above phenomenon, we conduct the following controlled experiment. We focus on a specific hidden terminal scenario where the SNRs of either sender to the receiver (when they are transmitting separately) is the same at around 10dB. We then keep one sender (lets say Bob) fixed and move the other sender (lets say Alice) closer to the receiver. For each location, we measure the average throughput achieved by Strider and the omniscient collision-free scheme as described in the previous experiment. We plot the relative throughput (i.e Strider throughput normalized by omniscient throughput) vs relative SNR (SNR of Alice - SNR of Bob) for both schemes in Fig. 5(B).

As Fig. 5(B) shows, the throughput of the collision free scheduler is slightly better ($\approx 5\%$) than Strider when the relative SNR is close to zero. The reason is that for Strider's decoding algorithm to get kickstarted, it needs to be able to decode the first block. But when the relative SNR is close to zero, Strider can take a long time before the first block can get decoded since collisions from the second node are treated as noise. However, as Alice moves closer to the receiver and her channel improves, Strider's throughput increases relative to the collision-free scheduler. The reason is that Alice's packets are decoded faster, while Bob achieves a throughput that is commensurate with his channel. In the collision-free omniscient scheme, even though Alice's channel has improved, she cannot take full advantage of it because Bob monopolizes the channel time to transmit his packets. When the SNR gap is nearly 10dB, the overall throughput is nearly 50% better than the collision-free scheduler.

10. TRACE DRIVEN EMULATION

Although Strider can run in real time on a USRP2 connected node, similar to prior work [33, 11] we turn to trace driven emulation to compare Strider with SoftRate, a state of the art conventional rate adaptation technique. This is for two reasons. First, SoftRate requires estimated BER control feedback immediately after every transmission from the receiver to the sender, but the USRP2s are not equipped to quickly transmit ACKs after a packet is received. Second, we want to compare the schemes over varied channel conditions, from static to rapidly changing, from no contention to heavy contention, to assess how consistently they perform across all scenarios. However, it is hard to generate controllable high-mobility and high-contention in experimental settings.

Trace: We collect real channel information for the simulations via two traces: one for mobility and the other for contention. We use the Stanford RUSK channel sounder [22] to collect channel state information for a 20MHz 802.11 wireless channel. The channel sounder is an equipment designed for high precision channel measurement, and provides almost continuous channel state information over the entire



(A) (B)

Figure 5: A) Strider eliminates hidden terminals. B) Strider's overall throughput improves as Alice is moved closer to the receiver.

measurement period, and can measure channel SNRs as low as -3dB. Our experiments are conducted at night on the band between 2.426 and 2.448GHz which corresponds to WiFi channel 6, and include some interference from the building's WiFi infrastructure which operates on the same channel.

- **Mobility Trace**: A mobile channel sounder node is moved at normal walking speed (≈3mph) in the testbed and the channel sounder node at the center (the blue node at the center of the testbed figure 3) measures the channel from the mobile node. These nodes record and estimate detailed channel state information for all frequencies in the 20Mhz channel, and therefore include frequency selective fading which we would not have seen with USRP2s that operate on 6.25Mhz bands. We collect around 100000 measurements over a 100 second period, and get a CSI sample every $1ms$ for one trace. We use 10 different walking path to collect 10 different mobility traces.
- **Contention Trace**: The channel sounder is placed at ten different locations in our testbed, and their channel to the central blue node is measured over a period of 100 seconds similar to the mobility traces above. We therefore collect 10 such traces. We also place two USRP2 nodes at all pairs of these 10 locations and use our hidden terminal technique described in Sec. 9.2 to determine if the two nodes are hidden terminals. We record this information along with the trace.

Emulator: We feed this trace to a custom emulator written using the the MIT Gnuradio OFDM Code [33] and Strider's implementation. For Softrate, the emulator implements a 802.11 style PHY augmented with soft decoding since SoftRate uses it for BER estimation at the receiver. Further, for both Strider and Softrate, the emulator implements a 802.11 style MAC with ACKs, CSMA and exponential backoff with the default parameters.

Simulating Mobility: To vary mobility, we replay the trace at different speeds. For example, $4\times$ mobility implies the channel measurements that spanned T seconds now span $T/4$ seconds. When a packet is transmitted at time t in simulation, the symbols in the packet are distorted using the corresponding channel measurement from the trace at time t. If the trace has been sped up $4\times$ to simulate mobility, the channel measurement at time t in the new trace will be the channel measurement in the original trace at time $4t$.

Simulating Contention: To vary contention, we pick different subsets of the 10 nodes from the contention trace and let them send packets whenever the simulated 802.11 MAC lets them. If a node is allowed to transmit at time t, then we look up the channel measurements from its trace at time t and distort its transmitted symbols accordingly. If two nodes concurrently transmit and collide, their symbols are individually distorted according to their respective channel traces, and the distorted symbols are added up at the receiver.

10.0.1 Performance with Mobility

We compare the performance of Strider under varying mobility

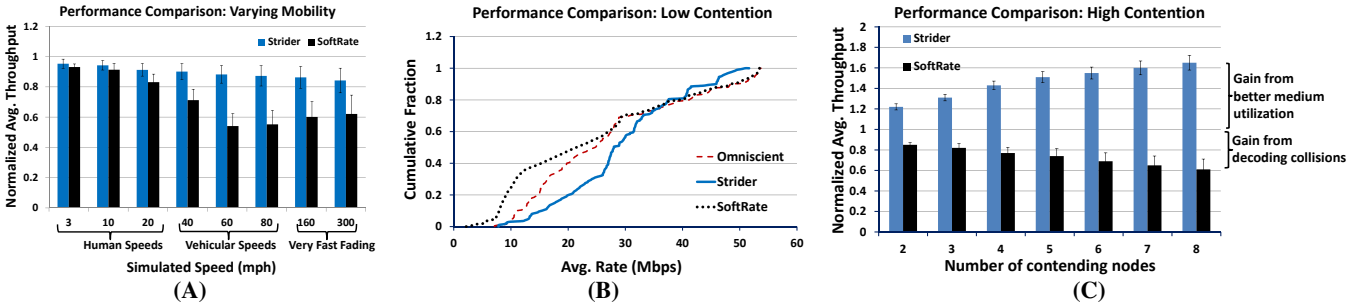


Figure 6: A) Strider outperforms SoftRate with increasing mobility. B) Strider provides gains because of better medium utilization at low contention. C) Strider outperforms both omniscient and SoftRate due to better medium utilization and ability to decode collisions in high contention scenarios.

by playing the trace at increasing speeds, from $1\times$ walking speed (3mph) to $20\times$ corresponding to vehicular speeds (60-80mph) to $100\times$ corresponding to 300mph. Note that the omniscient scheme has advance knowledge of all the channel states that affect each packet transmission, and picks the highest bitrate that can be correctly decoded at every instant. The other schemes are implemented as described before. The performance metric is the average throughput achieved by each scheme over a trace. We run the simulation for each trace and for each compared approach and for each speed. We compute the normalized throughput (i.e. throughput divided by throughput of omniscient scheme) achieved by each approach for all traces and for each speed, and then calculate the average normalized throughput at each speed. Fig. 6(A) plots the average normalized throughputs for the two schemes vs simulated speeds, along with error bars.

Analysis: Strider performs excellently, though it does exhibit some dropoff with increasing mobility compared to the omniscient scheme. At high mobility, Strider is around 15% off the omniscient scheme's rate. However, Strider still outperforms SoftRate by nearly 70% in vehicular mobility scenarios, and by 50% in very fast fading scenarios.

All schemes do fairly well at low mobility, which is expected. At human speeds, we do not see large fluctuations, the channel coherence time in our trace is around 100ms, a relatively long time given that a 802.11 sender would manage to transmit around 50 packets at the lowest bitrate, and probably more. Hence, once a rate adaptation algorithm locks on to the correct bitrate (which SoftRate achieves within one packet transmission time), bitrate adaptations are relatively infrequent. Therefore SoftRate performance is also close to the omniscient scheme.

SoftRate exhibits interesting behavior at higher mobilities. First, as mobility increases, channel coherence times drop, and bitrate adaptation decisions have to be made more frequently. Hence, the likelihood of a packet being transmitted at the incorrect bitrate increases, leading to the loss in performance. However, the surprising fact is that SoftRate performance drops and then recovers at very high mobility, corresponding to very fast fading scenarios. We *speculate* that the reason is the timescale at which Softrate adapts rate, which is every packet. Hence if a wireless channel has a coherence time which is on the order of $1-2$ packet transmission times, SoftRate is likely to make a mistake in the bitrate decision every second or third transmission. The coherence time in the vehicular mobility scenarios is on the order of a few milliseconds in our trace, just sufficient to transmit $1-4$ packets. Consequently SoftRate is constantly playing catchup, and often makes wrong bitrate decisions, leading to lower normalized average rate. However, as mobility increases further, the channel changes several times within a packet. Here SoftRate manages to average the channel over the packet transmission and accurately estimate BER. If the average around which the channel fluctuates stays the same across packets, then SoftRate finds the correct bitrate decision, and performance improves. This behavior is consistent with the findings of the SoftRate paper [33]. The SoftRate authors present evaluations over slow and very fast fading scenarios, but mention that in intermediate mobility where the channel changes every $2-3$ packets, their scheme suffers. [2]

10.0.2 Low Contention

Method: We compare Strider's performance with the omniscient scheme and SoftRate under low contention scenarios.In these experiments, we randomly pick two nodes from the contention traces, and simulate a 802.11 network with both of them communicating to an AP. We let both nodes transmit, with the 802.11 MAC scheduling access for Strider and SoftRate and run the simulation for 100 seconds. If these two nodes are hidden terminals according to our testbed measurements, then they cannot carrier sense each other in the simulation. Among the C_2^{10} pairs in our traces, only 12 pairs are hidden terminals. The omniscient scheme however uses a collision-free scheduler, and concurrent transmissions will be scheduled one after the other to eliminate collisions. We compute the average total throughput for each two node scenario, and then repeat for a different two node scenario. Fig. 6(B) plots the CDF of the throughputs for the three compared schemes.

Analysis: Surprisingly, Strider outperforms even the omniscient scheme in the low contention scenario. The median rate gain over omniscient is around 25% and around 35% over SoftRate. With SoftRate approximately 15% of the simulations perform quite badly (shown in the first quartile of the CDF) because these topologies correspond to the hidden terminals in our set. However, given that hidden terminals are relatively rare and the omniscient scheme uses a collision free scheduler, where do Strider's gains come from?

The key reason for Strider's performance is *better medium utilization*. Consider what happens when the 802.11 MAC schedules contending nodes for transmission. If the nodes are within carrier sense range, the MAC ensures that all nodes get equal number of opportunities to transmit a packet. However, if the contending nodes have differing channel qualities to the AP, then the node with the weaker channel monopolizes the channel time because the same sized packet requires a larger transmission time (since it has to use a lower bitrate). Hence the stronger node is unfairly penalized, which hurts overall network performance. As prior work has observed [30], the right MAC policy in such scenarios is to ensure "time based fairness", i.e. give all nodes equal amount of channel time regardless of their respective channel strengths.

In Strider, due to its rateless nature, all transmissions occupy the same amount of channel time. Since the 802.11 MAC ensures equal number of transmission opportunities for all contending nodes, Strider ensures that every node gets equal time on the channel. Thus the stronger node is not unfairly penalized. This leads to better medium utilization and consequently higher overall throughputs. Thus, un-

[2] please see Sec. 3.4 of the SoftRate [33] paper for a discussion.

like prior work which used mechanisms such as regulating per node queues to ensure time-based fairness, Strider achieves it for free.

10.0.3 High Contention

Method: The experiment is conducted similar to the low contention scenario, except we pick increasing numbers of contending nodes from our contention traces. We normalize the throughput of each experiment by dividing it with the throughput achieved by the omniscient scheme. We calculate the average normalized throughput across all experiments that have the same number of contending nodes and plot it in Fig. 6(C) with increasing contention.

Analysis: Strider significantly outperforms both the omniscient scheme and SoftRate as contention increases. When 8 nodes are contending, Strider is nearly 60% better than the omniscient scheme, and $2.8\times$ better than SoftRate. There are three reasons for this outperformance:

- First, Strider does better because of efficient medium utilization, and the gain increases with higher contention. The reason is that with more nodes, the problem of weaker channel nodes dominating channel time becomes even more acute. Strider's rateless property ensures that every node gets an equal amount of channel time regardless of its channel quality, and hence performs better.
- Second, in high contention collisions are more frequent. SoftRate's performance suffers since it cannot decode from collisions and has to resort to expensive backoffs to ensure nodes don't collide. Strider decodes from collisions and also eliminates all hidden terminal problems. We note that the collisions we observed in our simulations were typically between two transmissions, it was quite rare to see collisions between higher number of transmissions because the required node geometry (three pairwise hidden terminals) as well as synchronization in channel access inspite of three independent random backoffs are quite unlikely.
- Finally, SoftRate relies on feedback from a previous packet's ACK to pick the bitrate for the next transmission. However, in high contention scenarios, a node may not get to transmit a packet for several milliseconds, and its bitrate estimate from the previous measurement gets stale. This leads to inaccurate bitrates and a loss in throughput.

11. CONCLUSION

Strider provides a rateless and collision-resilient design, that consistently achieves very good performance across a wide variety of scenarios, ranging from low mobility to high mobility, from low contention to high contention and unknown channels to hidden terminals. We believe Strider can greatly simply wireless PHY design by eliminating the need for complicated rate adaptation protocols. Strider suggests a number of avenues for future work, including redesigning the MAC to take advantage of Strider's collision resilient code and extending it to 802.11n MIMO scenarios.

12. REFERENCES

[1] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang. MACAW: Media access protocol for wireless lans. In *Proceedings of the international conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*, 1994.

[2] J. Bicket. Bit-rate selection in wireless networks. *MS Thesis, Massachusetts Institute of Technology*, 2005.

[3] G. Caire, S. Guemghar, A. Roumy, and S. VerdÃž. Maximizing the spectral efficiency of coded cdma under successive decoding. *IEEE Transactions on Information Theory*, Jan 2004.

[4] J. Camp and E. Knightly. Modulation rate adaptation in urban and vehicular environments:cross-layer implementation and experimental evaluation. In *ACM MOBICOM*, 2008.

[5] U. Erez, M. Trott, and G. Wornell. Rateless coding and perfect rate-compatible codes for gaussian channels. In *Information Theory, 2006 IEEE International Symposium on*, pages 528 –532, july 2006.

[6] Free Software Foundation. Gnuradio. `http://gnuradio.org`.

[7] P. Frenger, S. Parkvall, and E. Dahlman. Performance comparison of harq with chase combining and incremental redundancy for hsdpa. In *IEEE VTC*, 2001.

[8] R. Gallagher. Low density parity check codes. In *PhD thesis, MIT*, 1962.

[9] S. Gollakota and D. Katabi. ZigZag decoding: combating hidden terminals in wireless networks. In *SIGCOMM '08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pages 159–170, New York, NY, USA, 2008. ACM.

[10] D. Halperin, T. Anderson, and D. Wetherall. Taking the sting out of carrier sense: interference cancellation for wireless lans. In *MobiCom '08: Proceedings of the 14th ACM international conference on Mobile computing and networking*, pages 339–350, New York, NY, USA, 2008. ACM.

[11] D. Halperin, A. Sheth, W. Hu, and D. Wetherall. Predictable 802.11 packet delivery from wireless channel measurements. In *ACM SIGCOMM*, 2010.

[12] K. Jamieson and H. Balakrishnan. Ppr: Partial packet recovery for wireless networks. In *ACM SIGCOMM*, 2007.

[13] G. Judd, X. Wang, and P. Steenkiste. Efficient channel-aware rate adaptation in dynamic environments. In *ACM MOBISYS*, 2008.

[14] T. Kailath, H. Vikalo, and B. Hassibi. Mimo receive algorithms. *Space-Time Wireless Systems: From Array Processing to MIMO Communications*, 2005.

[15] A. Kamerman and L. Monteban. Wavelan r-ii: A high-performance wireless lan for the unlicensed band. *Bell Labs Technical Journal*, 2, 1997.

[16] S. Katti, S. Gollakota, and D. Katabi. Embracing wireless interference: analog network coding. In *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 397–408, New York, NY, USA, 2007. ACM.

[17] L. E. Li, K. Tan, Y. Xu, H. Viswanathan, and Y. R. Yang. Remap decoding: Simple retransmission permutation can resolve overlapping channel collisions. In *ACM MOBICOM*, Sep 2010.

[18] S. Lin and P. Yu. A hybrid arq scheme with parity retransmission for error control of satellite channels. *IEEE Trans. on Communications*, 1982.

[19] M. Luby. Lt codes. In *Proc. of FOCS 2002*, 2002.

[20] D. Mackay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003.

[21] MadWiFi. Onoe rate control. `http://madwifi.org/browser/trunk/ath_rate/onoe`.

[22] G. V. L. J. N. Czink, B. Bandemer and A. Paulraj. Stanford july 2008 radio channel measurement campaign. In *COST 2100*, October 2008.

[23] R. Palanki and J. Yedidia. Rateless codes on noisy channels. In *ISIT*, 2004.

[24] A. Sarwate and M. Gastpar. Rateless codes for avc models. *Information Theory, IEEE Transactions on*, 56(7):3105 –3114, july 2010.

[25] T. Schmidl and D. Cox. Robust frequency and timing synchronization for ofdm. *IEEE Transactions on Communications*, Dec. 1997.

[26] S. Sen, R. R. Choudhury, and S. Nelakuditi. Csma/cn: Carrier sense multiple access with collision notification. In *Mobicom*, 2010.

[27] S. Sen, N. Santhapuri, R. R. Choudhury, and S. Nelakuditi. Accurate: Constellation based rate estimation in wireless networks. In *NSDI*, 2010.

[28] A. Shokrollahi. Raptor codes. *IEEE/ACM Trans. Netw.*, 14(SI):2551–2567, 2006.

[29] E. Soljanin, R. Liu, and P. Spasojevic. Hybrid arq in wireless networks. In *DIMACS Workshop on Networking*, 2003.

[30] G. Tan and J. Guttag. Time-based fairness improves performance in multi-rate wlans. In *Usenix Annual Technical Conference*, 2004.

[31] D. Tse and P. Vishwanath. *Fundamentals of Wireless Communications*. Cambridge University Press, 2005.

[32] J. Van de Beek, O. Edfors, M. Sandell, S. Wilson, and P. Borjesson. On channel estimation in ofdm systems. 1995.

[33] M. Vutukuru, H. Balakrishnan, and K. Jamieson. Cross-layer wireless bit rate adaptation. In *ACM SIGCOMM, Barcelona, Spain*, August 2009.

[34] D. Warrier and U. Madhow. On the capacity of cellular cdma with successive decoding and controlled power disparities. In *Proc. 48th IEEE Vehicular Technology Conf.*, 1998.

[35] S. H. Y. Wong, H. Yang, S. Lu, and V. Bharghavan. Robust rate adaptation for 802.11 wireless networks. In *Proceedings of the 12th annual international conference on Mobile computing and networking*, New York, NY, USA, 2006.