August 2006
UILU-ENG-06-2217
DC-225

Exploiting Wireless Broadcast By Opportunistic Packet Splaying

Vivek Raghunathan, Min Cao, and P. R. Kumar

Coordinated Science Laboratory
1308 West Main Street, Urbana, IL 61801
University of Illinois at Urbana-Champaign

REPORT DOCUMENTATION PAGE	*Form Approved OMB NO. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August 23, 2006	

4. TITLE AND SUBTITLE
Exploiting wireless broadcast by opportunistic packet splaying

6. AUTHOR(S)
Vivek Raghunathan, Min Cao, and P. R. Kumar

5. FUNDING NUMBERS
DARPA Contract N66001-06-C-2021
AFOSR under Contract No. F49620-02-1-0217,
NSF under Contract No. NSF CNS 05-19535,
NSF under Contract No. ANI 02-21357,
NSF under Contract No. CCR-0325716,
USARO under Contract No. DAAD19-01010-465,
DARPA/AFOSR under Contract No. F49620-02-1-0325,
DARPA under Contact Nos. N00014-0-1-1-0576.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1308 West Main Street
Urbana, Illinois 61801-2307

8. PERFORMING RGANIZATION REPORT NUMBER
UILU-ENG-06-2217
DC-225

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFOSR, 875 N. Randolph St., Arlington VA 22203
DARPA, 3701 N. Fairfax Dr. Arlington, VA 22203
NSF, 4201 Wilson Blvd, Arlington, VA 22203
USARO, 2800 Powder Mill Rd. Adelphi MD 20783-1197

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official position, policy, or decision, unless so designated by other documentation

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The IEEE 802.11 random access MAC suffers from expensive channel acquisition costs which are exacerbated by the use of TCP. *This floor acquisition mechanism is primarily a way to fashion unicast ``links'' out of what is fundamentally a broadcast medium*. The main thesis of this paper is that such unicast usage of the wireless channel is unnecessary, and that once the floor has been acquired, we can instead ``splay'' packets to as many potential receivers as possible using the fact that wireless is a broadcast medium. Splay is placed between the IP and MAC layers and reduces the expensive cost of channel acquisition by opportunistically combining packets intended for different receivers. Splay allows for the use of sophisticated coding approaches to augment the IEEE 802.11 stop-and-go ARQ. This also helps prevent IEEE 802.11 from incorrectly backing off exponentially in response to fading losses on intermediate quality links, which are quite common in practice. We are currently implementing Splay as a Linux kernel module.

14. SUBJECT TERMS
IEEE 802.11, floor acquisition, packet combining, wireless broadcast, TCP

15. NUMBER OF PAGES
20

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

Exploiting wireless broadcast by opportunistic packet splaying

Vivek Raghunathan, Min Cao, and P. R. Kumar

Abstract

The IEEE 802.11 random access MAC suffers from expensive channel acquisition costs which are exacerbated by the use of TCP. *This floor acquisition mechanism is primarily a way to fashion unicast "links" out of what is fundamentally a broadcast medium*. The main thesis of this paper is that such unicast usage of the wireless channel is unnecessary, and that once the floor has been acquired, we can instead "splay" packets to as many potential receivers as possible using the fact that wireless is a broadcast medium. Splay is placed between the IP and MAC layers and reduces the expensive cost of channel acquisition by opportunistically combining packets intended for different receivers. Splay allows for the use of sophisticated coding approaches to augment the IEEE 802.11 stop-and-go ARQ. This also helps prevent IEEE 802.11 from incorrectly backing off exponentially in response to fading losses on intermediate quality links, which are quite common in practice. We are currently implementing Splay as a Linux kernel module.

Index Terms

IEEE 802.11, floor acquisition, packet combining, wireless broadcast, TCP.

I. Introduction

The IEEE 802.11 random access MAC uses four-way handshaking mechanism to mediate access to the wireless channel. Every node maintains a contention window (CW) to estimate the channel interference level. When a node has a packet to send, it waits for a random backoff interval in $[0, \mathrm{CW}]$ slottimes and then attempts floor acquisition using a RTS frame to silence all nodes in the transmitter's neighborhood. If the intended receiver successfully decodes the RTS, it responds with a CTS silencing all nodes in the receiver's neighborhood. On receiving the CTS, the sender proceeds with a DATA frame containing the unicast payload packet, and the receiver replies with an ACK to complete the handshake. If the handshake fails, it is assumed that the loss was due to interference and the sender retries, with CW doubled to resolve contention for the channel.

Vivek Raghunathan, Min Cao and P. R. Kumar are with the Dept. of Electrical and Computer Engineering, and the Coordinated Science Laboratory, University of Illinois, Urbana-Champaign. Email: vivek, mincao, prkumar@control.csl.uiuc.edu

This material is based upon work partially supported by DARPA Contract N66001-06-C-2021, AFOSR under Contract No. F49620-02-1-0217, NSF under Contract Nos. NSF CNS 05-19535, ANI 02-21357 and CCR-0325716, USARO under Contract No. DAAD19-01010-465, DARPA/AFOSR under Contract No. F49620-02-1-0325, DARPA under Contact Nos. N00014-0-1-1-0576. Vivek Raghunathan is also supported by a Vodafone Graduate Fellowship.

The four-way handshake originated as a technique to fashion a reliable unicast link from what is essentially a broadcast channel, albeit one that is very expensive in terms of floor acquisition overhead. This is specially true at the higher data rates in IEEE 802.11b/g because of the high relative cost of PLCP and control frame headers that are sent out at the lowest data rates, and on intermediate quality links, where the exponential backoff mechanism's proclivity to mistake fading for interference is quite costly. These inefficiencies in fashioning unicast "links" from a broadcast medium are further exacerbated with asymmetric TCP connections, where small reverse direction TCP ACKs compete with forward direction TCP segments *of the same flow*.

Our fundamental argument is that the primary role of four-way handshaking is not in the construction of unicast links. Four-way handshaking often acquires the floor to other neighbors as a by-product. It can thus be merely thought of as a mechanism to acquire the floor to different receivers with different probabilities. *Thus, a more advantageous exploitation of four-way handshaking is in providing a probabilistic multi-receiver floor acquisition mechanism that simultaneously acquires the wireless floor to different receivers in as reliable a fashion as possible.*

Once four-way handshaking has acquired the floor, packets to different receivers can be simultaneously "splayed" over the wireless channel, thus taking advantage of the broadcast nature of wireless communication whenever possible. We present the design of such a packet splaying protocol called Splay. Splay is layered between IP and the MAC, and provides a tunneling mechanism to *opportunistically combine packets for different intended receivers*. Such jumbo Splay packets are sent to a particular receiver (called the "primary") by triggering the four-way handshaking mechanism. The Splay tunnel endpoint at each intended receiver processes the received packet burst to recover individual data packets. Splay only silences the neighborhood of the "primary receiver" using the four-way handshaking mechanism. The wireless channels to other intended receivers may still be lossy, although a good choice of "secondary receivers" will make the probability of this event small. Splay thereby allows for the use of erasure codes to improve the robustness of packet delivery to these "secondary" receivers. Since it incorporates built-in forward error correction, Splay can be configured to completely bypass link layer retransmissions and thus, provides performance benefits when used over lossy intermediate quality links where the proclivity of IEEE 802.11 to mistake fading for interference results in expensive random backoffs.

The Splay protocol can be completely implemented in software without requiring any changes to the IEEE 802.11 MAC. Its biggest advantage lies in the simple and rich flexibility it provides for the exploration of multi-receiver floor acquisition policies. The Splay packet combining policy at the sender can be configured from an `ioctl`-based user-space interface by specifying a list of $(matchrule, reward)$ two-tuples. A packet combination is assigned a score equal to the sum of $rewards$ for each $matchrule$ Boolean condition that it satisfies. Splay attempts to choose a packet combination that maximizes this score among all sets of packet combinations. It can be shown that the general problem of opportunistically combining packets in an optimal manner is computationally intractable. Instead, Splay uses a greedy online algorithm to combine packets according to the list of $(matchrule, reward)$ two tuples.

We are currently implementing Splay as a Linux kernel module. The rest of this paper is organized as follows: in Section II, we describe the expensive nature of IEEE 802.11's floor acquisition mechanism, specially when used with

TABLE I

FOUR WAY HANDSHAKE COSTS

Data Rate (Mbps)	Control overhead/Data transmit time	
	Data = 100 byte	Data = 1500 byte
1	127.5 %	8.5 %
2	255 %	17 %
5.5	701.25 %	46.75 %
11	1402.5 %	93.5 %

TCP. We describe Splay and its implementation design in Sections III, IV, V and VI. Finally, we discuss interactions with TCP, related work and conclude in Sections VII, VIII and IX respectively.

II. MOTIVATION

IEEE 802.11 uses a four-way RTS-CTS-DATA-ACK handshake to do floor acquisition and contention resolution. This mechanism results in two sources of heavy overhead:

1. The IEEE 802.11 specification [1] requires all control frames to be sent at one of the basic rates (1 Mbps in IEEE 802.11b) so that all potential interferers can decode the control frames. The resulting control overhead can be quite high compared to the time needed to transmit a packet, specially at the higher rates in IEEE 802.11b/g, (11/54 Mbps). For example, a 20 byte control frame at 1 Mbps looks like 220 bytes of overhead when the data rate is 11 Mbps.

2. IEEE 802.11 uses random backoff to resolve multiple nodes contending for the medium. Every node maintains a contention window (CW), initialized to $\text{CW}_{\text{min}} = 31$ slottimes, to estimate the channel interference level. A node must wait for a random interval chosen uniformly in $[0, \text{CW}]$ slottimes before initiating a four-way handshake. If the four-way handshake fails, the value of CW is doubled before initiating the next attempt.

We now discuss some typical scenarios where this inefficiency destroys performance in a big way. Consider first the scenario shown in Figure 1, with N co-located nodes and a base-station serving Voice over IP (VoIP) traffic. The performance in this scenario suffers because IEEE 802.11 is not very efficient at sending small packets, specially at higher data rates. In Table I, the overhead of four-way handshaking is compared to the cost of data transmission at different data rates and different packet sizes, assuming long preamble mode and ignoring random backoff periods.

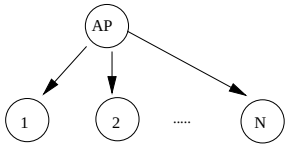

Fig. 1. AP serving N VoIP connections.

Consider next the scenario in Figure 2 with N nodes in a single line, and a TCP flow traversing this line from node 1 to node N. To successfully deliver packet p from 1 to N, IEEE 802.11 needs to successfully forward p along $k-1$ hops, and then successfully forward $ACK(p)$ along $k-1$ hops in the reverse direction [1]. The ACK packets in the reverse direction contend with the DATA packets in the forward direction for access to the wireless medium. The small size of TCP ACK packets exacerbates the inefficiencies of the four-way handshake in IEEE 802.11.

[1]Assuming delayed ACKs are disabled

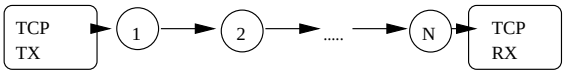

Fig. 2. Multi-hop TCP flow.

TABLE II

RANDOM BACKOFF COSTS

	Control overhead/Data transmit time	
ETX	Data = 100 byte	Data = 1500 byte
1.0	426 %	28 %
1.5	1183 %	79 %
2.0	2612 %	174 %
3.0	6059 %	404 %

Finally, consider Figure 3, where a source is transmitting to a destination over a single lossy link. Lossy fading links cause the four-way handshake to fail multiple times before finally succeeding. The IEEE 802.11 channel access mechanism implicitly assumes failure of the four-way handshake is due to interference from other transmitters, and does a binary exponential backoff before attempting to transmit again. This can result in expensive random backoffs when IEEE 802.11 is used over lossy links.

In Table II, we compare the cost of random backoff to the cost of data transmission at 11 Mbps at different link loss rates and packet sizes, ignoring four-way handshaking cost and inter-frame spacings. (ETX is related to forward and reverse link loss rates p_l^f and p_l^b as $ETX \triangleq \frac{1}{(1-p_l^f)(1-p_l^b)}$.) When losses are due to fading, and not interference, this expensive random backoff is not always necessary.

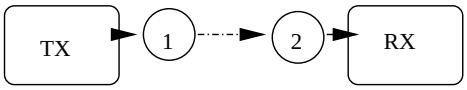

Fig. 3. Lossy link.

These inefficiencies of four-way handshaking primarily result from the attempt to fashion a unicast link from what is essentially a broadcast channel. In other words, the expensive cost of floor acquisition is a failure to take advantage of the broadcast nature of wireless communication. An alternative paradigm that takes better advantage of wireless broadcast is a transmitter-centric floor acquisition mechanism which acquires the channel simultaneously to multiple receivers in a probabilistic manner. This intuition is grounded by three observations:

1) Suppose that a transmitter T has acquired the floor to a particular receiver P (called “primary”) using a RTS-CTS handshake. Observe that the channel to some other secondary receivers may also get acquired as a consequence. This could happen, for example, if all of a secondary receiver’s neighbors have also been silenced by the RTS/CTS frames to/from the primary receiver. This is shown in Figure 4, where the secondary receiver S has neighbors T, P, A and B. A is silenced by RTS(T-P) and B is silenced by RTS(P-T) and thus, a transmission from T to S can safely proceed after acquiring the floor from T to P.
2) This could also happen if none of the secondary receiver’s unsilenced neighbors has a packet to send. For example, in Figure 5, the secondary receiver S has an additional neighbor C which is neither silenced by RTS(T-P) nor CTS(P-T). However, the transmit queue at C is empty, and thus the transmission from T to S can

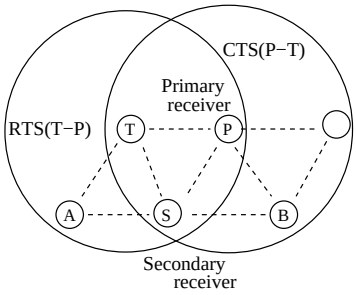

Fig. 4. All of the secondary receiver's neighbors are silenced by the RTS-CTS to the primary receiver.

safely proceed after acquiring the floor from T to P.

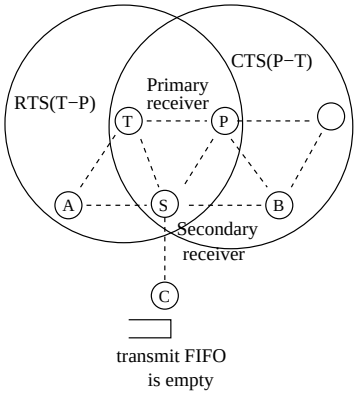

Fig. 5. Secondary receiver's unsilenced neighbors do not have a packet to send.

3) A third scenario is which the transmission from T to S can safely proceed after acquiring the floor from T to P is when all the unsilenced neighbors of S can sense carrier for the transmission from T to P, and will defer from interfering with this transmission. This is shown in Figure 6, where S has an additional neighbor C that is not silenced by RTS(T-P) or CTS(P-T), but can sense the carrier tone when T transmits to P.

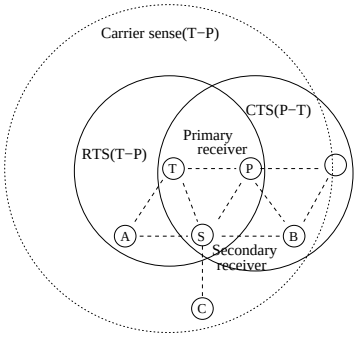

Fig. 6. Secondary receiver's unsilenced neighbors can sense the carrier of the node's transmission.

III. Proposed Protocol

We now describe the design of Splay. Splay is an attempt to address the inefficiencies of IEEE 802.11 floor acquisition by taking advantage of the fact that wireless is a broadcast medium. The observations in Section II suggest a natural solution to the inefficiency of the IEEE 802.11 four-way handshake:

1. Every node puts its wireless interface in "promiscuous" mode. When a node has a packet to transmit to a primary receiver, it runs through its neighbor list and identifies secondary receivers to which it can safely transmit a packet.

These neighbors are chosen so that the probability of the secondary receiver successfully decoding the packet is maximized.

2. The node then opportunistically combines packets to “good” secondary receivers with a packet to the primary receiver, adding appropriate Splay headers. This combined packet is placed in a packet with the primary receiver’s address as the destination address.

3. The combined packet is passed to the MAC, which sends the packet to the primary receiver using a four-way RTS-CTS-DATA-ACK handshake.

4. Since wireless is a broadcast medium, the secondary receiver also decodes the packet with some probability.

When this combining mechanism is employed, the transmitter only gets MAC ACK feedback from the primary receiver. On the other hand, the floor to the secondary receiver has not been completely reliably acquired using a RTS-CTS handshake and thus, may suffer packet loss if the reception at the secondary receiver happened to get destroyed because of interference/fading. By appropriately choosing the secondary receiver, the probability of this event can be reduced. However, we still need to incorporate a higher layer mechanism in Splay to guarantee reliable delivery for secondary transmissions. This could be a simple ARQ mechanism like that used by IEEE 802.11, or more sophisticated techniques like forward error correction.

Splay’s reliable delivery mechanism disables IEEE 802.11 retransmissions. The IEEE 802.11 32-bit CRC provides a bit-level reliability check and effectively converts the wireless channel between sender and receiver into a packet erasure channel. In the language of coding theory, the CRC is an *inner code*. To augment this inner code, Splay uses an *outer code* to deliver packets reliably over the erasure channel. Packet erasure codes can deal with bursty losses and are good candidate outer codes. This outer code operates over blocks of packets from the same transmitter to the same receiver. The receiver uses a block ACK to indicate successfully decoding of the block, and allow the transmitter to proceed with the next block.

A. Architecture

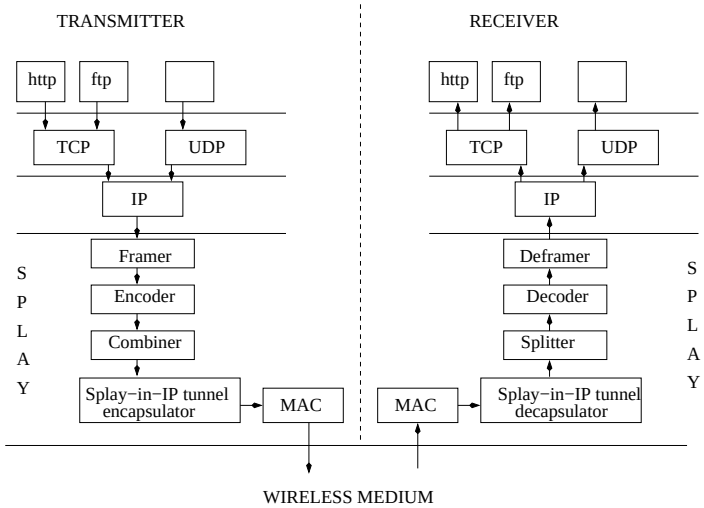

Fig. 7. Splay architecture.

The Splay architecture is shown in Figure 7. Packets are passed down to the Splay layer from the network layer after the next hop has been determined and may or may not be Ethernet encapsulated. Splay can carry either IP packets or Ethernet frames, or a combination of the two in its payload. Splay first queues the packets in a per next-hop queue and adds a frame header shown in Fig. 8 before passing "frames" to the encoder. This header is used to distinguish between IP and Ethernet frames at the deframing module. The framer optionally ensures that all frames have the same size. This is required if sophisticated erasure encoding schemes are used for reliability. In this mode, the framer may combine multiple packets of the same type (IP/Ethernet) into one frame to equalize frame sizes before adding the frame header. This operation does not preserve packet boundaries. The framing and encoding process can also be selectively disabled on a per-socket basis for applications that generate real-time packets and need expedited access to the channel without requiring reliability.

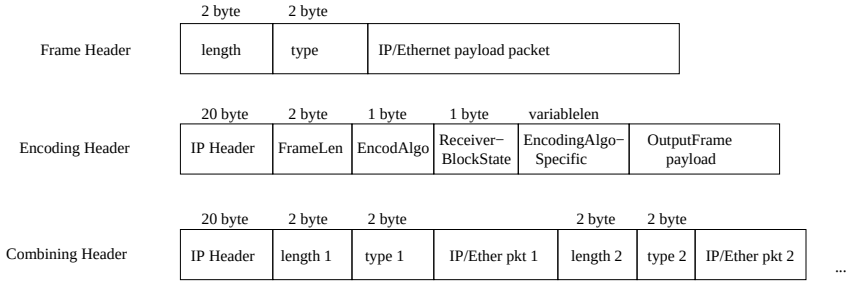

Fig. 8. Splay header formats.

The encoder operates on a per receiver basis and successively converts "input frames" received from the framer into "output frames", adding a header shown in Fig. 8. The encoder design can incorporate different erasure coding strategies and is not tied down to a particular strategy. It can also completely avoid the use of erasure codes, and work purely as a block ACK mechanism. We use a block encoding strategy, with all encoding algorithms operate on blocks of eight input frames, and use a block stop-and-wait ARQ. The first four bytes of the header contains three fields: $FrameLen$, $EncodAlgo$ and $ReceiverBlockState$. $FrameLen$ the length of each input frame arriving from the framer in bytes. $EncodAlgo$ specifies the erasure code algorithm used. The $ReceiverBlockState$ byte is used to piggyback decoding information for the reverse direction, with each bit indicating whether the corresponding frame has been successfully decoded or not. When the need arises, the encoder can also generate a dummy output frame with the $EncodAlgo$ and $FrameLen$ set to zero. Such an output frame is used when $ReceiverBlockState$ information from the decoder needs to be sent out in the reverse direction immediately. The format of the remaining part of the encoding transport header is variable and depends on the value of the $EncodAlgo$ field.

As an example, consider a digital fountain code like an LT code [2] [2]. The encoder for this code works by randomly picking a value d from a fixed degree distribution, and randomly combining d input symbols by XORing them together. In this case, the variable length $EncodingAlgoSpecific$ is a one byte field that is used to indicate which input frames are being XORed together in that particular output frame, with each bit indicating whether the corresponding frame is included in the addition or not. Another example of an encoder is the $Null$ encoder, which uses this one byte format of $EncodingAlgoSpecific$ to indicate the specific frame being transmitted. $Null$ does not do any encoding/decoding,

[2]LT codes work best with long block lengths and may not be ideally suited for Splay

and instead merely does retransmissions based on the information in $ReceiverBlockState$ block ACK field. *We emphasize that the encoder module only provides a mechanism to incorporate erasure coding, and thus, different erasure codes can be used in this framework.* The only abstraction that an encoding algorithm must follow is that it should accept input frames, and produce an output frame in the format shown in Fig. 8, packetized in an IP broadcast packet, with protocol number *SplayEncoderNum*. This output frame is then passed to the combiner. [3]

The combiner module of Splay opportunistically combines multiple packets destined for different next hops to reduce the cost of channel acquisition in IEEE 802.11. The combiner can be used to either do subnet layer splaying, in which the payload is Ethernet frames, or network layer splaying, in which payload is IP packets. When a transmit complete interrupt is received from the physical device, the combiner traverses the device transmit queue and evaluates different packet combinations, selecting the "best possible" one for transmission. (Alternatively, the combiner can be configured to pace packets into the network based on interference level. This is a soft scheduling mechanism similar to Overlay MAC [3].)

The combining mechanism consists of a list of $(matchrule, reward)$ two tuples. Each $matchrule$ represents a Boolean condition that may or may not be met by a packet combination, while the $reward$ represents the score obtained if the $matchrule$ is matched. The score assigned to a packet combination is the sum of $rewards$ for each $matchrule$ that the combination matches. Once a combination of packets has been selected from transmission, a Splay packet containing this packet combination is formed and encapsulated in an IP broadcast packet with TTL 1 and protocol number *SplayCombinerNum*, as shown in Fig. 8. This packet is encapsulated in an Ethernet header destined to the "primary receiver" of the combination, and sent over the wireless channel. [4]

We note here that the key to reducing the cost of floor acquisition is intelligent packet combination. For instance, if the conditions described in Figures 4, 5 and 6 are met, the combined packet will have a high probability of being decoded successfully by the primary and secondary receivers. *We emphasize that the combiner module merely provides a mechanism to combine packets, and leaves the exact combining policy unspecified.*

At the receiver end, the wireless interface receives all overheard packets in promiscuous mode. Splay superpackets with IP protocol number *SplayCombinerNum* are demultiplexed by IP to the Splay splitter. The Splay splitter extracts packets destined for the receiver from the combined Splay superpacket and passes them to the appropriate (Ethernet/IP) input routine. (This is subtle if the payload is IP; see Section III-C for a discussion.) Packets that were injected into the combiner by the framing+encoding process are IP packets with protocol number *SplayEncoderNum*, and are demultiplexed to the Splay decoder, which feeds them to to the corresponding decoding algorithm to recover the original input frames. $ReceiverBlockState$ is updated to reflect successfully decoded frames. If the $Null$ encoder was specified, this process reduces to simply providing a block ACK. As the decoder successfully retrieves frames, it passes them in-order to the deframer which parses them to extract individual packets and deliver them to the appropriate (IP/Ethernet) input routing for processing.

[3]The use of IP broadcast enables the IP input routine at potential receivers to successfully demultiplex received packets to the Splay decoder.

[4]The unicasting of the Ethernet frame enables the four-way handshaking to the primary receiver; the use of IP broadcast enables the IP input routine at potential receivers to successfully demultiplex to the Splay splitter.

B. Layering

Splay resides between IP and MAC in the stack, and is a *virtual MAC layer* on top of IEEE 802.11 with two differences:

1. It takes advantage of the broadcast nature of wireless to reduce IEEE 802.11 floor acquisition cost by splaying packets.
2. The use of erasure codes provides a more sophisticated mechanism for reliability to "secondary receivers".

The Splay framer/encoder and combiner produce two levels of tunneling, and the receiver consists of two successive demultiplexers above IP to decapsulate each of the encapsulations. This double tunnel separates erasure coding from the combining mechanism and allows them to be independently used.

C. Discussion

1. *Subnet layer splaying versus network layer splaying*: Splay can carry Ethernet or IP packets as payload. When subnet layer splaying is used, a receiver uses destination Ethernet address in individual frames contained in a Splay superpacket to filter out packets that are not addressed to it. On the other hand, when network layer splaying is used, all receivers that successfully decode a superpacket will receive all its constituent IP packets. This brings up a tricky question - which of these receivers should forward the IP packet to its destination? One simple way is to use a reverse path check - the only receiver that forwards it is the one for whom the sender is the next hop on the reverse path back to the source. On the other hand, these multiple receivers can collaboratively forward the packet by mediating the order in which each of them attempts to forward. For example, each receiver could wait for a time proportional to its ETX to the destination before forwarding. This takes us into the realm of protocol design for receiver oriented forwarding, of which the ExOR protocol [4] is an excellent example.

2. *Coding versus combining*: There is a natural architectural tension between erasure coding and combining. Erasure coding techniques codes work better when operating on a huge number of blocks at a time. To do this while keeping the delay bounded, frame sizes need to shrink. On the other hand, splaying aggregates packets in order to do multi-receiver floor acquisition to as many neighbors as possible. This necessitates large packet sizes to provide maximal amortization of floor acquisition overhead.

3. *Disabling of link layer retransmissions*: Disabling of link layer retransmissions also disables exponential contention window backoff in IEEE 802.11. One might argue that this is incompatible with IEEE 802.11, and is akin to what an "unpoliced selfish node" (and for that matter, Cisco Aironet cards) would do. We believe that this is reasonable provided the operating environment is limited by multipath fading effects (and high link ETX values) as opposed to interference. A backup contention resolution mechanism is provided to deal with high interference regimes by conservatively setting CW_{min}, and slowly adapting it linearly with the number of neighbors.

IV. Combiner Module

We now describe the combining mechanism and combination policies in greater detail.

TABLE III

RULE SPECIFICATION FOR $R(< p >)$

Attribute	Description	Allowed operators	Allowed values	Reward type
len	Pkt length of $< p >$	$\leq \ \geq \ =$	integer	fixed/scale
ack	Does $< p >$ contain TCP DATA and ACK of the same connection?	$=$	true/false	fixed
$numpkt$	Number of aggregated pkts in $< p >$	$\leq \ \geq \ =$	integer	fixed/scale
$fair$	Jain's fairness index of $< p >$	$\leq \ \geq \ =$	(0,1)	fixed/scale
age	Sum of queuing delays of pkts in $< p >$	$\leq \ \geq \ =$	real	fixed/scale
dst	Does $< p >$ contain pkt to dst d?	$=$	dst IP addr d	fixed
$fadingsuccess$	Total success probabilities of pkts in $< p >$	$\leq \ \geq \ =$	real	fixed/scale
$intersuccess$	Total success probabilities of pkts in $< p >$, except primary	$\leq \ \geq \ =$	real	fixed/scale
$metric$	Total metric of pkts in $< p >$	$\leq \ \geq \ =$	real	fixed/scale

A. Combination Mechanism

Let $\Omega = \{p_1, ..., p_N\}$ be the packets in the queue at node i. For each packet combination $< p > \in 2^{\Omega}$, we associate attributes describing various properties of the packet combination. For each attribute, we associate a set of permitted values and relational operators. A preliminary list of these attributes, operators and values is summarized in Table III. The $fadingsuccess$, $intersuccess$ and $metric$ attributes are obtained by looking up the kernel forwarding and neighbor tables.

A rule $R(< p >)$ is a Boolean expression on the attributes of a packet combination $< p >$. Associated with each rule R is a reward $Rw(R, < p >)$ that is assigned to the packet combination $< p >$ if $R(< p >) = 1$. The reward Rw can either be "fixed", i.e., a constant, or the special value "scale". When "scale" is used, the value of the attribute is scaled by the amount specified in the reward field to obtain Rw. The combiner is configured by specifying a list L of (R, Rw) two tuples. For each packet combination $< p >$, this list is traversed and the $rules$ which the packet combination successfully matches are determined. The sum of $rewards$ for these $matchrules$ is the score for this packet combination. We would like to find a subset $< p^* > \subset \Omega$ that maximizes the net reward:

$$< p^* > = \text{argmax}_{<p> \in 2^{\Omega}} \sum_{l=1}^{|L|} R(< p >).Rw(R, < p)$$

B. Combination Policy

This mechanism allows us to implement and experiment with a variety of multi-receiver floor acquisition policies to enhance performance:

1. MTU: selects any combination of packets that satisfies the IEEE 802.11 MTU requirements. This is a necessary condition for successful packet combination, otherwise MAC fragmentation will destroy the reduced floor acquisition costs. It must be used in conjunction with other rules.

2. $MaxComb$: combines as many packets as possible. $MaxComb$ provides the maximum amortization of floor acquisition cost and thus provides the maximum throughput benefit. It has two problems. Firstly, it treats the links to different neighbors identically, even though the underlying wireless channel to these neighbors may be different. (For example, the link to one of the receivers may temporarily be very bad due to small scale fading. An intelligent policy would avoid this link till it becomes good again.) Secondly, it might exacerbate IEEE 802.11 short-term unfairness if the packet length distribution to different neighbors is asymmetric. This can be avoided by adding additional rules to select specific sub-components of the aggregated packet in a "fair" manner.

3. $MaxFadingSuccessComb$: combines packets to maximize the sum probability of success (or equivalently, the sum throughput). For each neighbor j, let $p_f(j)$ denote the measured probability of failure for a packet from node i to destination j (p_f is related to the ETX of the link as $p_f = 1 - \frac{1}{ETX}$). Let us assume that all packet failures are due to multi-path fading effects, and that interference effects are negligible. Thus, $p_f(j)$ is independent of which receiver we select as the primary receiver. Then $MaxFadingSuccessComb$ selects the packet combination $< p >$ that maximizes the sum of $1-p_f$ over all packets in that packet combination. Since $MaxFadingSuccessComb$ uses the sum probability of success as the metric, it is similar to $MaxComb$ in the sense that it favors packet combinations with as many packets as possible. Further, it weights links with greater chance of success higher and thus, it exploits wireless multi-user channel diversity at the time scale of ETX adaptation.

4. $MaxInterferenceSuccessComb$: combines packets to maximize the aggregated packet's chance of being successfully decoded by as many receivers as possible. This rule is designed for environments where interference effects dominate the performance of IEEE 802.11. For each secondary receiver s of an aggregated packet, we assign a value v_s which is equal to the fraction of its neighbors that are in the one hop neighborhood of the transmitter or the primary receiver. The score v assigned to the aggregated packet is equal to the sum of these values v_s over all secondary receivers in the aggregated packet. $MaxInterferenceSuccessComb$ then attempts to maximize v over all packet combinations and choices of primary receiver.

We note that $MaxInterferenceSuccessComb$ could have been designed to minimize $w = \sum_s w_s$, where w_s is equal to the fraction of neighbors that are not in the one hop neighborhood of the transmitter of the primary receiver. This rule, which we call $MinInterferenceFailureComb$, appears superficially similar to $MaxInterferenceSuccessCo$ on first sight. To see that this is not the case, observe that the optimum solution to $MinInterferenceFailureComb$ is to simply not combine any packets, since $w = 0$ in that case. On the other hand, the strategy of no combining is never optimal for $MaxInterferenceSuccessComb$.

5. $LongTermFairness$: works by measuring the long term average throughput to each of a node's neighbors, and adjusting the reward values to favor neighbors which have received a lower share of the channel bandwidth. The fairness timescale depends on the timescale of throughput averaging.

6. $QueueingDelay$: works by measuring the time spent by a packet in the queue, and adjusting the reward values to favor packet combinations with older packets in them. This is similar to aging in CPU process scheduling, and prevents packets from being queued indefinitely.

C. Combination Algorithm

The rich space of packet combination policies makes the problem of finding an efficient generic combination algorithm quite difficult. In fact, even algorithms for simple policies like $MaxFadingSuccessComb$ can be intractable.

Theorem 1: $MaxFadingSuccessComb$ is NP-complete.

Proof: Consider the integer bin-packing problem where there are N objects, with object i having a size c_i and reward r_i. We would like to find the combination of objects to place in a bin of size C that maximizes the reward payoff without overflowing the bin. This is the following integer linear program:

$$\begin{aligned} &\text{IBP}(c_i, r_i, C): \\ &\max_{n_i} \sum_{i=0}^{N} n_i r_i \\ &\text{s.t.} \\ &\sum_{i=0}^{N} n_i c_i \leq C \\ &n_i = 0, 1 \end{aligned}$$

It is well known that integer bin-packing is NP-complete. Suppose at the instant of packet combination, there are N packets in the queue with packet i having length l_i and destination $d(i)$. We can transform $MaxFadingSuccessComb$ into integer bin-packing by setting $r_i = p_f(d(i))$ and $c_i = l_i \; \forall \; i$, and setting C to the MTU of IEEE 802.11. This can be used to establish that $MaxFadingSuccessComb$ is NP-complete. More precisely, we can establish a polynomial time transformation between any instance of IBP and a corresponding instance of $MaxFadingSuccessComb$, thus proving the NP-completeness. ■

D. Heuristic algorithm

We use a simple heuristic algorithm to solve the generic packet combination algorithm by trying to identify "good" packet combinations. The approach consists of using a greedy algorithm to solve an (off-line) bin packing problem. We assign each packet p a score s_p that evaluates the marginal reward obtained by including the packet in the combination, and then try to find a packet combination that maximizes the sum of packet scores subject to MTU constraints. Remember that rules are of two types: rules that assign a fixed reward based on whether a Boolean expression evaluates to true or false, and rules that assign a reward proportional to a computed metric. For the former rules, if the inclusion of the packet in a combination causes the expression to evaluate to true, the reward for the corresponding rule is added to the packet score. If the inclusion of the packet will cause the expression to evaluate to false, the reward for the corresponding rule is subtracted from the packet score. For the latter rules, a marginal

TABLE IV

MARGINAL REWARD COMPUTATION

Attribute	Packet Metric
len	length of packet
$numpkt$	one
age	queuing delay of packet
$fadingsuccess$	success probability of packet
$intersuccess$	success probability of packet
$metric$	metric of packet

reward proportional to the metric computed on the packet is added to the packet score, with the individual metrics as shown in Table IV.

Now, we consider the problem of maximizing the total packet score in a packet combination subject to link MTU requirements. This is the problem:

$$\max_{n_i} \sum_{i=0}^{N} n_i s_i$$

s.t.

$$\sum_{i=0}^{N} n_i l_i \leq MTU$$

$$n_i = 0, 1$$

and as in the case of $MaxFadingSuccessComb$, is intractable. [5]. While dynamic programming can be used to efficiently solve this problem, there is an alternative greedy algorithm to solve the problem that is much easier to implement:

1. Sort the packets in decreasing order of the packet score per byte of the packet, $\frac{s_i}{l_i}$.
2. Greedily add packets to the combination according to this ordering.
3. Stop if adding an additional packet to the combination will violate the MTU requirements.

This heuristic algorithm is not optimal, and has the following limitations:

1. It ignores scale rules like $fair$ whose marginal impact on a single packet cannot be computed.
2. It ignores fixed Boolean rules whose expression cannot be evaluated based on the presence or absence of a packet.
3. It ignores rules like $intersuccess$, where the total score depends on the permutation of packets in a combination.

Indeed, it is rules like the above ones that make the solution of the general problem difficult.

V. ENCODER AND DECODER MODULES

We now describe the implementation of an encoding algorithm in our encoding framework. We emphasize that the framework allows for the implementation of a variety of encoding strategies, and this section just provides guidelines for a candidate encoder. There are two encoding algorithms we are considering for an initial implementation:

1. Block stop-and-go ARQ (the $Null$ algorithm)
2. LT codes [2]

[5]It is equivalent to the 0-1 knapsack

A. LT codes

Luby Transform (LT) codes, invented by Michael Luby, were the first realization of the "digital fountain" concept introduced in [5]. These codes are rateless in the sense that the number of encoding frames that can be generated from the input data is potentially limitless. The decoder only needs to receive a number of encoded frames slightly greater than the number of input frames to be able to successfully decode the input data. In other words, LT codes are near optimal with respect to any erasure channel [2]. LT codes are non systematic in the sense that the set of input symbols are not necessarily part of the codebook generated at the transmitter.

The LT encoder works by randomly choosing a degree d from a degree distribution function. The encoder then combines d randomly chosen input frames by XORing the bits together to produce the output frame. In order to decode the frames correctly, the decoder needs to know, for each received output frame, the degree and the input frames which were combined to produce it. This information is carried in the $EncodingAlgoSpecific$ part of the encoding transport header as an extra byte. The single byte of $EncodingAlgoSpecific$ is used as a bit field to indicate which of the eight input frames were used to produce the output frame.

At the decoder, a belief propagation algorithm is used to decode the input frames. Suppose the receiver has a set O of output frames, along with the graph representing which input frames that were used to combine these output frames. For each output frame f, we maintain a variable d_f representing the number of input frames that were used to produce this output frame and have not yet been decoded correctly. We also maintain a set I of input frames that have been decoded, initialized to ϕ. Then, the decoding algorithm is as follows:

1. Check if an output frame f has been received. If yes, run through through the set I and XOR the value of each input frame in I with the output frame f. The value of f is the result of this XORing operation. f is then added to the set O. d_f is set to the number of neighbors of that frame, minus the number of elements in the set I. This reflects the fact that the value of f represents the XOR of the undecoded input frames that are not in I.
2. Run through the list O and check for an output frame with degree $d_f = 1$. If such a frame f exists, then it is a copy of the corresponding input frame, and the input frame can be recovered exactly. Add this input frame to the set I. The value of each of the output frames f' in O is replaced by XORing with the recovered frame, and set $d_{f'} := d_{f'} - 1$.
3. Remove all frames f such that $d_f = 0$ from O.
4. Check if I is equal to the set of all input frames. If yes, STOP. Else, go back to step (1).

The key to the design of the LT code is the choice of the degree distribution function $\{\rho(1), \rho(2), ...\rho(k)\}$. The seminal paper by Luby [2] demonstrated that the robust soliton distribution, described below, is a good choice of distribution.

Definition 1: (Ideal soliton distribution): The ideal soliton distribution is $\{\rho(1), \rho(2), ..., \rho(k)\}$, where

- $\rho(1) = \frac{1}{k}$.
- For $i = 2, ..., k$, $\rho(i) = \frac{1}{i(i-1)}$.

Let δ be the allowable failure probability of the decoder to recover the data for a given number k of encoding symbols.

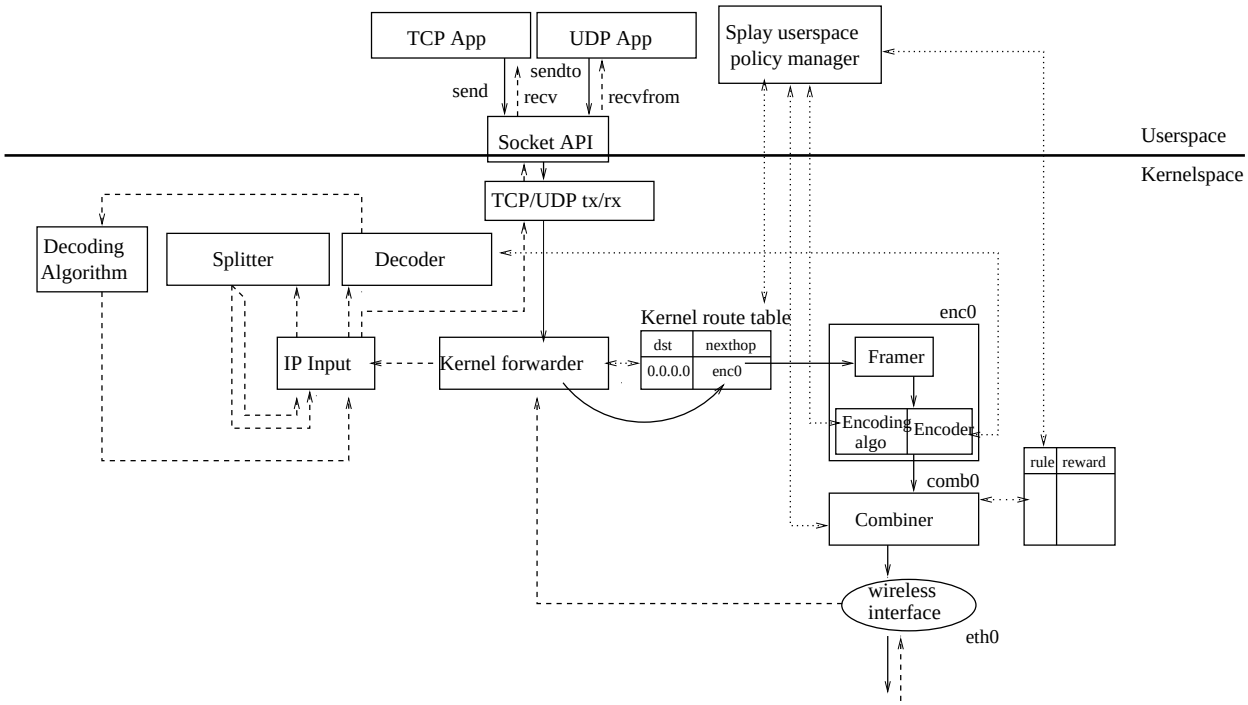

Fig. 9. Splay implementation architecture.

Definition 2: (Robust soliton distribution): Let $R = c \ln \frac{k}{\delta}$ for some $c > 0$. Define

$$\tau(i) = \left\{ \begin{array}{cc} R/ik & \text{for i = 1, ..., k/R - 1} \\ R \ln (R/\delta)/k & \text{for i = k/R} \\ 0 & \text{for i = k/R + 1, ..., k} \end{array} \right\}$$

Add $\tau(.)$ to the ideal soliton distribution $\rho(.)$ and re-normalize to obtain the robust soliton distribution $\mu(.)$[6].

VI. Implementation Architecture

We are currently in the process of implementing the Splay protocol as a Linux kernel module. The implementation architecture is shown in Figure 9, with dashed arrows indicating receiver data flow, solid arrows indicating the transmitter data flow and dotted double arrows indicating information exchange between modules.

A. Receiver design

1) Splitter: The splitter is implemented as a new transport demultiplexer above the IP layer. When the IP input module gets a broadcast "combined" IP packet with protocol field = $SplayCombinerNum$, it demultiplexes it to the Splitter transport. The "combined" packet has an IP header, and contains one or more packets. The 4 byte header before each of these packets enables the splitter to identify whether the packet is IP/Ethernet. The splitter simply runs through the jumbo packet, and extracts the constituent packets one by one. Ethernet frames destined for the node are delivered to the Ethernet input routine. All Ethernet frames not destined to the node are discarded. IP packets are sent back to the IP layer receive module for processing.

[6]In our implementation, $k = 8$

2) Decoder: The decoder is also implemented as a new transport demultiplexer above the IP layer. It receives IP broadcast packets with transport field set to $SplayEncoderNum$ from the IP layer receive module. The local encoder's transmit state information is stored in a hash table, keyed by destination address. The $ReceiverBlockState$ field in the received packet is used to update the local encoder's transmit state information of the corresponding destination by indexing into the hash table. If all the transmit packets in a block have been ACKed in the $ReceiverBlockState$ field, the transmitter_is_blocked flag for that next hop is cleared, and the local encoder is notified by calling encoder_transmitter_unblocked_notify(). The $EncodAlgo$ field in the packet is then used as a selector to pass the packet to the corresponding decoding algorithm.

3) Decoding algorithm interface: A decoding algorithm accepts packets whose $EncodAlgo$ field are set to the corresponding encoding algorithm. It processes the received packet according to the semantics of the decoding algorithm. If it successfully decodes any frame, it should set the corresponding bit in the node's $ReceiverBlockState$ for that destination. This information will be piggybacked to the destination in reverse direction packets. After successfully decoding a frame, the decoding algorithm places it in a receiver frame queue for that destination. It then runs through this queue successively, passing all possible in-order frames for that destination up to the deframer module for deframing. If the $Null$ encoding algorithm are used, the decoding algorithm is not invoked and received frames are passed directly to the deframer after appropriately updating the $ReceiverBlockState$ from the $EncodingAlgoSpecific$ field. Since frames are delivered in-order, the deframer design is considerably simplified and there is no need for sequence numbers.

4) Deframer: The deframer receives frames in-order from the decoding algorithm, and successively reconstructs IP/Ethernet packets out of the frames by parsing the framing header and extracting bytes across frames till the end of the packet is reached. The deframer also needs to take care of padded frames, and boundary conditions that arise from the splitting of the IP header across frames. After extracting the individual IP packets, it passes them to the appropriate (IP/Ethernet) input function for processing.

B. Transmitter architecture

We are implementing the framer and combiner as master-slave virtual devices which accept packets, process them in some way and then pass them to a slave device. We use the Linux tc utility to configure the slave device through user space. Suppose we wish to disable the framer and only use the combiner, we point the default route at the combiner device and configure the wireless interface as a slave to the combiner. To use both the combiner and framer, we point the default route at the framer, configure the combiner as a slave to the framer, and the wireless interface as a slave to the combiner.

1) Framer: A socket option is added to control framing on a socket, and the framing_enable flag is copied onto all packets generated on that socket. When the framer receives a packet, it checks the framing_enable flag on the packet. If the flag is clear, the packet is simply passed down to the slave device.

If the flag is set, the packet is then enqueued in a FIFO queue. If the queue fills up, the device's busy flag is raised to prevent higher layers from sending more packets. (Our implementation does not use a per receiver queue and thus,

there could be head of line blocking and other artifacts associated with a single queue.) The queue size is set to the approximate number of neighbors x 8 to ensure that all the neighbors' encoder queues can be kept full. The framer then runs through the list of all next hops, and calls framer_transmitter_unblock_notify() for all next hops with the transmitter_is_blocked flag set to clear.

The framer_transmitter_unblock_notify() function is called whenever the transmitter_is_blocked flag is clear for a next hop. It calls create_frame() to generate a frame and if successful, sends the frame down to the encoder. The framer_transmitter_block_notify() function is called whenever the transmitter_is_blocked flag is set for a next hop. As of now, this function does nothing.

The create_frame() function checks the framer queue to see if a frame can be generated from the available data. If the $Null$ algorithm is to be used, this is a trivial operation. On the other hand, if the encoding algorithm requires equal length frames, the test used is as follows:

1. If total packet length for that next hop in the queue is greater than the frame length, or
2. If total packet length for that destination in the queue is less than the frame length and (frame length - total packet len for destination) is less than IP version field length, or
3. If total packet len for that destination in the queue is less than the frame length and time elapsed since last frame is greater than sysctl_max_time_elapsed, or
4. If total packet len for that destination in the queue is less than frame length and sysctl_always_generate_on_invocation is set,

then a new frame is generated, padding with zeros at the end of the frame if necessary. Creating the frame simply involves removing upto a maximum of frame length unused bytes for the destination from the queue, and placing them in a frame data structure.

Finally, create_frame() adds a framing header (Fig. 8) to the new frame before passing it down to the encoder.

2) Encoder: When the encoder receives an input frame, it enqueues the frame in a per destination queue. The encoding algorithm is invoked to generate an IP packet containing the output frame if possible. If an output frame is generated, it is passed down to the slave device. When the all eight input frames of the block for that destination are received, the transmitter_is_blocked flag is set for that destination, and the encoder_transmitter_block_notify() is called.

The encoder_transmitter_block_notify() function invokes the encoding algorithm to generate additional frames to pass down to the slave device. The number of frames generated at this stage depends on the nature of the code used. For example, with systematic codes, each of the input frames is also an output frame. With the $Null$ encoder, the input frames are the only frames used as output frames. Thus, when encoder_transmitter_block_notify() invokes the encoding algorithm, eight output frames have already been generated, and the new invocation of the encoding algorithm need only generate extra frames if necessary. On the other hand, some encoding algorithms can only generate valid output frames after the entire block of eight input frames has been received. For these codes, the invocation of the encoder through encoder_transmitter_block_notify() to generate all the output frames.

3) Encoding algorithm interface: Whenever the encoding algorithm is invoked, it goes through the list of input frames and generates an output frame if possible. Systematic codes can generate output frames even if the entire list of input frames are not available. The encoding algorithm must append an IP header and an encoding transport header at the head of each output frame, with the format as specified in Section III-A and the EncodAlgo field set appropriately.

4) Combiner: The combiner is configured with the wireless interface as a slave device. It accepts IP packets from higher layers, which are queued on receipt in a combiner queue. Whenever the wireless device becomes available, the combiner invokes the combination algorithm to decide which subset of queued packets to combine. This subset of IP packets is concatenated into a single packet, and a unicast IP header directed to the primary receiver is appended. The combined IP packet is then passed down to the wireless device. The combining mechanism inside the kernel consists of a (rule, reward) list of two tuples, configured using netlink sockets through the tc user space application.

VII. Discussion: Expected performance improvements with TCP

One of the primary motivations of Splay is to reduce the channel acquisition costs of 40 byte TCP ACKs. This can be effected by adding the Splay rule 'ack = true' with a very high reward. (In fact, by making its reward greater than the sum of rewards of other rules, we can in effect prioritize this rule.)

Without the implementation on hand yet, we resort to quantifying the performance improvement due to Splay by simulating an idealized example with $n+1$ nodes in a line using ns-2. We first measure the performance of a single TCP flow carrying 1400 byte payload from the leftmost node to the rightmost node. Next, we investigate a flow pattern consisting of $n+1$ UDP flows in total. The first n flows are single hop UDP flows between nearest neighbors in the forward direction, and the $n+1^{th}$ flow is a one hop flow in the backward direction at the end of the line, as shown in Fig. 10. The first forward flow uses a 1412 byte CBR payload, while the other $n-1$ forward flows use a 1452 byte CBR payload. The single reverse UDP flow originates from the destination and uses 12 byte payload. By adjusting the (common) input packet rate of each flow, we can determine the saturation throughput. This simulates the situation when Splay combining is used to combine the ACK packets in the reverse direction with the data packets in the forward direction. (This calculation ignores the reverse direction traffic generated by FTP.) The performance difference between the two scenarios provides an upper bound on the performance boost we can expect when TCP is used over the Splay combiner (some of this improvement is due to our use of UDP in the second scenario instead of TCP.) This throughput comparison is shown in Fig. 11 as a function of n. (The throughput number shown is the total network layer goodput.) It seems like the best possible expected performance boost from Splay combining can be expected to increase from 15% to 130% as n increases from 2 to 11.

VIII. Related Work

Point-to-point MAC layer techniques to mitigate high floor acquisition costs in IEEE 802.11 like frame bursting and block ACKs have been included in the IEEE 802.11e standard [6]. The idea of using wireless broadcast natively has recently received a lot of interest. Receiver oriented forwarding [4] intelligently chooses the next hop for a packet

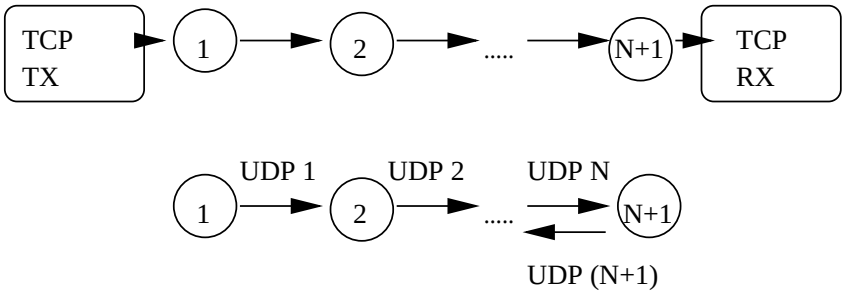

Fig. 10. Simulation scenario.

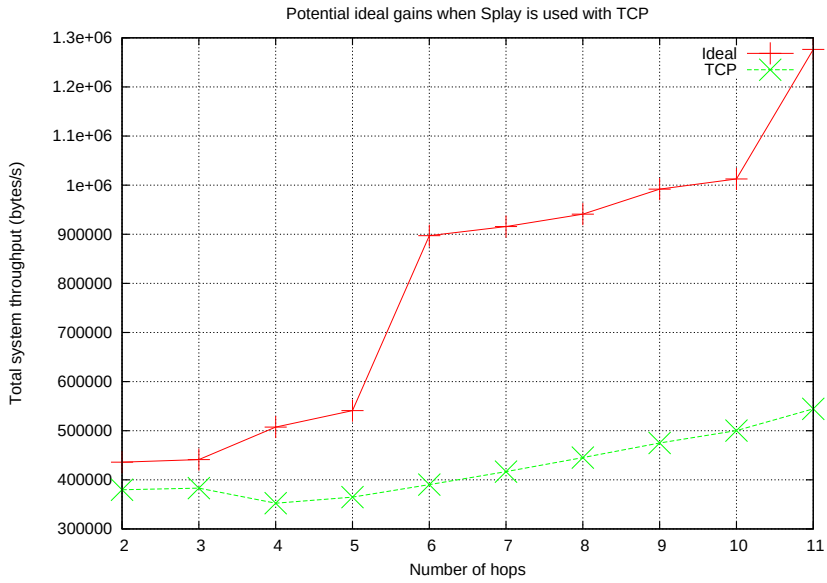

Fig. 11. Expected TCP performance boost from using Splay combiner.

after it has been broadcast on the air. This opportunistically takes advantage of "lucky events", where a packet makes a lot of progress towards its destination in one step. Such delayed next hop binding has also been explored in the MAC [7]. There is a growing interest in innovative ARQ techniques to replace IEEE 802.11 stop-and-go ARQ. Frame combining techniques aim to salvage a packet from successive, possibly erroneous, retransmissions [8], [9]. Hybrid ARQ uses a mixture frame combining and forward error correction for reliability [10]. There is a growing interest in using innovative coding approaches like network codes [11], [12].

IX. Conclusion

The traditional floor acquisition mechanism in IEEE 802.11 is very expensive in terms of overhead. This four-way handshaking is primarily a way to create a unicast communication link from what is fundamentally a broadcast medium. Our main thesis is that such unicast usage of the wireless channel is wasteful. Instead, we describe the design of Splay, a packet splaying protocol that attempts to acquire the floor and simultaneously transmit packets to as many potential receivers as possible. The design of Splay allows for the use of sophisticated erasure codes to replace the IEEE 802.11 ARQ for reliability. Splay is layered between IP and MAC and can be completely built in software. It is in the process of being implemented as a Linux kernel module.

REFERENCES

[1] IEEE 802.11 Wireless Local Area Network Working Group, *IEEE 802.11 Standard*. IEEE Standards for Information Technology, 1999.

[2] M. Luby, "LT codes," in *Proceedings of the 43rd Annual IEEE Symposium on the Foundations of Computer Science (STOC)*, 2002.

[3] A. Rao and I. Stoica, "An overlay MAC layer for 802.11 networks," in *Proc. MobiSys*, 2005.

[4] S. Biswas and R. Morris, "ExOR: Opportunistic multi-hop routing for wireless networks," in *Proc. ACM SIGCOMM*, 2005.

[5] J. W. Byers, M. Luby, and M. Mitzenmacher, "A digital fountain approach to asynchronous reliable multicast," in *Proc. ACM SIGCOMM*, 1998.

[6] IEEE 802.11 Wireless Local Area Network Working Group, *IEEE 802.11e/D4.0, : Wireless MAC and PHY specifications: MAC Enhancements for QoS - draft 802.11-01/566r3*, 2002.

[7] P. Larsson, "Selection diversity forwarding in a multihop packet radio network with fading channel and capture," *SIGMOBILE Mobile Computing Communication Review*, 2001.

[8] A. K. Miu, H. Balakrishnan, and C. E. Koksal, "Improving Loss Resilience with Multi-Radio Diversity in Wireless Networks," in *Proc. ACM MOBICOM*, 2005.

[9] P. Sindhu, "Retransmission error control with memory," *IEEE Trans. on Communications*, 1977.

[10] S. Lin and D. J. C. Jr., "Automatic-repeat-request error-control schemes," *IEEE Communication Magazine*, 1984.

[11] D. Lun, M. Medard, and R. Koetter, "Efficient operation of wireless packet networks using network coding," in *Proc. 2005 International Workshop on Convergent Technologies (IWCT)*, 2005.

[12] Y. Wu, P. A. Chou, and S. Y. Kung, "Information exchange in wireless networks with network coding and physical-layer broadcast," in *Conference of Information Sciences and Systems (CISS)*, 2005.