

JaldiMAC – Beyond Simulation

Yahel Ben-David, Seth Fowler, Shaddi Hasan
Department of Electrical Engineering and Computer Science
University of California, Berkeley
{yahel,sfowler,shaddi}@eecs.berkeley.edu

ABSTRACT

WiFi has been promoted as an affordable technology that can provide broadband Internet connectivity to poor and sparsely populated regions. A growing number of deployments, some of substantial scale, are making use of WiFi to extend connectivity into rural areas. However, the vast majority of the 3.5 billion people living in rural villages [1] are still unserved.

To reach these people, new technology must be developed to make small rural wireless Internet service providers (WISPs) profitable. In previous study [2], we have identified radio towers as the largest expense for WISPs; to reduce or eliminate this barrier to entry, we proposed a novel point-to-multipoint deployment topology that takes advantage of “natural towers” such as hills and mountains to provide connectivity even over great distances. In this work we make this design practical with two components that together make up JaldiMAC¹: Jaldi9K, an open source driver for commodity wireless hardware that offers comprehensive control over physical layer parameters and makes few assumptions about the overlying MAC layer; and JaldiTDMA, a TDMA scheduler that (i) enables and is optimized for point-to-multipoint deployments, (ii) adapts to the asymmetry of Internet traffic, and (iii) provides loose quality of service guarantees for latency sensitive traffic without compromising fairness. To our knowledge, JaldiMAC is the first integrated solution that combines all of these elements, and Jaldi9K is the first Linux wireless driver with the flexibility to effectively implement JaldiMAC on commodity hardware.

1 Introduction

Our goal is to extend broadband Internet connectivity to currently underserved rural communities in developing nations by developing a novel technical solution. In our previous work [2], we developed a novel deployment methodology aimed towards making the operation of rural wireless Internet service providers (WISPs) more profitable, and proposed the outline of a novel MAC protocol to support this deployment methodology. Traditionally, fixed wireless deployment topology make user of towers centrally located among subscribers and equipped with sector antennas. This approach is not economical for a rural WISP due to the high cost associated with towers for the low market densities. Our deployment model makes use of directional antennas atop distant mountains or hills to provide connectivity without the need of towers. More details about our deployment methodology, along with a comparison of different technologies available to WISPs are available in [2].

The focus of this work is the implementation of the MAC protocol outlined in [2] to support our novel deployment topology. We intend our MAC protocol to support different types of service, such as VoIP, audio/video conferencing, and bulk transfers. Our choice to integrate this concept from the beginning is motivated by the difficulty of retrofitting it onto an existing MAC protocol [8] [9] and by the inefficiencies of enforcing such policies at higher layers, which we discuss later. The often conflicting networking requirements of different applications make designing a MAC that provides good support for all of them a very complex task. Especially challenging from the perspective of the MAC layer is VoIP², which is very sensitive to jitter and uses very small packets. Many Internet applications exhibit dynamic and asymmetric traffic patterns that are also difficult to support. A holistic Time Division Multiplexed Access (TDMA) MAC design that incorporates all of these concerns is hard to produce for any team; to make the task more manageable, current solutions address each aspect individually, e.g., supporting asymmetric traffic patterns [10], providing service differentiation [9], or maximizing throughput [11] [12]. However, none offers an integrated approach to meet the requirements in their entirety. From our previous work, we have distilled a list of goals that we set out to achieve with JaldiMAC, our proposed MAC protocol:

1. Support a point-to-multipoint setup over long distances.
2. Respect dynamic traffic patterns with varying symmetry ratios.
3. Guarantee per-session fairness.
4. Provide strong quality of service (QoS) guarantees for different service classes.
5. Allow for administrative preferential treatment of subscribers and service classes.
6. Calibrate physical layer (PHY) parameters (bitrate, channel-width, transmit power, etc.) dynamically, based on changing physical RF conditions and traffic patterns.
7. Handle error correction to hide losses from overlaying TCP traffic while allowing congestion avoidance to work correctly.

The remainder of the paper begins with a study of related work in section 2 and continues in section 3 with a description of our system architecture, detailing both key components; our Jaldi9K kernel driver and the JaldiTDMA scheduler. We discuss our measurements and lab setup in 4 and conclude in section 5.

2 Background and Related Work

Although 802.11 was originally designed for LAN applications, its high proliferation and low cost drove its use beyond the

¹ Jaldi is the Hindi word for fast/quick.

² VoIP typically needs less than 16kbps of bandwidth and can tolerate latencies up to 1000ms, although user experience is better if kept below 400ms.

designer's intentions. Both its economic attractiveness and the ability to control some of its physical and MAC-layer parameters have inspired the use of 802.11 as a research platform [13] [14].

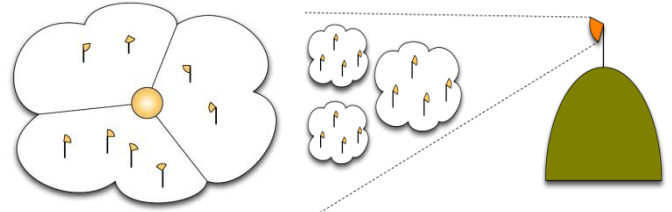
However, using 802.11 for a general-purpose wireless platform can be challenging. Its CSMA/CA MAC protocol was not intended for outdoors and fares poorly at overcoming collisions; this is a consequence of the combination of the hidden node problem, the difficulty of detecting collisions at the receiver, and the challenge of handling backoff fairly [15]. Long distances aggravate these problems, resulting in further inefficiencies and reduced channel capacity. Despite this poor performance, 802.11's low cost has inspired many researchers to explore ways of modifying the 802.11 MAC to maximize channel utilization and allow the deployment of high-bandwidth links over long distances. Most researchers suggest replacing the CSMA/CA algorithm with some type of TDMA in order to minimize collisions through synchronization [16] [10] [11] [12] [17] [18] [19]. The 802.11e standard was another attempt to address 802.11's shortcomings; unfortunately, 802.11e may starve traffic in some situations, and to the knowledge of the authors, it has never been fully implemented [9] [8].

2.1 Outdoor Application

Within the context of fixed outdoor applications, the field divides into three main categories: long-distance backhaul, "last mile" distribution, and mesh networks.

Long distance backhaul. 802.11 equipment is gaining a reputation of providing unmatched affordability while delivering excellent bandwidth over a very long distance. In fact, the experience of TDMA implementations such as WiLDNet [11] shows that available bandwidth is independent of distance with the only limitation being line-of-sight; a 382km link has been demonstrated [2]. These long-distance backhauls are ideal for linking cellular base stations and extending broadband Internet connectivity. Nevertheless, the design of WiLDNet assumes traffic is symmetric, making it very ill-suited for typical Internet applications where traffic is both highly asymmetric and dynamic. WiFiRe [12] tunes slightly better for Internet applications by assuming a 2:1 traffic ratio, but does not dynamically adapt to changing traffic patterns. WiLDNet also only allows for the creation of point-to-point links, which are impractical for the "last mile" links discussed below. The high-gain directional antennas which enable these long-distance links minimize interference as a welcome side effect; we will take advantage of this in section **Error! Reference source not found.**

"Last mile" distribution. The final links which service multiple stations from a connected center are collectively referred to as the "last mile". The stations are village homes or businesses; the central location is a base station connected to a wired or wireless backhaul. Traditionally, last mile solutions use 3 or 4 equally spaced sector antennas mounted on a tower in the center of the village [12]. Customer Premises Equipment (CPE) would use directional antennas aimed toward the central tower. This sector approach was modeled after the cellular telephone industry, and it is an unnecessarily expensive design for sparsely populated rural markets.



A: Conventional cellular-like sectors approach. B: Leveraging distant elevations.

Figure 1. Conventional deployment topology vs. our approach.

Mesh networks. At the root of many academic studies lies another approach for last mile connectivity: mesh networks [20] [21] [16] [22] [10] [17]. In a mesh, stations relay data over multiple hops; they can span long distances to offer an attractively decentralized and fault-tolerant design. Nevertheless, the cheap, widely studied single-radio mesh relays data in a store-and-forward fashion, which essentially halves the bandwidth at every hop, limits scalability, and involves complex computation that is challenging for low-cost hardware. In a mesh, stations should be in range of two or more other stations to allow relaying and provide redundancy; this prevents the use of directional antennas which would improve signal strength, reduce interference, and allow greater spatial density. Since directional antennas are crucial for long-distance links, long-distance mesh networks are currently unrealistic, although recent work on electronically steerable antennas [23] may eliminate this restriction. Our work must support long-distance links, so we do not explore this space further. However, we do borrow the concept of dynamic time slot allocation from JazzyMAC [10].

2.2 Improving Channel Utilization

Much research in this space focuses on improving channel utilization and density and compensating for the scarcity of non-overlapping channels [24] [10] [25] [12]. The optimizations in these studies improve efficiency when capacity, scale, and channel availability are an issue; in rural villages, however, this is not the case. Moreover, densely populated markets are able to utilize a growing number of solutions based on 4G technologies which may suit their requirements better than 802.11. For these reasons, we leave such considerations to future work.

2.3 Packet Loss

Sheth et al. [26] observed that rural wireless links have negligible loss; the main source of interference in urban networks is external WiFi networks, which are rare in rural, developing areas. The same paper notes that reducing the bitrate aggravates interference because longer transmission times increase the risk of collision; however, a lower sending bitrate increases receiver sensitivity, which is attractive for weaker signals attenuated over long distances in rural areas, as it yields a better signal-to-noise ratio (SNR). Local interference can be reduced further with techniques such as the use of higher quality antennas with smaller side lobes and better front-to-back isolation ratios, proper shielding of radios and coaxial cables, grounding, relocation of the system (even on the same roof), and changing the mounting height.

The effect of packet loss on TCP has been extensively studied [27] [28] [29] [30]. TCP assumes packet loss represents network congestion, but it is a ubiquitous characteristic of wireless networks which can be independent of network utilization. This problematic congestion assumption can be mitigated by either hiding the lossy wireless medium or using a modified TCP that differentiates between wireless loss and congestion [28]. Hiding loss better maintains the layering of the protocol stack; however, doing so aggressively can noticeably increase jitter and delay, which perturbs TCP timer calculations. The two most commonly used mechanisms to hide loss are forward error correction (FEC) and frame retransmission, also known as automatic repeat request (ARQ). Traditional 802.11 uses a per-packet ARQ scheme, but research in this field recommends FEC schemes in order to better mitigate bursty losses [11] [2].

Our topology minimizes the potential for interference and loss with highly directional antennas, and rural areas generally tend to exhibit low interference. Therefore, we do not focus on link-layer recovery schemes in this paper and leave the integration of such mechanisms for future work.

2.4 Extensions to 802.11

Previous work on alternative MAC layers [14] [13] [11] has leveraged the “packet injection” capability provided by several open-source wireless drivers to insert arbitrary packets at layer 2, and the Radiotap header standard provides a mechanism for controlling some basic physical layer parameters such as channel, bitrate, and TX power. However, the Radiotap standard does not provide the ability to tune any 802.11n physical layer parameters, even ones as basic as the Modulation and Coding Scheme (MCS). Lower level control over attributes such as configuration of hardware queues are completely inaccessible to higher level driver writers. Moreover, most common uses of packet injection tend to be oriented towards 802.11 network security applications [cite:aircrack-ng], and as such are not designed to support the implementation of non-802.11 MAC protocols.

In addition to these efforts, there have been several previous projects that have modified existing 802.11 drivers to support the development of novel MAC protocols. The open source Madwifi driver [cite:madwifi] has been modified for a variety of projects exploring novel MAC protocols for mesh networks. [cite:roofnet] [cite:ziptx] [6] Still others have developed MAC protocol development frameworks using heavily modified versions of Madwifi [8] [9]. However, all of these efforts have been focused on the previous generation of 802.11B/G hardware. While the concept is not new, to the best of our knowledge we are the first to develop a non-802.11 MAC protocol and MAC-agnostic hardware driver on 802.11n hardware.

3 JaldiMAC Architecture

The design and implementation of JaldiMAC proceeds from our insights about the cost structure faced by rural WISPs along with our desire to develop a flexible platform for future research using 802.11n hardware. JaldiMAC consists of two layers. The first of these defines the high-level behavior of the JaldiMAC protocol, which we refer to as JaldiTDMA. This layer is responsible for tasks such as building frame headers, calculating the TDMA

schedule, error control, and station addressing; we discuss it in depth in section 3.2. The second, lower layer is responsible for configuring the chip hardware and physical layer settings, as well as providing an interface for the higher layer to inject packets over the air. For this project, we’ve implemented JaldiTDMA using the Click Modular Router [cite:click] in user-space, while Jaldi9k is a Linux kernel module. Because of this, the interface between JaldiTDMA and Jaldi9k is simply the network interface that Jaldi9k exposes. While this ensures loose coupling between our system components, it is not an ideal long term solution, though our current implementation of JaldiTDMA in userspace requires it. Moving forward, JaldiTDMA should also move into the kernel, if for no other reason than performance. Because kernel support for Click is limited to a small number of older kernels for which Click patches exist, it may be simpler to implement JaldiTDMA as a standard kernel module. To support this, we would need to define a kernel-level API for JaldiTDMA and Jaldi9k. Defining this API, however, is outside the scope of this work.

Our architecture is somewhat similar to that employed in the Linux 802.11 stack [31], in that high-level protocol details are implemented in a hardware-agnostic way before being handed to a hardware-specific driver. Because of the complexity of the 802.11 standard, as well as the importance of interoperability between different vendors and their implementations, Linux has implemented a generic 802.11 layer that handles many common tasks associated with the 802.11 protocol, such as association and authentication. However, we take this design one step further: while hardware-specific drivers that take advantage of the Linux SoftMAC³ must implement significant and important protocol details such as beaconing and regulatory logic, we put all protocol implementation under the purview of the high-level component. Our hardware-specific, low-level driver is conceptually simple in comparison with the rest of the network stack; it provides only a minimal interface for sending arbitrary packets over the air.

The goal of this architecture is to provide a layer of abstraction between the MAC and physical layers, thus providing flexibility for future extensions to JaldiMAC, as well as the development of other non-802.11 compliant MAC layers. We strongly believe that cross-layer integration between the MAC and physical layers is important for the design of next-generation wireless solutions, since taking the unique physical properties of wireless into consideration can yield significant performance benefits as well as help avoid the pitfalls of exposing a false abstraction to higher levels of the networking stack. To be clear, we acknowledge the importance of layering for building an elegant and flexible system: rather than integrating their implementation, we provide an interface for sharing information between the layers and using this shared knowledge to make better decisions in the MAC layer. By directing all control over physical layer properties through this narrow interface, we are able to abstract the implementation of the protocol from the underlying hardware and driver, which provides a great deal of flexibility. Throughout our project, we have found this separation of concerns helpful in guiding our reasoning about the two distinct yet interdependent components of a full wireless MAC protocol.

3.1 The Kernel Driver - Jaldi9K

Our open source kernel driver for Atheros 802.11n chipsets, Jaldi9K, is an essential building block for implementing

³ The “mac80211” interface, not to be confused with [27].

JaldiMAC. It allows for transmission and reception of freely formatted packets, while also allowing complete control over the physical layer parameters as exposed by the hardware. Developing a new driver is necessary for a full, efficient implementation of JaldiMAC. In particular, we rely on precise timing feedback from the kernel driver to ensure RX and TX slots are accurately scheduled by the operating system and to make the higher layer aware of synchronization information that is necessary to make accurate scheduling decisions. The potential contribution of this driver is expected to go well beyond our project; we expect that the flexibility and degree of control it offers will enable many studies by other researchers. There are no affordable alternatives to Jaldi9K for 802.11n hardware; currently, researchers are generally restricted to costly Software Defined Radios or older hardware and limited flexibility of the drivers.

3.1.1 Detailed Design

While we were unable to complete the driver during the semester, we were able to make significant implementation progress and identify important requirements and goals for its design. Jaldi9K accepts two classes of frames: control and data. Transmission in Jaldi9K starts when a frame is received from Linux, presumably from a higher-level driver like JaldiTDMA. Each frame is marked with a “type” field in the header that specifies what kind of data it contains, in the case of data frames, or what kind of action the driver should take, in the case of control frames. Most of the actions that control frames request involve changing some physical layer parameter such as channel or bitrate. The header of the control frame contains a fixed-length bitmap specifying which physical layer parameters are present in the control frame, and the new values for each are found in the variable-length payload of the frame. Our control format is generally similar to the Radiotap standard, which despite the limited scope has proven to be flexible and extensible over time. We have left the formal specification of the control interface to future work.

All data frames are transmitted over the air in the same form they are received from the higher layer MAC protocol; we assume all relevant protocol headers have been supplied⁴. The distinction between data frames types is only used at the higher layer. JaldiTDMA needs some control over the time when frames are sent; this is accomplished using a type of control frame called a delay frame, which instructs the driver to wait for a specified period of time to send the next frame, switching the hardware to receive mode in the meantime. The driver sets kernel-level timers to ensure that transmission over the air begins as soon as possible after the requested amount of time has passed. Thus a single mechanism serves to define upstream slots for stations and to schedule the beginning of subsequent downstream slots. The higher-level driver determines delays using a straightforward calculation based upon the amount of data to be received and the bitrate. Thus, we have full control of timing at the JaldiTDMA layer, but benefit from the accurate timing the kernel can provide.

⁴ The hardware will append the Frame Check Sequence, a 32-bit CRC, to the end of the frame, as well as prepend a physical layer header. Additionally, the hardware will apply the specified 802.11 modulation scheme to the frame prior to transmission over the air. These per-frame modifications are transparent to the rest of our system (though we can optionally accept frames that fail the FCS check), and in any case are not possible to change without modifying the hardware.

The use of relative delays rather than absolute transmission times allows the higher layer to be unaware of nondeterministic latencies that may occur in the kernel and hardware; this design works because stations do not transmit until they receive a frame requesting them to do so.

When a data frame is received by Jaldi9K from the higher layer, we build a TX descriptor that contains per-packet hardware control instructions, including DMA information and frame aggregation flags. Each frame is stored in a queue in order of its desired transmission time (the higher level driver provides packets in order), and a timer is started whenever a delay frame is encountered. When the timer interrupt occurs, we dump the frames ready for transmission into a hardware transmission queue and initiate TX on each frame. Because the higher-level driver is responsible for error control, Jaldi9k can forget every frame as soon as it is transmitted.

Frame reception is initiated by an interrupt generated by the radio hardware. The Jaldi9K interrupt handler copies the received frame from the RX DMA buffer and appends the necessary RX descriptor, which conveys status information such as RSSI about the received packet. After checking a series of hardware receive filters - verifying the frame passed the CRC, for example - we pass the raw frame to the kernel, and presumably on to a higher-level driver like JaldiTDMA.

Because the Jaldi9K driver hides kernel-level and hardware-level delays from the higher layers, it is difficult for an overlying MAC protocol like JaldiTDMA to know exactly when all of the frames it has previously sent to the driver have actually been sent. To eliminate this problem, Jaldi9K will generate a “round complete” control frame and send it to the higher layer, in the same way that it might send a frame received over the air, when it discovers a “round complete” control frame at the head of its transmission queue. This simple mechanism is used to synchronize more complex behavior at the MAC protocol level. Our development process for the driver has been greatly assisted by the existence of ath9k, an open-source kernel driver for the hardware we are using. Unfortunately, the driver is highly unstable, tightly coupled with the implementation of the 802.11 protocols and the attendant regulatory compliance code, and poorly documented. As a result, even simply removing the 802.11 specific code from the existing driver has proven challenging. We hope that our Jaldi9K driver will provide a more accessible basis for future research leveraging 802.11n radio hardware.

Finally, since a key goal of Jaldi9k is to support very precise and accurate timing for frame transmission, we need to instrument our driver to supply profiling information about this and other aspects of driver performance. We do this by exporting data about the driver's operation to userspace via the kernel's debugfs filesystem. Currently we are only providing frame timing information, but we can easily extend our profiling interface as development continues. For each frame the driver handles we capture both its intended and actual transmission times using high-resolution kernel timers. We record these in two circular buffers which store the transmission times for the last few hundred frames. By polling the debugfs files that our driver exports, we can sample per-frame timings and gain an understanding of our driver's timing accuracy. While we have not been able to quantify the overhead that this profiling imparts to the system, we made similar changes for testing purposes to the ath9k driver and did not see a significant performance difference compared to an otherwise identical version of the ath9k driver that lacked the profiling

features. Table 1 shows key performance metrics from a series of experiments involving a single UDP flow between a client and host separated by a pair of wireless routers running identical kernels.

Table 1. ath9k with and without profiling overhead.

	Mean Throughput	Mean Jitter	Mean Loss
Profiling enabled	1.043 Mbit/s	0.285 ms	0.34%
Profiling disabled	1.043 Mbit/s	0.263 ms	0.43%

3.2 JaldiTDMA

JaldiMAC uses TDMA because it outperforms the standard CSMA/CA scheme of 802.11 for long-distance wireless. Because TDMA terminology is not standardized, we briefly define our terms here before discussing our design choices. Figure 2 presents a TDMA *schedule* that partitions time into continuous intervals; we distinguish a *window* that represents the horizon of the scheduling algorithm. The schedule consists of a repeating cycle of *rounds* with an internal structure called a *layout*. Each round begins with a *contention slot* for unscheduled requests like station joins; the size of the round is then the length of time until the next contention slot. We define a *slot* as the minimal allocatable time unit. A contiguous block of time in the round assigned to the same station is a *chunk*, and the set of all chunks a station receives in a given round is its *allocation*.

To avoid collisions, TDMA requires some method of synchronization. Maintaining a synchronized clock with sufficient accuracy is challenging [11], so we avoid this approach and rely on *polling*. In JaldiMAC, the master transmits the downstream data in chunks, rather than all at once as in some other protocols; included with each chunk of data is a control frame that indicates which station may transmit next and the length of time for which it may do so. The station begins transmitting after the downstream chunk is over, and continues until exhausting either its queues or its upstream chunk. If a station's allocation is not contiguous, it may go through this process several times over the course of a round. The time at which such a control frame is received thus serves to *implicitly synchronize* the transmitting station with the master.

The ARQ protocol used in 802.11 requires that each sent packet is acknowledged individually by the receiver; this is extremely inefficient for long distance links with high propagation delay. JaldiMAC instead employs a *bulk acknowledgement* scheme [11] using cumulative ACKs. Since our polling scheme alternates between downstream and upstream transmissions, we can implement this without unnecessary RX/TX switches by prefixing each transmission with any outstanding ACKs.

3.2.1 Dynamic Layout

Traditional TDMA schemes [12] [17] [11] use *static* layouts that only change when stations join or leave the network, and maintain a fixed ratio between upstream and downstream traffic. JaldiMAC uses the available bandwidth more efficiently by taking actual traffic needs into account to create *dynamic* layouts. Each station requests bandwidth in proportion to its anticipated traffic needs, and the master arbitrates these requests to produce a layout that is as "fair" as possible. Because each station may serve many users, requests are performed per session and not per station.

Stations normally make their requests during their upstream chunks to minimize the overhead of sustaining long transfers. Stations that had no upstream chunks in the previous round must request bandwidth in the contention slot. If there are many such stations, the potential for collision is high; to compensate, JaldiMAC proportionally expands the contention slot up to a maximum size. Collisions are further reduced by requiring each station to randomly choose a time within the contention slot for their requests.

3.2.2 Fairness

In the context of point-to-multipoint TDMA, *fairness* is the equitable sharing of time between clients. In JaldiMAC, each station reports its needs to the master, and the master decides how much time each station will be granted; this makes it natural to define fairness relative to the stations' per-session requests, leading us to use *min-max* fairness. Each session is automatically granted requested time up to its even share of the maximum round size; the excess is then split evenly among sessions that require more.

3.2.3 Service Classes

Network applications have widely varying requirements in terms of bandwidth, latency, and jitter. For example, file transfers are primarily sensitive to available bandwidth, while VoIP needs very little bandwidth but is highly sensitive to latency and jitter. JaldiMAC addresses these requirements by allowing stations to subdivide their requests into different *service classes*; these classes affect the station's allocation and the scheduling of its chunks.

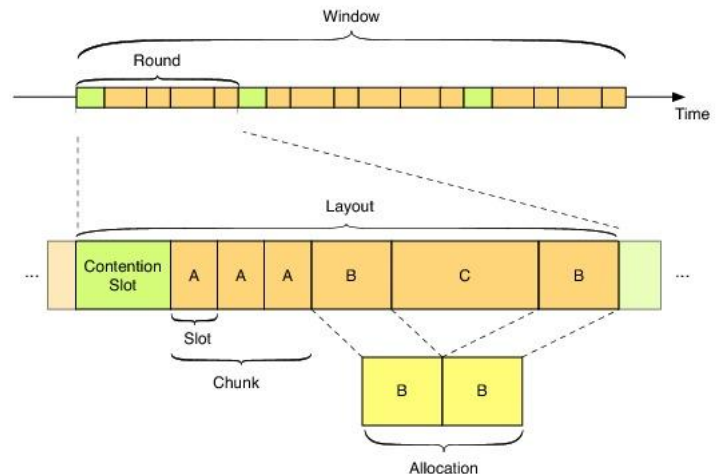


Figure 2. Our TDMA terminology.

Our experience suggests that two service models capture the requirements of the vast majority of types of traffic. *Bulk* traffic seeks to maximize throughput with no regard for latency or jitter. *Latency sensitive* traffic attempts to minimize latency and jitter at the expense of throughput. Fragmentation of latency sensitive packets can greatly increase latency; to avoid this, the maximum packet size required by the application determines a *minimum chunk size* for its traffic. Similarly, a latency sensitive application often generates packets at a predictable rate. If a chunk is scheduled too early for the packet it must service, the chunk is wasted and latency is increased; however, a late chunk increases latency as well. To capture this, we assign latency sensitive traffic a *period*. Thus, there are many latency sensitive service classes, depending on our choice of minimum chunk size and period.

Some combinations of latency sensitive service classes are impossible to satisfy simultaneously, and many choices of minimum chunk size and period may produce very inefficient layouts. Because of these restrictions, as well as concerns about the efficient implementation of the scheduling algorithm, we do not allow stations to request arbitrary latency sensitive service classes. Instead, administrators choose to allow specific classes in advance, and stations may choose how to classify their traffic from the service classes available. This design choice has another advantage: traffic with the same choice of minimum chunk size and period can be “gang scheduled” together, minimizing interference with other traffic as well as between traffic in the same class. In practice, we expect that most networks that are not running specialized applications will have use for two service classes: the bulk class for most traffic, and one latency sensitive class for VoIP traffic. Therefore, in the following discussion, we will focus on these two service classes, with the understanding that the system administrator may define more latency sensitive classes if needed.

3.2.4 Scheduling

Each packet arrives at the JaldiMAC scheduler already marked by a tool like *iptables* to indicate the station it is destined for and the service class it belongs to. This information is enough for JaldiMAC to determine the fair allocations for each station and begin the scheduling process. The scheduler creates a layout that gives each station the time it was allocated, enables implicit synchronization by ensuring that each upstream transmission is preceded by a downstream transmission, respects physical limitations like RX/TX switching speeds and guard bands, and maintains the requirements of each service class as closely as possible. Different approaches are taken for bulk traffic than for VoIP.

The scheduler does not analyze VoIP traffic in terms of individual packets; rather, its layout is computed in terms of active flows. Special *VoIP slots* are placed in the layout at regular intervals corresponding to the inter-packet arrival time of typical VoIP flows - in other words, to the period of the VoIP service class. Each VoIP slot has a size corresponding to the number of active upstream VoIP flows, up to an administrator-defined maximum. The flows are scheduled one right after the other within the VoIP slot, with small guard bands between them. These guard bands are not needed in the rest of the layout because of implicit synchronization, but VoIP slots are handled specially: the master announces each VoIP slot as it is reached, stating the list of stations that may transmit in order. Each station then transmits for a limited time after waiting for a delay calculated from its position in the list. This allows many VoIP flows to be handled in a short time, without requiring that the master switch from receive mode to transmit mode after each packet and notify the next station to transmit. Since latency sensitive service classes like VoIP generally send a very small amount of data at a time, the overhead of implicit synchronization would be quite high.

VoIP slots are effective at minimizing latency when the arrival time of each VoIP packet is unknown to the master; for upstream VoIP flows, (from stations to the master) this is always the case, and so the placement of VoIP slots must be essentially predictive. However, to keep latency to a minimum and add as little jitter as possible, it would be best if each VoIP packet was transmitted as soon as it arrived. For downstream VoIP flows, this is sometimes possible. Downstream VoIP packets are, in some sense, not scheduled at all; instead, the scheduler dynamically inserts them into the round. If they arrive during a downstream chunk, they are

inserted immediately after the current packet; this takes advantage of the point-to-multipoint nature of JaldiMAC, since every station can hear the master and therefore a VoIP packet for any station can be inserted in any other station’s downstream chunk. If they arrive during an upstream chunk, they are sent at the beginning of the next downstream chunk; implicit synchronization requires a downstream chunk between any two upstream chunks, so this will generally not incur significant delay. In some cases, due to bitrate differences between stations, the intended recipient may not be able to hear the VoIP packet during a particular downstream chunk; if this happens, the VoIP packet is held until it can be transmitted at an appropriate bitrate for the receiver.

Bulk traffic is scheduled in the space between VoIP slots. To minimize RX/TX switches, the scheduler tries to place as large a chunk as possible at all times, and tries to choose stations that can exhaust their entire allocation before the next VoIP slot. In JaldiMAC, there is no particular reason for upstream and downstream chunks for the same station to be scheduled together, so they are scheduled separately; this makes it easier to support implicit synchronization, by placing downstream chunks between upstream chunks, while also minimizing RX/TX switches. There is another concern which the scheduler has to address for bulk traffic: the minimum chunk size. Physical considerations such as the speed with which the radio hardware can switch between send and receive mode and the propagation delay between the master and stations result in a lower limit to minimum size of a chunk. The scheduler must take this into account when creating the layout.

The scheduler just described can be implemented using a simple, efficient algorithm. The fairness calculation requires state proportional to the number of sessions currently active, but the scheduler only requires state proportional to the number of stations. The most demanding requirement is queue size; the scheduler creates the next round’s layout during the contention slot, then waits until the following contention slot at the end of the subsequent round before processing any more bulk packets. (VoIP packets, as discussed above, are transmitted as soon as possible and do not wait for the next layout to be created.) This means that in the worst case, a packet may arrive in a bulk queue just after the contention slot for one round and not be transmitted until a chunk at the end of the following round, requiring a queue large enough to hold as many bulk packets as may arrive in two rounds. Assuming the scheduler maintains per-station input queues, the scheduler may mitigate this somewhat by scheduling chunks for stations with larger queues earlier in the round, but the queue size requirement is still significant. However, we do not view this as a limitation, since this is a problem shared by other TDMA implementations and we expect rounds to be relatively short most of the time.

4 Evaluation

We have implemented a sophisticated environment for automated experimentation, data collection and analysis. Utilizing a central server we dispatch traffic generation scripts on the 10 wired and 5 wireless embedded routers in our lab to run in parallel and collect the results. We configured the one of the wireless routers as the master (the “point” in a real world point-to-multipoint deployment) and the other 4 wireless routers as client stations. To emulate the hidden node problem experienced by clients with highly directional antennas in real-world long-distance point-to-

multipoint deployments, we utilized a combination of signal attenuators, RF isolation boxes around each station radio, and a directional antenna for the master. As a result, transmissions from each of the 4 wireless stations could only be heard by the master, and not each other, despite being collocated in the same room. In addition to collecting the reported output from the traffic generators, we also capture a complete packet dump of all traffic for each experiment, to provide for the case that we would need to further process or analyze the measurements in the future. Unfortunately, as we ran out of time before we could get a stable implementation of JaldiMAC working on the embedded routers, at this time we only present the measurement results that we find of interest for conventional WiFi, which we intend to compare with JaldiMAC once it becomes operational. While initially intended only as a base-line for comparison with JaldiMAC, we feel that our extensive measurements using conventional WiFi are of some interest, given the point-to-multipoint nature of the setup and especially the isolation between our client stations. This unique setup results in a perfect scenario of hidden-terminal that is difficult to create in a lab or to simulate. We also expose limitations of the embedded hardware that are often overlooked, such as their limited PPS (Packets Per Second) capacity. Unlike professional wired gear, these devices often lack specs for PPS but only report expected bandwidth capacity.

4.1 Lab Setup

For our testing and evaluation of JaldiMAC we have set up two lab environments each comprising multiple routers. The first is a wired-only lab consisting of five dual-interface routers. One interface on each connects to our master experiment server and the other connects directly to the other routers via an Ethernet hub, which allows experimentation with the TDMA scheduler in isolation. The wireless lab is similarly comprised of five wired routers, but rather than a hub these each then connect directly to one of the five wireless routers. With the four wireless station routers in the aforementioned isolation boxes and by utilizing attenuators and a directional antenna for the master, we thereby formed a unique radio environment in which the four stations cannot hear each other, yet all four can communicate with the master. Eliminating interference to and from external WiFi devices was also important and hence we have set up the lab in the basement of Soda Hall, using a room with a tin-plated ceiling and steel-reinforced walls as an RF isolation chamber. The wired routers are used to drive the wireless routers with various workloads and allow measurements similar to the wired-only lab. For all ten wired routers we have a back-channel Ethernet NIC for remote access and to synchronize the tests. In addition, we also have serial access for all of the wireless routers via an RS232 port to allow for flashing new firmware and kernels remotely. Finally, we use a smart PDU (Power Distribution Unit) to enable remote rebooting of individual components in case of a software hang, failure during the flashing process, or similar failure. We also began the process of setting up a long-distance outdoor testbed, but despite much effort we unfortunately failed to complete this task in time to utilize it in this paper.

A diagram of our indoor lab is provided in Appendix A.

4.2 Experimental Results

Over the course of the semester we ran hundreds of tests utilizing our indoor wireless testbed. Unfortunately, due to time and space limitations we present but two sets of them here.

4.2.1 Loss rate by number of stations.

Figure 4Figure 5 shows results from 36 of our experiments, in which 1 unidirectional stream per client is sent upstream from varying number of client stations (between 1 and 4) to the master, for various combinations of total injected network bandwidth (equally generated from the stations' streams) and packet sizes. Irregular

loss rates are observed when channel capacity is low, while a steady loss pattern is noticeable for high utilization that clearly increases number of stations.

4.2.2 Synchronization affecting jitter.

In Figure 5 we compare a proprietary router operating system called AirOS and the open source WiFi driver ATH9K. The key difference is the use of the simple RTS/CTS mechanism by AirOS to reduce collisions even among the hidden-to-each-other stations, which clearly improves overall jitter by lessening the need of 802.11's local retransmissions to recover packets lost by collision. We compare jitter for both traffic in the upstream direction from stations to the master and also the downstream direction from master to stations. The x-axis depicts a small subset of the many experiments we have run. "Serial" tests (the left half of the graph) involved running between variable upstream-downstream ratios of bandwidths and number of streams of various mix per client, with only 1 client active at a time, and averaging the results over all 4 clients. These traffic mixes represents various expected traffic patterns, such as bulk transfers, high volume streaming and also jitter sensitive traffic like VOIP. With only 1 active client there are no collisions for AirOS's employment of RTS/CTS to avoid (the master and client can both hear each other), and thus no clear advantage of AirOS versus ATH9K. However the "Parallel" tests (right half of Figure 4) illustrate a similar mix of per-client station traffic but with all 4 stations actively transmitting and receiving at once. Here the performance advantage of RTS/CTS is much more evident.

4.3 Driver performance evaluation

Our primary emphasis this semester was on implementing the Jaldi9k driver. Because we have not completely implemented Jaldi9k we are unable to present a thorough evaluation of its performance.. However, we were able to do a preliminary evaluation of our precision timing support in Jaldi9k. We supplied a frame to our driver every 100 milliseconds by means of a simple userlevel Click configuration; while much lower than the frame arrival rate we would expect to see in a normal usage scenario, we observed unpredictable frame arrival times to the kernel from userspace using shorter intervals. Rather than specifying a transmission time in the Click module from userspace, we defined the desired transmission time for each frame to be 5 milliseconds after it was received by our driver in order to isolate the effects of latency in userlevel. This interval is on the order of the period of our slots and thus provides an estimation of our ability to actually start slots at the proper time. We note that under real operation, this setup emulates the rate at which Jaldi9k will need to respond to TX interrupts: additional frames beyond the first in a slot are aggregated into a single burst and do not require their own timers.

The results of our preliminary profiling experiment can be seen in figure 3. In general, we found that the high resolution kernel timers we use are sufficient for achieving the precision event timing we desire. The median offset between desired and actual transmission time was 9.68 μ s, with a mean of 10.1 μ s and a

standard deviation of 1.38 μ s. While this result gives us some confidence that high resolution timers should give us the flexibility and accuracy Jaldi9k needs, we will need to re-evaluate this decision once we are able to properly test Jaldi9k under load. Another option if our current scheme proves insufficient would be to leverage beacon interrupt timers provided by the 802.11 hardware for slot timings. While this mechanism is not as conducive to the dynamic slot layout that JaldiMAC employs, it will likely serve as a much more accurate time source when the kernel is under load.

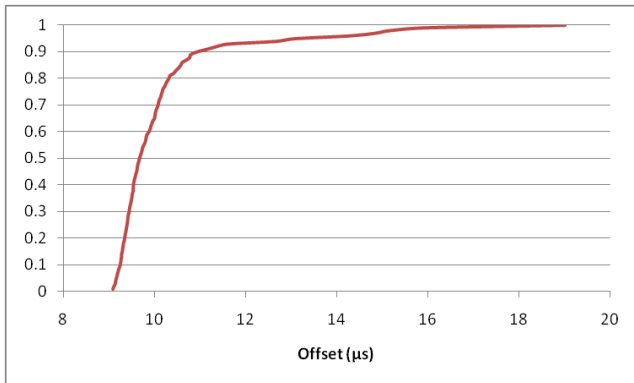


Figure 3. CDF of offsets between desired and actual transmission time.

5 Conclusion

We believe that WiFi is a cost effective and promising technology for extending broadband Internet connectivity to poor and sparsely populated communities. Our previous work [3] identified

costs associated with towers to be the primary barrier to entry for rural WISPs, offered a novel deployment methodology that takes advantage of mountains and hills along with highly directional antennas to maximize service area, and proposed the first version of a scheduler for JaldiTDMA, our long-distance TDMA MAC. In this work, we have completed an initial design for JaldiTDMA and an accompanying Linux kernel driver, Jaldi9K, which will allow us to implement JaldiMAC on real hardware. We have refined our architecture to be feasible given the hardware and cost constraints faced by rural WISPs. Additionally, we have built a testing environment to support future development of this project.

We have implemented and are currently testing JaldiTDMA using the Click Modular Router, and we have made significant progress in the implementation of Jaldi9K. Our initial tests have indicated that 802.11 does not provide adequate performance for rural WISPs under our deployment topology, even disregarding the effects of propagation delay that would be seen in long-distance links. We have made significant progress in refining our original JaldiMAC concept into a concrete form that can be deployed in the field, and we have developed a flexible system architecture that will facilitate the extensions to our initial implementation, such as manipulation of physical layer parameters to optimize channel utilization, that we have planned for the future.

Acknowledgments

We thank Matt Podolsky and Ian Davis, who took active part in this work and spent long nights in the office with the authors. We also thank Rabin Patra and Sergiu Nedevschi for the long brainstorming sessions and their guidance. The measurement data from their past experiments with long-distance links, along with the techniques they learned from implementing WiLDMAC, are invaluable. We also thank Anmol Sheth who joined our discussions over the phone and provided the draft for the yet unpublished paper [28].

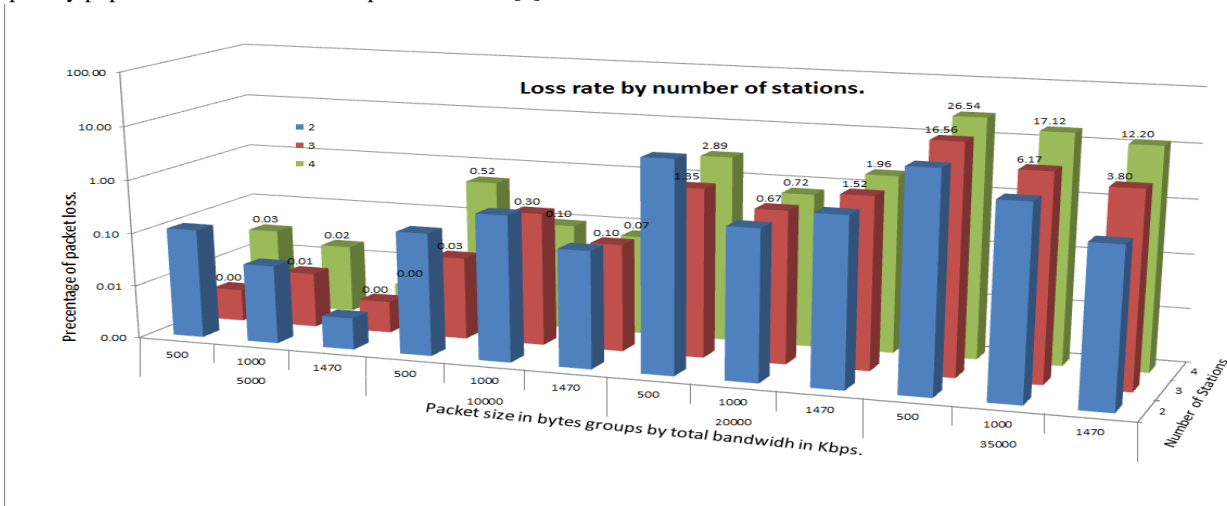


Figure 4. Unidirectional upstream loss rate (\log_{10} scale) versus number of stations with varied packet sizes grouped by total bandwidth.

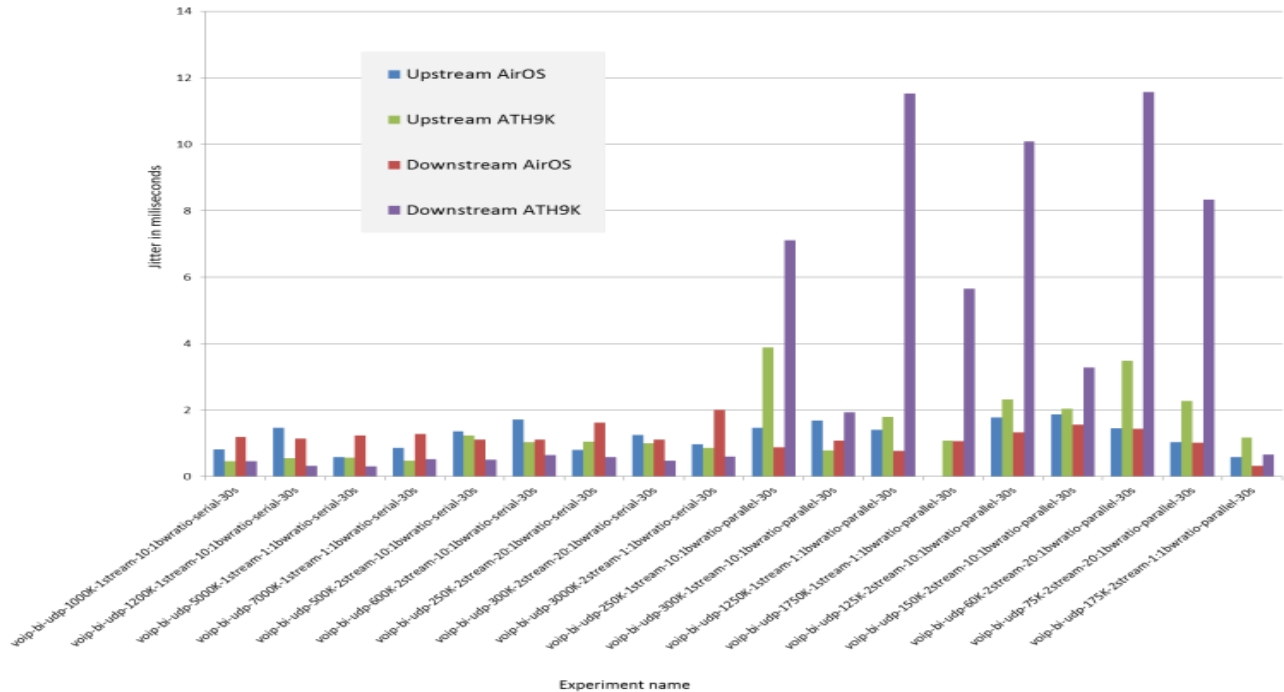


Figure 5. RTS/CTS dramatically reduces jitter.

References

- [1] The United Nations Population Database - World Urbanization Prospects: The 2007 Revision.
- [2] Yahel Ben-David, Matthias Vallentin, Seth Fowler, and Eric Brewer, "JaldiMAC: taking the distance further," , 2010, pp. 2:1--2:6.
- [3] Sung Park and Denh Sy, "Dynamic control slot scheduling algorithms for TDMA based Mobile Ad Hoc Networks," , 2008, pp. 1-7.
- [4] R Lenagala and Q Zeng, "Study of Dynamic MAC Layer Parameters for Starvation Prevention in the IEEE 802.11e MAC Layer Protocol," , 2006, pp. 124-131.
- [5] Sergiu Nedeveschi et al., "An Adaptive, High Performance MAC for Long-Distance Multihop Wireless Networks," , 2008.
- [6] Rabin Patra et al., "WiLDNet: Design and Implementation of High Performance WiFi Based Long Distance Networks," , 2007.
- [7] Krishna Paul, Anitha Varghese, and Sridhar Iyer, "WiFiRe: Rural Area Broadband Access Using the WiFi PHY and a Multisector TDD MAC," *New Directions in Networking Technologies in Emerging Economics, IEEE Communications Magazine*, 2006.
- [8] Michael Neufeld, Jeff Fifield, Christian Doerr, Anmol Sheth, and Dirk Grunwald, "SoftMAC - Flexible Wireless Research Platform," , 2005.
- [9] Ashish Sharma and Elizabeth M Belding, "FreeMAC: framework for multi-channel mac development on 802.11 hardware," , 2008, pp. 69--74.
- [10] Vaduvur Bharghavan, Alan Demers, Scott Shenker, and Lixia Zhang, "MACAW: a Media Access Protocol for Wireless LANs," , 1994, pp. 212--225.
- [11] Nirav, Bhaskaran Ashutosh, "Implementation and Evaluation of a TDMA MAC for WiFi-based Rural Mesh Networks," in *iSOSP Workshop on Networked Systems for Developing Regions (NSDR) 2009*, 2009, p. 6.
- [12] Bhaskaran Raman and Kameswari Chebrolu, "Design and Evaluation of a new MAC Protocol for Long-Distance 802.11 Mesh Networks," , 2005.
- [13] Ananth Rao and Ion Stoica, "An overlay MAC layer for 802.11 networks," , 2005, pp. 135--148.
- [14] Ashish Sharma, Mohit Tiwari, Haitao Zheng, and Santa U Barbara, "MadMAC: Building a Reconfigurable Radio Testbed Using Commodity 802.11 Hardware," , 2006.
- [15] Sonesh Surana et al., "Beyond pilots: keeping rural wireless networks alive," , 2008, pp. 119--132.
- [16] Freifunk Community Wireless Mesh Network, Berlin, Germany.
- [17] The AirJaldi Mesh Router, AirJaldi.Org, Dharamsala, India.
- [18] Shivanajay Marwaha, Jadwiga Indulska, and Marius Portmann, "Challenges and Recent Advances in QoS Provisioning, Signaling, Routing and MAC protocols for MANETs," in *Telecommunication Networks and Applications Conference, 2008. ATNAC 2008. Australasian,*

2008, pp. 97--102.

- [19] Tarana Wireless, Berkeley, CA.
- [20] Ramakrishna Gummadi, Rabin K Patra, Sergiu Nedeveschi, Sonesh Surana, and Eric A Brewer, "A radio multiplexing architecture for high throughput point to multipoint wireless networks.," , 2008, pp. 47-52.
- [21] Rabin Patra, Sonesh Surana, Sergiu Nedeveschi, and Eric Brewer, "Optimal Scheduling and Power Control for TDMA based Point to Multipoint Wireless Networks," , 2008.
- [22] Anmol Sheth et al., "Packet Loss Characterization in WiFi-based Long Distance Networks," , 2007.
- [23] Hari Balakrishnan, Srinivasan Seshan, Elan Amir, and Randy Katz, "Improving TCP/IP Performance over Wireless Networks," , 1995.
- [24] Hari Balakrishnan, Venkata N Padmanabhan, Srinivasan Seshan, and Randy H Katz, "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links," *IEEE/ACM Trans. Netw.*, vol. 5, pp. 756--769, 1997.
- [25] S Mascolo, C Casetti, M Gerla, MY Sanadidi, and R Wang, "TCP Westwood: Bandwidth estimation for enhanced transport over wireless links," , 2001, p. 297.
- [26] G Xylomenos, G C Polyzos, P Mahonen, and M Saaranen, "TCP performance issues over wireless links," *IEEE*

Communications Magazine, vol. 39, pp. 52--58, 2001.

- [27] M Vipin and S Srikanth, "Analysis of open source drivers for IEEE 802.11 WLANs," in *Wireless Communication and Sensor Computing, 2010. ICWCSC 2010. International Conference on*, 2010, pp. 1 -5.
- [28] Daniel Halperin, Wenjun Hu, Anmol Sheth, and David Wetherall, *802.11 with Multiple Antennas for Dummies*, 2010.
- [xx] [Radiotap]
- [30] AirJaldi.Org - Wireless Network, Dharamsala, India.
- [31] Akshaya E-Literacy Project.
- [32] Eric Brewer et al., "The Case for Technology in Developing Regions," *IEEE Computer*, vol. 38, pp. 25--38, 2005.
- [33] Sonesh Surana, Rabin Patra, Sergiu Nedeveschi, and Eric Brewer, "Deploying a Rural Wireless Telemedicine System: Experiences in Sustainability," *Computer*, vol. 41, pp. 48-56, 2008.

Appendix A

JaldiMAC Indoor Testbeds Network Diagram

Updated Oct. 12, 2010

