Cooperative Packet Scheduling via Pipelining in 802.11 Wireless Networks

Ramana Rao Kompella, Sriram Ramabhadran, Ishwar Ramani, Alex C. Snoeren
University of California, San Diego
La Jolla, CA 92093
{ramana,sriram,ishwar,snoeren}@cs.ucsd.edu

ABSTRACT

The proliferation of 802.11a/b/g based wireless devices has fueled their adoption in many domains – some of which are unforseen. Yet, these devices lack native support for some of the advanced features (such as service differentiation, etc.) required in specific application domains. A subset of these features relies on cooperative scheduling whereby nodes cooperate among each other to effectively manage resources such as power, throughput and interference in wireless networks. The trajectory of evolution in these devices has been primarily through new extension standards (such as 802.11e/s etc.) that offer support for these features. Plagued with long design cycles and cost overhead to upgrade, this process of upgrading creates an uphill task to users who want to use their wireless devices for different applications. In this paper, we argue that such cooperative scheduling extensions can be supported using a new layer on top of the existing MAC layer. We propose a $2\frac{1}{2}$-stage pipeline architecture as a generic mechanism to create such domain specific extensions and propose two such protocols, SPARTA (power conservation) and ARGOS (throughput guarantees) over the native 802.11/b/g MAC layer. Using a prototype we built over open source 802.11 wireless device driver, we present some preliminary evaluation of the architecture.

Categories and Subject Descriptors

C.2.6 [Computer-Communication Networks]: Network Architecture and Design – Wireless communication

General Terms

Algorithms, Design

Keywords

Cooperative scheduling, 802.11 wireless networks, Quality of Service, streaming video, proportional allocation, power conservation

1. INTRODUCTION

Wireless networks based on the 802.11 suite of protocols are increasingly becoming commonplace in different computing environments ranging from enterprises to homes. The total WLAN equipment market comprising of 802.11a/b/g is forecast to grow from 26 million devices shipped in 2003 to more than 160 million devices by 2006 [3]. These three protocols operate in exactly the same fashion, differing only in the supported set of transmission rates, channel codes, and frequency band. Together, these protocols are by far the most popular among the 802.11 family and, hence, form the *de facto* standard in wireless technology.

The explosive growth in 802.11-based wireless devices has resulted in their application to a rich set of scenarios—each with its own set of challenges and requirements. For example, wireless devices are increasingly finding place in home networks to support emerging streaming multimedia services as well as traditional broadband-access applications. In these scenarios, providing *throughput* guarantees to multimedia applications such as video-on-demand, VoIP, etc., is an important challenge that remains to be addressed. Another application of 802.11-based devices is in community networks and other mesh networks [11]. In these multi-hop situations, reducing the *interference* caused by individual nodes is a key challenge. In general, *power* is an extremely critical resource in wireless networks. This is true especially in ad hoc wireless networks or sensor networks, where the nodes are left unmanaged for extended periods of time.

Hence, the long-term success of 802.11-based wireless devices in varying application domains hinges on the development of novel mechanisms to manage three critical elements: throughput, interference and power. Effectively managing these three aspects requires *cooperation* from participating nodes. Service differentiation in wireless networks, for example, requires cooperation from nodes transmitting low-priority traffic to allow faster access by nodes transmitting high priority traffic. Reducing interference requires nodes to schedule medium accesses to avoid collisions so that channel can be utilized more efficiently. Similarly, many distributed power-saving mechanisms require cooperation among the participating wireless nodes [10]. In essence, each of these domain-specific mechanisms are instances of *cooperative scheduling*: this collaboration between wireless nodes to achieve a particular goal is the main subject of our paper.

One way to provide cooperative scheduling in different domains is to update the protocol at the firmware level. Indeed, such an approach has been taken by the emerging 802.11e standard to support service differentiation, 802.11s for mesh networks, and so on. However, designing new MAC protocols to support advanced features is both

- *non-trivial:* Every new protocol requires extensive design cycles, costs and a huge turn-around-time; and

- *non-exhaustive:* It is difficult to devise a protocol that can cater to all applications or anticipate future requirements.

Regardless, new protocols do not address the shortcomings of the existing installed base of legacy devices, leaving them unsuitable for the new application domains.

In this paper, we posit that, in order to fill this divide between existing technology and new requirements, cooperative scheduling functionality needs to be implemented as far as possible in *software.* The reason is that software upgrades are much faster to implement and can interoperate with the existing install base, resulting in a much cheaper evolutionary path. On the other hand, implementing MAC protocol features in software requires support from the operating system (such as fine-grained timers, low kernel scheduling overheads, etc.) and, hence, can potentially limit the granularity of these features. This particular philosophy of implementing features as far as possible in software has been proposed before in many different scenarios. For example, software defined radios [6] attempt to shift critical functionality performed by typical wireless devices into software leaving only bare minimum baseband processing—signal amplification and modulation—to the hardware. To the best of our knowledge, we are the first to apply this philosophy along with a concrete mechanism to perform cooperative scheduling over 802.11a/b/g based wireless networks.

We propose to implement cooperative scheduling protocol extensions using an *802.11 extension layer* interposed between the network layer and the 802.11 MAC layer in the wireless card device driver. This extension layer enables the implementation of many advanced medium access mechanisms that may be required in specific application domains. In this paper, we describe a novel $2\frac{1}{2}$-stage pipeline implemented in our 802.11 extension layer that supports the deployment of two different domain-specific cooperative schedulers. We briefly outline the implementation of SPARTA (to achieve power conservation) and ARGOS (to provide service differentiation) to demonstrate the broad applicability of our pipeline design.

2. ARCHITECTURE

In this section, we formalize the notion of cooperative scheduling in the 802.11 extension layer architecture. Fundamentally, cooperative scheduling involves three steps:

- *Estimation.* Each node must independently identify their own medium access requirements by estimating their current and future traffic demands.

- *Load exchange.* This local knowledge is propagated to each other via explicit or implicit mechanisms so each participating wireless node has global knowledge about the requirements of other nodes.

- *Scheduling.* Finally, using this global knowledge, each node can individually compute a global schedule and schedule packets accordingly.

At first glance, it might appear that cooperative scheduling can be best implemented at an access point (at least in wireless infrastructure networks). Indeed, the 802.11 PCF (Point Coordination Function) is a form of centralized scheduling that allows the access point to arbitrate the channel amongst participating nodes. However, there are many reasons why this might not be entirely desirable. First, access points are dedicated devices with specific features and it is not always easy to upgrade these access points. Second, the access points are not present in all scenarios such as multi-hop mesh networks or in ad hoc networks. Hence, our approach is to perform distributed scheduling using cooperation amongst participating nodes.

2.1 802.11 extension layer

The abstraction of cooperative scheduling is intertwined with the regular MAC layer functions as it shares the wireless medium between various nodes. Of course, in theory, cooperative scheduling can be performed at the application layer or through a generic interposition mechanism (using, say, TESLA [12], WRAPI [2]), as coordination packets themselves could be injected at any layer. That means, however, that each application would have to be re-written with an API that can support cooperative scheduling among applications. Besides, the MAC layer multiplexes packets from disparate applications before transmitting them on the wireless medium. This makes it the ideal layer to implement globally consistent scheduling disciplines amongst all participating wireless nodes.

One of our main objectives is to implement new scheduling algorithms (and other network functions that require scheduling) with minimal requirements on the device itself. Thus, the assumptions we make about the firmware of commodity 802.11 Wireless cards can be summarized as follows.

- Wireless cards support setting the transmit parameters of a card such as transmit rate, power, etc.

- Wireless cards implement the basic 802.11 DCF (Distributed Coordination Function) using CSMA/CA protocol.

- The MAC layer performs framing of the packets and implements the basic 802.11 state machine [7] (MAC level retransmissions, ACK generation etc.).

A typical wireless driver exports the following API for interaction with the network layer of the protocol stack: packet_tx() for transmission of a packet by the upper layer, packet_rx() for upper-layer packet reception, and, finally, stop_queue() and wake_queue() for transmit flow control.

Architecturally, our 802.11 extension layer is implemented as a sub-layer within the 802.11 MAC layer as shown in Figure 1. The packet transmit and receive calls from the upper layers to the MAC layer pass through the 802.11 extension layer where appropriate cooperative scheduling protocol (such as ARGOS or SPARTA) can be implemented. In order to perform scheduling, the 802.11 extension layer can optionally inject new control packets into the transmit path and receive control packets from other nodes through the receive path. Hence, both the packet_tx() and packet_rx() calls are trapped by the 802.11 extension layer to perform scheduling. The 802.11 extension layer also interacts with the upper layers to implement back-pressure mechanism to throttle the rate of packets sent by the upper layers.

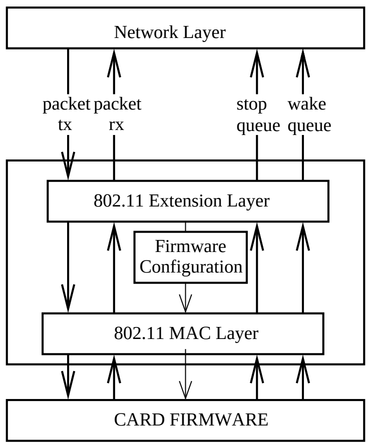

Figure 1: 802.11 extension layer architecture.

Usually, the device driver performs configuration of parameters such as rate of transmission, and transmit power on a per-packet basis taking into account the channel conditions such as signal strength. It normally selects the fastest rate code available subject to a maximum bit error rate and the minimum power required to transmit a packet at that rate. Given that scheduling is typically dependent on the rate and power of transmission, the 802.11 extension layer assumes control of these parameters. However, it controls these parameters while adhering to bounds identified by the device driver.

2.2 A $2\frac{1}{2}$-stage pipeline for cooperative scheduling

The three steps in cooperative scheduling (estimation, load exchange and scheduling) are implemented as a $2\frac{1}{2}$-stage pipeline. All participating nodes are time-synchronized (we discuss time synchronization in Section 4.1) to advance in a globally synchronized pipeline. Figure 2 shows the pipelined architecture for scheduling packets. In the *estimation* stage, each node estimates its traffic requirements using either explicit buffering (hence, delaying packets using a buffer) or through other history-based estimation methods (suitable for non-abrupt changes in traffic). This information is then communicated in the *load exchange* stage explicitly by injecting a "broadcast" packet into the transmit path. In the *scheduling* stage, each node has obtained knowledge about each node's requirement in order to arrive at a global schedule according to that particular cooperative scheduling policy.

The three stages of the pipeline are equal in duration. However, as can be seen from Figure 2, the first half of the load exchange stage overlaps with the estimation stage. The purpose of the load exchange stage is to exchange state information among participating nodes. Thus, a node broadcasts its load exchange packet at the end of the estimation stage. However, due to queuing delays and imperfect time synchronization, it may receive load exchange packets from other nodes, either sometime earlier or later than this. Therefore, the load exchange phase actually starts midway through the estimation phase and continues till the start of the scheduling stage. Due to the overlap with the estimation stage, the load exchange stage contributes only a *half* a stage to

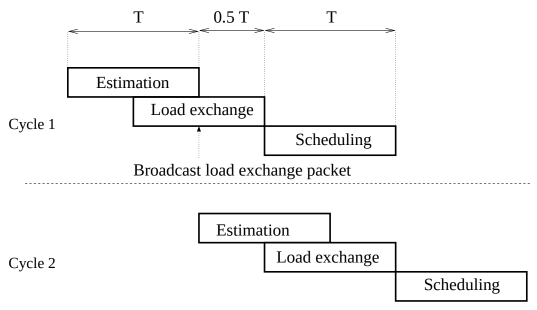

Figure 2: $2\frac{1}{2}$-stage pipeline architecture. Each pipeline stage is equal in duration, but estimation stage and load exchange stage for a given cycle overlap with each other.

the total latency of the pipeline; hence the name $2\frac{1}{2}$-stage pipeline. We note that the maximum delay experienced by a packet that arrives at the beginning of the estimation stage (if estimation is done using buffering) is $1\frac{1}{2}$ stages.

2.3 Prototype implementation

We have implemented the extension layer on a Linux platform running 2.4.28 kernel. For the 802.11 interface, we chose to use the Netgear WAG511 a/b/g card using the Atheros chipset. This chipset is well supported by a popular open source driver (madwifi [4]). The madwifi driver has been modified to make the necessary calls to the extension layer. As shown in Figure 1, the packet send and receive calls and the queue management functions were modified to be routed through the extension layer. This minimal modification makes the extension layer code portable across different device drivers.

We implemented the $2\frac{1}{2}$-stage pipeline (shown in Figure 2) using Linux kernel timers. The implementation consists of a core handler that can both schedule as well as process various events. Upon a packet transmit call from the upper layer, the core handler buffers the packet and activates the pipeline (if inactive) by scheduling two timers – estimation timer and load exchange timer that signify the completion of estimation stage and load exchange stage respectively. When the estimation timer expires, control returns back to the core handler where it performs two actions; first, it broadcasts a load exchange packet containing information about its own load and second, if there are packets to send this cycle, it keeps the pipeline active by scheduling both these timers again, otherwise stops the pipeline. In the receive path, load exchange packets broadcast from other nodes in the radio range are processed by the core handler. On the expiry of load exchange timer, the pipeline enters the scheduling stage and the core handler computes a global schedule based on load exchange packets received so far, and schedules packets accordingly.

3. APPLICATIONS

The goal of our 802.11 extension layer is to implement many different applications easily in the $2\frac{1}{2}$-stage pipelined architecture. In this section, we discuss how two different applications, SPARTA and ARGOS, can be implemented in this architecture.

3.1 SPARTA

In many ad hoc and sensor networks, power is the most critical resource for wireless devices as they are usually left unmanaged for extended periods of time. In many of these devices, transmit power is a significant fraction of the total power consumption of the device. In such devices, the energy required to transmit a packet increases exponentially with the rate of transmission. This observation can be exploited to conserve power by scheduling packet transmissions at the lowest possible rate [15]. We previously proposed SPARTA, a distributed MAC protocol that harnesses this tradeoff [10]. SPARTA uses a pipeline architecture similar to our $2\frac{1}{2}$-stage pipeline to first estimate the rate at which a node is generating traffic by buffering packets and then scheduling packet transmissions at an appropriate rate.

SPARTA's basic online distributed algorithm has one serious implementation challenge: the number of packets buffered in a pipeline stage is inferred by other nodes based on the transmission duration. However, such a scheme can suffer from inaccuracy when there are only a small set of discrete bandwidth levels available, as is the case in many 802.11b-based networks. Besides, SPARTA requires setting the cards in promiscuous mode to obtain all the packets on the wireless medium. Finally, the algorithm itself takes atleast n packets to be transmitted in order for the total rate to converge (n is the number of participating nodes). Using our $2\frac{1}{2}$-stage pipeline, however, we can share the load information by explicit transmit messages that increases the accuracy of the estimates (of course leading to higher overheads due to explicit broadcast packets) and does not require the cards to be configured in promiscuous mode.

3.2 ARGOS

To provide service differentiation among various applications, we propose a protocol called ARGOS that can be implemented using our $2\frac{1}{2}$-stage pipeline architecture. ARGOS provides proportional fair queuing between competing wireless nodes. For example, a wireless Video on Demand streaming server can be provisioned to receive twice the bandwidth that a laptop user receives. Other applications, like proportional fair queuing between types of traffic, are equally straightforward.

ARGOS works as follows. In the load exchange stage, each node broadcasts it weight and its current backlog of packets. This, along with an estimate of available bandwidth, enables each node to compute its *fair share*, i.e., how many bytes it is eligible to send in the corresponding scheduling interval. Available bandwidth is dynamically estimated in order to account for variations in channel capacity due to, for example, noisy environments and non-compliant nodes. This is done using a simple backpressure mechanism – feedback from the hardware layer as to how many packets it was actually able to transmit over the air in the previous scheduling interval. In the absence of backpressure, the estimate of available bandwidth is increased, and conversely, in the presence of backpressure, it is decreased. To ensure a consistent system-wide estimate of available bandwidth, nodes also exchange their estimates during the load exchange stage and use a average value to compute their fair shares.

We present some preliminary results using our implementation of ARGOS over the 802.11 extension layer. In our implementation, we provisioned the wireless video-on-demand streaming server (*gold* node) to receive twice the bandwidth

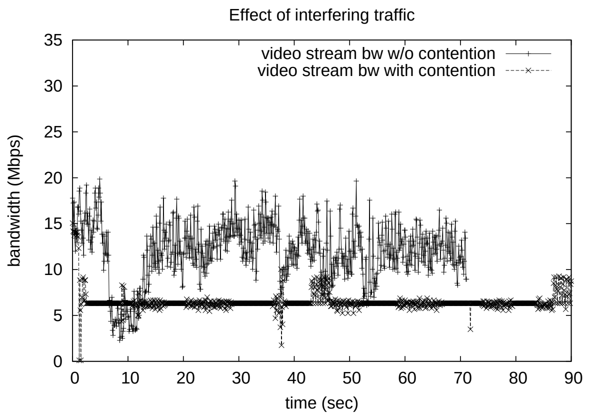

Figure 3: Video LAN experiment: streaming a VBR (Variable Bit Rate) video clip through the wireless medium with and without the presence of interference from another node. Note that for ease of comparison, the figure only shows the first 90 seconds of the experiment although in presence of contention, the video stream took 140 seconds to complete.

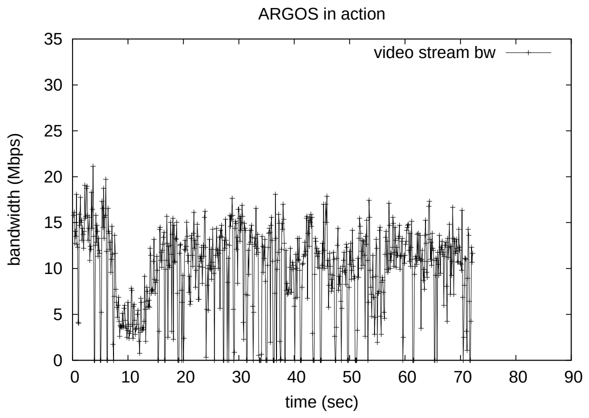

Figure 4: Bandwidth received by the wireless video streaming node with ARGOS.

in comparison to another node (*silver* node) running non-real-time applications. We allowed the silver node to generate UDP traffic as fast as it can by continuously performing application writes into the socket. Of course, if there is congestion in the network (at the available bandwidth point), the application writes into the socket would not go through (or block).

In the first experiment, the gold node streamed a variable bit rate encoded video stream onto a wired receiver both with and without an interfering agent. Figure 3 shows the aggregated bandwidth (measured as number of bytes transfered in a unit time) with and without this competing interferer. In the absence of any other traffic, the gold node received all the bandwidth and finished streaming of the video clip in about 70 seconds (the duration of the actual video clip). However, in the presence of the interfering agent, the video streaming node received only part of bandwidth. In this case, the video clip took almost 140 seconds to stream, leading to poor video quality at the receiver.

In the second experiment, we provisioned the video stream to receive twice the bandwidth in comparison with an interferer, which also implements ARGOS. The total channel bandwidth is about 26 Mbps in the 802.11a based network when operating in the 54 Mbps mode. Figure 4 shows the amount of bandwidth received by the video stream. We observed that the throughput received by the video stream is virtually identical to what it received in the absence of any interference. In addition, the total bandwidth of the video stream and the interferer was observed to be close to the channel capacity of 26 Mbps. This shows how ARGOS over our 802.11 extension layer can be provisioned to isolate real-time traffic flows such as video streams without impacting the total throughput of the channel.

4. DISCUSSION

In this section, we briefly review implementation challenges including system tradeoffs for our $2\frac{1}{2}$-stage pipeline design.

4.1 Implementation issues

Load exchange overhead. Explicit propagation of per-node traffic estimates using a control channel overlaid on the data channel causes interesting tradeoffs. The more frequent the exchange of information, the more the amount of channel bandwidth that would be used up for explicit coordination. Clearly, the overhead associated with coordination grows with the total number of nodes. However, the less frequent the exchange of information the less optimal the scheduling strategy can be. Thus depending on the application, the frequency of the load exchange needs to be chosen. For example, if the load exchange packet consists of about 200 bytes of data and is transmitted periodically every 10ms, then amount of bandwidth usage of this coordination stage is 0.016 Mbps per node. If there are many nodes in the system, coordination packets can occupy a non-trivial fraction of the bandwidth thus reducing overall throughput. In addition, these load exchange packets consume energy when transmitted. Thus, care must be taken to use explicit load exchange only when the total power savings outweigh the cost of these messages.

Time synchronization. In infrastructure networks, an access point transmits periodic beacon packets to allow mobile clients to sustain associativity. These beacon packets are typically transmitted every 100 ms and can aid in time synchronization. We divide the inter-beacon period into fixed intervals so that each node can advance in the pipeline with respect to these beacons. No special care needs to be taken to ensure beacon packets are always received by all the nodes as the nodes once synchronized can continue to operate synchronized and periodically calibrate with the beacon packets. In ad hoc networks, time synchronization is a bigger issue as there is no access point and hence no beacon packets. However, protocols such as NTP or RBS [5] can be run to synchronize the clocks across nodes. Such an overhead, of course, is not required in the infrastructure networks.

Delays and losses in load exchange packets. There can be losses and delays in these load exchange packets during periods of congestion that can affect the accuracy of a schedule in a given cycle. The advantage of explicit cooperation in the load exchange stage is the ability to self-correct these losses, as the history is completely lost and every pipeline cycle begins afresh with explicit coordination. In an experiment to

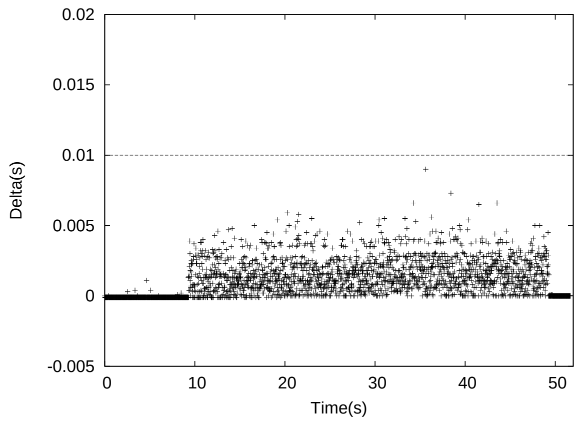

Figure 5: Difference between the expected time and arrived time (delta) of the load exchange packets as observed at a sniffer.

evaluate the effect of interference on the delay experienced by load exchange packets, we configured a wireless node to generate packets as fast as possible to occupy all available bandwidth in the channel. An interval size of 20ms has been chosen for the estimation and scheduling stages of the $2\frac{1}{2}$-stage pipeline. Two other interfering nodes have also been configured to generate traffic as fast as possible to compete with this node. In Figure 5, we plot the difference between the expected time and arrived time (delta) of the load exchange packets as observed at a sniffer. From the Figure, we can observe that all the packets have a delta much smaller than the load exchange stage interval of 10ms (shown as a line in the Figure) in spite of heavy congestion in the network. In fact, almost all the load exchange packets arrived within a delta of 5ms. It can also be observed that there is all deltas are non-negative, as congestion can only load-exchange packets to get delayed. Note that all packets were maximum sized frames (1400 bytes) and we used 802.11b network for these results. In effect, this is one of the worst possible scenarios, as large frame size and lower effective bandwidth (in 802.11b as compared to 802.11a) can cause maximum delay to the load exchange packets.

4.2 Application layer issues

Effect of buffering on TCP. Clearly, buffering can affect the perceived RTT experienced by TCP. As throughput is inversely proportional to RTT, therefore effective TCP throughput can be affected. While this is true, it might not immediately translate into human-perceivable delays for a lot of applications. However, if such a delay is indeed problematic, estimation can be performed passively by observing the rate at which packets are being generated using history. The second option is to reduce the pipeline stage to a smaller value thus trading off extra load propagation overheads.

Effect of buffering on real-time applications. Buffering does not cause any problem to streaming video/audio services as these applications usually have buffers to reduce the effect of jitter. Buffering on the other hand, can affect interactive real-time applications such as VoIP. However, the traffic characteristics of VoIP [13] have been found to be almost constant bit rates—thereby eliminating the need for explicit bandwidth estimation by buffering. The pipeline works the same for other applications that can be buffered, but during the load exchange stage, estimated load (based on history) for VoIP traffic and calculated load for other applications can be transmitted to perform scheduling.

5. RELATED WORK

Currently, there are few mechanisms that can provide throughput guarantees over existing 802.11a/b/g based wireless devices. Our pipelined architecture enables this functionality by arbitrating the wireless channel among competing applications. Most of the approaches suggested in literature [8, 16, 1] revolve around providing faster access to the channel by modifying the DIFS (Distributed Inter Frame Spacing) for higher priority packets thus providing statistical throughput guarantees. Unfortunately, however, these approaches require modifications at the firmware level. Our cooperative scheduling pipeline inherently provides "macro" scheduling (scheduling in sets of packets) as opposed to these that implement micro-scheduling. Micro-scheduling requires 802.11 protocol changes that are not feasible unless through an upgrade to new firmware. Typical scenarios, however do not require per-packet scheduling and hence can be sustained using our mechanism.

In [14], the authors propose the usage of a Time-Based Regulator (TBR) to provide fairness in channel access among contending nodes with different Transmit Rates. The TBR protocol works at the AP and uses a polling mechanism similar to PCF in arbitrating between various nodes to ensure high bandwidth nodes get a fairer share of the air time rather than channel access. In our cooperative scheduling framework, we can implement the same air-time scheduling in a distributed fashion by allowing nodes to periodically exchange information about the queue occupancies.

Of course, not all protocols that try to optimize power and throughput can be handled by our cooperative scheduling pipeline. For example, COMPOW [9] is a power control algorithm that tries to identify an optimal transmission power across all nodes in multi-hop wireless networks. This protocol operates at the network layer as it requires the entire topology of the multi-hop wireless network in order to determine an optimal transmission power.

6. CONCLUSION

Wireless networks based on 802.11a/b/g technology have enjoyed tremendous success in terms of their penetration into various application domains. However, they face tremendous challenges in supporting application specific features such as service differentiation—a lot of which are currently being addressed by new MAC protocol standards. In this paper, we identify a cooperative scheduling as a core service required by these extensions and described a mechanism that can allow for the seamless deployment of new coordinated scheduling protocols in the current install base without resorting to expensive and time consuming hardware upgrades. Using our pipeline architecture, we implemented scheduling in two entirely different application domains to achieve two different goals—power conservation (SPARTA) and service differentiation (ARGOS). We also outlined some of the implementation challenges and how our pipeline architecture can impact the performance of the upper layers while satisfying the goals of the particular domain they can be applied.

Acknowledgements

We thank Stefan Savage and Geoff Voelker for their valuable feedback and beneficial discussions. We also thank the anonymous reviewers for their comments. This work was made possible by NSF Grant ANI 0074004.

7. REFERENCES

[1] IEEE Draft Standard 802.11e. Wireless medium access control (MAC) enhancements for quality of service(QoS). Standard 802.11e, 2001.

[2] Wireless Research API. http://ramp.ucsd.edu/pawn/wrapi.

[3] Forward Concepts. http://www.analogzone.com/netp1020b.htm.

[4] Sourceforge MadWifi Driver. http://www.sourceforge.net/madwifi.

[5] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-grained network time synchronization using reference broadcasts. In *Proc. 5th USENIX Symposium on Operating Systems Design and Implementation*, pages 147–163, Boston, Massachusetts, December 2002.

[6] Software Defined Radio Forum. http://www.sdrforum.org.

[7] Institute of Electronic and Electrical Engineers (IEEE). Wireless medium access control (MAC) and physical layer (PHY) specifications. Standard 802.11, 1999.

[8] V. Kanodia, C. Li, A. Sabharwal, B. Sadeghi, and E. Knightly. Distributed multi-hop scheduling with delay and throughput constraints. In *In Proceedings of ACM MOBICOM*, July 2001.

[9] Vikas Kawadia and P. R. Kumar. Power control and clustering in ad hoc networks. In *Proc. IEEE Infocom*, San Francisco, California, April 2003.

[10] Ramana Rao Kompella and Alex Snoeren. Practical lazy scheduling in wireless sensor networks. In *In Proceedings of ACM Sensys*, November 2003.

[11] MIT RoofNet. http://www.pdos.lcs.mit.edu/roofnet/.

[12] Jon Salz, Alex C. Snoeren, and Hari Balakrishnan. TESLA: A transparent, extensible session-layer framework for end-to-end network services. In *Proc. 4th USENIX Symposium on Internet Technologies and Systems*, Seattle, Washington, March 2003.

[13] Sanaa Sharafeddine, Anton Riedl, Josef Glasmann, and Jaargen Totzke. On traffic characteristics and bandwidth requirements of Voice over IP applications. In *Proceedings of ISCC*, 2003.

[14] Godfrey Tan and John Guttag. Time-based fairness improves performance in multi-rate wlans. In *USENIX Annual Technical Conference*, June 2004.

[15] Elif Uysal-Biyikoglu, Balaji Prabhakar, and Abbas El Gamal. Energy-efficient packet transmission over a wireless link. *ACM/IEEE Transactions on Networking*, 10(4):487–499, August 2002.

[16] N. Vaidya, P. Bahl, and S. Gupta. Distributed fair scheduling in a wireless lan. In *ACM MOBICOM*, August 2000.