

Efficient Cooperative Scheduling in 802.11 Wireless Networks

Ishwar Ramani, Ramana Rao Kompella, Sriram Ramabhadran, Alex C. Snoeren

University of California, San Diego

La Jolla, CA 92093

{ishwar,ramana,sriram,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 were unforeseen. Yet, these devices lack native support for many advanced features (such as service differentiation, etc.) required in specific application domains. A subset of these features relies on cooperative scheduling whereby nodes communicate among themselves 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 (e.g., 802.11e/s, etc.) that offer support for these features. Plagued with long design cycles and significant cost overheads, this upgrade process creates an uphill battle for users who want to use their wireless devices for new applications that require inter-node coordination. In this paper, we argue that 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 domain-specific extensions. Using a prototype we built over an open-source 802.11 wireless device driver, we evaluate the architecture in a case study.

I. INTRODUCTION

Wireless networks based on the 802.11 suite of protocols are becoming a very popular and common means of networking in both enterprise and home environments. The total WLAN equipment market, comprising of 802.11a/b/g standards, is forecast to grow from 26 million devices (shipped in 2003) to more than 160 million devices by 2006 [1]. The success of 802.11-based wireless networks can in large part be ascribed to inexpensive, readily available equipment and ease of deployment. This explosive growth has resulted in applications to a variety of domains—each with its own set of challenges and requirements.

For example, wireless devices are increasingly being deployed in home multimedia networks [2] to support emerging streaming 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. Power conservation is yet another feature critical in making 802.11 a viable solution in scenarios involving long-term battery-powered operation.

The 802.11 protocol was designed to create a high-speed data network with shared access to the wireless spectrum and some basic guarantees on packet delivery. These new throughput and power requirements are not fully addressed in the original standard. To solve these problems, many new proposals like 802.11e [3] (QoS enhancements) and 802.11r [4]

(fast roaming) are in the standardization pipeline. These additions, each arguably successful in accomplishing their goals, are a reactive approach to solving emerging challenges. Unfortunately, they remain several years from pervasive, low-cost product availability. Moreover, these upgrades usually mandate firmware updates and protocol changes. In short, the current, standardized MAC-layer extensions are:

- *non-trivial*: Every new protocol requires extensive design cycles, investments 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 MAC protocols do not address the shortcomings of the existing installed base of legacy devices, leaving them unsuitable for the new application domains. We present an alternative, software-based solution to addressing some of the problems.

A software-based solution is independent of the 802.11 hardware, enabling it to be widely deployable, even having the same influence of a protocol modification. The disadvantage of using a software-based solution, however, is the lack of dedicated resources like hardware timers and processing power. For example, 802.11 uses precise timing information available in the hardware for implementing the DCF algorithm. A similar solution would be difficult to achieve in software. A software solution, therefore, has to rely on 802.11 DCF protocol for fine-grained scheduling, performing only macro-level scheduling. To offset this deficiency, each participating node can be made aware of the global demands (data requirements, transmission rates) of the shared resource (capacity, channel). This knowledge can be used to make optimal decisions for enforcing new policies over 802.11. For example, if nodes are aware of the amount of data to be transmitted at other nodes, they can share the channel using byte-level fairness. This overcomes the inherent unfairness in 802.11 caused by packet diversity [5]. A critical design choice can be made at this stage: To gain global knowledge nodes can either passively monitor the medium [6] or exchange explicit information about individual states. The latter has the advantage of providing an accurate picture of the network while using resources while the former has the potential to be less invasive, but can be unreliable in practice.

In [7] we proposed a scheduling policy called Covenant that meets all the design goals. It is software based and uses explicit exchange of information to extend the capabilities of

802.11. In essence, Covenant acts as an extension layer to the 802.11 mac layer and uses information exchange to cooperate for enforcing policies. To summarize the features of Covenant:

- Software based: Covenant is implemented in software and works with any 802.11 card. Hence it can be implemented on legacy 802.11 devices to provide new features.
- $2\frac{1}{2}$ -stage pipeline: Covenant uses a novel $2\frac{1}{2}$ -stage pipeline to coordinate among nodes. This pipeline is designed to provide efficiency and robustness to the mechanism.
- Interoperable: Using Covenant in a milieu of cards with regular drivers will only degrade Covenant nodes to regular 802.11 nodes. It does not affect performance of regular nodes.
- Broad applicability: Covenant can be used to provide many new functionalities to 802.11 like QoS guarantees, power conservation etc.
- Implemented prototype: We have a proof of concept prototype of Covenant successfully implemented and functional.

In this paper, we propose to evaluate Covenant further and in the process answer a number of important questions. For example, how does Covenant perform in an actual 802.11 network? How does this mechanism affect the behaviour of TCP and UDP traffic? How robust is the mechanism to external and internal load? We also try to analyze the benefits and performance of Covenant in a case study. We propose to use Covenant to make 802.11, a more attractive and better solution for wireless home networking.

The rest of the paper is split into two parts. In the first part, we analyze the mechanism, its performance and effect on higher layers. In the second part, we address problems of using 802.11 in wireless home networking using Covenant. The next section summarizes the design, architecture and implementation details of Covenant. In section 3 we study the performance of Covenant. Section 4 introduces our case study and problems that Covenant can address with experiments. In section 5 we show results from experiments using Covenant followed by conclusion. Discussion and future goals are in section 6 followed by conclusion.

II. COVENANT

In this section we briefly review the design and architecture of Covenant described in [7].

A. Design

The basic design of Covenant involves three stages:

- *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 mechanisms. Hence, each node is aware of the global state of the system and the demand on resources for the next window of time.
- *Scheduling*. Finally, using this global knowledge, each node can individually compute a global schedule and

transmit packets according to policy constraints. Due to explicit sharing of information, local decisions are consistent across all nodes. We point out here that this scheduling complements the underlying 802.11 scheduling policies (like DCF) to extend its functionality. Thus, on a fine-grained level, the packets are still scheduled using 802.11 since its best designed for transmission in a shared noisy medium.

These three steps are implemented in a $2\frac{1}{2}$ -stage pipeline as shown in Figure 1. In the *estimation* stage, each node estimates its traffic requirements for the corresponding scheduling stage using explicit buffering. This information is then communicated in the *load exchange* stage explicitly by injecting a “broadcast” packet into the transmit path. This broadcast packet consists information about the state on that particular node including details about queue occupancies, rate and other such parameters. In the *scheduling* stage, each node computes a schedule based on a globally consistent scheduling policy using the knowledge obtained through the load exchange packets.

Each of these stages is for one epoch and they have the same epoch time period. However, as can be seen from Figure 1, 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 *half* a stage to the total latency of the pipeline; hence the name $2\frac{1}{2}$ -stage pipeline.

One way to implement the load estimation stage is through active buffering of packets that arrive during this stage. This gives an accurate estimate of the number of packets that need to be scheduled during the scheduling stage of a given cycle. Buffering however, increases the latency experienced by individual packets as packets wait in the buffer awaiting their turn to be transmitted on the wireless medium. This increase in the latency can potentially affect both TCP and UDP traffic. We note that the maximum delay experienced by a packet that arrives at the beginning of the estimation stage is $1\frac{1}{2}$ epoch and given constant arrival rate, the average delay would be *one* epoch. This effect can be alleviated by using passive estimation techniques that avoid buffering such as exponentially weighted moving average (EWMA). In such a mechanism, packets are no longer buffered. If the EWMA prediction is right, then the expected set of packets directly arrive in the scheduling stage and are scheduled. Of course, buffering some packets might still be required if the appropriate scheduling discipline grants lesser bandwidth than the requirement of the node. The downside to this approach is that it may lead to inaccurate representation of system load if the traffic is not easily predictable. These issues are analyzed in detail in Section III.

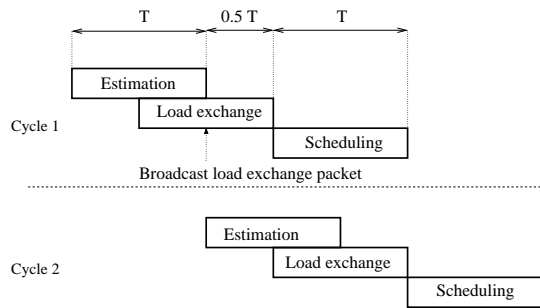


Fig. 1. $2\frac{1}{2}$ -stage pipeline architecture. Each pipeline stage is equal in duration, but estimation stage and load exchange stage for a given cycle overlap with each other.

Next important stage is the load exchange stage. In this stage, local knowledge is exchanged via explicit messages among participating nodes to obtain global knowledge. In order to obtain accurate global knowledge, the load exchange packets from all participating nodes have to be received during this stage. Time synchronization therefore, plays a crucial role in the accuracy of this stage. We solve this problem by borrowing from 802.11 design which faces a similar problem of synchronization for DCF. This is achieved by using the beacon packets which are generated by the access points periodically. Lack of external clocks like access points can be made up by using the load exchange packets from one of the participating nodes to synchronize like [8].

Finally, in the scheduling stage, packets are scheduled according to the global knowledge obtained via explicit load exchange messages and a pre-configured globally consistent scheduling policy. Packets that are not successfully transmitted in the scheduling stage are factored in the next cycle.

The main benefits of this design are efficiency and robustness. Since we implement the mechanism in a pipeline, for every estimation and load exchange state there is an overlapping scheduling stage transmitting packets. The node is never in an estimation stage without sending any packets, except of course, when the pipeline begins. Robustness is achieved due to the predominantly stateless nature of the pipeline. Each cycle operates with fresh dissemination of individual load estimates and global knowledge computed through these messages. Since the amount of persistent state in each of the nodes is minimal, it prevents inaccuracies from building up over time. Thus, the design is tolerant to occasional loss in load exchange messages or other control packets.

B. Implementation

Architecturally, Covenant is implemented as a sub-layer within the 802.11 MAC layer as shown in Figure 2 [7]. 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 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.

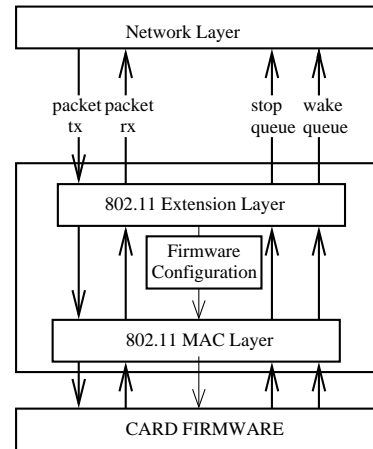


Fig. 2. 802.11 extension layer architecture.

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 [9]). The madwifi driver has been modified to make the necessary calls to the extension layer. As shown in Figure 2, 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 1) using linux kernel timers. The accuracy of the pipeline is dependent on the accuracy of this timer which is controlled by the real time clock in the kernel. We used a frequency of 1HZ for this clock which gives us 1ms accuracy. 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. The scheduling stage is implemented as a kernel thread whose flow is controlled appropriately by the core handler at the expiry of load exchange timer. This was done to provide a degree of freedom for implementing different scheduling policies and supporting back pressure control. This thread computes a global schedule based on load exchange packets received so far, and schedules packets accordingly.

The load exchange packets are constructed using a 802.3 packet format, with an unused protocol value. The payload of this packet contains basic information like node id, packet

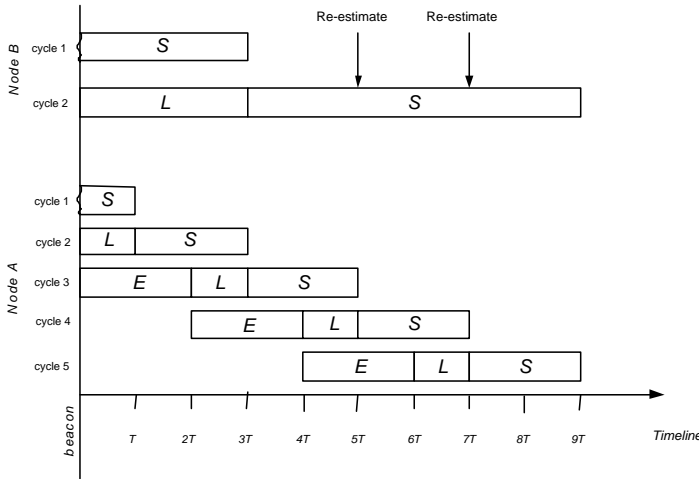


Fig. 3. Timeline of pipeline stages. S stands for scheduling stage, L stands for load exchange and E stands for estimation stage. As figure show cycle 1 in both nodes are cut half way to show the timeline starting from beacon arrival

sequence number, node transmission rate along with information pertinent to the scheduling policy. We also provide some feedback on the results of policy in previous epoch. This packet is sent to the 802.11 layer for transmission to other nodes.

C. Different Epoch Periods

Another degree of freedom offered in the architecture is to operate with different epoch values at different nodes. At first glance, it might appear counter-intuitive that such deployment offers any benefits. Epoch value closely controls the amount of throughput loss and also the delay experienced by packets that belong to a particular application. While having a lower epoch value clearly reduces the effect on delay, it increases the overhead. So, in order to reduce the overhead, certain nodes whose performance is not affected by large epoch periods can be run with higher epoch periods, so that the frequency of load exchange packets is reduced. This can bring down the amount of overhead for the control channel.

To achieve this, the Covenant architecture can be modified to have different participating nodes operating with different epochs. But at the same time constraints of synchrony have to be met. For example, a video streaming device can have an epoch value of 3τ , while a real-time device can have a value of τ . To guarantee that they have the same global estimate, the start of every scheduling stage should be synchronized and the nodes having a longer epoch have to re-estimate at the end of every smaller epoch. Since both the nodes should also synchronize with the beacons, both 3τ and τ should be a factor of beacon_interval.

This is better illustrated in Figure 3. Node B is running at three times the epoch of Node A. Since both the nodes are synchronized with the beacon, their load exchange phase starts at the same time. As seen from the figure, the schedule stage of Node A in cycle 3 starts at the same time as the schedule stage of Node B, cycle two. Hence at this time ($3T$) both the nodes are aware of the load estimate for time $3T$ to

$5T$ and they can schedule accordingly. Node B also captures the lodex packets generated by Node A at $4T$ and $6T$. At the end of the load exchange stage, Node B, re-estimates schedule at $5T$ and $6T$ (at the end of Node A's load exchange states). Node B has the benefit of making better policy decision since it has a bigger window of estimation. While Node A, saves on buffering delay by running on shorter epochs. Moreover load exchange stage of Node B can receive multiple lodex packets from A. To avoid confusion, the lodex packets will also contain its epoch value and the epoch_val its running on. This information can be used to figure out if the lodex packet is for the next scheduling stage or the current scheduling stage. For example, the L stage for Node B in cycle 2, should only use the lodex packets from L stage in cycle 3 for Node A.

In general, let the shortest epoch time used in the system be 2τ , then for another node to have a different epoch τ_1 ,

$$\tau_1 = \tau + 2\alpha\tau$$

where α is an integer value. Another constraint on the system is the beacon synchronization,

$$\begin{aligned} 2\tau\lambda_1 &= \text{beacon_interval} \\ (1 + 2\alpha)\tau\lambda_2 &= \text{beacon_interval} \\ \alpha &= \frac{\lambda_1}{2\lambda_2} - \frac{1}{2} \end{aligned}$$

Hence for α to be an integer $\frac{\lambda_1}{\lambda_2}$ should be an odd number. Thus for example, a VoIP client can set the epoch value at around 6ms (beacon_interval/18) to minimize delay and jitter, a TCP client can set the epoch value to 20ms (beacon_interval/6) to minimize bandwidth loss, a video client can set it to 50ms (beacon_interval/2) to maximize scheduling benefits. This feature make Covenant very flexible to the needs of different traffic. For simplicity, however, we assume that all nodes have equal epochs for the rest of the paper.

III. EVALUATION

In this section we evaluate the feasibility and performance of Covenant. The performance of Covenant is dependent on the control flow of information (load exchange packets) and the parameters of the system (epoch values).

A. Load exchange packets

The load exchange (lodex) packets are the control packets of the mechanism and its is critical for the performance of Covenant. There are two important requirements on them:

- They should be transmitted by the hardware on time.
- They should be received by other nodes in their load exchange stage.

The first requirement can fail since both control and data packets share the medium. Moreover due to our pipelined design there is always an overlapping scheduling phase where data packets are being transmitted. To study the effect of such background traffic on lodex packets we performed an experiment where a Covenant node shares the channel with two 802.11 nodes. To saturate the channel the two 802.11

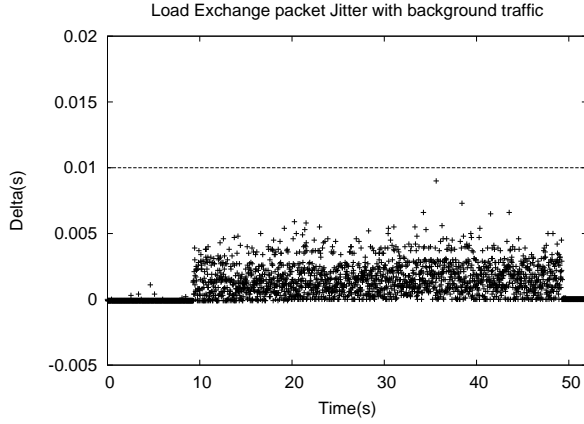


Fig. 4. This plot shows the deviation of the lodex packets from expected arrival time in the presence of two full-blast interfering nodes. In spite of two other interfering nodes, almost all the lodex packets arrived at an independent sniffer well within the allocated 10ms budget for the stage.

nodes were running an UDP packet generator sending traffic as fast as possible.

Figure 4 show the effect of this background traffic on the arrival times of the load exchange packets. This data is obtained by a passive sniffer operating in the same channel as the nodes. The graph plots difference between expected arrival time of the lodex packets and the actual arrival time. In ideal conditions, the lodex packets should arrive every epoch time period (20ms) and hence the difference should be *zero*. As the first 10 seconds of the graph shows, this is the case when there is no background traffic. Between 10 and 50 seconds two competing wireless nodes start transmitting packets as fast as they can. This affects the *jitter* experienced by the lodex packets due to increased contention in the wireless medium. As the figure shows, the worst case jitter, when the channel is saturated, does not exceed $\frac{1}{2}$ the epoch time shown by the straight line at 0.01 sec. Thus for an epoch time of 20ms all the lodex packets are sent within the lodex stage.

Since each lodex packet is essentially a broadcast packet there are no explicit acknowledgements. This leads to *packet losses* during channel congestion. In the experiment above, we observed a packet loss of 2%. The loss of lodex packets can lead to incomplete knowledge of the system load. To avoid this problem, we can maintain a short term *history* of lodex packets and extrapolate demand estimate on a lodex packet loss, although we have not yet deployed this optimization yet. Both the packet losses and jitter are smaller when the channel is not fully saturated or when TCP traffic is used ($< 1\%$ packet loss).

The second requirement needs some way of synchronizing the pipeline stages at all the participating nodes. As explained in design section, we propose to use beacons to achieve this synchronization. Beacon interval values are commonly set to 102.4ms and they use microsecond granularity while most kernel timers use millisecond granularity or higher. This could be a problem since we cant precisely sync with the beacons. Moreover another constraint is that all the epoch intervals must have the same time period. We solve this problem by using varying epoch times that averages to the beacon intervals

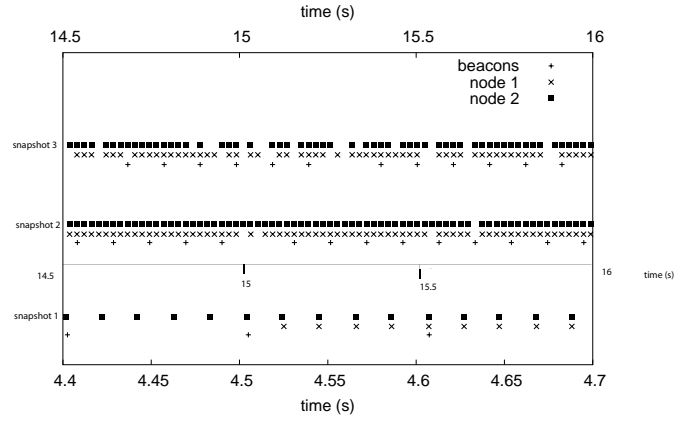


Fig. 5. Timeline of lodex packets and beacons. There are three different snapshots each showing the beacon arrival times, node 1 lodex packet arrival times and node 2 lodex packet arrival times. The top two snapshots use a different scale of timeline.

while keeping the variations small. The epoch time period is calculated as

$$next_epoch_int = \lfloor \frac{beacon_int}{num_epochs} \times epoch_val \rfloor$$

where $next_epoch_int$ is the time period for the next pipeline stage, $beacon_int$ is the beacon interval, num_epochs is a constant decides how many epochs are there between any two beacons and $epoch_val$ decides which epoch the pipeline is in between two beacons. This mechanism makes bootstrapping a new node very simple. When a new node wants to join the system it uses the last received beacon (which is always maintained by all the nodes) to figure out which epoch it has to join and starts accordingly.

Figure 5 illustrates the performance of this technique. In this experiment, two Covenant nodes are trying to synchronize with each other. We use a sniffer to monitor the traffic and try to see the effect of saturating the channel using UDP background traffic similar to the previous case. The graph depicts three different snapshots of the system. The bottom one representing a timeline of 0.300ms, show node 1 joining the system. Node 2 is clearly synchronized with the beacon and Node 1 uses the last seen beacon at 4.5 to bootstrap. The snapshot 2 represents a timeline of 1.5sec (different scale) and shows that synchrony is maintained in longer timescales also. Also note that the beacon packet got lost near 15sec but the nodes stay synchronized. The third snapshot is for 1.5 sec but with background traffic. Due to competing traffic there are more packet losses in this case but the nodes stay synchronized.

B. Number of nodes

The number of nodes participating in the scheduling discipline is another significant factor affecting performance. More nodes imply more load exchange packets which can consume bandwidth and capacity. One approach to solving this problem is to increase the epoch time periods, which in turn frequency of load exchange. But, this would cause packets to experience

larger delays. The complete size of the load exchange packet we have implemented including 802.11 headers is 208 bytes. If we include the time for other 802.11 actions (e.g. DIFS, PLCP, etc.), the airtime consumed by this packet for 802.11b card operating at 11 Mbps is $270\mu s$ and for 802.11a at 36 Mbps it is $148\mu s$. The bandwidth consumed by the control channel is $12.8Kbps$ (approximately .1% for 802.11b at 11 Mbps) per node when operating at $20ms$ epoch periods. As the number of nodes increase the saturation capacity also drops [?] and hence this value may become more significant. For ten nodes, the control channel will occupy $270\mu s + \delta$ (δ is the contention backoff time) for each node. Our solution is to increase the epoch time of the pipeline stages as a tunable parameter to offset this effect.

C. Epoch time period

The main parameter of the mechanism is the time period of the pipeline stages. This becomes more critical when the estimation stage uses buffering. This would contribute to packet delay and jitter which can affect both UDP and TCP traffic. The choice of this parameter also reflects a tradeoff in the system. A smaller value would make Covenant more transparent to other layers but decreases scheduling efficiency and increases control overhead (more lode packets). A bigger value would have the opposite effect. In this section, we analyze the effect of the time period on UDP and TCP traffic.

UDP traffic is the most common means of transport for multimedia applications. To study the effect of Covenant, we use evalvid [10] to trace the packet flow of a real multimedia stream. For this experiment, a node running Covenant streams the multimedia file (MPEG-4 video) over the access point to a networked machine. Traces are collected at both ends to study the delay and jitter caused by Covenant. This setup is repeated using a regular 802.11 node to compare performance. Figure 6 shows the effect of varying the epoch time on mean delay and mean jitter of the multimedia traffic. The mean delay increases with the epoch time since packets are buffered longer in the estimation stage. Compared to the regular node at 0 whose mean delay is at $8ms$, the delay is doubled to $18ms$ for a $20ms$ epoch. For non-real time multimedia streaming this delay will be easily captured by a playback buffer at the receiving side. For interactive multimedia traffic, adding a $10ms$ delay may affect the performance if the cumulative delay (Covenant + network) becomes larger than ones the codec can handle. For example, the VoIP can handle a delay of upto $250ms$ in the network.

As the top plot in Figure 6 shows, packet jitter is not affected by the epoch time. This is because multimedia streaming from a media source with playout buffer (like playing from stored media) will generate packets at fast rates. Since buffering in Covenant can only shorten the gap between the packets, it does not affect packets that are anyway very close to each other. Further because of the pipelined design, packets separated by an estimation stage also don't have any jitter in them. Hence, Covenant does not affect the inter-packet delay in this case. But this may not be valid for real time interactive traffic that generates packets in a periodic fashion (e.g., $20ms$ inter arrival

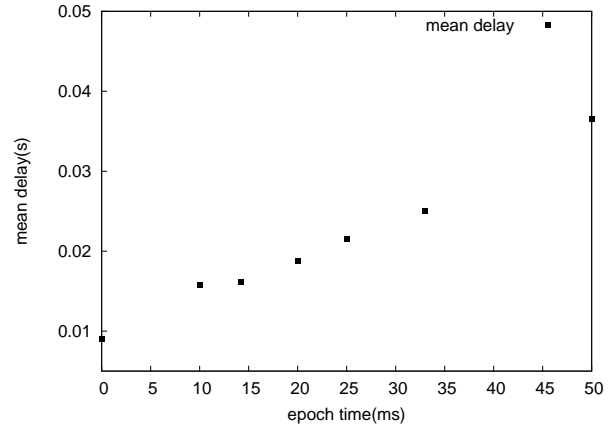


Fig. 6. Mean delay for multimedia streaming traffic. The mean jitter was constant at 2ms.

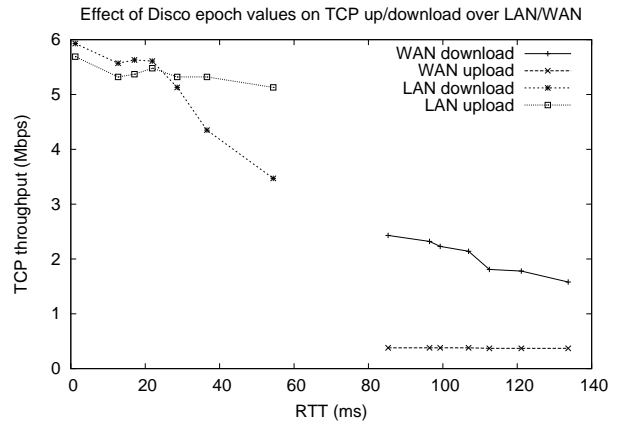


Fig. 7. TCP throughput

time for VoIP packets). In this, case we propose to use passive estimation as suggested in the design section.

The effect of Covenant on TCP traffic is a direct increase in the RTT as perceived from Covenant nodes. We performed experiments to study the extent of these effects while varying the epoch times. We evaluate TCP performance by measuring throughput in two different settings – LAN and WAN. In the LAN setting, a Covenant node transmitted packets to a wired desktop sharing the same access point. In the WAN setting, we used the planet-lab node in Cornell University and the Covenant node in San Diego. Although Covenant has an asymmetric effect on traffic (only outgoing traffic is affected), it can still impact both directions of closed loop protocols such as TCP. Hence, we tested the effect on both upload and download TCP traffic (shown in Figure 7). As expected, increasing RTT increases as the epoch time increases. Since the effect of epoch time on RTT is apparent, we plot throughput (median amongst five measurements) versus RTT instead of epoch times. As the graph shows, the effect of Covenant is more significant in the LAN than the WAN traffic since the wireless link is the bottleneck link in the former. The smallest RTT value is the base case when the node is running on regular 802.11. For the LAN setting at $20ms$, the drop in throughput is very marginal – around $300Kbps$ for upload and download. However, as we increase the epoch period from $20ms$ to $60ms$,

we observed significant throughput loss (about 2.5 Mbps) suggesting that smaller epoch values are preferable for TCP throughput intensive connections when the base RTT is small. For the WAN setting, there is no effect for the upload traffic, while download traffic suffers by 100 Kbps. We consider this epoch value to be a sufficient tradeoff between affecting TCP performance and Covenant performance. Another effect is that buffering ACKs affects TCP more in both WAN and LAN settings. The reason is that ACKs control the congestion response in TCP, and therefore delayed ACKs can be much more harmful to TCP throughput than that of the data packets.

In our analysis of the Covenant extension layer, we noticed that a 20ms epoch value is a good choice for tradeoff between delay and overhead. We use this value for implementing Covenant in our case studies. But Covenant provides the benefit of being tunable to meet different demands. For example, if the number of participating nodes go up, the epoch values can be set higher. This translates into smaller control overhead and more time in the load exchange state to receive other lodex packets (to overcome high contention delays). For delay sensitive traffic, making this value smaller will reduce the delay experienced.

IV. CASE STUDY

In the last two sections, we introduced the design of Covenant, its evaluation and performance. The success of this framework hinges on realizing its benefits in a real world scenario. To this extent, we use the home networking scenario to depict the efficacy of Covenant.

Wireless home networking is a new frontier for 802.11 networking [11] [2]. 802.11 networks' initial growth was supported by widespread adoption of wireless data networking in the home environment. But for 802.11 to grow into a complete networking solution for homes, certain challenges have to be addressed. Some of the characteristics of wireless home networks are:

- *Diverse traffic*: Networking in home environment is not restricted to PCs and laptops anymore. Devices like DVD players, VoIP phones are also data sources and destinations in the network. Broadly, media devices, communication devices and computing devices are part of the home network. Each group has its own set of features and requirements for the traffic flow. Conventional 802.11 devices are not designed to handle this variety, since the 802.11a/b/g protocols only provide fair packet based sharing among all participants.
- *QoS requirements*: Multimedia traffic needs some bandwidth guarantees for successful transmission. Interactive voice traffic require higher priority and are sensitive to jitter, while TCP occupies any unused bandwidth for transmission. 802.11 a/b/g cannot handle any of these requirements since they operate for providing basic packet delivery in the wireless network and do not provide mechanisms to support service differentiation.
- *Rate diversity*: It is common to have different nodes in a home environment operating at different rates because of their relative positions to the APs or to each other. In

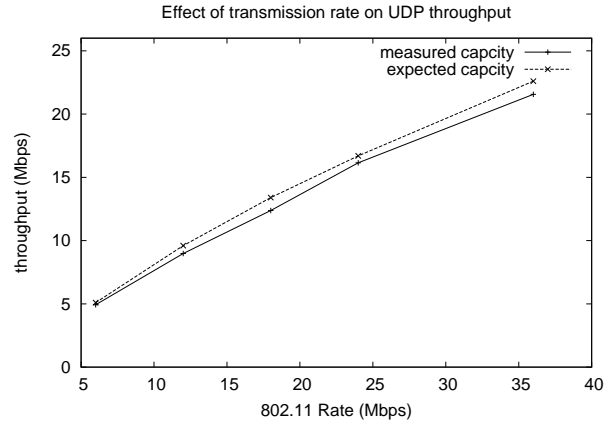


Fig. 8. Variation of saturation capacity over different operational rates.

such situations the aggregate throughput of the system is brought down due to packet based sharing of 802.11 [12]. This may not be a favorable policy for other traffic flows.

- *Packet diversity*: Packet sizes also vary greatly in such a network. For example, voice packets are 300 bytes in size [13], UDP streaming packets are 1300 bytes and TCP packets are 1500 bytes. This could also lead to a unfairness in sharing since 802.11 does not account for packet size but just the number of packets in allocating the share [5].
- *QoS-node associativity*: Each node has a distinct requirement for its traffic flow. Since most nodes are monolithic in terms of the traffic they generate. For example, DVD players generate high bandwidth UDP packets, VoIP phones generate high priority rtp traffic. Hence it is easier to decide on QoS policy at the link layer for higher layer traffic.
- *Cooperative allocation*: Since nodes belong to the same domain, they can cooperate to enforce globally optimal policy.
- *Full duplex packet flow*: Unlike conventional data networks, home networks have an equal share of upload and download traffic. Typically, laptop clients which generate web traffic tend to be download intensive, while video-streaming server tends to be upload intensive. Of course, devices such as VoIP phones are both upload and download intensive.

There are many challenges that need to be addressed for making 802.11 viable in this scenario and the architecture of the system is favorable for Covenant deployment.

A. Experimental setup

Most of our experiments are performed on two dell inspiron laptops running on linux-2.4.28 kernel with the Covenant drivers. For ease of implementation, we chose to use a conventional access point and all the destination nodes were machines on the wired network connected over a 100BaseT cable. This way, we ensure that all the traffic flow in the wireless channel is controlled by Covenant. All traces were collected at source, destination and a sniffer operating in the

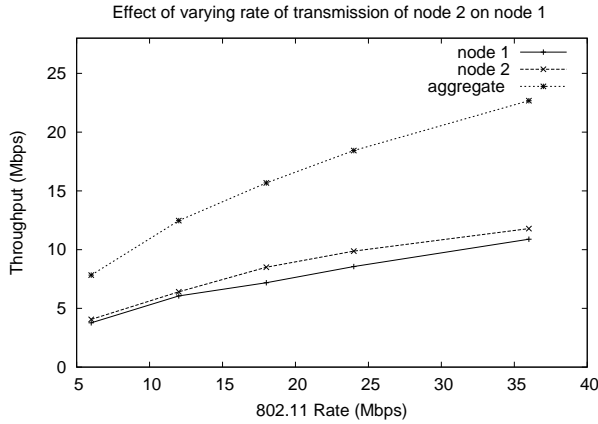


Fig. 9. Node 2 is operating at 36Mbps. Node 1 varies from 6 to 36Mbps. Aggregate bandwidth is affected.

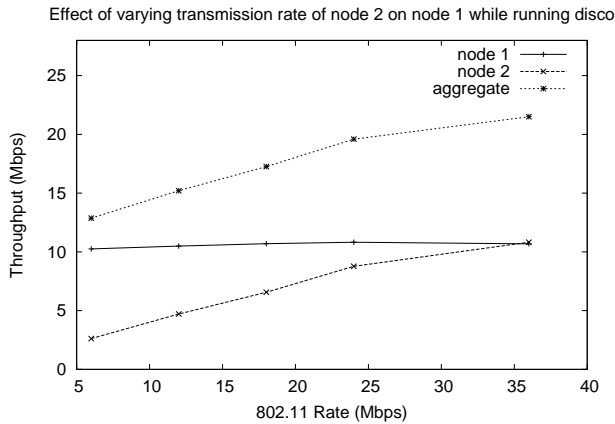


Fig. 10. Using Covenant Node 2's throughput remains the same due to air-time sharing

same channel. The experiments were conducted in both 11a and 11b network depending on the baseline capacity required. It is important for the nodes to estimate the channel capacity to calculate their share based on the scheduling policy. We use common estimation techniques based on the transmission rate, physical and MAC layer overhead [14]. Figure 8 shows a comparison of this estimate with the observed capacity. The values were averaged over 10 runs.

Thus each lodex packet contains information about rate (r_i) and load in bytes (b_i).

B. Rate and packet diversity

802.11 provides fairness based on packets generated without any flexibility for different policies. To demonstrate this effect, we used two nodes with regular 802.11 drivers using the 802.11a protocol. The nodes were running an UDP packet generator (at full rate) to a destination over the wired network in the LAN. The throughput is recorded at the destination. As Figure 9 shows, the throughput of node 1 which is operated at 36Mbps is brought down by the throughput of node 2 whose transmission rates varies (6,12,18,24 and 36Mbps). All the plotted values are averaged over 10 runs. The aggregate throughput of the system suffers because of this property as shown in the Figure. We resolve this problem by sharing load

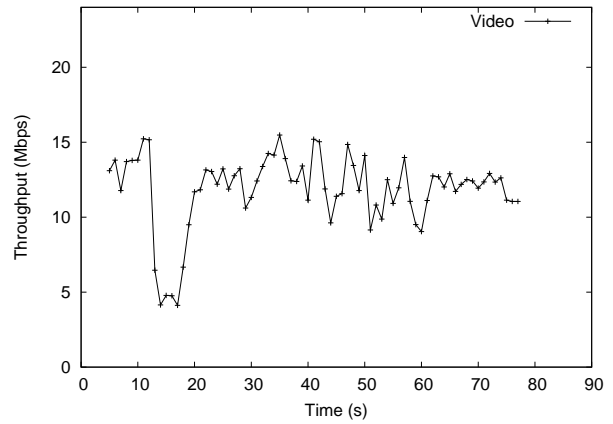


Fig. 11. Media traffic without contention

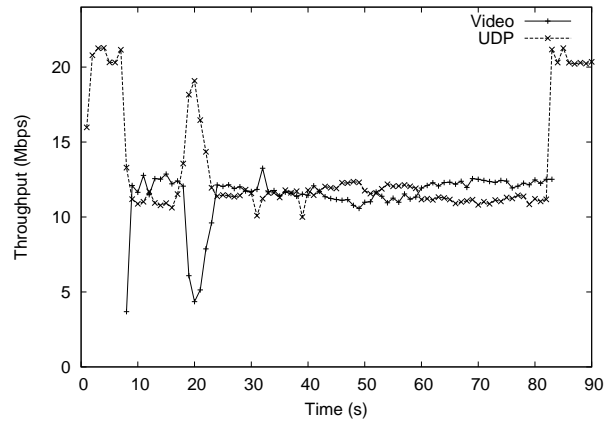


Fig. 12. Media traffic with contention (UDP traffic)

and the transmission rate in the lodex packets. Thus each node is aware of the number of nodes participating in the following scheduling stage, their load and the rates their operating. From this information, a simple air-time based sharing of the scheduling stage can be estimated. If any airtime slice is under-subscribed, the free time is redistributed among oversubscribed nodes.

Figure 10 shows the result of using this technique for improving aggregate throughput. The expected throughput for node 2 should be same as when it is sharing with another node at the same rate. This is consistent with the figure when node 2 is at 36Mbps. The aggregate throughput has also improved compared to Figure 9. A similar approach is used for packet diversity. Since each node is aware of the load (bytes to transmit) at other nodes, byte based fairness is achieved by allocating equal number of bytes at all nodes during each scheduling stage. The effect observed is similar to the previous case.

C. Bandwidth reservation

The notion of bandwidth reservation is not available in 802.11. To achieve this with Covenant, we assign a priority mode and level to each node indicative of its requirements. These two values give a greater degree of freedom in implementing different QoS policies. This enables the system meet

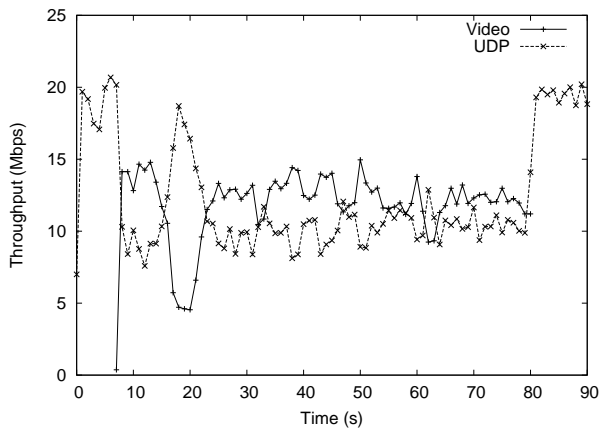


Fig. 13. Media traffic with contention(UDP traffic) and Covenant

unique requirements of each traffic. For example, an adaptive video traffic may just need a larger share of the capacity as much as possible. On the other hand, a high quality video traffic would need a guaranteed throughput. To meet these varied demands, the priority mode helps decide the kind of QoS required.

In the first case, we implemented proportional priority, in which each node is assigned a weight to decide its share of the resource. The resource could be air time or the size of data to send. The weights decide the proportion of resource each node gets. Similar to the previous experiment, any unused resource is redistributed among oversubscribed nodes. Let w_i, r_i and b_i denote the weight, operating rate and load in bytes, at each node i , then the share for the node is calculated as

$$share_i = w_i \times \tau \times r_i + \frac{w'_i}{n - k} \sum_{j=1}^k (w_j \times \tau \times r_j - b_j)$$

where n is the total number of nodes in the system and k is the number of under-subscribed nodes. w'_i is the proportional weight among the remaining $n-k$ nodes.

The experiments for studying this feature is performed using a popular media server (videolan [15]) playing a high quality variable bit rate MPEG-4 file at one of the nodes (running regular 802.11) while the receiver is a wired desktop in the same LAN. Figure 11 shows the instantaneous bandwidth measured at the receiver when only the videolan node is transmitting. Figure 12 shows the same graph with a background UDP traffic generated by another node. As soon as the videolan starts at 8 seconds, 802.11 fair sharing makes the two nodes share the channel equally. Compared to the single node case, the videolan traffic is affected by the UDP flow and its average bandwidth drops to 11Mbps from 13.8Mbps in the previous graph. Figure 13 shows the same experiment but with the two nodes running Covenant with air-time as the resource. The videolan node is assigned a weight of 3 while the UDP node is assigned a weight of 1. As the figure shows the videolan node now achieves the same bandwidth as the node in Figure 11. Another interesting observation is that the UDP traffic fills in any under-subscribed airtime (due to variable bit rate) with its

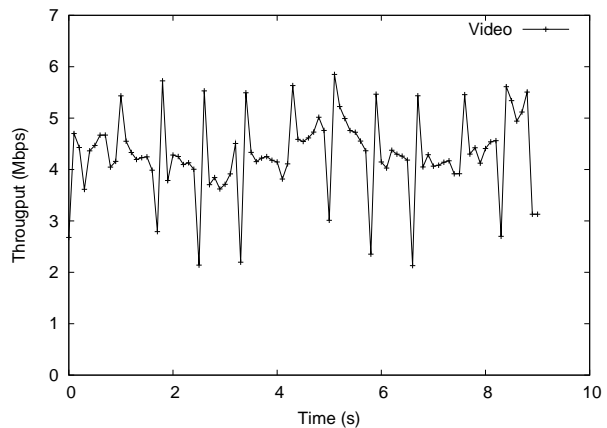


Fig. 14. Media stream without contention in 802.11b

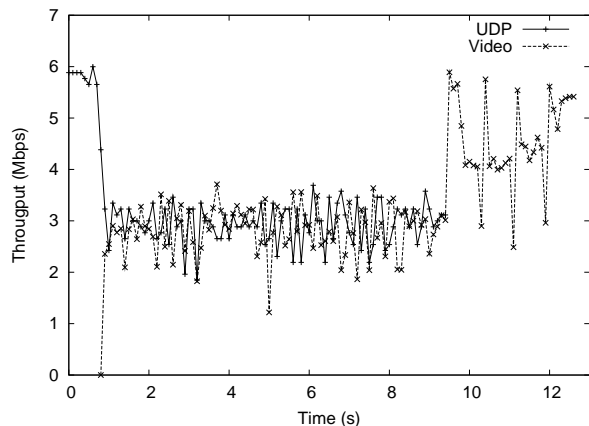


Fig. 15. Media stream with contention. The UDP traffic saturates the channel initially and shares it with the media stream.

traffic. This represents the advantage of using Covenant where it lets the high bandwidth traffic to perform unchanged while at the same time make optimal use of the resource.

For a high priority node, the priority level decides who get first share of the resource. The highest priority node gets full share of the channel depending on its load. While the remaining resource is used by the rest. To evaluate this method we use the same setup in 802.11b to saturate the network. Figure 14 shows a different media stream bandwidth for the regular case with no interference. Compare this to Figure 15 where the UDP traffic shares the capacity with the stream. The bandwidth drops from 4.5Mbps to 3 Mbps that leads to a significant loss in quality. Moreover the media stream takes a longer time to complete (12s) compared to 10 seconds in Figure 14.

In Figure 16, the media stream traffic is given higher priority to complete its throughput every scheduling stage. The UDP traffic only uses up any remaining resource left in the stage. As the graph shows, the throughput of the UDP traffic remains the same while the UDP traffic throughput drops down to 1Mbps.

V. DISCUSSION

Covenant is a solution with a wide range of applications. The ability to control packet scheduling and its transmission parameters along with global knowledge can be used to

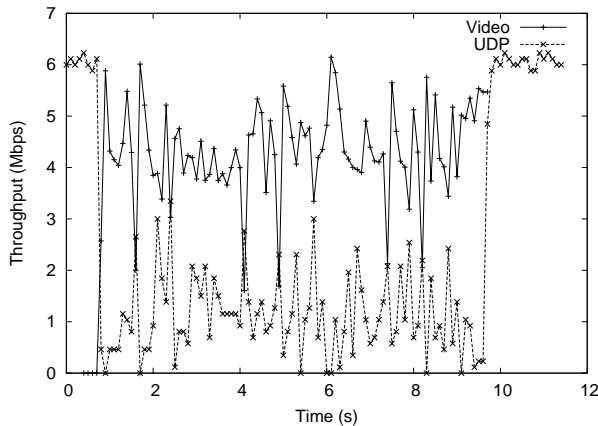


Fig. 16. Media stream with contention. Prioritized using Covenant

add novel and valuable features to 802.11. For example, Covenant can be used to improve the performance of the SPARTA protocol [6] for energy conservation. SPARTA uses information about load at other nodes to schedule packet at a slower rate to save power. By using Covenant, this information is readily available and can lead to accurate selection of rate. The flexibility of working in ad-hoc and mesh network environments also boosts the applicability of Covenant. One can use Covenant to inform other nodes about the load to avoid interference in mesh networks [16]. Covenant can also be used to gain benefits in areas other than packet scheduling. For example, handoff in 802.11 has problems with latency [17] and affects 802.11 mobility. Some solutions [18] suggest using a neighborhood graph of APs to make fast decisions. Using Covenant this information can be disseminated in the Iodex packets during the scanning process. The advantages in this process are the information is captured live and no infrastructure support is required.

Covenant's underlying philosophy is to give more control to the clients. The motivation here is based on the fact that the clients are in the best position to make decisions for optimal performance. The clients are aware of the channel conditions during transmission, they are closer to the applications generating the traffic and can better estimate the resource required for delivering packets. Covenant compliments this knowledge with information from other nodes participating in the channel. Clients in 802.11 network also have more computing resource, power and flexibility than access points. The 802.11 clients are usually laptops and PDAs that have powerful processors and large memories. The drivers in these devices are more easily accessible (ioctl, generic APIs) and upgradable.

Implementing Covenant was an interesting and rigorous process. It gave us new insights into the functioning of the kernel and how the accuracy of various functions are critical to the performance. We also realized that many hardware limitations can be overcome by smart software solutions and algorithms. This process also validated our approach of implementing a software based approach. Covenant has evolved through our implementation based on observations made during our experiments. Among the many ideas we chose to implement those with the greatest impact on performance. We have a

few other developments in the design board that is worth mentioning here.

Our implementation of Covenant assumes a standard channel capacity based on theoretical calculations. 802.11 spectrum being a highly noisy and variable channel can result in varying saturation capacities depending on location, interference and movement. The channel capacity is also based on the assumption that all the participating nodes use Covenant. This may not be the case if some legacy nodes are also being used in the neighborhood. In both these cases the estimated saturation capacity using theoretical approach can be inaccurate and lead to an attrition of Covenant benefits. One approach would be use rate estimation techniques for measuring channel capacity similar to [19]. This estimation technique can also benefit from information sharing using Covenant. For example, if the channel gets noisy due to interfering signals for non-802.11 devices, this effect will be noticed in all the other nodes. If the estimation technique used the information from other nodes, it can converge to the new rate faster.

Since nodes in Covenant share information about their load, they can estimate their share of the capacity. This knowledge can be used to indicate status regarding the network conditions to the upper layers. This upcall can be useful in applications that can adapt to network traffic [20]. One way of avoiding jitter is to replay the traffic as it comes to the extension layer thus maintaining inter-packet distance (over time). The current version of Covenant schedules all the packets at the beginning of the scheduling stage leading to jitter. This solution needs careful implementation to avoid timing issue in the scheduling stage leading to under-subscribing.

VI. RELATED WORK

The idea of using software approach in wireless networks is a popular one [21]. But with regards to 802.11, limitations of firmware APIs and support makes it difficult to implement. To our knowledge, our solution is one of the few practical implementations of this approach in 802.11. Our work is similar to the Overlay MAC [22] approach which is designed to implement a TDMA mechanism over the underlying 802.11 hardware. Our solution differs in the fact that we use explicit information sharing that can consume more resource but allow for highly optimal scheduling (dynamic knowledge of estimate). The pipeline mechanism and varying epoch intervals are also unique to our solution and our demands on clock synchronization are more relaxed.

Using explicit broadcast mechanisms for providing a control channel to add functionality is a common approach in many scenarios. Catch [23] performs information sharing using broadcasts packets to solve the free-rider problem in mesh networks. In most of these solutions the tradeoff is between the overhead and the benefits. Similarly Covenant can be practical only in scenarios where the advantage of having control over packet scheduling outweighs the capacity overhead of load exchange packets: like our case study.

The protocol based alternatives for QoS and service guarantees, use priority based channel access at the physical layer (eg. [24], [25], [3] and the references therein). These

mechanisms require firmware/hardware upgrades and a change in the standard. Covenant explores the other alternative of performing these in software with some tradeoffs. Another critical feature available in Covenant, is the ability to tune parameters like epoch times, priority mode and value. This degree of freedom is very critical in QoS implementations. 802.11e [3], the new standard for providing QoS guarantees, may face problems at high loads and cannot make strict guarantees. In these situations, Covenant can provide a convenient and easy alternative for various priority schemes and also handle the problem of rate diversity. To this extent, Covenant can be used in compliment with 802.11e devices to provide more tunability and performance. Covenant can also benefit from 802.11e by making load exchange packets high priority for best effort delivery.

VII. CONCLUSIONS

Wireless networks based on 802.11a/b/g technology have enjoyed tremendous success in terms of their penetration into various application domains – some of which are unforeseen. These application domain specific requirements such as service differentiation are currently being addressed by new MAC protocol standards. In this paper, we evaluated in depth Covenant, a cooperative scheduling layer that allows the implementation of flexible scheduling policies with minimal changes to the widespread 802.11a/b/g devices. Using experiments, we have shown that Covenant can be adjusted and tuned to a flexible mix of traffic types with minimal impact on the applications. Moreover, such flexibility can be achieved with only a small fraction (<0.1% per node) of the available channel for control packets. We also show real deployments of Covenant in home gateway scenarios to illustrate the applicability of Covenant in practice. While we have only began to scratch the surface, it appears that our architecture can be very useful in other application domains that require some form of cooperation.

ACKNOWLEDGEMENTS

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

REFERENCES

[1] Forward Concepts, ,” <http://www.analogzone.com/netp1020b.htm>.
 [2] DLink wireless media player, ,” <http://www.dlink.com/products/?pid=318>.
 [3] IEEE Draft Standard 802.11e, “Wireless medium access control (MAC) enhancements for quality of service(QoS),” Standard 802.11e, 2001.
 [4] Institute of Electronic and Electrical Engineers (IEEE), “Wireless medium access control (MAC) and physical layer (PHY) specifications,” Standard 802.11, 1999.
 [5] Godfrey Tan and John Gutttag, “Time-based fairness improves performance in multi-rate WLANs,” in *USENIX Annual Technical Conference*, June 2004.
 [6] Ramana Rao Kompella and Alex C. Snoeren, “Practical lazy scheduling in sensor networks,” in *Proceedings of ACM Conference on Embedded Sensor Systems*, Los Angeles, CA, Nov. 2003.
 [7] Ramana Rao Kompella, Sriram Ramabhadran, Ishwar Ramani, and Alex C. Snoeren, “Cooperative Packet Scheduling via Pipelining in 802.11 wireless networks,” in *Proceedings of ACM SIGCOMM E-WIND*, Aug. 2005.

[8] K.Romer, “Time synchronization in ad hoc networks,” in *In Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking and computing*, Aug. 2005.
 [9] Sourceforge MadWifi Driver, ,” <http://www.sourceforge.net/madwifi>.
 [10] J. Klaue, B. Rathke, and A. Wolisz, “EvalVid - A Framework for Video Transmission and Quality Evaluation,” in *In Proc. of the 13th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, Sept. 2003.
 [11] Zyxel VoIP WIFI phone, ,” <http://www.zyxel.com/product/P2000W.php>.
 [12] Martin Heusse, Franck Rousseau, Gilles Berger-Sabbatel, and Andrzej Duda, “Performance anomaly of 802.11b,” in *In proceedings of IEEE INFOCOM*, 2003.
 [13] Cision, ,” http://www.cision.com/pdfs/Tech_VOIP_Bandwidth.pdf.
 [14] Yang Xiao and Jon Rosdahl, “Throughput and delay limits of IEEE 802.11,” *IEEE Communication Letters*, vol. 6, no. 8, 2002.
 [15] VideoLan, ,” <http://www.videolan.org>.
 [16] MIT RoofNet, ,” <http://www.pdos.lcs.mit.edu/roofnet>.
 [17] Ishwar Ramani and Stefan Savage, “Syncscan: Practical fast handoff for 802.11 infrastructure networks,” in *Proceedings of IEEE INFOCOM*, Miami, FL, Mar. 2005.
 [18] Minh Shin, Arunesh Mishra, and William A. Arbaugh, “Improving the latency of 802.11 hand-offs using neighbor graphs,” in *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, Boston, MA, 2004.
 [19] Q. Xue and A. Ganz, “Proportional service differentiation in wireless lan using spacing-based channel occupancy regulation,” in *International Multimedia Conference*, New York, NY, May 2004.
 [20] Xiaomei Yu, Doan B. Hoang, and Dagan Feng, “A QoS control protocol for rate-adaptive video traffi c,” in *Ninth IEEE International Conference on Networks (ICON'01)*, Las Vegas, NV, Oct 2001.
 [21] Software Defined Radio Forum, ,” <http://www.sdrforum.org>.
 [22] Ananth Rao and Ion Stoica, “An overlay MAC layer for 802.11 networks,” in *In Proceedings of Networked Systems Design and Implementation of the 3rd international conference on Mobile systems, applications, and service*, Feb. 2005.
 [23] Ratul Mahajan, Maya Rodrig, David Wetherall, and John Zahorjan, “Sustaining cooperation in multi-hop wireless networks,” in *In Proceedings of Networked Systems Design and Implementation*, May 2005.
 [24] Imad Aad and Claude Castelluccia, “Differentiation mechanisms for IEEE 802.11,” in *In Proceedings of IEEE INFOCOM*, Apr. 2001.
 [25] Anders Lindgren, Andraes Almquist, and Olov Schela, “Quality of service schemes for IEEE 802.11 wireless LANs: An evaluation,” *Mobile Networks and Applications*, vol. 8, no. 3, 2003.