Covenant: An Architecture for Cooperative Scheduling in 802.11 Wireless Networks

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

***Abstract*—Wireless networks based on 802.11a/b/g protocols have gained wide-spread acceptance in both enterprise as well as home networks. However, these devices lack native support for many advanced features such as service differentiation, etc., that are required in specific application domains. In this paper, we propose Covenant, a software based cooperative scheduling framework to provide a rich set of features for applications that require nodes to cooperate with each other to satisfy system-wide objectives. We propose a novel $2\frac{1}{2}$-stage pipeline architecture as an efficient mechanism to implement cooperative scheduling among multiple nodes. We demonstrate how Covenant can be easily implemented in software, thus requiring absolutely no hardware or firmware changes to the already widely installed base of 802.11a/b/g based wireless devices. We also evaluate, using a real Linux based test-bed with Covenant drivers, the efficacy of the approach on two different scheduling disciplines: proportional priority and strict priority. We demonstrate that these scheduling disciplines are effective in providing service guarantees to multimedia applications even in the presence of other competing traffic.**

***Index Terms*—Scheduling, wireless, cooperative, video, QoS.**

I. Introduction

WIRELESS networks are becoming increasingly ubiquitous, in large measure, due to the popularity of the 802.11 based MAC layer protocols. While there are many protocols in the 802.11 suite and more protocols are constantly being added, 802.11a, b and g constitute the most popular among them and hence the de facto standards in commodity wireless networks. These three protocols provide the same mechanism for medium access and differ only in the supported set of transmit rates, and frequency band of operation.

Commodity wireless networks based on 802.11 can be used in a broad range of scenarios. Given the limited bandwidth in wireless channels, scheduling is often a key consideration in many of these scenarios, more so than in wired local area networks where bandwidth is plenty. For example, in wireless home networks, providing quality-of-service support is critical to delivering services like audio on demand, video on demand, voice over IP and high-speed Internet access to homes. *Throughput* guarantees are required by multiple competing high bandwidth multimedia streams delivered to wireless networked consumer electronic devices. There is, of course, a large body of research in the wireless community (e.g., [1]–[6]) with protocols and techniques to arbitrate channel access among competing nodes. However, most of these solutions suggest modifications to the de facto 802.11 protocol in order to perform the required form of scheduling. For example, the 802.11e standard offers Quality-of-Service (QoS) features by allowing higher priority packets to have lower congestion window as opposed to the lower priority packets. The downside to this, however, is the fact that existing installed base of 802.11abg based devices cannot directly be used in these settings.

While protocols to arbitrate the channel access among competing wireless nodes can be designed efficiently, it is an equally challenging problem to provide incentives for nodes to follow the protocol. Of course, the problem disappears when we consider a single administrative entity that controls all the participating nodes. In general, there are many scenarios, where nodes are required to cooperate among each other using a global scheduling discipline. For example, in home networks, media applications and the user's personal web browsing and email applications can cooperate in such a way as to satisfy simultaneously all the applications. In this paper, we concentrate on such scenarios where we assume that there is a single administrative domain such as a personal user or an enterprise. In our previous position paper [7], we outlined a framework to perform *cooperative scheduling*, where nodes cooperate with each other to share the limited bandwidth efficiently.

In this paper, our main contribution is the design of a software architecture called *Covenant* that enables easy implementation of the cooperative scheduling framework whereby nodes schedule amongst themselves in a *distributed fashion* in order to achieve a global objective. The key features of Covenant are as follows.

- Software based. Covenant is completely implemented in software and works with any 802.11 card including legacy devices to provide new features.
- Pipelined architecture. Covenant uses a novel $2\frac{1}{2}$-stage stage pipeline architecture to coordinate among nodes.
- Backward Compatible. In the presence of conforming and non-conforming nodes, Covenant degrades to the regular 802.11 thus the performance is no worse than 802.11.

We implemented Covenant by modifying an open source Linux based driver to create a flexible platform for new scheduling policies. We demonstrate the utility of Covenant through priority based scheduling policies that are not otherwise possible using off-the-shelf 802.11abg protocols.

Manuscript received September 23, 2008; revised February 13, 2009 and July 10, 2009; accepted August 25, 2009. The associate editor coordinating the review of this paper and approving it for publication was Prof. Q. Zhang.

I. Ramani is with Vudu, Inc.

R. R. Kompella is with the Computer Science Department, Purdue University, West Lafayette, IN, 47906 USA (e-mail: kompella@cs.purdue.edu).

S. Ramabhadran and A. C. Snoeren are with the Computer Science and Engineering Department, University of California, San Diego, La Jolla, CA 92093 USA (e-mail: {sriram, snoeren}@cs.ucsd.edu).

Digital Object Identifier 10.1109/TWC.2009.081272.

Scheduling in wireless networks using global knowledge is a rich area of research both in single-hop as well as multi-hop scenarios; the novelty of Covenant is not in the scheduling algorithms *per se*, but in the ability to achieve global scheduling objectives in a decentralized fashion without requiring any firmware changes.

The rest of the paper is organized as follows. In Section II, we present the architecture of Covenant including the $2\frac{1}{2}$-stage pipeline architecture briefly. We then, present controlled experiments to study the feasibility of Covenant in Section III. In Section IV, we present evaluation results of scheduling disciplines we have implemented using Covenant. Finally, we discuss related work in Section VI and conclusions in Section VII.

II. Architecture of Covenant

In this section, we describe the design and architecture of Covenant. We also discuss the pipelined implementation of our design and implementation details on 802.11-based wireless networks. The main assumption in our system design is that nodes are willing to cooperate amongst themselves. We believe this is a reasonable assumption in many scenarios where the nodes are controlled by a single authority. For example, in home wireless networks, the owner controls the wireless nodes that are competing for the same channel (e.g., wireless speakers, laptop nodes). In provider wireless networks, the service providers may potentially control the software on the wireless nodes, in which case nodes can be made to cooperate with each other. In such scenarios, the goal of our system is to provide a way to arbitrate channel access among competing nodes in such a way that some global performance objective could be satisfied. An example of such objective in a home network scenario is that, wireless speakers get all the bandwidth they need to sustain while the laptop node obtains whatever remaining bandwidth unused by the wireless speakers.

A. Design

The basic design of Covenant involves three stages:

- *Estimation*. Each node independently and periodically identifies its own medium access requirements.
- *Load exchange*. This local knowledge is then propagated to every other node so that each node obtains knowledge of the global demand on the channel.
- *Scheduling*. Finally, in this stage, each node independently computes a schedule based on this global demand and transmits packets accordingly.

In Covenant, each of the three above stages operates in a pipelined fashion (as shown in Figure 1) across all the nodes. Therefore, in every time interval each node directly controls what packets to send in that particular interval based on the global demands on the channel. The pipeline design is critical since it ensures that the scheduler itself is work-conserving and does not remain idle unless the node has no packets to send as can be seen from the figure where the scheduling phase of the cycle 2 immediately follows the one in cycle 1. In the Figure 1, we also show how two nodes interact in our Covenant architecture (note that APs can also be nodes in our architecture) using packet timelines. In parallel, both node 1 and 2 can observe the set of packets that need to be transmitted in the next round of scheduling. They then exchange the information using load exchange packets that allows them to compute how many packets they would like to transmit during that round of scheduling. The scheduling phase will then use DCF to transmit the exact number of packets from nodes 1 and 2 to transmit the packets.

The *estimation* stage in each node allows the to understand what the traffic demands are for a given round of scheduling. The stage itself is implemented through explicit buffering in the MAC layer; thus our design requires no special API between the MAC layer and the applications. Applications running on the node typically generate packets that pass through the MAC layer (in the device driver corresponding to the wireless card) where the packets are intercepted and stored in a small buffer. The number of packets collected in a small interval of time gives an estimate of the traffic requirements for that particular node. Thus, applications themselves do not need to explicitly state what their requirements are and can operate without modification.

At the end of the estimation stage, the node knows how many packets it needs to send within the scheduling round. This estimate of the node's demand is then communicated in the *load exchange* stage explicitly by injecting a "broadcast" packet (called the load exchange packet) into the transmit path. This broadcast packet consists of information about the state on that particular node including details about queue occupancy, transmission rate and other such parameters, depending on the particular scheduling discipline. This information, in some cases, can be obtained by observing the transmissions themselves so that explicit communication is not required. For example, in [8], the authors propose a distributed mechanism where the load can be deciphered from the rate of packet transmissions on the medium if every node follows the same protocol. However, for many other parameters of interest, there might be no implicit mechanism to convey the information and one would have to depend on explicit messages to transmit this information.

Finally, 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. Using this scheduling policy, the nodes determine whether to transmit the packets or to hold them back until it is their turn to transmit the packets. It is in this stage that conformance to the protocol is important, as malicious or non-conformant nodes that do not follow the protocol can hurt the overall system objectives. However, in the cooperative scheduling framework, we can expect all the nodes to be conformant to the protocol. The scheduling policy itself is scenario-dependent; in Section IV, we will describe a case study of a real deployment of Covenant including the particular scheduling policies that we used in that scenario.

It is important to realize that the system relies on accurate sharing of load information to arrive at a global scheduling policy. Clearly, this is not feasible on a per-packet basis (or micro-scheduling), as the control channel overhead would be too high. Thus, Covenant performs macro-scheduling at the granularity of intervals (equal to the pipeline stages). Thus,

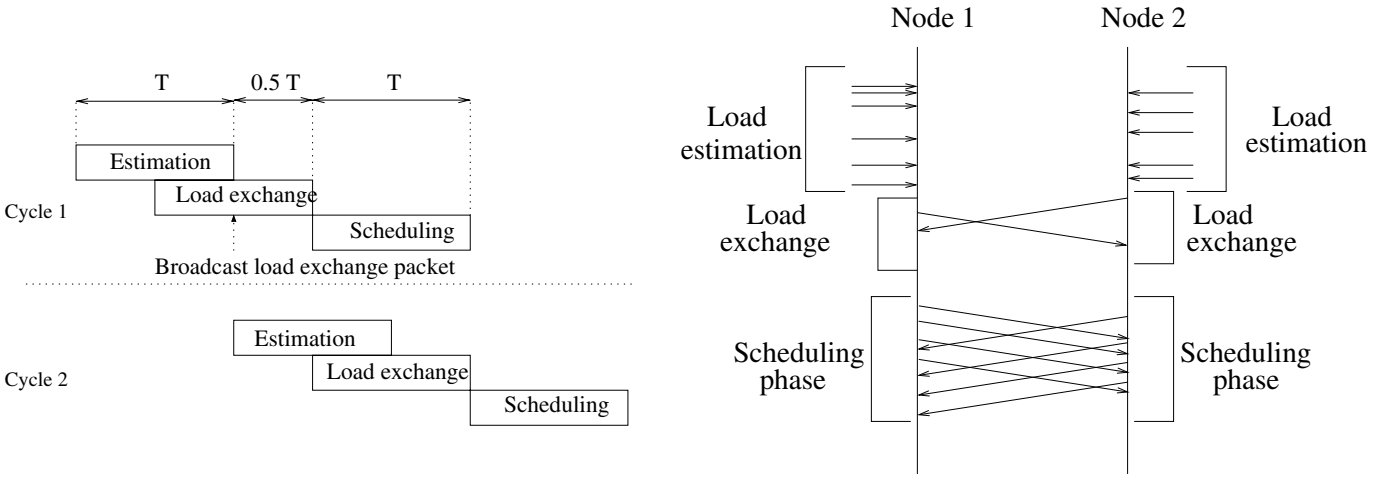


Fig. 1. $2\frac{1}{2}$-stage pipeline architecture. Each pipeline stage is equal in duration, but estimation stage and load exchange stage for a given cycle overlap with each other.

each node determines what set of packets to transmit in a given interval and schedules these packets for transmission. However, individual packets across nodes contend using regular 802.11 DCF for channel access while the number of packets and the type of packets scheduled are dictated by the global scheduling discipline.

B. Pipeline

We now discuss the pipeline implementation in more detail. As shown in Figure 1, each of the stages in the pipeline is for one epoch and they all have the same epoch time period. While all pipeline stages are typically non-overlapping, the load estimation and load exchange stages overlap partially because of the following reason. The load exchange packets are transmitted immediately after the load estimation stage. Ideally, each node should receive the load exchange packets from other nodes exactly after their corresponding load estimation stage or slightly later if we factor in the channel propagation delays. However, due to inexact time synchronization of the pipelines of transmitting and receiving nodes, load exchange packets can potentially arrive earlier than the expiry of the corresponding load estimation stage of the receiver. To accommodate this, we designed the load exchange stage to overlap in part with the load estimation stage, so that any load exchange packet received slightly earlier than the expiry of the load estimation stage would still be considered. This overlap, however, contributes only a *half* stage to the overall latency of the pipeline; hence the name $2\frac{1}{2}$-stage pipeline.

Note that all stages, particularly the estimation and scheduling stages are of equal sizes. This is important since we want the pipeline to be such that the same stage in two different cycles should have no overlap and also that the scheduling stage is work-conserving (i.e., there should exist no gap between two scheduling stages in consecutive cycles). Both these conditions are met by allocating equal time periods to all the stages in the pipeline.

Earlier, we mentioned that we implement load estimation stage in Covenant 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.

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 similar to [9].

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. In this stage, there could be packets that could not get transmitted either because the channel capacity decreased or due to other reasons. Packets that are not successfully transmitted in the scheduling stage are re-factored in

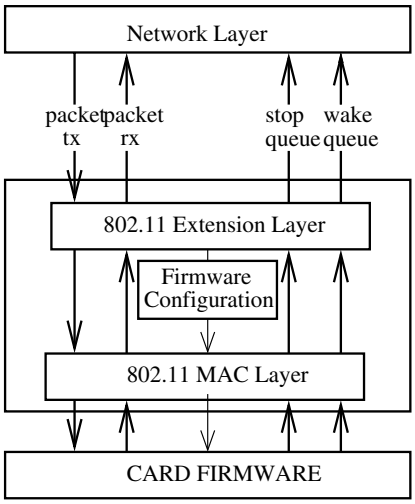


Fig. 2. Implementation of the $2\frac{1}{2}$-stage stage pipeline architecture within existing 802.11 based wireless cards.

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 simultaneously 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.

C. Implementation

Architecturally, Covenant is implemented as an extension within the 802.11 MAC layer [7] as shown in Figure 2. 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.

For our implementation, we chose to modify the open source madwifi driver for Atheros chip-set based Netgear 802.11abg cards. We implemented the $2\frac{1}{2}$-stage pipeline in this driver using kernel timers. The accuracy of the pipeline is dependent on the granularity of the clock. We set the parameter jiffies, $HZ = 1000$, so that the clock has an accuracy of 1ms. The implementation consists of an event handler that processes various events (both transmit and receive) and schedules new events using kernel timers to drive the pipeline. 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 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 injected into the transmit path of the driver with an appropriate 802.11 header with broadcast destination address. On the receive path, load exchange packet is intercepted by the core event handler so that it can update its global state for that particular cycle. Similarly, upon receipt of a beacon or any other synchronizing packet, the core event handler intercepts the packet, records the internal timestamp so that it can synchronize with other nodes.

III. Evaluation

In this section, we evaluate the feasibility and performance of Covenant. The accuracy and performance of Covenant is dependent on many things such as the frequency of the control packets (load exchange packets) are transmitted, the parameters of the system such as epoch values, the number of nodes in the system and so on. So, in this section, we use targeted experiments to evaluate the feasibility of Covenant on our test-bed.

A. Load exchange packets

The load exchange packets are the control packets of the mechanism and hence it is important for these control packets to:

- be transmitted on time by the hardware on the node; and
- be received by other nodes in their load exchange stage.

The first requirement can fail since both control and data packets (either from this node or other nodes) share the medium. Moreover, due to our pipelined design there is always an overlapping scheduling phase where data packets are being transmitted. Due to this competition, the load exchange packets can suffer delays. One way to alleviate this is to insert the packet into the highest priority queue among multiple priority queues that typically are supported by many wireless cards. If there is only one priority queue, we can insert the load exchange packets at the head of the transmit queue directly, which again ensures that the load exchange packets are given higher priority than the rest of the packets in the scheduling stage of the pipeline. Finally, if the 802.11 DCF is fair across nodes, the load exchange packets get transmitted within the allocated budget of time.

On the other hand, even if a given node transmits the load exchange packets on time, it might still be the case that the packet is not received in the corresponding load exchange stage of the receiver, perhaps due to imperfect time synchronization between the transmitter and receiver pipelines. Hence, time synchronization is extremely important to ensure that load exchanges happen between nodes perfectly.

To study the effect of background traffic on load exchange packets we performed an experiment where a Covenant node shares the channel with two other wireless nodes. To saturate the channel the two 802.11 nodes continuously generate UDP traffic as fast as possible without performing any flow control.

Figure 3 shows the effect of this background traffic on the arrival times of the load exchange packets. This data is obtained by a passive sniffer (an 802.11 node in monitor mode) operating in the same channel as the nodes. The

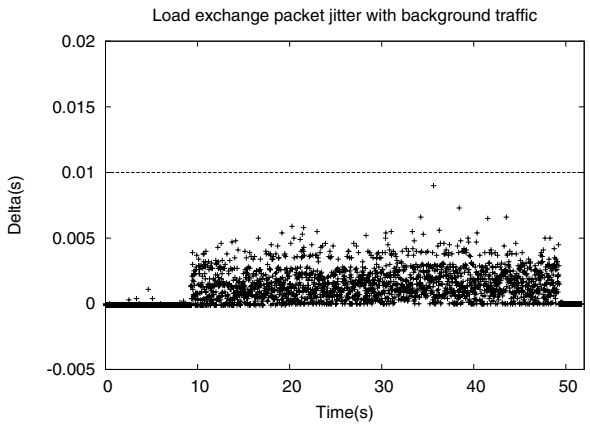


Fig. 3. This plot shows the deviation of the load exchange packets from expected arrival time in the presence of two full-blast interfering nodes. In spite of two other interfering nodes, almost all the load exchange packets arrived at an independent sniffer well within the allocated 10 ms budget for the stage.

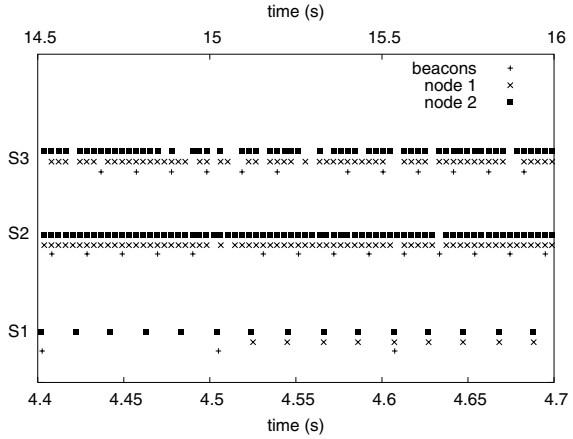


Fig. 4. Timeline of load exchange packets and beacons. There are three different snapshots (S1, S2 and S3) each showing the beacon arrival times, node 1 load exchange packet arrival times and node 2 load exchange packet arrival times. The top two snapshots use a different scale of timeline.

graph plots difference between expected arrival time of the load exchange packets and the actual arrival time. In ideal conditions, the load exchange 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 load exchange 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 load exchange packets are sent within the load exchange stage.

Since each load exchange packet is essentially a broadcast packet there are no explicit acknowledgments. This leads to *packet losses* during channel congestion. In the experiment above, we observed a packet loss of 2%. The loss of load exchange packets can lead to incomplete knowledge of the system load. To avoid this problem, we can maintain a short term *history* of load exchange packets and extrapolate demand estimate on a load exchange 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 102.4 ms and they use microsecond granularity while most kernel timers use millisecond granularity or higher. This could be a problem since we can't 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 while keeping the variations small and bounded. 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 4 illustrates the performance of this technique. In this experiment, two Covenant nodes try 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 snapshot representing a timeline of 300 ms, shows Node 1 joining the system. Node 2 is clearly synchronized with the beacon and Node 1 uses the last seen beacon at 4.5s 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 15 sec 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 270μs and for 802.11a at 36 Mbps it is 148μs. 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 [10] and hence this value may become more significant. For ten nodes, the control channel will occupy 270μs + δ (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 load exchange 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 [11] 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. The mean delay increases with the epoch time since packets are buffered longer in the estimation stage (not shown in a figure due for brevity). Compared to the regular node at 0 whose mean delay is at $8ms$, the delay is slightly more than 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 that the codec can handle. For example, the VoIP can handle a delay of upto 250ms in the network.

Packet jitter was also 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 time for VoIP packets). In this, case we can 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 5). 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 0.3 Mbps 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 0.1 Mbps. 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.

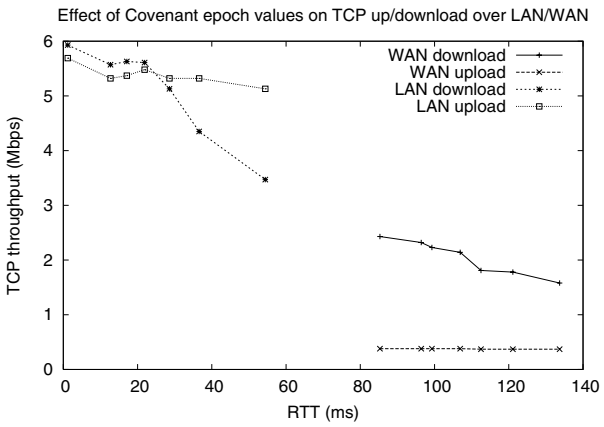


Fig. 5. TCP throughput

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 load exchange 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. Therefore, we use the home networking scenario in order to demonstrate the benefit of this framework.

Wireless home networking is a new frontier for 802.11 networking [12] [13]. 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 typical characteristics of wireless home networks are as follows.

- *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 such situations the aggregate throughput of the system is brought down due to packet based sharing of 802.11 [14]. 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 [15], UDP streaming packets about 1300 bytes and TCP packets are 1500 bytes. This could also lead to unfairness in sharing since 802.11 does not account for packet size but just the number of packets in allocating the share [6].
- *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 real-time 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 where traffic is predominantly download, home network scenarios involve 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.

From the points above, it is clear that 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

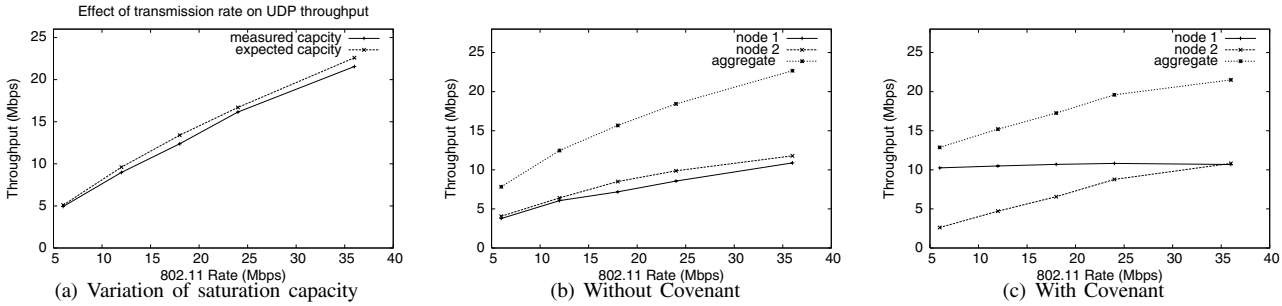
We performed most of our experiments on two Dell Inspiron laptops running on linux-2.4.28 kernel with the Covenant drivers. In order to emulate a realistic setting, we chose to use a conventional access point and all the destination nodes were machines on the wired network connected over a 100BaseT ethernet interface. This way, we ensured that all the traffic flow in the wireless channel is controlled by Covenant. All traces were collected at source, destination and a passive sniffer operating in the 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 [16].

B. Rate and packet diversity

802.11 provides fairness on a per-node basis without any flexibility for different policies. Suppose there are two nodes, one with a lower transmit rate and the other with a higher rate. Since 802.11 provides equal number of transmissions to both the nodes, that means the aggregate throughput of both the nodes considerably reduces. This is referred to as the 802.11 performance anomaly and has been studied before in literature [6], [17]. We show how we can alleviate this using Covenant.

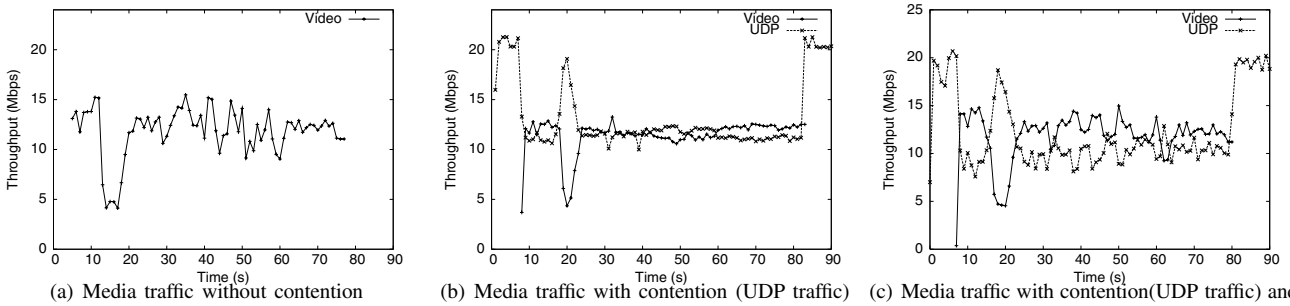
In our experimental setup, we used two nodes with regular 802.11 drivers using the 802.11a protocol. The nodes were running an UDP packet generator (at full rate) to a destination over the wired network in the LAN. The throughput is recorded at the destination. As Figure 6(b) shows, the throughput of 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 this property. We resolve this problem by sharing load and the transmission rate in the load exchange 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 airtime 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 6(c) 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 6(b). 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



(a) Variation of saturation capacity (b) Without Covenant (c) With Covenant

Fig. 6. Variation of aggregate bandwidth of two nodes with unequal transmit rates. In the subfigure (a), node 2 is configured to operate at 36 Mbps, while node 1 varies from 6 to 36 Mbps. Without Covenant, we can see that node 1 operating at lower rates affects the aggregate throughput achieved significantly. In the subfigure (b), we use Covenant to ensure that we split the bandwidth using air-time instead of channel access fairness provided by regular 802.11.



(a) Media traffic without contention (b) Media traffic with contention (UDP traffic) (c) Media traffic with contention(UDP traffic) and Covenant

Fig. 7. In this experiment, we study the effect of Covenant on a media stream with and without contention from other UDP traffic generators. The media node is configured to obtain at least 3 times the bandwidth received by another node.

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 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. There are two kinds of scheduling disciplines we have experimented with: proportional priority and strict priority.

Proportional priority. 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 and the weights decide the proportion of the resource each node gets. Similar to the previous experiment, any unused resource is redistributed among oversubscribed nodes thus not wasting any available bandwidth. 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

$$share_i = w_i \times \tau \times r_i + \frac{w_i'}{n-k} \sum_{j=1}^{k} (w_i \times \tau \times r_i - b_i)$$

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 [18]) 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. We used 802.11a network for this experiment so that the available channel capacity of 802.11a (which is much larger than 802.11b) can sustain the video clip we experimented with.

Figure 7(a) shows the instantaneous bandwidth measured at the receiver when only the videolan node is transmitting. Figure 7(b) 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 node 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 7(c) 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 7(a). Another interesting observation is that the UDP traffic fills in any under-subscribed airtime (due to variable bit rate) with its 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.

Strict Priority. The next policy we use is *strict priority*, in which the priority level decides who get first share of the

resource. Therefore, the highest priority node gets full share of the channel depending on its load, while the remaining resource is allocated to the next higher priority and so on. For this experiment, we used a similar setup to the previous experiment. This experiment, however, differs from the previous one in the following aspects.

- First, this experiment is run over 802.11b wireless network instead of 802.11a, which has considerably lesser available channel capacity.
- Second, the video stream (shown in Figure 8(a)) is different from that of previous experiment (shown in Figure 7(a)). The video stream in this experiment has a peak bandwidth usage of 4.5 Mbps in comparison with about 15 Mbps of the previous experiment.
- Third, we experimented with a different scheduling discipline, strict priority instead of the proportional scheduling in the previous experiment.

In Figure 8(b) when the UDP traffic shares the capacity with the stream, we can observe that the bandwidth received by the media stream drops from 4.5 Mbps (in Figure 8(a)) to 3 Mbps that leads to a significant loss in quality. Moreover the media stream takes 12 seconds which is longer time to complete compared to 10 seconds in Figure 8(a). In Figure 8(c), 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 1 Mbps. Thus, using this scheduling discipline, we can ensure nodes receive bandwidth in the order of their priority, highest priority to the lowest priority, very easily with the Covenant architecture.

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 add novel and valuable features to 802.11, in addition to the scheduling disciplines outlined in this paper. For example, Covenant can be used to improve the performance of the SPARTA protocol [8] for energy conservation. SPARTA uses information about load on 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 among each nodes. Covenant can potentially be applied in many other situations. For example, we can use Covenant to reduce interference in mesh networks [19]. We will examine such applications in our future work.

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 [20]. 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 up-call can be useful in applications that can adapt to network traffic [21]. 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.

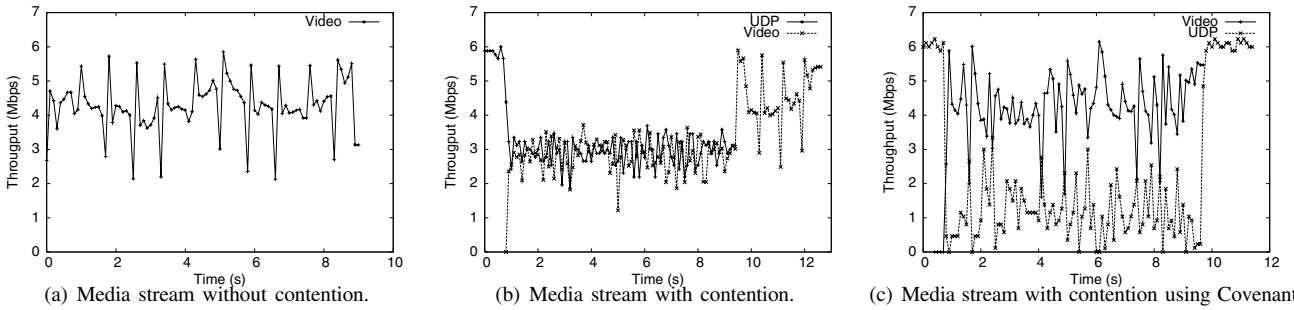
Note that during the load exchange phase, each node sends its estimation result by a broadcast packet. However, potentially this broadcast packet may not be received by all receivers in the interference range. We avoid this problem in infrastructure wireless LANs since the load exchange packets are typically forwarded to the access point which then re-broadcasts it to the rest of the nodes (unless each node spoofs the 802.11 frame as though it is actually broadcast by the AP). This re-broadcasting will typically ensure all the required nodes hear the load exchange message. In *ad hoc* wireless networks where there exists no such access point, we can only solve the problem by introducing a forwarding process. This problem is outside the scope of this paper; we consider it as part of our future work.

VI. Related Work

The idea of using software approach in wireless networks is a popular one [22]. 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 [23] 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 [24] 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 main alternatives for QoS and service guarantees that exist today involve using priority based channel access at the physical layer (eg. [4], [25], [26] and the references therein).



(a) Media stream without contention. (b) Media stream with contention. (c) Media stream with contention using Covenant

Fig. 8. In this experiment, we study the effect of Covenant on a media stream both in the presence of contention and without contention from other nodes in the system. For these experiments, we provisioned that the node that transmits the media stream has the highest priority.

For example, 802.11e standard involves providing faster access to the channel by modifying the DIFS (Distributed Inter Frame Spacing) for higher priority packets thus providing statistical guarantees. We outline the advantages and disadvantages of Covenant in comparison with these approaches.

Advantages and disadvantages of Covenant. Covenant fundamentally does not require any hardware upgrades and thus can operate with a wide install base of 802.11 a/b/g wireless cards. In contrast, the DIFS-based schemes require a change in the standard forcing firmware or hardware upgrades which can be difficult to achieve. Covenant also provides a flexible platform for implementing alternate approaches such as the time-based regulator (TBR) suggested in [6] that provides air-time usage fairness across nodes. It is not clear how such schemes can be implemented using the DIFS approach.

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 [26], 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.

Because Covenant inherently provides "macro" scheduling (scheduling in sets of packets) as opposed to the existing schemes, Covenant provides only coarse-granularity QoS as opposed to the other DIFS-based approaches that can implement per-packet priority (or micro-scheduling). Many typical scenarios, however, do not require per-packet scheduling and hence can be sustained using our mechanism.

Covenant requires cooperation across nodes, which incidentally, other schemes also require. If all nodes in the wireless channel use the same low value of DIFS, then the QoS functionality in 802.11e will not be effective. Thus, in this regard, we believe the assumption of coordination is not new.

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.

References

[1] N. Vaidya, P. Bahl, and S. Gupta, "Distributed fair scheduling in a wireless lan," in *ACM MOBICOM*, Aug. 2000.

[2] Z. Tang and J.J. Garacia-Luna-Aceves, "A protocol for topology-dependent transmission scheduling in wireless networks," in *WCNC 99*, Sept. 1999.

[3] J. Ju and V.O.K. Li, "An optimal topology-transparent scheduling method in multihop packet radio networks," *IEEE/ACM Trans. Netw.*, vol. 6, no. 3, June 1998.

[4] Anders Lindgren, Andraes Almquist, and Olov Schela, "Quality of service schemes for IEEE 802.11 wireless LANs: An evaluation," *Mobile Netw. Applications*, vol. 8, no. 3, 2003.

[5] V. Kanodia, C. Li, A. Sabharwal, B. Sadeghi, and E. Knightly, "Distributed multi-hop scheduling with delay and throughput constraints," in *Proc. ACM MOBICOM*, July 2001.

[6] Godfrey Tan and John Guttag, "Time-based fairness improves performance in multi-rate WLANs," in *USENIX Annual Technical Conf.*, June 2004.

[7] Ramana Rao Kompella, Sriram Ramabhadran, Ishwar Ramani, and Alex C. Snoeren, "Cooperative Packet Scheduling via Pipelining in 802.11 wireless networks," in *Proc. ACM SIGCOMM E-WIND*, Aug. 2005.

[8] Ramana Rao Kompella and Alex Snoeren, "Practical lazy scheduling in sensor networks," in *Proc. 1st ACM Conf. Embedded Sensor Syst.*, Los Angeles, CA, Nov. 2003.

[9] K.Romer, "Time synchronization in ad hoc networks," in *Proc. 2nd ACM international symp. Mobile ad hoc netw. computing*, Aug. 2005.

[10] G. Bianchi, "Analysis of the ieee 802.11 distributed coordination function," *IEEE J. Special Areas Netw.*, vol. 18, no. 3, pp. 535-547, Mar. 2000.

[11] J. Klaue, B. Rathke, and A. Wolisz, "EvalVid - A Framework for Video Transmission and Quality Evaluation," in *Proc. 13th International Conf. Modelling Techniques Tools Computer Performance Evaluation*, Sept. 2003.
[12] "Zyxel VoIP WIFI phone," [Online]. Available: http://www.zyxel.com/product/P2000W.php.
[13] "DLink wireless media player," [Online]. Available: http://www.dlink.com/products/?pid=318.
[14] Martin Heusse, Franck Rousseau, Gilles Berger-Sabbatel, and Andrzej Duda, "Performance anomaly of 802.11b," in *Proc. IEEE INFOCOM*, 2003.
[15] Cisilion, [Online]. Available: http://www.cisilion.com/pdfs/Tech_VOIP_Bandwidth.pdf.
[16] Yang Xiao and Jon Rosdahl, "Throughput and delay limits of IEEE 802.11," *IEEE Commun. Lett.*, vol. 6, no. 8, 2002.
[17] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda, "Performance anomaly of 802.11b," in *IEEE Infocom*, 2003.
[18] VideoLan, [Online]. Available: http://www.videolan.org.
[19] MIT RoofNet, [Online]. Available:http://www.pdos.lcs.mit.edu/roofnet.
[20] Q. Xue and A. Ganz, "Proportional service differentiation in wireless lan using spacing-based channel occupancy regulation," in *International Multimedia Conf.*, New York, NY, May 2004.
[21] Xiaomei Yu, Doan B. Hoang, and Dagan Feng, "A QoS control protocol for rate-adaptive video traffic," in *Ninth IEEE International Conf. Networks (ICON'01)*, Las Vegas, NV, Oct 2001.
[22] Software Defined Radio Forum, [Online]. Available: http://www.sdrforum.org.
[23] Ananth Rao and Ion Stoica, "An overlay MAC layer for 802.11 networks," in *Proc. Networked Systems Design Implementatioedings 3rd international conf. Mobile systems, applications, service*, Feb. 2005.
[24] Ratul Mahajan, Maya Rodrig, David Wetherall, and John Zahorjan, "Sustaining cooperation in multi-hop wireless networks," in *Proc. Networked Syst. Design Implementation*, May 2005.
[25] Imad Aad and Claude Castelluccia, "Differentiation mechanisms for IEEE 802.11," in *Proc. IEEE INFOCOM*, Apr. 2001.
[26] IEEE Draft Standard 802.11e, "Wireless medium access control (MAC) enhancements for quality of service(QoS)," Standard 802.11e, 2001.

Ishwar Ramani is a media software engineer at VUDU. He obtained his masters degree in computer science at UC San Diego and B. Tech degree in computer science at IIT Madras. His interests are in systems, networking and multimedia.

Ramana Rao Kompella (M'07, ACM M'07) is currently an Assistant Professor in the Department of Computer Sciences at Purdue University. His main research interests include fault-management in IP networks, scalable algorithms and architectures for high speed switches and routers, and scheduling in wireless networks. He received his Ph.D degree from UCSD in 2007, M.S from Stanford University in 2001, and B.Tech degree from IIT Bombay in 1999.

Sriram Ramabhadran is a Ph.D. student in the systems and networking group at University of California at San Diego. His interests are in peer-to-peer systems and router algorithmics. He obtained his B. Tech degree from Indian Institute of Technology, Madras in 1998.

Alex C. Snoeren (S'00, M'03) is an Associate Professor in the Computer Science and Engineering Department at the University of California at San Diego. His research interests include operating systems, distributed computing, and mobile and wide-area networking. Prof. Snoeren received the Ph.D. degree in computer science from the Massachusetts Institute of Technology (2003) and the M.S. degree in computer science (1997) and B.S. degree in computer science (1996) and applied mathematics (1997) from the Georgia Institute of Technology, Atlanta.