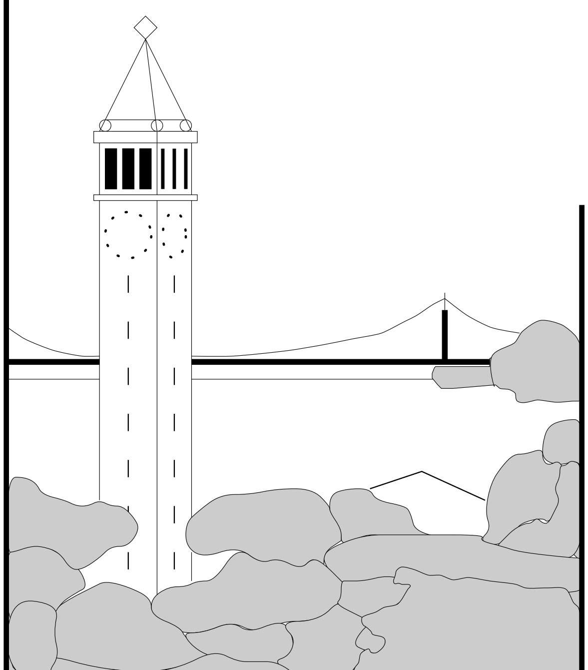
An Overlay MAC Layer for 802.11 Networks

Ananth Rao , Ion Stoica
{*ananthar,istoica*}@*cs.berkeley.edu*



Report No. UCB/CSD-4-1317

April 2004

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Abstract

The widespread availability of the 802.11-based hardware has made it the premier choice of both researchers and practitioners for developing new wireless networks and applications. However, the ever increasing set of demands posed by these applications are stretching the 802.11 MAC protocol beyond its intended capabilities. For example, 802.11 provides no control over allocation of resources, and the default allocation policy is ill-suited for heterogeneous environments and multi-hop networks. In this paper, we take a first step towards addressing these problems by presenting the design and the implementation of an *overlay MAC layer* (OML), that works on top of the 802.11 MAC layer. OML uses loosely-synchronized clocks to divide the time in equal size slots, and employs a distributed algorithm to allocate these slots among competing nodes. We have implemented OML in both a simulator and a wireless test-bed using the Click modular router. Our evaluation shows that OML can provide better flexibility as well as improve the fairness, throughput and predictability of 802.11 networks.

1 Introduction

In recent years, the widespread popularity of the 802.11 protocol has made it the de facto choice for developing and deploying a multitude of wireless networks and applications. Apart from the traditional model of a last-hop wireless link to an access point, 802.11 networks are used to setup wireless infrastructures in corporate networks, long-haul links using directional antennae in rural areas [10], and more recently multi-hop networks for broadband Internet access, *e.g.,* rooftop or mesh networking.

Despite providing the initial ease in deployment for many of these wireless technologies, the 802.11 protocol does pose serious limitations in addressing the different demands of these emerging applications. The original design of 802.11 MAC protocol was carefully engineered for the wireless LAN environment [22] and many of the underlying design assumptions may not hold in the new operating environments. The primary drawback in the 802.11 protocol is that it *does not provide any support for application-specific control over resource allocation*. For example, the traditional 802.11 resource allocation policy is known to be unfair in a multi-hop wireless network, and, in the worst case, can potentially starve certain nodes in the system without providing any throughput. Additionally, Bharghavan *et al.* [11] have shown the existence of a trade-off between throughput and fairness in resource allocation in multi-hop wireless networks which is dependent on the underlying topology connecting the nodes. The simple access control model of 802.11 does not incorporate topology based constraints to achieve this trade-off.

While several approaches have been discussed extensively in literature to address these problems, almost all of these require modifications to the 802.11 MAC layer. While some proposals [9, 11] provide the design of new MAC protocols with enough parameters for different applications, others have optimized the MAC protocol for specific targeted applications [21]. Over the past few years, it has become abundantly clear that the process of obtaining the approval of the IEEE and FCC organizations for any modifications to 802.11 is a laborious and notoriously difficult process. In the process, none of these proposals have moved towards universal deployability.

To address the dichotomy between the limitations of the 802.11 MAC protocol and the deployability problems of new MAC protocols, we propose the design of an Overlay MAC layer (OML) which does not any require changes to the 802.11 MAC hardware or the standard. The main idea of our solution is to perform the access control and scheduling at a layer above the MAC layer. Since our solution implements MAC functionality in software on top of the existing 802.11 MAC, we call it Overlay MAC Layer (OML). OML allows users to implement application-specific resource allocation, in the same way overlay networks allow users to implement application specific routing. To achieve this goal, OML uses loosely synchronized clocks to divide the time in equal size slots, and then uses a distributed algorithm to allocate these slots across the competing nodes. The slot allocation algorithm, called Weighted Slot Allocation (WSA), implements the weighted fair queueing policy [16], where each of the competing nodes that has traffic to send receives a number of slots proportional to its weight.

The contribution of this paper is two-fold. First it presents the design and the implementation of a MAC layer above 802.11, and it demonstrates by simulations and experiments that such an overlay layer can offer both more control, and better performance (by minimizing losses due to contention) than the ubiquitous 802.11. Second, it extends the fair allocation algorithm proposed in [9] by adding support for arbitrary weights to nodes and support for single channel multi-hop networks. Our solution requires no changes to the 802.11 layer, and it is compatible with native 802.11 clients.

The rest of the paper is organized as follows. In Section 2, we survey some related work in this area. Section 3 motivates the need for an overlay MAC protocol based on measurements performed in a test-bed. In Section 4 we describe the challenges involved and our design on the overlay MAC. Section 5 describes our test-bed and Section 5 describes how we implement OML in this test-bed. Sections 7 and 8 present results from simulations and the testbed respectively. Finally, we present future work and open issues in Section 9 and conclude our paper in Section 10.

2 Related Work

Our design of OML is motivated by some recent work which exposes problems with the 802.11 MAC in certain application scenarios. In [18], the authors describe how the presence of heterogeneous data rate senders can affect the system throughput in 802.11. They do not propose a solution to mitigate this problem. OML provides a solution for this problem and at the same time also addresses other issues like ad-hoc routing etc. The Roofnet and Grid projects at MIT[3, 2] have built multihop networks using 802.11 in ad-hoc mode. The primary focus of their work is on network layer issues; if fact we use the routing software developed by these projects in our testbed. Their results also indicate several problems like low throughput and very unpredictable performance of 802.11 at the MAC layer. We experience similar problems in our testbed also, and we recognize that many of these problems are because of interference from other senders and capture effects of the radio hardware. OML eliminates these problems by scheduling transmissions. On the other hand, [15] tries to improve throughput by adaptively picking better routes. Our work is complementary to improvements in the routing protocol. Extremely Opportunistic Routing[12] (ExOR) proposes a modification to the 802.11 MAC so that any eligible node that receives a packet clearly can send an ACK and forward the packet. Since OML schedules transmissions based on the sender only, and not on the destination of a packet, OML can be used on top of ExOR also.

A plethora of MAC protocols have been proposed to improve the predictability and the resource management capabilities in wireless networks. These solutions either use sophisticated back-off protocols [11], or slot allocation algorithms based on reservations[14, 25, 8, 9] to implement allocation policies such as WFQ[16]. Some MAC protocols use a hybrid of both approaches[14, 25]. Yet another approach is to use simple extensions to 802.11, *e.g.,* Sadegi *et al.* propose Opportunistic Rate Control (OAR) in [27] to address the problem of heterogeneous data rates. All these protocols work at the MAC layer and assume full control over the hardware and the physical layer. In contrast we assume that OML, can use only the limited interface exposed by most 802.11 cards to control packet transmission.

WSA uses pseudo-random functions in a similar way as the Neighborhood-aware Contention Resolution (NCR) proposed by Bao and Garcia-Luna-Aceves in [9] and [8]. Weighted Slot Allocation (WSA) algorithm builds on NCR, and extends it in two aspects. First, unlike OML, NCR assumes that the interference graph in a multi-hop network consists of isolated, easily identifiable cliques, *i.e.,* if node A interferes with B and B with C, then A interferes with C. Second, while NCR uses *pseudo-identities* to support integer weights, WSA can support arbitrary weights.

Several papers[26, 29, 17] provide sophisticated and very accurate mechanisms for clock synchronization in a multi-hop network. In the design of OML, we assume clock synchronization as an available primitive, and could possibly use any one of these algorithms.

Our solution is similar in spirit to the overlay network solutions that aim to improve routing resilience and performance in IP networks[6, 30, 13, 7]. Overlay networks try to overcome the barrier of modifying the network layer (*i.e.,* IP layer) by employing a layer on top of the IP to implement the desired routing functionality. Similarly, OML runs on top of the existing 802.11 MAC layer, and its goal is to enhance the MAC functionality without changing the existing MAC protocols.

3 Limitations of the 802.11 MAC

In this section, we will illustrate three specific limitations of the 802.11 MAC protocol using a wireless network testbed comprising six nodes. These limitations are:

1. In a network comprising of nodes with heterogeneous transmission rates, the 802.11 standard provides a very low system throughput since it attempts to be fair to every competing node.
2. 802.11 can sometimes cause significant unfairness and unpredictable performance when the interference regions of different senders overlap at a receiver.
3. 802.11 MAC provides sub-optimal resource allocation in multi-hop wireless networks.

3.1 Limitation 1: Heterogeneous transmission rates

The 802.11 standard aims to allocate an equal number of *transmission opportunities* to every competing node. However, as shown in previous work [18], this fairness criterion can lead to a low throughput in a network in which nodes transmit at widely different sending rates.

We illustrate this behavior using a simple experiment comprising of two heterogeneous senders connected to a single access point. We emulate heterogeneous senders by fixing the data-rate from Node 1 to the access point to 54 Mbps and varying the data-rate of Node 2 between 6 Mbps and 54 Mbps. Figure 1 shows the average throughputs of two TCP flows originated at the two nodes. This experiment shows that while the behavior is fair as nodes see equal performance irrespective of their sending rate, it hurts the overall system throughput. In particular, as the sending rate of Node 2 decreases from 54 Mbps to 6 Mbps, the system total throughput decreases from 24 Mbps to 7.2 Mbps. In addition, this behavior leads to poor predictability. For example, if Node 1 is the only active one, it will have a throughput of roughly 24 Mbps. However, when Node 2 starts transmitting at 6 Mbps, the throughput observed by Node 1 drops suddenly to 3.6 Mbps.

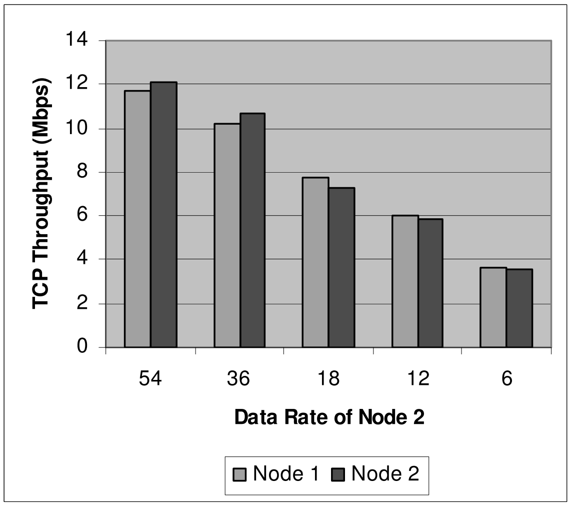


Figure 1: 802.11 throughput in the presence of heterogeneous data rate senders

Receiver \ Sender	1	2	3	4	5
1		Very Good			
2	Very Good		Fair		
3		Good		Very Good	
4			Very Good		Fair
5				Fair	

Figure 2: Received signal strength (RSS) in a chain configuration of our test-bed. The quality of the signal strength, namely, "Very Good or Fair" is shown as reported by the device driver of the network card.

3.2 Limitation 2: Interference and Signal Strength

An assumption often used in modeling and simulating wireless networks is that the interference is binary, *i.e.,* two nodes either interfere or don't interfere. However, this assumption is not necessary true in practice. The failure of this assumption can lead to significant unfairness, and unpredictable performance.

To illustrate this, we consider a simple testbed consisting of five nodes arranged in a chain, with nodes being numbered from 1 to 5. The network card of each node is set to operate at 6 Mbps. Figure 2 shows the quality of the link in each direction for every link in the chain measured as a function of the relative signal strength (RSS). Even though some of the links are asymmetric and the signal strengths varies significantly, measurements results show that, in the absence of any other traffic, each link can support a TCP flow with throughput of about 4.6 Mbps in each direction.

Next, consider two long lived TCP flows across a single hop along two different non-overlapping links in our chain topology. Table 3.2 reports the measured throughputs of each flow in various configurations. Under the simplistic assumptions of binary-mode interference and a perfect MAC protocol, we would expect each flow to receive around 2.3 Mbps if they interfere, and 4.6 Mbps if they don't. Instead, we find that the throughput varies between 0.01 Mbps and 4.6 Mbps.

By inspecting the measured data at the nodes (not shown here), we inferred that the interference observed by the two flows is not symmetric. For example, when we initiate two TCP flows along the wireless links $3- > 2$ and $5- > 4$, we notice one of the TCP flows to receive a meager throughput of 0.01 Mbps (row 4 in Table 3.2). This is illustrated by the fact that the RSS from node 3 at node 4 is much weaker than the RSS at node 4 from node 5. Hence the throughput of a flow from $5- > 4$ is limited by the weaker interference caused by Node 3's transmission to 2. Consequently, much alike the case of heterogeneous transmission rates, the flow from $5- > 4$ receives a very low throughput (Node 4 attempts to be fair in its allocation of transmission opportunities to 3 and 5 irrespective of the fact that 3 is communicating with 2). Additionally, we observe that the effect of one flow on another, is unpredictable. Consider the scenario where initially only one flow is active in the system and a second flow starts at a later time. The first flow can potentially experience a sudden decrease in throughput when the second flow starts. This decrease in throughput cannot always be predicted and can be highly variable depending on the signal interference that one flow induces on the other.

3.3 Limitation 3: Fairness in Multi-hop Networks

In a multihop network, the fairness policy implemented by 802.11 can lead to decreased system throughput and in some cases can completely shut-down certain flows. Nodes in a multi-hop network forward packets on behalf of other nodes in addition to generating their own traffic. Hence, the transmission opportunities of a node are divided proportionally between these two types of traffic. The fairness policy of 802.11 does not account for the additional traffic at a node and hence can generate a suboptimal allocation of resources.

Consider the example in Figure 3 where nodes N_1, N_4, N_5 and N_6 each generate a single flow to node N_2. Assume the interference range is twice the transmission range. Then N_2 cannot receive when nodes N_4, N_5 or N_6 are transmitting, and N_3 cannot receive when N_1 is transmitting. According to the 802.11 fairness policy, N_1 and N_3 each get $1/3$ of the bandwidth (of N_2), while the rest of the nodes share the rest. As a result, N_4, N_5, and N_6 get only $1/9$ of the entire

First flow			Second flow		
Sender	Receiver	Throughput (Mbps)	Sender	Receiver	Throughput (Mbps)
1	2	3.7	3	4	4.64
1	2	4.4	4	3	4.62
2	1	2.81	3	4	4.22
3	2	4.57	5	4	0.01
2	1	4.58	5	4	4.2
3	2	0.64	4	3	4.63

Table 1: Throughput of two simultaneous flows in the test-bed

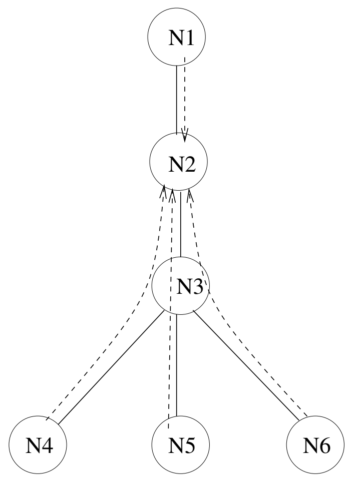


Figure 3: Example of interaction of multiple flows in multi-hop networks

capacity. It is worth noting that a better solution, would be to allocate $3/7$ of the capacity to node N_3, and $1/7$ of the capacity to each of the other senders. This way, the transmission rates of nodes N_4, N_5, and N_6 will increase from $1/9$ to $1/7$ of the capacity.

While this is an analytical example, we will show via both simulations and experiments that in a multi hop network, 802.11 can indeed lead to significant un-fairness. Actually, in some cases, the un-fairness is so pronounced that it causes flows to be shut-off.

4 Design

In this Section, we present our solution, an overlay MAC layer (OML), that alleviates the problems described in Section 3. We first state our assumptions.

4.1 Assumptions

The primitives available for the design of OML are determined by the interface exposed by the device driver. For the sake of generality, in this paper we make minimal assumptions about this interface. In particular, we assume that

- The card can send and receive both unicast and broadcast packets.
- It is possible to set the wireless interface in promiscuous mode to listen to all transmissions from its 1-hop neighbors.
- It is possible to disable the RTS-CTS handshake by correspondingly setting the RTS threshold.
- It is possible to limit the number of packets in the card's queue to a few packets. This is critical for enabling OML to control packet scheduling because once the packets are in the card's queue, OML has little control over when these packets are sent out.

We note that these assumptions already hold or can be enforced, eventually by modifying the device drivers, in most 802.11 cards.

While carrier-sensing is a very important primitive for the design of a MAC protocol, we do not assume that OML can use this primitive. This assumption reflects the fact that most 802.11 cards do not export the current status of the channel or the network allocation vector (NAV)[1] to the higher layers.

Since 802.11 uses the unlicensed ISM bands for communication, we cannot assume that all interfering stations will implement an overlay MAC. Thus, a station using OML should coexist with an unmodified 802.11 station, *i.e.,* an OML station should neither hurt the performance of another 802.11 station, nor suffer significantly in the presence of other 802.11 stations.

4.2 Solution

As noted in the previous section, the only control that OML can exercise over packet scheduling is *when* to send a packet to the network card; once the packet is enqueued at the network card, OML has no control on when the packet is actually transmitted. Thus, ideally, we would like that when OML issues a send request to the network card, the network card transmits the packet immediately (or at least with a predictable delay), and the packet to be delivered to the next hop or destination with high probability.

[1]NAV is maintained internally in the hardware to keep track of RTS, CTS and reservations in the packet header

To implement this idealized scenario, we propose a solution that aims to (a) limit the number of packets queued in the network card, and (b) eliminate the node interference, which is the major cause of packet loss and unpredictability in wireless networks. Goal (a) can be simply achieved by reducing the buffer size of the network card.

To achieve goal (b), we divide the time into slots of equal size l, and allocate the slots to nodes according to a weighted fair queueing (WFQ) policy [16]. We call this allocation algorithm the Weighted Slot Allocation (WSA) algorithm. WSA assigns to each node a weight, and in every interference region allocates slots in proportion to nodes' weights. Thus, a node with weight two will get twice as many slots as a node with weight one in the same interference region. Only nodes that have packets to send contend for time slots, and a node can transmit only during its time slots. Since a time slot is allocated to no more than one node in an interference region, no two sending nodes will interfere with each other. This can substantially increase the predictability of packet transmission, and reduce packet loss at the MAC layer. However, in practice, it is hard to totally eliminate the interference. In an open environment there might be other devices out of our control (*e.g.,* phones, microwave ovens), as well as other nodes that run the baseline 802.11 protocol which can interfere with our network. Furthermore, as we will see, accurately estimating the interference region is very challenging.

The reason we base WSA on the WFQ policy is because WFQ is highly flexible, and it avoids starvation. WFQ has emerged as the policy of choice for providing QoS and resource management in both network and processor systems [16, 23, 31]. Note however that WSA is only one allocation mechanism that can be implemented in OML; one could easily implement other allocation mechanisms, if needed.

There are three questions we need to answer when implementing WSA: (a) what is the length of a time-slot, l, (b) how are the starting times of the slots synchronized, and (c) how exactly are the times slots allocated among competing clients. We answer these questions in the next three sections.

4.2.1 Slot size

The slot size l is dictated by the following considerations:

1. l has to be considerably larger than the clock synchronization error.
2. l should be larger than the transmission time, *i.e.,* the interval between the time the first bit of the packet is sent to the hardware, and the time the last bit of the packets is transmitted in the air. This interval includes the queueing time in the network card.
3. Subject to the above two constraints, l should be as small as possible. This will decrease the time a packet has to wait in the OML layer before being transmitted and will decrease the burstiness of traffic sent by a node.

In our evaluation, we chose l to be the time it takes to transmit about 10 packets of maximum size (*i.e.,* 1500 bytes). We found this value of l to work well in both simulations and in our implementation.

4.2.2 Clock synchronization

Several very accurate and sophisticated algorithms have been proposed for clock synchronization in multi-hop networks[26, 29, 17]. We believe it is possible to adapt any of these algorithms to OML. However, clock synchronization is not the focus of our work. For evaluation of OML, we have implemented a very simple algorithm that provides adequate performance within the context of our experiments on the simulator and the test-bed.

We attempt to synchronize all the clocks in the network to the clock at a pre-designated *leader node*. We estimate the one-way latency of packet transmission based on the data-rate, packet size and other fixed parameters of the 802.11 protocol. We then use this estimated latency and a timestamp in the header of the packet to compute the clock skew at the receiver.

4.2.3 Weighted Slot Allocation (WSA)

In this section, we describe the implementation of WSA. The challenge is to design a slot allocation mechanism that (a) is fully decentralized, (b) has low control overhead, (c) and is robust in the presence of control message losses.

For ease of explanation, we divide the presentation of our solution in three stages. In the first two stages, we assume that every node in the network can receive transmissions from every other node. This also implies that only one sender can be active in the network at any given time. Furthermore, in the first stage, we assume that all nodes have unit weight. In the final stage, we relax both these assumptions.

Step 1 - Network of diameter one with unit weights: One solution to achieve fair allocation is to use pseudo-random hash functions as proposed in [9]. Each node computes a random number at the beginning of each time slot, and the node with the highest number wins the slot. Since the random numbers are based on the use of pseudo-random hash function, a node can easily compute the numbers of the other nodes without any explicit communication from those nodes.

Let H be a pseudo-random function that takes values in the interval $(0, 1]$. Consider c nodes, n_1, n_2, ..., n_c, that compete for time slot t. Then each node computes the value $H_i = H(n_i, t)$ for $1 \le i \le c$, where H is a pseudo-random function, and t is the index of the slot in contention. A node n_r wins slot t if and only if

$$\arg\max_{1 \le i \le c} H_i = r. \tag{1}$$

Since H_i is a pseudo-random random number, it is equally likely that any node will win the slot. This results in a fair allocation of the time slots among the competing nodes.

A node n_s will incorrectly decide that it can transmit if and only if it is unaware of another node n_i such that $H_i > H_s$. While the probability that this can happen cannot be neglected (*e.g.,* when a node n_i joins the network), having more than one winner occasionally is acceptable as the underlying MAC layer will resolve the contention using CSMA/CD. As long as such events are rare they will not significantly impact the long term allocation.

Step 2 - Network of diameter one with arbitrary weights: Let w_i denote an arbitrary weight associated to node i. Then we define $H_i = H(n_i,t)^{1/w_i}$, and again allocate slot t to node r with the highest number H_r. The next result shows that this allocation will indeed lead to a weighted fair allocation.

Theorem 1. *If nodes $n_1,\ldots,n_c$ have weights $w_1,\ldots,w_c$, and H is a pseudo-random function that takes values in the range $(0,1]$, then*

$$P[(\arg\max_{1\leq i\leq c} H_i) = r] = \frac{w_r}{\sum_{j=1}^{c} w_j} \tag{2}$$

Proof. The probability distribution function for H_i is given by

$$\begin{aligned} P[H_i \leq x] &= P[H(n_i,t)^{1/w_i} \leq x] \\ &= P[H(n_i,t) \leq x_i^w] \\ &= x_i^w \end{aligned}$$

Hence the probability density function for H_i, obtained by differentiating the distribution function is given by

$$f_{H_i}(x) = w_i x^{w_i - 1} \text{ for } 0 \leq x \leq 1 \tag{3}$$

Next, we compute the distribution function of the $\max[H_i, H_j]$ as follows,

$$\begin{aligned} P[\max(H_i,H_j) \leq x] &= P[H_i \leq x \text{ and } H_j \leq x] \\ &= P[H_i \leq x].P[H_j \leq x] \\ &= x^{w_i + w_j} \end{aligned}$$

By induction on the above result, we get

$$P[\max(H_{i_1},\ldots,H_{i_n}) \leq x] = x^{w_{i_1} + \ldots + w_{i_n}} \tag{4}$$

Now, let $W = \sum_{i=1}^{c} w_i$ and let Y_r denote the random variable given by

$$Y_r = \max_{1\leq i\leq c, i\neq r} H_i$$

Note that H_r is the maximum if and only if it is greater than Y_r. From Eqs (4) and (3), the distribution and the density functions of Y_r are given by

$$P[Y_r \leq y] = y^{\sum_{1\leq i\leq c, i\neq r} w_i} = y^{W - w_r}$$

$$f_{Y_r}(y) = (W - w_r) y^{W - w_r - 1}$$

Finally, we can compute

$$\begin{aligned} P[(\arg\max_{1\leq i\leq c} H_i) = r] &= P[H_r > Y_r] \\ &= \int_0^1 \int_0^x f_{H_r}(x) f_{Y_r}(y)\, dx\, dy \\ &= \int_0^1 w_r x^{w_r - 1} y^{W - w_r}|_{y=0}^{y=x} dx \\ &= \int_0^1 w_r x^{W-1} dx \\ &= w_r/W \end{aligned}$$

□

Step 3 - Larger diameter network: To enable frequency reuse in networks of a larger diameter, WSA must be able to assign the same slot to multiple nodes as long as they don't interfere with each other. Ideally, the decision to allocate a new time slot should involve only nodes that interfere with each other. Therefore, at any given node we use only the hash values computed for nodes that interfere with this node in determining whether or not it should transmit. As a result we ensure weighted fairness only between nodes within the same contention context. Globally the resulting allocation depends on (a) which other nodes a given node interferes with, (b) the interaction between multiple *partially overlapping* contention contexts and (c) the interaction of fairness constraints at each hop of a multi-hop flow.

Computing the set of nodes that interfere with a given node is difficult to achieve in a distributed manner. To get around this problem we make the assumption that a node can interfere with all the nodes within k-hop distance, where k is given. Thus, when a node wants to contend for a slot, it will broadcast its intention to all nodes within k hops. The set of nodes with which one node interferes is then the set of nodes from which it heard a broadcast message. The node will then consider only those nodes when deciding whether it has won the slot or not.

There is a clear trade-off between the level of interference and the bandwidth utilization in choosing the value of k. As the value of k increases, both the probability of interference and the utilization decrease. The reason why the utilization decreases is because, as k increases, a k-hop region will cover more and more nodes that do not interfere with each other in the real system. In this paper we assume two values of k, $k = 1$, which represents an optimistic assumption as the interference range is typically greater than the transmission range, and $k = 2$, which as we found in our experiments is a more conservative assumption.

However, interference regions can cause *race conditions*. It is possible that n_i thinks that n_j is the winner of a slot, but n_j does not transmit in that slot, since it knows another node n_l within k hops such that $H_l > H_j$. Thus, event though n_i and n_l do not interfere, n_i ends up not transmitting. We use two mechanisms to address this problem.

Inactivity timer: Assume $k = 1$. At the beginning of each slot if a node n_i thinks that node n_j is going to transmit in that slot, it initializes an inactivity timer to listen for transmissions from that node. If this timer expires and the node still hasn't heard any transmissions from n_j, it assumes that the slot is still free. If n_i has the next highest hash value after n_j it starts transmitting in that slot. When $k = 2$, nodes n_i and n_j are not necessary within one hop distance. In this case, nodes within one hop of both n_j and n_i will announce n_i that n_j is silent[2]. In practice, we set the inactivity timer to be the time it takes to transmit three maximum-sized packets. This helps us avoid false-positives due to packet losses.

Localized resolution: The above mentioned problem is more likely to occur when we resolve contention over a large number of partially over-lapping contention contexts in a slot. In Section 4.2.4, we present a mechanism that reduces the number of competing nodes, which in turn reduces the probability of race condition in the case of over-lapping regions.

4.2.4 Amortizing the cost of contention resolution

Even though our contention resolution mechanism is fairly light-weight, it is more expensive than the hardware based contention mechanisms used in 802.11, CSMA/CD, and RTS/CTS, respectively. For example, some fraction of a slot might be wasted due to the inactivity timer.

To amortize the cost of the contention resolution, in this section we present a simple mechanism that basically allocates a node more than one slot at once. A straightforward solution would be to allocate a number of consecutive slots to a node once it wins the competition. However, this would increase the delay experienced by a packet as a packet now needs to wait multiple slots before being transmitted. We next present a simple algorithm to address this drawback.

The idea is to form groups of N consecutive slots as shown in Figure 4. The *index* of a time-slot is defined as the position of the time-slot within a group of N slots. If a node is allowed to transmit in the slot with index i, the node is implicitly allowed to transmit in the index i of the next group

[2]To suppress multiple announcements from one-hop neighbors of n_j to n_i, we enforce that on all the nodes within 1-hop of both n_i and n_j, only the node with the highest hash value will notify n_i.

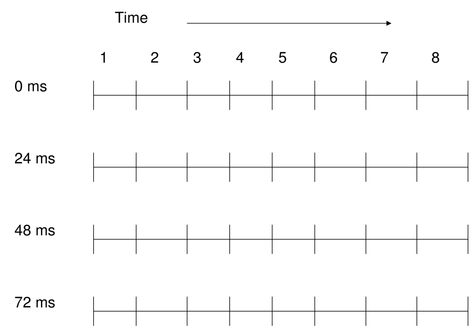


Figure 4: Groups of slots used in the overlay MAC (**draw a better figure**

with probability p. In other words, the node will relinquish slot i with probability $1 - p$. Once the node relinquishes the slot, other nodes are allowed to compete for the slot. This mechanism amortizes the cost of contention by $1/(1 - p)$.

Another advantage of this mechanism is that when a node relinquishes a slot, we only have to resolve contention among the nodes within k hops from it. Thus, not only do we use the contention-resolution mechanism less often, but we also reduced the likelihood that the hash-based mechanism will suffer inefficiency due to overlapping regions.

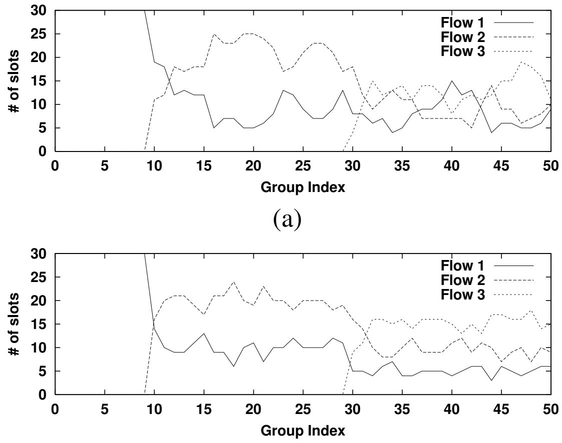
However, the reduction in the overhead does not come for free. A node needs now to wait for $1/(1-p)$ slots on average before it can compete. As a result it will take the system longer (*i.e.,* by a factor of $1/(1 - p)$) to converge to the fair allocation when a new node joins or leaves the competition.

To address this issue, we make a slight modification to our earlier definition of of H_i. Let o_i be the number of slots implicitly owned by the node n_i when contending for a timeslot. We now redefine

$$H_i = H(n_i, t)^{\frac{W o_i / N}{w_i^2}}$$

In other words, if the true weight of node n_i is w_i, node n_i uses a virtual weight $w'_i = \frac{w_i^2}{W o_i / N}$ instead of w_i. In steady state, $o_i \approx \frac{w_i N}{W}$ and hence $w'_i \approx w_i$. Thus, w'_i inflates a nodes weight when it has less than its fair share of slots, but diminishes its weight when it has more than its fair share. Thus, a new sender that becomes active will gain about $(1 - p)N$ slots in each round and quickly ramp up to its fair rate.

To illustrate the advantage of using an inflated weight w'_i, we use a simple simulation involving three flows with weights 1, 2, and 3 respectively. We assume $N = 30$, and $p = 0.95$. Figure 5 plots the number of slots allocated to each flow as a function of the group index. As expected,



(a)

(b)

Figure 5: Number of slots in every group when using (a) default weights (w_i), and (b) inflated weights (w'_i).

when using inflated weights allow new flows to reach their fair allocation faster. In addition, the fluctuations in steady state behavior are smaller than in the case of using default weights.

The choice of N impacts both the short-term fairness and the time a winner needs to wait to receive subsequent slots without competing again. At the limit, if $N = 1$, the winner is allocated $1/(1-p)$ consecutive time slots, which hurts the short-term fairness. If $N = \infty$, the winner is allocated one slot at a time as in the baseline algorithm. Based on our experimental results, we chose $N = 20$.

4.2.5 Reducing the control overhead

OML employs control messages to signal (a) when a node has relinquished a slot, and (b) which nodes enter the competition for a slot. Next, we present two simple optimizations to reduce the signaling overhead of OML.

First, we make use of another pseudo-random function H' to decide if a node should relinquish its slot, *i.e.,* if node n_i owns the slot t, it will give up the slot only if $H'(n_i, t) < (1-p)$. Again, H' is a pseudo-random function with range $(0, 1]$ and other nodes can compute this value without any communication.

Second, we include in the header of each packet the queue length at the sender. Thus, if the queue becomes empty during the current slot, the node's neighbors are implicitly notified, and the remaining node with the next highest H_i is allowed to transmit in the rest of the slot. If on the other hand the queue of the sender is not empty at the end of the time slot, then its one-hop neighbors will implicitly assume that the node will compete in the next time slot.

In summary, only nodes that join the competition need to transmit individual control messages. All the additional information about active nodes in the interference region is piggy-backed in the packet header. To deal with node failures, we remove a node from the list of contenders if we don't receive an update on its queue-length for an extended period of time.

4.3 Putting everything together

Figure 6 shows the WSA pseudocode. For readability, here we assume that the interference region is the same as the transmission region, *i.e.,* $k = 1$.

Each node maintains a list of all nodes in its interference region that are active, and a map of which time slot is allocated to which node. Since nodes are in promiscuous mode, each node can maintain the list of active nodes by simply inspecting the queue lengths in the headers of the packets it hears about. If a node does not hear from a neighbor for a predefined interval of time, it assumes that the neighbor is no longer competing and removes it from its list. This allows the algorithm to be robust in the presence of packet loss and node failures.

To implement a k $(= 2)$ hop interference region, a node simply piggybacks the information about all its one-hop neighbors in the packets it sends. This information is spread across the headers of all packets it sends during the slot to reduce the per-packet overhead. This allows each node to learn about its two-hop neighborhood. Similarly, when a node becomes active it broadcast a control message to its two-hop neighborhood.

Recall that the size of the interference region determines both the level of interference and the network utilization. The level of interference can be significantly higher for $k = 1$ than $k = 2$. While the MAC layer still resolves the contentions using CSMA/CD, contentions at the MAC layer reduce the level of control of OML on packet scheduling. On the other hand, the network utilization can be lower for $k = 2$ than $k = 1$, as there can exists nodes at two-hop distance that do not interference with each other in a real system.

5 Experimental Test-bed

In order to motivate our work in Section 3, we use some examples of actual observed performance in an 802.11 testbed. In this Section, we describe the test-bed that we use for these results. Our test-bed currently consists of 6 wireless nodes based on commodity hardware and software.

The hardware consists of booksized computers equipped with a 2.4GHz Celeron processor and 256MB of RAM. Each computer is also equipped with a Netgear WAG511[4] tri-mode PCMCIA wireless Ethernet adapter. This card is capable of operating in 802.11a,b and g modes, but we conduct all out experiments in 802.11a mode to avoid interference with the production 2.4GHz wireless network in our building.

```
function scheduler()
  while (TRUE)
    getSlotIndex(&slotId, &slotIndex);
    // check whether you already own the slot
    if (owner[slotIndex] ! = myAddress)
      setInactivityTimer();
    if (H'(slotId, owner[slotIndex]) < (1 − p))
      // contendForSlot updates owner[]
      contendForSlot(slotId, slotIndex);
    while (!endOfCurrentSlot())
      if (owner[slotIndex] == myAddress)
        p = getPacket(omlQueue);
        if (p) send(p);

function recvPacket(p)
  if (p.type == CONTEND or p.header.q_size > 0)
    active_list.add(p.src);
    // don't content for this slot
    cancelInactivityTimer();
    // active list may have changed; recompute winner
    computeWinner(slotId, slotIndex);
  if (p.type == RELINQUISH or p.header.q_size == 0)
    contendForSlot(slotId, slotIndex);
  if (p.header.q_size == 0)
    active_list.remove(p.src);
  if (p.type == DATA)
    // deliver packet locally if this node is
    // the destination; otherwise route it
    processData(p);

function contendForSlot(slotId, slotIndex)
  if (!isEmpty(omlQueue) and totalOwnedSlots == 0)
    // other nodes may not be aware this node is active
    sendContentionMessage();
  computeWinner(slotId, slotIdx)

function handleInactivityTimer()
  contendForSlot(slotId, slotIndex);

function cleanActiveList()
  // This function is called periodically by every node
  // to time-out failed nodes from the active_list
  for all n ∈ active_list
    if (curr_time - n.time_added > FAIL_TIMEOUT)
      active_list.remove(n);
```

Figure 6: The WSA algorithm for an one-hop interference region.

We have installed Linux (kernel 2.4.22) in all these systems along with the MadWiFi[1] driver for the wireless cards. For routing, we use the Click software router[20] because it provides an easily extensible modular framework. Also, the MIT Grid[2] project provides a readily downloadable implementation of DSR[19] and AODV[24] on top of Click.

6 Implementation

In this Section we describe how OML is implemented in our testbed. We start with a brief overview of Click and then describe the new *elements* we have implemented in Click. Finally, we show how these elements are used in a typical multi-hop wireless router configuration.

6.1 Overview of Click

The Click Modular Router[20] is a software architecture developed at MIT for building routers. Here, we only cover the essentials required to understand our implementation of OML. For further details, we refer the reader to [20].

Elements are the building block of a Click router. Elements typically perform some simple packet processing task and are implemented as a C++ class. Packets flow in and out of elements through *ports*. A router is a directed graph of elements connect through their ports using *push* or *pull connections*. In a push connection, data flow is initiated by the upstream element (*e.g.,* input to a queue), whereas in a pull connection, the downstream element is the initiator (*e.g.,* output from a queue). The configuration for the router which defines the vertices and edges of the directed graph is specified using a simple text file.

In addition to ports which deal with the data flow, elements can also implement *handlers*. Handlers are used to implement control operations outside the data flow, *e.g.,* a queue might export a handler that reports the current length of the queue. A downstream element that implements a scheduling algorithm will use this handler to decide which input to dequeue from.

6.2 OML Elements in Click

The majority of OML functionality is implemented using the two elements, *TimeSlotEnforcer* and *ContentionResolver*. *TimeSlotEnforcer* is responsible for making sure that send requests are issued only in the allowed time slots. The length of a slot (l) and the number of slots in a group (N) are specified through a configuration file. This element is included just before the packets are sent to the device in the router. It implements one *pull* input and one pull output. It does no packet manipulations, but the downstream request is forwarded up stream only if the node is allowed to send packets

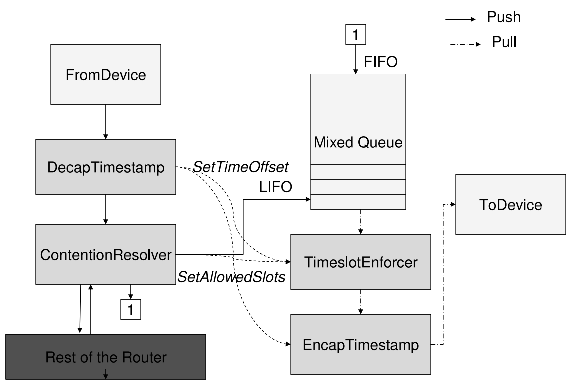


Figure 7: Click configuration for OML

in that slot. It also implements two handler functions. *TimeSlotEnforcer::SetAllowedSlots* takes a bitmap of length N as argument to notify the element about which slots are active. *TimeSlotEnforcer::SetTimeOffset* is used to set a clock skew relative to the system time for synchronization.

The *ContentionResolver* implements a bulk of the control signaling for OML. Every packet being sent to or received from the device flows through the *ContentionResolver*. This element manipulates packet headers as well as introduce new control packets for signaling. It has two *push* outputs, one for regular packets and one for high-priority control packets that go directly to the head of the output queue. It communicates with the *TimeSlotEnforcer* using the *SetAllowedSlots* handler.

The *EncapTimeStamp* and *DecapTimeStamp* elements handle clock synchronization. *EncapTimeStamp* implements a *pull* input and output and is connected to the output of *TimeSlotResolver*. The *DecapTimeStamp*element removes the header added by *EncapTimeStamp* and computes a new clock offset. It also implements the *DecapTimeStamp::GetTimeOffset* handler which is used by both *EncapTimeStamp* and *ContentionResolver*.

A typical router configuration that implements OML is shown in Figure 7. We do not show the elements of the routing protocol in this figure. The OML elements can be used either with the DSR or AODV elements developed in the GRID project or with the *LinearIPLookup* element to implement static routing.

7 Simulation Results

In this Section, we evaluate the benefits of OML by using Qualnet[5], a commercial packet-level wireless simulator. In particular, we use simulation to make three points. First, despite the additional control overhead, the throughput of an OML/WSA network is comparable to the throughput of an 802.11 network. This is because the loss in throughput due to the control overhead is offset by the fact that OML/WSA experiences much lower contention. Second, WSA significantly improves the fairness in a multi-hop network. In particular, while in a multi-hop 802.11 network, a significant percentage of TCP flows are shut-off, OML/WSA completely avoids this problem. Third, we show that WSA allows fine grained resource control by accordingly setting the weights of the nodes.

We next describe the simulation setting. Each node in the simulation is equipped with an 802.11a network interface operating at 6 Mbps. We use the outdoor two-ray propagation model which yields a radio range of about 350 m. In all experiments we hold the density of the network constant at 50 nodes per square km, and place the nodes at random locations. The time it takes to transmit an 1500 byte packet, including the overhead of the MAC and physical layers is about 2 ms. Hence, we choose a slot time of $l = 10$ ms, and a group with $N = 20$ slots. We have disabled the RTS-CTS handshake in all the simulations since it yields better performance both with and without OML. We use the AODV[24] routing algorithm, and run each simulation for one minute.

7.1 Collisions and Throughput

As mentioned earlier, OML/WSA trades off less frequency re-use and some control overhead in packet headers for fewer collisions and better fairness. To evaluate the effects of this trade-off on throughput, in this experiment we vary the size of the network from 15 to 50 nodes, and measure the throughput achieved by a number of simultaneous UDP flows. All flows originate at different nodes, but have the same sink. Such a traffic pattern models a multi-hop network used to access the Internet through a single gateway.

Figure 8 shows the effect of the size of the network on the total number of (a) DATA packet transmissions, (b) ACK transmissions and (c) successful unicast transmissions. For each network size we consider 10 simultaneous UDP flows. The difference between (a) and (c) gives the total number of packet retransmissions at the MAC layer. Of these (a)-(b) are due to the loss of DATA packets, and (b)-(c) are due to the loss of ACK packets. Because with OML we make the conservative assumption that all nodes in a two hop neighborhood interfere, there is less frequency reuse and the total number of transmissions attempted is lower than in 802.11. On the other hand, since OML does not allow competing senders to be active at the same time, there is hardly any MAC layer retransmission. In contrast, in the case of 802.11, up to 25% of the packets are retransmitted. Applications that use broadcast transmissions (*e.g.,* routing protocols) can greatly benefit from fewer collisions since broadcast packets are not acknowledged and retransmitted at the MAC layer.

Figure 9 plots the average throughput of 5 or 10 simul-

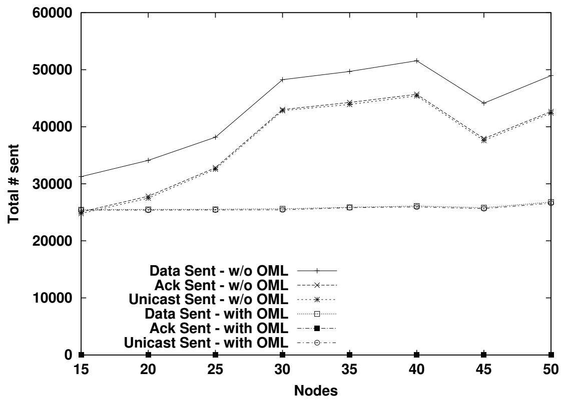


Figure 8: Packet retransmissions in a multi-hop network with 10 flows

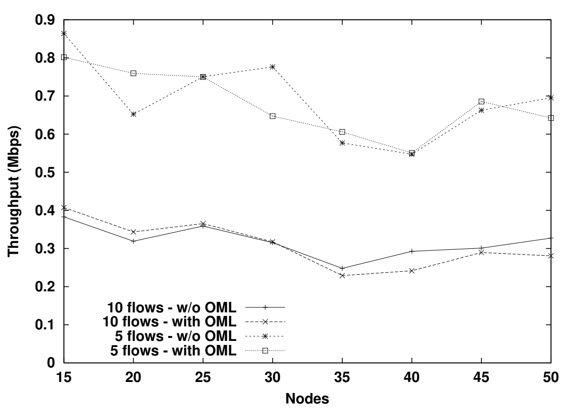


Figure 9: Average throughput in a multi-hop network with and without OML

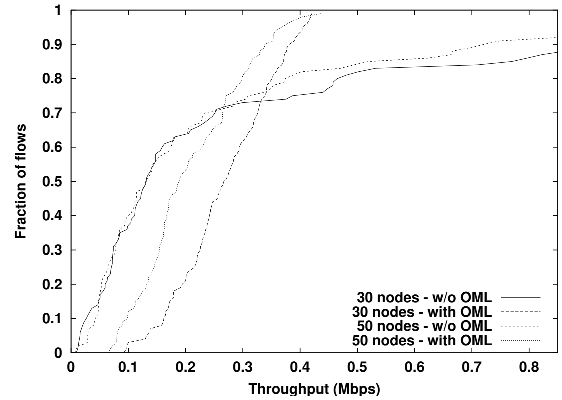


Figure 10: CDF of throughput received by flows in a multi-hop network

taneous flows both with and without OML. Even though OML uses additional control overhead permits less frequency reuse, the throughput with OML is pretty close to the throughput without OML. Also, it is important to note that since OML aims to provide better fairness, we devote more resources to flows that use longer routes. Thus, OML trades throughput for fairness in the case of longer routes as compared to 802.11.

7.2 Fairness

A well-known problem with multi-hop 802.11 networks is that short flows, in terms of number of hops, receive a much higher throughput than long flows. To show how OML/WSA can address this problem, in the following experiment we set the weight of each node to be equal to the number of unique IP source addresses seen in its output queue. Thus nodes that forward on behalf of other nodes will have a higher weight. Within a single node, we maintain separate queues for each IP source address and implement round-robin scheduling between these queues. If all nodes contend with each other, the weight allocations and the scheduling algorithm will ensure that all flows receive equal throughput irrespective of their length.

Figure 10 shows the cumulative distribution function (CDF) of the throughputs of 10 simultaneous flows averaged over 10 simulation runs. We report results for both a 30 node and a 50 node network. As shown in Figure 10, in the case of 802.11 several long flows are shut-off, about 40% of the flows receive less than 100 kbps. In contrast, the throughputs of flows under OML/WSA are more evenly distributed, with only 2% or 10% of the flows below 100 kbps in the 30 node case and the 50 node case respectively. This is at the expense of the high bandwidth flows getting less bandwidth than in the case of 802.11. However, we believe that this is the right trade-off as *connectivity* is the most important service provided by a communication network.

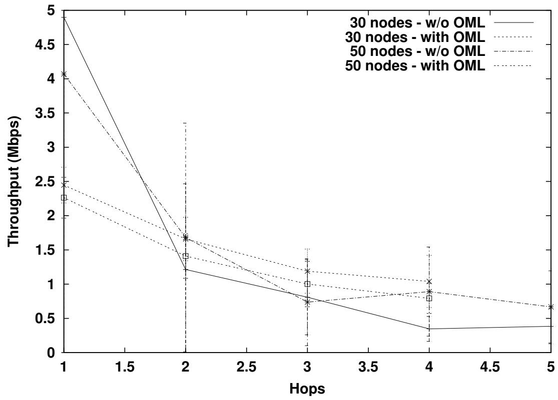


Figure 11: Relationship between throughput and the number of hops

In Figure 11 we plot the average throughput of a flow as the function of the number of hops. As expected, 802.11 favors shorter flows, but OML/WSA is more fair. For example, in the 30-node case, four-hops flows receive less than 10% of the throughput received by one-hop flows on average. With OML, four-hops flows receive more than half the throughput of 1-hop flows.

7.3 Flexibility

In this section, we demonstrate the flexibility of OML/WSA using two examples.

7.3.1 Weights for nodes

In this experiment, we assign to each flow a random weight in the set $\{1, 2, 3\}$. The weight of a node is equal to the sum of the weights of all flows which have queued packets at that node. At each node, OML serves the flows using a weighted round robin algorithm. Figure 12 shows the CDF of the flows of different weights over 10 simulation runs, with 10 flows in each run. The allocation for a flow not only depends on its weight, but also on which other flows it competes with. Hence, the allocation is not perfectly in accordance with the weights of the flows. However, we can see that flows of higher weight are much more likely to receive a higher throughput than flows of lower weight.

7.3.2 Bandwidth allocation by nodes or by flows

In the previous examples, we have considered only the case in which at most one flow originates at a node. Next, we consider a simulation scenario in which 10 flows originate at only 4 nodes, where one flow originates at node 1, two flows originate at node 2, and so on. To illustrate the ability of OML/WSA to control the bandwidth allocation we consider two cases: (a) each node has the same weight, (b) the weight

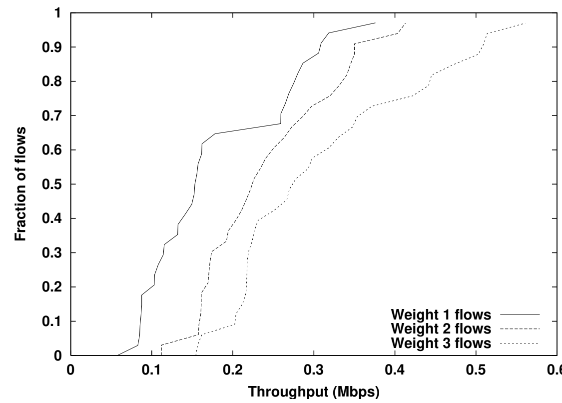


Figure 12: CDF of throughput received by flows with different weights

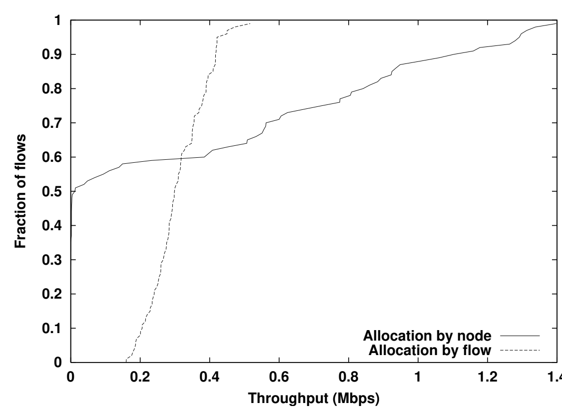


Figure 13: CDF of throughput received by flows when using per-flow and per-node bandwidth allocation

of a node is equal to the number of flows it handles. Figure 13 plots the CDF of the throughput received by each flow in the two cases. In case (a), even though the bandwidth is divided fairly among nodes, the throughputs received by the flows differ widely with almost 50% of the flows being virtually shut-off, and some flows receiving close to 10 Mbps. In contrast, in case (b), no flow is shut-off, with all flows getting a throughput between 1 and 3.5 Mbps.

8 Results from the testbed

In this section, we evaluate OML/WSA using the experimental testbed described in Section 5. The main results in this section can be summarized as follows. Section 8.1 shows that OML/WSA can increase the overall throughput of the system by fairly allocating transmission times, instead of opportunities of transmission like in the 802.11 protocol. Section 8.2 shows that even in a simple chain topology involving one-hop flows, 802.11 can cause a significant number of TCP flows to experience starvation, while OML avoids this phenomenon. Section 8.3 shows that, as expected, the starvation problem is even worse in the case of multi-hop routing, but OML can still handle this problem. Finally, in Section 8.4, we use a simple setting involving two flows to evaluate the allocation accuracy of WSA.

8.1 Heterogeneous data rates

In this experiment, we re-consider the scenario described in Section 3.1, where two nodes are simultaneously sending one TCP flow each. One node operates at 54 Mbps, and the other node operates at rates ranging from 6 to 54 Mbps. We use the WSA algorithm to allocate the bandwidth, with each node having the same weight. This leads to each node receiving an equal channel-access time, rather than an equal number of transmission opportunities like in 802.11. Note that this allocation implements the temporal-sharing policy as proposed in [28]. Figure 14 shows the throughputs of both flows. The two flows receive throughputs approximately proportional to the rates they are operating at, and the total system throughput drops less than in the case of 802.11 when the second node is operating at 6 Mbps.

8.2 Chain topology

In this experiment, we configure our test-bed as a five-hop chain, and start two simultaneous one-hop flows along random links in this chain. Each node in the chain operates at 6 Mbps. Figure 15 shows the CDF of the throughput of these flows based on 50 trials. We consider three scenarios: (a) baseline 802.11 (without OML), (b) OML assuming one-hop interference regions, and (c) OML assuming two-hop interference regions. When using OML each flow is assigned a unit weight,

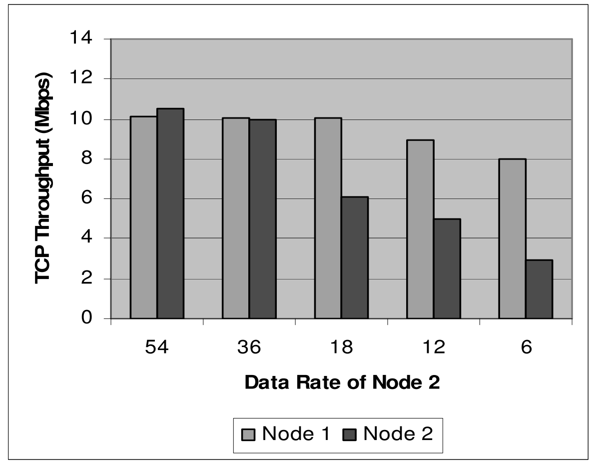


Figure 14: 802.11 throughput in the presence of heterogeneous data rate senders using OML

Without OML, in 26% of all the cases, one of the two flows is not even able to establish a TCP connection. This is shown in Figure 15, where 13% of all flows have zero throughput. Furthermore, about 20% of all flows receive less than 1.5 Mbps. In contrast, with either version of OML only about 5% of the flows have a throughput below 1.5 Mbps. When using two-hop interference regions, no flow receives less than 0.3 Mbps, and over 85% of the flows receive between 2 and 2.6 Mbps. On the other hand, when using one-hop interference regions, there are far more flows with higher throughputs: around 22% of the flows have throughputs around 4.5 Mbps. These results illustrates the trade-off between using one-hop and two-hop interference regions. Using two-hop interference regions lead to a more conservative spatial reuse of the channel. Hence, there are cases in which two nodes cannot transmit simultaneously even though they do not interfere in the real system. In contrast, using one-hop interference regions lead to higher throughputs, but at the cost of hurting the fairness. Indeed, as shown Figure 15, in this case there are significantly more flows which experience low throughputs than when using two-hops interference regions.

8.3 Multi-hop routing

In this section, we study how OML can improve the flow throughputs in a multi-hop routing network. For this purpose, we use a simple Y-shaped topology of the nodes as described below. Flows A and B originate at nodes 1 and 2, respectively, and terminate at node 4, after being routed via node 3. Flow C is an one-hop flow from node 5 to node 4. The weight of each node is proportional to the number of

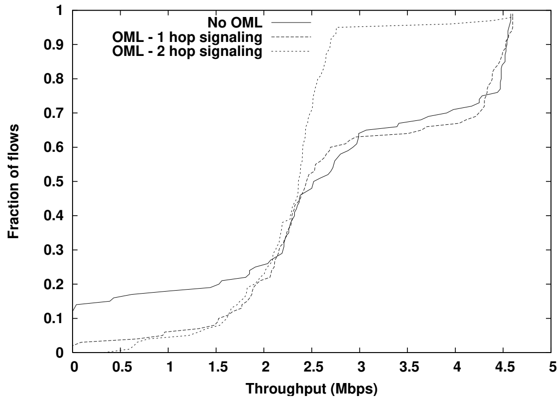


Figure 15: CDF of throughput of 1-hop flows in the network

Active Flows	Throughput (Mbps)		
	Flow A	Flow B	Flow C
A&C w/o OML	0.14		4.40
A&C with OML	1.39		2.56
B&C w/o OML		0.06	4.48
B&C with OML		1.01	2.58
A&B w/o OML	1.31	1.65	
A&B with OML	1.11	1.11	
A,B&C w/o OML	0.11	0.03	4.42
A,B&C with OML	0.86	0.54	2.57

Table 2: Interaction of flows in an ad-hoc topology

flows it sees: the weights of nodes 1, 2, and 5 are one, the weight of node 3 is two, and the weight of node 4 is three. In this experiment, OML assumes two-hop interference regions.

When only one of the three flows is active, flows A and B receive about 2.24 Mbps, while flow C receives 4.55 Mbps, with or without OML.

Table 2 shows the throughputs of each flow when more than one flow is active. Without OML, flow C effectively shuts-off the two-hop flows. In contrast, OML allocates to each flow at least 1 Mbps, when only one of the two-hop flows is active, and at least 0.54 Mbps, when all flows are active. The reason why the throughputs of the two-hop flows are not higher is because the two-hop interference assumption is violated. According to this assumption, either node 1 and 2 can send data simultaneously with node 5 without interfering to each other. However, this assumption did not hold at the time this experiment was conducted. Let flow A' be the one-hop flow from node 1 to 3. When flows A' and C were simultaneously active, C received 4.53 Mbps whereas A' received only 0.69 Mbps.

Weight of A	1	1	1	1
Throughput of A	2.26	1.48	1.36	0.92
Weight of B	1	2	3	4
Throughput of B	2.20	2.97	3.06	3.64

Table 3: Throughput received by senders with different weights

8.4 Weighted allocation

In this experiment, we evaluate the accuracy of WSA, and thus its ability to provide per-node service differentiation by manipulating the node weights. We consider two nodes A and B that send a TCP flow each to a third node C. We set the weight of node A to 1, and assign a weight ranging from 1 to 4 to node B. Table 3 shows the throughput achieved by the nodes for each combination of weights. As expected, the ratio of throughputs obtained by the two nodes tracks closely the ratio of their weights.

9 Open Issues

The current design of OML should be viewed as a first iteration. As we gather more experience with using OML, we expect the OML design to evolve significantly. For example, in this paper we have considered only the performance of long-lived TCP flows. We expect that short-lived flows to put additional stress on OML. Such flows will affect the accuracy of the *active_list* which in turn will lead to additional control messages. We plan to extensively evaluate OML with web-like traffic load. Another potential problem is that OML can introduce additional packet delays as a node is allowed to send packets only during its allocated time slots. We plan to evaluate to what extent these delays affect interactive applications. OML does not work well in the presence of unmodified 802.11 networks. In this case, *all* OML nodes will receive an aggregate bandwidth equivalent to that of a *single* 802.11 node. One solution to address this problem would be to detect when there is competing traffic, and then allow multiple OML nodes to send data in the same time slot. Finally, in multi-hop networks the interplay between the flow constraints and the interference constraints makes it difficult to precisely formalize the notion of fairness achieved by WSA. We plan on further studying how to define and achieve weighted fairness in the presence multiple partially overlapping contention contexts.

10 Conclusions

In this paper, we have described the design and the implementation of an overlay MAC layer (OML) solution which addresses some of the limitations of the 802.11 MAC layer.

By using both simulation and experimental evaluation, we show that while 802.11 may cause TCP flows to experience starvation, OML can avoid this problem. In addition, we show that OML can reduce the contention in the network, and provide service differentiation among nodes, with relatively low control overhead.

The power of the OML approach is that it allows us to implement MAC layer functionality *without* modifying the existing 802.11 protocol. In this respect, our approach is reminiscent of the overlay network solutions that aim to implement network layer functionality such as resilient routing on top of the existing IP layer. Ultimately, OML would enable us to experiment with new scheduling and bandwidth management algorithms, and evaluate their benefits to the existing applications, before implementing these algorithm in the MAC layer.

References

[1] MadWifi. http://madwifi.sourceforge.net/.

[2] MIT grid project. http://www.pods.lcs.mit.edu/grid/.

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

[4] Netgear. http://www.netgear.com/.

[5] The Qualnet Simulator from Scalable Networks Inc. http://www.scalable-networks.com/.

[6] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, Oct. 2001.

[7] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *SIGCOMM*, 2002.

[8] L. Bao and J. Garcia-Luna-Aceves. Hybrid channel access scheduling in ad hoc networks. In *Tenth International Conference on Network Protocols (ICNP)*, 2002.

[9] L. Bao and J. J. Garcia-Luna-Aceves. Distributed dynamic channel access scheduling for ad hoc networks. In *Journal of Parallel and Distributed Computing*, 2002.

[10] P. Bhagwat, B. Raman, and D. Sanghi. Turning 802.11 inside-out. *SIGCOMM Comput. Commun. Rev.*, 34(1):33–38, 2004.

[11] V. Bharghavan, S. Lu, and T. Nandagopal. Fair queueing in wireless networks: Issues and approaches, 1999.

[12] S. Biswas and R. Morris. Opportunistic routing in multi-hop wireless networks. *SIGCOMM Comput. Commun. Rev.*, 34(1):69–74, 2004.

[13] Y.-H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *ACM SIGMETRICS 2000*, pages 1–12, Santa Clara, CA, June 2000. ACM.

[14] I. Cidon and M. Sidi. Distributed assignment algorithms for multihop packet radio networks. *IEEE Trans. Comput.*, 38(10):1353–1361, 1989.

[15] D. S. J. D. Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. In *Proc. of ACM Mobicom 2003*, 2003.

[16] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Symposium proceedings on Communications architectures & protocols*, pages 1–12. ACM Press, 1989.

[17] J. Elson and D. Estrin. Time synchronization for wireless sensor networks. In *PDPS Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing*, pages 186–186.

[18] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda. Performance anomaly of 802.11b. In *Proceedings of IEEE INFOCOM 2003*, San Francisco, USA, March-April 2003.

[19] D. B. Johnson, D. A. Maltz, and J. Broch. Dsr: The dynamic source routing protocol for multihop wireless ad hoc networks. In *Ad Hoc Networking*, 2001.

[20] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, 2000.

[21] C. Liu, Y. Ge, J. Hou, M. Fitz, W.-P. Chen, and R. Jain. Osu-mac: A new, real-time medium access control protocol for wireless wans with asymmetric wireless links. In *21st International Conference on Distributed Computing Systems*, 2001.

[22] L. M. S. C. of the IEEE Computer Society. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. *IEEE Standard 802.11*, 1999.

[23] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Trans. Netw.*, 1(3):344–357, 1993.

[24] C. Perkins. Ad hoc on demand distance vector (aodv) routing, 1997.

[25] L. Pond and V. Li. A distributed time-slot assignment protocol for mobile multi-hop broadcast packet radio networks. In *MILCOM*, 1999.

[26] K. Romer. Time synchronization in ad hoc networks. In *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, pages 173–182. ACM Press, 2001.

[27] B. Sadeghi, V. Kanodia, A. Sabharwal, and E. Knightly. Opportunistic media access for multirate ad hoc networks, 2002.

[28] B. Sadeghi, V. Kanodia, A. Sabharwal, and E. Knightly. Opportunistic media access for multirate ad hoc networks. In *Proc. of ACM Mobicom 2002*, Sep. 2002.

[29] U. Schmid and K. Schossmaier. Interval-based clock synchronization. *Real-Time Systems*, 12(2):173–228, 1997.

[30] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *Proceedings of ACM SIGCOMM 2002*, August 2002.

[31] C. A. Waldspurger and W. E. Weihl. Lottery scheduling: Flexible proportional-share resource management. In *Operating Systems Design and Implementation*, pages 1–11, 1994.