EFLoW: End-to-end Fairness using Local Weights

Ananth Rao
Ion Stoica

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2007-107
http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-107.html

August 23, 2007

Acknowledgement

We would like to thank Prof. Robert Morris at MIT for letting us use their wireless testbed. We would also like to thank Karthik Lakshminarayanan, Rodrigo Fonseca and Matt Caesar for their useful comments.

EFLoW: End-to-end Fairness using Local Weights in Wireless Networks

Ananth Rao
University of California - Berkeley
Email: ananthar@cs.berkeley.edu

Ion Stoica
University of California - Berkeley
Email: istoica@cs.berkeley.edu

***Abstract*—Recent research indicates that multihop wireless networks can suffer from extreme imbalances in the throughput achieved by simultaneous competing flows. However, even with global knowledge of the topology and traffic patterns, computing the fair allocation is know to be an NP complete problem for most definitions of fairness. Previous work has either (a) focused on the MAC layer which only provides fairness guarantees within a single contention neighborhood or (b) used a centralized coordinator to compute the global allocation of throughput to flows. While the former approach does not address the problem in its entirety, the latter is very hard to implement under dynamic channel and traffic conditions. In this paper, we bridge this gap by augmenting approach (a) with a simple distributed transport-layer algorithm called *EFLoW*. *EFLoW* assumes that the MAC layer supports weighted fair allocation among nodes that directly compete with each other. We refer to the weights at this layer as the *local weights* since they have significance only within a single contention neighborhood. *EFLoW* computes the local weights using an iterative increase-decrease algorithm which guarantees convergence to an end-to-end max-min fair allocation under certain assumptions. In each iteration, *EFLoW* only uses state obtained from within a given node's contention region. We have implemented *EFLoW* in both a simulator and a real system on top of the Overlay MAC Layer (OML). Our results show that *EFLoW* can avoid starvation of flows and improve fairness by about 90% with only a 15% reduction in total network throughput when compared to standard 802.11.**

I. Introduction

Multi-hop wireless networks has been a subject of research for many years now. Initially, the primary motivating applications for such networks revolved around emergency response systems and military scenarios. However, the proliferation of the 802.11 standard and the availability of inexpensive hardware has spurred interest in a new application called "mesh networks." Many companies [1], [2] are considering using multi-hop wireless networks as an alternative to DSL for providing last mile connectivity to the Internet for customers. Such networks can also be very useful in the context of emerging markets and developing countries, where little to no infrastructure is available [3].

However, both simulations and deployments based on the current generation of hardware (mostly based on the 802.11 standard) show very poor fairness between competing flows. In fact, the fairness problem can be so severe that some flows are completely shut out from sending any data at all [4], [5]. This translates to outages in connectivity in the case of mesh networks. As a result, the connectivity of a particular node to the Internet might be affected in presence of flows from other nodes.

Both the Medium Access Control (MAC) layer and the transport layer play an important role in determining the fairness achieved by a multi-hop wireless network. The MAC layer controls how the nodes within a contention region share access to the transmission medium. Previous work on fairness at the MAC layer has focused on allowing higher layers control over how transmission opportunities are shared between these nodes, typically by assigning weights to each node. For example, [6] and [7] propose contention based schemes where the back-off algorithm is modified based on the weight assigned to a node. Recent research [4], [8] propose time-division multiplexed (TDMA) schemes where the number of slots assigned to a node is proportional to its weight. However, the MAC layer is oblivious of end-to-end flows; it is the job of the higher layers to compute these *local* weights in such a way that the desired allocation is achieved. While this is straight forward in the case of single-hop networks, it is known to be an NP-complete problem in multi-hop networks [9] due the interaction between flow constraints (the allocation at each hop of a flow must be the same) and interference constraints (no two competing nodes can transmit at the same time).

Previous work focusing on end-to-end fairness at the transport layer [9], [10] has typically relied on a centralized coordinator to compute the global allocation for all flows. This allocation depends on both the capacity of links in the network and the configuration and bandwidth demands of flows. In a mesh network, changes in either of the above can trigger frequent updates (which require communication with the coordinator) for a number of reasons. Firstly, measurement studies [11] have shown that the capacity of links, which depends on the current Signal-to-Noise Ratio (SNR), varies on short time scales and is hard to predict. Secondly, the workload primarily consists of HTTP flows which arrive and depart frequently. Finally, in the case of multi-hop flows, we must ensure that the allocation of bandwidth at each hop is consistent. Thus, changes in capacity or demands at any intermediate hop of a flow will trigger changes at every hop in the flow. This leads to a cascading effect and can quickly lead to reconfiguring the entire network.

In this work, we develop a distributed transport layer algorithm called *End-to-end Fairness using Local Weights* (*EFLoW*), which can be used in conjunction with any MAC

scheme with support for weighted-fair allocation within a contention neighborhood. We show that by using a simple additive-increase multiplicative-decrease algorithm to compute these weights, we can obviate the need for an expensive control protocol or global reconfiguration. Our control protocol is lightweight and involves only exchanging information between nodes in the same contention region. Under certain assumptions, we show that *EFLoW* converges to a Max-Min fair allocation of bandwidth to flows. We also implement and evaluate *EFLoW* in both simulator and a real test-bed using 802.11 radios and the Overlay MAC Layer (OML) [4] which enables us to perform time slot scheduling on top the 802.11 MAC.

The rest of this paper in organized as follows. In Section II, we present an overview of related work on the MAC layer and on achieving fairness in wireless networks. In Section III we define Max-Min fairness and outline some of the challenges in achieving this in wireless networks. In Section IV, we describe *EFLoW*, a simple way of achieving the above mentioned definition of fairness assuming support for local weights from the underlying layer. Section V presents details regarding the Weighted Slot Allocation (WSA) algorithm and the Overlay MAC Layer (OML), which we use to implement *EFLoW* in a real test-bed. Results from our evaluation, using both simulations and the test-bed are presented in Section VI. Finally, we conclude our paper and mention possible directions for future research in Section VII.

II. Related Work

Both the MAC and the transport layer play an important role in determining the fairness achieved by a wireless network. Most MAC layer proposals address fairness amongst directly competing nodes. There are several ways to achieve this depending on whether the MAC is contention-based or time-slot based.

Contention-based algorithms: Contention-based algorithms are based on carrier-sensing and random access. The basic idea behind CSMA/CA is that nodes independently pick a back-off window at random and the node that picks the lowest back-off wins access to the medium. Collisions can occur due to nodes picking the same back-off window, or due to hidden terminal problems in multi-hop networks. Several proposals aim to achieve fairness by controlling how the back-off window is chosen by different nodes. Examples of this approach include [6], [7], [12].

Time-division multi-plexing(TDMA): In TDMA approaches, time is divided into slots and and only nodes that don't interfere with each other are allowed to transmit during a given slot. The schedule is pre-computed in a centralized or distributed manner depending on the approach. Examples in literature include [8], [13]. These approaches require knowledge of the interference patterns, and can easily solve the hidden terminal problem. However, the schedule has to be recomputed every time the quality of a link changes or when flows arrive or depart.

However, the MAC layer is oblivious of end-to-end flows. *EFLoW* augments the mechanisms provided in these MAC layer proposals to achieve per-flow fairness at the higher layers.

End-to-end fairness: In [14], the authors propose an adaptation of the fair queuing algorithm where each node adds its virtual time to the header of each packet it sends. This approach works well for the wireless-LAN environment where all nodes can hear each other. However, for multihop networks, since a given node can be part of a large number of contention contexts, it is not clear which virtual time to use at that node.

Inter-TAP Fairness Algorithm (IFA) is a scheme proposed by Gambiroza et al. in [9]. Their definition of fairness suffers from two drawbacks. Firstly, they do not show how to compute the correct allocation in the general case even with complete knowledge of link qualities and interference patterns. Secondly, due to the ingress aggregation constraints, their scheme prevents nodes from using the available bandwidth efficiently. When there are two flows originating at a node, one experiencing very little contention and the other a lot of contention, the definition compares the total throughput of both flows with that of other nodes. This introduces an artificial dependence between the two flows and prevents the node from sending data on the flow experiencing little contention so that its other flow is not starved by the network.

So far, the IFA algorithm has not been implemented completely in a simulator or in a test-bed. The architecture proposed in [9] requires (a) measurement of every link capacity and interference by each node and (b) a centralized coordinator that will gather complete information about the entire network and compute the fair allocation (which is itself possible only for special cases). Such an architecture is hard to realize especially when link qualities and traffic patterns are changing rapidly.

In [10], Yi et al. propose to implicitly limit the TCP flow rates by delaying the ACKs at intermediate nodes to achieve fair bandwidth allocation. However, much like IFA, computing the correct rate requires centralized co-ordination in their approach also. This can be tricky under dynamic conditions.

Lu et al. propose a distributed mechanism to coordinate the back-off window between the nodes in a contention-based MAC to achieve end-to-end fairness [15]. However their analysis assumes that the interference range and the carrier-sense range are the same and that the effect of packet losses due to collisions is negligible. Measurement of 802.11 based hardware [16] shows that the interference range is typically much greater than the carrier-sense range. Also, recent work [5] shows that all contention-based MACs are susceptible to a generic co-ordination problem in multi-hop networks. In practice, the resulting inefficiency due to collisions may be severe enough to cause flow throughputs to span several orders of magnitude. *EFLoW* is not tied to any particular mechanism at the MAC layer and can avoid this problem by using a TDMA-based solution.

Ee and Bajcsy propose new transport layer algorithms for congestion control and fairness in [17]. Their approach is

fully distributed and ensures that all nodes get to transmit approximately the same number of packets to the destination. However, it is targeted at sensornets and supports only one specific traffic pattern; a tree where the root is the only sink. It cannot easily be generalized to less restrictive traffic patterns.

III. Max-Min fairness

There are many different definitions of fairness used in wireless network literature [18]. Though *EFLoW* could possibly be adapted to other definitions of fairness, for simplicity and to focus on a single goal, we chose to implement Max-Min fairness. The key idea behind Max-Min fairness is that we should not provide more throughput to any flow if it prevents us from giving at-least as much throughput to other flows [18]. Weighted Max-Min fairness is a generalization of this approach with support for weights, and is formally defined below. A similar definition was used in [15].

A. Max-Min Fairness

Consider a network with n flows. Let f_i be the throughput received by the i-th flow. The allocation is said to be Max-Min fair if for each flow i, an allocation of $f_i + \epsilon_1$ is possible only by reducing the allocation of another flow j to $f_j - \epsilon_2$ where $f_j < f_i$. On the other hand, if we can find a subset H of flows $\{m_1, m_2, \ldots, m_k\}$, which all have throughput higher than f_i, and if we can increase the throughput of f_i by only decreasing the throughput of flows from H, then the allocation is not fair.

Informally, Max-Min fairness states that inequality in throughput of flows is allowed under the condition that penalizing the richer flows wouldn't provide any benefits to the poorer flows. On the other hand, if we can provide better service to any flow by only penalizing flows that receive a higher throughput, then the allocation is not max-min fair.

The above definition does not provide any means to differentiate service between different flows. A simple method to address this limitation is to associate a weight with each flow. When comparing two flows, we always check if their throughputs are in proportion to their weights. More formally, if flow f_i is assigned a weight of w_i, we make a simple change to the above definition by modifying the condition that $f_j < f_i$ to $f_j/w_j < f_i/w_i$. Similarly, for all $t : f_t \in H$, $f_t/w_t > f_i/w_i$.

To illustrate this definition, consider the example in Figure 1, which shows a network with three flows (f_1, f_2, f_3). Assume a one-hop interference model *i.e.,* two links interfere iff they share an end-point. Then, (a), (b) and (c) are three possible allocations in terms of the fraction of time for which the source is active. It is easy to see that (a) is unfair because we can increase f_1 without decreasing any other flow's allocation. Similarly, (b) is not fair because we can increase f_3 by decreasing f_2 and $f_2 > f_3$. Finally, (c) is fair even though f_1 receives twice the throughput of the other flows. This is because node C is already saturated and we cannot increase f_2 or f_3 even if we decrease f_1.

Weighted Fair Queueing (WFQ) [19] is a popular method for providing Max-Min fairness when all flows compete for the same resource. WFQ states that throughput must be assigned to flows in proportion to their weight. If the allocation to any flow is more than its demand, the surplus is again divided among the rest of the flows in proportion of their weights. This process continues until there is either no surplus capacity or when all demands are satisfied. In wired networks, per-hop WFQ guarantees end-to-end Max-Min fairness. But because of the shared nature of the medium, this is not true in wireless networks.

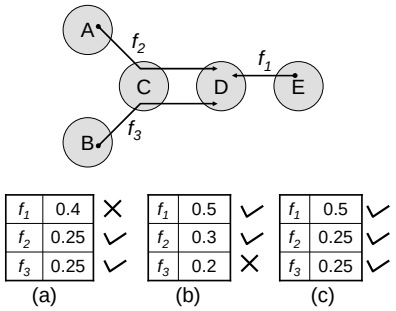

Fig. 1. Example for Max-Min Fair allocation

IV. Achieving End-to-end Fairness

In this section, we describe *EFLoW* , a distributed transport layer algorithm that works on top a MAC layer that provides the higher layers control over resource allocation within a single contention neighborhood. More specifically, we assume that the MAC layer allows the transport layer to dynamically associate a weight with each node, and allocates transmission opportunities to each node in a contention region proportional to its weight by implementing a Weighted Fair Queueing (WFQ) discipline [19]. We believe that this function is ideally implemented as part of the MAC layer since it is responsible for arbitrating channel access between directly competing nodes.

We use the term *local weight* to refer to the weights at the MAC layer. They are only used to resolve conflicts between directly competing nodes. In contrast, the weights defined in Section III are assigned to flows (not nodes or links) and have global significance. In Section IV-A, we explain why achieving end-to-end fairness is hard even when the MAC provides support for local fairness. We then present the design of *EFLoW* in Section IV-B.

A. End-to-End Fairness in Multi-hop Networks

In this section, we explain how we can leverage the support for local weights at the MAC layer to achieve end-to-end fairness. We show that

- We can assign local weights in such a way that the end-to-end allocation is Max-Min fair.
- To the compute the correct local weights, we need global knowledge of the topology and traffic demands.

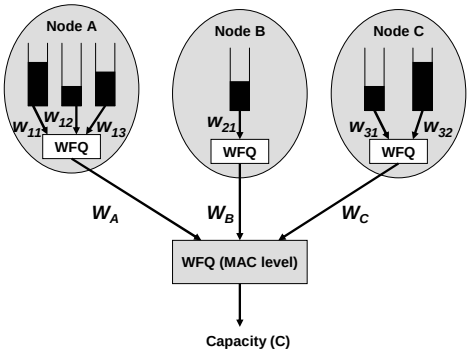

Fig. 2. Scheduling within a single contention neighborhood

For ease of explanation, we first consider a network in which all nodes are in the same contention region. We generalize this to a true multi-hop network in the next step.

1) Stage 1: Single Contention Region: Consider a simple network consisting of three nodes (A, B and C), where there are up to three flows originating at each node (Figure 2). Scheduling happens at two levels in our system. First, within a node, we have to decide which packet will be sent to the wireless interface. Second, at the MAC layer, we have to decide which node gets access to the medium. Unless otherwise specified, we assume that a node uses WFQ to schedule its flows.

Since all nodes compete with each other, ideally we would like to achieve the functionality of a single WFQ system with six queues. To achieve max-min fairness, we have to assign to each node a local weight that is the sum of the weights of all its *backlogged* flows. Note that a flow f is said to be *backlogged* at a node N if the arrival rate of f at N is greater than its service rate. For instance, the weight of node A, W_A, should be computed as $W_A = w_{11}+w_{12}+w_{13}$, assuming that all flows of node A are backlogged. In this case, the allocation to the first flow is given by,

$$\begin{aligned} f_{11} &= \frac{w_{11}}{w_{11}+w_{12}+w_{13}} \times (\text{Alloc. to A at MAC}) \\ &= \frac{w_{11}}{w_{11}+w_{12}+w_{13}} \times \frac{W_A}{W_A+W_B+W_C} \times C \\ &= \frac{w_{11}}{W_A+W_B+W_C} \times C \end{aligned}$$

Thus, computing a schedule at the MAC layer requires the knowledge of the demands of the all the six flows as well as the capacity C at the MAC layer.

2) Stage 2: Mutli-hop Network: To illustrate the additional challenges that arise in a multi-hop network, we revisit the example topology in Figure 1. Assuming a one-hop interference model, Figure 3 shows a network in which (a) there are multiple contention regions and (b) the same node (node C) is a part of multiple contention regions. The final allocation to node C is the *minimum* of its allocation from regions 1 and 2. So if we assume that region 1 is more congested, it will determine the final allocation at C. This in turn means that the queues in C will appear as backlogged in region 1, but as not backlogged in region 2. Because of this dependence, we also need the information from region 1 to compute the local weights for nodes in region 2.

To further complicate matters, the arrival rate of a multi-hop flow at a node depends on its service rate at the previous hop. This introduces a dependence between contention regions that may be far away from each other in the network topology. Because of the cascading effect of these dependencies we cannot determine the local weights without complete knowledge of the state of the network. This includes the topology, the interference patterns, routes and the flow demands all of which can change fairly frequently. In fact, since computing the Max-Min fair allocation itself is NP complete [9], computing the weights is also an NP complete problem[1].

Next, we present a *decentralized* algorithm that approximates the ideal allocation without requiring global state. At any given node, our algorithm uses only information gathered from its contention region, i.e., from nodes it directly competes with.

B. EFLoW

The key idea behind our algorithm is that instead of trying to compute the correct local weights directly, we use an iterative process that brings us closer to correct assignment in each step and oscillates around it in steady state. The next theorem provides the theoretical building block of our design by providing an alternate characterization f max-min fairness.

Definition 1. *If flow f_1 is backlogged at node N_1, we define this backlog to be a* permissible backlog *iff for every flow f_2 passing through node N_2 that contends with N_1, the service rate of f_2 at N_2 is no greater than the service rate of f_1 at N_1 (in proportion to the weights of f_1 and f_2).*

Theorem 1. *The allocation is Max-Min fair if and only if every node where any flow is backlogged is a permissible backlog.*

Proof:

For ease of explanation, in this proof we assume that every flow has unit weight (the proof can easily be generalized to non-unit weights). The proof is by contradiction.

$\Longrightarrow$: Assume the contrary, *i.e.,* there is a flow f_1 backlogged at N_1 and a flow f_2 through N_2 within the contention neighborhood of N_1 being serviced faster at N_2. Since N_1 and N_2 directly compete with each other, we can increase N_1's transmit rate by decreasing N_2's transmit rate. Thus we can increase f_1's allocation at N_1 by decreasing the f_2's allocation at N_2. But since f_2 is being serviced at a higher rate, this violates the definition in Section III.

$\Longleftarrow$: We first make the observation that in order to increase the allocation f_1 of a flow, we have increase its allocation at

[1]Given the local weights, we can directly compute the resulting allocation in linear time

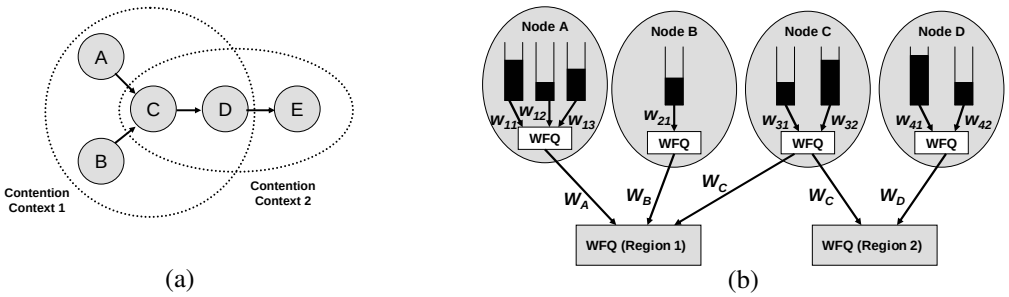

Fig. 3. Multiple contention regions in a multi-hop network

at least one node N_1 where it is backlogged. In order to do this we have decrease the transmit rate at at least node N_2 which competes directly with N_1. But since f_1 at N_1 is a permissible backlog, no flow at N_2 is being serviced faster than f_1 at N_1. Therefore, we cannot decrease the allocation of any flow at N_2 without violating the definition of Max-Min fairness in Section III.

■

Note that the condition in Theorem 1 can be verified at each node using only information gathered from its own contention neighborhood. In contrast, our earlier definition required a global view of the network. In a nutshell, our approach is to identify impermissible backlogs and use an additive-increase multiplicative-decrease algorithm (similar to the TCP's congestion control algorithm) to adapt the weights to fix it. Initially, we set the local weights of all nodes to 1 and perform the following operations periodically:

1) Service nodes using their current local weights for a time window, W. W is a system-wide constant, which is expressed as a multiple of time-slots. Within each window, each node receives a number of time slots that is proportional to its weight.
2) Estimate the service rate of a unit-weight backlogged flow at each node over the previous window.
3) Compare each backlogged node N's service rate with the service rate of every node it competes with. If any of these nodes has a higher service rate than N, add N to the set I.
4) At the beginning of each time window, add a positive constant α to the local weight of all nodes in the set I (additive increase). Reduce the local weight of all other nodes by multiplying it by a constant β where $0 < \beta < 1$ (multiplicative decrease).

Note that step 3 identifies all nodes with impermissible backlogs and adds them to the set I. In step 4, we increase the weights of all these nodes while decreasing the weight of nodes which are either (a) not backlogged or (b) have only permissible backlogs. This moves to system closer to its ideal state in the next time window.

To implement of step 2, each node has to keep track of the service rates of its queues. Since we use a *virtual time* (VT) based implementation of WFQ for scheduling flows within a node, the required book-keeping is already taken care of. Recall that VT measures the service (*i.e.,* the number of bits) received by a continuously backlogged flow with unit-weight. Let t_W be the start of a window period, and let N.VT(t) denote the virtual time of node N at time t. Then, N.VT($t_W + W$)−N.VT(t_W) represents the service received by a continuously backlogged flow during the window starting at t_W. Note that this service is exactly the value we need at step 2 within a multiplicative constant, *i.e.,* $1/W$. Thus, at step 3 we can directly use the progress of the virtual time in the previous window to compare the service rate of queues at different nodes.

Step 4 requires broadcasting the the service rates of queues in a two-hop neighborhood. For 1-hop neighbors, this is easily accomplished by including the current virtual-time as part of the *EFLoW* header of every packet. By listening in promiscuous mode, nodes can easily determine the service rates of their immediate neighbors. In order to disseminate this information to 2-hop neighbors, every node also includes a list of its 1-hop neighbors and their service rates in the *EFLoW* header.[2] Due to this *lazy* approach, nodes may not always have the most up-to-date information about their two-hop neighbors. But in practice, we found that the AIMD algorithm is robust enough to converge despite this limitation.

Figure 4 shows the pseudocode for updating the local weights. We experimented with values of $W = 20$ and $W = 40$ timeslots and found that either choice works well in practice. The function *queue.ExpWtAvg()* returns the exponentially weighted average queue size which we use to determine if the node is backlogged.

V. Overlay MAC Layer

As mentioned in Section IV, *EFLoW* assumes that the underlying MAC layer supports weighted fair allocation within a contention neighborhood. Though some MAC algorithms proposed in earlier work [6], [7] do support this feature, hardware implementing these algorithms is not commercially available yet. The 802.11 Distributed Co-ordination Function (DCF), the prevalent MAC standard used in almost all wireless hardware sold today does not have any support for prioritizing service to nodes. However, the Overlay MAC Layer (OML)

[2]The *EFLoW* header is variable in length, and was 48 bytes on average in our experiments. Further optimizations (compression, differential coding etc.) are possible, but are outside the scope of this work.

```
function Node::ProcessEndWindow()

  //first check if we are backlogged
  if queue.ExpWtAvg() < congestionThreshold
    //not congested hence multiplicative decrease
    weight ← weight × β
    return

  myServiceRate ← VT(currTime) −
          VT(currTime − W)

  maxServiceRate ← myServiceRate
  for all N ∈ ContentionList
    nServiceRate =
      N.VT(currTime) − N.VT(currTime − W)
    if nServiceRate > maxServiceRate
      maxServiceRate = nServiceRate

  if maxServiceRate >= myServiceRate
    //additive increase
    weight ← weight + α
  else
    //congested, but must go into MD
    weight ← weight × β
```

Fig. 4. Pseudocode for the function executed by every node at the end of W time slots

proposed in [4] provides a workaround for this issue by adding support for weights without any changes to 802.11 hardware. OML implements TDMA scheduling within a k-hop neighborhood (we set $k = 2$ in our experiments) at a layer between the MAC and transport layers.

In order to achieve this, OML limits buffering at the MAC layer and below to exactly one packet (which the hardware is attempting to transmit) through a simple change to the device driver. This allows us to maintain all queues within OML and tightly control the timing of transmission attempts by the hardware. But since we don't have direct control over the hardware, it is very hard to achieve perfect clock synchronization and enforcement at the boundaries of the time slots. To get around this issue, OML uses *coarse-grained* time slots (20 ms).[3]

OML can schedule time slots with very low overhead; almost all control messages are piggybacked on existing data traffic. A full description of the protocol and its implementation is available in [4]. However, one of the drawbacks of using coarse-grained time-slots is that when the queue at a node is full, all packet arrivals will be dropped until the node is allowed to transmit again. This translates to a huge burst of losses which can force TCP flows back into slow-start. To avoid this, we try to prevent queues from ever getting full by implementing a variation of Random Early Dropping (RED) at every node. The key difference between RED as described in [20] and our implementation is that instead of drop-tail, we have per-flow fair-queuing at every node. Hence, when an packet arrival triggers an early drop, we drop a packet from the longest sub-queue (as opposed to dropping the incoming packet). With this simple change, we found OML to be a good vehicle to implement and test *EFLoW*.

[3]With 802.11a radios at 6Mbps, we can transmit about 10 MTU sized packets in each time slot.

VI. Results

We have implemented *EFLoW* on top of OML in both a simulator and a testbed. In this section we will describe both and present our results. More results are available in our technical report [21].

A. Simulation

We have implemented our algorithm in *ns2* [22], an event-driven packet-level network simulator. We use the simulator primarily for two reasons. Firstly, we can easily change the topology or the size of the network in a simulator. Secondly, the simulator allows us to implement an Oracle which uses global knowledge to approximate the ideal allocation. We use this Oracle (see Section VI-A1) for comparison with *EFLoW*.

We simulate 802.11a radios at 6Mbps in each of our experiments. The radio range was around 250m using the two-ray propagation model [23]. Since our goal is to study the end-to-end behavior of flows along a given path, we use static pre-configured routing to avoid additional complications due to the execution of a routing protocol. RTS/CTS was disabled in all the experiments because there is no need for RTS/CTS when using OML. Surprisingly we saw better fairness and throughput with RTS/CTS disabled even when not using OML. As reported in some earlier work [11], [24], our observations confirm that the overhead of RTS/CTS is not justified in relation to the number of collisions it prevents.

To quantify the fairness of the outcome, we use the metric defined by Chiu and Jain in [25]. Consider a system with M flows, with weights $w_1, w_2, \ldots, w_M$, each receiving a throughput $x_1, x_2, \ldots, x_M$. The fairness index F is defined as

$$F = \frac{(\sum_i x_i/w_i)^2}{M \sum_i x_i^2/w_i^2}$$

Note that $F = 1$ when each flow's throughput is exactly in proportion to its weight, and $F = 1/M$ when only one flow receives the entire throughput. Note that Max-Min fairness does not always yield $F = 1$. The throughput allocated to a flow depends on the level of congestion and interference it experiences in addition to its weight (see the example in Figure 1). However in our experience, we found F to a useful metric in comparing the *relative* performance of different schemes, and is easy to compute unlike the true Max-Min fair allocation.

1) Ideal Allocation: As mentioned earlier, computing the ideal allocation is a hard problem even in the case of a simulation. Instead of trying to compute it exactly, we use the following slot allocation algorithm (called the Oracle) to approximate the ideal schedule. The algorithm uses global knowledge to sort all nodes based on how much service their queues are receiving. We can then prioritize assignment of the slot to nodes which have received the least service. The pseudocode for this is shown in Figure 5.

Note that this is only an approximation because (a) we break ties arbitrarily in *GetLeastService* and (b) we still use OML as apposed to a more fine-grained scheduling discipline.

```
function ::AssignSlot(NodeList NL)

  for all N ∈ NL
     N.active = false

  while NL ≠ φ
     N = NL.GetLeastServiced()
     N.active = true
     for all M ∈ NL
        if M.Interferes(N)
           NL.Remove(M)
```

Fig. 5. Pseudocode for the Oracle to approximate the ideal allocation

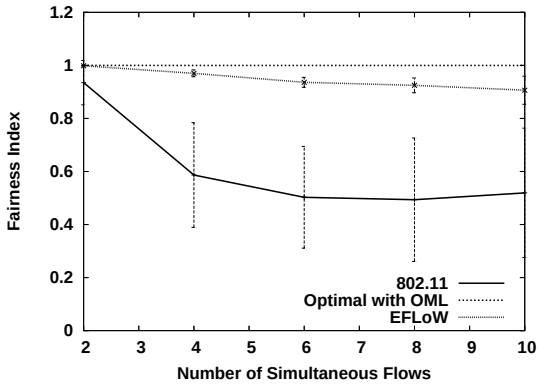

Fig. 6. Effect of the number of simultaneous flows on fairness with and without *EFLoW*

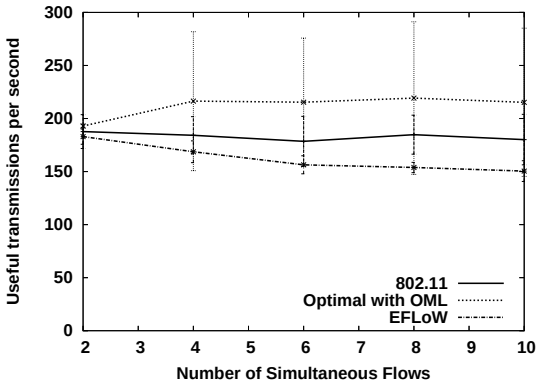

Fig. 7. Effect of the number of simultaneous flows on utilization with and without *EFLoW*

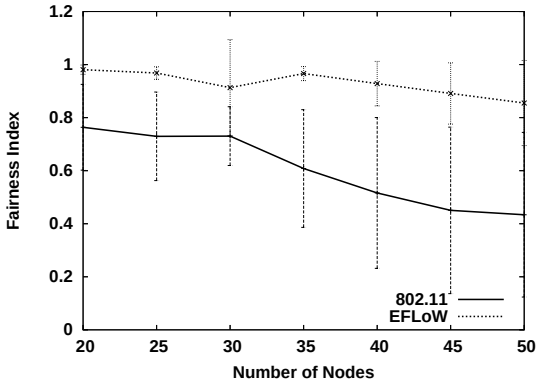

Fig. 8. Effect of the network size on the performance of 802.11 and *EFLoW*

To quantify the benefits of *EFLoW*, our first step is to compare the fairness and efficiency with that of 802.11 for UDP traffic (Section VI-A2). We then study the scaling behavior of OML by increasing the network size in Section VI-A3. Next, we study the performance of TCP flows in Section VI-A4. Finally, we look at the performance of short flows in Section VI-A5.

2) Fairness and Efficiency: In our first experiment, we construct the network by placing 30 nodes at random in a 550mx550m square. We simulate a number of simultaneous UDP flows in this network, all terminating at the same sink-node.[4] Figure 6 shows the fairness index F as we increase the number of flows from 2 to 10, averaged over 10 simulation runs. The 802.11 MAC shows poor fairness (index below 0.6) when there are 4 or more flows. On the other hand, *EFLoW* shows very little degradation in fairness even with a large number of flows; the fairness index is always greater than 0.95. The Oracle yields a fairness index which is almost exactly 1 in all cases.

Next, we compare the utilization of the transmission medium under 802.11 and *EFLoW*. OML can lead to some inefficiency in using the medium due to control overheads. Since *EFLoW* is implemented on top of OML, we expect the utilization be lower when using it. To quantify this overhead, we plot the number of *useful* transmissions per second versus the number of flows in Figure 7. A packet transmission is considered *useful* if the packet is eventually delivered to its destination. The efficiency is within 85% of 802.11 even with 10 flows, which we think is reasonable considering the 90% improvement in fairness.

3) Scaling Network Size: In this experiment, we simulate 5 simultaneous flows while scaling the network size from 20 to 50 nodes. We keep the node density constant by increasing the dimensions of the square accordingly. The average fairness index for each case is shown in Figure 8. We can see that the performance of 802.11 progressively worsens as we increase the network size, whereas *EFLoW* shows only a slight degradation.

4) Long-lived TCP flows: In this Section we investigate the performance of TCP flows with and without *EFLoW*. To get a closer look at the throughput of the different flows, we use a Cumulative Distribution Function (CDF) plot instead of the fairness index. For each type of flow (TCP, UDP with and without *EFLoW*), we repeat experiments with 10 simultaneous flows in a 30 node network 10 times and thus obtain 100 flow throughputs. The resulting CDF plots are shown in Figure 9. The first line shows that the starvation problem under 802.11 is much worse for TCP than UDP; 40 flows have throughput close to zero. In other words, when competing with 9 other simultaneous flows, there is a 40% chance that almost no data will reach the destination! From the second line in Figure 9,

[4]We consider this star traffic pattern to be indicative of the scenario where all nodes are trying to reach the same wired gateway.

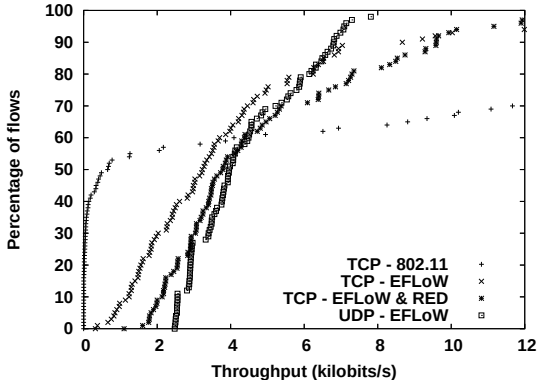

Fig. 9. Comparison of performance of TCP and UDP flows

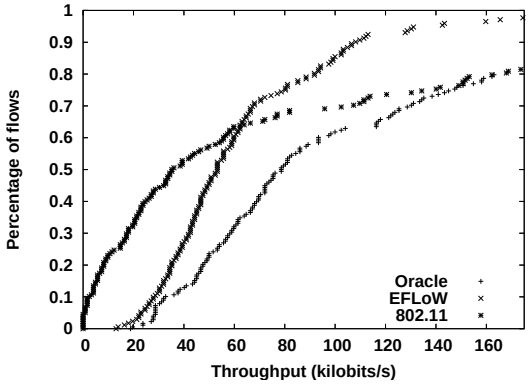

Fig. 10. Performance of short TCP flows

we can see that the performance of *EFLoW* is also not as good as with UDP traffic; 30% of the flows receive less than 2 kbps. However, this problem is significantly reduced when we implement random dropping at every node as described in Section V. With this modification to queue management, 90% of the flows receive at least 2 kbps (see the third line in Figure 9).

5) Short TCP flows: The convergence time of *EFLoW* can have an impact on the performance of short TCP flows. To study this impact, we simulate the following pattern of flows. There are two long flows active during the entire duration of the simulation. In addition we instantiate short flows at exponentially distributed intervals with a 1.25s mean. The duration of each short flow follows a Pareto distribution (the exponent is set to 3) with a 5s mean.[5] Thus, in expectation, 4 short flows and 2 long flows are active at any give time. We repeat this experiment 10 times and plot the CDF of the throughput of the short flows in Figure 10. With 802.11, nearly 30% of the flows experience starvation ($<$ 20 kbps). Despite the overhead of coarse-grained time-slots, *EFLoW* outperforms 802.11 by providing at least 20 kbps to 90% of the flows. Comparing *EFLoW* to the performance of the Oracle shows that there is 30% to 40% reduction in throughput because of the convergence process. However, since the flows are short, the overhead in terms of *completion time* is typically less than 3s.

6) Random flows vs. Star traffic pattern: We now evaluate the performance of *EFLoW* under a random traffic pattern (the source and sink for each flow is chosen at random) and compare it to the performance under the star pattern. Once again, like in Section VI-A4, we look at the CDF plots of throughputs of 100 flows from 10 runs. From Figure 11, we can see that some flows suffer starvation under 802.11 especially for the star traffic pattern; 30% of the flows receive less than 1 kbps. *EFLoW* ensures that 96% of the flows get at least 2.5 kbps.

Note that 802.11 performs better under the random traffic pattern when compared to the star traffic pattern. Since the flows are spread over a larger area, they experience less contention; there is no *hot spot* like the gateway node in the star pattern. This is reflected in the performance of *EFLoW* as shown in the second CDF plot of Figure 11. But even in this case, the fairness is better under *EFLoW*.

[5]The short flows model web browsing and the long flows model file sharing applications/downloads.

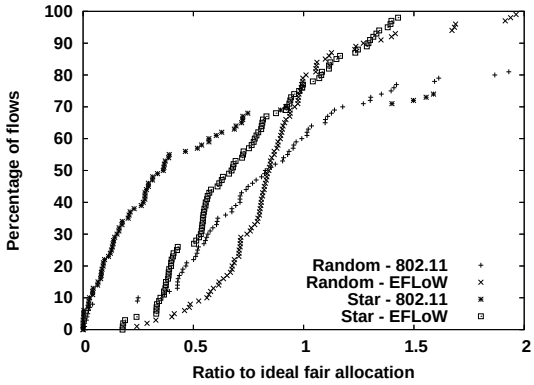

Fig. 11. Performance of 802.11 and *EFLoW* using a star or a random traffic pattern

B. Evaluation in a real system

In this section, we describe the testbed that we used to implement our algorithm. Our testbed consists of 26 wireless nodes based on commodity hardware and software. The hardware consists of small form-factor computers equipped with a 266 MHz Geode processor and 128MB of RAM. Each computer is also equipped with a Netgear [26] tri-mode mini-PCI wireless Ethernet adapter. This card is capable of operating in 802.11a,b and g modes, but we conduct all our experiments in 802.11a mode to avoid interference with another production 2.4GHz wireless network operating in same environment.

The software consists of Linux (kernel 2.4.26) and the the Click software router [27] from MIT, as well as our implementation of OML and the AIMD algorithm in Click. The nodes are placed a single floor of a typical office building.

1) Two simultaneous flows: In this experiment, we start simultaneous flows TCP flows in our test-bed. We pick two random sources and sinks and measure the throughput of both

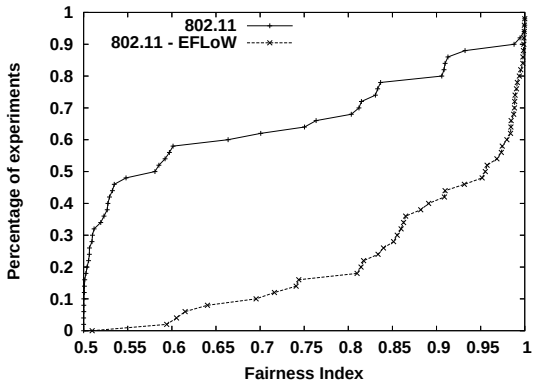

Fig. 12. Fairness Index with two simultaneous flows in our testbed (TCP)

flows over 2 minutes. We use the two-hop neighborhood for determining the interference range of a node, and we only consider cases where at least one node from the first flow interferes with one node from the second flow. Since we set the weight of each flow to be one, we expect both flows to achieve the same throughput in this case.

We repeat the experiment 50 times, and in each case measure F both with default 802.11 as well as with *EFLoW*. We plot the CDF of F over the 50 runs in either case in Figure 12. Recall that since there are two flows, F takes values from 0.5 (least fair) to 1 (most fair). We see that without *EFLoW*, $F \approx 0.5$ in 36% of the cases. With *EFLoW* , such an inequitable distribution happened only once and $F > 0.8$ in 90% of the cases. The average throughput of each flow was 1478 kbps with 802.11 and about 1011 kbps with *EFLoW*. However, when the consider the total number of useful transmissions (throughput of the flow multiplied by the number of hops in the flow), *EFLoW* was only 7% less efficient than 802.11. Thus *EFLoW* can significantly improve the fairness of the system without sacrificing much efficiency.

VII. Conclusions and Future Work

We hope to show through our work that it is indeed possible to achieve good fairness in multi-hop wireless networks using time-slot based MACs even under very dynamic traffic and channel conditions. There are two components that help us achieve this. Firstly, we use an underlying slot allocation algorithm (WSA) that provides support to the higher layers to assign weights to nodes. Secondly, we use an end-to-end feedback based algorithm to converge to the correct set of weights to use in WSA so that we obey the flow constraints at each hop, and also achieve end-to-end Max-Min fairness. We are able to evaluate these algorithms using both simulations and on a real testbed using OML which provides a framework to build time-slot based MACs on top of the underlying MAC layer.

In future work, we plan to address the following two limitations of *EFLoW*. Firstly, the current implementation of *EFLoW* assumes a 2-hop interference region. This assumption may not always hold. While it may be possible to measure the exact interference pattern in advance, a more general solution would be start with an optimistic model and *learn* the interference pattern based on any observed degradation in performance of the model. Secondly, *EFLoW* does not provide any delay guarantees; we only look at the long term performance of flows. We plan to look at ways to augment *EFLoW* to address the requirements of real-time traffic. One possible method is to reserve certain time slots for specific flows in advance. Lastly, we also plan to run more benchmarks on *EFLoW* including more realistic workloads, for example by replaying actual traffic logs.

References

[1] "Tropos networks: Metro-scale mesh networking," http://www.tropos.com/.
[2] "Strix systems," http://www.strixsystems.com/.
[3] P. Bhagwat, B. Raman, and D. Sanghi, "Turning 802.11 inside-out," in *HotNets-II*, 2003.
[4] A. Rao and I. Stoica, "An overlay MAC layer for 802.11 networks," in *MOBISYS*, 2005.
[5] M. Garetto, T. Salonidis, and E. Knightly, "Modeling per-flow throughput and capturing starvation in CSMA multi-hop wireless networks," in *INFOCOM*, 2006.
[6] T. Nandagopal, T.-E. Kim, X. Gao, and V. Bharghavan, "Achieving mac layer fairness in wireless packet networks," in *MOBICOM*, 2000.
[7] I. Chlamtac and S. Kutten, "A spatial-reuse tdma/fdma for mobile multihop radio networks," in *INFOCOM*, 1987.
[8] 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.
[9] V. Gambiroza, B. Sadeghi, and E. Knightly, "End-to-end performance and fairness in multihop wireless backhaul networks," in *MOBICOM*, 2004.
[10] Y. Yi and S. Shakkottai, "Hop-by-hop congestion control over a wireless multi-hop network," in *INFOCOM*, 2004.
[11] "MIT roofnet," http://www.pdos.lcs.mit.edu/roofnet/.
[12] J. Silvester, "Perfect scheduling in multi-hop broadcast networks," in *ICCC*, 1982.
[13] L. Bao and J. Garcia-Luna-Aceves, "Hybrid channel access scheduling in ad hoc networks." in *ICNP*, 2002.
[14] P. Dugar, N. Vaidya, and P. Bahl, "Priority and fair-scheduling over a wireless LAN," in *MILCOM*, 2001.
[15] H. Luo, S. Lu, and V. Bharghavan, "A new model for packet scheduling in multihop wireless networks," in *MOBICOM*, 2000.
[16] J. Padhye, S. Agarwal, V. N. Padmanabhan, L. Qiu, A. Rao, and B. Zill, "Estimation of link interference in static multi-hop wireless networks," in *IMC*, 2005.
[17] C. T. Ee and R. Bajcsy, "Congestion control and fairness for many-to-one routing in sensor networks," in *SenSys '04*. New York, NY, USA: ACM Press, 2004, pp. 148–161.
[18] D. Bersekas and R. Gallaher, *Data Networks*. Pretice Hall, 1987.
[19] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in *SIGCOMM*, 1989.
[20] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, 1993.
[21] A. Rao and I. Stoica, "EFLoW: End-to-end Fairness using Local Weights in Wireless Networks," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2007-88, Jul 2007.
[22] "The Network Simulator - ns2," http://www.isi.edu/nsnam/ns/.
[23] W. Y. Lee, *Mobile communications engineering*. McGraw Hill, 1982.
[24] R. Draves, J. Padhye, and B. Zill, "Comparison of routing metrics for multi-hop wireless networks," in *SIGCOMM*, 2004.
[25] D.-M. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Comput. Netw. ISDN Syst.*, vol. 17, no. 1, pp. 1–14, 1989.
[26] "Netgear," http://www.netgear.com/.
[27] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263–297, 2000.