IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS) | 978-1-6654-0443-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/INFOCOMWKSHPS51825.2021.9484546

Support for Differentiated Airtime in Wireless Networks

Daniel J. Kulenkamp
CIDSE, Arizona State University
Tempe, AZ, USA
dkulenka@asu.edu

Violet R. Syrotiuk
CIDSE, Arizona State University
Tempe, AZ, USA
syrotiuk@asu.edu

***Abstract*—Future wireless networks must be able to support Quality of Service (QoS) requirements of emerging 5G and other next-generation applications. REACT is a distributed resource allocation protocol can be used to negotiate airtime among nodes in a wireless network. In this paper, REACT is extended to support QoS airtime. Experimentation on the w-iLab.t wireless testbed in an ad hoc setting shows that these extensions allow REACT to converge on allocations where any node requesting the higher class of airtime receives its allocation, while nodes requesting the lower class are allocated remaining airtime fairly.**

***Index Terms*—IEEE 802.11 wireless networks, Quality of Service, QoS, airtime, distributed protocols**

I. Introduction

In the modern world, wireless networks are everywhere and are used by billions of people. According to the Cisco Annual Internet Report for 2018 to 2023, the 5.1 billion mobile subscribers in 2018 are predicted to grow to approximately 5.7 billion by 2023 [1]. With the emergence of 5G, new applications that require Quality of Service (QoS) support are possible including ultra reliable, low latency applications, such as autonomous vehicles, or high capacity applications such as virtual reality. Wireless local area network (WLANs) are envisioned as an offloading solution for 5G networks, where the 5G infrastructure is unavailable or overloaded [2]. However, this means that WLANs also need to support QoS at a level comparable to 5G.

One mechanism for achieving QoS support in WLANs is airtime allocation. *Airtime* is the amount of time the channel is sensed busy due to frame transmissions [3]. In order to provide QoS support in a wireless setting, we need to ensure that devices that want a higher airtime allocation than others get it, if possible. To provide guarantees on metrics such as delay, airtime allocations need to be consistent and unaffected by other transmitting devices.

The most common channel access method for WLANs today is the Wi-Fi standard, or IEEE 802.11 [4]. Its main mechanism is the distributed coordination function (DCF), a carrier sense multiple access protocol with collision avoidance (CSMA/CA). The IEEE 802.11e amendment introduced traffic prioritization in the enhanced distributed channel access (EDCA) protocol using *access categories* (ACs) for different traffic classes [5]. Each class has its own contention window values, allowing for higher priority traffic to be transmitted in the network before lower priority traffic. The QoS support provided by IEEE 801.11e is still limited, as there are no guarantees on delay, jitter, throughput, or other metrics.

Neither IEEE 802.11 nor IEEE 802.11e allocates airtime to individual nodes. A distributed resource allocation protocol, REACT, can be used for this purpose [6]. REACT negotiates an airtime allocation among neighboring nodes, where nodes concurrently *demand* and *offer* airtime. If nodes demand more airtime than is available, the remaining available airtime is divided equally among the nodes for fairness. Once REACT has converged on an allocation for a node, the allocation must be realized. The original work on REACT proposed a scheduled MAC protocol where the allocation corresponded to slot assignments, implemented in simulation [6]. Later work used a contention-based method, tuning the contention window using an algorithm based on renewal theory (RENEW [3]), and with a control theoretic approach (SALT [7]). Through experimentation on a testbed with these two tuning approaches, REACT has lower delay and jitter statistics than IEEE 802.11, with only a relatively small reduction in throughput [7].

We propose an extension to REACT, $\text{REACT}_{\text{QoS}}$, which extends the original algorithm to support two different classes of airtime, QoS airtime and best effort (BE) airtime. Using the same concept of demanding and offering airtime, $\text{REACT}_{\text{QoS}}$ allows nodes to request and receive QoS airtime, which is prioritized by the algorithm. The remaining airtime is distributed among the remaining BE nodes as in the original algorithm. Since REACT was originally designed, implemented, and evaluated in an ad hoc wireless network setting, we study $\text{REACT}_{\text{QoS}}$ in an ad hoc setting as well.

The contribution of this work is threefold. First, the REACT algorithm is extended to support differentiated traffic. Secondly, $\text{REACT}_{\text{QoS}}$ is implemented and also updated to allow for dynamically changing demands, where prior work focused on static demands. Finally, to improve the performance of the tuning, a traffic shaping mechanism using Linux traffic control (`tc`) is introduced to prevent nodes from obtaining higher airtime than they are allocated. We demonstrate that the new algorithm can successfully allocate differentiated airtime for nodes requesting a higher priority of service, while equally sharing remaining airtime among BE nodes.

The rest of this paper is organized as follows. We first present the QoS extensions to REACT in §II. In §III we describe the experimental setup used to evaluate REACT$_{\text{QoS}}$ using the w-iLab.t testbed, as well as the experiments that were run, presenting the results in §IV. Our plans for next steps for REACT, including application to infrastructure networks, are found in §V, and finally in §VI we present our conclusions.

II. Supporting Airtime Differentiation in REACT

We first present the REACT$_{\text{QoS}}$ algorithm, and then describe its implementation architecture.

A. *REACT$_{QoS}$ Algorithm*

As mentioned in §I, REACT negotiates the airtime allocations for nodes in a wireless network. The idea behind QoS support in REACT$_{\text{QoS}}$ is to provide two classes of airtime which nodes can request. This allows some nodes to receive a higher airtime allocation for their applications. In an ad hoc setting, each node runs both an *auctioneer* and a *bidder* algorithm, which maintains the list of *offers* and *claims*, respectively, for the node and its adjacent auctions (neighbors).

We define two traffic classes: QoS and BE. In the QoS class, a node is guaranteed its allocation if available, while the BE class offers no guarantees on how much airtime a node receives. In this work, all traffic from a node is treated as a single flow. Therefore, each node is considered either a QoS node or a BE node, and must select whether it is requesting QoS airtime or BE airtime. We now present the bidder and auctioneer algorithms that form REACT$_{\text{QoS}}$.

Algorithm 1 presents the REACT$_{\text{QoS}}$ bidder. Each bidder i maintains three sets: B_i^O is the set of BE offers, Q_i^O is the set of QoS offers, and α_i is the set of auctions bidder i has joined. Additionally, the bidder keeps track of two variables, q_i and b_i which are its QoS and BE *demands* or *claims*, respectively. Initially, all sets are empty and demands are 0.

Though the bidder keeps track of a QoS and a BE demand, only one is allowed to be positive at a time. When the bidder receives a new offer from auctioneer j, it updates its sets accordingly. The main function for the bidder is UpdateClaim (lines 16-26). If q_i is positive, this means the demand is a QoS demand. The node must check whether it has received QoS offers from each of its neighboring nodes, and that the offer is at least as large as the demand. If any of the offers from neighboring nodes is not as large as the demand, it sets b_i to the QoS demand q_i, and sets q_i to 0. After it checks all neighboring offers, it sets *QoS claim* to q_i and *BE claim* to b_i. If q_i is not positive, then it sets *QoS claim* to 0 and sets *BE claim* to be the minimum of all offers in B_i^O and b_i. Finally, it sends the tuple $(\textit{QoS claim}, \textit{BE claim})$ to all auctions in α_i.

Algorithm 2 presents the REACT$_{\text{QoS}}$ auctioneer. Similar to the bidder, the auctioneer maintains three sets: B_j^C is the set of BE claims, Q_j^C is the set of QoS claims, and β_j is the set of bidders at auction j. It maintains one variable c_j, the capacity of its resource, which is the airtime to auction at that node.

The main part of this algorithm is the function UpdateOffer (lines 18-47). In this function, two sets are maintained: R is the set of all *satisfied* QoS bidders, and C is the set of all *satisfied* BE bidders. As the algorithm works through claims, it adds bidders to these sets accordingly. The variable A_j is the remaining airtime that has not been allocated and is initially set to the capacity of auction j, c_j. The boolean flag *done* keeps track of when the algorithm has terminated.

Algorithm 1 REACT Bidder for node i.

```
1:  upon initialization do
2:      $\alpha_i \leftarrow \emptyset$        ▷ set of neighboring auctions of bidder $i$
3:      $B_i^O \leftarrow \emptyset$, $Q_i^O \leftarrow \emptyset$        ▷ set of BE, QoS offers
4:      $q_i \leftarrow 0$, $b_i \leftarrow 0$        ▷ QoS, BE demand for bidder $i$
5:  upon receiving a new demand magnitude $(q_i, b_i)$ do
6:      UpdateClaim()
7:  upon receiving offer $(x_j, y_j)$ from auctioneer $j$ do
8:      $Q_i^O[j] \leftarrow x_j$
9:      $B_i^O[j] \leftarrow y_j$
10:     UpdateClaim()
11: upon bidder $i$ joining auction $j$ do
12:     $\alpha_i \leftarrow \alpha_i \cup j$
13:     UpdateClaim()
14: upon bidder $i$ leaving auction $j$ do
15:     $\alpha_i \leftarrow \alpha_i \setminus j$
16: procedure UpdateClaim()
17:     if $q_i > 0$ then        ▷ either $q_i$ or $b_i$ is positive
18:         for offer $\in Q_i^O$ do
19:             if offer $< q_i$ then
20:                 $b_i \leftarrow q_i$
21:                 $q_i \leftarrow 0$
22:         $\textit{QoS claim} \leftarrow q_i$
23:         $\textit{BE claim} \leftarrow b_i$
24:     else
25:         $\textit{QoS claim} \leftarrow 0$
26:         $\textit{BE claim} \leftarrow \min(\{\textit{offers}[j] : j \in B_i^O\}, b_i)$
27:     send $(\textit{QoS claim}, \textit{BE claim})$ to all auctions in $\alpha_i$
```

Lines 23-25 check if all bidders have been satisfied or constrained by this auction; a bidder is *constrained* by an auction i if it cannot increase its claim based on a higher offer from a different auction j. Then, *BE offer* is set to the remaining airtime plus the maximum claim in B_j^C, to ensure that a bidder can increase its claim if it is no longer constrained by an adjacent auction. Lines 26-46 are where QoS claims are satisfied and whether BE claims need to be constrained.

REACT$_{\text{QoS}}$ first attempts to satisfy each of the QoS bidders, subtracting out their claims from the pool of available airtime (lines 28-37). If the claim is less than the remaining airtime, we can satisfy this request and move the bidder to the set R (lines 30-32). We set `done` to `False`, because we may need to update the BE offer with the new available airtime. If we cannot satisfy the claim (line 34), we instead move the bidder to the BE auction, and remove it from the QoS auction (lines 35-37), again setting *done* to `False`. This allows a QoS node to still receive some airtime, rather than having wait until the next round of REACT messages are exchanged.

Algorithm 2 REACT Auction for node j.

```
1:  upon initialization do
2:      $\beta_j \leftarrow \emptyset$                          ▷ set of bidders at auction $j$
3:      $B_j^C \leftarrow \emptyset$, $Q_j^C \leftarrow \emptyset$       ▷ set of BE, QoS claims at $j$
4:      $c_j \leftarrow 0$                           ▷ capacity of resource $j$
5:  upon receiving a new capacity of $c_j$ do
6:      UpdateOffer()
7:  upon receiving claim from bidder $i$ do
8:      if claim is QoS claim then
9:          $Q_j^C \leftarrow Q_j^C \cup claim$
10:     else
11:         $B_j^C \leftarrow B_j^C \cup claim$
12:     UpdateOffer()
13: upon bidder $i$ joining auction $j$ do
14:     $\beta_j \leftarrow \beta_j \cup i$
15: upon bidder $i$ leaving auction $j$ do
16:     $\beta_j \leftarrow \beta_j \setminus i$
17:     UpdateOffer()
18: procedure UPDATEOFFER()
19:     $C \leftarrow \emptyset$, $R \leftarrow \emptyset$        ▷ set of satisfied BE, QoS bidders
20:     $A_j \leftarrow c_j$
21:     done ← False
22:     while (done = False) do  ▷ all bidders are satisfied
23:         if $(R \cup C = \beta_j)$ then
24:             done ← True
25:             BE offer ← $A_j + \max(\{claims[i] : i \in B_j^C\})$
26:         else
27:             done ← True
28:             for $q \in \{Q_j^C \setminus R\}$ do
29:                 QoS offers[$q$] ← $A_j$
30:                 if claims[$q$] <= QoS offer then
31:                     $R \leftarrow R \cup q$
32:                     $A_j \leftarrow A_j - claims[q]$
33:                     done ← False
34:                 else                  ▷ move bidder to BE auction
35:                     $B_j^C \leftarrow B_j^C \cup q$
36:                     $Q_j^C \leftarrow Q_j^C \setminus q$
37:                     done ← False
38:             if $|B_j^C \setminus C| > 0$ then
39:                 BE offer = $A_j / |B_j^C \setminus C|$
40:             else
41:                 BE offer = $A_j$
42:             for $b \in \{B_j^C \setminus C\}$ do
43:                 if (claims[$b$] < BE offer then
44:                     $C \leftarrow C \cup b$
45:                     $A_j \leftarrow A_j - claims[b]$
46:                     done ← False
47:     send (QoS offers, BE offer) to all bidders in $Q_j^C \cup B_j^C$
```

In line 38, we check if there are still unsatisfied bidders, and if so, we divide the remaining airtime up among them; else, we set it to the available airtime as all claims are satisfied. Finally, for each remaining unsatisfied bidder, if its claim is less than the BE offer, we can satisfy that bidder, subtract its claim from the remaining available airtime, and set `done` to `False` to ensure we iterate again to update the offers. Once all bidders are satisfied or constrained, we send the set of QoS offers and the BE offer to all bidders in $Q_j^C \cup B_j^C$.

Figure 1 gives an example of the operation of REACT$_{\text{QoS}}$. In this case, node 4 is a QoS node and is demanding 50% QoS airtime. The other nodes are demanding 100% BE airtime. We see that node 4 receives its QoS request, but the rest end up claiming less than their request. Nodes 2 and 3 are constrained due to node 4's request, because the auction at node 3 only has 50% of its airtime left over to offer as BE airtime. It splits it evenly between the remaining nodes at its auction (nodes 2 and 3), leaving each to claim half. Node 1 claims more airtime because node 2 offers 50%. Node 2 can offer 50% because after nodes 2 and 3 claim 25% each, it has 50% left over. No auction here is aware of all four nodes due to the line topology; the algorithm relies on indirect information from neighboring auctions to determine allocations.

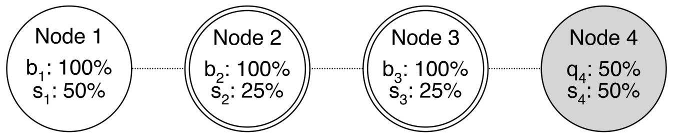


Figure 1: Example of REACT$_{\text{QoS}}$ on a line topology: b_t gives node t's initial BE demand; q_t is node t's initial QoS demand; s_t gives node t's ultimate airtime allocation. White backgrounds represent BE nodes, gray represent QoS nodes. Double edged circles represent nodes whose airtime is constrained. The dotted lines represent bidirectional links.

B. REACT$_{QoS}$ Architecture

Figure 2 shows the architecture of the REACT$_{\text{QoS}}$ implementation. REACT$_{\text{QoS}}$ communicates directly with the driver functions, the SALT contention window tuner, and the traffic shaper (here, a token bucket filter or TBF). The TBF operates on data traffic before it enters the queue (TX-Q-DATA), so REACT$_{\text{QoS}}$ can use the shaper to control the rate at which the node sends traffic. The traffic shaper uses a node's allocation, along with the channel rate (which is set equally on all nodes for the experiment) to estimate the rate of traffic. REACT$_{\text{QoS}}$ control messages bypass the shaper, using a separate TX-Q-CTL queue that goes directly to the driver. Additionally, REACT$_{\text{QoS}}$ interacts directly with the driver to receive control messages and packet statistics, which it uses to conduct the auction as well as to tune the contention window. It passes the claim for its node to the CW estimation module, which sets the CW parameters in the driver.

We use Linux `tc` as our TBF, to restrict the load of a node to be no greater than that of its allocated airtime as a percentage of the total channel capacity. For example, if a node is allocated an airtime of 25%, and the channel rate is $6\,\text{Mbps}$, then its traffic shaper is set to $1.5\,\text{Mbps}$.

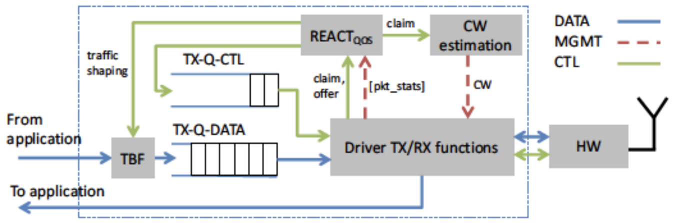


Figure 2: Architecture of REACT$_{\text{QoS}}$ implementation.

III. Experiments and w-iLab.t Environment

Experiments are conducted on the w.iLab-t testbed in Zwijnaarde, Belgium [8]. ZOTAC type nodes (equipped with Atheros cards using the *ath9k* driver) are used for the experiments. The SALT tuner requires two kernel extensions: one to tune the CW and another to enable setting the CW to a value other than a power of two. REACT$_{\text{QoS}}$ is implemented in Python 3 on an Ubuntu 16.04 OS.

As in past work, in REACT$_{\text{QoS}}$ we only allow each node to offer 80% of the airtime. This is done to ensure that its control messages (which are enqueued in a separate priority queue; TX-Q-CTL in Figure 2) are received without being impacted by the data load on the nodes.

An importable module allows REACT to run from a script. This allows new demands to be enqueued at any time to trigger the REACT$_{\text{QoS}}$ auction. The auction adjusts its demand for the node and sends out new messages to adjacent auctions, accordingly.

Two sets of experiments are conducted, each on a different ad hoc topology: a line consisting of four nodes, and a complete topology of four nodes as Figures 3 and 4 show, respectively. We select the complete topology to demonstrate REACT$_{\text{QoS}}$ behavior when all nodes are aware of each other, and the line topology to demonstrate behavior with the presence of the exposed node problem [9]. In the figures, black lines represent bidirectional links and red arrows represent flow paths. In the complete topology flows run in a circular fashion, while flows are run between the outer nodes and their adjacent neighbor in the line topology. Flows were generated between nodes using MGEN[1]. Packets were generated using a Poisson distribution, had a fixed size of 1024 bytes, and were sent using UDP. The flow rates were set to 6 Mbps.

We run a set of four experiments to demonstrate the performance of REACT$_{\text{QoS}}$, as well as to demonstrate the behavior under a dynamic load scenario. Each experiment is run twice, once under each topology. Experiment 1 is conducted to show performance of IEEE 802.11, with REACT disabled. Experiment 2 shows the implementation of REACT without QoS extensions enabled. Experiment 3 shows REACT$_{\text{QoS}}$ running, with one of the four nodes requesting airtime from the QoS class. Finally experiment 4 combines both REACT$_{\text{QoS}}$ and dynamic demands, with events and their times listed in Table I.

[1]https://www.nrl.navy.mil/itd/ncs/products/mgen

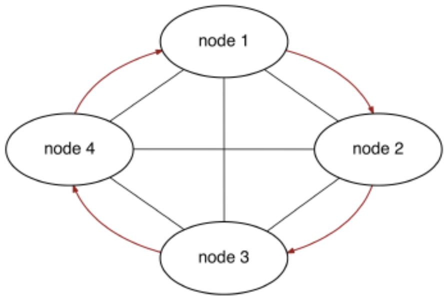


Figure 3: Complete topology of four nodes with flow paths.

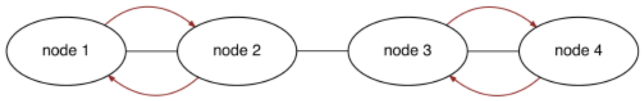
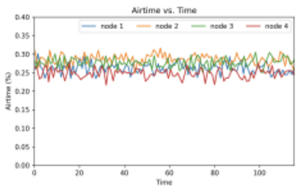


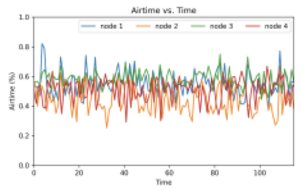
Figure 4: Line topology of four nodes with flow paths.

IV. Results and Discussion

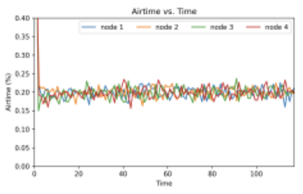
Beginning with experiment 1, we see that in the complete topology IEEE 802.11 performs quite well (Figure 5a). Each node is allocated about 25% of the airtime for the entire experiment run. For the line topology in Figure 5b, however, IEEE 802.11 has much more variable performance.



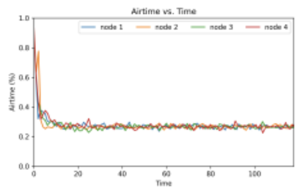
(a) 802.11 complete topology



(b) 802.11 line topology



(c) REACT complete topology



(d) REACT line topology

Figure 5: Experiment 1 and 2: 802.11 v. original REACT in complete and line topology. For (c) and (d), each node is requesting 100% of the offered airtime (because 80% is offered, this results in about a 20% allocation per node).

Experiment 2 provides an insight into how REACT performs. From Figures 5c and 5d we see that REACT converges on a much tighter airtime allocation, i.e., the means of the airtime of the four nodes are much closer. The airtime each node receives in REACT is less than under IEEE 802.11 (partially due to 80% of the airtime being allocated), but this much more consistent allocation is an improvement.

To compare between the two approaches, we analyze the variance; however, to account for the period of time that REACT takes to converge, we only compute the variance

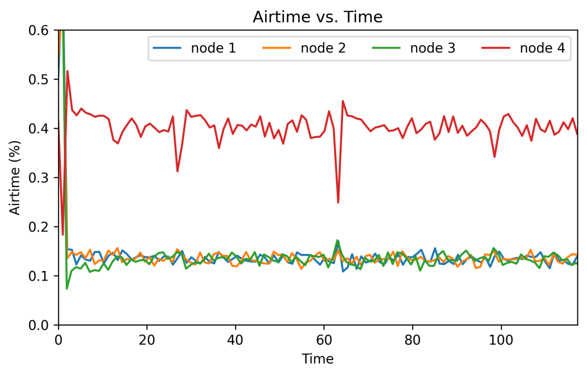


Figure 6: Experiment 3: REACT$_{\text{QoS}}$ in complete topology. Node 4 is requesting 50% QoS airtime, and the rest 100% (again, 80% is offered, so node 4 receives 40%).

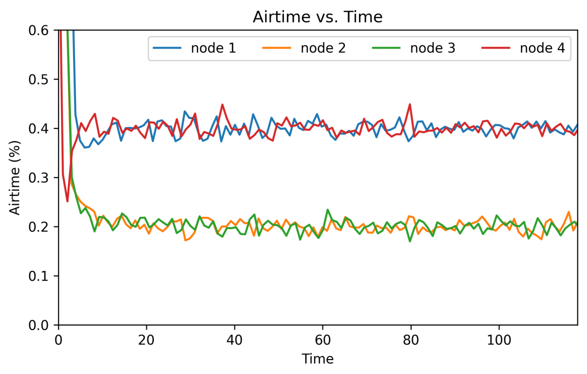


Figure 7: Experiment 3: REACT$_{\text{QoS}}$ in line topology. As in Figure 6, node 4 is requesting 50% QoS airtime, with the remaining nodes requesting 100% BE (with 80% offered).

for the time after REACT has converged (at second 3 for the complete topology; at second 10 for the line). For the complete topology, the variances for the nodes range from 0.017% to 0.471% for 802.11, while for REACT they range from 0.016% to 0.024%. For the line topology, variances for 802.11 range from 0.3% to 0.877%, while for REACT they range from 0.012% to 0.02%. There is some improvement in the complete topology, but with the line topology the improvement is much more apparent, as can be seen when comparing between Figure 5b and Figure 5d.

In Experiment 3, node 4 requests 50% of the offered airtime, which translates to 40% of the actual airtime. With the complete topology (Figure 6), node 4 receives about 40% airtime, while the remaining three nodes receive about 13% each. The airtimes for the line topology are more complicated to understand (Figure 7), however, this is the same scenario presented in Figure 1. As before, node 4 requests half of the offered airtime, which it correctly receives. Node 1 also received half of the offered airtime, since it is only constrained by the auction at node 2. Nodes 2 and 3 are constrained by multiple auctions, which results in them receiving less airtime than either node 1 or node 4. This can be seen in Figure 7, where nodes 1 and 4 receive half the offered airtime (40%) and nodes 2 and 3 receive a quarter (20%). Additionally, REACT$_{\text{QoS}}$ takes four seconds to converge in the complete topology, and 8 seconds for the line topology. The line topology takes longer to converge because information has to travel two hops in the worst case, where a complete topology is by definition fully connected and information has to travel at most one hop.

Finally, for Experiment 4, Figures 8 and 9 show node 1 requesting 50% QoS airtime, with remaining nodes adjusting their demands dynamically according to the events in Table I. Note that node 1 has a consistent allocation throughout the experiment and is not affected by the demand changes at other nodes. In the complete topology, even when nodes 2 and 3 adjust their demands at seconds 120 and 180, the sum of the three BE demands is higher than 50% of the airtime split three ways. Therefore the airtime remains equally split between the three nodes, so no change is reflected in the figure. In contrast, in the line topology, we see that when node 2 increases its demand to 50% of the airtime at second 120, it is allocated more of the airtime, since there are fewer nodes competing for the airtime at node 2. Later at second 180, nodes 2 and 3 increase their demands further — they are both constrained the neighboring auctions and have to split the airtime. Though REACT$_{\text{QoS}}$ took twice as long to converge initially in the line topology (20 s vs. 9 s in the complete topology), re-convergence times for both topologies averaged only four seconds.

Table I: Events for Experiments 4

Time (s)	Node	Demand	QoS
0	node 1	50%	Yes
0	node 2	10%	No
0	node 3	10%	No
0	node 4	100%	No
60	node 2	20%	No
60	node 3	20%	No
120	node 2	50%	No
180	node 2	80%	No
180	node 3	50%	No

V. Next Steps for REACT

REACT$_{\text{QoS}}$ is a first step to a more complete solution to realize airtime allocations and provide QoS support in a wireless network. However, there is much that could be done to further the work presented here.

First, this implementation of REACT$_{\text{QoS}}$ impacts the delay and jitter performance. Prior work shows that REACT sacrifices a small amount of throughput in order to achieve superior delay and jitter [7]. Due to the traffic shaper used (the `tbf` module from Linux `tc`[2]), this implementation has worse delay and jitter. Our focus in this paper is to achieve varied airtime allocations; we leave it to future work to improve the tuning algorithm to not sacrifice delay or jitter.

[2]https://www.man7.org/linux/man-pages/man8/tc-tbf.8.html

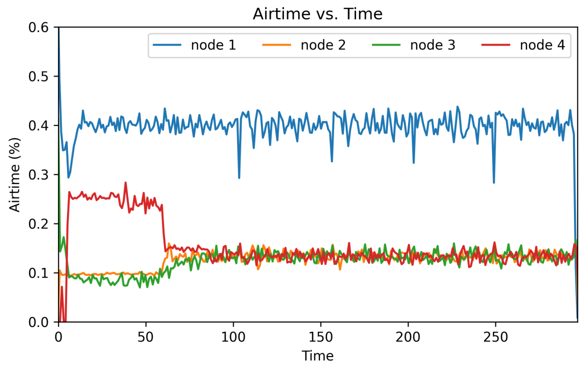


Figure 8: Experiment 4: REACT$_{\text{QoS}}$ in complete topology, dynamic and varied demands according to Table I.

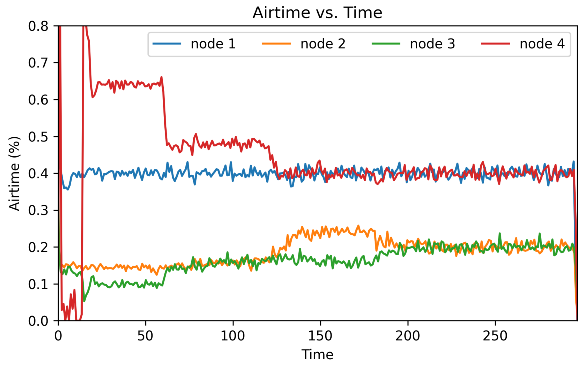


Figure 9: Experiment 4: REACT$_{\text{QoS}}$ in line topology, dynamic and varied demands according to Table I.

Another improvement that could be made is to implement an admission control scheme for the QoS requests. Currently, the algorithm "downgrades" nodes from a QoS request to a BE request if the request cannot be satisfied. An explicit mechanism for denying a request to ensure QoS nodes do not overwhelm the network may be preferable.

REACT$_{\text{QoS}}$ was implemented and evaluated in an ad hoc wireless network scenario. We believe that the algorithm could help improve the performance of WLANs in a managed access point (AP) scenario. In this case, one REACT auction could be run per-AP, with clients only running the bidder portion of REACT. The AP would then have full knowledge of its network and could make decisions in a centralized manner. Furthermore, if APs were in the same collision domain, a REACT auction could be run between entire AP-subnets, over a wired connection, to negotiate airtime between APs.

The use of REACT$_{\text{QoS}}$ in an AP scenario suggests the use of Software-Defined Networking (SDN) in a wireless network. There has been much previous work on SDN and network slicing in the wireless domain, such as the EmPOWER system [10], [11]. But much of this work (including EmPOWER) operates higher in the network stack and still depends on IEEE 802.11 for the lower level MAC protocol. It may be possible to use the more precise airtime realization mechanism of REACT$_{\text{QoS}}$ to give nodes a precise allocation determined by a centralized controller. Furthermore, conducting slicing in the uplink direction is not a solved problem, which REACT$_{\text{QoS}}$ has the potential to improve.

VI. Conclusions

In this work we proposed REACT$_{\text{QoS}}$, a distributed protocol to allocate airtime with QoS support. This allows nodes to request a higher class of airtime which, if available at adjacent auctions, guarantees the node a higher airtime allocation. Through experimentation on the w-iLab.t testbed we have shown that this mechanism is successful at achieving varied allocations. This extension, combined with traffic shaping and dynamic demands, allow us to tune the airtime nodes receive and opens the door to improved QoS support in wireless networks.

Acknowledgements

This research was supported in part by the U.S. National Science Foundation (NSF) NeTS Award 1813451.

References

[1] Cisco, "Cisco annual internet report (2018-2023)," March 2020. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[2] I. Elgendi, K. S. Munasinghe, and A. Jamalipour, "Traffic offloading for 5g: L-lte or wi-fi," in *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2017, pp. 748–753.

[3] D. Garlisi, F. Giuliano, A. L. Valvo, J. Lutz, V. R. Syrotiuk, and I. Tinnirello, "Making WiFi Work in Multi-hop Topologies: Automatic Negotiation and Allocation of Airtime," in *2015 IEEE 35th International Conference on Distributed Computing Systems Workshops*. Columbus, OH, USA: IEEE, Jun. 2015, pp. 48–55.

[4] "IEEE Std 802.11™-2016, IEEE Standard for Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 11: Wireless LAN Medium Access Control," p. 3534, 2016.

[5] Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification. Amendment 7: Medium Access Control (MAC) Quality of Service (QoS), ANSI/IEEE Std 802.11e, LAN/MAN Standards Commit- tee of the IEEE Computer Society Std., 2005.

[6] J. Lutz, C. J. Colbourn, and V. R. Syrotiuk, "ATLAS: Adaptive Topology- and Load-Aware Scheduling," *IEEE Transactions on Mobile Computing*, vol. 13, no. 10, pp. 2255–2268, Oct. 2014.

[7] M. J. Mellott, D. Garlisi, C. J. Colbourn, V. R. Syrotiuk, and I. Tinnirello, "Realizing airtime allocations in multi-hop wi-fi networks: A stability and convergence study with testbed evaluation," *Computer communications*, vol. 145, pp. 273–283, 2019.

[8] S. Bouckaert, P. Van Wesemael, J. Vanhie-Van Gerwen, B. Jooris, L. Hollevoet, S. Pollin, I. Moerman, and P. Demeester, "Distributed Spectrum Sensing in a Cognitive Networking Testbed," in *Towards a Service-Based Internet*, W. Abramowicz, I. M. Llorente, M. Surridge, A. Zisman, and J. Vayssière, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, vol. 6994, pp. 325–326, series Title: Lecture Notes in Computer Science.

[9] J. Mvulla, Y. Kim, and E.-C. Park, "Probe/PreAck: A Joint Solution for Mitigating Hidden and Exposed Node Problems and Enhancing Spatial Reuse in Dense WLANs," *IEEE Access*, vol. 6, pp. 55 171–55 185, 2018.

[10] E. Coronado, R. Riggio, J. Villalon, and A. Garrido, "Lasagna: Programming Abstractions for End-to-End Slicing in Software-Defined WLANs," in *2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*. Chania, Greece: IEEE, Jun. 2018, pp. 14–15.

[11] E. Coronado, S. N. Khan, and R. Riggio, "*5G-EmPOWER* : A Software-Defined Networking Platform for 5G Radio Access Networks," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 715–728, Jun. 2019.