Dyson: An Architecture for Extensible Wireless LANs

Rohan Murty, Jitendra Padhye, Alec Wolman, Matt Welsh

TR-02-09

Computer Science Group
Harvard University
Cambridge, Massachusetts

Dyson: An Architecture for Extensible Wireless LANs

Rohan Murty[†], Jitendra Padhye[‡], Alec Wolman[‡], Matt Welsh[†]
[†]Harvard University,[‡]Microsoft Research

ABSTRACT

As wireless local area networks (WLANs) continue to evolve. the fundamental division of responsibility between the access point (AP) and the client has remained unchanged. In most cases, clients make independent decisions about associations and packet transmissions, using only locally available information. Furthermore, the IEEE 802.11 standard defines a very limited interface for transferring information between the APs and the clients. These factors impede customization of WLANs to meet site-specific challenges, and in a more general sense, impede rapid innovation to face challenges posed by new applications such as VoIP.

This paper describes Dyson, an extensible architecture for WLANs, targeted primarily at enterprise scenarios. Our architecture is based on centralized, global management of channel resources. To provide extensibility, the interface between the infrastructure and clients is simple and relatively low-level, and can be controlled through a programmatic interface. Clients provide primitives that allow the central controller to control many aspects of client behavior. The controller can also instruct clients to gather and report information about channel conditions. We show that using these simple primitives, and by leveraging historical information, the network designer can easily customize many aspects of the WLAN behavior.

We have built a prototype implementation of Dyson, which currently runs on a 23-node testbed distributed across one floor of a typical academic building. Using this testbed, we examine various aspects of the architecture in detail, including a range of policies for improving client-AP associations, providing user-specific airtime reservations, mitigating the effects of interference, and improving mobile handoffs. We show that Dyson is effective at providing greater efficiency while opening up the network to site-specific customizations.

1. INTRODUCTION

Wireless networks are struggling to innovate in the face of new application demands, such as media streaming, voice over IP, and increasing use of mobile devices, such as WiFi enabled phones. At the same time, achieving efficient use of the wireless spectrum is becoming more challenging, given that most wireless LANs perform network management in an entirely decentralized fashion. Enterprises wishing to roll out new applications, services, or policies in a wireless LAN are faced with the ossification of standards and the high variance across different vendors' implementations. In this paper, we argue that it is time to rethink the architecture of wireless networks from the ground up, to enable greater observability, control, and extensibility to meet future needs.

Although wireless LANs have evolved over time, including significant improvements to the PHY and MAC layers, one critical aspect of the WLAN architecture that has remained unchanged is the interface between clients and access points. Each node makes independent decisions about AP associations, PHY data rates, transmission power level, and other parameters that have a substantial impact on overall network performance. These decisions are typically based on *local* observations of the network state, with no explicit coordination between nodes. Further complicating the matter, different vendors introduce very different policies in their WLAN firmware and software, as this is viewed as an opportunity for innovation and competition. However, wresting control over the network is difficult or impossible given this "every station for itself" mentality.

The current trend in commercial WLANs is moving toward the use of a central controller that manages access points across an enterprise [1, 2]. However, this approach does not incorporate the perspective of the *clients* in the network. We argue that effective network management must involve observation and control in a holistic manner, involving both access points and clients. While some research systems [20, 12, 31] and standardization efforts [15] try to address this issue, they are hamstrung by limitations of the 802.11 design.

In this paper, we present *Dyson*, a new wireless network architecture that is designed to support *global network observation* and *historical knowledge*, *deep control*, and *extensibility* to meet future needs. In Dyson, both clients and access points coordinate with the network infrastructure and provide detailed measurements on location, radio channel conditions, connectivity, and observed performance. Measurements are stored in a persistent database, allowing the infrastructure to adapt behavior based on historical knowledge of network state. For example, the system can learn user mobility patterns in order to improve handoff performance. Further, clients and APs expose a control interface permitting the infrastructure to manage many aspects of their operation, including associations, channel selection, PHY rate, and transmission throttling. Dyson also provides a Python-based scripting API that allows the central controller's policies to be extended for site-specific customizations and new optimizations that leverage historical knowledge.

Put together, these features provide the controller with extensive visibility into the network's state, including information that can only be gleaned from clients, such as the presence of hidden terminals and signal strength from multiple APs. Dyson's control framework can yield better overall network efficiency, such as optimizing client/AP associations using knowledge of channel utilization and interference. Finally, Dyson enables a high degree of extensibility of the network's policies, making it easy to customize behavior, such as by providing user-specific airtime reservation, or shifting

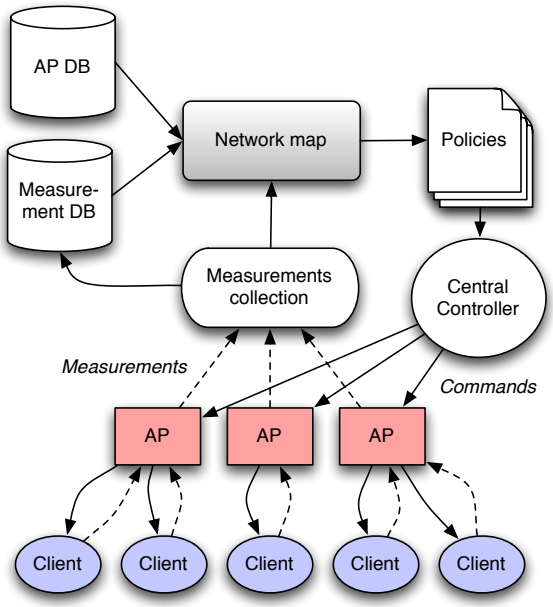


Figure 1: The Dyson network architecture.

VoIP clients to a separate channel to avoid interference. Because it is programmable, Dyson is also intended to provide a vehicle for research into new mechanisms for managing wireless LANs.

This paper makes the following contributions. First, we present the Dyson architecture (Section 2) and an implementation deployed over a 23-node testbed deployed across one floor of a typical academic building (Section 3). Second, using this testbed, we examine various aspects of the architecture in detail, and demonstrate a range of policies (Section 4) for optimizing associations, handling VoIP clients, reserving airtime for specific users, and optimizing handoffs for mobile clients. Third, we perform an extensive performance study of Dyson and show that this architecture is effective at improving network performance. We also discuss related work (Section 5) and areas for future work (Section 6).

2. DYSON ARCHITECTURE

The Dyson network architecture, shown in Figure 1, consists of a number of wireless *clients*, *access points* (APs), and a single *central controller* (CC). As described below, both APs and clients report measurements to the infrastructure, which are used to construct a dynamic *network map* representing the state of the network. Measurements are also logged to a database for historical analysis, and static information on AP location and MAC addresses are stored in a separate AP database. A set of administrator-defined *policies* are used to trigger network configuration changes via *commands* delivered to APs and clients by the CC.

Dyson builds upon existing 802.11 standards, including CSMA MAC and the format of the data and management frames. As a result, Dyson can be implemented entirely using existing 802.11-compatible hardware. The key difference between Dyson and existing enterprise WLANs is the manner in which network management and control is performed. The Dyson architecture requires both clients and APs to be Dyson-aware. However, a Dyson-enabled client can operate both in 802.11 and Dyson modes. In Section 6 we discuss mechanisms to support legacy clients.

The use of a central controller in enterprise WLANs is widespread.[1] For example, in Aruba [1] networks, the CC is responsible for assigning radio channels and transmission power levels to individual APs based on global observation of the network traffic. Dyson significantly augments this design by extending both observation and control to the wireless clients as well as the APs. Dyson clients are responsible for collecting periodic *measurements* of channel and traffic conditions and reporting them to the CC, as well as responding to *commands* from the CC that control many aspects of transmission parameters, as described below.

A key question that arises in this regime is how much control clients should yield to the infrastructure. At one extreme, the CC could control clients at a very fine-grained level, for example, by dictating individual packet transmission timings. However, this design would require substantial control overhead, and would fail to respond rapidly to local changes in channel conditions (e.g., interference) at the client. In Dyson, we opt to affect control at a higher level, namely that of channel allocations, client-AP associations and throttling. Although cruder than packet-level control, this design strikes a balance between the overhead for command issue and the ability of the network to drive towards more efficient configurations.

One implication of this design is that we assume that Dyson clients are willing participants in the system, and are capable of accurately and truthfully responding to measurement requests and commands. There is, of course, the potential that malicious or buggy clients could misbehave and degrade network performance. However, we argue that the degree of trust that Dyson places in clients is not substantially greater than that in conventional 802.11 networks, in which it must be assumed that clients correctly obey the protocol. We assume that Dyson clients are authenticated using 802.1x.

The power of the central controller derives from its global knowledge of the state of the network and ability to control both APs and clients at fine granularity. The CC also maintains a database to store received measurements, permitting long-term historical analysis of network performance.

A key benefit in Dyson is the ability to collect *client-side* measurements, providing the CC with greater visibility and control over the network state. Client-side information can be used to resolve sources of ambiguity that would arise with AP-only observations. Examples include detection of hidden terminals, awareness of mutual connectivity between APs and clients, and mapping channel airtime utilization. While client participation has been explored by several previous systems, such as MDG [12] and SMARTA [6], Dyson provides a flexible framework in which a wide range of policies can be specified programmatically.

2.1 Measurement collection

[1]Note that the CC need not be physically centralized, as this functionality can be replicated across multiple physical hosts for reliability and scalability.

Measurement	Description
numPackets	*Number of pkts received*
totalBytes	*Total bytes received*
totalRSSI	*Total RSSI of received pkts*
connectivity[]	*List of tuples* $\langle srcmac, numPkts, totalRSSI \rangle$
packetsPerPhyRate[]	*One counter for each PHY rate*
totalAirtime	*Airtime used by packets (size* $\times$ *PHY rate)*
numTxFailures	*Number of Tx failures*
numRetransmissions	*Number of ARQ retransmissions*
airtimeUtil	*Channel airtime utilization*

Table 1: Measurements collected by Dyson nodes.

In Dyson, both clients and APs are responsible for collecting passive measurements on the state of the network, reporting measurements to the CC, and responding to commands issued by the CC to affect local parameters. As described above, the granularity of measurements and commands is chosen to avoid high overheads for client/CC interactions, but still yield adequate control over client behavior by the infrastructure.

Measurement collection in Dyson supports network-wide optimizations based on both AP and client-side knowledge of the network state. This provides the CC with global information on various factors that affect client performance, such as traffic patterns, interference, hidden terminals, and congestion. This approach obviates the need for a separate wireless monitoring infrastructure [14, 8].

Each client and AP in the system records a set of statistics, summarized in Table 1. For each received packet, a set of counters are incremented to track the total number of packets, total packet size, total airtime utilization, and other measures. Dividing counters by the number of received packets can be used to calculate mean values over a measurement window. Clients maintain a single set of these counters, whereas the AP maintains these counters for each associated client, allowing measurements to be collected for each separate uplink. In addition to the per-packet statistics, nodes record the mean airtime utilization (reported by the radio hardware) of the radio channel.

APs periodically query their associated clients for their measurements, which report the data to the AP and clear their counters. The AP then pushes the collected client measurements, as well as its own, to the CC. The AP's measurement collection period can be adjusted by the CC to tradeoff reporting latency and measurement traffic overhead. Our measurements in Section 4.7 show that for moderate-sized networks, the traffic overhead is less than 1%.

2.2 Network map

The central controller uses collected measurements to maintain a *network map* representing the global state of the Dyson network. The network map is the key data structure accessed by Dyson's policies (Section 4) in order to drive reconfiguration. The network map is updated each time new measurements are pushed to the CC by an AP. Policies can read the complete network map as well as push new information into the network map. This allows individual policies to augment the global state maintained by the CC, as well as policies to be composed.

The map consists of several components:

- **Node location:** A table of the physical location of each AP and client in the system, indexed by MAC address. AP locations are static, whereas client locations are computed using the algorithm described in [13]. This information can be used for determining the physical location of network hotspots, as well as for reducing handoff latency for mobile clients, as described in Section 4.6.

- **Connectivity:** A directed connectivity graph is maintained, where vertices represent nodes (clients or APs) and edges represent the ability of one node to overhear packets of another node. For each unique MAC address that a node overhears during a measurement interval, the mean RSSI value of packets from that MAC address are reported to the CC. The connectivity graph contains a directed edge for each pair of MAC addresses. While clients are only capable of reporting links on their current channel, APs can use a secondary radio to perform background scanning and report observed connectivity on every channel. An edge is removed from the graph if no packets are observed on the link for 30 sec. The connectivity graph is used in assigning clients to APs, detecting hidden terminals, and managing handoffs.

- **Airtime utilization:** Each node measures the airtime utilization of the radio channel in its vicinity. The network map includes a hash table mapping a node's MAC address and channel number to its airtime utilization estimate. This information can be used by a wide range of policies to detect congestion, balance uplink and downlink fairness, and optimize client/AP associations. APs can measure airtime on every channel using the secondary scanning radio.

- **Historical measurements:** Collected measurements are also stored in a persistent database, permitting policies to make use of historical information when making decisions about network reconfiguration. As an example, a policy may wish to consider the historical interference pattern between two APs, or variance in the network congestion at different hours of the day, when driving network reconfigurations. The handoff optimization policy described in Section 4.6 uses historical information on node connectivity and mobility patterns to improve handoff latency.

The network map serves primarily as input to the various policies for driving network configurations. However, it can also serve an auxiliary role to assist a network administrator in understanding AP coverage and sources of performance degradation. For example, visualizing the airtime utilization graph as well as the associated client and AP locations can provide real-time information on network hotspots.

2.3 Central controller

`SetRate(`r`)`	Set PHY rate
`SetChannel(`c`)`	Set channel
`SetTxLevel(`t`)`	Set transmission power level
`SetCCAThresh(`t`)`	Set CCA threshold
`SetPriority(`p`)`	Set 802.11e priority
`Throttle(`r`)`	Throttle outgoing traffic at the specified rate r
`Handoff` $(c, ap, chan)$	Handoff client c to AP ap on channel $chan$
`AcceptClient` (c)	Associate AP with client c
`EjectClient` (c)	Disassociate client c

Table 2: The Dyson command API. Commands in bold are applicable to APs only.

Figure 2: Dyson testbed deployment

The central controller is responsible for managing the entire Dyson network based on collected measurements from clients and APs. Its job is to apply administrator-defined *policies* to the current network map, and issue *commands* to set parameters of clients and APs according to the policy decisions.

The Dyson command API is shown in Table 2. These commands are intended to provide a rich set of knobs for controlling the network's operation while limiting overheads for command issue. Commands set parameters such as the transmission power level, CCA threshold, 802.11e priority levels, and PHY data rate. The `Handoff`, `AcceptClient`, and `EjectClient` commands control client-AP associations, as described in the next section. Note that clients do not decide themselves which AP to associate with; this is under the control of the Dyson infrastructure.

The CC sends commands to APs directly. Commands to clients are relayed through the AP with which the client is associated; in this way the client need not be aware of the CC's identity, and the CC's functionality can be decentralized. Commands are exchanged using MAC-layer control messages which are ACKed by the receiving node. For AP-client commands, ARQ is used to ensure commands are delivered reliably.

2.4 Policy Engine

Dyson's architecture is designed to support extensibility, composability, and separation of concerns, in order to tune network performance as well as impose site- and client-specific policies. Each policy is encapsulated in a software module that runs on the CC, takes the network map as input, and issues commands to APs and clients as output. As described above, policies can also update and augment the network map itself.

Dyson has a predefined set of policy modules providing commonly-used functionality, but it is possible for new policies to be implemented and loaded into the central controller as needed. Policies are implemented in Python and are relatively easy to write, as we will show below. This approach enables network designers to update the policies used by a Dyson network installation over time in response to new demands or shifting priorities. We also envision third parties developing new policies for Dyson that can be readily plugged into an existing deployment.

In our current design, policy composition and dependencies must be handled manually by policy designers. There is nothing to prevent two policies from "competing" (say, by issuing conflicting commands in response to the same event in the network); each policy should clearly document its own behavior to avoid unexpected results.

Each policy runs as a separate thread on the CC and is responsible for its own scheduling. Typically, a policy will run with some predefined period, but a policy can also trigger execution on some condition being met (for example, an update to some element in the network map). Standard thread synchronization primitives can be used to implement more sophisticated cross-policy interactions.

In Section 4, we demonstrate a set of policies that highlight different aspects of Dyson's global network visibility and deep control over both APs and clients.,

3. IMPLEMENTATION AND TESTBED

We have implemented a prototype of the Dyson architecture using the ALIX 2c2 single-board computer (500 MHz AMD Geode processor with 256 MB DRAM) running FreeBSD 7, coupled with dual CM 9 Atheros-based 802.11a/b/g radios. Each node can act as either a Dyson client or an access point; only APs make use of the second radio for collecting channel utilization measurements.

We have deployed a testbed to 23 nodes across one floor of an academic office building, as shown in Figure 2. Each node is connected to an Ethernet network for control. The central server is implemented on a separate machine running FreeBSD with 2 GB of RAM. All experiments presented in this paper use 802.11a to avoid interference with existing 802.11b/g networks in the building.

To support Dyson, it was necessary to modify the FreeBSD Atheros driver to add support for statistics collection and the Dyson command API, as well as to disable local autorating. Each node runs a Python-based daemon that exposes the Dyson measurements and command API via an XML-RPC interface, and communicates with the modified Atheros driver through *ioctl* calls. The central controller is also implemented in Python; policies are loaded as Python modules at startup time.

The commands listed in Table 2 were implemented via modifications to the Atheros driver. Most of the commands (such as `SetTxLevel`, `SetChannel`, and so forth) simply set driver parameters. `Handoff` informing a client to switch channels and associate with the specified AP. This

eliminates the need for scanning, provided the destination AP is still on the specified channel. The `Throttle` command makes use of *dummynet*, a FreeBSD traffic shaping tool, to limit the rate of outgoing traffic. `Throttle` simply sets the dummynet outgoing bandwidth limit on the radio interface to the specific rate.

4. POLICIES AND EVALUATION

To illustrate the power of the Dyson framework, in this section we describe and evaluate six separate policies that demonstrate the key facets of our design. In particular, we present the following policies implemented using Dyson's policy API:

1. *Capacity aware associations* that assigns clients to APs based on airtime availability;
2. *Interference mitigation* using connectivity information from clients and APs;
3. *VoIP-aware handoffs* that causes VoIP and bulk clients to be assigned to different APs;
4. *User-specific airtime reservations* that gives priority to certain clients over others;
5. *Uplink/downlink load balancing* to mitigate the impact of bulk upload clients on throughput fairness; and
6. *Predicting handoffs* based on historical knowledge of a user's mobility patterns.

Taken together, these policies highlight Dyson's key design elements: client-side observation, use of historical knowledge, site-specific customization, and deep control. We also present a range of microbenchmarks to evaluate the scalability and performance of the Dyson design.

Each of these policies is intended to demonstrate Dyson's capabilities, rather than serve as an optimal solution to a particular problem. A great deal of prior work [6, 7, 12, 20, 21, 22, 24, 29, 30] has investigated each of these problems in detail. Our claim is that the Dyson architecture opens up the network infrastructure, permitting rapid innovation and greater visibility and control over the network as a whole. Dyson is also intended to be general-purpose and support all of these disparate policies within a single framework.

4.1 Capacity-aware association

Dyson places control over client-AP associations with the infrastructure. Prior work [20] demonstrated the efficacy of an intelligent centralized association policy in WLAN networks with the overall goal of increasing aggregate capacity. The key idea is to use information on airtime utilization and an estimate of the feasible PHY rates to determine the best AP with which to associate a given client. In DenseAP [20], this policy was implemented in an *ad hoc* manner; with Dyson, it is readily implemented as a short (under 30 lines of code) policy module in Python, as shown in Figure 3.

The policy runs every 5 s. On each iteration, it scans over a list of received probe requests from clients. A given probe request may have been overheard by multiple APs. For each

```
# Input: client MAC, list of (AP MAC, RSSI) for
#   each received probe request
# Output: client MAC, AP with highest
#   available capacity
def (clientmac, heard_list):
  global ap_list, ap_list_lock, ratemap
  best_ap = None
  max_ac = -1

  # Compute available capacity for each AP
  # Pick AP with the highest value
  for (apmac, rssi) in heard_list:
    ap_list_lock.acquire()
    data_rate = ratemap.get_rate(rssi)
    airtime = ap_list[apmac].airtime
    avail_capacity = data_rate * (1.0 - airtime)
    if avail_capacity > max_ac:
      max_ac = avail_capacity
      best_ap = ap_list[apmac]
    ap_list_lock.release()

  # Assign channel if no clients already
  if (best_ap.channel == -1):
    best_ap.assign_channel()
  # Associate client
  best_ap.AcceptClient(clientmac)

def run(self):
  global pending_associations
  global pending_associations_lock

  while (True):
    pending_associations_lock.acquire()
    map(compute_ac, pending_associations)
    pending_associations = []
    pending_associations_lock.release()
    time.sleep(5)
```

Figure 3: The Dyson capacity-aware association policy.

AP, the *available channel capacity* is computed, which is the product of the estimated PHY rate at which the client and AP will communicate, and the inverse of the AP's measured airtime utilization. The PHY rate is determined using a *rate map* that maps the RSSI of the received probe request to the max feasible PHY rate for that client/AP pair. The rate map computation is performed separately and not shown in the code in Figure 3.

The AP with the maximum available capacity is selected as the one with which to associate the client. If the AP currently has no clients, a channel is assigned to it, and the AP is then instructed to accept the client's probe request (by sending a probe response). This policy is used as the default association policy in Dyson and is used by the subsequent policies unless otherwise specified. We have performed experiments to confirm that its performance is similar to DenseAP's association scheme [20]; these results are omitted due to lack of space.

4.2 Interference mitigation

All wireless networks suffer from interference, both from sources within the network and external ones. The problem of mitigating the impact of interference from nodes within the network has received much attention from the research

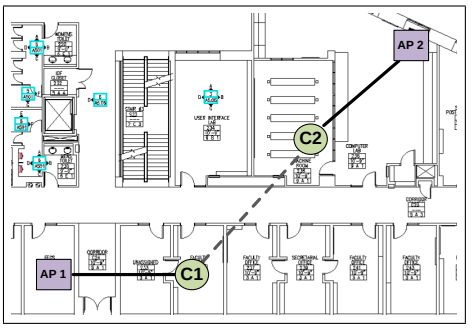


Figure 4: Interference example. The two clients determine that they interfere with each other, despite being associated with different APs.

community. Dyson's global connectivity graph can be used to mitigate the effects of interference between nodes, for example, changing AP channel assignments.

In some cases this interference can only be detected by clients. Figure 4 shows one example in which two clients, C_1 and C_2, are associated with different APs that happen to be on the same channel. The APs themselves would not readily recognize the interference condition. Dyson's network-wide measurements collection permits deeper inspection of interference relationships than can be obtained at APs alone.

We implemented a simple policy that periodically scans the global connectivity graph and detects cases in which two APs and two clients form an interference relationship similar to that in Figure 4. The policy changes the channel of the AP (and its clients) with the smaller number of associated clients. The affected nodes are informed of the channel switch directly via a command, thereby avoiding the overhead of re-discovery and re-association if the policy were to simply change the channel of the AP. Note that this simple greedy algorithm might induce a new interference condition elsewhere in the network, necessitating another channel switch. To avoid oscillatory behavior, we do not change an AP's channel more than once every 10 minutes.

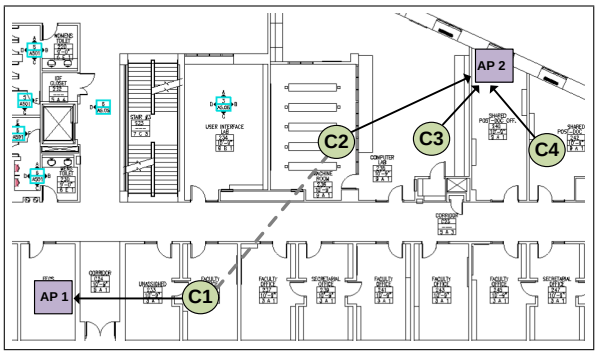
Evaluation

To demonstrate this policy in action, we set up two APs and clients in our testbed described in Section 3, as shown in Figure 5(a). Our setup consisted of every client sending uplink traffic to the AP it is associated with. We used *iperf* to generate saturating UDP flows.

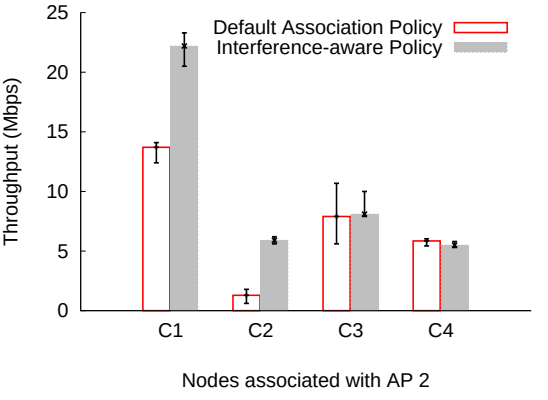
We first ran the capacity-aware association policy, resulting in the client associations shown in Figure 5(a). These APs do not interfere with each other, and the association policy assigned both APs to the same channel. As a result, clients C1 and C2 interfered with each other, and the throughput of both clients suffered.

Next, we ran the interference-aware correction policy. The policy correctly detected the problem, and fixed it by moving AP1 to a different channel, and re-associated the clients. As a result, clients C1 and C2 no longer interfered with one another. The resulting improvement in throughput of *all* clients is evident in Figures 5(b).

This policy leverages Dyson's ability to use client coop-



(a) Node layout showing interference between C1 and C2.



(b) Throughput impact of interference mitigation on AP2's clients.

Figure 5: Impact of interference mitigation policy.

eration effectively in improving the performance of the network. Without cooperation from the client (which provides connectivity measurements), global knowledge (which channel is each AP on, its clients, and their connectivity measurements), and deep control (ability to control both clients and APs), it would not have been possible to detect and address this problem. A more complex version of this policy can historical knowledge of the network into account. For example, once an interference pattern between locations is determined, the system can proactively assign APs and clients in those locations to different channels.

4.3 VoIP-aware handoffs

As a another example of the power of Dyson to enable network-wide optimizations, we present an example policy in Figure 6 that assigns VoIP clients to a different set of APs than other clients, to increase overall VoIP call capacity and avoid bulk transfers from impacting VoIP call quality. This policy assumes that clients have been classified as VoIP or non-VoIP clients, for example, based on the client's MAC address (e.g., for WiFi VoIP handsets).

For each VoIP client that is assigned to a non-VoIP AP, the policy identifies a new VoIP-specific AP with which to associate. For each VoIP AP that the client can potentially connect to (based on the connectivity graph), the available capacity metric is computed, as described earlier. The client is simply handed off to the VoIP AP with the highest avail-

```
# Determine if two nodes within range of each other
def can_hear(node1, node2):
  return (connectivity[node1][node2] == 1 and
          connectivity[node2][node1] == 1)

# Find best VoIP AP and handoff client to it
def do_handoff(client)
  global ap_list, ap_list_lock
  max_ac = -1
  best_ap = None
  for ap in aplist:
    if (is_voip_ap(ap) and can_hear(ap, client)):
      # Select AP with highest available capacity
      # (Code not shown...)
  client.ap.Handoff(client, best_ap)

# Return only VoIP clients
def cull_voip_clients(c):
   return c.is_voip() # Based on MAC addr

def run(self):
  global ap_list, ap_list_lock
  while (True):
    ap_list_lock.acquire()
    for ap in apmap:
      # Only worry about non-VoIP APs
      if (is_voip_ap(ap)): continue

      # Get list of VoIP clients
      voip_clients = filter (cull_voip_clients,
        ap.clients)

      # For each VoIP client, find nearest AP
      for c in voip_clients: do_handoff(c)

    ap_list_lock.release()
    time.sleep(10)
```

Figure 6: VoIP-aware handoff policy.

able capacity.

Although there are more sophisticated techniques to improve VoIP capacity in WiFi networks [30], this policy is simply intended to demonstrate Dyson's interfaces and programmability. This simple policy can be extended in various ways. For example, the assignment of APs as VoIP or non-VoIP (which is currently static) can be performed in a dynamic fashion based on VoIP call load. Likewise, the number of VoIP clients assigned to each AP could be taken into consideration. We elide the details due to lack of space.

Evaluation

We carry out the following experiment. We configured two nodes near each other as access points, and another four nodes as clients. The capacity-aware association policy described in Section 4.1 was used, resulting in two clients being associated with each AP. The APs were assigned to different channels by the association policy.

Two clients, on separate APs, initiated a bidirectional VoIP flow while the other two clients began large saturating download traffic using *iperf*. The VoIP flows each use a standard g729 VoIP codec that generates 50-byte packets at a rate of 31.2 Kbps.

The bulk flows adversely affect the VoIP flows in terms of

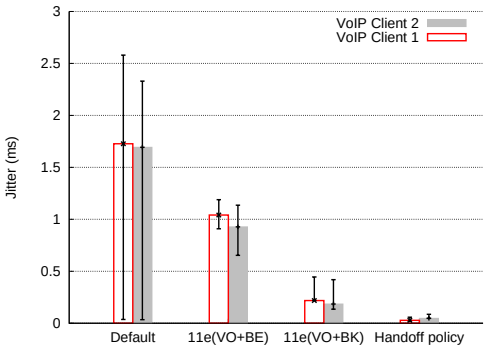


Figure 7: Effect of 802.11e prioritization and VoIP-aware handoffs on VoIP jitter. ***This is an experiment with two VoIP clients competing with two bulk-download clients, with two access points on different channels. Using the* default *policy, one VoIP client and one bulk client are assigned to each AP. The* 11e(VO+BE) *policy uses 802.11e prioritization, assigning bulk clients to the* best effort *queue.* 11e(VO+BK) *assigns bulk clients to the* background *queue.* Handoffs *uses handoffs to segregate the VoIP and bulk clients on different APs.***

introducing increased packet jitter, which causes the quality of the VoIP call to degrade. A common requirement for VoIP calls is that jitter should be no greater than 2ms [3]. Figure 7 shows that with the default configuration, up to 2.17 ms of jitter is induced by the bulk flows on each VoIP call. Keep in mind that this is with a small number of clients.

Next, we enabled the VoIP handoff policy, which migrates the VoIP clients to one of the APs and the bulk flows to the other. As Figure 7 shows, this substantially reduces the jitter to a mean of 0.02 ms. Of course, this also causes the bulk transfers to share the channel on a single AP, causing their throughputs to degrade; prior to migration, each bulk flow obtained 24 Mbps of throughput. After migration, each bulk flow degrades to 12 Mbps. This is an explicit tradeoff between providing good service to VoIP clients versus the (arguably less severe) impact on bulk flows.

As an alternative, we also experimented with using 802.11e priority levels, with a simple policy that uses the `SetPriority` command. We set up one experiment in which the VoIP clients were configured to use the 802.11 *voice* priority and the bulk clients to use the 802.11e *best effort* priority, while maintaining the original AP associations. Another experiment uses the 802.11e *background* priority, which is lower than best-effort. As the figure shows, 802.11e priorities do mitigate some of the jitter effects, but do not operate as well as the handoff policy. Each bulk client received 24 Mbps of throughput using the best-effort priority, and 18 MBps using the background priority. In general, it will not always be possible to cleanly separate VoIP clients from others in the network, so in general a combination of migration (where possible) and 802.11e priority levels is likely to be the most effective solution.

Note this policy could have been implemented in prior systems, such as SMARTA [6], MDG [12], or DenseAP [20].

However, Dyson enables a network designer to develop and deploy policies such as this one within an organized framework. This arises from the rich control interface and programmatic API. Therefore, Dyson can facilitate more complex versions of this kind of policy. For example, the CC could dynamically determine the number of APs in a given area that should be devoted exclusively to VoIP traffic based on traffic demands and client locations.

4.4 User-specific airtime reservation

While current WiFi networks are capable of prioritizing traffic, they are not capable of reserving a certain fraction of airtime for a specific user or groups of users. However, the network designer can easily accomplish this task using the Dyson framework. We implemented a simple policy that reserves a fixed amount of airtime for a preferred user. A high-priority client c_h is identified by its MAC address. For all other clients $\{c_1, c_2, ...c_k\}$ associated with the same AP, the residual airtime $R = 1 - \sum_i ATU(c_i)$ is computed. If the residual airtime is less than the target airtime for c_h, the policy iterates through the list of non-high-priority clients, and *throttles* each of their transmission rates by 10% of their current throughput. This is performed using the Dyson `Throttle` command to the clients, shown in Table 2. Throttling is performed periodically until the residual airtime exceeds the target.

This approach makes no assumptions about the nature of client traffic, and simply "searches" for the throttle setpoints that yield adequate airtime to the high-priority client. It is also conservative, in the sense that clients are throttled equally, without regards to their load. A straightforward enhancement would throttle higher-load clients first. Note that when c_h disassociates with the AP, the low-priority clients are unthrottled; likewise, when a client moves to another AP is throttle is released. Multiple high-priority clients can also be supported on a single AP as long as their airtime targets do not exceed 100%; in that case, each high-priority client receives a weighted proportional share of the airtime.

Evaluation

We first demonstrate the efficacy of this policy in a limited setting, and then move onto a larger setting. The setup consisted of an AP and two clients placed in close proximity of each other. These are deemed to be regular clients. We then introduce a third high-priority client with an airtime reservation target of 50%. All clients in the system were performing downloads using *iperf*. For the purposes of exposition, the policy runs every 10 sec. As seen in Figure 8, the policy correctly detects that the high-priority user has not received the requisite share of airtime and throttles the regular users to compensate.

We now demonstrate this policy in a larger setting across different APs. The setup consists of four APs (AP1, . . . AP4) and 11 clients. As before, one of the clients is given an airtime reservation of 50%. For this experiment, we suspend the capacity-aware association policy and we manually set the APs are on different channels, and associate one non-privileged client with AP1, two non-privileged clients with AP2, and so on. The privileged client is nomadic. It associates with each of the four APs in turn for 10 minutes each.

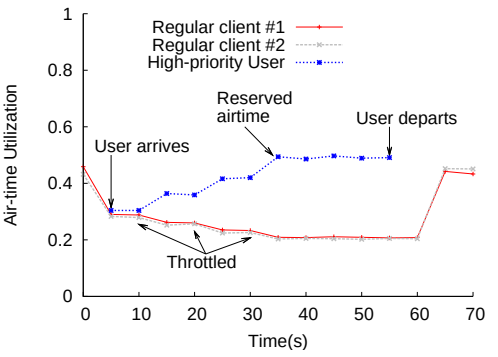


Figure 8: A time series plot of the policy in action. The setup consisted of two clients in close proximity associated with the same AP. The high-priority user arrives shortly before 5s and departs at 55s. The throttle for regular users is released once the high-priority user departs.

All clients download data as fast as they can using *iperf* UDP flows. We first perform the experiment without any reservation policy, and then repeat it by reserving 50% of the airtime for the privileged user. We repeated the experiment 10 times.

The impact of the policy is shown in Figure 9. In the absence of the reservation policy, the fraction of airtime received by the privileged user drops as the number of non-privileged clients increase. However, when the reservation policy is in force, the privileged user always receives the 50% reserved fraction of the airtime.

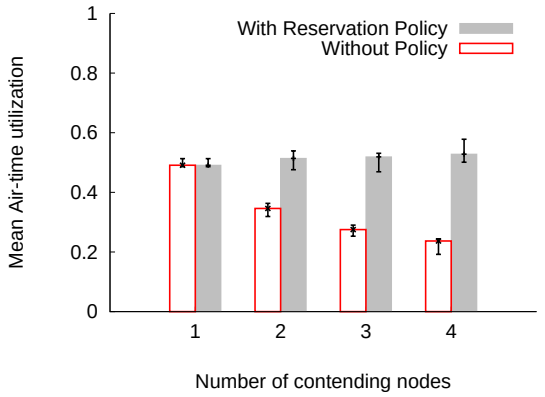
The throughput received by the privileged client is shown in Figure 9(b). Notice that the even though the privileged client receives a fixed amount of airtime, the throughput it achieves varies for different APs. This is primarily due to variability in the quality of the radio link between the privileged client and each of the APs.

This particular policy highlights two aspects of the Dyson architecture. First, its ability to use throttling as a feasible mechanism in providing quality of service to clients in a wireless network. Second, this is an example of a site-specific customization that is readily implemented through a small amount of Python code at the central controller. This is just one example of a range of policies that can be implemented to prioritize different users or traffic classes.

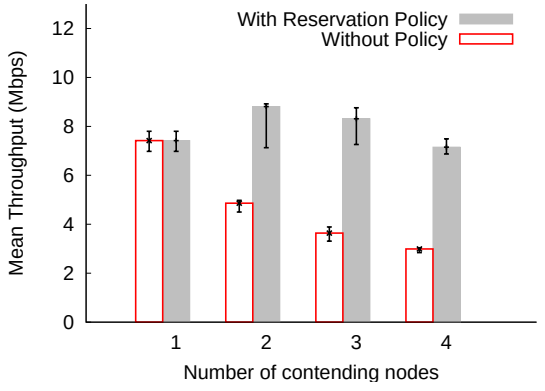
4.5 Uplink/downlink load balancing

In 802.11, fairness is arbitrated on a per-node basis. As a result, APs and clients all have the same number of transmission opportunities over time. As a result, downlink traffic from APs to clients is limited by the AP's ability to acquire the channel in the presence of multiple competing uplink flows [24]. Although various studies have shown that 80% of WLAN traffic tends to be downlink [4], even a small number of uplink flows can impact fairness.

To address this problem, we implemented a Dyson policy that attempts to balance the total volume of uplink and downlink traffic handled by an AP. For each AP, associated clients are classified as either predominantly upload or download, based on the ratio of their throughput in each direction. We

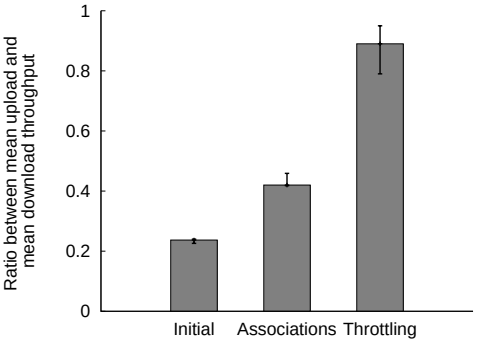


(a) Impact on airtime

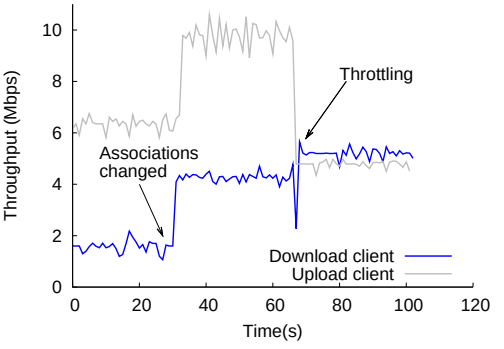


(b) Impact on throughput

Figure 9: Impact of airtime reservation policy on the airtime and throughput received by a single privileged user competing with several other clients. Error bars represent 10th and 90th percentiles.



(a) Ratio of median download throughput to median upload throughput



(b) Timeseries

Figure 11: Uplink/downlink anomaly and its resolution.

then compute the ratio of the mean throughput for upload and download clients. If the ratio exceeds a specified threshold, it suggests that upload clients are dominant and that rebalancing is required for this AP.

As a simple approach, the policy throttles upload clients in an attempt to bring the upload/download ratio closer to 1. Upload clients are ordered by decreasing uplink throughput, and the "heaviest" upload client is throttled to 50% of its current throughput. The policy then sleeps for 10 sec and re-evaluates the upload/download ratio, iteratively throttling the highest-throughput upload client until the ratio between the mean upload and mean download throughput at the AP falls is less than 1.5.

Evaluation

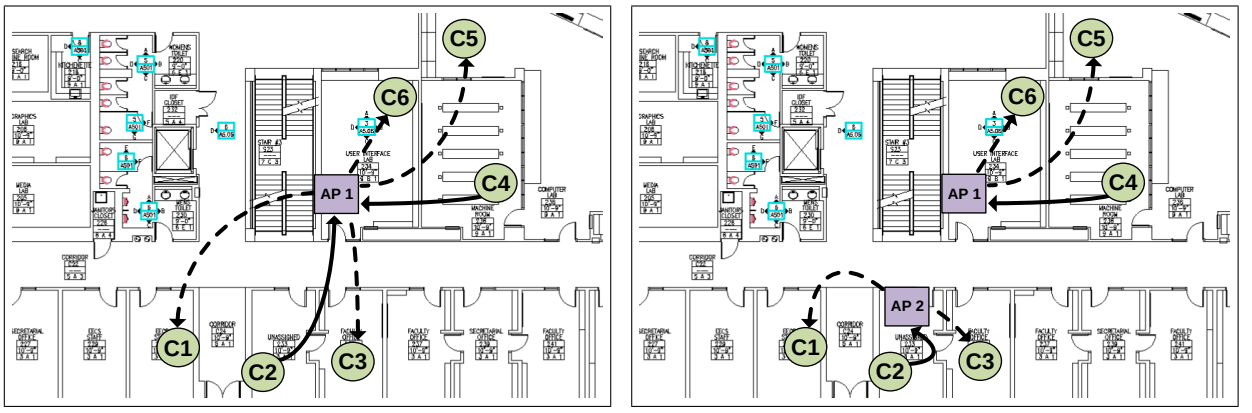
We conduct the following experiments to demonstrate this policy in action. The setup consists of two APs and 6 clients. We initially forced all clients associate with AP1, as shown in Figure 10(a). Two clients begin uploads to a server on the wired network, and four clients begin downloads from the same server. The traffic consists of saturating UDP flows generated using *iperf*. We have verified that neither the wired network nor the server is a bottleneck at any time.

A total of three stations are contending for the channel at any time (the two upload clients and the AP). Therefore, each upload client gets 1/3 of the transmission opportunities, and the four download clients together share the remaining 1/3. All other things being equal, the throughput ratio between each of the four download clients and each of the two upload clients should be 1:4. Figure 11(a) shows the median download/upload throughput ratio taken over a set of 10 experiments. The ratio is 0.237, which is as we would expect.

The first question is how much this situation can be improved by migrating some of the clients to a separate AP. In Figure 10(b) we enable the second AP and manually assign one upload client and two download clients to it. As shown in Figure 11(a), the throughput ratio between download and upload clients is about 1:2, as we would expect.

However, even after reassociating clients, there is still a significant throughput inequity. We next enable the Dyson uplink/downlink load balancing policy, which throttles the upload clients on each AP until the throughput ratio is closer to 1. Figure 11(a) shows that we achieve a ratio of 0.89 after the policy is enabled. Figure 11(b) shows the throughput for one upload client and one download client over time, as well as the points at which clients were reassociated and the upload client throttled.

To study this effect at a larger scale, we performed an experiment with 4 APs and 15 client nodes. Client-AP associ-



(a) Original associations (b) Improved associations using a second AP

Figure 10: AP-client associations for the uplink/downlink load balancing experiment. Dashed lines represent download flows and solid lines represent upload flows.

ations were determined using the capacity-aware association policy (Section 4.1). Note that different APs have a different number of associated clients. Each AP is assigned to a different channel by the policy.

One client associated with each AP generates upload traffic, while others generate download traffic. We ran the experiment twice, first without the uplink/downlink load balancing policy running, and then with the policy enabled. Figure 12(a) shows the distribution of the throughput obtained by each of the clients with and without the policy running. There is a clear bandwidth inequity in the default case, but the policy produces a much more balanced distribution of network capacity to each client.

Of course, achieving fairness is often at odds with maximizing overall network capacity. Figure 12(b) shows the aggregate throughput at each AP before and after the policy was enabled. As the figure shows, there is only a slight dip in overall bandwidth usage at each AP, 5.7% on average.

This scenario illustrates a number of features of the Dyson architecture. First, notice that the problem cannot be solved without cooperation from the clients. This is an inherent limitation of AP-only systems, such as DenseAP [20]. Second, notice that the Dyson policy interface is quite flexible: we could have just as easily designed a policy to achieve other criteria, as in the airtime reservation case described earlier. Also, it demonstrates an effective feedback loop in terms of control and information. Information from clients about their airtime and throughput is used to make changes to clients in the network with the overall goal of bringing about fairness.

4.6 Handoff prediction

Dyson can use historical knowledge of client mobility patterns to optimize AP handoffs. Since mobile handoffs are expensive and can lead to temporary connectivity loss, it is important to avoid redundant or poorly-chosen handoffs. The key idea is to predict the next AP a client will encounter while roaming, in order to avoid handing off to a different AP that will quickly go out of range. This is possible, since in many workplaces, users are more likely to travel along certain paths than others.

While many handoff prediction algorithms have been studied [26, 28], as a simple demonstration we make use of an order-1 Markov predictor, which is constructed as follows. For each client, *training data* is collected that consists of the history of the client's handoffs, represented as a sequence of tuples $\{h_1, h_2, ...h_n\}$ where $h_i = \langle loc_i, AP_i \rangle$ indicating the location and AP identity for each handoff in the trace. Next, we compute the conditional transition probabilities P that indicate the next AP to be assigned at each location as follows:

$$P(AP_{i+1} = a' | loc_i = l) = \frac{N(l, a')}{N(l)}$$

where $N(l, a')$ represents the number of times that the client trace contains a sequence of two tuples $\{\langle l, a \rangle, \langle l', a' \rangle\}$, and $N(l)$ is the number of tuples containing location l.

When a client requests an AP handoff (due to the RSSI of its current AP dropping below a threshold), the handoff prediction policy determines the *most likely* AP, $\hat{a}$, based on the client's current location l as $\hat{a} = \arg\max_{a \in A} P(a|l)$. Although much more sophisticated optimizations are possible, this approach works well in practice. The system can further improve the prediction process by learning from both its failures and successes. Our current policy does not implement these algorithms.

Evaluation

To evaluate the effectiveness of this approach, we designed an experiment in which we configured 7 APs at various locations and had a single mobile client roaming the building along several paths. Figure 13 shows the layout and the paths traveled. Each path was traversed a different number of times, as shown in the figure.

The handoff optimization policy constructs a Markov model to estimate the next AP to be encountered at each location. For example, when the user approaches location B, the historical mobility data suggests that it is more likely that she will turn left (towards locations C or D), rather than right (towards location E). This implies that at location B, the next

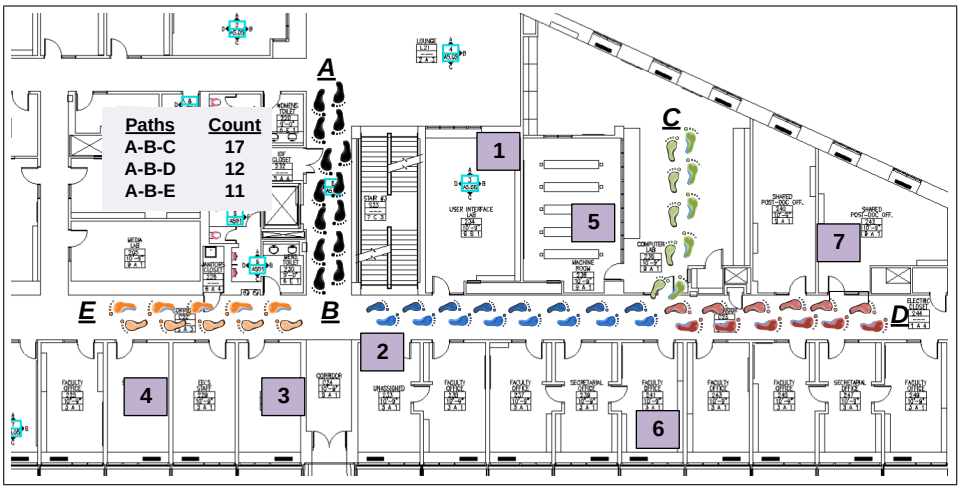


Figure 13: Mobility paths discovered by the optimized handoff policy. This figure shows a partial map of our testbed with access point locations labeled with numbers. The walking pattern of the mobile user is indicated using footprints. Points A, B, C, D, and E represent start and end points for the different paths. Each unique path segment determined by the policy is labeled with a separate color. The inset shows the number of times each path was traversed during our training session.

AP to be used should be AP 2, rather than APs 3 or 4.

After the training phase, we repeated the experiment with and without the handoff prediction policy enabled. During the walk, the mobile client ran a g.2347 audio VoIP session, and we measured the number of handoffs incurred during each handoff. Each path was traversed the same number of times as during the training phase.

Figure 4.6 shows the number of handoffs experienced by the mobile clients, with and without the policy. As expected, the client undergoes fewer handoffs when the prediction policy is enabled. While this is a simple approach, this policy demonstrates the value of collecting historical information and using it to tune the network's behavior in specific ways. Dyson's programmatic interface makes it easy to implement such policies, and provides a vehicle for exploring a range of algorithms.

4.7 Microbenchmarks

There are two aspects of the Dyson system that impact its performance and scalability. The first is the overhead induced by handoffs. While handoffs affect the performance of any wireless network, they are used extensively by Dyson's policies for handling mobility, interference mitigation, and segregating VoIP clients. The handoff overhead has ramifications on the agility of the system. The second factor to consider is the load imposed by clients and APs on the CC, as well as the CC's ability to react quickly to changes in network state. In this section we measure these aspects of the system.

Handoff overhead: We measured the time taken for a MAC-layer handoff of a client from one AP to another. We configured two APs (on different channels) and a single client, which was initially associated with AP1. The CC then issued a `Handoff` command to migrate the client to AP2. AP1 receives this command and relays it to the client who quickly switches channels and associations. This process also includes informing AP2 to permit the new association.

Step	Time(ms)
Handoff command executed	0
Message reception (at client)	0.120
Channel change	5.6
Authentication	0.159
Association	0.359
Total	6.238

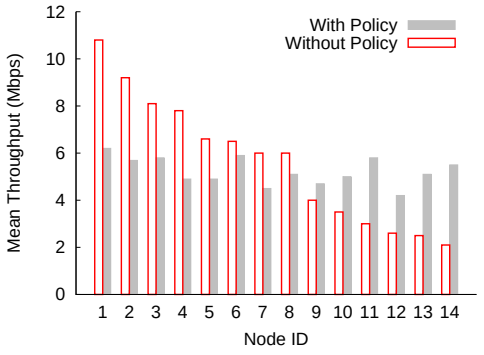
Table 3: Handoff overhead in Dyson.

The MAC-layer handoff overhead includes the time for the command transmission to the client, the time for the client to switch channels (between to 5 to 7 ms on the Atheros chipset), as well as the client's reassociation with the new AP. Of course, the end-to-end delay experienced by an application will be longer, for example, due to the settling time of the spanning-tree algorithm on the wired backbone.

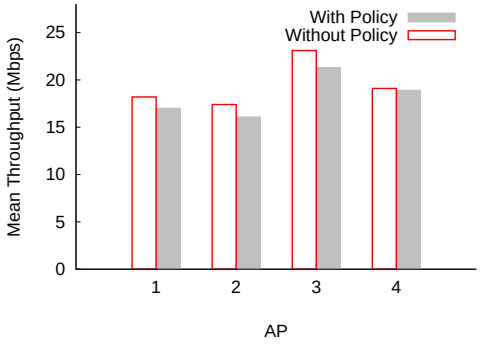
The results are shown in Table 3, which shows that a MAC-layer handoff requires approximately 6.2 ms in our current prototype. This process can be further optimized, as demonstrated in [25]. Also, the use of protocols such as IAPP (Inter-Access Point Protocol) at a higher layer, in which APs cache packets during a handoff and forward them to the destination AP, can mitigate the packet loss incurred during a handoff. We have not yet implemented this approach in Dyson.

Central controller scalability: A number of factors impact the performance of the Dyson central controller, including the number of clients, the number of active policies, and the interval at which APs report statistics. We measure the effect of each of these factors, configuring the testbed with 6 APs and 17 clients.

We enabled the default capacity aware association policy at the CC. To exercise it, over a period of 10 minutes



(a) Throughput for various nodes with/without load balancing



(b) Total throughput at each AP with/without load balancing

Figure 12: Large-scale uplink/downlink anomaly experiment.

we repeatedly manually disassociated clients and then reassociate them with the AP. We repeated this experiment while varying the statistics reporting interval from 1 to 10 sec. As shown in Table 4, the median CPU utilization incurred even at aggressive reporting intervals is very small.

With the same setup, we enabled three different policies at the CC and repeated the same experiment. As seen in Table 5, in all cases, with 23 nodes the central controller's CPU utilization is still fairly low.

Access point and client overheads: We are also concerned with the CPU utilization of the access point. Recall that we use an ALIX 2c2 with a 500 MHz AMD Geode, and the Dyson software is implemented in Python. We configured an AP with eight clients and varied the intervals at which the AP reported statistics to the CC. As seen in Table 6 the median utilization is still low.

Since clients are periodically sending statistics to the AP, we also measured the CPU utilization at a client over a period of ten minutes. We found the modified Dyson drivers added negligible overhead in terms of CPU and memory utilization ($< 1\%$).

Traffic overhead for measurements collection: We also measured the traffic sent by clients and APs to the CC. The AP's measurement collection period can be adjusted by the CC to tradeoff reporting latency and measurement traffic overhead. As an estimate of this overhead, each client measurement packet requires at most 850 bytes, including MAC headers. At the lowest OFDM PHY rate of 6 Mbps, this requires 1184μs to transmit (accounting for MAC and framing overheads). Therefore, an AP with 20 clients will require less than 1% of the radio channel for statistics collection. The AP sends all client statistics as well its own statistics to the CC. We measured the traffic sent by an AP with six clients to the CC. With a statistics reporting interval of 5 sec, the AP generates 1638 bytes/sec in traffic to the CC, which includes overheads induced by the use of XML-RPC. This is a small fraction of the backhaul wired network capacity.

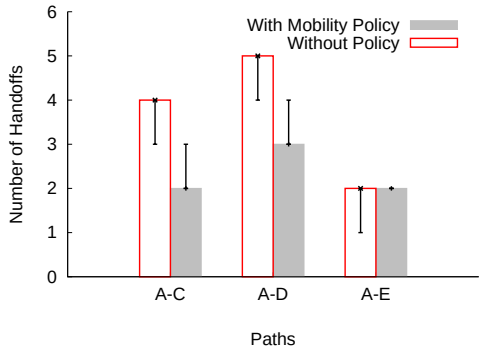


Figure 14: Impact of the handoff prediction policy on number of handoffs.

Stats interval (s)	10th	Median	90th
1	0.1%	1.4%	2.3%
5	0.1%	0.5%	1.1%
10	0.0%	0.2%	1.3%

Table 4: CPU utilization at the CC measured over a period of 10 minutes for varying statistics reporting intervals.

5. RELATED WORK

Dyson is complementary to a broad class of prior work on improving the performance and scalability of wireless networks through new techniques at the MAC and PHY layers [29, 10, 16]. Our focus is on the higher-level aspects of network management that can be obtained through global observation and deep control.

Several commercial systems use some form of global knowledge or a central controller for managing WLAN deployments. Aruba [1] uses central controller to do network-wide channel and power management to mitigate interference, while Meru [2] uses a central controller to speed up handoffs for mobile clients. Detailed information on how these systems work is difficult to come by - the marketing literature does not reveal much. However, commercial vendors are hampered by the need to maintain backwards compatibility with existing 802.11 networks. To the best of our knowledge, no commercial system includes a client component.

Research systems such as DenseAP [20] and DIRAC [33] also propose a centralized architecture. However, both systems explicitly assume that no special software can be run

Policy	10th	Median	90th
Capacity-aware	0.1%	0.5%	1.1%
Up/down link	0.4%	1.4%	1.9%
Mobility	0.2%	0.4%	1.0%

Table 5: CPU utilization at the CC for three different policies. A statistics reporting interval of 5 seconds was used.

Stats interval (s)	10th	Median	90th
1	3.5%	4.3%	9.3%
5	0.8%	3.9%	6.1%
10	0.0%	3.5%	6.2%

Table 6: CPU utilization at an AP with eight clients measured over a period of 10 minutes, for various statistics reporting intervals.

on clients, and thus are limited in what they can accomplish. For example, in Section 4.2, we have shown how Dyson improves upon the association policy implemented in DenseAP using client feedback.

Several research systems use a limited form of client cooperation. In MDG [12], clients get information from APs via special fields in the Beacon packets, and the client driver uses this information to make various decisions (e.g. associations). However, the specified interface is quite limited, and is more akin to the one proposed in the 802.11k standard [15]. Similarly [19], uses feedback from clients to enable use of partially overlapping channels. SMARTA [6] uses client-cooperation via micro-probing [5] to construct a conflict graphs [23] of the network. The Dyson architecture, on the other hand, provides a general-purpose API for managing clients and APs, and can be viewed as a generalized version of these systems.

Systems such as SoftRepeater [9] and CMAP [31] specifically focus on client cooperation to improve WLAN performance. In SoftRepeater, clients with good connections relay packets for poorly-connected clients. Similar functionality can be implemented as a policy in the Dyson framework. In CMAP, clients collaborate to build an interference map of the network, which is used to schedule transmissions. Dyson's network map is a generalized version of CMAP's interference graph.

Another interesting design point is explored in [27]. The idea is to use bare-bones APs with analog-to-digital converters such that they are oblivious to the PHY/MAC layers being used at the client. As a result, all intelligence in the network is pushed to the clients. The Dyson approach is practical, and can be deployed with off-the-shelf 802.11 hardware.

Outside of the networking space, many systems have explored the use of extensibility via add-on modules with a well-defined programmatic interface. SPIN [11] and Exokernel [17] are classic examples of opening up the operating system interface to permit greater flexibility and application-specific control. Likewise, Lance [32] provides a policy module interface to customize data collection from a wireless sensor network.

6. DISCUSSION AND FUTURE WORK

Our prototype of Dyson has shed light on several directions for future work. First, our current design assumes that all clients will be able to provide periodic measurement reports regardless of their power state. Power-constrained clients such as mobile phones routinely turn off their WiFi interfaces (power save mode), and hence may not always be able to collect or report these measurements. This raises the question of what the impact of intermittent measurements collection will have on efficacy of Dyson policies. If the density of non-power-constrained clients (e.g. laptops on people's desks) is sufficiently high, good measurements can still be collected. Alternatively, a separate monitoring system like DAIR [8] can be used. In some cases, the design of polices itself will have to change to deal with partial information. We are exploring these alternatives further.

Another issue that we have chosen to leave aside for now is support for legacy 802.11 clients. One simple approach would be to assign legacy clients to a separate set of access points, on separate channels, so that they do not interfere with the rest of the Dyson network. This is tantamount to deploying a separate WLAN infrastructure for legacy users. Another approach is to admit legacy clients into the Dyson network, although the infrastructure would be unable to gather measurements or control their behavior. Note that some aspects of Dyson's control interface (such as associations) can be used with legacy clients. For now, we have opted to focus on rethinking the architecture without the constraints imposed by legacy client support.

We have designed Dyson primarily for enterprise networks, where clients are under the control of a central IT department and do not need incentives for running the measurement software. We have also not considered the impact of malicious users reporting false measurements or not responding to commands. These concerns are addressed partially by the fact that in most enterprise networks, WLAN users are explicitly authenticated using protocols such as 802.1x. Another interesting possibility is to identify malicious users by comparing measurement reports from different clients [18].

In the current Dyson prototype, clients perform only passive measurements. This was done for the sake of simplicity. We plan to explore the possibility of asking clients to perform *active* measurements, e.g., asking a client to transmit a series of probe packets to measure loss rate more accurately. Concerns about overhead and battery drain will likely limit how often such active measurements are carried out. In the same vein, one may also ask certain clients to relay packets for other clients [9]. We have not considered such possibilities in the current prototype.

Finally, we note that while it is quite easy to write new Dyson policies, it does require some expert knowledge, especially to avoid unwanted interactions between polices that run simultaneously. We do not expect an average system administrator to have the requisite skill set. We believe that if Dyson is deployed in a widespread manner, a new class of experts in programmable network management will arise that will write and distribute pre-packaged policies.

7. CONCLUSIONS

We have presented Dyson, a new architecture for extensible wireless LANs. Dyson provides a network architecture evolves with new challenges and application demands. By "opening up" clients for measurements collection and control, Dyson breaks down the traditional barrier between the infrastructure and its clients, offering substantial benefits for network management. Dyson's programmable policies framework makes it easy to customize the network's operation for site-specific needs and new services. The framework also makes it easy to store historical information about network performance, and leverage it to fine-tune network parameters.

We have demonstrated a wide range of policies for managing associations, specialized traffic classes (such as VoIP), mitigating interference, optimizing mobile handoffs, and airtime reservations for specific users. Put together, these policies elucidate the benefits of the Dyson architecture in terms of supporting a high degree of network visibility, control, and customization. Our extensive measurements of these policies on a 23-node testbed confirms Dyson's benefits for network management.

8. REFERENCES

[1] Enterprise solutions from aruba networks, http://www.arubanetworks.com/solutions/enterprise.php.

[2] Meru networks - virtual cell, http://www.merunetworks.com/pdf/whitepapers/.

[3] A reference guide to all things voip, http://www.voip-info.org/wiki/view/qos.

[4] N. Ahmed, S. Banerjee, S. Keshav, A. Mishra, K. Papagiannaki, and V. Shrivastava. Interference Mitigation in Wireless LANs using Speculative Scheduling . In *MobiCom*, 2007.

[5] N. Ahmed, U. Ismail, S. Keshav, and D. Papagiannaki. Online Estimation of RF Interference. In *CoNEXT*, 2008.

[6] N. Ahmed and S. Keshav. SMARTA: A Self-Managing Architecture for Thin Access Points. In *CoNEXT*, 2006.

[7] A. Akella, G. Judd, S. Seshan, and P. Steenkiste. Self Management in Chaotic Wireless Deployments. In *MobiCom*, 2005.

[8] P. Bahl, R. Chandra, J. Padhye, L. Ravindranath, M. Singh, A. Wolman, and B. Zill. Enhancing the Security of Corporate Wi-Fi Networks Using DAIR. In *MobiSys*, 2006.

[9] V. Bahl, R. Chandra, P. Lee, V. Misra, J. Padhye, D. Rubenstein, and Y. Yu. Opportunistic Use of Client Repeaters to Improve Performance of WLANs. In *CoNext*, 2008.

[10] Y. Bejerano and R. S. Bhatia. MiFi: a framework for fairness and QoS assurance in current IEEE 802.11 Networks with Multiple Access Points. In *Infocom*, 2004.

[11] B. Bershad, S. Savage, P. Pardyak, E. G. Sirer, D. Becker, M. Fiuczynski, C. Chambers, and S. Eggers. Extensibility, safety and performance in the SPIN operating system. In *Proc. the 15th SOSP (SOSP-15)*, 1995.

[12] I. Broustis, K. Papagiannaki, S. V. Krishnamurthy, M. Faloutsos, and V. Mhatre. MDG: Measurement-driven Guidelines for 802.11 WLAN Design. In *MobiCom*, 2007.

[13] R. Chandra, J. Padhye, A. Wolman, and B. Zill. A Location-based Management System for Enterprise Wireless LANs. In *NSDI*, 2007.

[14] Y.-C. Cheng, M. Afanasyev, P. Verkaik, P. Benko, J. Chiang, A. C. Snoeren, G. M. Voelker, and S. Savage. Automated Cross-Layer Diagnosis of Enterprise Wireless Networks. In *SIGCOMM*, 2007.

[15] IEEE. *IEEE 802.11k-2008 — Amendment 1: Radio Resource Measurement of Wireless LANs*. June 2008.

[16] G. Judd and P. Steenkiste. Fixing 802.11 Access Point Selection. In *SIGCOMM Poster Session*, Pittsburgh, PA, July 2002.

[17] M. F. Kaashoek, D. R. Engler, G. R. Ganger, H. M. Briceño, R. Hunt, D. Mazières, T. Pinckney, R. Grimm, J. Jannotti, and K. Mackenzie. Application performance and flexibility on Exokernel systems. In *Proc. the 16th SOSP (SOSP '97)*, October 1997.

[18] R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan. Sustaining Cooperation in Multi-Hop Wireless Networks. In *Proc. Networked Systems Design and Implementation (NSDI)*, May 2005.

[19] A. Mishra, V. Shrivastava, S. Banerjee, and W. Arbaugh. Partially-overlapped Channels not considered harmful. In *ACM Sigmetrics*, 2006.

[20] R. Murty, J. Padhye, R. Chandra, A. Wolman, and B. Zill. Designing High-Performance Enterprise Wireless Networks. In *NSDI*, San Francisco, CA, April 2008.

[21] A. Nicholson and B. Noble. BreadCrumbs: Forecasting Mobile Connectivity. In *MOBICOM*, 2008.

[22] A. J. Nicholson, Y. Chawathe, M. Y. Chen, B. D. Noble, and D. Wetherall. Improved access point selection. In *MobiSys*, 2006.

[23] J. Padhye, S. Agarwal, V. Padmanabhan, L. Qiu, A. Rao, and B. Zill. Estimation of Link Interference in Static Multi-hop Wireless Networks. In *IMC*, 2005.

[24] S. Pilosof, R. Ramjee, D. Raz, Y. Shavitt, , and P. Sinha. Understanding TCP fairness over Wireless LAN. In *INFOCOM*, 2003.

[25] A. Sharma and E. M. Belding. FreeMAC: Framework for Multi-Channel MAC Development on 802.11 Hardware. In *ACM SIGCOMM PRESTO*, 2008.

[26] M. Shin, A. Mishra, and W. A. Arbaugh. Improving the Latency of 802.11 Hand-offs using Neighbor Graphs. In *Mobisys*, 2004.

[27] S. Singh. Challenges: Wide-Area wireless NETworks (WANETs). In *MOBICOM*, 2008.

[28] L. Song, U. Deshpande, U. Kozat, D. Kotz, , and R. Jain. Predictability of WLAN mobility and its effects on bandwidth provisioning. In *INFOCOM*, 2006.

[29] A. Vasan, R. Ramjee, and T. Woo. ECHOS - Enhanced Capacity 802.11 Hotspots. In *Infocom*, 2005.

[30] P. Verkaik, Y. Agarwal, R. Gupta, and A. C. Snoeren. SoftSpeak: Making VoIP Play Fair in Existing 802.11 Deployments. In *NSDI*, 2009.

[31] M. Vutukuru, K. Jamieson, and H. Balakrishnan. Harnessing Exposed Terminals in Wireless Networks. In *NSDI*, 2008.

[32] G. Werner-Allen, S. Dawson-Haggerty, and M. Welsh. Lance: Optimizing high-resolution signal collection in wireless sensor networks. In *Proc. 6th ACM Conference on Embedded Networked Sensor Systems (SenSys'08)*, November 2008.

[33] P. Zerfos, G. Zhong, J. Cheng, H. Luo, S. Lu, and J. J.-R. L. DIRAC: a software-based wireless router system. In *MOBICOM*, 2003.