6.033 DESIGN PROJECT REPORT

RHO: A Distributed AP Connection Protocol

Rajeev Parvathala, Hunter Gatewood, and Ömer Cerrahoğlu

{*rparvat, hcg, omerc*}*@mit.edu*

Recitation: Karen Sollins, 11 am

May 6, 2016

I. Introduction

We propose the design of a scalable, distributed system protocol allowing clients to dynamically and wirelessly connect to the Internet at a large institution, particularly at MIT. Although this system is currently specified to meet MIT's needs, our design scales well to larger and more demanding scenarios.

At the system level, clients desiring Internet access connect to access points (APs). These APs are in turn connected to the MIT network, through which they have and provide access to the Internet. APs are scattered across the campus to allow clients in a wide area to connect to the network. Another important part of the system is the centralized server. In MIT's use case, IS&T will provide a single server machine that is connected to the Internet and to all the APs by wired connections. This is an easy place to put centralized storage and shared computing resources. There are two different kinds of clients: large clients, who require their maximum bandwidth a majority of the time (*e.g.* Netflix streamers), and small clients, who have a lower maximum required bandwidth and use far less than this maximum on average (*e.g.* someone checking their email intermittently).

RHO's design is based on five key design goals, which we will explain: average user happiness, quality recommendations, high network utilization, scalability, and fault tolerance.

First, and most importantly, we want to minimize average user[1] unhappiness. There are different kinds of clients who may use our system, and we value each client's happiness equally. We assume users will be unhappy if they are unable to connect to the Internet for an extended period of time (more than three seconds), or if the throughput they receive through the system is too low.

We also want to provide good AP recommendations—if a given client is unable to connect to any in-range AP, the client should receive the location of a nearby, underutilized AP, through which the client will be able to connect to the Internet.

We also want to maximize utilization of network resources. The client throughput load should be dispersed across APs as evenly as possible to allow for the maximal number of clients connections.

Finally, to allow the system to grow as necessary, *RHO* should be scalable, decreasing the load on the central server, and fault tolerant, allowing system functionality to continue in the face of AP or server downtime.

[1]A client is a machine with the potential to gain Internet access through connection with an AP. A user is the owner of such a machine.

In response to these design goals, *RHO* utilizes a client-heavy workload with relatively long connection times. Although AP connection is relatively time-consuming, we guarantee high satisfaction of connected clients who remain responsive. The design consists of clients discovering all in-range APs, polling each of them for their congestion levels, and then sequentially requesting a connection with APs in order of least to most congested. Any small client who connects to an AP and remains both responsive and in range will always remain connected to the server. However, an AP may choose to stop servicing a large client in order to make room for multiple other small client; this allows an AP to prioritize happiness of multiple small clients over that of a large client. The only roles of the server are to collect data and make recommendations, so the system can scale easily without undue load on the server. We also provide safeguards which promote system resilience in the face of temporary server failures and extended AP failures.

II. Design

We now present the *RHO* design in further detail. We section the design as follows:

The first section consists of technological specifications. We present the components of our system and their capabilities.

The next section contains technical considerations. In this section we present technical details regarding our protocols.

The final three sections overview the communication between system components. We begin with a higher-level overview of the system and end with further details of the AP-client protocol and IS&T server-AP protocol, respectively.

A. Technology Specifications

There are three main components of our design: the clients, the APs and the server (run by IS&T at MIT), as seen in figure 1. All the APs in our system are connected to a single network. Each AP runs a Unix-like system on a machine with 1.2 GHz processor speed, 32MB of RAM and 64 MB of flash storage.

Clients communicate with APs through a wireless network: there are 12 channels, the last of which is a broadcast channel unused in *RHO*. Each AP is operating on one of the 11 channels (never on channel 12), so that, for each client, the APs in range are all operating on different channels. The range of an AP is 125 ft. APs and clients communicate through frames, which we detail below. Each AP can accommodate at most 128

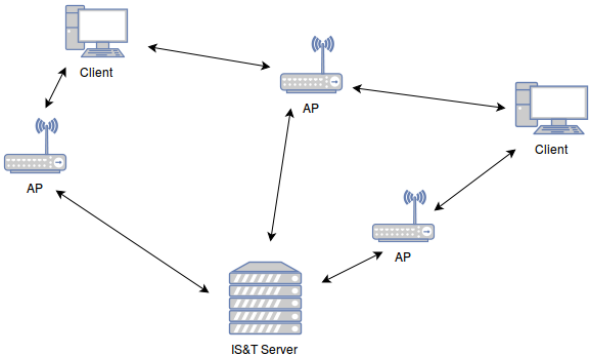

Fig. 1. A simplified view of the network layout. Clients can communicate with up to 11 nearby APs, and each AP can communicate with the central server.

clients at a any particular time. The IS&T server and the APs communicate through a wired network.

Each clients is equipped with two software modules, a **monitor** and a **controller**. The monitor determines the *required throughput* and the *achieved throughput*. The required throughput is the maximum throughput the client will ever need (which in turn determines whether the client is *small* or *large*). The controller polls the monitor every second to get information about the current performance.

B. Technical Considerations

1) Messages: Client-AP communication is performed through *frames*, which include

- a source MAC address,
- a destination MAC address,
- a metadata code (categorizing the frame's data), and
- the frame's actual data (sent to or from the client).

For all messages[2], we assume both the source and destination MAC addresses are correct and supplied (for heartbeats the destination MAC address is the broadcast address). In general, clients will only pick up on messages that are sent from APs in range and with destination address equal to either the broadcast address or the client's MAC address.

A metadata code of 0 signifies actual data, while a non-zero code represents control data (i.e. messages sent in our protocols). For each message type we are going to choose a different code[3]. As we have one byte available for metadata, we can assign each message type a unique code.

[2]We make the simplifying assumption that all control messages are passable within a single frame. This assumption is further assessed in the evaluation section.

[3]As these codes can be arbitrary, we will not detail actual values in our design.

In general, all encoded strings will be sent in *UTF-8* format, and all encoded numbers will be sent in big-endian *int_32* format.

2) Stored Information: The server already has stored the locations of each AP (latitude, longitude, room and building number). Additionally, we also store the available throughput, number of bytes sent and number of unique clients for each AP (for each second).

Each AP stores a table of all the clients currently connected to it, which it updates as needed. For each client, the AP also stores its size (*small* or *large*, whether it can kick the client (which we detail below), the client's maximum and desired throughputs.

As the server can fail, APs also store (a) the number of unique clients and number of bytes sent for every second for the last two minutes and (b) information about nearby APs (their available throughput and location, received from the server). This ensures APs can still provide AP recommendations during server downtime.

Clients store their average throughput, which they update as described below. They also store how many times they have been kicked out of a connection.

3) Computing Average Throughput: APs keep track of each client's throughput. For a reliable value, APs keep the desired (average) and maximum throughput of each client. While the maximum throughput cannot change, the average can, and clients must periodically update this value.

For reliability, clients compute an exponential weighted moving average of their throughput. The exponential average has two advantages over a pure average: first, and most importantly, more weight is given to recent throughput values, as opposed to past throughputs. Second, there is no need to store all achieved throughputs in the past, as the exponential average can be updated with the following formula: $newAverage = \beta * oldAverage + (1-\beta) * achievedThroughput$, where achieved throughput is the throughput for the last second (which can be found using the controller, as explained above). β is a sub-unitary positive constant, which can be chosen as $\beta = 0.9$, for example (we want β to be fairly close to 1, as otherwise our computed average is going to put too much weight on the present throughput).

With knowledge of each connected client's average throughput, APs can then calculate their own "available" throughput. Note that if the AP only uses the average throughput for each client (and then subtracts this from how much it can offer), at times when many clients might expect a higher throughput, the AP will not be able to satisfy every client's needs and therefore some users may be unhappy. For large clients, the APs will just use their maximum throughput as the *used throughput* for that

client. This is because large clients generally use close to their maximum throughput. For small clients, however, their maximum throughput is considerably larger than their average throughput (though at times they might use more then their average throughput), so the APs will consider these clients' used throughput to be a function of the maximum and average throughput, namely $usedThr = \alpha * maxThr + (1 - \alpha) * avgThr$, where α is a subunitary constant[4]; more on α in section III.A.1.

C. Overall Behavior

In our design, clients attempting to connect to the network send each AP in range a message requesting congestion information. In response, each AP sends a message containing its total available throughput, allowing that client to choose a minimally congested AP. After choosing such an AP, the client sends a message to that AP requesting a connection.

Along with congestion information, the AP will also send GPS coordinates and the floor number of the closest room to the client; this information is preloaded onto the AP by the server. The client will average all the GPS coordinates and floor numbers it receives from APs in range to determine an "approximate location" of the client, storing this information until a connection is established.

Generally, our protocol disperses the AP load well across any group of local APs, so we generally expect new clients to be able to connect to the least congested AP. However, if all the nearby APs are highly utilized (which indicates a very high network usage for this area), the client may have no in-range AP it can connect to. In this case, the client will request a recommendation from an AP in range, providing the averaged GPS coordinates and floor numbers it received from APs. This AP will in turn relay the provided information to the central server, which will find a nearby underutilized AP and send its location to the client through the AP. The server is best equipped to provide this recommendation because it is updated with the congestion information of each AP each second and knows the locations of all APs.

Each AP stores the MAC address and the client size (large/small) of all connected clients in memory. This is done to simplify (a) the initial communications with clients, (b) aggregating and sending the necessary meta data to the server for IS&T's records, and (c) decide which client to kick, if necessary. When a new client connects, their information is updated, and the AP regularly (every ≈ 5 seconds) sends pings to the clients to make sure they are still connected; this is to avoid refusing connections when in reality certain clients have moved out of range or have connected to other APs.

The server also regularly pings each AP to ensure that they are still connected. In the response (ack) to this ping, each AP will send the information required to be collected by the IS&T: number of new clients connected and bytes sent in the last two minutes (which was stored in the APs as explained in section II.B.2).

D. Client-AP Interaction Overview

1) Establishing Connection: In general, if a client ever sends a message to an AP and doesn't receive a response within 5 seconds, it stops trying to connect/being connected to the AP and begins a new round of finding a new AP. This is to handle the cases where APs unexpectedly reset and where the client moves out of range of the AP. Pseudocode for the connection establishment is provided in *Algorithm 1*.

```
valid_channels = []
for c = 1 ... 11 do
    Wait 30 ms for Heartbeat on Channel c
    if Heartbeat received then
        valid_channels.add(c)
        Send Request_Congestion
        Store Send_Congestion throughput/location
    end
end
Sort valid_channels from high to low throughput
for c in valid_channels do
    Send Attempt_Connection(my_throughput)
    if Accept_Connection received in 10ms then
        return
    end
end
if this is a small client then
    for c in valid_channels do
        Send Request_Kick(my_throughput)
        if Accept_Connection received in 10ms then
            return
        end
    end
end
if len(valid_channels) > 0 then
    channel = valid_channels[0]
    Compute avg_latitude, avg_longitude, avg_floor
    Send Need_Recommendation to channel
    Display Recommend_AP data received
end
```

Algorithm 1: Client Connection Establishment

A client will first cycle through all eleven possible channels and listen for an AP *Heartbeat*. If it hears

[4]Note that $usedThr$ is calculated by the client itself, and sent to the AP as normal as their "desired" throuput.

TABLE I
CLIENT TO AP MESSAGES

Request_Congestion	Ask an AP for its current available throughput.
Attempt_Connection(maximum_throughput, desired_throughput, kickable)	Request permission to connect to an AP. Data included: the client's desired throughput and maximum throughput in kB/s, and whether they are "kickable", *i.e.* if they haven't been kicked out of APs for more than 7 times in the last hour
Request_Kick(desired_throughput)	Request that an AP kick out a large client to make room for this client. Only sent by small clients.
Client_Ack(average_throughput)	Respond to an AP's *Ping_Client* and send it's average throughput.
Need_Recommendation(longitude, latitude, floor)	This is sent when there are no APs nearby that can serve the client, and the client needs a recommendation about where a freer AP is. Additional data: approximate longitude and latitude coordinates of the client, and approximate floor number.

TABLE II
AP TO CLIENT MESSAGES

Heartbeat	Broadcast from a given AP every 30 ms.
Send_Congestion(available_throughput, longitude, latitude, floor)	Reply to a client's *Request_Congestion* with the AP's current available throughput. Data included: AP's current total available throughput in kB/s, GPS coordinates of the AP, floor number of closest nearby room.
Accept_Connection	Accept a client's desire to connect to this AP; sent in response to *Attempt_Connection*.
Recommend_AP(ap_location)	Recommend that a client try to connect to a different AP if no APs in range are free. Data included: null-terminated, UTF-8 encoded string describing the location of the new AP.
Ping_Client	Ensure an existing client connection is still active.
End_Connection	Notifies a client that it will no longer be served by this AP.

TABLE III
AP TO SERVER MESSAGES

AP_Ack(new_clients, bytes_sent, available_throughput)	Tells the server that this AP is still connected and active, and supplies desired statistics. Additional data: number of new clients connected in the last second, number of bytes sent in the last two minutes and available_throughput
Request_Recommendation(longitude, latitude, floor client_MAC)	Sent when a congested AP needs to recommend an open AP to a client. Additional Data: longitude, latitude, floor, and MAC address, all supplied by client.

one, it will send a *Request_Congestion* message in order to find out the total available throughput (computed as described in section II.B.3) and the GPS and floor locations of the AP.

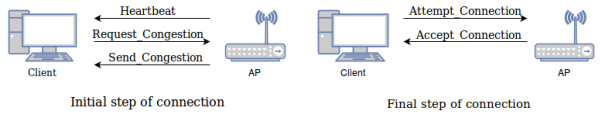

Fig. 2. Client-AP dialogue during a client's connection attempt.

After this, the client orders all APs it received a *Send_Congestion* message from, from most to least available throughput. It will then iterate through this list and send a *Attempt_Connection* message to each AP. If it receives an *Accept_Congestion* back, then the client has successfully established a connection. The AP will only accept the client if its current available throughput is at least as much as the clients needs and if it doesn't already have 128 clients connected to it. Note that while the client was gathering the available throughputs for each AP, new clients might have connected to this AP. Therefore, it is crucial that the AP is the one to ultimately accept the connection. Otherwise, it proceeds to the next AP on the list.

If the client is unable to connect to any APs in range initially, then if it is a small client, it will cycle through all the APs just as last time, but this time instead sending *Request_Kick* messages, which indicate that it wants the AP to kick out a large client and accept this client if possible.

Finally, if it is still not connected, the client will average the GPS coordinates and the floor numbers it received from the *Send_Congestion* messages. It will then request a recommendation from the initially least congested AP, supplying this location data. When it receives a recommendation back, the user will be notified that it needs to move somewhere else to connect to the

TABLE IV
SERVER TO AP MESSAGES

Ping_AP	This is to ensure that the AP is still connected, and to indicate the desire to retrieve the periodic monitoring data from the AP.
Send_Recommendation(new_location, client_MAC)	This is in response to the the *Request_Recommendation* message that an AP sends to the server. Recommends a particular AP for the given client to move to. Additional data: utf-8 string describing location of nearby AP, and client MAC address
AP_info(AP_congestions)	This is sent every minute to let each AP know relevant information (available throughput and locations) of nearby APs, so that APs can make proper recommendations in case the server goes down.

MIT network.

An AP will only accept a client connection if the provided desired throughput from the client is less than the AP's remaining throughput. If an AP receives a *Request_Kick* message from a small client and has enough throughput to accept the client, it will just accept it. Otherwise, if the AP is connected to a large client, it will kick out the large client and accept the new connection from the small client. Obviously, we don't want to kick the same client for too many times. Therefore, when a client is trying to connect to the AP (*i.e.* when sending the Attempt_Connection message) they are also going to send a bit *kickable*, which they are setting true if they have been kicked out of APs for at most 7 times in the last hour.

2) Maintaining Connection: In order to make sure that a given client is still in range and active, the AP will send each client a $Ping\_Client$ message every 5 seconds. If the client does not respond within two seconds, then the AP will assume the client is no longer connected, and will remove the client from its table of connected clients.

There is only one scenario in which an AP may forcibly disconnect from a client even if the client is in range and responding: when a large client gets kicked out in order to make room for a small client. In this scenario, the AP will send a *Disconnect* message to the large client and remove it from the AP's table of connected clients.

E. AP-Server Interaction Overview

In general, the server isn't very involved in the formation of connections in the system; it's main responsibilities are making recommendations for clients who are unable to connect, and gathering the required data from the APs.

Every second, the server will send a *Ping_AP* message to each AP; each AP will then respond to this message with an *AP_Ack* message, containing the number of new clients connected and number of bytes sent in the last two minutes. The server will store this information in a table. The normal Round Trip Time (RTT) between the server and the AP is about 5-10 ms. Therefore, if it has been more than 2 seconds since we sent a ping to an AP, then, assuming network reliability, we can presume that the AP is no longer active.

Once an AP is no longer active, we don't have to send it pings too often; the only reason to send pings to figure out when the AP goes back online. So we can send out pings once every five minutes, so that if the AP comes back online we get it back in our system within five minutes. Upon learning an AP is inactive, we can move it from our set of active APs to our set of inactive APs; we will only recommend active APs to clients when they are unable to connect to any APs in range.

When the server receives a *Request_Recommendation*(longitude,latitude,Client_MAC) message from an AP, it computes what the closest active, non-congested AP to the provided coordinates; congestion is measured as the most recent bytes per second that the server received. The server collects this data every second anyway, and it is pre-loaded with the location information of each AP. It then sends back a recommendation to the AP that requested the recommendation.

Note that the IS&T server might go down; in order to deal with this situation, the server is going to send every minute a message (AP_Info) to each AP containing information about its nearby APs. This information is the location of the APs and their available throughputs. If the AP will have to make a recommendation to a client and the IS&T server doesn't respond, they are going to use the contents of the last AP_Info message to give a proper recommendation. Note that the AP still tries to first contact the server, as it can give more accurate recommendations; however, if the server is down, the AP is going to use the (slightly stale) data it has to give a recommendation.

III. EVALUATION

We now begin a topical evaluation of the *RHO* system, covering both conceptual requirements and analyzing

specific use-cases. Within the requirements evaluation, we first examine the system from a client's perspective, then transition into network utilization, data polling, scalability, and security arguments.

A. Requirements

1) Client Performance: User Unhappiness. As a first principle, we expect a client c to be unhappy only in the case of high network utilization in the range of c.

Within the connection dialogue, if a client is unable to connect to any in-range AP, we do not allow this client to connect. This design choice (a) minimizes user unhappiness by keeping all other connected users happy (meeting their desired throughput) at the expense of making only a single user unhappy (zero throughput), and (b) keeps the network connections stable (which will come into play during the use-cases section below).

In the large-small trade-off scenario (where we kick a large client off an at-capacity AP if a small client requests to connect), the single point lost in happiness is transferred from the connecting small client to the kicked large client. However, this equal shift in unhappiness frees additional throughput on the highly-utilized AP, resulting in greater potential for future happiness.

Finally, α (see section II-B3 for a description of α) also affects user happiness. Upon installation of a system, we recommend setting $\alpha = 1$ (*i.e.* needed throughput for each client is considered to be its maximum throughput), which ensures that connected users never signal unhappiness due to insufficient throughput. Then, as IS&T data is collected, the system admins should empirically choose a value of α which sets $Ex[\text{max \# times a client is unhappy per hour}] \approx q$, where $0 \leq q \leq\approx 8$, and admins can choose a value of q they feel makes an appropriate trade-off. This allows higher utilization of the network while still keeping user unhappiness low.

Thus, we expect small clients to be unhappy an average of q times per hour. We also expect large clients to be unhappy q times per hour in up to reasonably utilized areas; in highly utilized areas, we use the guarantee that a large client won't be kicked more than 8 times an hour to guarantee a maximum unhappiness per hour of $8+q$, which is ≤ 10 for $q \leq 2$. Therefore, we expect *w.h.p.* that no client will be unhappy more than 10 times per hour[5].

[5]We argue that no system can make a stronger guarantee. As a trivial proof, assume an at-capacity network with an effectively infinite number of clients trying to connect. There will be at least one client unable to connect to an AP over the entire hour, resulting in > 10 unhappiness clicks during that hour.

AP Recommendation. While the server is up, AP recommendations to a requesting client are always correct (*i.e.* the recommended AP is near the client and has available throughput) at the time of the recommendation.

While the server is down, however, our method of caching relevant AP recommendations leads to the stale cache problem—*i.e.* the likelihood of an incorrect recommendation increases proportionally with the amount of time the server has been down. We accept the stale cache issue as a part of our system because (a) our solution to the faulty-server problem is simple, reducing system complexity, (b) the server downtime is minimal (less than 2 minutes) and presumably infrequent, meaning the likelihood of a stale cache is also minimal, and the need for such best-guess recommendations will be infrequent, and (c) an "incorrect" recommendation is possible for any recommendation scheme, as the network topography can shift while a user walks into range of the new AP.

Thus, we feel our use of caching to provide AP recommendations during server downtime is superior to a "more correct" scheme.

Minimizing Disconnections. The only time a client is forcibly disconnected is the case of the large-small trade-off. This means we have a maximum number of disconnections per hour of $8L$, where L is the number of large clients in the network, and 8 because no large client will be kicked more than 8 times an hour. This is $O(L)$, but we expect the actual number of disconnections to (a) be small in practice and (b) always benefit average client happiness, as explained above.

Acceptable Connection Time. We chose to make a design trade-off by increasing the time to connect (TTC) to allow the client to make informed connection choices. This is the core trade-off of our system—sacrificing an optimal connection time to improve many other areas of the system (high, stable network utilization, low user unhappiness, distributedness and scalability).

In the worst case, the RTT (round-trip time) of a single message between a client and an AP is 10 ms, the worst-case latency for the AP-server interaction is also 10 ms, and the time to switch between channels is 5 ms. Finally, we define R to be the time it takes the server to extract an AP recommendation (where $R = O(A \log A)$, the asymptotic lookup time for range trees, and A is the number of APs in the system).

Now, worst-case, for each of the 11 channels a client must

- listen for a heartbeat ($30 + 5$ ms/channel),
- request and receive congestion ($10+5$ ms/channel),
- request and be denied the kick of a large user ($10+5$ ms/channel), and

- request and receive an AP recommendation ($10 + 10 + R + 5$ ms).

Each $+5$ ms come from the time it takes to switch channels. Thus, for 11 channels, the worst-case $TTC = 11(35 + 15 + 15) + (10 + 10 + R + 5) = 740 + R$. For an MIT-sized network, we can reasonably assume $R \approx 300$ ms, leaving us with $TTC \approx 1$ sec. While much longer than the theoretical minimum worst-case TTC of ≈ 20 ms (98% smaller), we believe our system's TTC makes a favorable trade-off which loses little (the average human likely won't notice the difference between a 20 ms connection and a 1 sec connection) and achieves many design goals in return.

2) Network Performance: Maximizing Network Utilization. A client is always able to connect to an AP if there is an AP in range of that client with the necessary available throughput. Furthermore, a client will always connect to the least-utilized[6] AP. This ensures that the network topography has "hills" rather than peaks or plateaus.

The use of "tickets"[7] was a design choice we considered because it ensured maximal diffusion of connections throughout the network. However, we decided our solution was more optimal because (a) tickets add another layer of complexity, (b) our current system ensures "good-enough" diffusion, and (c) ticketing can cause overreactions to certain use-cases (specifically, the case of high churn), drastically increasing user unhappiness to little network benefit.

As discussed in the user happiness section, a client is only forcibly disconnected in the large-small trade-off. This design choice affects high-utilization areas by (a) increasing overall client happiness, (b) providing "at least as good" network utilization[8], and (c) minimizing client shuffling (similar argument to "minimizing disconnections" above).

Communication Overhead. There are two main forms of communication: client-AP and AP-server. As AP-server communication is 100% control traffic by system definition, we focus on the client-AP traffic.

[6]By least-utilized we mean the AP with the most remaining throughput, not necessarily the AP with the least current throughput.

[7]In the ticket system, an at-capacity AP with a connection request from client Alice can issue a "ticket" to a connected client Bob, then kick Bob and connect Alice. Bob then goes through the connection process on his own. If he can't make any connections (*i.e.* no in-range APs can accept his throughput), Bob can use his ticket as a pass to be readmitted to his original AP. The AP would then issue a ticket to a new user Carol (only if Carol had not recently been issued a ticket), if necessary, to make room for Bob.

[8]High local utilization implies that *w.h.p.* there will be small clients to take up the vacant throughput created by kicking the large client, and the large client will be free to connect to a potentially less utilized part of the network.

Arguments to each control "function" are all 32-bit integers, which easily fit inside a reasonable frame size (*e.g.* 2,300 bytes). Additionally, we assume, worst-case, that every AP has (a) the ability to operate on 150 filtered frames/second and (b) the minimum 54 Mbit/sec data rate to share between connected clients. We evaluate frame- and bitrate-based arguments.

- Frame-based. As an AP can only filter out and operate on 150 control frames per second, the frame-based argument is the more important of the two. Let's say we have C connected clients, and R unconnected clients entering the connection protocol with the AP. Pinging clients takes an average of $C/5$ frames per second (filtering one ack per connected client every five seconds). Negotiating connections takes $3R$ frames per second (filtering the congestion, connection, and kick requests).
  If the AP is at capacity, it filters $C/5 = 26$ frames per second, or 17% of the available filtering (and can ignore other control data).
  If the AP has no connections, it can handle connection attempts from $3R = 150 \implies \approx 50$ clients per second. In the general case, we expect this to be sufficient. However, in the event that too many clients attempt to dialogue with the AP, excess frames will be dropped due to an insufficient filtering. A client whose packet is dropped simply continues its connection protocol as if the AP sent a negative response (or was unreachable).
  Such a situation is acceptable because (a) dropped packets mean the AP will soon be more actively contributing to network utilization, (b) the ignored clients will likely be able to connect to another in-range AP, and (c) clients who find no in-range APs will try the connection protocol over multiple times, meaning they will still be able to connect within a few seconds.
  Thus, the limited capacity for filtering does present a marginal challenge to the client-AP interaction, but the overall system still handles the churn use-case gracefully. Additionally, as an AP gains stable connections, its control traffic percentage decreases (and dropping excess connection dialogue frames is increasingly acceptable as there become fewer spots to fill).
- Bitrate-based. To assume an extreme worst-case, let's say each client and AP interaction sends every possible message once per second. This results in each client-AP pair exchanging 11 integers[9] per

[9]Attempting connection sends 1 integer, requesting a recommendation and sending a recommendation require 3, and sending congestion requires 4.

second, or 352 bits/sec. An AP can have up to 128 connected clients at any given time, resulting in $350(128) \approx 45,000$ bits/sec. This results in $\frac{4.5\times10^4}{5.4\times10^7} \approx 0.1\%$ control traffic.

3) IS&T Data Polling: Data sent to the server by each AP includes a record of the number of bytes transferred per second (which the AP can calculate near-instantaneously) and the number of connected clients per second (which, in our system, APs calculate and store). As each of these records is a single (32-bit) integer, it takes 64 bits/sec per AP, on average, to send these records. Additionally, the AP-server RTT is at most 10 sec, meaning the worst-case latency is 1% of the transmission window (1 sec).

In the face of server failure, APs store this data locally. However, because the server is expected to be down for at most 2 minutes at a time, the maximum size of these accumulated records is still only $\approx (60)(2)(64) \approx 7.7$ kbits/sec if transferred over 1 second, a value we utilize below when examining the theoretical limits of the system's scalability.

4) Scalability: Limiting Factors. As described above and below, the client-AP protocol successfully handles a large range of client-AP dialogue scenarios. Thus, due to the modular, distributed nature of *RHO*'s design, the main bottleneck involves the AP-server interactions. Specifically, we examine this interaction in terms of transmission and storage. We use $C \coloneqq$ the number of clients in the network and $A \coloneqq$ the number of APs in the network. Additionally, we evaluate suitability for an MIT-type network, with a 25,000 clients, 4,000 APs, and a server with transmission speed of 1 Gbit/sec and storage of 10 TB.

- Transmission. First, as shown above, IS&T data transmission takes an average of 64 bits/second per AP. Additionally, let's assume worst-case that every client needs a recommendation every second, where a recommendation (consisting of three 32-bit integers) takes $\approx$ 100 bits. This leaves us with an effective worst-case bitrate of $64A + 100C$ bits/sec. For an MIT-type network, we have $64A + 100C = 64(4,000) + 100(25,000) \approx 3 \times 10^6$, which is less than 0.5% of the AP-server bandwidth.
- Storage. Let's assume IS&T dumps their collected records from the server every day (using then clearing $(64)(60)(24)A \approx 92,000A \approx 0.4$ Gbits per day for MIT's network, or less than 0.05% of total storage). This leaves essentially the remainder of the server's 10 TB of storage for whatever structure it uses to calculate AP recommendations. For the sake of evaluation, let's assume the structure is a range tree, which, if we use only longitude and latitude as measures of distance, has $O(C \log C)$ asymptotic storage. For some constant of operation γ, we have server storage $= \gamma(C \log C) \implies 10^{10} = \gamma(25,000 \log 25,000) \implies \gamma \approx 91,000$. This is a very high allowable constant of operation, implying that the range tree will easily fit in the server's storage for MIT-sized networks.

Theoretical Limit to Scalability. Assuming the same ratio of APs to clients, and with a more conservative estimate of recommendation frequency (perhaps an average of 1 recommendation every 30 seconds per client), the transmission bottleneck can support a network $(1/0.5)(30) = 60$ times the size of MIT, or $1,500,000$ clients and $240,000$ APs. The storage capacity depends on the constant of operation γ, so we won't evaluate the storage bottleneck.

5) Security Considerations: A correctly implemented *RHO* system should enjoy ample security benefits. We examine these from three perspectives: a passive listening attack, a data dump, and a loss of control.

Passive Listener. With proper, modern encryption techniques, a passive listener should not be able to gain any information from intercepted messages. Additionally, with proper authentication, especially between the APs and the server, an active listener would be unable to perform a man-in-the-middle attack.

Data Dump. The only data kept on permanent storage is the server's proximity data structure (due to its potential size). However, because this structure only stores the locations of APs (and no information about clients), an attacker with access to a data dump of any form would gain no additional client data.

Loss of Control. If an attacker gained control of the server, they would see client MAC addresses only when the client needed an AP recommendation. Meanwhile, if an attacker gained control of an AP, they would see the MAC addresses of every client who is either currently connected to the AP or attempting to connect to the AP. They could also write a script to store all MAC addresses from filtered frames, allowing excessive client profiling power.

We accept these security implications because profiling from a compromised server is difficult (only a small number of MAC addresses actually reach the server), and because any negative issues result from a full compromise of a piece of the system.

B. Use Cases

1) Single User: In the single-client case, the client takes $\approx$ 1 second to connect to any of the nearby APs. If any high-capacity APs are in range, the client will connect to a high-capacity AP.

2) Flood: In this case, we have a flood of ≈ 100 new, immobile clients entering an area and then staying relatively motionless for a longer period of time (*e.g.* students entering a large lecture hall for an hour-long class). Because each AP can process a maximum of 150 control frames per second, each client will have $\approx (150-100)/150 = 1/3$ chance of being ignored by an AP, implying a $(1/3)^{11} \ll 0.01\%$ probability of failing to communicate with any APs. Thus, we can say *w.h.p.* that every client will be able to connect to an AP, and, by the connection protocol, the resulting connection will have high diffusion.

3) Churn: In this case, we have a high turnover, low stability system where clients are constantly moving in and out of range of various nearby APs (*e.g.* students moving through a hallway during passing period, at a rate of about 10 student-turnovers per second). This use-case is the main reason why we avoided the "ticketing" concept, instead opting for the more stable system where small clients are never kicked from their AP. In this scenario, we account for both the happiness of previously-connected clients and clients in the churn.

For the previously-connected clients, our design ensures that small previously-connected clients will never be kicked from their AP. Large, happy clients, however, may be kicked if the churn causes high local utilization. Because a highly unhappy (8 clicks in the last hour) large user will never be kicked, however, the churn won't cause unacceptable user unhappiness. Additionally, assuming the kicked large client is not directly in the churn (*e.g.* some number of feet away), the large client will naturally be connected to lower-utilized AP which is out of range of the churn.

For clients in the churn, we first need to make some assumptions. Let's say clients walk around 5 feet/sec, and that they traverse on average 30% of an AP's 125-foot diameter while in range of the AP, or $\approx 125/3 \approx 40$ feet. This means a client spends an average of $40/5 = 8$ seconds in range of any particular AP. Thus, a client in the churn will be disconnected for $\approx 1/8 \approx 13\%$ of their time in the churn. Although our system's relatively large TTC leads to this high in-churn downtime, we deem it an acceptable trade-off because (a) we expect the majority of clients to be stationary, (b) we expect many mobile clients to be small clients (checking their email rather than watching a movie), for which moderate downtime is less detrimental, and (c) this design trade-off both optimizes for the common case and allows for the achievement of our main design goals.

IV. Conclusion

RHO aims to make efficient use of the underlying wireless network and promote high average client happiness. We have taken a client-based approach, where connections are formed with as much information about local network topology as possible. With this care in forming connections, however, we we can guarantee that a client will never be disconnected unless it benefits average user happiness—and, *w.h.p.* no client will be unhappy more than 10 times per hour. By caching information about free APs and allowing the central server to make dynamic recommendations based on which APs are currently available, we ensured tolerance against arbitrary hardware failures; additionally, keeping AP-server messages to a minimum allows *RHO*'s scale to fit very large networks.

One possible criticism of our design is the scenario where a large client attempts to connect in an area with many small clients connected to APs near capacity. Although the nearby APs together may have enough accumulated bandwidth to serve the client, each AP individually is unable to accept the large client. Because we never shuffle around small clients, this is a potential under-utilization of the network.

We considered a number of ideas to combat this issue. However, we believe the necessary increase in the complexity and decrease in network stability would not be worth the small benefit gained in these uncommon cases. Furthermore, if these large client were to be accommodated, many other smaller clients would have to switch APs, decreasing average user happiness (a primary design goal).

We do not foresee any large problems which must be addressed before this design can be implemented; the main hurdles are the implementation details of each part of the protocol, and ensuring that the GPS/classroom information stored at each AP remains accurate across network upgrades. Thus, we believe this system is ready to be implemented on a network of MIT's scale.

V. Acknowledgments

We would like to thank our recitation instructor Dr. Karen Sollins, our TA Jacqui De Sa, and our tutorial instructors Rebecca Thorndike-Breeze and Jessie Stickgold-Sarah for their valuable advice throughout the semester. Their comments helped us improve our design and presentation.