

# Signal-Dependent Mesh Networking

Christopher Ayoub, EID cfa348

## 1. INTRODUCTION

This project explores the idea of adapting the location of a wireless access point itself using robotics or other physical machinery in order to optimize the performance (throughput) for client devices. Emerging wireless technologies, such as 60 GHz, can provide very high throughput, but require a more direct line-of-sight to client devices than 2.4 GHz or 5 GHz Wi-Fi networks. Other papers have also found that even slight physical movements of the access point or its antenna can provide significantly higher throughput to a client device. Additionally, many of the newer consumer-grade access points today involve multiple access points working together as a mesh, and I was interested in exploring this kind of technology.

## 2. APPROACH

My approach uses an arm-like model to transport an access point around in physical space. This is in contrast to a drone-based situation, as drones are much more complicated and are much more prone to accidental damage. The remote access point then gains connection to the overall wired network through a mesh networking connection to a stationary wireless node, located in the central axis of the system.

The initial idea I had in mind for the design of this would be a robot that can propel around on a track, mounted to the ceiling. However, due to my lack of experience with robotics, as well as the problems with supplying power to such a machine, I decided on a simplified version of this which will hopefully simulate the same idea. Figure 1 shows the current state of the arm mechanism. One end of the arm contains the battery-powered access point, while the other end is a simple counterweight for stability. The center of the arm (obscured from the photo) is a motor that is fixed to a platform. The motor allows the arm to rotate very quickly in both directions, providing the remote AP with a full 360 degree range of motion. This model could easily be reproduced into something that would hang from a ceiling (which is where most APs usually are).

This circular model allows the system to respond to changes in a client device's physical location, as measured by reported current signal strength. With reported changes in signal strength, the remote AP could theoretically move just back and forth in 1D space, yet that is not as versatile as a 2D-based system. My model allows for better adaptations to changes in location. In this system, we are only making adjustments based on a single scalar measurement (signal strength). As a result, it would be nearly impossible to accurately adapt to changes in 2D space with a non-circular model, unless there were additional location sensing components (e.g. localization metrics from sensor networks).



**Figure 1:** Arm mechanism centered on a motor, fixed to a platform. A counterweight is on the left for stability, while the remote AP sits on the right. The central node sits near the middle of the arm, but it is not attached. Tape is used to secure the motor to the platform, for stability.

## 3. RELATED WORK

The inspiration for this project primarily comes from a paper that proposes different models for access points that physically move [1]. In the experiment, they use access points mounted onto line-following Roomba robots and present their findings. However, they also propose the idea of ceiling-track mobility for an access point,

which led to the development of my robotic model.

Another paper shows the gains on throughput with proper physical adjustment of antennas in both MIMO and single antenna scenarios [2]. Instead of moving just the antenna, however, in my project, I am moving the entire access point with the antenna attached in order to hopefully minimize interference.

Finally, a different paper analyzes the 60 Ghz wireless protocol and its sensitivity to environment structure [3]. In their paper, they use various ambient reflectors to best propagate the signal to client devices. Using 60 Ghz access points and clients would be a great extension to the project that I am doing, as I believe it would show a much higher change in relative performance as the arm runs its calibration cycle.

#### 4. ADJUSTMENT ALGORITHM

The adjustment algorithm is based on the intuitive nature of the circular model, and is rather simple to understand, yet is quite effective. The initial space is divided into quadrants. From the starting position, the arm will execute quarter (90 degree) turns while the signal strength (as measured at the AP) improves. Once we notice a degradation in signal strength after a turn, we know that the best position for the arm exists somewhere between the current position ( $x$ ) and the previous position ( $x - 90$ ).

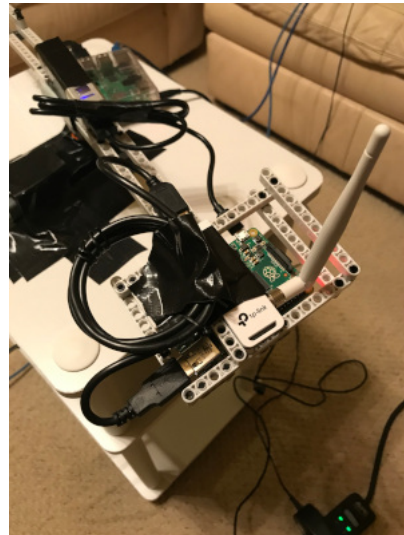
We can then divide the current quadrant up into a set of smaller divisions as desired, and find the best position among these. In this case, we choose to divide the quadrant in half, and we will choose either the middle of the quadrant ( $x - 45$ ) or the previous position ( $x - 90$ ), whichever provides a better signal. This provides an overall resolution and division of the space of eighths. Theoretically, the space can be divided even further as necessary, but because of the small physical size of the arm (1 foot, as measured from the central axis), it was not worthwhile to divide further than this.

#### 5. IMPLEMENTATION

For the access points, I am using a pair of Raspberry Pi single-board computers. These are quite ideal for this project, as I am able to easily configure and analyze the metrics of these access points, which would be harder with an off-the-shelf device. Both are running a distribution of the Linux kernel.

The APs are created using a software called hostapd [4], which uses the specified wireless adapter (which must support AP mode) to create a wireless network. I am using 802.11n with 2.4 GHz for both APs, as the hardware used does not support 5 GHz in AP mode. With this project, we will simply look for relative performance changes rather than absolute performance (which will not be exceptional, due to the limited processing power of the small computers).

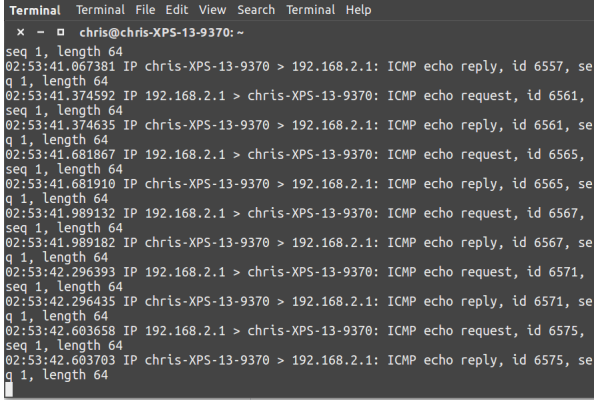
The remote node (a Raspberry Pi 2), physically mounted on the arm, has two wireless adapters. The standard wireless adapter (Edimax EW-7811Un) connects to the central node as a client, and the high-gain wireless adapter (TP-Link TL-WN722N) [5] creates a dedicated access point over a separate subnetwork using NAT. Despite the usage of NAT (hardware limitations prevent the usage of link-layer forwarding), the devices on this network can still interact with devices on the wired network, as both networks use the same subnet mask, and a different set of IPs are assigned on each network. The wireless AP created at the remote node runs on a different wireless channel, as to not interfere with the wireless AP from the central node. Outbound packets are correctly forwarded at the network layer using custom iptables rules [6]. Clients are assigned an IP address using DHCP through a software called dnsmasq. Finally, udev rules are used to ensure that the same interface name is assigned to the same wireless adapter, regardless of the port used [7].



**Figure 2: The remote node attached to the arm. The large, white antenna is part of the AP that clients connect to.**

In addition to hosting the primary access point, the remote node also exposes the current signal strength of all its current clients in dBm. This is returned from a Python servlet as a JSON key-value object, which maps the client's MAC address to its signal strength. The servlet also provides another important function. Based on the specified MAC address, the servlet will indefinitely ping (ICMP) the desired client, until a subsequent stop is requested. This ping is important so that the signal strength metric that is retrieved is current and accurate. Without any network traffic, the signal strength at the client does not change, so we ensure this by sending some traffic ourselves. An important as-

pect of this was making sure to send these ping packets at a faster rate than the polling rate for updated signal strength. Thus, a requirement in this system is that the client will send a response to such ICMP packets. Mobile devices do not readily respond to such packets, so an alternative option should likely be designed for these clients.



```

Terminal Terminal File Edit View Search Terminal Help
x - □ chris@chris-XPS-13-9370: ~
seq 1, length 64
02:53:41.067381 IP chris-XPS-13-9370 > 192.168.2.1: ICMP echo reply, id 6557, se
q 1, length 64
02:53:41.374592 IP 192.168.2.1 > chris-XPS-13-9370: ICMP echo request, id 6561,
seq 1, length 64
02:53:41.374635 IP chris-XPS-13-9370 > 192.168.2.1: ICMP echo reply, id 6561, se
q 1, length 64
02:53:41.681867 IP 192.168.2.1 > chris-XPS-13-9370: ICMP echo request, id 6565,
seq 1, length 64
02:53:41.681910 IP chris-XPS-13-9370 > 192.168.2.1: ICMP echo reply, id 6565, se
q 1, length 64
02:53:41.989132 IP 192.168.2.1 > chris-XPS-13-9370: ICMP echo request, id 6567,
seq 1, length 64
02:53:41.989182 IP chris-XPS-13-9370 > 192.168.2.1: ICMP echo reply, id 6567, se
q 1, length 64
02:53:42.296393 IP 192.168.2.1 > chris-XPS-13-9370: ICMP echo request, id 6571,
seq 1, length 64
02:53:42.296435 IP chris-XPS-13-9370 > 192.168.2.1: ICMP echo reply, id 6571, se
q 1, length 64
02:53:42.603658 IP 192.168.2.1 > chris-XPS-13-9370: ICMP echo request, id 6575,
seq 1, length 64
02:53:42.603703 IP chris-XPS-13-9370 > 192.168.2.1: ICMP echo reply, id 6575, se
q 1, length 64

```

**Figure 3: A running instance of tcpdump [8], capturing the incoming ICMP requests and corresponding responses from the client during an execution of the adjustment process.**

The arm mechanism was built using an old Lego robotics kit that I had. This worked well, as I was able to easily modify the construction of the arm to build the ideal design I desired. This arm mechanism is controlled by a motor controller (Lego NXT), which can be controlled over USB or Bluetooth. With this project, I am using USB due to its better reliability and latency. I am using the open-source firmware leJOS [9] for the motor controller, a Java-based firmware. The central node is able to issue simple commands to the motor, such as degree-based rotation in either direction, which is theoretically accurate within 2 degrees. The distance from the central axis to the end of the arm is about 1 ft. The system is moderately stable, but would benefit from a more stable construction as a solid piece, rather than as multiple pieces connected together (due to the nature of the materials).

The central node is a Raspberry Pi 3B+, configured as an Ethernet-to-802.11n bridge. This is essentially a standard access point, yet it provides the benefit of using little power and is programmable. The central node provides several functions: provide access to the wired network, send commands to the motor controller to produce movements in the arm, and frequently poll the arm node to receive updated signal strength metrics.

The central node, when requested, will instruct the arm node to begin pinging the desired wireless client. Then, it will run an implementation of the adjustment algorithm, signaling the arm’s motor to move whichever

direction and amount is required. These movements are informed by a sample of current signal strength, which is retrieved from the arm node’s servlet. Several samples are taken, and the average of these is calculated for usage. Once the adjustment process is complete, the central node finally instructs the arm node to stop pinging the client.



**Figure 4: The central node, attached via Gigabit Ethernet to the wired network. The node has an integrated 802.11ac wireless adapter, and serves a dedicated wireless network to the remote node.**

My shell scripts and code to configure the system is stored on the public GitHub repository at <https://github.com/chrisayoub/Mesh>.

## 6. EVALUATION

For the evaluation, I wanted to test the local network performance (throughput) in isolation rather than the overall Internet speed. For this, I used a tool called iperf [10] which measures throughput for a client to a configured server. The test runs with the arm node, as the arm node will always have a strong signal to the central node, given the close proximity. Thus, we are measuring the single-hop performance gains for the client based on adjustments to the arm node’s position.

It was important to conduct the experiment with the APs running on the most optimal channels possible. I used a tool called LinSSID [11] to do a rogue network scan, providing me with all of the coexisting 2.4 GHz networks. Channels 11 and 1 were used by the highest-strength, interfering channels, so I chose to use channels 6 and 1 for my APs. Note that the experiment was conducted as far away as physically possible from the existing access points, and I do not believe their interference substantially affected any of the results presented.

For my tests, I placed a client laptop at various places

Channel	Privacy	Cipher	Signal ▲
149	WPA2	AES	-59
11	WPA2	AES	-59
1	WPA2	AES	-66
149	WPA2	AES	-67
11	WPA2	AES	-68
149	WPA2	AES	-68
6	WPA2	AES	-70
149	WPA2	AES	-70
149	WPA2	AES	-71

**Figure 5: Interfering networks in the test environment. 2.4 GHz networks utilize channels 1, 6, 11.**

around the room, and placed the arm at different initial positions for each placement of the client. I ran a throughput test using iperf both before and after a single run of the adjustment algorithm.

```

~$ iperf -c 192.168.2.1
-----
Client connecting to 192.168.2.1, TCP port 5001
TCP window size: 85.0 KByte (default)
-----
[ 3] local 192.168.2.64 port 60702 connected with
[ ID] Interval      Transfer    Bandwidth
[ 3] 0.0-10.1 sec   13.4 MBytes 11.1 Mbits/sec
~$

```

**Figure 6: A sample execution of iperf to get experimental network throughput values.**

One observation I made regarding the algorithm is that it is "optimal" in all but one case. When the client resides approximately 90 degrees to the right of the arm, the arm will eventually move close to the client, but not as close as possible. This is due to the nature of the algorithm's implementation always picking "left" as the initial rotation direction. The algorithm can be likely improved through a better design to accommodate this case, or we could use some randomization for the initial direction used.

In some cases, the adjustment algorithm did not intuitively position the arm to be as perfectly close as physically possible to the client device. However, it still positioned it such that the signal strength was optimal. This was especially relevant when obstacles and obstructions were introduced, providing interference and modifications to the signal (multi-path). Thus, even if the arm does not move to the closest position to the client, it will still provide the most optimal signal that it can.

## 7. RESULTS

Values are measured in Mb/s.

	Pos. 1 8.5 ft	Pos. 2 18 ft (behind door)	Pos. 3 3 ft
Initial	41.8	45.9	47.6
Test 1	47.2	45.8	47.9
Test 2	44.1	46.1	47.7
Test 3	43.2	45.4	47.8

In every test case, the arm always moved from a location that was further away from the client to a location that was physically closer to the client, showcasing the algorithm's benefit in terms of proximity. However, as the data shows, the performance gains are only marginally beneficial. This shows that while an increase can be expected, the performance increase is not very significant due to the already strong propagation of the Wi-Fi signal. However, if we were to use a interference-sensitive signal, such as 60 GHz 802.11ad, this model would be much more beneficial in terms of throughput.

As a side note, running the iperf server on the arm node (AP transmitting to the client) performed much better than the inverse. Perhaps this is indicative of the design focus of wireless clients (priority in receiving rather than transmitting).

Note that the throughput value presented during my in-class presentation was based off a single run of the experiment, rather than multiple, due to time constraints, and was running in this inverse configuration, which led to a result that is inconsistent from the values shown here.

## 8. 60 GHZ

My interpretation of the results lead to the natural application of 60 GHz Wi-Fi to this system, as it focuses on optimizing short-range performance. With the FCC recently allowing for the usage of 60 GHz unlicensed spectrum [12], some manufacturers have released various access points and adapter cards with support for the 802.11ad standard, which utilizes 60 GHz.

802.11ad is fully supported by hostapd, the software used to create access points. In my attempts to apply 802.11ad to my project, I was loaned a 60 GHz mini PCI-e wireless card (Dell DW1601/Wilocity 6120), along with a Acer laptop with a 60 GHz 802.11ad card. My plan was to configure a set-up where one central laptop uses the DW1601 card with an extended antenna cable in which the antenna is attached to the arm of the robot. Then, the Acer laptop would act as a client, and I would re-run the experiment.

Unfortunately, despite the Acer laptop working perfectly, and despite locating an extended antenna cable (using the proprietary X.FL antenna connector) that



would make such an implementation possible, I discovered that the DW1601 card does not have proper drivers available for use, in both Windows and Linux. In Windows, the card is specifically intended strictly for use with a corresponding wireless dock, and thus you cannot connect to any 60 GHz Wi-Fi networks. In Linux, there is a Wilocity (now defunct) driver available, but it is only usable with newer versions of the Wilocity wireless cards [13]. I do not believe that this specific card actually has any capable driver for usage with Wi-Fi, although it does use 60 GHz signals. It seems that the card may need a specific firmware in order to enable Wi-Fi mode, but I have been unsuccessful in locating this or a relevant guide for the card.

Based on my searches thorough online marketplaces, it seems difficult to locate any new, consumer-grade 802.11ad wireless cards, and even Intel has announced their plans to discontinue their 802.11ad WiGig hardware in order to focus on the application of 60 GHz to wireless VR instead [14]. I am not sure if the 802.11ad standard will truly take hold for usage in Wi-Fi networks, but 60 GHz does seem to be a promising frequency for usage in specialized, high-bandwidth applications.

## 9. FURTHER WORK

One problem that I currently face is how the arm AP actually connects to the central AP. Currently, it is not a full mesh from the perspective of the link layer. The devices connected to the arm AP are on their own subnetwork through NAT, and packets are simply forwarded through customized IP rules. This is due to a hardware limitation of the Raspberry Pi and a protocol limitation of wireless MAC, preventing the system from doing a true wireless bridge across two wireless NICs. I have seen various solutions to this problem, but the most promising involves using a true link-layer mesh protocol, such as batman-adv [15]. However, with my current approach, the devices on each network are still able to fully communicate with devices on the other respective network. A link-layer protocol for forwarding frames would simply provide a performance enhancement due to less overhead.

Additionally, the ideas of the project could also be utilized with 802.11n MIMO/beamforming systems in which only the antennas move physically to optimize performance. Here, we only used 802.11n without beamforming due to limitations in the physical adapters and the Raspberry Pis. However, more advanced hardware with more functionality might come with the cost of bigger physical size and higher power consumption. Such costs must be weighed and evaluated in the context of the overall system.

As mentioned before, we are restricted in how the robotics adapt to the client's location due to relying on

just a single scalar. However, if there were additional localization metrics available, as in sensor networks, (perhaps through the usage of other signals, such as Bluetooth), there could be a more complicated system that would better adapt to the exact user location in 3D space, providing even further throughput gains.

Finally, the scope of this project only involve the usage of a single client device connected to the remote AP. In a futuristic scenario, I can speculate that APs could be extremely cheap and small such that each user could receive a dedicated, fast connection to the underlying wired network with their own wireless AP. Ignoring this, a natural extension of my project could be to apply a more complicated decision algorithm to the arm's movement in response to multiple connected devices.

## 10. CONCLUSION

My project involves physically moving an access point in response to environmental changes in order to diminish the negative effects of such changes and ensure a positive client experience. In completing my project, I have shown that it is at least physically and technically possible to implement such a system. Future work could design and build better and more robust robotic systems that would be more optimal for deployment to enterprise environments.

## 11. REFERENCES

- [1] M. Gowda, N. Roy, and R. R. Choudhury, "Infrastructure mobility: A what-if analysis," in *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, p. 19, ACM, 2014.
- [2] F. Adib, S. Kumar, O. Aryan, S. Gollakota, and D. Katabi, "Interference alignment by motion," in *Proceedings of the 19th Annual International Conference on Mobile Computing & Networking*, MobiCom '13, (New York, NY, USA), pp. 279–290, ACM, 2013.
- [3] T. Wei, A. Zhou, and X. Zhang, "Facilitating robust 60 ghz network deployment by sensing ambient reflectors.," in *NSDI*, pp. 213–226, 2017.
- [4] <https://www.raspberrypi.org/documentation/configuration/wireless/access-point.md>.
- [5] <https://github.com/lwfinger/rtl8188eu>.
- [6] <https://somesquares.org/blog/2017/10/Raspberry-Pi-router/>.
- [7] <https://raspberrypi.stackexchange.com/questions/50107/how-to-reserve-wlan0-for-embedded-wifi-on-raspberry-pi>.
- [8] <https://askubuntu.com/questions/430069/how-to-monitor-who-is-pinging-me>.
- [9] <http://www.lejos.org/nxj.php>.
- [10] <https://askubuntu.com/questions/7976/how-do-you-test-the-network-speed-between-two-boxes>.

- [11] <https://launchpad.net/ubuntu/+source/linssid>.
- [12] <https://www.fcc.gov/document/fcc-modifies-part-15-rules-unlicensed-operation-57-64-ghz-band>.
- [13] <https://wireless.wiki.kernel.org/en/users/Drivers/wil6210>.
- [14] <https://www.engadget.com/2017/09/10/intel-focuses-wigig-efforts-on-wireless-vr/>.
- [15] <https://www.open-mesh.org/projects/batman-adv/wiki/Wiki>.