Proximity Wi-Fi

Stuart Cheshire

Draft-14
23rd February 2017

At present this is a design document, rather than a protocol specification.

It is also, most categorically, just the personal opinion of the author, Stuart Cheshire.

This document does not have any official support from Apple, and should not be taken as any indication of any future Apple product plans. In fact, it would be fair to say that it currently faces substantial opposition within Apple, so its future is far from certain. That said, please treat this document as confidential anyway, and do not share it with anyone else. Despite all the caveats and disclaimers above, someone in the press might nonetheless think this boring document contains something interesting that's worth reporting upon.

This document's current purpose is to describe concepts, and the information to be conveyed in messages between peers, but not all the precise details of the formats of those messages.

If we achieve consensus on the concepts, then precise specifications of how the information is to be represented in packets will be added later.

At this stage everything in this document is open for discussion. For example, this document currently proposes measuring time in units of microseconds, but if you know a good reason why different units would be better, then speak up and join the discussion.

Simulation results are presented towards the end of this document. If you find yourself skeptical about some of the ideas expressed here, you may want to skip ahead to take a glance at the simulation results, and if you find those results intriguing you can then return to reading about the details of how those results were achieved.

Introduction

The goal of Proximity Wi-Fi is to allow Wi-Fi devices to communicate with other nearby Wi-Fi devices, within direct (one hop) radio range, independent of any external infrastructure.

The idea is that if my device has a Wi-Fi radio, and your device has a Wi-Fi radio, then it should be possible to use those radios to communicate directly, peer-to-peer, regardless of the good behavior, bad behavior, or complete absence, of external infrastructure. This usage model is analogous to a group of friends using handheld walkie talkie radios at an isolated ski resort, to communicate peer-to-peer high up on the mountain, far away from any mobile phone service.

Today, users commonly exchange documents and pictures via email, Apple Messages, WhatsApp, and other similar services that exchange data by sending them to some well-connected Internet hub and back. This works reasonably well, at least in situations where good Internet connectivity exists. Although transferring data by sending it over a wide-area connection and back again is generally less efficient than a direct peer-to-peer exchange would be, the average user may not be aware of that, or may not care. For peer-to-peer exchanges to become popular and widely used, performing these direct transfers needs to be as easy and reliable as the current techniques, with which users are already familiar and comfortable.

The focus of Proximity Wi-Fi is casual data exchange. Proximity Wi-Fi is not trying to replace current managed networks. In cases where it makes sense to set up a well-managed network with one or more Wi-Fi access points, it still makes sense to do that. The competition for Proximity Wi-Fi is not Wi-Fi access points or Gigabit Ethernet connections; the competition is the use of USB memory sticks for data transfer. If all mobile phones had USB ports for connecting USB memory sticks then perhaps we wouldn't even need Proximity Wi-Fi, but since not all mobile phones have USB ports, but do have Wi-Fi chips, we need a way for devices to exchange data using that Wi-Fi hardware, instead of a USB memory stick.

An unusual aspect of Proximity Wi-Fi, and other similar peer-to-peer Wi-Fi technologies, is that they are all technologies without any hardware of their own. When Ethernet, USB, Bluetooth and Wi-Fi were each invented, those inventions all included the necessary hardware to make them work. No one expected to be able to add Ethernet or USB to a computer without adding any hardware.

In contrast, peer-to-peer Wi-Fi technologies are second-class citizens, which need to make do with the Wi-Fi hardware that already exists. Peer-to-peer Wi-Fi technologies need to "borrow" a share of time on the existing Wi-Fi radio, without unduly disrupting the existing operation of that Wi-Fi radio — meaning that peer-to-peer Wi-Fi must coexist simultaneously with a device's traditional infrastructure association with a Wi-Fi access point.

This aspect is both a benefit and a limitation. The benefit is that it makes peer-to-peer Wi-Fi very cost-effective — it provides additional functionality with the Wi-Fi hardware that many future devices will have anyway. The limitation is that every aspect of the protocol design must take into account the fact that peer-to-peer Wi-Fi may not have exclusive use of its radio hardware.

The WDS (wireless distribution system) mechanism is a similar example of a single Wi-Fi radio serving double duty. WDS allows a single radio in an access point both to serve clients associated with that access point, and also to relay those clients' packets wirelessly to other access points, saving the cost of installing a traditional wired Ethernet backbone to interconnect access points. Apple's experience with WDS and DWDS (dynamic WDS) in the AirPort base station product line showed that it has poor performance, causing one engineer to describe it as, "Sending one radio to do the job of two." Using the Wi-Fi radio instead of a wired Ethernet backbone to interconnect access points is convenient, but it should be no surprise that this convenience has some associated cost. Proximity Wi-Fi should be viewed in the same light. Being able to do direct peer-to-peer transfers using a device's existing radio is convenient, but at the cost that Proximity Wi-Fi has to share that radio with its other existing uses.

The performance cost of time-sharing one radio between two different concurrent tasks can be avoided by adding a second radio dedicated to Proximity Wi-Fi. It's possible that future high-end devices may offer this, but even if some devices do offer dual radios, a significant benefit of Proximity Wi-Fi remains that it provides a cost-effective way for low-cost devices to achieve both infrastructure and peer-to-peer connectivity without needing two radios. For example, a low-cost home thermostat may use a single Wi-Fi radio to provide both remote access via an Internet connection using the home's access point, and local control from mobile phones and similar devices using peer-to-peer Wi-Fi. A home thermostat doesn't need multi-megabit throughput, and being able to time-share a single Wi-Fi radio to provide both infrastructure and peer-to-peer connectivity offers a compelling cost saving.

At the lowest end of the cost spectrum, it's possible that there may be Wi-Fi radios that lack the ability to change channel rapidly, or to signal power-saving modes to the AP. Such radios may:

(a) Support only Proximity Wi-Fi, supporting solely peer-to-peer Wi-Fi, and not allowing association with an AP. Such radios would function a little like a Bluetooth or IEEE 802.15.4 device, in that they wouldn't ever associate with a Wi-Fi AP, except they would operate at a higher data rate than a Bluetooth or IEEE 802.15.4 device.

(b) Support Proximity Wi-Fi only when not associated with a Wi-Fi AP. This would still be valuable for the first-run out-of-the-box setup experience, and for devices that the customer chooses not to associate with a Wi-Fi AP. Such a device would most likely include a physical reset button to return the device to factory defaults and repeat the first-run out-of-the-box setup process.

(c) Support Proximity Wi-Fi only when associated with a Wi-Fi AP which is on 2.4 GHz channel 1, 6 or 11. As explained later, a Proximity Wi-Fi device can offer services by listening on just one of these three channels, and, also as explained later, the vast majority of Wi-Fi Access Points use one of these three channels. This means that the majority of Proximity Wi-Fi devices can offer services over Proximity Wi-Fi while generally remaining on their associated Access Point channel almost all the time, changing channel only when required occasionally to send a unicast reply to a discovering client listening on a different channel.

Terminology Notes

This document uses the TU (802.11 Time Unit) when talking about time values. For most purposes, 1 TU can be considered to approximately one 1 millisecond. To be precise, 1 TU is 1.024 milliseconds. This value is chosen in the 802.11 specifications because it allows devices to use a 1 MHz clock, and then shift that value right by ten binary digits to yield a value that counts in units of 1024 microseconds.

All multi-byte quantities in Proximity Wi-Fi are represented least-significant byte first (little-endian). This is consistent with the IEEE 802.11 convention, which is the opposite of the big-endian representation used in most Internet protocols.

In this document, the term “server” does not refer to data center infrastructure. The term “server” is used in the broadest sense, to mean any software, running on any device, listening for incoming packets or connections. The “server” device could be a television, a printer, a smartphone, or a smartwatch.

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “NOT RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted according to the usual convention for Internet standards described in “Key words for use in RFCs to Indicate Requirement Levels” [RFC2119].

Prior Work

There are various Wi-Fi technologies that can facilitate some form of peer-to-peer communication.

Wi-Fi Infrastructure Modes — both a lone Basic Service Set (BSS), and a collection of Basic Service Sets operating as an Extended Service Set (ESS) — work very well and are widely used, both for access to wider networks like the Internet, and for local peer-to-peer communication such as AirPrint and AirPlay. When appropriately configured, infrastructure Wi-Fi networks facilitate peer-to-peer communication perfectly well, and this is the case in most residential wireless networks. Since 1999 Apple laptops have supported peer-to-peer communication while on an infrastructure connection. However, there are also many cases where infrastructure Wi-Fi networks do not facilitate peer-to-peer communication:

- In some cases, peers may be associated with different Wi-Fi networks offered by different access points, or associated with different virtual Wi-Fi networks offered by a single access point, e.g., an access point offering both a private network and a separate "guest" network. Often there is limited communication between these different links. Link-local discovery packets may not be forwarded between the links, and network address translation (NAT) may be performed independently for the two links, further hindering peer-to-peer communication. Rectifying this situation requires the users to be aware of the different Wi-Fi network names (SSIDs), and for one or both to users switch Wi-Fi network. In some cases, the users may not have, or be willing to share, the necessary Wi-Fi network passwords or credentials to enable them to join the same Wi-Fi network.

- In some cases, like many enterprise networks, there may be multiple access points offering the same user-visible Wi-Fi network name (SSID). In these cases, even when devices appear to be "on the same Wi-Fi network" to the users, they may in fact be associated with different access points, with limited communication between those access points.

- Finally, in many public Wi-Fi Hotspot configurations, even peers associated with the *same* access point are intentionally prevented from communicating with each other. Historically, in an era when the spread of Windows viruses was rampant, Hotspot operators felt this was a necessary precaution to guard against being accused of allowing a client's Windows PC to become infected with malware. These days portable devices run more robust software, and the threat of malware tends to come from the other side of the world, not the other side of the café, but this blocking of peer-to-peer communication is likely to remain in place.

The original Lucent ORiNOCO WaveLAN cards used in the 1999 Apple iBooks also supported an “**ad hoc demo mode**”. This mode had been created only to enable sales demos of the WaveLAN product technology, and was never intended to be used in shipping products.

An improved version of the Lucent “ad hoc demo mode” had been standardized by the IEEE in the 802.11-1999 standard as **IBSS** (Independent BSS) mode. IBSS mode allows for the creation of *named* peer-to-peer networks. In some communities, IBSS mode is still confusingly called “ad hoc” mode, referring to its Lucent precursor. Apple supports creating IBSS networks on macOS, by selecting “Create Network…” from the Wi-Fi menu. This offers the user the opportunity to “Create a computer-to-computer network” and asks them to choose a name and a radio channel for the new peer-to-peer network. Other peers must then join this named network. Apple iOS devices can join an existing IBSS network, but not create a new one of their own. As currently implemented, devices joining an IBSS network lose their infrastructure Wi-Fi connectivity. In addition, it is only possible to be a member of one IBSS network at a time. In principle, it might be possible to overcome this limitation with improved software, but even if that were the case, other problems remain. Using IBSS mode requires that users be aware of the details of both creating and joining a named network. One user is responsible for creating the network and ensuring that it has a unique name (IBSS mode is not very scalable and breaks down if too many devices in an area use the same IBSS network name, thereby resulting in the formation of an overly large and unstable IBSS cluster). The other user is responsible for then selecting the correct named network from what could be potentially a long list of available choices. As with infrastructure Wi-Fi networks, there is no way to discover what services are being offered on an IBSS network without joining it first. This burden makes IBSS mode, like the infrastructure modes, unreasonably difficult for casual peer-to-peer use.

Because of the limitations of IBSS mode, in 2008 the Wi-Fi Alliance defined **Wi-Fi Direct,** also known as **Wi-Fi P2P**. In July 2011 Apple shipped support for a stripped-down subset of Wi-Fi Direct (with a custom solution to enable Bonjour discovery), to support the AirDrop feature on Mac OS X 10.7 Lion. Apple never supported Wi-Fi Direct on iOS devices.

[We need to expand this section with more discussion of the technical shortcomings that caused Apple to abandon Wi-Fi Direct. We need more information about the problems of Wi-Fi Direct, both to present solid arguments why it is not a good technology, and to learn from its limitations and problems so that we don’t repeat the same mistakes in Proximity Wi-Fi. If you have knowledge in this area, please contribute text.]

In 2003, the IEEE Task Group “S” began work on the design of a peer-to-peer mesh networking protocol, eventually published in 2011 as **802.11s**. Like Wi-Fi Direct, this technology did not gain widespread adoption. The operation of the 802.11s protocol is similar to IBSS mode. There is a distributed beacon-generation protocol, where the devices with the fastest clock ‘wins’. It was designed with a similar goal in mind as WDS — as a replacement for the Ethernet backbone connecting Access Points. As such it designed for a relatively static environment of non-mobile devices, manually administered, and connected to AC power. It does not take mobility of devices into account, or energy conservation for battery-powered devices. An 802.11s mesh is a well-defined concept, and the membership of the mesh is unambiguous. Any device is either a member of a particular 802.11s mesh, or it is not. Any member of a particular 802.11s mesh is able to communicate with all other members of that mesh, and all members in a mesh use the same channel, so no channel switching is required. Since an 802.11s mesh cannot be infinitely large, this means that the universe of all 802.11s devices has to be divided up into countless separate small meshes, with invisible boundaries between the meshes. Two devices that are physically adjacent to each other, but not members of the same 802.11s mesh, will be unable to communicate with each other using 802.11s. This division of the world into disjoint 802.11s meshes will either have to be performed by a human administrator, or performed automatically (so that humans are unaware of where the invisible boundaries lie), either of which makes 802.11s unsuitable for the kind of casual, yet dependable, direct peer-to-peer communication we wish to achieve.

Because of [unspecified] concerns about Wi-Fi Direct, in parallel with shipping Wi-Fi Direct in Mac OS X in 2011, Apple's iOS team was simultaneously working on creating a competing peer-to-peer technology, **Apple Wireless Direct Link (AWDL)**. To encourage industry adoption Apple briefly contemplated changing the name to **Availability Window Direct Link** (retaining the same acronym), but eventually the AWDL design was taken to the Wi-Fi Alliance and published (with superficial modifications) in 2015 under the new name **Neighbor Awareness Networking (NAN)**, also known as **Wi-Fi Aware**. The primary design principles behind both AWDL and its successor NAN are the same, so for the purposes of this discussion, the terms AWDL and NAN will be used interchangeably. The primary design principle in NAN/AWDL is that if *all devices* can be synchronized to a common clock, a lot of problems become easier. This statement may be true, but meaningless if the initial precondition cannot be met. The difficulty is knowing what devices need to synchronize. The unstated assumption in NAN/AWDL that *all devices* means "all devices that you, as an individual, may want to communicate with, and no others", and that this community of mutually synchronized devices naturally forms a closed set that has no overlap with any other synchronized groups of devices. Unfortunately, there is no way for NAN/AWDL to know reliably which devices you may want to communicate with in the future. It may be tempting to think that a device only needs to synchronize with nearby devices around it, but the problem is that those devices also need to synchronize with the devices around them, and they need to synchronize with the devices around them, and so on, without limit. Synchronizing only with the closest device doesn't help. A user sitting on their couch in an apartment building may be closer to their neighbor's television mounted on the wall behind them than they are to their own television on the opposite wall. And that may be true of the neighbors on both sides too. The transitive closure of this relationship results in all devices in a building having to synchronize, or perhaps all devices in a city, which may be millions of devices. In a world of mobile devices and uncertain wireless communication, synchronizing a million devices is an intractable problem.

Hence, we need a different approach.

Because of problems with having AWDL enabled all the time, in fact today's Apple products use AWDL very sparingly, activating it only when needed. Bluetooth Low Energy (BLE) is used as the primary discovery mechanism, and BLE is used to trigger nearby devices to activate AWDL when needed for further communication and bulk data transfer.

Alternative Technologies

Any discussion of peer-to-peer wireless communication options would be incomplete if it did not mention other potentially useful wireless technologies like NFC, IEEE 802.15.4, or Bluetooth.

In particular, since AWDL is not enabled continuously, it depends on using BLE to determine when AWDL is required, to activate it only when needed. This fact naturally leads to the observation that, if the initial triggering is performed using Bluetooth, perhaps the entire discovery and subsequent data transfer could performed using entirely Bluetooth alone.

This observation is true.

However, there are two details that make a pure Bluetooth-only solution less desirable.

Today's Wi-Fi data rates are much higher than Bluetooth, making large data transfers over Wi-Fi much faster than Bluetooth.

In addition, Wi-Fi is well suited for IP-based networking, and facilitates the use of a broad range of IP-based facilities, like HTTP, many file sharing protocols, AirPrint printing, etc. Creating applications using IP-based facilities helps make them hardware agnostic, so they can run over any current or future IP-compatible communications hardware.

While, in principle, Bluetooth can also be used to carry IP packets, this has not been its focus. Bluetooth products have tended to implement Bluetooth-specific protocols like Hands-Free Profile, Phone Book Access Profile, A2DP (Advanced Audio), AVRCP (Remote Control), Object Push Profile, BPP (Printing), etc. Apple's AirPrint software is built on IPP (Internet Printing Protocol) and works over any IP-capable network, but it does not work for printing to Bluetooth printers, and Bluetooth BPP printing does not work over IP networks.

In short, adopting pure Bluetooth as the technology for peer-to-peer communications is certainly possible, and is likely what the Bluetooth community might advocate, but with current Bluetooth technology it is much slower than Wi-Fi, and much less flexible than IP, which future-proofs applications and allows them to run over many current and yet-to-be-invented communications technologies, both wired and wireless.

IEEE 802.15.4 supports IP and has range and energy consumption similar to Bluetooth, but has a throughput of just 0.25 Mb/s, which makes it much slower than Wi-Fi.

Coordination Without Synchronization

The problem with “synchronizing” is that “synchronizing” is a transitive operation. If A is synchronized with B, and B is synchronized with C, then A is synchronized with C.

If A and all of its neighbors are synchronized, and B and all of its neighbors are synchronized, and B happens to be a neighbor of A, then that means that all of A’s neighbors and all of B’s neighbors have to be synchronized, even devices that may never communicate, and never benefit from that synchronization. The recursive application of this property is the transitive closure, and results in the whole community of connected devices trying to synchronize, perhaps over distances of hundreds or even thousands of wireless hops.

To make an analogy, suppose over breakfast at home in the morning I synchronize my watch with the rest of my family, and you synchronize your watch with the rest of your family. Then we both arrive at work, and agree to synchronize our watches with each other. One of us has to change. Suppose I adjust my watch to match yours. Now I’m no longer synchronized with my family. That evening, at home after work, I have to have to tell my family to resynchronize with my new time. But during the day my daughter synchronized her watch to her school friends, and expects me to change mine to match hers. So I adjust my watch to match hers. And then I come to work the next day and tell you to adjust your watch to match mine. But you tell me you’ve already adjusted your watch to match your wife and her workmates. So I adjust my watch again, and go home and tell my family again to adjust their watches, and tell my daughter to tell all her schoolfriends that they need to update theirs too, and their families, and so on. It is not clear whether this process would ever complete before the children finish school and go off to college and the whole process starts over again.

If synchronizing all devices cannot be achieved, then we need to consider what we *can* achieve, to facilitate efficient peer-to-peer communication. This document describes a proposed peer-to-peer communication scheme that doesn’t have “synchronize all devices” as a prerequisite first step. Instead of “synchronization”, this communication scheme relies on what this document calls “pairwise coordination”.

To make an analogy for the “pairwise coordination” concept, suppose Alice observes that Bob’s watch is five minutes ahead of hers. And Alice observes that Carol’s watch is seven minutes ahead. Alice now asks Bob to meet her at the café at 11:35. And she asks Carol to meet her there at 11:37. Alice herself arrives at the café when her watch shows 11:30. All three arrive at exactly the same time, yet none of them had to adjust their watches, or their families’ watches, or *their* workmates, and so on.

By this technique, Alice, Bob and Carol are able to coordinate their activity *without* having to adjust their watches, or anyone else’s. They are conscious of, and keep track of, pairwise differences between clocks, but no clock ever has to have its value changed, and there is no single clock that is designated the “true” clock against which all other clocks are compared.

This document describes a proposed peer-to-peer communication scheme that is built on "pairwise coordination" rather than "synchronization". Each device keeps track of the respective time offsets between its own local clock and the clock of each peer within range around it, but no device ever has to change its own clock to match the clock of a peer, and no device ever expects a peer to do the same for it.

This proposed solution is tentatively called "Proximity Wi-Fi".

Scenarios

Proximity Wi-Fi is designed to:

1. Facilitate discovery of services (and, as a side-effect, discovery of the devices providing those services) available within direct radio communication range, and
2. Facilitate pairwise unicast communication between clients and servers within direct radio communication range.

Proximity Wi-Fi is designed to support casual networking. Proximity Wi-Fi is not designed to compete with, or replace, other technologies that perform their intended task better than Proximity Wi-Fi would. In situations where it makes sense to set up a traditional Wi-Fi Access Point, that should be done.

When two stations are associated with the same AP, and are able to communicate via that AP in the conventional way (sending traffic via the AP, which takes two hops), but wish to communicate more efficiently, that should be done using Tunneled Direct Link Setup (TDLS, IEEE 802.11z), not Proximity Wi-Fi or any other similar peer-to-peer Wi-Fi technology. TDLS is designed for the particular special case where two stations are on the same channel, associated with the same AP. A general-purpose peer-to-peer Wi-Fi solution cannot depend on these assumptions, and consequently, in the cases where TDLS is applicable, TDLS will always be able to perform as well or better than a more general-purpose solution. In many common residential scenarios where there are two stations near to each other (e.g., iPhone and Apple TV in the living room) and a single AP that is far away (e.g., in the garage where the Internet connection comes in), the direct one-hop station-to-station transmission provides a significant benefit compared to the conventional two-hop station-to-AP-followed-by-AP-to-station transmission. The direct TDLS communication offers:

1. Higher throughput (because the distance is shorter,
   so it can generally use a faster Wi-Fi modulation rate)
2. Lower delay (because it's one hop instead of two,
   plus the data rate is higher, so the time to send the data is reduced)
3. More efficient use of the shared spectrum
   (the data is sent over the air only once instead of twice, and at a higher rate,
   so the total air time expended to send it is dramatically less).

The focus of Proximity Wi-Fi is quick, easy, and reliable peer-to-peer connectivity. In cases where the highest possible throughput is desired, or the absolute lowest possible latency, it makes more sense to use a traditional Wi-Fi access point — or even physical connectivity like an Ethernet, Thunderbolt, or USB3 cable. The area where Proximity Wi-Fi excels is for casual networking — the cases where the user just wants to do a quick data exchange, and it wouldn't be worth the effort of setting up an AP or connecting a cable. To succeed at this task, Proximity Wi-Fi needs to be quick and easy to set up, and, above all, very dependable. While we do want to achieve the highest throughput and lowest latency we can, those goals are secondary to reliability. Initial simulation results show a simple non-optimized version of Proximity Wi-Fi on 802.11n hardware achieves about 70 Mb/s throughput, which is fast enough to transfer a 3 MB JPEG image in under half a second. Certainly transferring at 1 Gb/s would be nice if it were possible, but not if that would take an extra five seconds of setup time or be less reliable in some other way. Making the user wait five seconds wondering whether something is going to work is a sure way to have them reaching for that USB stick, or launching WhatsApp, or using some other dependable way of transferring data. To be a success, users need to come to expect Proximity Wi-Fi to be (a) completely dependable at least 99.99% of the time, and (b) fast enough to both complete short data transfers in a few seconds, and stream high quality video (e.g., 4k video at 20-30 Mb/s).

Another example use of Proximity Wi-Fi is interacting with a home thermostat. For seeing the current temperature, and issuing commands to change the set temperature, even the slowest Wi-Fi OFDM rate, 6 Mb/s, is vastly more than is needed. What a home thermostat does require, though, is absolutely reliable communication every time.

In traditional Ethernet networks, multicast is used for two very different purposes. One is for discovery and rendezvous; the other is efficient bulk delivery to multiple simultaneous receivers. Proximity Wi-Fi explicitly supports the discovery-type operations traditionally implemented using multicast (including things like Bonjour Service Discovery and IPv6 neighbor discovery). Supporting one-to-many bulk-delivery multicast is explicitly a non-goal. Later, if a compelling scenario is discovered, bulk-delivery multicast could be supported by having devices send multicast packets multiple times on multiple channels, selected such that all intended recipients have an opportunity to receive each packet at least once, but such scenarios might be better served using existing techniques like using traditional infrastructure access points.

Proximity Wi-Fi is designed to provide connectivity that is effectively "always on", where clients can discover and communicate with nearby servers, independent of traditional Wi-Fi access point association. Specifically, a client and server that are within radio range should be able to communicate:

1. When either or both are not associated with any access point.
2. When client and server are associated with different, unrelated, access points.
3. When client and server are associated with an access point that deliberately blocks peer-to-peer communication between associated stations.

In other words, Proximity Wi-Fi devices that are within range should be able to communicate without dependence on any external infrastructure, or even in the face of external infrastructure that is actively uncooperative with regard to facilitating peer-to-peer communication.

Use cases include short transfers (photo transfer between phones, walk-up printing, etc.) and long transfers (e.g., all-day binge watching Netflix, streamed from a portable device to a TV with Proximity Wi-Fi, or to a HDMI device plugged into a TV).

Another important use case is the out-of-the-box setup experience for home devices, like printers, thermostats, pool controllers, garden sprinkler controllers, security cameras, televisions, amplifiers, Blu Ray players, etc. Today, getting such devices onto your home Wi-Fi network is achieved by entering the Wi-Fi network password one character at a time using a remote control, or using Reverse Advertising [US Patent US7532862], where the new device offers its own SSID for a client device to join, via which the client device configures the new device to join the actual home Wi-Fi network. This process can be an onerous experience, and is an impediment to widespread success of such devices.

Proximity Wi-Fi should not prevent other uses of the Wi-Fi radio, such as traditional Wi-Fi access point association. Future devices may include multiple Wi-Fi radios, but devices with only a single Wi-Fi radio need to be able to time-share that radio between Proximity Wi-Fi and other radio uses.

For energy efficiency reasons, battery-powered devices may not want to keep the Wi-Fi radio powered 100% of the time. For the purposes of Proximity Wi-Fi, these power-saving idle times can be considered as another “use” of the radio.

For these reasons, Proximity Wi-Fi may operate at a 100% duty cycle (e.g., on a dedicated walk-up printer, connected to AC power, where a constant 500 mW to power the radio is acceptable) or at a lower duty cycle, where some of the radio time is devoted to other uses or power-saving idle periods.

The lower the Proximity Wi-Fi duty cycle, the more difficult it will be for clients to discover the device and the services it offers. At extremely low Proximity Wi-Fi duty cycles (including 0%) other technologies may be used to “wake up” Proximity Wi-Fi, such as Bluetooth, or even a simple manual button to activate Proximity Wi-Fi when needed.

Because Wi-Fi spectrum is a finite shared resource, Proximity Wi-Fi devices should be mindful about their use of that spectrum. Considerations of efficiency should include a balance of both energy usage by the device itself, which is a private matter for the device, and usage of the radio spectrum, which is a public shared resource. For this reason, Proximity Wi-Fi is designed to consume no radio spectrum at all when it is not being actively used.

Proximity Wi-Fi Design Principles

This section lists some underlying principles behind this preliminary design of Proximity Wi-Fi.

[Different people may approach this topic with different backgrounds and assumptions, so this section provides a summary of the guiding principles behind Proximity Wi-Fi, so that they can be critiqued or debated as necessary. As this document evolves into a formal specification, this section will be shortened or removed, or possibly retained in some form as an appendix.]

Design for Debuggability: When creating hardware, Design for Testability is an important industry principle. When creating protocols, Design for Debuggability is similarly important. During the development stages, a design that is amenable to easy debugging is beneficial for making rapid progress, to fix the inevitable bugs that always occur early in an implementation. Moreover, particularly for networking technologies, designing for debuggability can remain crucial in production too, in a world where the entire environment cannot be fully controlled. Even if the software on two communicating peers is fully debugged and absolutely flawless, those peers may find themselves in an environment where devices around them do no have such perfect software. In such cases, when things fail, it is advantageous to be able to debug the communication failure using only logs and packet traces from the two peers in question that are having trouble communicating. Ideally, communication between two peers should depend only on the correct operation of software on those two peers, not on the correct operation of software on multiple unknown other devices. For example, the vast majority of DHCP problems can be debugged by studying no more than packet traces at the DHCP client and the DHCP server. The same is true for NFS, FTP, HTTP, IMAP, SMTP, etc. When troubleshooting these familiar protocols, in the rare cases that some unknown other device is responsible for the failure, debugging is much more difficult and time consuming. A design that minimizes dependency on unknown other devices helps improve reliability.

Minimize Interdependency: In some ways this principle overlaps with the *Design for Debuggability* principle above, but it is important enough that it warrants further mention. The combinatorial explosion of possibilities that results from a system that depends on the correct operation of an unknown number of devices, from multiple vendors, with software of unknown quality, is intractable. The only way to design a reliable system is to aggressively minimize interdependency with other devices. The minimum number of devices involved in a pairwise communication is two, and consequently a design for a reliable system should strive to require only the correct operation of two devices: the two that are communicating.

Minimize Modes: When a problem is encountered, it can be tempting to address it by adding a new mode designed to handle that specific situation. The trouble with this approach to protocol design is that as every new mode is added the design becomes more complicated and more difficult to understand and debug. One mode may be power-efficient but result in slow discovery. Another mode may provide quicker discovery but consume more power. By definition, if a protocol has multiple modes, that must mean that different modes are suitable in different situations, which means that now it's important to be in the right mode. So, in addition to having to develop and debug software to implement every different mode, we now have to develop and debug the software responsible for choosing what mode to be in. The more modes a system has, the greater the opportunities for the system to be in the wrong mode at any given time. When reasoning about a protocol, having multiple modes can encourage wishful thinking, with designers unconsciously assuming in any given scenario that all devices are magically in the best mode for that scenario. If we could achieve the opposite — a system that has only one mode — then we would have eliminated the possibility of being in the wrong mode. In reality this extreme may not be achievable, but to the extent we can reduce the number of modes, we can reduce the possibilities for being in the wrong mode. Naturally, a system will need to adapt its behavior in different circumstances, but the design principle being advocated here is that these behavior variations should form a smooth continuum of gradual changes, rather than a set of discrete modes, with abrupt switches between modes that exhibit radically different behaviors. The system should adjust its behavior in a smooth continuous way, not in abrupt discontinuous jumps.

Bonjour Foundation: Proximity Wi-Fi deliberately does not strive to be agnostic about service discovery; picking a single service discovery mechanism is necessary to ensure that all devices can discover each other successfully. Proximity Wi-Fi is explicitly defined to support the Bonjour Service Discovery model alone; this is the dominant service discovery model in the industry today, available on Apple products, Android, Microsoft Windows, Linux, and various embedded systems.

Bonjour Record Discovery: Proximity Wi-Fi devices implement service discovery (and, incidentally, device discovery) using Bonjour service discovery records. Bonjour service discovery is typically performed using three main record types — PTR, SRV and TXT — but defining the Proximity Wi-Fi in terms of primitive records instead of in terms of explicit service discovery is more flexible and allows for more general applicability. For example, using this scheme, discovering a device's "A" and "AAAA" address records becomes just another use of the general record-discovery mechanism, rather than a separate special case that has to be handled separately. Other record types can be advertised and discovered as necessary, instead of requiring further special-case handling.

Limit Idle Traffic: If Proximity Wi-Fi is successful, we need to anticipate that it will become widely used. It is important that the protocol is designed such that widespread deployment will not lead to it failing to work. Consuming spectrum to perform concrete work on behalf of a user is appropriate — the amount of spectrum used is related (sometimes loosely) to the amount of useful work being done. In contrast, constantly using spectrum for background maintenance activities, when no user-visible work is being done, is less defensible. If a printer, sitting idle waiting for the user to print something, is constantly generating background maintenance traffic twenty-four hours a day, then the total amount of traffic generated for background maintenance activities could easily dwarf the amount of traffic used for actual printing. Traffic used for actual printing scales in proportion to the useful work being done; background maintenance traffic scales in proportion to the number of devices, even when no useful work is being done. We want to avoid the situation where Proximity Wi-Fi is so widely adopted that all available spectrum becomes used for background maintenance activities, and no spectrum is left to do actual work. These leads to the two principles below, discovery is an active process (it consumes shared spectrum), but advertising is passive (it involves only listening).

Active Discovery: In Proximity Wi-Fi, discovering records is an active task, involving the transmission of query messages. This consumes radio spectrum.

Passive Advertising: In Proximity Wi-Fi, publishing records is a passive task, involving the listening for query messages and responding. In the absence of query messages this consumes power but no radio spectrum. This is important for applications like a walk-up printer, which is powered on and available 24 hours a day. It is important to note that in the absence of clients looking for a service, advertising is effectively a non-operation as far as observers on the network are concerned. It is merely a potential service, waiting to come into existence when a client wants to use it.

This design principle assumes that discovering records is a relatively rare task, initiated primarily by conscious user action, whereas publishing records is comparatively a much more common state, embodied by all devices that are offering services available to potential clients.

No Clear Network Boundaries: In Proximity Wi-Fi there are no announcement packets when a service comes online, and no goodbye packets when a service goes offline. In traditional Bonjour using Multicast DNS, a device joining a network (connecting a cable to an Ethernet switch, or associating with a Wi-Fi access point) is a discrete event, visible to software, and warrants announcement packets declaring the services that have now joined that network. In Proximity Wi-Fi there are no discrete “networks” for a device to “join”, just the ever-changing cloud of peers that are within radio range. Since the device offering the services, and the peers that may be seeking one of those services, are all mobile, there are no discrete “joining network” and “leaving network” events.

No Transitive Closure: In Proximity Wi-Fi the usual transitive closure property (if A can talk to B, and B can talk to C, then A can talk to C) of traditional multi-access links like Ethernet does not apply.

A device “joins” a service by discovering it and sending packets to it. A device “leaves” a service by finishing using it, or by failing to get packets through to it in a reasonable time, and aborting the connection.

Proximity Wi-Fi Operation

Like the Internet Protocol itself, Proximity Wi-Fi is *connectionless*. Devices (actually, various software entities running on those devices) communicate by sending packets, and by listening for packets sent by others. Any given packet may or may not be received by other devices, depending on whether they are in range, and on other factors like interference. Hence the concept of being "connected" or "not connected" is nebulous at best, and any design that tries to assume that this nebulous concept can somehow be made concrete is likely to result in problems due to that flawed assumption. A reliable design has to accept and embrace the underlying uncertainty associated with wireless communication between mobile devices.

Proximity Wi-Fi uses three *social channels*, Wi-Fi 2.4 GHz channels 1, 6 and 11.

These three channels are available for use worldwide, which simplifies design. (Using different channels in different judicial localities would require devices to include GPS or some other way of knowing where they are in the world, along with a database of the various radio regulations that apply in different locations. We would also need to consider what happens at the boundaries between different judicial localities, and what happens when regulations change over time. This would excessively complicate the design.)

Wi-Fi 2.4 GHz channels overlap with neighboring channels such that (in the USA) 1, 6 and 11 are only three fully non-overlapping Wi-Fi 2.4 GHz channels. Because of this, general best practices for administering access points recommend avoiding using 2.4 GHz channels other than the three non-overlapping channels 1, 6 and 11. Consequently, the three Proximity Wi-Fi *social channels* are the three 2.4 GHz Wi-Fi channels that the vast majority of 2.4 GHz Wi-Fi stations are on anyway, by virtue of being associated with an access point on one of these channels.

Devices advertising Proximity Wi-Fi records are required to be listening on at least one of the *social channels* for from time to time. Two listening strategies are allowed:

(a) Blocks of at least 60 TUs of continuous listening on any social channel, with the intervals between listening start times randomly spaced from 250 TUs to 750 TUs, yielding an average of 60 TUs of listening on a social channel every 500 TUs. Also acceptable are longer blocks of continuous listening and/or shorter intervals between listening blocks. This yields a radio duty cycle of 12% for Proximity Wi-Fi.

(b) Blocks of less than 60 TUs of continuous listening, spaced such that at least 250 TUs out of any 500 TUs period (50% of the time) is spent listening on a social channel.

Note that a device advertising Proximity Wi-Fi records, associated with an AP that is following the general best practice of using Wi-Fi 2.4 GHz channel 1, 6 or 11, can trivially meet the listening requirement simply by remaining on its existing infrastructure channel for at least the required minimum amounts of time. Such a device can meet its requirement for 60 TUs of continuous listening on a social channel without having to change channel, and without the degradation in throughput, latency, or jitter associated with spending time off-channel.

The social channel is only required for the initial rendezvous between devices, and consequently, the amount of traffic on the social channel is light. Large bulk transfers are generally conducted on other channels.

When a device wishes to discover peers (technically, to discover Bonjour records being advertised by potentially unknown nearby peers) it transmits discovery messages requesting the Bonjour record(s) it seeks.

Discovery messages are sent on all three social channels, at 6 Mb/s, using 802.11g OFDM (Orthogonal Frequency-Division Multiplexing). Each group of three messages is sent as close as possible in time, subject to how quickly the radio can change channel. These bursts of three discovery messages are sent every 50 TUs for as long as the client is seeking to discover records held by peers. With this transmission rate, within 750 TUs every peer in range should have received a discovery message at least once. After a few seconds of continuous discovery the rate of sending discovery messages decays to a lower rate, to avoid excessive load on the social channels.

OFDM is selected instead of the older CCK (Complementary Code Keying) modulation scheme used by 802.11b because Apple, Samsung, and other players in the industry are looking to phase out support for CCK. When designing a protocol for the next decade and beyond (which is a reasonable time scale to think about — Bonjour printers from ten years ago are still in use and working correctly today) it is important to design for where the industry will be in five years, not where it was five years ago. Currently, all 802.11g/n/ac devices are required to implement CCK in order to listen for transmissions from older 802.11b devices (even devices not associated with the same access point) and adjust their behavior when one of these older devices is detected. This adjustment results in a throughput drop of 10-25%. Consequently, a single CCK device (e.g. a home thermostat) can cause performance degradation for all other devices in a 100 m radius around it. Old CCK devices will continue to exist for many years to come, but it is in the industry's interest to avoid encouraging the creation of new CCK devices that will perpetuate this problem even longer.

Proximity Wi-Fi devices are not required to support all the 802.11g rates. A simple Proximity Wi-Fi device like at home thermostat may support only the 802.11g OFDM 6 Mb/s rate.

Discussion point: Evidence seems to be that even the lowest-cost Wi-Fi chips today support the higher data rates. If this is true, and the higher data rates achieve sufficient range, we could consider sending discovery messages at a higher data rate, which would result in more economical spectrum usage. If we decide to require all Proximity Wi-Fi devices to support a minimum rate higher than 6 Mb/s, then clients performing discovery would be free to choose the appropriate data rate to use. Client software for performing a person-to-person exchange of photographs might choose to use a higher data rate to intentionally limit the range of discovery to people nearby. Client software for interacting with a home thermostat might choose to use the lowest data rate for maximum range so as to be able to interact with a thermostat on the far side of the house.

While sending discovery messages, a Proximity Wi-Fi device that is also advertising Proximity Wi-Fi records of its own adopts listening strategy (b), since it obviously cannot meet the requirement of listening continuously for 60 TUs on one channel while at the same time meeting the other requirement of sending every 50 TUs on three different channels.

Typically a Proximity Wi-Fi device sending discovery messages will spend at most 5 TUs to change channel and send each of the three messages, which accounts for at most 15 TUs out of every 50 TUs (30%). For the remaining 70% of the time, the device can spend up to 20% to maintain its infrastructure association, and the remainder listening on one of the social channels for incoming discovery messages. Note that, as above, a device associated with an AP that happens to be using Wi-Fi 2.4 GHz channel 1, 6 or 11 can perform the task of maintaining its infrastructure association and the task of listening for incoming discovery messages on the same channel.

Discovery messages and certain other Proximity Wi-Fi messages (as indicated below) include a "Receiver Map" data structure, which indicates at what times, and on what channels, the sender of this message plans to be listening in the future. These channels may be 2.4 GHz channels, 5 GHz channels, or other supported channels such as 900 MHz or 60 GHz. This Receiver Map tells peers at what time and on what channel the requester will be listening for responses. A requesting device that is not advertising any Proximity Wi-Fi records of its own is still required to listen on one of the 2.4 GHz channels for at least one 25 TU block of time every 100 TUs in order to be available to receive responses to its queries, and indicates its choice of channel and time in its outgoing Receiver Map to inform peers of its choice. This MUST be one of the 2.4 GHz channels, since some Proximity Wi-Fi devices may only support 2.4 GHz operation.

Replies are not required to be on the same channel as the discovery message that elicited them. This is because a device may have tuned only briefly to one of the social channels to send a discovery message, but is spending most of its time tuned to its infrastructure channel, which is a better channel for it to receive replies.

Later in the communication lifecycle the Receiver Map also allows for moving some communication, as appropriate, to other channels, to keep the traffic on the social channels low.

Proximity Wi-Fi is designed around two main goals: first facilitating fast reliable discovery, and subsequently facilitating fast reliable unidirectional unicast packet delivery. Higher-layer protocols like TCP make use of that unidirectional unicast packet delivery capability, separately in each direction, to provide reliable bidirectional connections. Initial simulation results show a simple non-optimized version of Proximity Wi-Fi on 802.11n hardware achieves about 70 Mb/s throughput, consisting of a reliable stream of unidirectional unicast data packets going in one direction, and, only loosely coupled, a reliable stream of unidirectional unicast acknowledgement packets going in the opposite direction.

Because of the nature of how Wi-Fi is designed, at any given moment a Wi-Fi device's transmit and receive circuitry must be tuned the same channel. This is because, before a device transmits on a channel, it must first listen on that channel to avoid interfering with an existing transmission on that channel.

This requirement results in the following implications:

(1) during a time interval where a device's "Receiver Map" data structure has committed to being on a certain channel, the device may transmit and/or receive on that channel.

(2) during a time interval that remains uncommitted in the "Receiver Map" data structure, a device is free to tune to any channel to transmit, but it should not expect to receive any unicast data from peers during that time interval, because peers have no way of knowing what channel it will be on during that time interval.

In order to perform a unicast transmission to a known peer, a sender must wait until such a time as the peer's Receiver Map indicates it is committed to be on a certain channel, and either (a) the sender also happens to be committed to be on the same channel at that time or (b) the sender's Receiver Map shows it to be uncommitted at that time, so it is free to tune to the receiver's channel.

If Receiver Maps include too little committed time, there will be limited opportunities to communicate. If Receiver Maps include too much committed time and leave too little time uncommitted, senders will not be free to change channel at will. This can create the situation where a peer is listening and ready to receive, but the sender has already committed to be on a different channel at that time.

Empirical testing shows good results when about $^1/_3$ of the time is committed in advance, and $^2/_3$ of the time is left uncommitted so that senders are free to change channel as necessary.

No clock synchronization between peers is implied or required to interpret the Receiver Map data structure. The Receiver Map includes the sender's own Proximity Wi-Fi clock value at the moment of transmission, and all other time values in the Receiver Map are interpreted relative to that sender clock value.

Each Proximity Wi-Fi device is responsible for making its own autonomous decisions about scheduling its radio usage, depending on its activity and other observed radio traffic. Various factors contribute to this decision process. A device offering services, that has AC power, and is otherwise idle, may listen for most or all of the time on the social channels, so that it is easily discoverable. A device that wishes to save power, or is currently busy transferring data with an infrastructure access point, may listen on the social channel less of the time.

A device that is offering no services and currently not issuing any discovery queries of its own need not listen on any 2.4 GHz channels at all. A device that is actively transferring a large amount of data with another Proximity Wi-Fi peer may choose to listen on a different channel for some portion of the time (and indicate that in its Receiver Map) in order to move bulk traffic off the social channel. Peers indicate which radio bands and rates they support, so a device can take this information into account when deciding which other channel(s) it will listen on. Naturally a device should endeavor to listen on channels and radio bands supported by the peers with which it intends to communicate.

In some cases the querying device may have some idea which device it expects to respond, and if it has a Receiver Map for that device it can use that to guide its choice of when and on what channel to send its queries. However, in the absence of the expected reply, it should also send query messages on the social channels, as described above.

Query messages include Known Answer lists to suppress unnecessary responses for services that the querier already knows about. Note that records in Known Answer lists are used *only* for this purpose. Other devices hearing these query messages SHOULD NOT add any Known Answer records to their own caches. Known Answer records are not authoritative. They are simply one device expressing its own personal idea of current network reality from its own vantage point in the network, which may not be correct for other devices. Just because one device is in range of a given peer and can hear from it does not necessarily imply that another device is in range of that same peer and will be able to hear from it.

[We may decide to use a different mechanism to suppress redundant replies. Instead of using Known Answer lists, a device could use the 802.11 acknowledgement of its reply as evidence that its reply has been received, instead of requiring the discovering device to include replies in a Known Answer list.]

Devices querying for Proximity Wi-Fi records as a result of direct user action should send query messages more rapidly than sending queries in the background not directly caused by explicit user action.

The query rate should decay over time for a long-lasting query. Other information, such as device movement, should be taken into account in deciding when to raise or lower the query message transmission rate.

The Receiver Map also includes data indicating on which Wi-Fi bands, channels and rates the device is capable of sending. This data helps guide other devices in their choice of Receiver Map. For example, a peer that is in active communication with some device may decide to listen on some other band or channel for some portion of the time (and announce this in its Receiver Map) and its decision about which bands and channels to listen on is guided by knowledge of which bands and channels the sending device supports.

Discovery queries are not acknowledged by receivers, except inasmuch as discovery messages may elicit discovery responses from one or more peers.

When a device receives a query for a record it is publishing, it sends a unicast response to the querier, at a time and on a channel indicated in the Receiver Map sent by the querier. The usual Bonjour “additional record” mechanism is used to include additional records that may be of interest — for example, a response containing a PTR record should include the associated SRV, TXT, A, and AAAA records describing the service.

The response message also includes the responder’s own Receiver Map.

Devices should adjust their own Receiver Maps according to their current and anticipated future activity. [We need to consider a mechanism to dampen oscillations. We don’t want to have the situation where device A is sending a file to device B, and currently they have entirely disjoint Receiver Maps, and device A decides to adopt device B’s Receiver Map in its entirety while simultaneously device B decides to adopt device A’s Receiver Map in its entirety, resulting in them swapping over and still having entirely disjoint Receiver Maps. Possibly this could be achieved by having devices make Receiver Map adjustments slowly and incrementally, so that conceptually they “meet in the middle” instead of overshooting each other. Introducing some intentional randomness into Receiver Map adjustment and announcement times could help de-synchronize Receiver Map announcements, so that one device has time to see the announcement from its peer before making its own decisions about what local adjustments to make.]

All Proximity Wi-Fi devices maintain a cache of the Receiver Maps of Proximity Wi-Fi peers they have heard from recently. When a Proximity Wi-Fi device sends a message likely to be answered by a particular Proximity Wi-Fi peer, this Receiver Map cache helps the device send the message at a time (and on a channel) when it is more likely to be received by the expected receiver. For example, when a device issues a query for an SRV or TXT record, it should pay attention to the Receiver Map of the device that sent the PTR record giving the name of the SRV or TXT record. Sending the query at a time when that device is known to be listening is more likely to elicit the desired response.

A device with limited storage may choose to limit the number of Receiver Map cache entries it stores. When the Receiver Map cache is full, the device may discard cache entries according to its own local policy. In deciding which cache entries to discard, a device may take into account various factors, including the signal strength with which a Receiver Map cache entry was received, the time elapsed since the Receiver Map cache entry was received, how recently the device communicated with that peer, and whether the device anticipates communicating again with that peer in the near future.

Dilution of Influence

The problem with synchronization is that synchronization is an all-or-nothing concept. A device can’t synchronize its clock with two different other clocks. A device can only synchronize its clock with two different peers if their clocks are also synchronized, and the logical conclusion of this is that everything would need to be synchronized. This unbounded spread of a rigid constraint though the entire mesh of communicating devices becomes an impossible requirement. The workable alternative is that a device can allow its behavior to be partially influenced by its peers, instead of totally dictated by them. This application of partial constraints is what this document refers to as “Dilution of Influence”.

If a device broadcasts information (like its clock and its channel availability) to its immediate neighbors, then this potentially imposes constraints on those immediate neighbors. If those neighbors choose to adopt that device’s clock and channel availability map in its entirety, and broadcast that to their neighbors, then they propagate that constraint outwards, potentially without limit. This results in the whole mesh having to move in rigid lock-step.

However, there is a weaker alternative, which can work. Suppose a device (B) chooses to adopt only *half* of the channel availability from a peer (A) it is actively communicating with. For the other half of its channel availability map it makes its own independent decisions, based on its own requirements, like maintaining its infrastructure connection. Now, a peer (C) that is paying attention to the channel availability of (B) also only allows that to constrain half of its radio usage, and decides the other half independently. So device (C) that is paying attention to (B) is only $^1/_4$ influenced by (A). And device (D) that is paying attention to (C) is only $^1/_8$ influenced by (A). This exponential decay of influence means that after a few hops through the mesh, the effect of any particular device has been so diluted that it is negligible. The behavior of any given device is constrained only by its immediate peers, and indirectly, to a weaker extent, by their peers, and even more indirectly, to an even weaker extent, by their peers. This dampening of influence means that we can reason meaningfully about the behavior of a little group of devices, because the effects of distant devices are sufficiently weak that they make no practical difference. It’s like the way we can calculate the orbits of the planets around the sun ignoring the effects of the infinite number of distant stars, because even though those distant stars do exert some small gravitational influence on all the planets, they are so far away that this gravitational influence is too weak to have any significant effect.

Clock Accuracy

All Proximity Wi-Fi devices MUST have a clock with a precision of one microsecond or better, and accuracy better than 500 parts per million at all temperatures in all operating conditions. With clocks of this accuracy, in the worst case the aggregate drift between a sender's and a receiver's clocks could be up to 1000 parts per million, or 0.1%.

Time values in Proximity Wi-Fi packets are generally represented using 32-bit values in microseconds. This is a modular (cyclic) time value, which means it repeats, roughly every 70 minutes. When interpreted relative to the current time value, this allows us to represent times in the range $\pm 2^{31}$ μs, roughly from 35 minutes in the past to 35 minutes in the future. To avoid ambiguity in interpreting values at the extreme edges of the range, it is RECOMMENDED that time values are limited to expressing times in the range $\pm 2^{30}$ μs relative to the current time, roughly from 17 minutes in the past to 17 minutes in the future.

All Proximity Wi-Fi devices transmitting a packet MUST be able to insert a 32-bit timestamp value into that outgoing packet giving the value of its own local clock at the moment actual transmission over the air begins, with an error of no greater than ±512 μs. (Current Wi-Fi hardware can do this, for sending beacon frames.)

All Proximity Wi-Fi devices receiving a packet MUST be able to tag that incoming packet with a 32-bit timestamp value recording the value of its own local clock at the moment reception of that packet began, with an error of no greater than ±512 μs. (Current Wi-Fi hardware can do this, for receiving beacon frames.)

These error tolerances mean that at the moment a packet is received, there could be an initial error of up to ±1 TUs in the interpretation of time values, and as the data sits in memory ageing, this error could increase at a rate of ±1 TUs per second.

This error bound needs to be taken into account when interpreting time values received from peers some time in the past. For example, Receiver Map availability windows need to be interpreted as having guard bands at the beginning and end to allow for possible clock differences. A Receiver Map entry received 5 seconds ago, from a peer advertising availability on a given channel for 16 TUs, needs to be interpreted allowing for a possible 6 TUs clock difference. A 6 TUs guard band at the start and end of the 16 TUs available time leaves a 4 TUs usable window in the middle. By 7 seconds after reception, a 16 TUs availability window is no longer useful and SHOULD be discarded. A Proximity Wi-Fi device MAY retain availability window data longer than this rule suggests, on the grounds that approximate information is better than no information at all, but care must be taken when using outdated information of this kind.

Peers that are in active communication (see Pairwise Unicast Communication below) should exchange Receiver Maps frequently enough for them to remain useful. When sufficient time has elapsed that the start and end guard bands each consume 12.5% of an availability window (leaving 75% of the time usable) an updated Receiver Map SHOULD be sent. This means that if a single Receiver Map update is lost, the next one should arrive by the time the usable time has decreased to 50% of the availability window. When appropriate, Proximity Wi-Fi devices MAY send Receiver Map updates more frequently than this. To give a representative example, if a receiver advertises a Receiver Map with listening slots 48 TUs long, then after 5 seconds the listening slots will have accumulated 6 TUs guard bands at the start and end, and it is then time to send an updated Receiver Map.

Receiver Map Format

In keeping with the IEEE 802.11 convention, all multi-byte numeric quantities in Proximity Wi-Fi are transmitted least-significant byte first (little endian).
This is the opposite of the big-endian convention used in most Internet protocols.

A Receiver Map begins with an indication of which bands and channels this device supports. [Format TBD. We may also want to encode a notion of 'preferred' channels, but only if we can think of a credible way that a peer would use this information.]

Next in the Receiver Map is:

- A four-byte expiration time, in microseconds, in the time base of the sender's local clock. In conjunction with the sender's timestamp placed in the packet on transmission, and the receiver's time tag attached to the packet on reception, the receiver is able to adjust this value to be a time relative to its own local clock. When this time is reached, the entire Receiver Map MUST be considered invalid, and discarded. The sender makes no commitment regarding its channel use beyond the expiration time of the Receiver Map.
- A two-byte quantity giving the repeat cycle of this Receiver Map in IEEE 802.11 TUs (1.024 milliseconds). The Receiver Map is considered to repeat over and over, with period given by the repeat cycle time (subject to the constraint that regardless of the number of repeats, the validity ends at the expiration time, given above).
- A two-byte count giving the number of entries in the Receiver Map

Following the Entry count is an array of those Receiver Map Entries. Each Receiver Map Entry is eight bytes long. A typical Receiver Map may contain 10-20 Entries, making the size approximately 100-200 bytes. Each Receiver Map Entry contains five fields:

- Six-bit band identifier:
  0 - 900 MHz
  1 - 2.4 GHz
  2 - 3.65 GHz
  3 - 4.9 GHz
  4 - 5.0 GHz
  5 - 5.9 GHz
  6 - 60 GHz
- Two-bit band width identifier:
  0 - 20 MHz
  1 - 40 GHz
  2 - 80 GHz
  3 - Other
- One-byte channel number, giving the channel within the indicated band
  (note that there is overlap in channel number assignments —
  e.g., 2.4 GHz and 5 GHz both have a "channel 11" —
  see <https://en.wikipedia.org/wiki/List_of_WLAN_channels>.)

- Two-byte slot duration, in IEEE 802.11 TUs. This allows durations from 1 TU to around 65 seconds. The slot duration SHOULD NOT be zero.
  Receiver Map Entries received with a duration of zero SHOULD be silently ignored (just that individual Entry, not the entire Receiver Map).
- Four-byte slot start time, in microseconds, in the time base of the sender's local clock.

The Receiver Map Entries MUST be placed in the Receiver Map in order of increasing slot start time. If a device receives a Receiver Map where the slot start times are not in order of increasing slot start time, that entire Receiver Map MUST be silently ignored.

The sender SHOULD NOT include Receiver Map Entries which would no longer be useful when their time arrives, according to the clock accuracy calculations above.
For example, the Receiver Map SHOULD NOT include Entries for 15 TUs slots more than 7 seconds in the future, or for 45 TUs slots more than 22 seconds in the future.

Since it takes time to tune the radio to a different channel, consecutive Receiver Map Entries will typically not be contiguous in time. There will be need to be (at least) brief gaps between consecutive time slots to allow time for tuning to the new channel. Since different radios require different amounts of time to tune to a different channel, it is not possible to safely make assumptions about another device's channel tuning times. Consequently, the advertised Receiver Map MUST make explicit allowance for that particular radio's channel tuning time, and only advertise listening time slots when it will truly be tuned and listening on that channel.

It is possible that future devices may have multiple Wi-Fi radio receivers, which can operate concurrently. In this case it is acceptable for the Receiver Map to have no time gap between consecutive time slots, or even overlapping time slots. Peers MUST be prepared to receive such Receiver Maps with overlapping time slots, and handle them correctly.

Receiver Map Entries are firm commitments to be listening on a certain channel at a certain time in the future. Devices SHOULD NOT, except in extreme circumstances, renege on these commitments. If a peer chooses to transmit to a device at a time the device had promised to be listening, and the device does not respond, the peer may justifiably conclude that the device is no longer there, which has the potential to result in a poor user experience if the device is in fact still there.

Because of the requirement that channel commitments, once made, are firm, devices should not overcommit their time, nor should they announce channel commitments too far in advance. Typically a device that is in a mostly idle state will announce channel commitments no further than 10 seconds into the future, and will make channel commitments for at most $^1/_3$ of that time. The remaining $^2/_3$ is left uncommitted, to give the device freedom to adjust its behavior according to changing requirements. If would be unfortunate if a user wished to exchange a file between two devices, and had to wait for ten seconds before the file transfer could start, because both devices had already committed to be on different channels for the coming ten seconds.

Simulation results confirm good results if a device commits no more than $^1/_3$ of its radio time in advance, leaving $^2/_3$ free for future needs.

With current single-radio designs, it is anticipated that a device with AC power (e.g., a printer) that is in a mostly idle state will divide its time in one of three ways:

- If the device is not associated with any AP, then it will plan to spend $^1/_3$ of its time on one of the Proximity Wi-Fi social channels of its choosing, and $^2/_3$ of its time uncommitted. During the uncommitted time it MAY, at its own discretion, continue to listen on the same channel, or on a different channel, or shut down its radio briefly to save power.
- If the device is associated with an AP on a channel that happens to be one of the Proximity Wi-Fi social channels, then it will plan to spend $^1/_3$ of its time on that infrastructure channel (timed so as to receive beacon frames from the AP and perform other tasks necessary to remain associated), and the other $^2/_3$ of its time uncommitted. During the uncommitted time it MAY continue to listen on the infrastructure channel, or on a different channel, but it is not required to do so.
- If the device is associated with an AP on a channel other than one of the Proximity Wi-Fi social channels, then it will plan to spend $^1/_6$ of its time on its infrastructure channel (to receive beacon frames from the AP and stay associated), $^1/_6$ of its time on one of the Proximity Wi-Fi social channels, and $^2/_3$ of its time uncommitted.

Discovery Message Format

Proximity Wi-Fi Discovery Messages are encoded in the same format as Multicast DNS query and response messages [RFC 6762], but with no IP or UDP header.

Proximity Wi-Fi Discovery Messages MUST include a Receiver Map data structure, to inform discovered peers how to respond to the requester.

Proximity Wi-Fi Discovery Messages are sent as 802.11 Action Frames.

Device Addressing

Proximity Wi-Fi uses only IPv6.

A Proximity Wi-Fi device configures a self-assigned IPv6 link-local address on its Proximity Wi-Fi interface, algorithmically derived from its Wi-Fi MAC address using the SLAAC algorithm [RFC 4862]. This allows a device to determine the Wi-Fi MAC address corresponding with an IPv6 link-local destination address algorithmically, without requiring IPv6 Neighbor Discovery packets.

As long as the Wi-Fi MAC addresses are unique, the corresponding IPv6 link-local addresses will be unique too.

For privacy reasons, devices may want to change their Proximity Wi-Fi MAC address from time to time, which will result in their IPv6 link-local addresses changing too.

When a device changes its address in this fashion, it appears to peers that the old device has departed, and a new device has arrived. In effect the device has shed its old identity, and taken on a new identity, without overtly revealing to peers that it is in fact still the same device.

[As we evolve this specification we will add specific rules about exactly when and how to select a new Proximity Wi-Fi link-local address.]

In some scenarios, to enable communication with devices more than a single hop away, devices may also configure other IPv6 addresses (non link-local), to be used in conjunction with a mesh routing protocol. This is an area of possible future development.

If IPv6 Neighbor Discovery needs to be used to map IPv6 destination addresses to Proximity Wi-Fi MAC addresses, it needs to be performed using Proximity Wi-Fi discovery packets, not traditional multicast or broadcast, since traditional multicast and broadcast are not supported by Proximity Wi-Fi.

Pairwise Unicast Communication

When a Proximity Wi-Fi device wishes to engage in unicast communication with a peer (typically TCP or UDP) it sends a unicast packet addressed to the peer's IPv6 link-local address.

The packet is sent at a time and on a channel where the peer's Receiver Map indicates it is likely to by listening. Unicast packets are acknowledged at the link layer, and the rate they are sent is determined using the Wi-Fi Minstrel rate adaptation algorithm, or similar.

Some portion of transmitted unicast packets include the sender's Receiver Map, at a rate appropriate to keep the peer's copy of the Receiver Map sufficiently up to date. The sender's and receiver's clocks may not run at precisely the same rate, so periodically updating the Receiver Map is necessary to prevent the information in it becoming too old to be useful, and also allows the sender to adjust its Receiver Map as necessary to accommodate changes in its activity.

Unicast packets are acknowledged in the usual IEEE 802.11 manner.

If a device loses contact with a peer with which it is communicating (or has no Receiver Map for the unicast destination address in a packet) then to establish (or reestablish) contact with that peer it sends the packet repeatedly using multicast on the social channels, in the same fashion as discovery packets.

Multipath Discovery

The basic Bonjour model assumes that if a service called "Printer" is discovered on the Ethernet interface, and a service called "Printer" is discovered on the Wi-Fi interface, they may or may not be the same service. As more devices have multiple interfaces (including having both Proximity Wi-Fi and infrastructure Wi-Fi at the same time on a single physical interface) it may be desirable to include some kind of unique identifier in the service discovery data, to determine reliably when the two discovered services are in fact just two different paths to the same service instance. This de-duplication of discovered services could be designed to share a common unique identifier with the multipath connectivity layer described above.

Multipath Connectivity

It is desirable for application software to support multipath-capable protocols, such as Multipath TCP [RFC 6824]. There may be scenarios where a service is discovered over Proximity Wi-Fi, but a better path (such as Gigabit Ethernet) to the same service is available. Multipath-capable protocols would allow application software to take advantage of the better communication path, or to "fail over" to a different path if the first path fails for some reason (e.g., the user walks out of range of Proximity Wi-Fi). Depending on the usage, "better" might mean higher throughput, lower latency, less power consumption, or a path that is superior in some other way.

Discovery/Departure

Traditional Bonjour uses record lifetimes of an hour or more. This is because departure from a network is a clearly defined event, which causes the cache to be flushed. In the case of Proximity Wi-Fi, departure may involve just moving a few feet out of range. For this reason, the lifetime for Proximity Wi-Fi records should be limited to no more than 60 seconds. [With operational experience we can fine-tune the exact record lifetime.]

Proximity Wi-Fi does not use the Passive Observation Of Failures (POOF) feature of Multicast DNS [RFC 6762]. On a traditional network, the multicast service model provides high probability that a multicast message received by one device on the link should be received by all devices on the link. Therefore, on a traditional network link with the usual transitive closure property, failure to observe expected responses to a given query issued by another device on the link is a reasonable indication that the expected responder is no longer present. Proximity Wi-Fi does not have any concept of a link, where all devices on the link are equally reachable from all other devices on the link. Each Proximity Wi-Fi has its own unique collection of peers it can hear from, and the fact that some other nearby device is unable to communicate with one of your peers is not a firm indication that you will not be able to communicate with that peer.

Name Conflicts

When a device receives Proximity Wi-Fi records conflicting with the names of its own records, it needs to select new names for those records.

[We need to debate whether this name change should be permanent or temporary. Increased device mobility brings increased opportunities for name conflicts. Users find forced name changes annoying already. On the other hand, a single change to create a permanent stable name may be better than an ever-changing unstable temporary name.]

An approach that is commonly suggested to avoid this issue is to use GUIDs, but unless we’re going to display the GUIDs to the users and expect them to remember and select between different GUIDs, then we’re going to display human-meaningful names instead, and if two services appear using the same human-meaningful name, then we’re back to having a name conflict problem to solve.

Mobility

Location information (from GPS or other sources) could be used as a factor to influence the frequency of transmissions and retransmissions. A device that has changed location could issue more frequent queries to discover new devices that have come into range, and to ascertain which previously seen devices are no longer in range.

Security Considerations

Defining Proximity Wi-Fi purely in terms of bilateral exchanges between the communicating peers helps limit the extent to which other nearby devices can inadvertently interfere with that communication. Intentional interference of various kinds is also possible, and should be guarded against in appropriate ways.

Jamming: Active malicious jamming of the radio spectrum is infeasible to guard against by technological means. This is better addressed by legislative regulation, such as the FCC rules in the USA.

Flooding: A rogue device could answer queries by generating an excessive number of responses, flooding the querying device with an excessive number of fake services. This would make it hard for the user to find the desired service among a huge list of fake services. In extreme cases this could result in resource exhaustion on the querying device, leading to complete failure to discover the desired legitimate service.

Masquerading: A rogue device could answer queries with responses containing the name of a legitimate service. This may lead to the user connecting to the rogue device instead of the intended device. End-to-end application-layer security (e.g., TLS) is required to guard against this attack. A Trust On First Use (TOFU) model or SRP-based peer identification would be appropriate to obtain the required TLS certificates to confirm the identity of the peer being communicated with.

False Information: Because a device only pays attention to peer Receiver Maps of peers with which it is in active communication, a peer that populates its Receiver Map with intentionally false information only influences other devices that it is able to entice into active communication. And the “dilution of influence” principle means that even if these devices were to partially modify their own Receiver Maps as a result of this deliberate misinformation, the effect of this misinformation is limited to the immediate neighborhood of the rogue device.

Tracking: Using the same IPv6 address(es) consistently could make it easier for untrusted eavesdroppers to track a device (for example, a user entering the same café around the same time every morning). Periodically changing the (pseudo-random) IPv6 address(es) helps mitigate against this tracking vulnerability. Similarly, using the same Wi-Fi MAC address(es) consistently could make it easier for untrusted eavesdroppers to track a device, and periodically changing these address(es) helps mitigate against this tracking vulnerability. However, this alone is not sufficient to prevent tracking. If a device consistently advertises a particular service type with a particular instance name, that could be vulnerable to tracking. If a device is consistently browsing for an uncommon service type, or consistently seeking a particular service type with a particular instance name, that could make the device vulnerable to tracking.

Exposing Vulnerable Software: Wi-Fi hotspot operators have gone to some lengths to block peer-to-peer communication, to protect vulnerable software from suffering an attack from another member of the (small) community of nearby peers. Peer-to-peer Wi-Fi has the effect of circumventing that operator protection against attack from a nearby peer. Because of this, implementers SHOULD design their APIs so that an application with listening sockets only receives packets or incoming connections over a peer-to-peer Wi-Fi interface if the application developer explicitly “opts in” to receive that functionality. This helps avoid the situation where old insecure software with known security bugs inadvertently becomes exposed to a new avenue of attack because the underlying operating system implemented peer-to-peer Wi-Fi. Software explicitly “opting in” to receive unrestricted peer-to-peer communication needs to take responsibility for having its own strong cryptographic security.

This recommendation to make peer-to-peer Wi-Fi an opt-in capability is further reason why implementing Tunneled Direct Link Setup (TDLS) is important. TDLS allows clients to benefit from the improved performance of one-hop direct communication, without sacrificing the traditional Wi-Fi operational model and simple security model offered by a traditional access point using something like WPA2 security.

[As this specification evolves, we will need to add further anticipated security risks, and techniques to mitigate those risks.]

Simulation Results

In the summer of 2016 our student intern Mohammed Hawari implemented Proximity Wi-Fi in the ns-3 network simulator. The results were extremely encouraging.

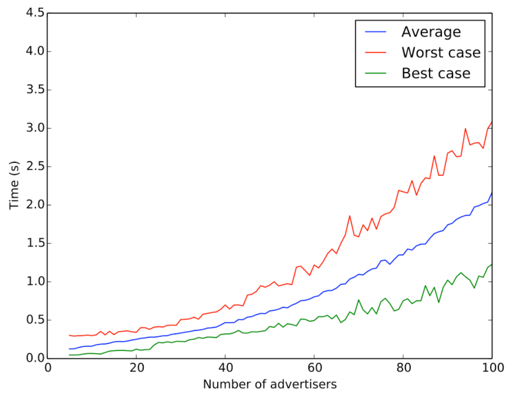
Discovery Simulation

For the discovery simulation, we deliberately chose an extreme test case.

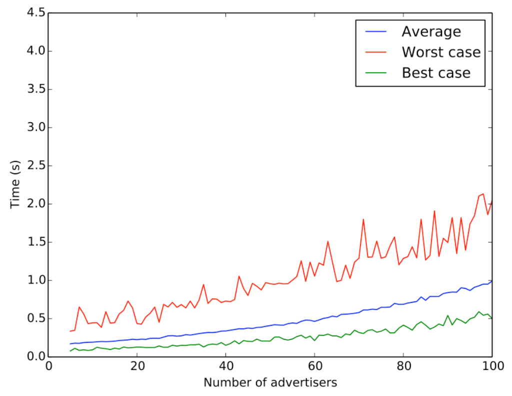
We modeled a school classroom, with three teachers with iMacs, and a number of students with iPads from 5 to 100.

Initially we recorded the discovery success rate after five seconds as a percentage, but that turned out to always be 100% (*every* teacher's iMac discovered *every* student's iPad), so it didn't make a very interesting graph.

Since the discovery was always 100% successful, we then recorded how long it took to reach 100% success. For each different number of students, we ran the simulation 100 times, with different randomized starting conditions. For each run of 100 simulations, the graphs below show the best, average, and worst times to reach 100% success. The left graph shows the results when all devices are on channel 1. The right graph shows the results when the devices are spread across multiple access points three spread across the three common W-Fi channels. Initially we expected to get better results when all devices were on the same channel, because it would reduce channel switching, and indeed, for small numbers of devices this appears to be true. But as the number of devices increases, spreading the devices across three channels reduces the amount of contention, and results in faster discovery.



Channel 1 only



Channels 1,6,11

Data Transfer Simulation

For the data transfer simulation we modeled a single sender transmitting a 2MB JPEG image simultaneously to twenty receivers, randomly spread across the three common Wi-Fi channels.

This resulted in a generous queue of outgoing TCP packets at the sender, destined to different receivers, listening on different channels, at different times, each according to its own individual Receiver Map. The sender's packet scheduling algorithm selected packets to send based on how long the packet had been waiting, and next suitable availability in the recipients Receiver Map.

In the return direction there was a stream of TCP acknowledgement packets directed from the twenty receivers back to the single sender, similarly subject to the sender's advertised Receiver Map.

No additional work had to be done to make the simulated Proximity Wi-Fi clients support twenty simultaneous transfers instead of just one. It simply resulted naturally from the Proximity Wi-Fi packet scheduling algorithm picking the next packet to send based on some simple rules, and from TCP's ability to build a reliable byte stream from an underlying datagram service.

Using simulated 802.11n hardware, the twenty simultaneous transfers of the 2MB JPEG completed in an average of 15 seconds.

Like the discovery simulation, this is a similarly extreme test case. Today most people are happy when they are able to AirDrop a single image successfully to one person, and attempting to transfer simultaneously to twenty receivers is extremely rare. Nevertheless, with Proximity Wi-Fi even this extreme case works flawlessly.

More on Synchronization

Feedback from NAN/AWDL proponents has argued that time synchronization, or the lack of it, in Proximity Wi-Fi, is no different from how NAN/AWDL works. The feedback argues that, just like Proximity Wi-Fi, NAN/AWDL devices don't synchronize clocks either. NAN/AWDL devices don't actually change the value in their hardware time registers to match the Anchor Master (Top Master). Instead they just store an offset giving the difference between their local clock value and the single shared timebase that all devices have agreed. Thus, the feedback argues, there is no clock synchronization in NAN/AWDL either, just recording of local offset values that allow conversion of time values between each device's local time and the single agreed common timebase.

This argument misses a very important point. In protocol design, internal implementation choices are largely irrelevant. A protocol specification documents external behavior, not the internal implementation choices of how that external behavior is achieved. One device could implement a protocol using Intel x86 machine code instructions, while another implements the same protocol using ARM machine code instructions. One device could have a hardware clock that counts in microseconds, while another has a hardware clock that counts in nanoseconds, and then divides that clock value by 1000 to get a value with microsecond resolution. One device could change its hardware clock value to match some external clock, while another could use a local hardware clock value plus an offset to generate values matching the external clock.

The objection to synchronization in NAN/AWDL is not concerned with synchronization of *internal* implementation details, which are irrelevant and invisible to outside observers; the objection is to *external* synchronization of behavior. NAN/AWDL requires all devices to be listening on the same channel, at the same time, during Discovery Window Zero, so that senders know the time and channel to send packets when all devices will be listening on that channel. It is this constraint on external behavior that is problematic.

Regardless of whether a NAN/AWDL device changes its local clock to match an external time value, or just records the difference, all NAN/AWDL devices have to *agree* on a single common external time value against which to record those local differences, and then, using that agreed common external time value, all perform externally visible actions at the same time. Historically, distributed agreement protocols have proven difficult, even on reliable wired networks. On a wireless network, with packet loss, interference, and devices that don't remain stationary, distributed agreement protocols are even harder. Several years of large-scale operational experience with AWDL on iOS 6, iOS 7, iOS 8, iOS 9, and iOS 10 would appear to support the idea that it's difficult to make this work reliably.

This property leads to an important functional difference between NAN/AWDL and Proximity Wi-Fi. NAN/AWDL devices need to arrive at a mutual agreement on what single common external time value they will all use (regardless of how it may be represented internally). That need for multi-party mutual agreement is the problem.

Acknowledgements

Thanks for suggestions and comments from Christophe Allie, Daniel Borges, Bob Bradley, Charles Dominguez, Chris Hartman, Pete Heerboth, Marc Krochmal, Lawrie Kurian, Olivier Mardinian, Tim McCoy, Avery Pennarun, John “Spiff” Saxton, David Schinazi, Vividh Siddha, Sidharth Thakur, Brian Tucker, and Andreas Wolf.

Special thanks to Mohammed Hawari, who spent his summer internship validating the Proximity Wi-Fi design using the ns-3 network simulator, and contributing valuable refinements and improvements.

[If I’ve inadvertently omitted your name, please let me know and I’ll add you.]