A Network Testbed for Ad-Hoc Communications using Raspberry Pi and 802.11

Edoardo Biagioni
University of Hawai'i at Mānoa
esb@hawaii.edu

Abstract

We have built a testbed ad-hoc wireless network to evaluate the AllNet ad-hoc networking protocol. The testbed currently consists of 4 Raspberry Pi Zero W embedded systems and a linux laptop, all using 802.11/WiFi ad-hoc (IBSS) mode. The embedded systems are placed in a line such that each is only able to reliably communicate with one system before it and one system after it in line.

The testbed displays phenomena that are observed in real life, including:

- *greater delay to reach devices that are farther away*
- *variability in the round-trip time to each device*
- *the current version of the AllNet protocol (AllNet 3.2) successfully prioritizes messages. In particular, trace messages, which are sent with least priority, are rarely delivered if mainstream data traffic from the AllNet network is allowed onto the testbed.*
- *system connectivity varies over time, sometimes allowing direct links between systems that are normally unable to communicate*

The paper includes practical considerations of testbed deployment using the Raspberry Pi, and an analysis of the performance of the AllNet protocol that is driving improvements in the design and implementation.

1. Introduction

Wireless ad-hoc networks have been simulated extensively over the past two decades. The simulation often involves a degree of mobility, frequently within a confined space – the "bouncing balls" model. When real testbeds of wireless ad-hoc networks do appear in the literature, they are typically expensive to deploy in both time and resources.

The Raspberry Pi is a linux-capable ARM-based device. It was originally designed for educational purposes, but it is also suitable as the core of an embedded system. This testbed uses the Raspbian distribution of Linux, which is the default operating system for the Raspberry Pi – in general, one could use any of several distributions of different operating systems. Many models of the Pi are available, including the Raspberry Pi Zero, the simplest model of the Raspberry Pi line. In February 2017, the Raspberry Pi Zero W [6] upgraded the Raspberry Pi Zero to include WiFi 802.11 and Bluetooth connectivity.

The price of the basic Raspberry Pi Zero W system is $10 each. While this does not include a power supply (nor a keyboard or display), any of the widely-available 5V micro-USB power sources can be used – the Raspberry Pi Zero W is rated at 0.5W when idle, and under 2W (400mA at 5V) under all conditions. The keyboard and display are only needed for system installation and maintenance, so a single keyboard and display can be used for any number of Raspberry Pis.

The AllNet project [1] is designed for communication among mobile devices over ad-hoc networks as well as the Internet. Evaluating the performance of AllNet over the Internet is relatively simple: the AllNet project operates a number of virtual machines which, together with some real systems, self-organize into a Distributed Hash Table (e.g., see [11]) to support Internet-based communication for AllNet. Since AllNet is designed to be frugal with resources, one of the real systems is an old 32-bit Linux laptop.

Evaluating the performance over ad-hoc networks (rather than only the Internet) suggests the use of an actual ad-hoc network. An evaluation network can be built using actual mobile devices such as cellphones or tablets, but such devices typically are more expensive than the Raspberry Pi – or, in the case of older models, have more limited availability. Further, such systems have fewer interfaces, support a reduced selection of operating systems, and may be more subject to theft if deployed in unsecured locations.

For initial testing of the AllNet protocol over ad-hoc networks, we have therefore built a network using one Linux laptop and four Raspberry Pi Zero W devices. The Raspberry Pis have been deployed in stationary locations around our office building, whereas the laptop offers mobility when desired. The AllNet messages are transmitted over 802.11 Wi-Fi used in ad-hoc ("`ibss`") mode, which has has a range of about 30-40 meters in this building[1]. The overall maximum distance between the endpoints of this 4-hop network when each device is placed so as to connect to only one device before it and one device after it in line is approximately 100m measured along the line.

Such deployment for testing is complicated by the fact that connectivity between any two devices also varies over time, a real-life phenomenon that is hard to realistically reproduce in simulation. Because of this variability, our four-hop network was occasionally a three-hop network, a few times a two-hop network, and sometimes just disconnected. Finding a deployment that most reliably behaved as a four-hop network required trial and error and the cooperation of many occupants of our office building.

The contributions of this paper include:

- a description of the testbed, built using four off-the-shelf Raspberry Pi Zero W and a Linux laptop.
- results on the short- and medium-term performance of a realistic low power ad-hoc network under controlled conditions.
- motivations and design of the AllNet low-power ad-hoc communications protocol, together with measurements of its performance and considerations about future improvements

2. Description of the Testbed

2.1. Hardware

The Raspberry Pi family of computers was designed for educational purposes, to encourage individuals to learn to apply technology to solve new problems. Different versions of the Raspberry Pi vary in the connectivity they support, with many having an Ethernet interface. Differences between the Raspberry Pi Zero and earlier versions of the Raspberry Pi indicate that the Pi Zero was designed to provide a simpler and cheaper system. The Raspberry Pi Zero W augments this simplicity with a radio that supports 802.11 Wi-Fi and Bluetooth. These systems are available for about $10 per system. Such a system, like all Raspberry Pi systems, includes an SD card for persistent storage.

An embedded system based on the Raspberry Pi Zero W must add to this at least a 5V micro-USB power supply. To support a user interface, an Raspberry Pi Zero W must be further augmented with a display, keyboard, and/or mouse.

The display and keyboard were used in this testbed only for initial setup and configuration: while operating as part of the testbed, each device had a power supply but no other peripherals. Other testbeds may opt to use a different version of the Raspberry Pi (or other system) that provides 802.3 Ethernet as well as wireless communications, particularly to provide an access and debugging network. For our testbed, however, forgoing the Ethernet gave us more flexibility in placing the units in offices and other spaces that are in daily use and don't necessarily have convenient Ethernet ports nearby. Data and logs that were not reported over the network were stored on the SD card, which could from time to time be removed and saved to a regular computer.

In essence, these Raspberry Pi Zero W devices are used as one would use devices in the Internet of Things (IoT). Unlike some IoT devices, for this testbed each device is plugged into wall power rather than a battery.

2.2. Software

The operating system we run on the Raspberry Pi Zero W is a version of Raspbian, a distribution of Linux ported to the Raspberry Pi family of computers. Because the development took place on standalone computers rather than the Pi, for the Pi we selected a version of Linux that did not provide a GUI, instead relying on the command line to perform all necessary tasks. The absence of a GUI saves both processing power and space on the SD card.

Once cross-compiled for the ARM, AllNet runs unmodified on the Raspberry Pi. The AllNet command-line utilities are available, and are sufficient for our testing. Two such utilities in particular must be mentioned.

- AllNet trace [3], or `trace`, sends a special AllNet message that, on systems that are not heavily loaded, elicits a specific response (trace reply) message. This can be used in a manner analogous to `ping`, `traceroute`, or a combination of both
- `allnet-sniffer` records all messages

[1]We and others have found that the range is greatest outdoors and varies depending in part on the composition of intervening structures. This paper does not describe direct measurements of transmission range

received and the time at which they were received.

AllNet trace may be used for both performance monitoring and network diagnostics. Specifically, AllNet trace can be used to see whether any given device in the network can be reached, and if so, what intermediate devices the message visited in reaching the target device. AllNet trace can also be used to elicit a response from any device in the network. Because AllNet traffic is prioritized and trace messages have the lowest possible priority, trace and response messages may be arbitrarily delayed or discarded if other traffic is present. This low priority is a feature, preventing the use of trace messages for denial of service, and also guaranteeing that if trace messages are successfully delivered, other traffic should also be delivered.

An additional program of note is `xtime`, which sends out a time signal. If clocks on different devices are synchronized, `xtime` is useful for measuring network latency. Conversely, if clocks are not synchronized, the `xtime` messages can be used to set the local clock to the received time. Since the clock on the Raspberry Pi does not track time when the Pi is depowered (on the next boot, the clock is set to the most recently saved time), in our testbed `xtime` can be used to set the time on the Raspberry pi devices after they have been booted. Since AllNet often takes several seconds to deliver a message on the ad-hoc network, these “synchronized” clocks may still be a few seconds late.

Each trace message may request that devices forwarding the trace also record in the outgoing trace their ID and the local time at which the message was received. These IDs and timestamps are then returned in the trace reply message. The AllNet trace program only prints these timestamps if they are sane, that is, if they fall between the local time when the original trace was sent and the local time when the corresponding trace reply was received.

2.3. Connectivity and Geometry

The testbed is deployed on one floor of an office building which has a square floorplan, with the sides of the square approximately 50m long.

Along the outside of the floor are offices and stairwells and elevators – the elevators and stairwells are important because they are in concrete shafts. In contrast, apart from the occasional concrete pillar, the walls between the offices are less dense than concrete.

Inwards of the offices are research spaces, which have cubicle dividers, and then an open hallway all around the floor.

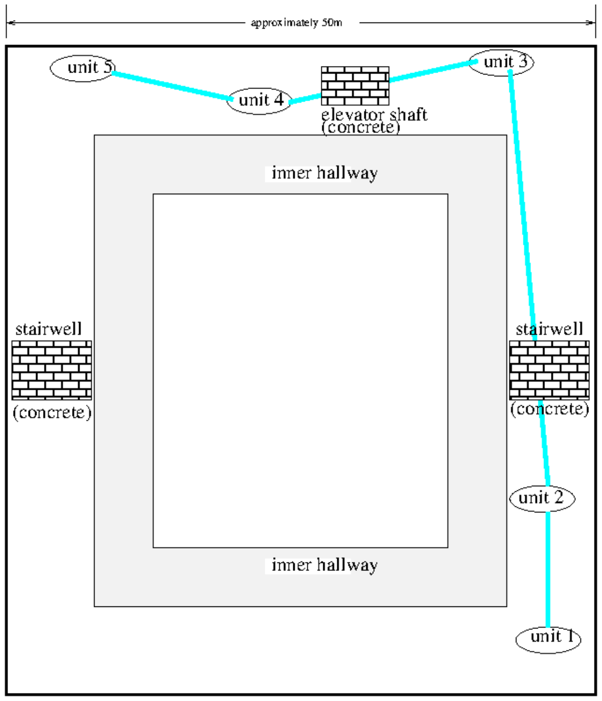

Figure 1. Approximate map showing the location of units in this ad-hoc network testbed. Colored lines indicate connectivity.

Inwards of the hallway are labs and service areas, including some concrete walls.

The laptop from which all experiments are performed (unit 1) is located in one corner of the building. All the other units, unit 2 through unit 5, are Raspberry Pi Zero W devices. Unit 2 is in a research space about 5m from unit 1. Unit 3 is in an office at the far end of the same side of the building. Unit 3 is often not reachable from unit 1, but is reliably reachable from unit 2.

Unit 4 is halfway down the next side of the building, and unit 5 near the end of the same side, as shown in Figure 1. Unit 4 can normally only directly communicate with unit 3 and unit 5.

2.4. Maintenance

Because our testbed does not include a backup maintenance network, accessing the devices in our testbed requires physically removing the device, or at least the SD card, to a location where it may be accessed. A viable alternative (that we did not practice) would be to instead bring the monitor and keyboard to the device.

The typical procedure when a software defect is detected and fixed is to re-install all AllNet software on each of the SD cards. Generally, the standard

AllNet source code is cross-compiled on a desktop computer (Intel x86 architecture) using `gnueabi` to produce ARM executables, then installed to each of the Raspberry Pi Zero W devices by inserting the device's SD card into the computer and writing the files directly to the SD card.

On the Raspberry Pi Zero W units, the allnet daemon is started automatically by placing an appropriate script in `/etc/init.d`. After re-installing the SD and rebooting the unit, if desired, the local clocks on the individual units can be reset using `xtime` in one of two ways: either from the laptop at one end of the testbed, or by carrying the laptop into the range of each individual unit. The latter delivers the `xtime` messages over fewer hops, and therefore results in more accurate local clocks. Even with this more accurate setting, however, it is normal for the Pi clocks to be off by a second or more.

Occasionally, the Raspberry Pi devices would stop responding to trace messages. This is a problem that many iOT devices can be expected to exhibit during development. For this testbed, further testing is required to determine whether the problem was with the devices and/or operating systems, or with the AllNet software. In each of these cases, rebooting resolved the problem.

3. Results

We have used the testbed to run AllNet traces from the laptop to itself and to each of the Raspberry Pi devices. Because AllNet is designed to carry interpersonal data, particularly text messages, we are most interested in end-to-end message latency rather than throughput – because AllNet transmits each message multiple times and, to save energy and bandwidth, frequently refrains from transmitting, overall throughput is likely to be low, and not likely to reflect the usefulness of AllNet.

A typical trace was performed on April 11th, 2018. consisting of the following:

- first, 10 broadcast trace messages were sent over the course of an hour (each 6 minutes apart).
- next, about 100 individual trace messages were sent to each of the Raspberry Pi devices, each 8 minutes apart, for a total of about 13 hours and 20 minutes.

In the broadcast trace, 45 responses out of a possible 50 were received, with units 4 and 5 responding eight times each and unit 3 responding 9 times. In the one case that unit 3 failed to respond, units 4 and 5 responded, probably meaning unit 3 received the trace message but the response from unit 3 did not make it back to unit 1. This case is identified as trace 2 in Figure 2.

```
1: b1  0 hop    0.001s rtt   0.000s ts
1: b2  1 hop    6.968s rtt   2.648s ts
1: b3  2 hop   28.135s rtt   3.998s ts
1: b4  3 hop   35.465s rtt  26.860s ts
1: b5  4 hop   92.018s rtt  38.343s ts
2: b1  0 hop    0.003s rtt   0.002s ts
2: b2  1 hop    6.820s rtt   0.722s ts
2: b4  3 hop  109.502s rtt  21.794s ts
2: b5  4 hop  144.764s rtt  27.808s ts
```

Figure 2. Two broadcast traces on April 11th, 2018. The unit addresses are `b1` for unit 1, `b2` for unit 2, and so on. Note that unit 3 did not respond to the second trace, but likely did forward the message to unit 4, which reports being 3 hops away. Timestamps (`ts`) are based on device-local time, which is accurate for the device initiating the trace, and likely one or more seconds behind for all the other devices.

```
5: b2  1 hop    1.951s rtt
5: b3  2 hop   11.890s rtt   2.770s ts
   b1  0 hop                 0.001s ts
   b2  1 hop
   b3  2 hop                24.085s ts
   b4  3 hop                54.887s ts
5: b5  4 hop  236.619s rtt  61.543s ts
```

Figure 3. The response to message number 5 from a more extended trace (also on April 11th) sending one message to each of the Raspberry Pi units, and requesting forwarding details for the message sent to unit 5. Careful readers may note that no response was received directly from unit 4, yet by studying the response from unit 5 it is apparent that unit 4 received and forwarded the original trace message.

The individual traces were started at the same time, but each delayed by a different amount: 478, 479, 480, and 481 seconds each, respectively, for units 2, 3, 4, and 5. In addition, trace messages to unit 5 requested that the ID and local time of any unit forwarding the trace message be recorded in the trace message itself.

In this more extended trace, part of which is shown in Figure 3, 100% of the traces sent to unit 2 got a response, 98% of the traces sent to unit 3, 92% of the traces sent to unit 4, and 23% of the traces sent to unit 5. These results are summarized in Table 1.

Although the 23% response rate for unit 5 looks dismal, this is for an implementation under test. Just two days later, after some improvements to the code, unit 5 had a 96% response rate and minimum/mean/max response times of 14.3/56.9/156.5s, as shown in Table 2.

In this latter trace, 101 trace messages were sent to unit 5, and 97 replies were received. Of these 97 traces

destination	response rate	min time	mean time	max time
unit 2	100%	1.4s	6.9s	24.8s
unit 3	98%	2.0s	28.6s	141.4s
unit 4	92%	13.6s	52.6s	160.3s
unit 5	23%	28.6s	133.8s	278.8s

Table 1. Result of sending about 100 trace messages to the Raspberry Pi Zero W units at 8-minute intervals (April 11th, 2018).

destination	response rate	min time	mean time	max time
unit 2	100%	0.8s	6.8s	18.8s
unit 3	100%	0.9s	17.9s	82.9s
unit 4	96%	5.8s	27.9s	97.0s
unit 5	96%	14.3s	56.9s	155.5s

Table 2. Result of sending about 100 trace messages to the Raspberry Pi Zero W units at 8-minute intervals (April 13th, 2018).

```
    b1 0 hop              0.001s ts
    b2 1 hop              4.556s ts
    b3 2 hop              4.482s ts
    b4 3 hop             11.035s ts
37: b5 4 hop 52.359s rtt
```

Figure 4. Trace message 37 to unit 5 (April 13th, 2018). This kind of trace message requests that addresses and (local) time be added to trace messages as they are forwarded.

that resulted in replies, 46 took the expected 4-hop path over all the units, 50 took a 3-hop path from unit 1 to unit 3, bypassing unit 2, and 1 took a 2-hop path from unit 1 to unit 3 then directly to unit 5, bypassing both units 2 and 4. Once an AllNet device receives a trace message, it ignores any duplicate trace messages received from its other neighbors, so the trace messages accurately reflect the shortest path to each responding destination.

In all these tests, the largest round-trip time recorded was on February 2nd, 2018: unit 5 replied to unit 1 in over 583s. This test was unusual in that unit 1 had been told to wait for as long as 600 seconds (10 minutes, more than the usual 1-8 minutes) for a reply. However, other tests in which replies were sought for up to 1,800 seconds (30 minutes) did not record replies taking any longer than this.

4. Analysis

This testbed was built specifically to evaluate the performance of AllNet in ad-hoc peer-to-peer mode.

Based on early work determining that the energy cost of keeping the radio on, even in receive mode, is a substantial contributor to the overall energy cost of a minimal embedded device [12], the design of the AllNet ad-hoc protocol was specifically aimed to keep the radio off most of the time, while limiting the latency to what is acceptable in the transmission of interpersonal communications such as text-messages intended for other people. The goals in the initial design of this protocol were [4]:

- to limit the duty cycle ("on" time) of the radio device to about 1%
- to forward text messages with reasonable latency

To satisfy these two goals, the protocol was designed to have a 5 second cycle on ad-hoc networks (as opposed to on the Internet, where AllNet messages are sent immediately). At a randomly selected time within this cycle, a device X sends an AllNet *beacon* message to announce it is ready to receive, then listens for a *beacon interval* of up to 50ms for another AllNet device to respond.

If another AllNet device Y is listening and receives a beacon and has messages to send, it waits for a time uniformly selected from the beacon interval, then sends a *beacon reply*. Once X receives a beacon reply in response to its beacon message, it sends to Y a *beacon grant* announcing how long it is willing to receive – this time is also typically 50ms. At the default 802.11 ad-hoc data rate of 1Mb/s, 50ms can be used to send up to approximately $6,000$ bytes.

Devices Z that overhear a beacon grant directed to another device must keep quiet for the time the receiver is willing to receive – this is similar to the 802.11 RTS/CTS mechanism.

Devices that have high-priority data to send, e.g. devices with data generated locally, keep their interfaces powered continuously so they have more opportunities to send their data. Devices that are in energy saving mode (because they have no high priority data to send, or for any other reason) turn the interface on before sending a beacon, then turn it off after the completion of a beacon cycle, to satisfy the goal of limiting the duty cycle. However, for the evaluation described in this paper, all interfaces were placed in high-priority mode and were never turned off. This results in a basic 5s cycle for each device to send its beacon and receive data from at least one neighboring device. Since the transmission may occur at any time within the 5s cycle, round-trip times are not necessarily multiples of 5s.

The bigger limitation is the 1% duty cycle of transmissions. If there are many messages queued for transmission, only the highest priority 6KB or so will be sent in any 5s interval. Since any data traffic has higher priority than and pre-empts trace messages, with random forwarding any significant data traffic will effectively prevent the transmission of trace messages. This has been observed on several occasions, and led us to isolating the test network from the wider AllNet network for the purpose of these tests.

4.1. Analysis of a Re-implementation of ad-hoc forwarding

In practice, trace messages lose out to data messages due to the random forwarding model. A transmission model in which message are forwarded to each device at most once, which is a particular kind of gossip or epidemic transmission model, allows high-priority traffic to be sent first, but then also allow lower-priority traffic such as trace messages to be sent, whereas with random forwarding only the high-priority traffic really has a chance to be sent.

This model has been implemented in an experimental version of the AllNet software, which is planned to be released as AllNet 3.3.

This experimental software does not (yet) use beacons. Instead ad-hoc interfaces are treated like any other interface, and messages are sent whenever they are ready. This is analogous to turning off RTS/CTS for 802.11 devices, and also removes any reference to a 5s cycle.

Part of a trace from AllNet 3.3 is shown in Figure 5. Of 10 trace messages sent in this test, 9 were received within a 5-seconds timeout, with a minimum time of 0.4s, maximum time 4.4s and a mean of 2.0s.

```
   b1  0 hop
   b2  1 hop
   b3  2 hop
   b4  3 hop
7: b5  4 hop   4.377s rtt
```

Figure 5. A trace message to unit 5 on September 7th, 2018. Clocks were not synchronized, so timestamps have been omitted.

The most interesting challenge of epidemic forwarding is its statefulness – messages are to be forwarded to a peer only if the peer has not already received the message. This can be accomplished by assigning a unique identifier to each device, or at the time beacons are exchanged by having each device report which messages it has already received before receiving new messages. The second strategy requires additional communication and processing at the time of the beacon exchange, so for now we are implementing the first strategy. Each device self-selects a random ID, which is likely to be unique as long as the IDs have enough bits (AllNet uses 128 bits). Each device must then keep track of the IDs to which each cached message has already been sent. The device should then only forward cached messages that have not already been sent to this ID.

To see how this works, consider device X sending message M to device Y. After sending, device X marks M as sent to Y, meaning it will never again send M to Y [2]. However, it is possible that Y was lost in transit, in which case the message will only ever be received by Y if Y receives it from a different host Z, or if the original sender creates a new message M' with the same contents but enough difference that devices will treat M' as different from M. The latter should happen automatically if a data message is not acknowledged within a timeout, but has not yet been implemented in this experimental version.

5. Related Work

The AllNet project has been ongoing since 2011, so there are many publications related to different aspects of the project. The most relevant to this work is the one describing in detail the trace messages themselves [3]. The computation of social distance [2], used in

[2]The message cache in version 3.3 is designed to take a fixed amount of space, which means that over time this information may be forgotten and message M may actually be resent to Y some time in the future. In what follows, we ignore this uncommon sequence of events.

determining message priority, is designed (and as of 2018, not yet implemented) to take place by exchanging pseudonymous information about individuals in one's social network. A recent overview of the project can be found in a 2017 presentation [5]. The ideas about gamifying AllNet to engage users, also not yet implemented, are described in an early AllNet paper [7].

There are many other ad-hoc networks, both proposed and implemented. Most notable in the news has been Firechat [10], which is available for iOS devices but does not provide security for messages. Another notable deployed wireless ad-hoc network is FabFi [8], which aims to build an infrastructure that serves as an alternative to the Internet. Unlike AllNet, these networks generally do not use the Internet for message passing, even when available. Like AllNet, Firechat has mobile devices forward messages, whereas FabFi does not.

Gossip and epidemic protocols are also widespread and in general these terms cover a variety of different algorithms.

Finally, wireless ad-hoc networking testbeds are abundant. Many of the latest efforts are being sponsored by the U.S. National Science Foundation's "Platforms for Advanced Wireless Research" program [9]. These are generally large-scale efforts using dedicated hardware and designed for general-purpose experimentation. In contrast, the testbed described in this paper, particularly the use of the Raspberry Pi Zero W devices, is inexpensive and can be adapted for use by individual researchers.

6. Future Work and Conclusion

The experience here has been a driver to move from the "priority-informed random forwarding" model of earlier versions of AllNet (3.2.4 and prior) to the "at-most-once" transmission model, as mentioned in Section 4. This refactoring is in progress and should be available in AllNet versions 3.3 and above.

Random forwarding (whether or not informed by priority) is stateless, since data is forwarded based on current connectivity rather than past history. In contrast, the the "at-most-once" transmission model requires tracking whether a currently connected device has received a specific message before. This generally requires more storage, but storage, in the range of several megabytes, is one resource that at this time even resource-starved mobile devices seem to have in abundance.

We also look forward to experimenting with the Bluetooth capabilities of the Raspberry Pi Zero W devices.

To summarize our experience in this paper, we have found that Raspberry Pi Zero W devices can be quite useful to researchers. Even in this age of inexpensive cloud computing, which provide researchers access to large numbers of virtual devices, the ability to build a real wireless testbed out of inexpensive and reliable devices with all the WiFi and Bluetooth capability that Linux supports has value for the practical evaluation of research in wireless ad-hoc networks.

References

[1] Biagioni, "Ubiquitous Interpersonal Communication over Ad-Hoc Networks and the Internet", at the 47th HICSS (Hawaii International Conference on Systems Sciences), in January 2014.

[2] Biagioni, "Distributed Anonymous Computation of Social Distance", CCNC 2016, the 13th Annual IEEE Consumer Communications and Networking Conference, 9-12 January 2016, Las Vegas.

[3] Biagioni, "A Diagnostic Tool for Ad-Hoc and Delay-Tolerant Networks", 42nd conference on Local Computer Networks (LCN 2017), October 8-12, 2017, Singapore.

[4] Biagioni, "Ubiquitous Interpersonal Communication over Ad-Hoc Networks and the Internet". Unpublished, 2013. Available from http://alnt.org/2013.sigcomm.pdf

[5] Biagioni, "AllNet: ubiquitous interpersonal communications", presentation from September 18th, 2017. http://alnt.org/cis-talk.pdf

[6] Brodkin, "New $10 Raspberry Pi Zero comes with Wi-Fi and Bluetooth", Ars Technica, February 28, 2017. https://arstechnica.com/information-technology/2017/02/new-10-raspberry-p

[7] C. Desiato and E. Biagioni, "Sharing Networking Resources to Create a Pervasive Infrastructure", Ninth Int'l Conference on Technology, Knowledge, and Society, 13-14 January 2013, Vancouver, Canada.

[8] Matthew Humphries, "FabFi: an open source wireless network for $60 per node", geek.com, June 27 2011. https://www.geek.com/chips/fabfi-an-open-source-wireless-network-for-60-p

[9] National Science Foundation, "Platforms for Advanced Wireless Research, PAWR", https://advancedwireless.org/ (retrieved June 2018)

[10] Tom Simonite, "The Latest Chat App for iPhone Needs No Internet Connection", Technology Review, March 28, 2014.

[11] Stoica, Morris, Karger, Kaashoek, and Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications", ACM SigCOMM 2001.

[12] Lin Zhong, "Power Consumption by Wireless Communication", lecture notes, 2011. Available from http://www.ruf.rice.edu/˜mobile/elec518/lectures/3-wireless.pdf