Adventures in Networking

Paul Lutus
http://arachnoid.com

January 27, 2016

Most recent revision: June 19, 2017

Abstract

This article is as much a personal reminiscence as an exposition on networking methods. It covers several decades of time and describes a progression from the "sneakernet" networking approach, moves through intermediate steps including my experiments in transoceanic radio networking, and finally describes some modern network management methods like Zeroconf.

Contents

List of Figures

1 Networking Chronology

1.1 From None to Peer-to-Peer

Figure 1: My first computer (1977)

Early in the history of electronic computing the mere existence of a computer was sufficiently extraordinary that any thought of connecting one computer to another was put aside. Years later, as the personal computer era began, I went through the same process on a smaller scale – a stack of cassette tapes held the data generated by my one computer (see Figure 1), and the thought of having more than one computer was quite beyond imagining. I eventually acquired more computers, the machines needed to share data, but a cassette tape or a floppy disk and a short walk (a "sneakernet"[1]) seemed the obvious solution.

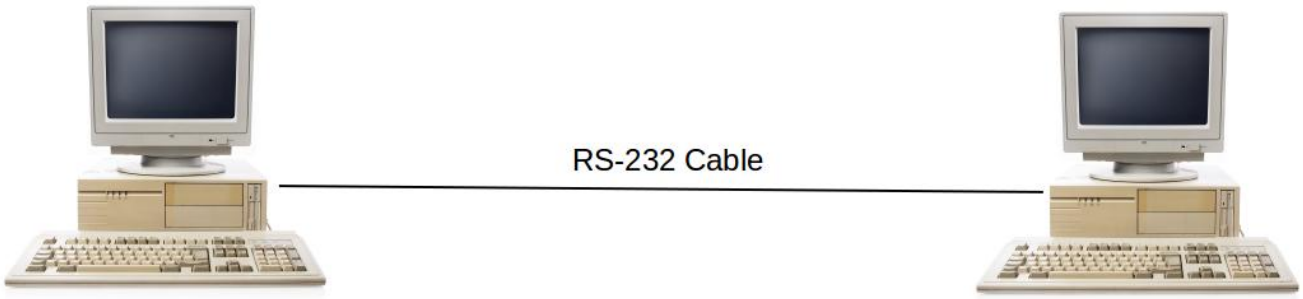

Figure 2: Simple Peer-to-Peer Network

A few years later, after I had acquired more and better machines, machines able to generate too much data for the sneakernet, I created my first actual network – a peer-to-peer scheme based on a serial cable that linked two machines by way of their RS-232[2] ports. The data transfer rate was very slow, but faster than the cassette tapes and floppy disks that preceded it.

Consider how this simple network functioned. Since there were only two machines, no name resolution was required, indeed the scheme simply extended the Unix stream/pipeline[3] idea, with a Unix socket at each end of the circuit. There were no sophisticated features – I couldn't see the other machine's filesystem, instead I would set up a pipeline on one machine, then move to the other machine to complete the transfer. Very primitive.

1.2 High Seas Network

Figure 3: Solo ocean sailing

About this time I began one of my more ambitious personal adventures – an around-the-world solo sail in a small sailboat. Even though I was deliberately getting away from it all, I was aware that the voyage would extend over several years, so I needed a way to communicate with the world.

After some experiments before the voyage began, I decided to use a method called "packet radio"[4], a peer-to-peer scheme that relies on a radio link, in my case ham radio[5].

For this network, the scheme included two computers running a messaging application I wrote for the occasion – at each end, the user could sit and type a message, then put it in a queue for later transmission. The transmission took place when I next made radio contact from wherever I happened to be sailing[6].

Once I established radio contact, any queued messages were transferred bidirectionally, to be printed on paper at my house (so the people living there wouldn't need to know much about computers), or to be read from a display on my boat (to save precious paper while on the high seas).

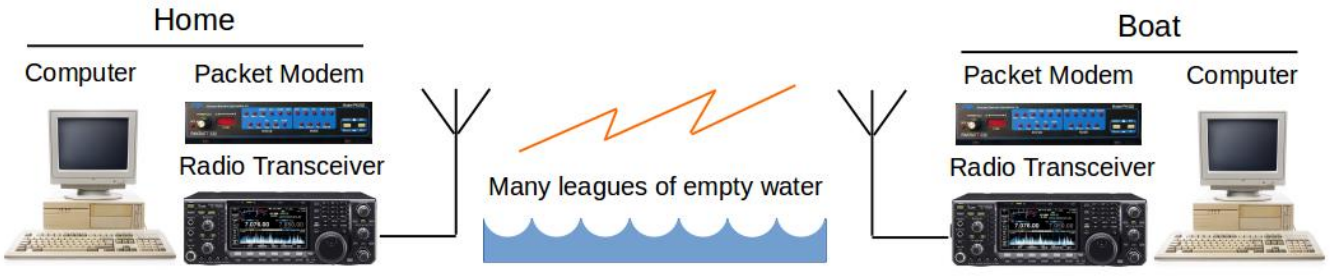

Figure 4: High Seas Packet Radio Network

This network was a bit more complicated than the earlier peer-to-peer scheme. Instead of a cable connecting

two machines a short distance apart, the computers were supported at each end by a packet modem, then a radio transceiver, then a lot of empty air.

This scheme worked reasonably well as long as I was on the same side of the planet as my house, but I eventually sailed so far that the radio link became unreliable. Some years later, having circled the globe, I sailed back to within several thousand miles of my home address and tried to reëstablish contact. The radio link was still working, but it seems the people at my house were no longer checking the message printer – not surprising, since it had been several years since the most recent message had arrived.

So, aware I had no other way to make contact, I wrote a little program that repeatedly rang the computer's bell and printed an alert message on the monitor. I tested it on the boat, transmitted it across the packet network and activated it remotely. According to later reports, it sounded like a fire alarm in my quiet house. It alerted my friends that I was once again within range of the packet network.

I tell this story just because it's interesting, but also to justify my calling a ham radio packet link a network. Obviously if I can transfer an executable file from some distant place on the Pacific Ocean to my house in Oregon, then run it remotely ... it's a network.

1.3 First Hub

After my solo sail, and at about the time the Internet first appeared in people's lives, I set up a network with more than two nodes, so a hub was required to manage things. If memory serves that hub's cabling was coaxial, not multiconductor, so it was a serial-data scheme (not a parallel transmission of multiple bits, but a time sequence of individual bits), and by modern standards it was very slow. But it was my first network with a topology resembling the modern network paradigm – a scheme that in principle could accommodate any number of machines.

I think that should provide enough history – now we can discuss some networking specifics.

2 Networking Methods

In this section I'll discuss some of the problems that a multi-node network creates – address assignment and resolution, things like that – but in an informal way with an emphasis on problem solving. The narrative assumes a Linux platform.

2.1 Fixed Addressing

In my first networks that required each machine to have a unique address, I simply configured each machine with a static address, then populated each machine's `/etc/hosts` file with the assigned addresses, and made sure this line:

```
hosts:  files dns
```

was present in `/etc/nsswitch.conf`, to assure that names were resolved by first consulting `/etc/hosts` before an online search was carried out. To expand a bit on this point – for the two essential network configuration tasks:

- For address *assignment*, I manually configured each machine's Network Interface Card (NIC) with a fixed address and netmask.
- For address *resolution* (and as shown above), I populated `/etc/hosts` file with a list of system names and assigned addresses, then used `/etc/nsswitch.conf` to direct the machine's name resolver to first consult `/etc/hosts`.

Remember this two-element process – address *assignment* and address *resolution* – because it plays a part in what follows.

The described scheme is the simplest way to assign unique network addresses and accomplish name resolution. But it doesn't scale well – as the number of machines increases, and if the network configuration changes over time, it becomes confusing, labor-intensive and eventually unworkable. Each machine needs to be hand-configured, and each machine needs to have an up-to-date copy of the `/etc/hosts` file.

2.2 DHCP

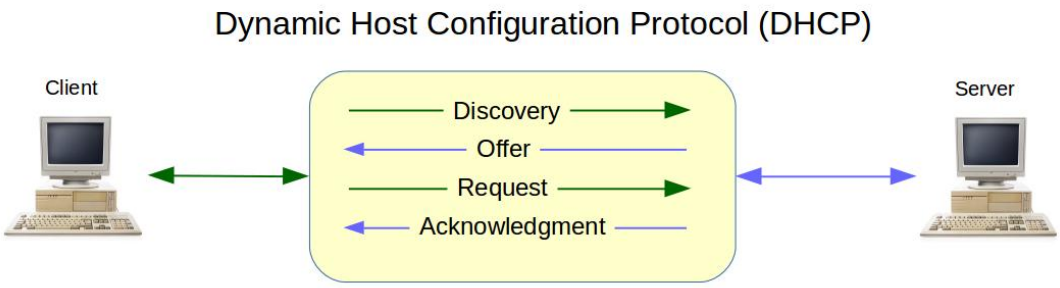

Figure 5: DHCP Address Resolution Scheme

The next step in sophistication, and the next step I took in my personal networks, was Dynamic Host Configuration Protocol or DHCP[7]. In the DHCP scheme, instead of requiring that someone manually configure a static address for each networked machine, this sequence is automatically carried out:

- A client machine broadcasts a *discovery* request on the local network (in this context "broadcast" means to no particular destination, and "discovery" means to find out whether there is a DHCP server available).
- Assuming there is a DHCP server available, it broadcasts a reply containing its own address and an *offer* of a particular client address.
- The client then typically responds on the server's address with a *request* for the offered client address.
- The server *acknowledges* the request and the offered address is assigned to the client.
- The assigned address typically has a time limit, after which the above process is repeated. This allows the DHCP server to reclaim addresses from machines that have gone offline, and for other reasons.

A drawback to DHCP in its basic form is that, even though each participating machine gets an address by which to communicate with the local network and other networks, without additional information the local machines can't communicate with each other. In the default DHCP configuration there's no provision to resolve local network names – each machine is given an address that isn't provided to the other machines, and the addresses typically change every two hours.

So, to summarize this level of networking, its primary advantage is there's no requirement to manually assign addresses or edit the content of `/etc/hosts` every time the network topology changes, but the drawback is that, even though the client machines can communicate with local and distant networks, without additional information they can't communicate with each other.

Because my machines need to communicate with each other, I required more than DHCP can offer by itself.

2.3 DHCP Address Reservation

DHCP address reservation is a scheme in which the DHCP server is provided with a table of Media Access Control (MAC)[8] addresses and corresponding IP addresses, so when the server receives a request, it hands out the IP address that specific device has been assigned in the table.

This scheme is slightly better than static address assignment, because it doesn't require the network administrator to manually configure each machine's network interface – instead, the administrator maintains two lists:

- A cross-reference list containing each machine's MAC address along with an associated IP address, and provided to the DHCP address reservation scheme.
- An up-to-date `/etc/hosts` file, shared among all the machines, containing machine names and DCHP reserved addresses.

Over the years I've owned a number of wireless routers that I was able to configure with DHCP reservations (for which each manufacturer seems to have a different name[9]), but I continued to search for a better scheme, more intelligent and flexible, maybe one that would work with devices that couldn't be instructed by means of an `/etc/hosts` file, like Android devices.

2.4 Samba

This is somewhat of a digression, since I didn't end up using the Samba[10] scheme. Samba is an open-source embodiment of Microsoft's Server Message Block (SMB)[11] network management method. Samba is installed by default on most modern Linux distributions, and once configured and enabled, it allows machines running both Linux and Windows to find each other by host name.

Unfortunately Samba's resemblance to a real name resolution/network services scheme is slight – it's not very well integrated into the Linux command set. One can access local machines by way of some utility programs like smbclient, but unlike more robust network protocols, you're obliged to use an unnatural and temporary environment to list directories and transfer files (not unlike ftp, which requires one to enter a special, limited command environment).

To summarize, Samba possesses many of the same undesirable traits as Windows – no file and directory case sensitivity, a primitive file date/time and permissions scheme, no symbolic links, and others. It just wasn't suitable for one of my primary network tasks: synchronizing two directory trees so they're identical – same files, same dates and times, same permissions. Samba/SMB is an interesting way to build a bridge between Linux and Windows machines, but it can only do this by adopting most of the undesirable traits for which Windows is famous.

3 Practical Solutions

The prior section described some methods that didn't work for me, or that worked poorly, or that were too labor-intensive for a large network. This section describes some that work well and are in current use. Two issues are covered:

- Automatic address assignment and management, and name-to-address mapping.
- Network interactions including communications and file transfers.

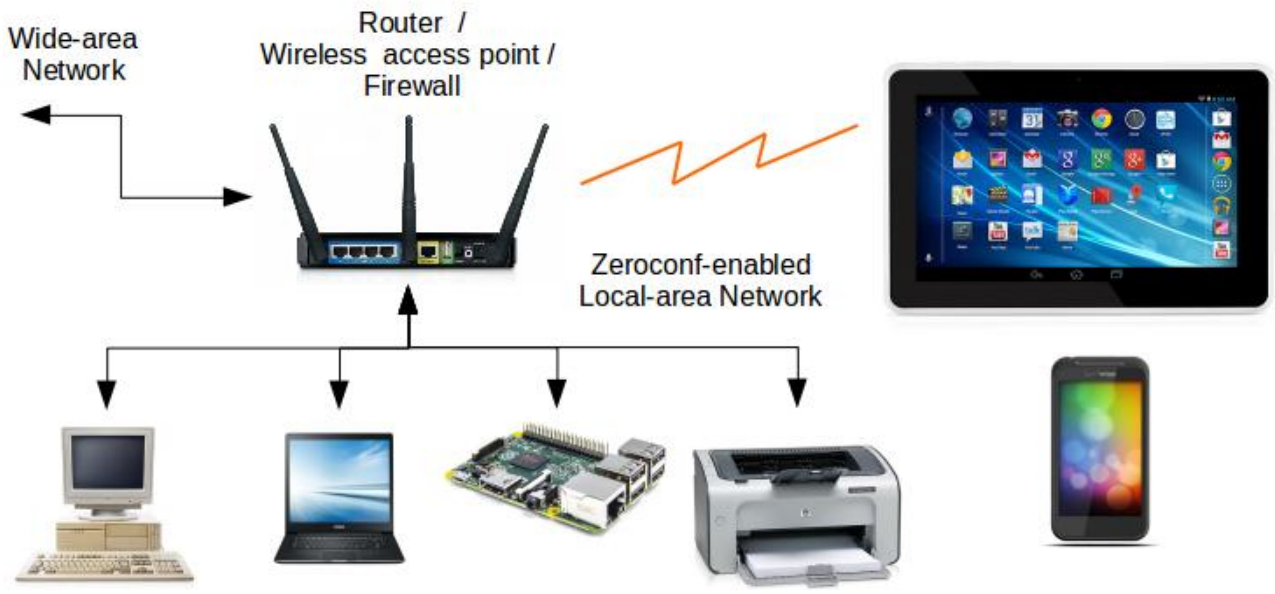

Figure 6: Modern Local Network

3.1 Zeroconf

Zeroconf[12] is a relatively new, clever scheme that provides local network name resolution and service/resource advertising with no manual configuration required, hence its name.

In the Zeroconf scheme, a DHCP server still assigns IP addresses, but once assigned an address, each machine uses the Zeroconf protocol to broadcast essential information to, and acquire information from, other machines, and there is no central Zeroconf server. Within a short time each machine on the local network acquires current information about machines and peripherals, their IP addresses, and what services and resources they offer.

The basic Zeroconf assumption is that the network will change over time. As machines enter and leave the network, as peripherals are attached and removed, and as the DHCP server periodically assigns new IP addresses, the shared Zeroconf information about services and resources is automatically brought up to date.

That's the good news. The bad news:

- Because Zeroconf's purpose is to maintain accurate and timely network state, participating machines broadcast an update any time they experience a change. This creates a lot of local network traffic.
- Zeroconf network configuration is based on trust, not verification, and local-network spoofing is easy and an obvious threat. This means:
  - The scope of the Zeroconf protocol should be limited to local, implicitly trusted networks.
  - Those who acquire Zeroconf information should use secure protocols when they apply it – in other words, don't assume Zeroconf is reliable in the way that Secure Shell is.
- Synchronization: there are times when a machine goes offline or its configuration or IP address has changed, but the Zeroconf shared state isn't updated in a timely way for all participating machines. In fairness, this is a problem common to all automatic service/resource discovery schemes.

In the Zeroconf scheme, a DHCP server still provides addresses and routing information, but most other network information is managed by Zeroconf.

3.1.1 Zeroconf on Android

On my local network, nearly all network-capable devices support Zeroconf, with the conspicuous exception of Android devices – unless they're equipped with a Zeroconf-capable application. Well, as it happens, I know of an Android application that provides Zeroconf information to the local network. I know this because I wrote it. Recent versions of my free program SSHelper[13] provide Zeroconf broadcasts of device name, IP address, and available services like the Secure Shell protocol.

3.1.2 SSHelper

This is a digression, mostly about a recently-corrected bug in SSHelper. I recently replaced my wireless routers, but the new routers, notwithstanding their many advantages over the old, don't have an easy way to write a DHCP reservation table. Until this equipment update I had been solving the name resolution problem by generating a reservation table of about 30 MAC and IP addresses for the various devices on my network and writing it directly to the old routers, without any fuss or hand entries. The new routers don't have this ability, but rather than abandon either the routers or the desire to have automatic name resolution, I decided to move to a complete reliance on Zeroconf for the first time. (Until now I had regarded Zeroconf as a nice protocol, but not essential.)

To prepare for this change, I updated all my (Debian-based) Linux machines with Zeroconf-capable software:

```
# apt-get install avahi-daemon avahi-utils avahi-dnsconfd avahi-discover libnss-mdns
```

As some of my readers may be aware, to resolve a name using Zeroconf (i.e. to distinguish a Zeroconf network name from one requiring DNS resolution), one simply appends ".local" to a network name: "ping hostname.local"*. I already had SSHelper installed on all my Android devices, so I began testing name resolution and service discovery on the local network. I quickly detected a problem – even though my desktop and laptop machines worked perfectly with Zeroconf, my Android devices weren't updating the Zeroconf shared state with their current addresses, and they sometimes became inaccessible to the local network.

An examination of the output created by –

*If desired, the ".local" domain name may be changed in `/etc/avahi/avahi-daemon.conf`

```
$ avahi-browse -a
```

– showed why. Over a period of hours, each of my Android devices began to be listed multiple times with different IP addresses. After some investigation, I uncovered a bug in my SSHelper code that wasn't retiring old instances of the Zeroconf reporting/negotiation class. So I fixed that bug, recompiled and updated SSHelper and all is well (SSHelper versions 8.2 and newer have this error corrected).

3.1.3 Zeroconf on Windows

Windows is famous for avoiding advances in technology that aren't owned by Microsoft, but with respect to Zeroconf, it's possible to make it available, and the software is free. Here are the steps:

- Acquire the Bonjour Print Services installer[14] from the Apple Support website[15].
- Either run the provided installer program as it is, or use the Windows utilities 7Zip[16] or WinRAR[17] to extract and run `Bonjour64.msi` from within the Apple Printer Services installer program. The second option has the advantage that you may avoid installing some things you might not want.
- Even though the Apple Web page makes it sound as though you're only acquiring a way to find printers, the installed service is a full embodiment of Bonjour/Zeroconf, and in a mixed Windows/Linux/Android* local network as shown in Figure 6, all the machines should be able to communicate with each other.

3.2 Secure Shell

The above section describes my present method to assign and resolve IP addresses, but there's still the issue of communications and file transfers between machines. For this I rely primarily on the various embodiments of the Secure Shell[18] protocol.

For file transfers there are alternatives on the Linux platform, like Samba and the Network File System (NFS)[19], and for communications there are telnet and others, but over a period of years experience has taught me that the available Secure Shell clients (`rsync, ssh, scp, sftp`) are faster, more secure and more reliable than the alternatives.

3.2.1 SSH Tutorial

I can't resist adding a short tutorial on one of my favorite subjects, one that causes much confusion for those configuring SSH on Linux systems.

First, to make sure you have the required Secure Shell elements:

```
# apt-get install openssh-server openssh-client rsync
```

In modern Linux distributions it's very likely these programs are already in place, and the above command may only serve to confirm this.

Here's a typical Secure Shell setup procedure:

- Generate one or more SSH keys, essential to the Secure Shell security scheme:
  - First, check to see if a key has already been generated:
    ```
    $ ls ~/.ssh/*.pub
    ```
  - If a suitable key is not yet present, generate one:
    ```
    $ ssh-keygen -t rsa -N '' -f ~/.ssh/id_rsa
    ```
  - The above command generates two files, a private key file named `id_rsa` and a public, shareable key file named `id_rsa.pub`. *Only share the public key.* And for more about the `ssh-keygen` arguments, read the man page[20].
- SSH logon problems very often result from wrong permissions for the user's .ssh directory or its contents. Here's an example of an .ssh directory with all permissions set correctly:

*With SSHelper installed.

```
drwx------  2 username users   4096 Jan 17 13:46 .
drwxr-xr-x 43 username users   4096 Jan 21 15:54 ..
-rw-r--r--  1 username users  10912 Jan 21 09:26 authorized_keys
-rw-r--r--  1 username users  10912 Jan 21 09:26 authorized_keys2
-rw-r--r--  1 username users    917 Jan 17 14:13 config
-rw-------  1 username users    668 Jan 11 03:10 id_dsa
-rw-r--r--  1 username users    605 Jan 11 03:10 id_dsa.pub
-rw-------  1 username users    227 Jan 17 13:03 id_ecdsa
-rw-r--r--  1 username users    177 Jan 17 13:03 id_ecdsa.pub
-rw-------  1 username users    411 Jan 17 13:46 id_ed25519
-rw-r--r--  1 username users     97 Jan 17 13:46 id_ed25519.pub
-rw-------  1 username users   1675 Jan 17 13:03 id_rsa
-rw-r--r--  1 username users    397 Jan 17 13:03 id_rsa.pub
-rw-r--r--  1 username users   7896 Jan 21 09:30 known_hosts
```

  - In summary, private key files and the `.ssh` directory should only be readable/writable by the directory's owner, other files should be publicly readable and only writable by their owner.
  - If the permissions on your `.ssh` directory and/or its files aren't the same as shown above, this may be the reason for failed logons and other problems.
  - The above example also shows a variety of generated keys. These keys are generated by changing the `-t` argument to `ssh-keygen` – valid arguments are `rsa`, `ecdsa`, and `ed25519`. An older key type, `dsa`, is deprecated and is to be abandoned[21].

- To analyze SSH logon problems, use the `-v` (verbose) command-line option. More repetitions of `-v` provide more detail:

  ```
  $ ssh -vvv user@hostname
  ```

- To allow passwordless logons, share your public key with the target system like this:
  - The simplest, default method:

    ```
    ssh-copy-id username@hostname
    ```

  - The above command copies the public key component of the default identity key (which at the time of writing is ∼/.ssh/id_rsa.pub) and adds it to the target system's ∼/.ssh/authorized_keys file. To copy a key other than the default identity key:

    ```
    ssh-copy-id -i /path/to/identity_file.pub username@hostname
    ```

  - Once this step has been taken, and assuming the right key has been copied to the right location on the target system, it will no longer be necessary to enter a password to gain access.

- When the above procedure is complete, SSH utilities like `rsync`, `sftp` and `scp` will function without requiring the entry of a password for each transaction.

3.3 Summary

My present network setup, described above, is much more robust and requires much less babysitting than in years past. The Zeroconf protocol almost completely eliminates the effort required for static addressing and/or DHCP address reservation.

Online discussions of this approach sometimes make the point that Zeronconf doesn't scale well and isn't secure, and these appear to be legitimate criticisms. But for a small local network, this approach greatly simplifies life compared to its alternatives.

To summarize, Zeroconf has ended my earlier practice of detecting MAC addresses of newly acquired systems and peripherals, adding the MACs to a cross-reference list of MACs and statically assigned IP addresses, writing the list to my routers for DHCP address reservation and disseminating an updated `/etc/hosts` list to each system for name resolution. Instead, I only need to find out whether devices I acquire support Zeroconf. If they do, problem solved – I can simply attach the device to my network and it will auto-negotiate with DHCP and other Zeroconf-enabled devices on my LAN, and such devices become accessible in seconds. In the event that Zeroconf isn't available in a new device:

- Linux desktops and laptops either support Zeroconf by default or they can be set up in seconds:

  ```
  # apt-get install avahi-daemon avahi-utils avahi-dnsconfd avahi-discover libnss-mdns
  ```

- As explained above, Windows systems need only run the free executable `Bonjour64.msi` that's included with Apple's "Bonjour Print Services for Windows" installer[14].

- Android devices can run my free application SSHelper[13], which includes a Zeroconf server.

- Most of the new tiny computers like the Raspberry Pi[22] support Zeroconf out of the box, and if not, they can be easily configured.

For the few devices that don't support Zeroconf and that can't be programmed to support it, I assign static addresses and include them in `/etc/hosts`. And for such devices, I wait for the day they'll be updated with Zeroconf capability, so they can be removed from the `/etc/hosts` "dunce cap" list.*.

There's one more exception to a pure Zeroconf network. Any port forwards through my routers from the wide area network to the local one must be routed by way of a static local-network port and address, so the target machine must have a static (or DHCP reserved) address. This issue would be solved if my routers supported Zeroconf, but they don't (and there's no OpenWrt[23] version available for them, another option that's sometimes available).

Notwithstanding these small wrinkles, the Zeroconf scheme represents a huge improvement in robustness and simplicity over what came before. You might even say that a Zeroconf-enabled network exhibits a tiny bit of native intelligence.

Thanks for reading!

References

[1] Sneakernet – A slang term describing physical transfer of data using portable storage media.
[2] RS-232 – A widely used protocol for serial communications.
[3] Pipeline (Unix) – a simple but powerful way to communicate between processes.
[4] Packet radio – a peer-to-peer networking scheme that relies on a radio link.
[5] Amateur radio – a non-commercial hobby that involves transmitting as well as receiving radio messages.
[6] Confessions of a Long-Distance sailor – my free sailing book.
[7] Dynamic Host Configuration Protocol (DHCP) – a scheme to automate network address assignment.
[8] MAC address – a unique address provided to each network-capable peripheral device.
[9] DHCP : Overview – a bit about DHCP address reservation.
[10] Samba – an open-source embodiment of Microsoft's Server Message Block (SMB) network management method.
[11] Server Message Block (SMB) – Microsoft's network management method.
[12] Zero-configuration networking (Zeroconf) –a scheme to automatically configure a network.
[13] SSHelper – my free Android application that provides Zeroconf services.
[14] BonjourPSSetup.exe – Windows installation executable for Bonjour/Zeroconf.
[15] Bonjour Print Services for Windows – Webpage that explains about Bonjour/Zeroconf on the Apple support site.
[16] 7-zip – a Windows compression/decompression utility.
[17] WinRAR – another Windows compression/decompression utility, but not free.
[18] Secure Shell (SSH) – a reliable and very secure communications protocol in wide use.
[19] Network File System (NFS) – a network file management and transfer protocol.
[20] ssh-keygen(1) - Linux man page – always a good idea to read the manual.
[21] OpenSSH 7.0 disables ssh-dss keys by default – an online notice that "dsa" keys are being abandoned.
[22] Raspberry Pi – a hugely popular tiny computer running Linux.
[23] OpenWrt – A Linux distribution designed for wireless routers.

*My personal goal is to have an `/etc/hosts` file consisting entirely of "`127.0.0.1 localhost.localdomain localhost`"