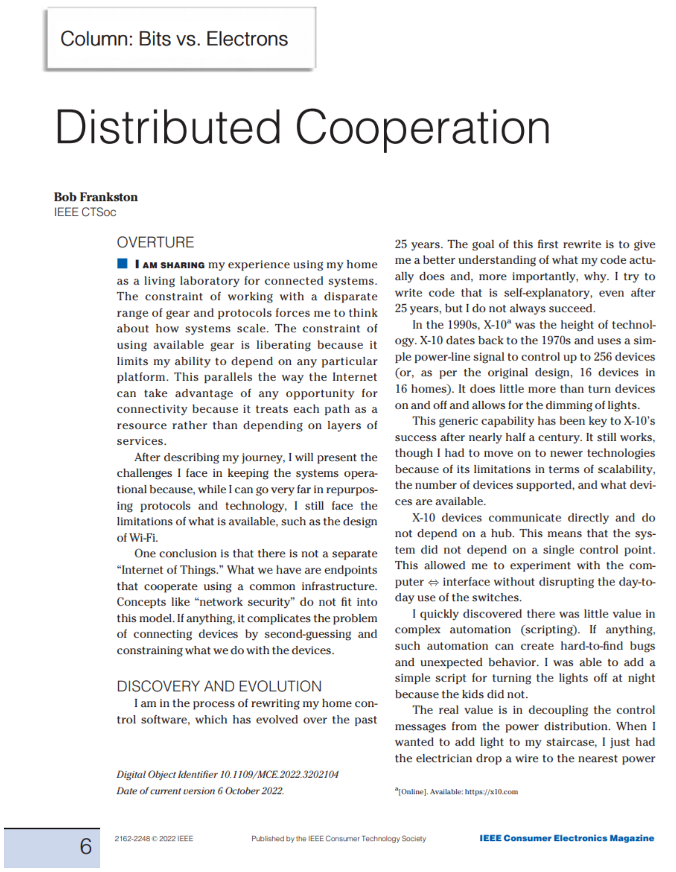
Distributed Cooperation


Column: Bits vs. Electrons

Distributed Cooperation

Bob Frankston
IEEE CTSoc

OVERTURE

I am sharing my experience using my home as a living laboratory for connected systems. The constraint of working with a disparate range of gear and protocols forces me to think about how systems scale. The constraint of using available gear is liberating because it limits my ability to depend on any particular platform. This parallels the way the Internet can take advantage of any opportunity for connectivity because it treats each path as a resource rather than depending on layers of services.

After describing my journey, I will present the challenges I face in keeping the systems operational because, while I can go very far in repurposing protocols and technology, I still face the limitations of what is available, such as the design of Wi-Fi.

One conclusion is that there is not a separate "Internet of Things." What we have are endpoints that cooperate using a common infrastructure. Concepts like "network security" do not fit into this model. If anything, it complicates the problem of connecting devices by second-guessing and constraining what we do with the devices.

DISCOVERY AND EVOLUTION

I am in the process of rewriting my home control software, which has evolved over the past 25 years. The goal of this first rewrite is to give me a better understanding of what my code actually does and, more importantly, why. I try to write code that is self-explanatory, even after 25 years, but I do not always succeed.

In the 1990s, X-10[a] was the height of technology. X-10 dates back to the 1970s and uses a simple power-line signal to control up to 256 devices (or, as per the original design, 16 devices in 16 homes). It does little more than turn devices on and off and allows for the dimming of lights.

This generic capability has been key to X-10's success after nearly half a century. It still works, though I had to move on to newer technologies because of its limitations in terms of scalability, the number of devices supported, and what devices are available.

X-10 devices communicate directly and do not depend on a hub. This means that the system did not depend on a single control point. This allowed me to experiment with the computer ⇔ interface without disrupting the day-to-day use of the switches.

I quickly discovered there was little value in complex automation (scripting). If anything, such automation can create hard-to-find bugs and unexpected behavior. I was able to add a simple script for turning the lights off at night because the kids did not.

The real value is in decoupling the control messages from the power distribution. When I wanted to add light to my staircase, I just had the electrician drop a wire to the nearest power

Digital Object Identifier 10.1109/MCE.2022.3202104
Date of current version 6 October 2022.

[a][Online]. Available: https://x10.com

Overture

I'm sharing my experience using my home as a living laboratory for connected systems. The constraint of working with a disparate range of gear and protocols forces me to think about how systems scale. The constraint of using available gear is liberating because it limits my ability to depend on any particular platform. This parallels the way the Internet can take advantage of any opportunity for connectivity because it treats each path as a resource rather than depending on layers of services.

After describing my journey, I will present the challenges I face in keeping the systems operational because, while I can go very far in repurposing protocols and technology, I still face the limitations of what is available such as the design of Wi-Fi.

One conclusion is that there isn't a separate "Internet of Things". What we have are endpoints that cooperate using a common infrastructure. Concepts like "network security" do not fit into this model. If anything, it complicates the problem of connecting devices by second-guessing and constraining what we do with the devices.

Discovery and Evolution

I'm in the process of rewriting my home control software, which has evolved over the past twenty-five years. The goal of this first rewrite is to give me a better understanding of what my code actually does and, more importantly, why. I try to write code that is self-explanatory, even after twenty-five years, but I don't always succeed.

In the 1990s, X-10[i] was the height of technology. X-10 dates back to the 1970s and uses a simple power-line signal to control up to 256 devices (or, as per the original design, 16 devices in 16 homes). It does little more than turn devices on and off and allows for the dimming of lights.

This generic capability has been key to X-10's success after nearly half a century. It still works, though I had to move on to newer technologies because of its limitations in terms of scalability, the number of devices supported, and what devices are available.

X-10 devices communicate directly and do not depend on a hub. This means that the system didn't depend on a single control point. This allowed me to experiment with the computer⇔interface without disrupting the day-to-day use of the switches.

I quickly discovered there was little value in complex automation (scripting). If anything, such automations can create hard-to-find bugs and unexpected behavior. I was able to add a simple script for turning the lights off at night because the kids didn't.

The real value is in decoupling the control messages from the power distribution. When I wanted to add a light to my staircase, I just had the electrician drop a wire to the nearest power source. No need for running a wire just to control the light (AKA, wired logic).

The fact that X-10 is slow, and the signal often got lost forced me to design the system to be resilient. My controllers show the pending state (request) separately from the actual state. I only considered a light as being on when I got a message back confirming the state. This is closed loop signaling.

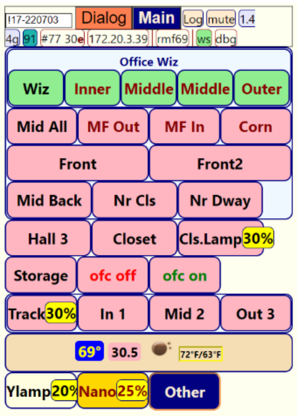
The limited capabilities bounded the problem I was trying to solve and allowed me to start simply. As I added new device families, I had to extend the design point. Over time, those extensions became increasingly creaky, hence the need for a rewrite. Fortunately, the programming tools have improved since I started to write the app using Visual Basic 6. I converted to Dotnet when it became available.

I'm now well beyond X-10, but some of the limitations of the original implementation remain. For example, I don't yet handle color. That's OK for now because, in day-to-day use, that isn't important, and, when necessary, I can use the apps for each family of devices.

The purpose of the initial rewrite is to give me a better understanding of the code. I did the rewrite in TypeScript, which is essentially JavaScript with type annotation. The annotation facilitates understanding the code and gives me the ability to refactor the code. The overall systems architecture consists of independent elements – programs and devices – which can cooperate. The asynchronous paradigm in JavaScript is far simpler than coordinating multiple threads in Dotnet implementation because I have to worry less about race conditions thanks to the cooperative single-threading.

The other key simplifier has been the use of JSON bundles which make it easy to share data among cooperating applications and, just as important, cooperating hosts but more about that later. I can use the same bundles whether the messages are sent via HTTP, WebSockets, MQTT, TCP, and even data files. I can also use generic tools such as Visual Studio Code, CURL, Postman, Firefox, and many more. Alexa also uses JSON

Note that I am saying "JSON bundle" rather than REST[ii] which has a narrower definition.



Using a common messaging format makes it easy to implement controllers as web apps. Unlike conventional light switches, they update to show the actual state of the light. I can also switch any of the surfaces to control other parts of the house and add features such as displaying the weather.

That said, there is also value in the tactile interface of physical switches. I use both.

It is important to reduce the points of failure. Thus, whenever feasible, I send messages directly to other devices. Shelly[iii] and Tasmota[iv] accept JSON messages, and each device has a built-in web interface. This is far better than needing a separate app for each family of devices.

When I can't go directly to a device, I implement shims so that the architecture is normalized to emulate direct connections. To improve system integrity, I run the shim app on multiple machines – both PCs and Raspberry Pi's so that there isn't a single point of failure.

One problem with resilience is that it is sometimes hard to know something is failing since everything seems to work. The system, as a whole, continues to function even when there are major problems, either in hardware or my software.

It's worth noting that this approach to resilience is what makes the Internet able to build reliable services on an unreliable infrastructure. There isn't one definition of reliable, so such problems can't be solved inside a network.

Rewriting the app has given me a chance to understand the code that has accumulated over the decades.

I refer to my style as scaffolding or kneading. At any point, the code is working and operational – a necessity since I'm controlling my house and my wife and family are living here. This imposes a strong discipline that assures I'm taking a pragmatic approach.

Relationships

When first implementing the system, I implemented a simple rule capability. If a device changes, then change the state of another device to match it or perform an action. It was meant to be a temporary implementation, but the very simplicity turned out to be a feature. It allowed the rules to be inspected. The rules implicitly defined relationships. Thus, a web app in a device could treat a set of devices as a group and not only turn on a set of lights – they could reverse the process and query the lights to get the average brightness.

I'm planning on implementing explicit support for managing relationships between devices.

The other aspect of relationship management – binding – also needs work. Today I might name a bulb ksink3, indicating it's the *3rd light on the sink-side of the kitchen*. That description should be part of the user interface, but the implementation should be more abstract since that bulb may have different roles. Some of those might be associated with the bulb, and some with the location. This shift in focus is an example of the benefit of learning by doing.

The lessons about managing relationships apply just as well to the global DNS. The lack of an abstract identifier for linking means we can have only one "John Smith" in the world. This creates an artificial scarcity of meaningful words, which is why the entire Internet unravels every year. This wouldn't be necessary if we separated the descriptive name from the linking name. It would allow me to put a URL in a device without worrying about the linkage going stale or, worse, doing something entirely unexpected when the name is repurposed!

Challenges

I have over 300 endpoints in my house with over 250 Wi-Fi devices. This gives me a chance to stress-test my approach and understand what scales and what does not.

Maintaining a very distributed system is challenging, especially for non-technical users. I ran into this recently when a light would sometimes turn on seemingly for no reason. When looking into an unrelated issue, I discovered I had left a test rule in place on HomeAssistant (HA) to turn on the light at dusk. We need better tools to diagnose such issues.

I've written[v] about how I leverage existing Internet protocols and tools. But the approach has its limits. Today's Internet works very well for accessing websites but not so much for peer connectivity among devices.

Wi-Fi is a wonderful enabler because it just carries packets without imposing its smarts, but there are few tools for solving the problems when it fails. Wi-Fi is still far better than alternatives such as Bluetooth and Zigbee, which do impose their smarts.

One limitation of Wi-Fi is that the security model is based on the idea of perimeter security. With wired networks, we assume that all devices on the same physical medium trust each other. The SSIDs are an attempt to emulate this border but create the need for a cumbersome onboarding process. When, as in my case, I have over 250 devices, the idea of changing the SSID for moving devices to a new network is daunting. It also makes it difficult to share devices across network boundaries. I can't simply give my neighbor permission to control a shared driveway light because we have different network boundaries.

I shouldn't need a separate app for each device family and shouldn't have to log in just to turn on a light. A far better approach is to build a web server into each device. A web server can be as simple as responding with a fixed string that gives the device name.

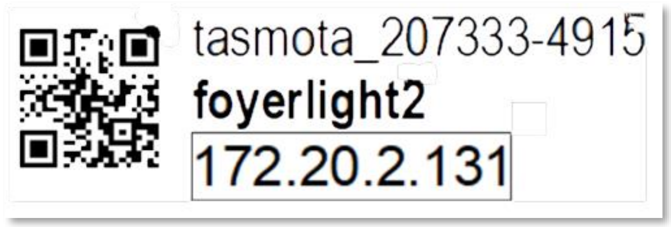
This is where the JSON bundles are valuable because they can be used to directly control a device via an HTTP message. The fixed string can also contain a client-side application that uses the full capability of the user's browser without using any resources on the device.

Having a web interface built into the device means I don't need an app per device. I can either use the built-in interface or write my own.

The ability to send messages directly from one device to another avoids depending upon hubs and other intermediaries, but it does require better tools for managing relationships. This is something I plan to work on in my next rewrite. Today I manually put the http messages in devices when I can. For webapps, I have to use intermediate shim programs because of the limitations imposed on Progressive Web Apps. PWAs are valuable because they can be installed and run even if my main server is down.

One challenge is managing certificates for HTTPS connections. Because many devices do not provide tools for managing credentials, I have to rely on perimeter security for now and use HTTP. That's OK for the short term since there is limited harm that can be done. I have to work with what I have.

To manage my devices, I carefully label each one. I generate my own labels. Some manufacturers do put serial numbers on each device which is very helpful. **Such labeling should be the norm and provide a description on port 80.**


tasmota_207333-4915
foyerlight2
172.20.2.131

Industry Structure and an IoT?

One takeaway from my experience is that the value is in the software applications more than in the hardware. Inexpensive "smarts" have commoditized basic hardware. An ESP8266 with Wi-Fi and Bluetooth can be purchased for $2. There may be value in a particular light fixture style, but that value is reduced if locked into a particular family of devices.

Today the nature of consumer technology has changed. This is evident in changing the name of the Consumer Electronics Society to the Consumer Technology Society. We've come a long way since the days when the CE Society split from the broadcast society, and consumer control was limited to turning dials.

The shift in value from the hardware to the software creates a dissonance in the market. This is why companies want to work with partners so they can capture the value of others' software. We see a similar challenge with network service providers, with 5G[vi] being an attempt to claw value back into the network.

These efforts work at cross-purposes with using the hardware as a resource. The web wouldn't have happened if we could make free use of the protocols and build on them. The industry needs to come to terms with the commoditization of hardware.

We need to shift from just providing solutions to providing the opportunity for innovation. Generic connectivity (as in a Public Packet Infrastructure[vii] is a key enabler. Such an infrastructure is independent of any particular device using that infrastructure. If anything, trying to build security into the network creates problems because it adds complexity, as in my example of the share driveway light.

A more subtle problem is the definition of security. The perimeter security model doesn't really provide security since any perimeter breach leaves all the devices and interfaces exposed. What we need is a nuanced understanding of trust relationships. These relationships aren't a property of the network. A radio or length of fiber doesn't know which bits are trustworthy or not. It can guess, but such guesswork at cross-purposes with new applications. When I read about cybersecurity, I often see the presumption of certain use cases or ways to use the devices.

Bad actors are everywhere, so it is important that my devices can survive attacks without hiding behind a castle wall – a secure perimeter. We're not quite there, so I do put up with some awkwardness using HTTP rather than HTTPS. We need better tools to manage certificates.

Looking Ahead.

I'm excited by the possibilities as we move to devices with web-based interfaces using JSON bundles. It is becoming increasingly easy to define direct relationships between devices so that a light switch (AKA controller) can turn on a light without having points of failure in the path. I'm writing about my experience to show that there is a market for devices with open interfaces. These can drive the future just as the gamers created today's market for high-performance computing that has given us today's rich interfaces on our portable computing devices (AKA smartphones).

In my previous column, I wrote about the Public Packet Infrastructure[viii] , which complements the commoditization of devices that enables us to deploy devices that don't just interconnect in our homes – they can be deployed anywhere in the world.

The term "Internet of Things" doesn't do justice to the new possibilities.

In the 1970s, some of us jumped ship from large computers to discover what was possible with personal computers, and we changed the world. Today we have an even more exciting opportunity. We are no longer limited to what we can do on a computer screen. The physical world is now programmable. We are at the very earliest stages of exploration. Strap in for the ride.

[i] https://x10.com
[ii] https://en.wikipedia.org/wiki/Representational_state_transfer
[iii] https://shelly.cloud
[iv] https://tasmota.github.io/docs/
[v] https://rmf.vc/IEEEBetaLife
[vi] https://rmf.vc/IEEE5GPast
[vii] https://rmf.vc/IEEEPPI
[viii] https://rmf.vc/IEEEPPI