RTC THE MAGAZINE OF RECORD FOR THE EMBEDDED COMPUTING INDUSTRY

Open Source and the Internet of Things: Roles, Reach and Rationale for Deploying OSS

Open source software will help drive the IoT build-out, but dominance in IoT technologies is not a foregone conclusion. Open source does indeed dominate large swaths of intelligent networking and Cloud platform software. For that to translate into IoT dominance, developer communities will have to cross key gaps and implement technologies essential to the Internet of Things.

BY BILL WEINBERG, BLACK DUCK SOFTWARE – JANUARY 2015

The build out of the Internet of Things is outpacing desktop and mobile computing. By 2020, over 50 billion intelligent devices (Cisco) will connect to and exchange information over the Internet with an economic impact of nearly US$2 trillion. This huge cohort of “things” comprises staggering diversity, from recognizable computers to infrastructure devices to sensors, light switches, and thermostats.

The impact will span the gamut of industries and applications – medical, agriculture, manufacturing, consumer electronics, transportation, and energy. Like the existing Internet, the emerging IoT will rely upon and instigate adoption of ofpe source software (OSS) technologies and open standards.

With the range of applications and constituents, divergent visions exist for building out and benefiting from the IoT. Some see the IoT as an incremental extension of existing computing technologies and methods (including open source); others herald the IoT as a revolution that will reinvent the IT industry and spur a major paradigm shift (Figure 1).

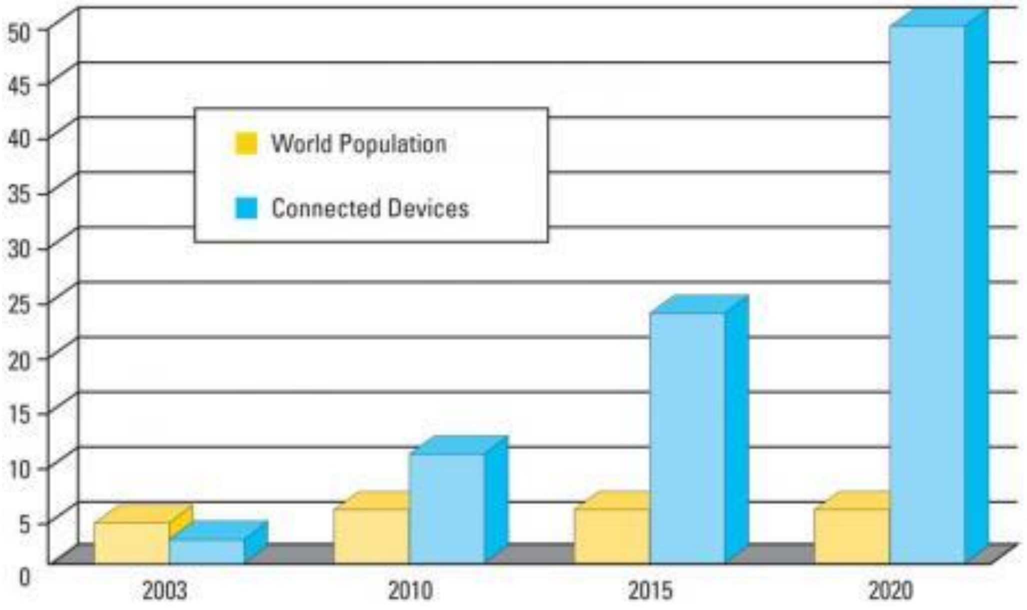

Figure 1 – Growth in Human Populations and Connected Devices through 2020 (Cisco)

This article explores the evolution of IoT technology, and ecosystem, and the role and reach of open source software in building and sustaining the IoT, from infrastructure to applications and other value-added device content. Specifically, it investigates how OSS can support competing and complementary architectures and meet looming IoT challenges.

Competing Visions for IoT

The IoT engenders excitement and inspires optimism about its potential. However, those prognostications will be subject to different, competing visions and paths for the actual build out. Monetization of the IoT, especially of open source software supporting that build out, will be subject to competing technical and financial models.

The two prevalent views of the architecture and make-up of the Internet of Things are Many Peers and Many Leaves.

Many Peers extends the current connected universe: the IoT comprises a network of “compute peers,” deployed with Linux, Android and comparable high-level OSes, running on 32- or 64-bit hardware, communicating over TCP/IP with applications running on a LAN or in the Cloud. Many Peers are just that, peers, but they do exist in a hierarchical context extending from the edge of the IoT to the Cloud.

Many Leaves envisions an extension of the machine to machine (M2M) paradigm – a vast collection of simple end-point systems, deployed with deeply embedded operating systems or no OS at all, running on a mix of 8-, 16-, and 32-bit hardware including RFID, communicating via specialized interconnects and protocols. Traffic from these systems traverses specialized gateways to reach local application servers (a.k.a. “the fog”) or travels onward to the Internet and up to the Cloud. These two visions are not incompatible and devices implementing both paradigms already populate the nascent IoT.

A stark distinction exists between two approaches to populating the different technical tiers of the IoT, with important implications for collaboration and interoperability—vertical integration vs. horizontal diversity.

Vertical Integration: There already exist silos of IoT devices and protocols. Providers of premises control equipment, home appliances, medical and e-health devices, and other gear are providing extensive, interoperating product lines with devices designed to function together and/or with Cloud and mobile apps. An example is home monitoring with network cameras and home automation equipment. A half-dozen vendors supply various lines of cameras, other sensors and actuators, along with Cloud and mobile apps to view and control them. In a mono-branded environment, vendors provide near-flawless out-of-the box and user experiences but do not interoperate with nearly identical devices from other vendors.

Horizontal Diversity: With a single IoT tier, there are less ambitious and more open suppliers. These companies offer fewer devices and device types but strive to interoperate with similar gear from other vendors (cameras with cameras, smart light switches with similar kit, etc.) and with third-party infrastructure devices as well. To achieve quality interoperability, vendors across tiers must implement using open IoT standards (e.g., MQTT) vs. enabling interoperation only with their own devices. They must also eschew the addition of “secret sauce” to differentiate their own wares and support brand affinity, and limit interoperability.

As exemplified in today’s web standards (HTTP, HTML, SOAP, etc.), the best way to foster interoperability is with open standards and shared, re-usable open source implementations. While open source is and will continue to be instrumental to the IoT, its presence and utility is not uniform across all elements of the network with its various node types and data paths (Figure 2).

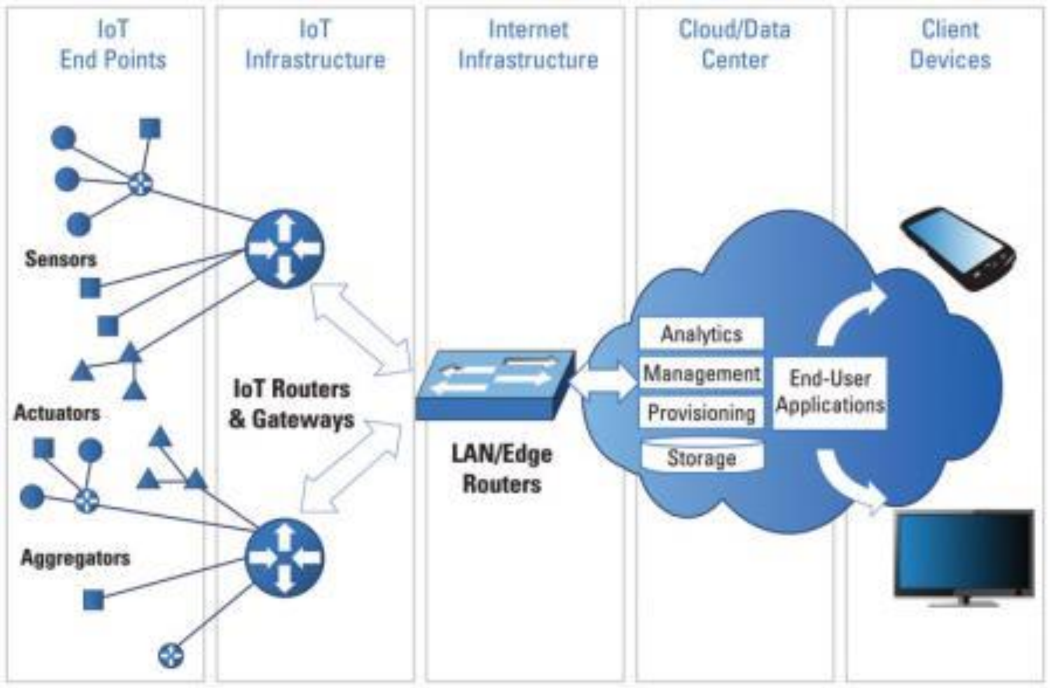

Figure 2 – IoT Node Types and Data Paths

End Points or Leaf Nodes

An area often overlooked in IoT discussion is the population of “dumb” devices – smart labels, inductive slugs and other RFID devices, used in manufacturing, inventory control, and elsewhere to track presence and location of objects and materials. These devices are stateless and passive, reporting only an ID and relatively small amounts of data when energized by scanning equipment.

The role of OSS in such passive devices lies not in the RFID tags and slugs themselves but in the equipment that activates them and in supporting the applications that act upon data they generate.

The “things” par excellence that comprise the leaf nodes of the prototypical IoT are simple end points—mono-function, ubiquitous, free-standing sensors and actuators. Imposing low power consumption and lower cost, these devices can be mostly stateless or highly stateful; they can be “headless” or boast device-local UI/UX functions; they can be completely independent or tightly networked with their peers; their communications can be terse or “chirpy,” and the data they transmit and receive can be slow-changing or highly dynamic. Think light switches and sockets, thermostats and HVAC controls, motion sensors and perimeter alarm switches, soil moisture and air temperature sensors.

The prototypical leaf node deploys fairly minimal software, supporting core functionality for sensing or affecting its environment and communicating state/status information upstream. Such devices may benefit from an actual embedded OS or just run a main loop and device service code. With minimalist 4/8/16-bit CPUs, they are unlikely to deploy a full OSI TCP/IP stack, instead employing point-to-point communications, mesh networking, 6LowPAN or partial IP networking (UDP, etc.).

The role of OSS in such devices is tactical or incidental. OEMs may choose a nominally open source RTOS (e.g., TinyOS, FreeRTOS) or a proprietary kernel to manage resources and simplify application programming. Developers will surely use OSS tools to create leaf node devices, and semiconductor suppliers will provide open source device drivers and other elements to support them, but the applications running on them will likely remain closed as are many other types of device software.

Device manufacturers insist that today and for the foreseeable future they need to retain all rights to differentiating technology in hardware and software vs. sharing development and maintenance responsibilities through collaborative development.

Peer-level leaf devices serve many of the same functions as simple end points, but with two key differences. They are better provisioned, with 32- or even 64-bit CPUs and additional RAM and they are more likely to bundle routing and/or gateway functionality into a single package. Additionally, they are by definition multi-function devices with the potential (or the necessity) of deploying enterprise-peer OSes – Linux, BSD, versions of Windows, etc.

These devices represent more interesting opportunities for OSS, from system software—especially Linux and Android—up through middleware and application frameworks, as well as routing software. The openness of value-added application software is subject to the same OEM IP constraints as on simple leaf nodes, with perceived disincentives to release such differentiating functionality as open source.

However, with fewer resource limitations and constraints on bills-of-material (BoM) costs, these types of devices are easier to build and accessible to "home brewing" and the creativity of the maker movement. We are already seeing a range of peer-type leaf nodes being implemented by hobbyists, researchers, and low-to-moderate volume integrators using readily available low-end commercial off-the-shelf (COTS) hardware - including versions of the Raspberry Pi, Arduino and BeagleBoard designs.

Infrastructure

At this level, the IoT still resembles machine-to-machine (M2M) networking. Mission-specific devices transmit context-dependent information across a point-to-point or mesh network, aggregated, buffered and conditioned by application-specific gateways and routers. In M2M, these devices communicate over a LAN to computers tasked with control, data analysis, etc. Today, they bridge and forward to the larger Internet and onward to Cloud servers.

These gateway devices deploy 32- or 64-bit CPUs and are provisioned with industrial network and serial connections – Zigbee, 6LowPAN, RS-422 and other specialized interconnects – as well as more familiar Wi-Fi, Bluetooth and Ethernet connectivity for both LAN- and WAN-ward communications. Depending on the number and variety of connected leaf devices, the "chattiness" of those devices, and the mix of public and private source and destination packets, IoT infrastructure devices may also log and buffer IoT traffic, time and space compress packets, and perform analysis and conditioning of the data in those packets before sending traffic upstream to the Cloud or downstream to local devices.

These nodes provide opportunities for OSS deployment and for the evolution of new OSS implementations: embedded Linux provides a flexible platform with native IP networking, IP routing software and standardized local file systems. New IoT frameworks are almost universally first hosted on Linux, as are most popular programming languages and tool kits.

The broader infrastructure of the Internet, from local wireless networks to broadband and mobile baseband access, to edge and core networking, is already teeming with open source software:

- Embedded Linux and Carrier Grade Linux in access points, routers, gateways, firewalls, media gateways, and other networking and telecommunications equipment
- Open source routing packages, security libraries, network management tool kits, high-availability enablers, and other network-related middleware
- BSDLite-derived TCP/IP stacks paired with proprietary embedded OSes
- Embedded web servers and web application components used to support configuration and management interfaces

The ongoing rollout of software defined networking (SDN) and Network Functions Virtualization (NFV) is also providing ample new opportunities for open source development to support Internet infrastructure.

As with Internet infrastructure, the Cloud is substantially built on OSS components – Linux, virtualization platforms, orchestration and management software, application support libraries and other Cloud middleware, and the tools and frameworks developers use to build and deploy code are all open source.

Not all Cloud software is open (e.g., Microsoft Azure), nor is software that implements infrastructure as a service (IaaS) and platform as a service (PaaS) readily available as open source, for example, the code behind Amazon Web Services or Rackspace Cloud Hosting. And, while code that implements IoT applications and IoT-centric software as a service (SaaS) solutions leverages OSS, there is no impetus for that code to be open source itself. Like Android, IoT platforms and tools derive from open source components and are themselves open, the majority of applications remain closed and proprietary.

End-User Software

End-user IoT software supports monitoring, control and configuration of IoT devices and analytics of the data generated by one or more IoT end point devices. These applications also provide domain-specific functionality relating to the functioning of one or more IoT devices. End-user IoT applications are typically manifested as web applications or mobile apps, but can really come in any form, such as parts of Big Data analytics packages.

As with the existing marketplaces of mobile apps and the even broader universe of web applications, IoT end-user apps certainly benefit from the existence of open source development tools and middleware, but have no particular impetus towards being open source themselves. Reasons include small audiences for niche apps unlikely to engender and support communities; mostly traditional per-unit business models; freeware with revenue from advertising or in-app purchases that don't accrue additional benefit from the "frictionless" distribution model of OSS; strong affinity with a particular brand/company that regards its end-user apps as conferring proprietary advantage. An overview of the likelihood of open source being used is given in Figure 3.

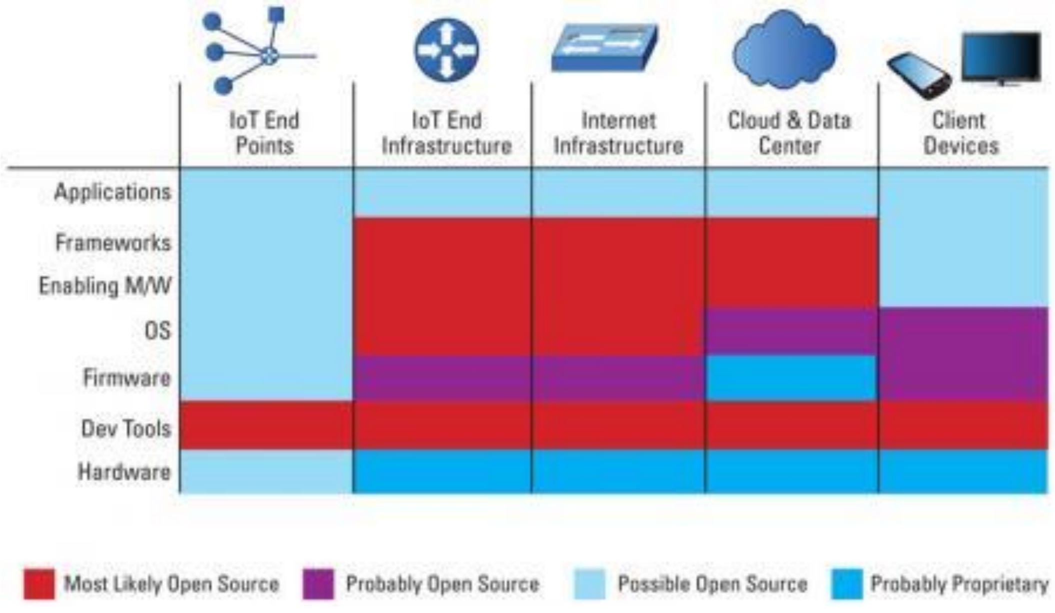

Figure 3 – IoT Open Source "Heat Map"

Meeting Key IoT Challenges with OSS

Having established the various roles of OSS in the ongoing IoT build-out, and substantiated the design share enjoyed today by embedded OSS platforms (as well as their likely future dominance), let's pause to consider how OSS can address some of the most daunting challenges facing the Internet of Things – security, privacy, and IP rights.

The history of OSS and security has been a roller coaster. First, a market accustomed to security-by-obscurity was slow to embrace community oversight as a means of tracking exploits and correcting the software defects that enable them. After years of debate, IT industry professionals finally came to appreciate the "many eyes" approach of OSS communities to detecting and addressing security risks and the essential requirement to keep software up-to-date. The low defect rates of OSS code were borne out by independent studies such as Coverity Scans. Then came the OpenSSL Heartbleed bug and the pendulum began to swing back, again casting critical eyes upon OSS security, even as community developers acted quickly to remedy the vulnerability.

Privacy-wise, OSS stepped up to enable protection of individuals' data by implementing strong encryption for the masses (SSL, SSH, PGP, etc.) and by supplying building blocks for mobile security and data protection—whether or not they are currently employed to great effect.

The IoT presents its own set of security and privacy challenges. For one thing, the myriad device types are built with greatly varying degrees of security expertise. There is a rich mix of public and private data with the potential to disrupt operation of industrial and energy systems, and of life-critical connected devices.

Device manufacturers historically relied upon security-by-obscurity (and by simplicity), but practices have been changing. OEMs face a combination of evolving pressures. These include the burgeoning attacks on devices and the networks to which they attach and the increasing enterprise-type software present on current generation devices. This has resulted in customer and market requirements for more secure device operation and better defense around information on them.

As with enterprise software, intelligent device designers depend on OSS, and dynamic updates to it, to meet the security threat landscape. These practices apply as well to peer-level IoT nodes. But mono-function and limited-function end-points face a different threat profile. Being out in the world, end-point devices are subject to physical attack. They can be commandeered or stolen, re-provisioned (re-flashed) with malware and re-deployed. Simple end-point devices still face vulnerabilities from poorly implemented interfaces and authentication, buffer overflow exploits, developer back doors, etc. IoT end-point devices, even if relatively secure, still present exposed serial, network, USB and other physical interfaces, and can also be "cooked" and otherwise abused to induce failure modes of interest to malefactors.

Beyond that, local wireless / mesh networks and can be spoofed and subject to man-in-the-middle attacks. Low-cost / high-volume devices are also likely to be deployed once and never updated, and so don't benefit from security updates – either by design or because low-bandwidth networking is not suited to deploying update images, especially across thousands of fielded devices.

None of the above is insurmountable. The most ubiquitous "things" on the Internet today are mobile phones and tablets, which stand out as a morass of security problems. Mobile OEMs, system software developers, network operators, application software vendors, IT departments, and end-users find themselves playing security "whack-a-mole", despite the efforts of the global developer communities around Android and other platforms both open and proprietary.

While important, open source is just one factor in a comprehensive IoT security and privacy paradigm. Equally important are best development practices and development tools to augment and enforce those practices.

What does make OSS more amenable to security remediation is its very openness. Beyond "many eyes making bugs shallow", readily available source code and published OSS project information (e.g., on GitHub and from Black Duck OpenHub) enable automation of otherwise tedious and technically-challenging activities. These include identifying security vulnerabilities, out-of-date versions, inactive or poorly maintained projects; assessing and remediating components with known vulnerabilities and monitoring IoT device and application codebases to ensure future security.