Building a Networked Appliance

John Sellens

jsellens@monbox.com

@jsellens

USENIX LISA 27, 2013

November 6, 2013

Notes PDF at http://www.monbox.com/notes/

A tale of designing and building a small networked computing appliance product.

Notes:

- Happy ending is still under development
- But so far so good

Not This:

I had an idea

for a thing . . .

And over time,

the idea changed . . .

And I learned a lot.

This is the story of the thing.

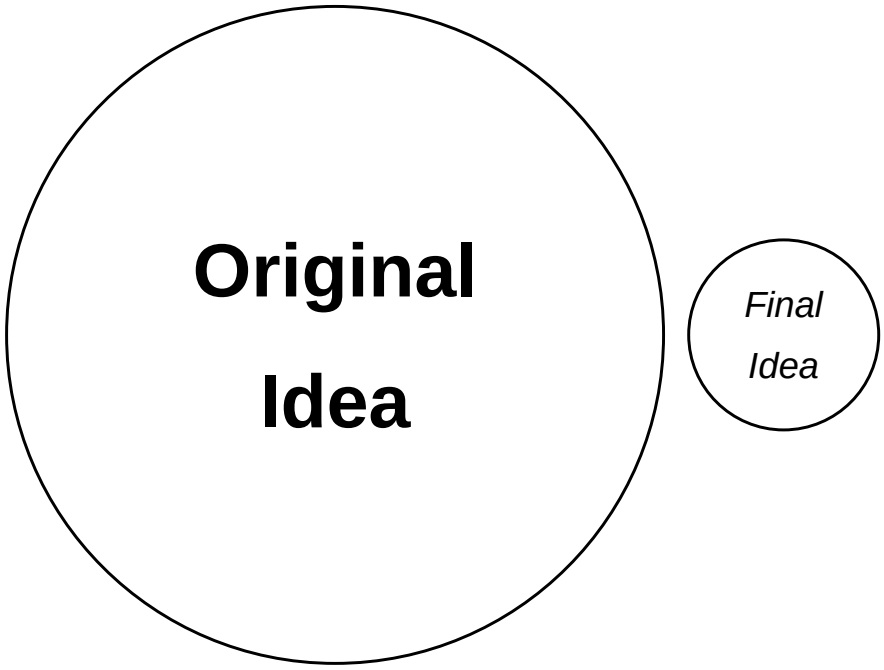
Notes:

- I’m not going to tell you much, if anything, about this particular thing
- This is meant to be a technical talk, and not a marketing talk
- I will of course be happy to tell you everything about this particular thing
  - Just as soon as I step away from the podium

What's the big idea?

Notes:

- A small, mostly self-contained network device
- Intended to be deployed in remote or hard to reach networks
- Central management server
- Usually no hands-on setup



Notes:

- I originally thought about and started working on a larger, more ambitious project/product
  - More singing, more dancing
  - Included a kitchen sink
- But I saw some others attempt that and either fail, or give up
- So I scaled back, and focused in

Narrowing the Focus

- A larger idea means
  - More effort, more time, more complication
  - Higher cost, harder to sell, more risk
- A more focused idea means
  - Quicker, less complicated, easier to explain
  - Lower cost, easier to sell, less risk
  - Easier to “pivot” and re-aim along the way

Notes:

- What can one person reasonably accomplish?
- In a reasonable timeframe?

Original Overview

- Management on the appliance
  - Web interface is primary
  - Basic console (or keypad) interface
  - No general purpose access
- Connect to the appliance, something happens, results returned

  User $\longrightarrow$ Appliance $\longrightarrow$ Target

  User $\longleftarrow$ Appliance $\hookleftarrow$
- Assumption that you can reach it on the network

Notes:

- No general purpose access i.e.
  - No typical shell access
  - No typical administration management
- I did have ideas of some remote control, but my assumption was that you could access the appliance itself for most management

Revised Inverted Approach

- But what if you can't reach it on the network?
  - Firewall, unreliable links, someone else's network?
- Most of the management would be remote
  - Appliance contacts mgmt server for direction
  - Reports back to user or via a proxy

    User $\longleftrightarrow$ Manager $\longleftrightarrow$ Appliance $\longleftrightarrow$ Target
- Appliance needs only outbound HTTPS
- Often don't have to connect to the appliance itself

Notes:

- Of course, there's still management on the appliance itself
  - If you can get to it
  - Or for initial configuration
- Idea being just plug it in where you need it, then it configures, operates from there

Special Purpose, General Purpose

- Building a special purpose appliance
- Using general purpose hardware and OS
- Need to
  - Choose hardware platform
  - Choose OS platform
- What were the choices?
- **Motivations: cost effective, reliable**

Notes:

- Custom hardware and custom OS would be expensive and time consuming
- Confession: the hardware platform blocked me for a while

Generic Intel box

Notes:

- Generic rackmount Intel server
  - This one from General Technics gtweb.net
  - Real disk, modem, etc.
- Add a half height bay mounted LCD and keypad
  - From Matrix Orbital www.matrixorbital.com
- A generic server was more feasible with the large “Original Idea”

Cobalt Raq3

Notes:

- Cobalt Raq3 web hosting appliance
  - Refurbished
  - Add a little memory
  - And an IDE compact flash interface and card
- Older AMD, LCD support requires older Fedora 5 Linux
- Serial console only, no video
- Under $200, but harder and harder to find

Mini-ITX System

Notes:

- Mini-ITX system from mini-box.com
- PicoLCD
- Compact flash slot, or disk
- VGA and keyboard/mouse
- More than 1 rack U high
- A weird size and shape
  - Too big for a module, doesn't fit in a rack
- Under $300

Embedded Systems

Notes:

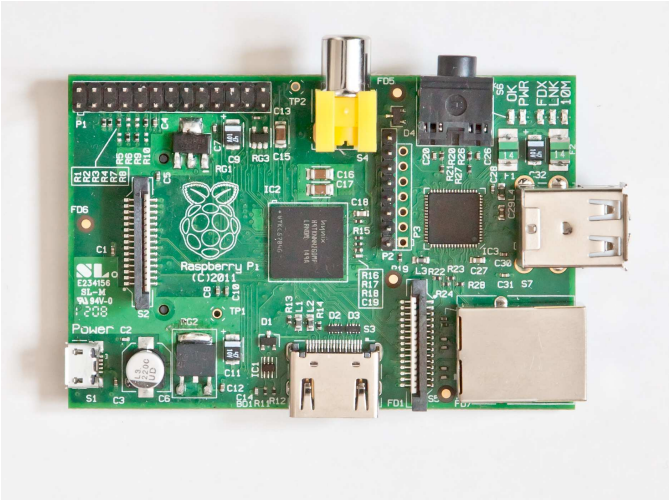
- This is a Soekris net4801 board — 256MB, 233MHz processor, about $200 — soekris.com
- Standard embedded systems seem to be too limited, and at the same time, relatively more expensive

Plug Computer

Notes:

- Marvell Sheevaplug plug computer
  http://www.marvell.com/solutions/plug-computers/
- ARM processor, 512MB or 1GB flash and RAM
- USB serial console, no display
- $100, other models for more
- Other plug computers are available, with similar specs

Raspberry Pi Model B



"Well, duh"

Notes:

- ARM processor
- 512MB RAM, SD card
- USB, HDMI, audio
- Hardware serial number in `/proc/cpuinfo`
- Large, enthusiastic and active community
  - Lots of hobbyist interest
- $35, plus case, power, SD card
- First available spring 2012
- "This, for the Piranha brothers, was the turning point."

Choice of Operating System

- Influenced by other examples
  - m0n0wall, pfsense, . . .
- Choice of hardware can restrict OS choice
- FreeBSD has NanoBSD
  - Designed for flash memory
- Fedora and friends have some read only root support
- Debian and friends have . . .

Notes:

- I'm not crazy about limited choices
- I didn't really look at embedded or special purpose operating systems

I Took the Easy Way Out

- Raspbian — the “standard” Debian variant for the RPi
  - Supports the hardware, good infrastructure
  - Compiled for hardware floating point
  - Available relatively early on
- There are alternatives — Linux, FreeBSD
  - With varying levels of support and completeness

Notes:

- Using the stock Raspbian is not without some challenges, as we shall see

But Not Without Previous Steps

- Before the Raspberry Pi was available
- Freesbie, NanoBSD experiements
- Implemented on Raq3 with Fedora 5
  - Been running one at home for about 3 years . . .
  - Easy recipe for read only root filesystem
- Nice base for “portability”

Notes:

- FYI I’m a FreeBSD fanboy from way back

Wrapping Packages

- Makes sense to build software in packages
  - Need package infrastructure for rebuilt packages
  - May as well use it for everything
- RPM vs DEB — I don't believe the hype
  - "I hate it, but I use it"
  - DEB — more convoluted, harder to build, and put in a repository
- I use EPM to generate `.rpm` and `.deb` from common configs

Notes:

- NanoBSD is monolithic — updates are via complete new image
- EPM has some quirks
  - http://www.epmhome.org/
- These days FPM is likely a better choice for package building

Respositories and Updates

- With RPM, creating a repository is
  - File in `yum.repos.d` with URL
  - Add/remove `.rpm` files, run `createrepo`
  - Update with `yum -y update`
- With DEB, creating a repository is
  - File in `apt/sources.list.d` with URL
  - Directory hierarchy, `reprepro` to remove and add each `.deb` in turn
  - Update with `apt-get -y upgrade` or `dselect-upgrade` or `dist-upgrade`

Notes:

- Plus package signing with GPG
- It just seemed to me that RPM had one way to create a respository and do an update, that was fairly obvious
- While DEB has many different tools and methods, a complicated structure, convoluted behaviour, and none of it is very straightforward
- Apparently, I'm not very forward thinking

Software Basics

- Configuration, management, maintenance tools
  - Non-interactive command line tools
  - Console menu mechanism for basic setup
  - Web interface for more detailed config
- Much of the base functionality uses standard tools
  - SSH, lighttpd, domain-specific packages, etc.

Software Toolbox

- I like shell scripts and simple config files
- Simple shell and PHP libraries
- Configuration tools — setconfig, saveconfig
  - source-able config files for shell and PHP
  - Static `machine.cfg` identifies this device
- Periodic and boot processing tools

Notes:

- Theory is that many of these tools could be applied to different devices
  - Code reuse for the win!

Which Came First?

- Shell/cli/toolkit or web?
- Deliberate: command line tools first
  - Basic functionality first
  - Pretty wrapper last
- Which means tools can be used by
  - Web interface
  - Periodic or boot processing
  - API and provisioning tools

Notes:

- Tried to follow the “UNIX philosophy”
  - Lots of small self-contained tools

If a machine crashes in a remote forest, does `fsck` make a sound?

Don’t Kill The Disk

- I didn’t re-implement the read-only root filesystem on the Raspberry Pi
  - Until that fateful day . . .
- tmpfs memory disk and bind mounts
  - Writeable copies of files/directories
- And a set of read-only/read-write tools

Notes:

- Underlying mechanism based on work derived from the Stateless Linux project
- I learned: if you restart a service using a binary on a read-write filesystem, you can’t remount back to read-only
- Convenient side effect: less chance of wearing out the SD card

Local Management and Control

- Local console runs a simple menu
  - Basic system and network configuration
  - Update, reboot, status, shell access
- If you have network access to the device
  - SSH can be enabled for remote menu access
  - Local web interface for config and status
- Default read-only root means breaking it is harder
  - But not impossible . . .

Remote Management and Control

- Central management server can request changes
- Device calls in every 15 minutes, gets commands
  - Set of scripts processes a small set of commands
  - Work to do, password change, trigger updates, . . .
- Reports in as work is done and hourly summary
  - “Call home” for status and problem solving

Notes:

- Remote control and reporting can be disabled
  - But I like to think it’s very useful
  - And low risk for most

Security and Reliability

- With console access, you need no password
  - Why worry? SD card is removeable
- Over the network, SSH, SSL
- Management access allows enabling SSH
  - Assumes you're in a "reasonable" environment
- Little protection against deliberate code modification

Notes:

- Custom code is all shell or PHP
- Might transition to compiled, signed, obfuscated code
  - Harder to break
  - Harder to copy
- Contrast with the Busybox example
  - One binary, multiple names

Development and Testing

- Non-monolithic tools make things easier
  - No long build stage
- Some can be tested on dev workstation
- Virtualization — ARM not Intel
  - Will run under qemu (with minor mods)
- Test devices can use test software
  - Internal software repository for test packages
  - Local mgmt web can be NFS client mounted

Notes:

- qemu is convenient
  - A little quirkier than more mainstream virtualization mechanisms
  - But can emulate non-PC hardware
- SD cards or images can be mounted on other machines
  - For repair or examination

Provision and Deploy

- Each device is assigned a name, for dynamic DNS
- Device initial build is mostly automated
  - Standard SD card image, with initial boot script
  - Device boots, contacts internal config server
  - Downloads per-machine config by serial number
  - Initial config tasks, generates random password
  - Prints instructions for customer
- Boot server registers device in mgmt database

Notes:

- Dynamic DNS is updated at boot, and in hourly “call home”
- The theory is a name is easier to deal with than a serial number
- Management database includes name, MAC, (Raspberry Pi) serial number
- New devices are assinged to an “inventory” account
  - Moved to a customer account when sold

Central Management Service

- Management server is key to working in isolation
- User connects SSL to management, as does device
- Summary information, call home data
- Change password, request update, reboot
- Configure tasks for device to retrieve

Notes:

- Various account management tools
- Overview of all devices

Management Service API

- User API access to management services
- Configure tasks, retrieve results, trigger actions
- Allow integration with existing business processes
- SSL, secret token, simple syntax
  - No XML, JSON, YAML, . . .

Notes:

- Intention is to require little or no manual or interactive configuration

Keeping Me Up Nights

- What happens if software attacks?
  - Software update fails badly?
  - Stock Raspbian update fails badly?
- Loses network connectivity and doesn't recover?
  - Especially with WiFi
- SD card failure?
- Raspberry Pi failure?

Notes:

- All these can be mitigated of course
- But remote, unattended devices are a little nerve-wracking

What's Next?

- Planned enhancements
  - Features, management, reliability
- Additional tools for integration

And . . .

- Curious? I can talk all day . . .
  - Hallway track, mail, . . .
- Remember: Technical specs do not a product make
  - Ignored: parts acquisition, assembly, ship, sales, marketing, administration, . . .
- Questions?