

A wee server for the home

Sudarshan S. Chawathe

2018-03-24

Home server: what? why?

- Something to provide small-scale local services
 - Printing from local network
 - File server
 - Easily and privately share files with household
 - Destination for backups of other computers, photos, videos
 - Music server
 - Control playback on attached home audio system
 - Serve music to play elsewhere
 - Stream music from elsewhere
 - Web server: Photo and video galleries
 - Personal XMPP/Jabber chat server
 - Landing spot for remote login
 - Wake up other computers using wake-on-LAN.
 - Email server, ... ?
- Under personal control.
 - Free (libre)
 - Independent of non-local network
 - availability, latency, bandwidth

Why a *wee* server?

- Low power consumption
 - Always-on is a nice if it only uses a few watts.
 - Low heat dissipation
- Compact
 - easily stash on a shelf, behind other equipment, ...
- Low cost
 - ~ 100 USD.
- Hardware options that are more open
 - than mainstream servers
- Fun
 - low-risk hardware experimentation: flashing, etc.
 - easy hardware interfacing
 - blinking lights, motors, sensors, ...

This presentation

- For, and by, a non-expert
 - Not very novel or unique; see FreedomBox, ...
 - Expert advice welcome
- Brief how-to and invitation
 - Buy, build, configure a wee home server
 - Use, learn, and contribute to libre software
 - One person's choices and experience
 - *not* comprehensive, nor ideal
 - but actually used, long term
- Small technical excursions (still non-expert)
 - udev rules
 - randomness
- Sharing
 - experiences with home servers
 - suggestions, concerns, future directions

Hardware choices

- many options
- examples, not exhaustive lists
- what I chose and why

Hardware: Main server/single-board computer

- Desired
 - Open/libre hardware
 - Low power consumption (and heat output)
 - Low cost
 - Sufficient compute resources
 - CPU, RAM, display, network, USB, ...
 - for? priorities? video? disk-network throughput? ...
- Contenders: *non-exhaustive*
 - BeagleBone Black (BBB) [my choice]
 - easily available in US (+ from my favorite retailer)
 - Debian GNU/Linux pre-installed option
 - very competitive cost: ~ 55 USD
 - lots of ports, accessible hardware pins (for other hardware fun)
 - Cubieboard
 - Olimex A20
- Tempting but no-go:
 - Raspberry Pi ~ 35 USD
 - octa-core “TV boxes” ~ 80 USD

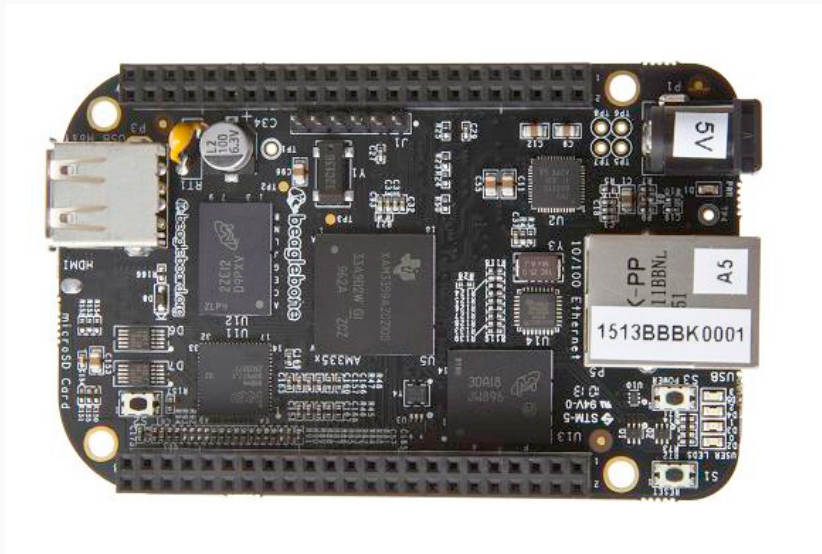
Hardware: Essential accoutrements

- Case
 - Not just vanity
 - Consider byzantine mobile hardware fault injectors (cats).
- Good power supply
 - Don't skimp; power "brown-outs" will cause much pain.
- USB hub(s)
 - powered better though add clutter.
 - need to separate hi- and lower-speed devices?
- Cables
 - (micro-)USB
 - Ethernet (Cat 5e, 6)
 - (?) (micro-)HDMI to HDMI/DVI/VGA
- (?) Keyboard, mouse

Hardware: Peripherals

- Main (OS, etc.) storage: small, fast disk.
 - Can use on-board 4 GB flash in BBB.
- File-server: external USB disk
 - “portable”: single attachment, smaller, usually quieter, cooler, lower power draw.
 - or “desktop”: slightly lower cost, no power drain on USB, larger, noisier, warmer, bit more clutter.
- Print server: (existing?) USB or network printer.
 - USB seems more predictable.
- Music server (to home audio): USB sound card.
 - digital audio output is nice.
 - generic CM106-based one seems fine: not audiophile
 - BBB can do HDMI audio (untested)

BeagleBone Black (BBB)



BeagleBone Black specs

- From <https://beagleboard.org/black>
- Processor: AM335x 1GHz ARM Cortex-A8
 - 512MB DDR3 RAM
 - 4GB 8-bit eMMC on-board flash storage
 - 3D graphics accelerator
 - NEON floating-point accelerator
 - 2x PRU 32-bit microcontrollers
- Connectivity
 - USB client for power & communications
 - USB host
 - Ethernet
 - HDMI
 - 2x 46 pin headers

Setting up a wee server using BBB

- what I did (why)
- what I learned
- did it work?
- was it fun?

Assemble hardware

- BeagleBoard Black Rev C.
 - Element 14 edition: Debian (Jessie, c. late 2017) preinstalled.
 - We'll update to Debian Stretch right away...
- Snap fit case
 - leave top off for now, to access boot button
- Connect USB cable to gadget port.
 - Don't plug in yet
- Separate (barrel connector) power supply
 - Don't power up yet

Prepare micro-SD card with firmware update

- Instructions from <https://beagleboard.org/getting-started>
- Get `bone-debian-9.3-iot-armhf-*-*-*-4gb.img.xz`
- Checksum

- Create file `sha5sums` with checksum from above site:

```
33fc557f32005c811bd449a59264da6b4a9b4ea9f87a1ee0aa43ae651c7f33d1
bone-debian-9.3-iot-armhf-2018-03-05-4gb.img.xz
```

- `sha256sum -c sha256sums`
- Write image to spare micro-SD card:
 - Previous contents obliterated

```
xzcat bone-debian-9.3-iot-armhf-2018-03-05-4gb.img.xz | dd
of=/dev/F00
```

- Or with some visualization

```
pv -cN rd < bone-debian-9.3-iot-armhf-2018-03-05-4gb.img.xz | \
xzcat | pv -cN xzcat | dd of=/dev/F00
```

Edit firmware on microSD card

- In order to flash BBB's eMMC storage
 - This feature not turned on by default
- Mount just-written microSD card on host computer

```
mount /dev/F001 /mnt/t
```

- Edit `/mnt/t/boot/uEnv.txt` carefully:
 - Uncomment (remove the #-prefix) line that has:

```
cmdline=init=/opt/scripts/tools/eMMC/init-eMMC-flasher-v3.sh
```

- unmount and remove microSD card

```
umount /mnt/t
```

Update firmware

- Insert prepared micro-SD card into BBB's slot.
- Prepare to power the board using its barrel-connector and separate power source
 - not via USB, since current requirements during flashing may exceed USB limits
 - (true?—good advice nevertheless)
- Get BBB to boot from microSD
 - Normally, BBB boots from eMMC
 - Was oddly hard to find how:
 - From https://elinux.org/Beagleboard:Updating_The_Software
 - Hold switch S2 (Boot Switch, [diagonally across from barrel conn.]) down by pressing on it and holding it while plugging in the power cable.
 - Continue to hold the button until the first User LED comes on.
 - The board will start flashing the eMMC and the LEDs will flash to show activity. It will take about 45 minutes to flash the board.
 - It took only about 10 minutes.
- Can also flash eMMC from BBB command-line after boot.
 - untested

Connect BBB as USB gadget


- No need for the barrel power connector now.
- Plug host end of USB cable to setup computer
- Three results (plus blinking lights):
 - USB disk `/dev/sdb` or similar
 - with 17 MB vfat partition `/dev/sdb1` named "BEAGLEBONE"
 - Serial port terminal on `/dev/ttyACM0` or similar
 - working user-id needs `dialout` group membership for permissions
 - Ethernet (over USB) device: `eth1` or similar
 - permissions via Gnome/network-manager, ...?
- But, didn't quite work as above for me

Browse BBB on-board docs

- Got BEAGLEBONE partition mounted.
 - Can browse files, view HTML ones in browser, etc.

Index of file:///media/chaw/BEAGLEBONE/

[Up to higher level directory](#)

Name	Size	Last Modified	
 App		05/25/2017	05:16:32 PM
 Docs		05/25/2017	05:16:32 PM
 Drivers		05/25/2017	05:16:32 PM
 LICENSE.txt	41 KB	05/25/2017	05:16:32 PM
File: README.htm	31 KB	05/25/2017	05:16:32 PM
 README.md	1 KB	05/25/2017	05:16:32 PM
File: START.htm	31 KB	05/25/2017	05:16:32 PM
File: autorun.inf	1 KB	05/25/2017	05:16:32 PM
 scripts		05/25/2017	05:16:32 PM

On-board BBB HTML docs

Linux

`mkudevrule.sh`

Driver installation isn't required, but you might find a few udev rules helpful.

Note: Additional FTDI USB to serial/JTAG information and drivers are available from www.ftdichip.com/Drivers/VCP.htm.

Note: Additional USB to virtual Ethernet information and drivers are available from www.linux-usb.org/gadgets/joshuawise.com/horndis.

Step 3

Browse to your Beagle

Using either [Chrome](#) or [Firefox](#) (Internet Explorer will **NOT** work), browse to the web server running on your Beagle. You will load a presentation showing you the capabilities of the board. Use the arrow keys on your keyboard to navigate the presentation.

- ▶ Click here to launch: <http://192.168.7.2>
Older software images require you to EJECT the BEAGLE_BONE drive to start the network. With the latest software image, that step is not required.

Install udev rules using script

- What does `mkudevrule.sh` do?

```
cat > /etc/udev/rules.d/73-beaglebone.rules <<EOF
ACTION=="add", SUBSYSTEM=="usb", ENV{DEVTYPE}=="usb_interface", \
    ATTRS{idVendor}=="0403", ATTRS{idProduct}=="a6d0", \
    DRIVER=="", RUN+="/sbin/modprobe_b_uftdi_sio"

ACTION=="add", SUBSYSTEM=="drivers", \
    ENV{DEVPATH}=="/bus/usb-serial/drivers/ftdi_sio", \
    ATTR{new_id}="0403_a6d0"

ACTION=="add", KERNEL=="ttyUSB*", \
    ATTRS{interface}=="BeagleBone", \
    ATTRS{bInterfaceNumber}=="00", \
    SYMLINK+="beaglebone-jtag"

ACTION=="add", KERNEL=="ttyUSB*", \
    ATTRS{interface}=="BeagleBone", \
    ATTRS{bInterfaceNumber}=="01", \
    SYMLINK+="beaglebone-serial"
EOF

sudo udevadm control --reload-rules
```

Try Ethernet-over-USB to BBB

- Gnome/NetworkManager tries to bring up `eth1` but stalls at “Connecting...”

- Peek in `/var/log/syslog`

```
dhclient[22752]: DHCPDISCOVER on eth1 to 255.255.255.255 port 67
interval 13
dhclient[22752]: DHCPDISCOVER on eth1 to 255.255.255.255 port 67
interval 21
dhclient[22752]: DHCPDISCOVER on eth1 to 255.255.255.255 port 67
interval 17
NetworkManager[27515]: <warn> [1521786722.8343] dhcp4 (eth1):
request timed out
```

- Try manual `eth1` setup with address `192.168.7.1`
 - No response to HTTP request to `192.168.7.2`
 - No response to ping either.

Try serial port (USB) connection to BBB

- Noticed `syslog` messages indicating new serial port `/dev/ttyACM0`
- Try getting serial console.
 - `minicom -s`
 - Port `/dev/ttyACM0`, 115200 baud, 8N1
 - no responses

Try console on HDMI display port?

- Another way to get to console:
 - connect monitor using micro-HDMI
 - and USB keyboard, mouse
- But I was wanted the other method to work...
 - It all seemed to be set up so nicely.
 - Must be something small that's awry.
 - [This ate up a lot of time!]

Fix udev rules

- `lsusb`

```
Bus 001 Device 015: ID 1d6b:0104 Linux Foundation Multifunction  
Composite Gadget
```

- But `mkudevrule.sh` has

```
ATTRS{idVendor}=="0403", ATTRS{idProduct}=="a6d0",
```

- Fix above in installed rules:

```
/etc/udev/rules.d/73-beaglebone.rules
```

- Some sources on the Web suggesting ModemManager may be the culprit.

- Could try uninstalling it

- Nicer to have it ignore the BBB by adding `udev` rule:

```
ATTRS{idVendor}=="1d6b", ATTRS{idProduct}=="0104",  
ENV{ID_MM_DEVICE_IGNORE}="1"
```

- Unplug-replug BBB and...

BBB communication success

- Web server at 192.168.7.2.
 - Serves set-up docs (same as BEAGLEBONE partition)
 - Interaction with BBB using live BoneScript

BoneScript interactive guide

BoneScript is a JavaScript library to simplify learning Linux. This web page is able to interact with your board

Example

run

restore

```
1 var b = require('bonescript');
2 b.pinMode('USR0', b.OUTPUT);
3 b.pinMode('USR1', b.OUTPUT);
4 b.pinMode('USR2', b.OUTPUT);
5 b.pinMode('USR3', b.OUTPUT);
6 b.digitalWrite('USR0', b.HIGH);
7 b.digitalWrite('USR1', b.HIGH);
8 b.digitalWrite('USR2', b.HIGH);
9 b.digitalWrite('USR3', b.HIGH);
10 setTimeout(restore, 2000);
11
```

- Serial console on `/dev/ttyACM0`
 - Login prompt states username and password.
 - Log in and change password
 - Now we have a mostly standard Debian setup; smooth sailing.

Storage

- Connect “portable” USB disk (or other choice)
- Check device assignment.
 - `/var/log/syslog`

- Verify size

```
blockdev --getsize64 /dev/sda
```

- Mount to check manufacturer-provided stuff

```
fdisk -l /dev/sda  
mkdir /mnt/t  
mount /dev/sda1 /mnt/t  
ls -lR /mnt/t  
umount /mnt/t
```

- Check SMART data

```
smartctl -d sat -a /dev/sda  
smartctl -t short -a /dev/sda
```

- Test a bit (obliterate contents of `/dev/FOO`)

```
badblocks -vsw /dev/FOO
```

- takes a long, long time for large devices

Prepare for filesystem

- encrypted(?)
- write random data (obliterating contents again)

```
dd if=/dev/urandom of=/dev/F00 bs=4096  
DDPID=$!
```

- to check progress

```
kill -s USR1 $DDPID
```

- Alternatives
 - /dev/random? (no) openssl? (maybe)
 - later

Create encrypted partition and filesystem

- cryptsetup (obliterate /dev/FOO again)

```
cryptsetup luksFormat /dev/F001  
cryptsetup luksOpen /dev/F001 BAR  
ls -l /dev/mapper/BAR
```

- ext4

```
mkfs.ext4 -L BAZ /dev/mapper/BAR  
mount /dev/mapper/BAR /mnt/t
```

- before disconnecting

```
umount /mnt/t  
mount  
cryptsetup luksClose BAR
```

Using filesystem remotely

- scp
- sshfs
- rsync
- Unison
- syncthing
- git
- NFS
- OpenAFS

My storage use

- ssh, scp, sshfs as baseline
 - manual syncing works fine for many cases (one master copy)
- rsync for bigger jobs
 - photos, videos, user-level backups
- unison for more complex (bidirectional) syncing
 - still need some higher-level policy to avoid crazy conflicts
- limited use of syncthing mainly for Android phones
 - slurp photos and videos off phone
 - dump reading material and music, videos onto phone
 - hopefully can use unison soon.
 - untrusted device: (semi-)public material only

Deeper explorations

- udev rules
 - tweaking was needed for communicating with BBB.
 - quite handy for setting device permissions
 - a lot seems to now be already set up
- sources of random data
 - for initializing disks before setting up encrypted partitions
 - options
 - “quality”
 - speed

udev rules

- userspace rule-based system to manage device nodes in `/dev`
- event-conditions-actions rules
 - event: device addition, removal, change (from kernel)
 - e.g., a USB WiFi device detected.
 - ACTION=="add"
 - conditions: (key op value), ... [logically anded]
 - e.g., ATTRS{idVendor}=="01ac", ATTRS{idProduct}=="b2d1"
 - actions: (key assign value), ... [sequence of assignments]
 - e.g., SYMLINK="/dev/usbwif1"
- all rules matching an event are executed
 - event is not "consumed" by rule execution

udev rules files

- Files `*.rules` in
 - `/lib/udev/rules.d/`
 - `/run/udev/rules.d/`
 - `/etc/udev/rules.d/`
- Files from all three pooled and evaluated in lexical order
- Exact file-name matches: one from earlier directory ignored

udev rule matching

- Several match (condition) keys:
 - ACTION, NAME, ATTR, ...
- and assignable (action) keys:
 - NAME, SYMLINK, OWNER, MODE, ...
 - shell-like globbing
 - *, ?, [afk], |
- devpath: device path
 - ATTR v. ATTRS, etc.
- keys from external programs
 - condition: PROGRAM, RESULT
 - action: RUN{program}, RUN{builtin}
- IMPORT{program}, IMPORT{cmdline}, ...
- LABEL and GOTO
- string-substitutions

udev example: included rules file

```
/lib/udev/rules.d/73-usb-net-by-mac.rules
```

```
# Use MAC based names for network interfaces which are directly or
# indirectly on USB and have an universally administered (stable) MAC
# address (second bit is 0). Don't do this when ifnames is disabled
# via kernel command line or customizing/disabling 99-default.link (or
# previously 80-net-setup-link.rules).
```

```
IMPORT{cmdline}="net.ifnames"
```

```
ENV{net.ifnames}=="0", GOTO="usb_net_by_mac_end"
```

```
ACTION=="add", SUBSYSTEM=="net", SUBSYSTEMS=="usb", NAME=="", \
  ATTR{address}=="?[014589cd]:*", \
  TEST!="etc/udev/rules.d/80-net-setup-link.rules", \
  TEST!="etc/systemd/network/99-default.link", \
  IMPORT{builtin}="net_id", NAME="$env{ID_NET_NAME_MAC}"
```

```
LABEL="usb_net_by_mac_end"
```

udev example: modified setup for BBB

```
/etc/udev/rules.d/73-beaglebone.rules
```

```
ACTION=="add", SUBSYSTEM=="usb", ENV{DEVTYPE}=="usb_interface",  
  ATTRS{idVendor}=="1d6b", ATTRS{idProduct}=="0104", DRIVER=="",  
  RUN+="/sbin/modprobe_b_uftdi_sio"
```

```
ACTION=="add", SUBSYSTEM=="drivers",  
  ENV{DEVPATH}=="/bus/usb-serial/drivers/ftdi_sio",  
  ATTR{new_id}="1d6b_0104"
```

```
ACTION=="add", KERNEL=="ttyUSB*", ATTRS{interface}=="*BeagleBone*"  
  ATTRS{bInterfaceNumber}=="00", SYMLINK+="beaglebone-jtag"
```

```
ACTION=="add", KERNEL=="ttyUSB*", ATTRS{interface}=="*BeagleBone*"  
  ATTRS{bInterfaceNumber}=="01", SYMLINK+="beaglebone-serial"
```

```
# get ModemManager to ignore BeagleBoneBlack:  
ATTRS{idVendor}=="1d6b", ATTRS{idProduct}=="0104",  
  ENV{ID_MM_DEVICE_IGNORE}="1"
```

Random data for disk initialization

- Write random data to disk before creating encrypted filesystem
- Make cryptanalysis harder
- Large disks + slow source of random bits = trouble.
- Which source of random bits?

/dev/urandom v. /dev/random

- From “Myths about /dev/urandom” Huhn, 2014:

[myth] /dev/urandom is a pseudo random number generator, a PRNG, while /dev/random is a “true” random number generator.

Fact: Both /dev/urandom and /dev/random are using the exact same CSPRNG (a cryptographically secure pseudorandom number generator). They only differ in very few ways that have nothing to do with “true” randomness.

- Can we verify?

Looking into /dev/[u]random

- Source of data? Let's look at the source:

```
sudo apt-get install linux-source
cd /usr/src
pv -cN rd < linux-source-4.9.tar.xz | xzcat | pv -cN xz | sudo tar
  xf -
grep -rl --include=*.c -e '/dev/urandom' linux-source-4.9
```

- From `random.c` Ts'o, 1999:

```
/* ... /dev/random is suitable for use when very high
 * quality randomness is desired (for example, for key generation or
 * one-time pads), as it will only return a maximum of the number of
 * bits of randomness (as estimated by the random number generator)
 * contained in the entropy pool.
 *
 * The /dev/urandom device does not have this limit, and will return
 * as many bytes as are requested. As more and more random bytes are
 * requested without giving time for the entropy pool to recharge,
 * this will result in random numbers that are merely
 * cryptographically
 * strong. For many applications, however, this is acceptable.
... */
```

- Change in Linux kernel 4.8.

Faster random bits?

- /dev/urandom

```
dd if=/dev/urandom of=/dev/null bs=1024 count=1000000
```

- 1024000000 bytes (1.0 GB, 977 MiB) copied, 6.77982 s, 151 MB/s
- Were earlier implementations slower (~ 10 MB/s)?

- openssl

```
openssl enc -aes-256-ctr -pass pass:"$(dd if=/dev/urandom bs=128 count=1 >/dev/null | base64)" -nosalt < /dev/zero | dd of=/dev/null bs=1024 count=1000000
```

- 1024000000 bytes (1.0 GB, 977 MiB) copied, 1.08417 s, 945 MB/s

Observations

- Setting up SoC devices has gotten a lot easier.
- There are still surprises.
- Many interesting detours
- Results:
 - usable system
 - new knowledge
 - fun tinkering

How well has it worked, technically?

- Using 3 wee servers for 10+ years.
 - Hardware: Linksys NSLU2 *slug*
 - Processor: XScale-IXP42x Family rev 1 (v5l)
 - BogomIPS : 266.24
 - On 24x7; still going strong.
 - Some tasks too taxing
 - decoding FLAC audio
 - video: none (OK)
- Moving to BeagleBone Black.
 - aging current hardware (NSLU2)
 - faster disk, network
 - run more/heavier services

How well has it worked, in household?

- Stability is easy to appreciate
- Command line is a mixed bag
 - power and convenience is appreciated
 - Need for reminders
 - cheat-sheets are great
- Tendency to “switch back to the norm”
 - fallacy that non-libre must mean easier

Summary

- A wee home-server: low-cost, high-fun
- Libre software makes it easier to
 - acquire, learn, troubleshoot, contribute
- Software users ? software community



- Contact:
 - Sudarshan S Chawathe
 - <http://chaw.eip10.org/>
 - chaw@eip10.org.

QR Code



References i

Huhn, T. (2014). Myths about `/dev/urandom` .

<https://www.2uo.de/myths-about-urandom/>. Updated November 2016.

Ts'o, T. (1999). `random.c`—a strong random number generator . Linux kernel source code. Version 1.89.