Jonathan James Elliott
B.Sc. Computer Science
Linux SoC Power Management
M.Sc. Advanced Computer Science
September 7, 2012

Supporting Documentation:
http://www.lancs.ac.uk/pg/elliotj3/

Disclaimer

"I certify that the material contained in this dissertation is my own work and does not contain unreferenced or unacknowledged material. I also warrant that the above statement applies to the implementation of the project and all associated documentation. Regarding the electronically submitted version of this submitted work, I consent to this being stored electronically and copied for assessment purposes, including the Department's use of plagiarism detection systems in order to check the integrity of assessed work.

I agree to my dissertation being placed in the public domain, with my name explicitly included as the author of the work."

Date: September 7, 2012

Signed:

Abstract

The power consumption of mobile devices is an increasing issue. As the devices become faster and provide more features they consume more power and this results in less battery life. This project was started to reduce the power consumption of two platforms the TP-Link TL-WR703N and the Raspberry Pi. By developing a power management system which allows the platforms to monitor their own current consumption and respond intelligently when then battery is low. Although the project didn't succeed in developing a power management system it does provide the means for the platforms to monitor their own current usage using the developed current monitor. The current monitor is used to compare the Raspberry Pi and the TP-Link TL-WR703N to determine which platform is more appropriate for the ongoing Mobile Backpack Router project. Existing Linux power management features have been explored and found but unfortunately aren't implemented in the two platforms.

Contents

List of Figures

1. Introduction

The Linux SoC Power Management project was started to compare and reduce the power consumption of two platforms, the TP-Link TL-WR703N and the Raspberry Pi with the required peripherals connected, with the aim of allowing both platforms to monitor their own power consumption and respond accordingly to a low power situation. This is to prolong their active state when being powered from a portable power source such as a battery. This is going to be achieved by developing a power management system to maintain the current consumption status and decide when devices should be placed into a low power state.

2. Background

2.1. TP-Link TL-WR703N

The TP-Link TL-WR703N is a commercial Wireless N 3G capable portable router, using an Atheros AR9331 SoC (System on a Chip). The Atheros AR9331 consists of,

- the Atheros AR7240 CPU (a MIPS 24kc core) operating at 400MHz
- a Wireless N 150Mbps adapter
- a 10/100Mb wired Ethernet adapter
- a USB 2.0 controller

With the AR9331 is 32MB of RAM, a 4MB flash chip, Ethernet port, USB 2.0 port and a Micro USB port. The Micro USB port is only used to provide the router with power at 5v with a current draw of 185mA and the USB 2.0 port is intended for the 3G USB Modem. The router ships with proprietary TP-Link firmware, however OpenWrt have ported their Linux distribution to the platform which will be used within this project.[29] The only standard way of communicating with the TL-WR703N is via the Ethernet or Wireless interfaces using either the telnet or the secure shell protocol.

2.2. Raspberry Pi

The Raspberry Pi is a single board computer which uses a Broadcom BCM2835 SoC. The Broadcom BCM2835 consist of,

- an ARM1176JZF-S CPU operating at 700MHz
- a Broadcom Dual Core VideoCore IV GPU with configurable CPU/GPU RAM split
- a Digital Signal Processor
- a USB 2.0 controller

With the Broadcom BCM2835 is 256MB of RAM, an external Ethernet controller with an Ethernet port, a Micro USB port, an external USB 2.0 hub with 2 USB ports, a HDMI port, a Composite RC socket, a DSI connector, a 3.5mm audio jack, an SD card reader, a GPIO header and UART header. Unlike the TP-Link TL-WR703N, the Raspberry Pi doesn't have any persistent storage such as a flash chip, so instead using SD card is required to boot from and save user data to. The Raspberry Pi is designed to be used with a mouse, keyboard and a monitor but can be communicated with remotely by using the Ethernet interface with the secure shell protocol. The Raspberry Pi officially has support for ArchLinux, Debian and Fedora Linux distributions.[30]

2.3. Related Project

The Linux SoC Power Management project stems from the Backpack Router project as a solution to reducing the power consumption will aid this projects in its development.

The Backpack Router project uses the Ubiquiti RouterStation which is being powered by a 15V Lithium Ion battery. The RouterStation has a USB hub with a USB 3G modem connected to the USB header.

On the internal mini-PCI sockets are 3 WiFi adapters which are the Ubiquiti XtreameRange 2 with a 2.4GHz omnidirectional 5dBi antenna connected to each one of them. Each of the 3 WiFi adapter has different roles, one of them is used to create the access point to allow mobile devices to connect to it. Another is used to provided the MANET protocol ad-hoc link to another backpack router and the last one is used to connect to other APs to receive an Internet connection.

The backhual connection to the internet used by the router depends on the available options with a WiFi connection being the preferred choice. If a WiFi connection doesn't exist it will attempt to connect to the 3G network, and when this fails it will attempt to connect via Satellite being provided by a separate WiFi AP. When there is no backhual connection available the router will use a MANET protocol to ad-hoc with another visible mobile router.[10]

3. Current Monitor

3.1. Research

3.1.1. Battery Monitor ICs

There are a number of ICs (Integrated Circuits) which can monitor current usage to our requirements but no pre-built device which means a circuit will be developed around the chosen IC. The requirements of the IC are that it needs to be able to monitor current and communicate the readings to the platforms.

The DS2438 is developed by Maxim and called the "Smart Battery Monitor" it provides the ability to monitor current, voltage and temperature and accumulates the discharge and charge current. It provides a 1-Wire bus interface which is developed by Maxim and is used to communicate the readings back to another device, even so its called the 1-Wire bus it requires two wires the ground and the data. This is because a circuit is required to read the bits across the wire by measuring the difference in voltage high for 1 and low for 0.[1]

The DS2762 is developed by Maxim and called the "High-Precision Li+ Battery Monitor With Alerts" and provides the ability to monitor current, voltage and temperature and accumulates current. Like the DS2438 it also uses the 1-Wire bus interface which is used to reads the measurements from the registers and configure the IC. Unlike the DS2438 it can be purchased with an internal 25mOhm sense register and also provides a Li+ Safety circuit which prevents over and under voltage protection and over current and short protection. It also has to ability to provide alerts to the host when the accumulated current or temperature has exceeded the user configurable limits.[23]

The DS2745 is developed by Maxim and called the "Low-Cost I^2C Battery Monitor" it provides the ability to monitor the current, the voltage and the temperature and accumulates current. Another feature the DS2745 has is a low power mode which stops all the measurement activity and only allows access to the current accumulators and the configuration registers using the I^2C bus.

The I^2C bus is known as a 2-Wire serial bus and requires a clock(SCL), a data(SDA) and a ground wire. The clock wire is used to synchronize the data over the data wire. The ground is used to create a circuit so the voltage can be measured to sense when the SCL and SDA wires go high and low.

Data is transmitted on the SDA wire in 8 bits per sequence with the SCL wire pulsed by the sender to signal the start of a sequence. Each bit is acknowledge by the receiver by sending an acknowledgment bit to the sender. When the sequence is over the send pulses the SCL wire to signal its ready to receive the next byte.[24]

The BQ26231 is a Texas Instruments "Low-Cost Battery Coulomb Counter For Embedded Portable Applications" and features the ability to monitor Li-Ion, Li-Pol and NiMH batteries. The BQ26231 monitors the current and voltage of the battery and also has accumulators for charge, discharge and self discharge to show the total charge and discharge of the battery over time.

Retrieving the readings from the BQ2631 requires using the HDQ interface which is a single wire interface and requires a data and ground wire. The BQ2631 has the ability to calibrate itself and all the host has to do is tell the BQ2631 to calibrate.[25]

3.1.2. Existing applications

The DS2438 has already been used in an existing router based project called "11km Wireless Link to a Remote Site" by the "ICTP-ITU-URSI School on Wireless Networking for Development". The project involved using two Linksys WRT 54 GL routers one as a client to the original base station and the other as a local AP to create a repeater which was placed 11km away from the original base station.

A Smart Battery Monitor was developed which uses a DS2438, a MAX1614 and a DS2406 with a lead acid battery. The MAX1614 is a "High-Side, n-Channel MOSFET Switch Driver" which is used to connect and disconnect the load from the battery. The DS2438 is used to monitor the voltage of the battery and when it drops below 10.5v the DS2406 which is a "Dual Addressable Switch Plus 1Kb Memory" is used to instruct the MAX1614 to disconnect the load which turns the routers off. This is done via the 1-Wire bus, to connect the load again requires using the push button provided. Unfortunately even so a Smart Battery Circuit with the ability to automatically power on the router when the battery was charged was designed and tested in the lab it was never deployed due to a lack of parts.[9]

The DS2438 is built into the Crossbow Stargate Processor Module which was developed by multiple research groups within Intel. The platform uses the Intel PXA255 Xscale RISC based processor running at 400MHz with an Intel SA1111 StrongArm companion chip for multiple I/O access. The Stargate also provides a PCMCIA and Compact Flash connector, a 51-pin expansion connector for IRIS/MICAz/MICA2 Motes based peripherals such as Crossbows family of wireless network sensors, headers for 2 serial ports and an I2C device and has the option of a Lithium Ion battery.

The Crossbow Stargate has the ability of adding the daughter board which provides a RS232 serial port, 10/100mb Ethernet port, USB support and a JTAG interface. The datasheet doesn't directly identify the DS2438 it only shows a Gas Gauge in the Stargate's block diagram.[26] The report by Vladislav Petkov titled "Using the DS2438 Battery Monitor on Crossbow's Stargate" identifies and describes the DS2438 on the Crossbow Stargate Processor Module and also discusses how to utilize the abilities of the DS2438.[7]

3.1.3. Battery Monitor IC decision

The battery monitor IC that will be used to implement the current monitor is the DS2438 because it has all the necessary features and uses the 1-Wire bus.

The DS2762 also has the necessary features and implements the 1-Wire bus but it also provides a Li+ Safety circuit and the ability to alert the host on the battery state. These extra features are unnecessary for this project because the current being drawn by the platforms is being monitored and not the state of a battery.

The DS2745 uses the I^2C bus which poses a problem when connecting it to the TP-Link TL-WR703N as it requires two GPIOs, one for the SCL and the other for the SDA. This is a problem because the TL-WR703N requires some modification to the gain access to the unused GPIOs. Modifying the TL-WR703N to access one is risky enough but trying to get access to two GPIOs provides an even greater risk of damaging the platform.

The BQ26231 uses Texas Instruments own HDQ interface to communicate with the host which uses 1 wire for data. The Linux support for the HDQ interface is only available for the TI OMAP 2430/3430 platforms which means it can't be used with the GPIOs on the Raspberry Pi and the TL-WR703N.

The DS2438 has been used in a previously documented project such as the "11km Wireless Link to a Remote Site" and the commercial product called the Crossbow Stargate Processor Module. When researching the DS2762, DS2745 and the BQ26231 I failed to find an example of these ICs being used in a project.

3.2. Design and Implementation

3.2.1. Current monitor design

The designed current monitor uses a DS2438 "Smart Battery Monitor" IC instead of the other ICs researched due to availability and ease of programming. A 1-Wire Bus Interface to retrieve the readings can be connected to the GPIOs on both the Raspberry Pi and the TP-Link TL-WR703N, and controlled using an existing kernel module.

Figure 3.1 shows the circuit diagram for the current monitor with the components labeled. USB-GND is connected to VSENSE+ and ROUTER-GND is connected to VSENS- across this are the components R1, R2 and C1. R1 is a 50mOhm resistor known as Rsense, the voltage is measured across this resistor by the DS2438 and used for the current measurements. C1 is a 0.1 microfarad tantalum capacitor and R2 is 100kOhm resistor and these implement an RC low pass filter with a cutoff of approximately 15.9Hz which filters interference, according to the data sheet this doesn't have to be implemented but is recommended[1].

VDD is connected to ROUTER-5V and USB-5V which is VCC on the USB A and MICRO-B USB plugs, C3 which is a 220 microfarad electrolytic capacitor is used to decouple AC noise and smooths voltage ripple on the 5v supply. DQ is the 1-Wire Bus interface and is connected to R3 which is a 4.7kOhm resistor which is also connected to ROUTER-5V and USB-5V this pulls DQ up to 5V and then terminates as an RJ-11 Plug which is the standard connector for 1-Wire Bus devices. The 4.7kOhm resistor is necessary because the DS2438 DQ pin is open drain meaning if there wasn't a pull up resistor then the voltage would remain at 0.3v at all times.

VAD is connected to C2 which is another 220 microfarad electrolytic capacitor which decouples AC noise and smooths voltage ripple, VAD could then be connected to the battery to get an accurate voltage reading to determine the state of charge. The resistors used have a 1% resistance tolerance which means they are within 1% of their value, this provides a level accuracy especially since the voltage across the Rsense resistor is measured and its value is used in the equation to retrieve the current draw. The current monitor was designed to be used with a USB power supply as both the Raspberry Pi and TP-Link TL-WR703N use Micro USB as the power connector.

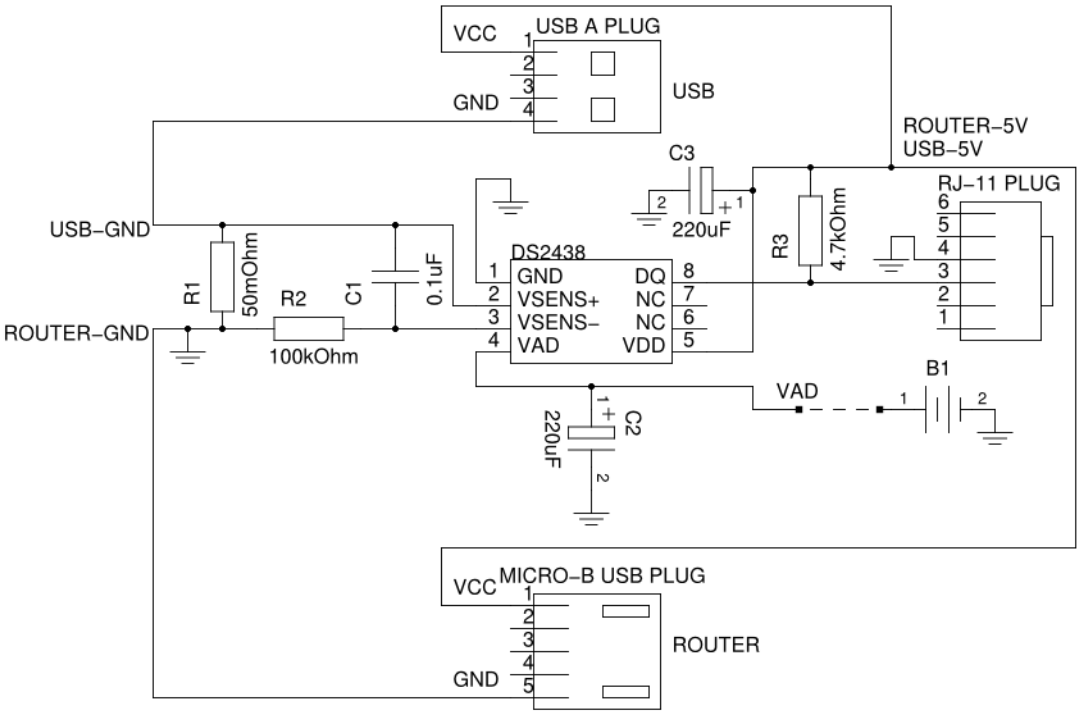


Figure 3.1.: Current Monitor Circuit Diagram

Figure 3.2 shows the PCB layout from the top of the board. The components are labeled exactly the same as the circuit diagram with the addition of LK1 which is a link used to connect the VAD track over the top of the ground track to the header on the board.

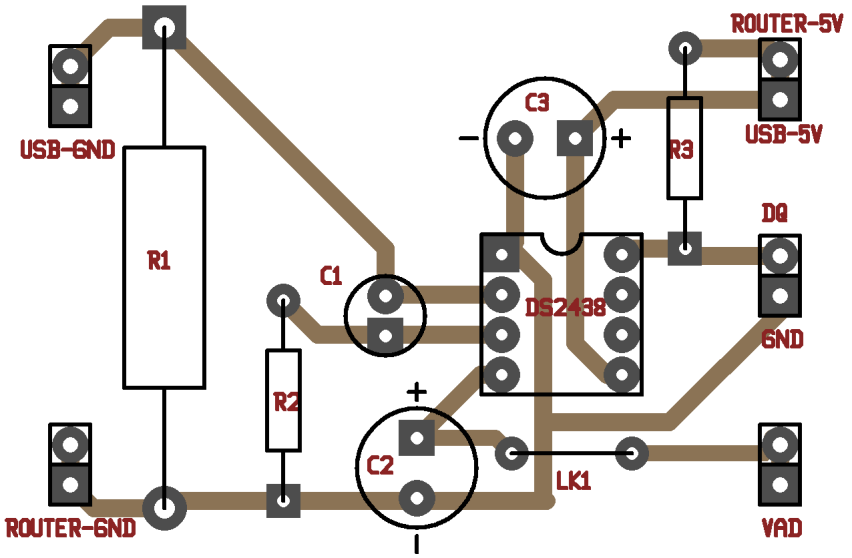


Figure 3.2.: Current Monitor PCB Layout

3.2.2. Modifications for the Raspberry Pi

The version of the current monitor shown in Figures 3.1 and 3.2 only works with the TP-Link TL-WR703N and not the Raspberry Pi. This is due to DQ being pulled up to +5V power rail - the Raspberry Pi's GPIOs can only support 3.3V logic levels, otherwise damage occurs[2].

The solution is to add a 3.6V zener diode across DQ and GND to prevent more than 3.6V being pulled to DQ. The reason a 3.6V zener diode is being used instead of a 3.3V one is because of the 4.7kOhm resistor connected to DQ and +5V power rail. The resistor prevents all the current passing to DQ if the zener diode were to go fault and short to ground, however the resistor also lowers the voltage.

When a 3.3V zener diode was used it lowered the voltage to 2.47V, which posed a problem for the DS2438. The 2.47v reading was close to the low value of 2.4V rated for DQ by the DS2438, so by using a 3.6v zener diode it raised the voltage by 0.30v, making it 2.77v - which is an acceptable voltage.

Figure 3.3 is the circuit diagram of the current monitor modified for the Raspberry Pi. The diagram is the same as Figure 3.1 with the addition of Z1 which is connected to DQ and GND this is the 3.6v zener diode.

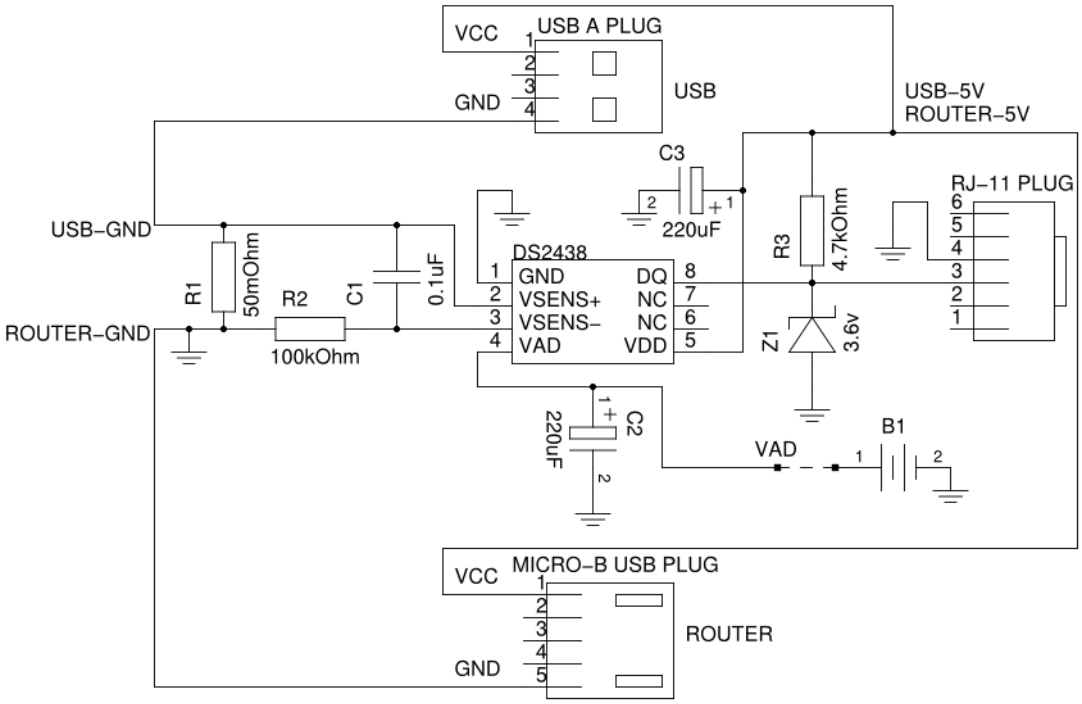


Figure 3.3.: Raspberry Pi Current Monitor Circuit Diagram

Figure 3.4 is the PCB layout for the current monitor for the Raspberry Pi and shows the position of Z1 which is the 3.6v zener diode.

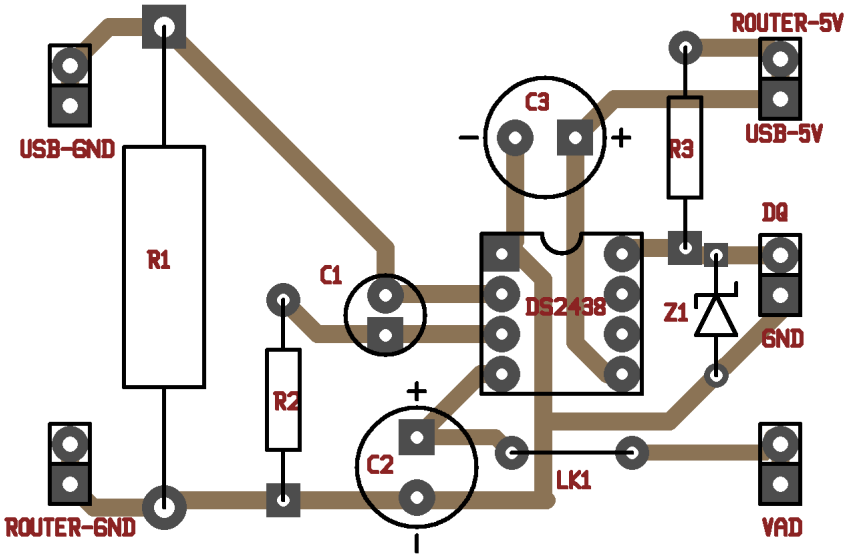


Figure 3.4.: Raspberry Pi Current Monitor PCB Layout

Connecting the current monitor to the Raspberry Pi is straight forward as the GPIOs are exposed as pins on the board and just requires an RJ11 cable with pin headers being produced[2].

3.2.3. Modifications to the TP-Link

Connecting the current monitor to the TP-Link TL-WR703N required some modification. The R17 surface mount resistor pulls GPIO29 to ground because its not being used[3]. To use the GPIO requires unsoldering the surface mount resistor and connecting a wire to the GPIO header and another wire to ground as the 1-Wire Bus requires the signal and the ground. For the ground the R17 pad wasn't used instead a ground point was found near the R17 resistor with a larger surface area which is easier to soldier. Very fine wire was used to solder to the GPIO pad and then attached to an RJ11 cable because the ground pad was large the RJ11 cable could be soldiered straight to it. The fine wire was then glued to the top of the Atheros SoC to prevent the wire from being pulled off the board. The RJ11 cable is glued to and sticks out of the TP-Link TL-WR703N case as a dongle and a coupler is used to connect the RJ11 plug to the RJ11plug of the current monitor.

3.2.4. Further alterations to support USB device monitoring

The Current Monitors have been further altered from the original designs to support the ability to monitor the load of a platform and a powered USB hub together. This is necessary because of Raspberry Pi's requirement of a powered USB hub to support high power USB devices as the internal hub can't provide more than 100mA of current.[22]

The alteration includes the addition of a DC Output Power Supply Lead from Maplin code AQ81C which requires and provides the ability to change the DC jack as the size of the DC jack varies between different USB hubs and takes a maximum of 3A of current. The size of DC jack the Dynamode USB-H40-A2.0 4 Port USB hub requires is a 3.5x1.35mm DC jack. The following circuit diagrams show the positive side of the DC Output Power Supply Lead has been wired to the ROUTER-5V side and the negative to the ROUTER-GND side of both current monitors.

3.2.5. Calibrating the current monitor

The DS2438 comes factory per-calibrated but its recommended to calibrate the DS2438 to ensure the current reading is 0 when no platforms are connected. The calibration data is known as the offset and stored in the offset register. When a current reading is taken the measurement is added to the offset and the result is stored in the current registers. Writing to the offset register involves writing the offset to the offset bytes on the page in the scratchpad and then sending the command to commit that page to the EEPROM.

The procedure to calibrate the DS2438 involves disconnecting the platform from the current monitor and powering it using a separate power supply. The IAD bit is disabled in the status/config register which disable current monitoring and then a zero is written to the offset register. The IAD bit is then enabled and a current reading is taken after which the IAD bit is disabled again. The reading taken is inverted using two's compliment and written to the offset register. Finally the IAD bit is enabled again and the DS2438 is now calibrated.

3.2.6. Verifying the DS2438 results

Verifying the results of the DS2438 required another device to be developed which allows two multimeters to be connected between the power supply and the current monitor.

The device consits of two terminals, a USB socket and a lead with a USB plug on a piece of Veriboard. One terminal is used to monitor the voltage and the other terminal is for monitoring the current, the board also has a jumper which connects the current terminals together so voltage can be tested alone. This is required because the multimeter needs to be connected in series to the circuit to be able to monitor the current. Voltage is measured in parallel across the 5V and ground.

The USB socket is fully wired to the USB plug which allows individual USB peripherals to be tested with the multimeter as data is required to initial certain peripherals. The multimeter that was used to verify the current is a UNI-T UT60E which measures DC current in the ranges of 400 to 4000 microamps, 40 to 400 milliamps and 4 to 10 amps.

The accuracy of each range is +-(1%+2) in microamps, +-(1.2%+3) in milliamps and +-(1.5%+5) in amps. The meter was used in the 4A range by using the provided accuracy specification this will produce the error of the reading and the two possibilities for the exact reading from the meter.

3.3. Configuring the Platforms for the Current Monitor

3.3.1. TP-Link TL-WR703N

As OpenWrt is being used on the TP-Link TL-WR703N to use the current monitor requires compiling OpenWrt with the required kernel modules enabled. These kernel modules are wire, w1-gpio-custom and w1-gpio. The wire kernel module adds support for the Dallas 1-wire bus, w1-gpio-custom adds the ability to specify up to 3 different GPIOs as 3 different buses to be used for the 1-wire bus as parameters at the launch of the module and w1-gpio is the w1 GPIO bus master driver.

A patch had to be applied to the code that handled communicating on the 1-wire bus in the wire module. The patch added the ability to disable IRQ interrupts when performing reading, writing and resetting the bus operations and increasing the amount of time the bus master holds the bus low during the reset operation from 480 to 500 microseconds[4]. This patch is necessary because there are a limited number of microseconds to read bits being transmitted across the bus if an interrupt was to occur. The result was that the bus master could miss a bit and corrupt the overall result.

Another patch was required in the w1-gpio-custom module to prevent a segmentation fault occurring when trying to launch the module. This was due to a change within the w1-gpio module which added a callback function pointer to enable and disable the external pull-up resistor to let devices on the bus know if the processor is driving the bus or not[5]. The w1-gpio-custom module didn't initialize the pointer which caused the segmentation fault, as there isn't a function to handle this callback and was initialized to 0.

Every time the TP-Link TL-WR703N is flashed the configuration files are erased which means the w1-gpio-custom module would need to be reconfigured each time. Modifying the MakeFile by adding just the AutoLoad option won't work since parameters can't be specified. Instead, a patch can be applied to functions.sh in the base-files package which updates the script to look for a configuration file in "/etc/config" with the module's name in the uci config format and applies the parameters when loading the module[6]. After modifying the w1-gpio-custom package to include a configuration file and place it into the correct folder when OpenWrt is compiled, the module configuration is now built into the ROM.

The kernel itself can contain specific module drivers for 1-wire devices, but there isn't a module available for the DS2438, so a default generic file descriptor called "rw" is provided to give raw read and write access to the device.

3.3.2. Raspberry Pi

The Raspberry Pi loads its firmware and boots from the SD card. Using the provided Debian image for the Raspberry Pi required updating the firmware in the boot partition to boot the kernel compiled from the latest Raspberry Pi kernel source. The Raspberry Pi also required the enabling of the kernel modules such as wire, w1-gpio and the addition of the patched w1-gpio-custom module from OpenWrt, and inserting into the kernel sources.

When booting from the compiled kernel and configuring the modules, using the test program it frequently took 3 to 4 attempts to get a read from the DS2438 without a CRC failure. Original thought suggested the addition of the zener diode added noise to the signal but after trying the modified current monitor on the TP-Link TL-WR703N the test program read from the DS2438 perfectly every time with the CRCs matching.

The patch for wire module hadn't been applied to the Raspberry Pi kernel source, and the OpenWrt patch failed to apply. This is because the Raspberry Pi kernel is based on 3.1.9 and the OpenWrt kernel is based on 3.3.8. After comparing the kernel source for the wire kernel modules with the OpenWrt kernel which has the patch applied, all the original w1 modules in the Raspberry Pi kernel source were updated to the level of the OpenWrt kernel.

During the comparison there was a difference in the method w1_read_bit, before the patch used on the OpenWrt kernel. The method didn't disable IRQs, and this was the cause of the corruption. A new patch was generated to fix this issue, and to add the w1-gpio-custom module. After this new kernel was compiled and booted the test program read the DS2348 perfectly every time with the CRCs matching.

4. Power Management

4.1. Introduction

When it comes to power management there are two major standards Advanced Power Management (APM) and Advanced Configuration and Power Interface (ACPI).

APM was developed by Microsoft and Intel and allows the operating system to send power management instructions to the BIOS which then controls the power states of the devices and the system. The BIOS can also generate events which the operating systems APM driver polls to be informed of events such as a low battery notification.[12]

ACPI is the successor of APM as it removed the need for the operating system to talk to the BIOS to change the power state of the devices and the system. It was designed to be far more than just a power management solution but also replace the Plug and Play BIOS and the MultiProcessor specifications.[11]

However ACPI and APM are only available for desktop computers, laptops, netbooks and not embedded devices such as the Raspberry Pi and the TP-Link TL-WR703N. There is no set standard to manage the power consumption across embedded devices. Each of the devices will have unique specific hardware interfaces developed for dealing with power management.

Looking into these specific interfaces could provide the ability to reduce power consumption if they are implemented for the hardware within the Linux kernel. Implementing these interfaces would require having the data sheet for the SoC which describes the various registers available to configure the hardware but this is proprietary information and won't be easy to get hold of. Any progress made on implementing the different available interfaces won't be portable between the TP-Link TL-WR703N and the Raspberry Pi.

There are some generic interface which will be compatible across the platforms like the ability to disable network interfaces such as the wireless interface to reduce power consumption as this is tailored towards the wireless interface and its drivers and the Linux network architecture.

4.2. Linux Power Management Frameworks

The Linux kernel supports both the APM and ACPI power management standards and implements a power management framework that adds the ability to place the whole system into sleep, suspend and hibernate. The power states themselves have to be implemented across all the device modules for the system to be able to enter the state fully.

The "Runtime Power Management Framework for I/O Devices" adds the ability to control the power state of individual devices connected to the system while the system is running as opposed to placing the whole system into a different power state. The support for this framework has to be implemented into the device drivers.[20]

The Linux USB core has the ability to allow the kernel to autosuspend individual USB devices when they are idle for a period of time and for the USB device to remotely wakeup the system. By default autosuspend is disabled on all USB devices except USB hubs because certain USB devices fail to implement the standard correctly. The usbcore can be configured to enable or disable autosuspend and remote wakeup. The number of milliseconds individual

peripherals have to be idle before the kernel suspends them is also configurable. The "Runtime Power Management Framework for I/O Devices" has to be implemented and enabled for the usbcore to enable these power management features.[21]

There are two CPU power management frameworks cpufreq and cpuidle, the cpufreq framework implements the ability to adjust the CPU clock speed and the cpuidle framework places the processor into a sleep mode when the CPU is idle.[13]

The pm_qos framework provides power management for quality of service this allows subsystems, applications and drivers to register their performance requirements by setting one of the available parameters. These parameters include cpu_dma_latency, network_latency and network_throughput. The second power management for quality of service framework provides an API to manage per device latency requirements.[15][16]

The mac80211 infrastructure for wireless cards supports the pm_qos framework and has the ability to dynamically save power when the device is associated to a network and is receiving no traffic for a certain amount of time. When dynamic power saving is on the wireless interface will sleep for the DTIM interval.[17] The AP assists it clients by buffering their frames while they are in a low power mode. The DTIM is an element which is sent in certain beacon frames from the AP which indicates to clients in low power modes that there are buffered frames to collect.

When the DTIM is sent is determined by a value which indicates in how many beacon frames to send the DTIM, if the value was 1 it would be sent every beacon frame if the value was 10 it would be sent every 10 frames.[18] The interval at which beacon frames are sent is controlled by the Target Beacon Transmission Time Time (TBTT) which specifies in Time Units (TU) when the AP must send a beacon to its clients. A Time Unit represent 1024 microseconds a default value is 100 TU which is 102400 microseconds.[19] When the DTIM is sent the wireless interface wakes up and collects the buffered frames waiting for it.

The network latency pm_qos parameter affects whether dynamic power management is on or off, if an application specifies a network latency which is greater than the beacon interval then dynamic power management will be off for the duration the application requires that network latency. If the value is greater than the beacon interval then it is compared with the DTIM interval to see if power management can be enabled to ensure the buffered frames can be delivered within each DTIM interval instead of at each beacon.[17]

The dynamic power mode can only really assist the wireless interface when it is in managed mode and not being used in master mode which means if the router was being as a wireless router than this feature would have no use but the TTBT and the DTIM could be tweaked to reduce power consumption on the AP and allow its clients to spend more time in a low power state.

Unfortunately even so mac80211 supports these power saving features there not implemented across every single wireless device module which uses the infrastructure. This means the choice of wireless USB dongle can have an effect on the ability to reduce the amount of current being consumed.

4.3. Extensions of Linux Power Management

Android is an operating system which runs on embedded devices such as smartphones and sits on top of the Linux kernel. As the Linux kernel provides basic power management interfaces Android implements its own power management layer on top of the kernel. Applications request resources such as CPU, wireless and GPS using wakelocks which hold the device in an active state. When no wakelock exists for the specific device then it is powered down.[14]

4.4. Examining The Platforms

The TP-Link TL-WR703N Linux kernel source code doesn't implement cpufreq framework which means the cpufreq tool can't be used to change the CPU clock speed. The cpuidle framework isn't implemented which means there is no method of putting the CPU into a deep sleep when its idle. There is no code implemented within the kernel to put the TP-Link TL-WR703N into a low power state such as sleep, suspend or hibernate and power off.

If the wireless interface isn't needed then it can be completely disabled by setting the wireless disabled option in the configuration files this causes a 30mA drop in the current usage. Using ifconfig to take the wireless interface down just leaves the interface unconfigured and powered as ifconfig couldn't detect the wireless interface but iwconfig and iw was able to.

The USB power management capabilities aren't enabled by default because they require the "Runtime Power Management Framework for I/O Devices" which is also disabled because the TL-WR703N doesn't have any power management capabilities.

The Raspberry Pi Linux kernel source code is also missing the cpufreq and cpuidle frameworks and doesn't implement the methods to suspend sleep or hibernate however there is a method to power off the Pi but it doesn't actually power the system down instead it begins performing the steps to reboot the system and then tells the GPU not to reboot and the system then remains in a halt state.

The Raspberry Pi doesn't have a wireless interface which means the chipset of the chosen wireless USB dongle will have an affect on whether or not the wireless interface can be placed into a low power mode.

The Raspberry Pi's GPU has its own RISC based core which boots the Pi by loading the GPU firmware and kernel image from the SD card and then starting the ARM core. Unfortunately this means the GPU cannot be disabled as it is a necessary component. The sound card also has very limited support within Linux and provides no way to disable or suspend the device. As with the TL-WR703N the USB power management is also disabled because of the lack of any power management implementation for the platform.

5. Testing Environment Setup

5.1. Application

5.1.1. Logging application

A test program was written in C which uses this file descriptor to send commands to the DS2438 to retrieve the temperature, voltage, current, discharge accumulator and the elapsed time. The DS2438 calculates and transmits the CRC over the 1-Wire Bus, and the test program uses the DOW CRC look up method as described in the Maxim application note 27 to validate the communication.[27] Both results are compared to determine is the received data is accurate. The commands and how the values are stored and calculated are described in the DS2438 Datasheet[1].

When building the test application I found the datasheet was unclear on how the CCA, DCA and ICA registers increment instead a paper called "Using the DS2438 Battery Monitor on Crossbow's Stargate" written by Vladislav Perkov which discusses using the DS2348 on the Crossbow Stargate describes how these registers are incremented. The paper also provides details on the ICA and DCA registers and provides the formulas to retrieve the Ah and mAh from the DCA register value[7].

The logging application called ds2438_logger was developed from the test program. The first version of the logging application, took an interval in milliseconds, a sample number and a file path as parameters and queried the current monitor at the interval for the sample number of times, producing a file using the file path parameter which contains the results.

During the initial testing of the logging application an issue was discovered which caused the last number of results to all be equal when the application was querying the current monitor for a large number of times. This issue was caused by the rw file descriptor which reached a point were it wouldn't allow anymore writing to and reading from and so the value in the buffer was the same. The solution was to close and reopen the file descriptor when the exact number of bytes to write to and read from wasn't returned by the write and read methods.

5.1.2. Logging daemon

The logging application was converted into a daemon to allow init to load the application on startup which required an init.d script to be written which launched the daemon with the specific parameters. The ability for the logging daemon to control the led using a bash script was also added to allow for the monitoring of the logging without being connected to the platforms by Ethernet or Wi-Fi as this consumes power and will affect the results. The led blinks a certain number of times depending on the status of the logging for example two blinks means the logging has started and three blinks means the logging has finished.

At this point a new feature was added which adds the ability to turn specific devices on and off by calling a bash script with parameters. Three new parameters called interface which represents the device to bring up and down, up-time in milliseconds is the time to bring the device up and down-time in milliseconds is the time to bring the device down during the test were added to the logging daemon. This is to allow the logger to capture the difference between the power consumption when the network interfaces and external peripherals are in different power states.

During the execution of the logging daemon on boot the logger would always catch the end of the boot process this was shown by the high current usage and then the sudden drop followed by leveling out. To solve this issue required implementing another parameter called wait which allowed the logger to wait the specified number of milliseconds before beginning logging. This would require the timing of the boot process to get the wait value to be specified as the parameter.

5.1.3. Further development

During development to monitor the power consumption of the Raspberry Pi, some new features were added to the logging daemon to collect more information during the tests. This new information includes the voltage and temperature from the DS2438.

The voltage is important as an sudden drop could potentially bring the test points on the Raspberry Pi below the stable range. As the current usage increase this places strain on the power supply and the voltage tends to drop so there should be a direct link between the current usage and the voltage drop. The temperature of the environment the power supply is being used in can affect how efficiently it functions this means the power supply might produce a lower voltage or withstand less current.

Monitoring the CPU usage is important as the more time the CPU is spent active the more current the platforms will draw. The CPU usage is monitored by taking multiple readings of a processes CPU time, measured in a unit called "jiffy", for each state from the "/proc/stat" file descriptor and calculating the percentage difference of the jiffies the CPU has been active as opposed to idle from the total difference. It's for this reason there is no CPU usage record for the zero time interval. A jiffy is roughly 1/100th of a second on most CPU architectures, but is kernel and platform dependent.

The test application has separate methods to retrieve the current, temperature and voltage. These methods were used in the logging application but caused an issue with the interval time being constantly exceeded. This is because each of the separate methods read the DS2438 which results in 3 separate reads. However the voltage, temperature and current are stored within the same page in the EEPROM, meaning the page only needs to be read once. A new method was created which extracts all the necessary data from the EEPROM using only one read which solved the issue.

Network benchmark applications such as iperf can give a good insight into the current and CPU cost of transferring large amounts of data over a network. This is why the logging daemon has been extended to have the ability to launch iperf as a client. As this only shows the platform generating and transferring traffic a new feature and program called iperf_signal was developed. The logging daemon connects to the client running iperf_signal and sends a message this causes the client to start iperf to an iperf server which is running on another client connected to the platform. The logging daemon can also start an iperf server before logging has begun to allow the client to connect to itself and produce results of the platform processing traffic.

During the implementation of the network benchmark feature an issue occurred which resulted in the results file containing repetitions of the same data and missing parts when the logging daemon sent the message to the clients to start iperf. The solution to this was to use fflush to flush all the data to the file before the call was made to the clients.

The program developed to calibrate the DS2438 is based on the existing code base for the logging daemon as follows the procedure discussed in the datasheet to calibrate the DS2438. The logging daemon only sends commands to and reads data from the DS2438. The existing write method was simple and had no verification of the data written to the scratchpad. This means if data corruption was to occur there was no way to check and the corrupted data would be committed to the EEPROM.

The improved write command writes the data to scratchpad reads it back and compares the CRC of the data written to the CRC of the data read. If the CRC's match then the command to commit the data to the EEPROM is sent. Otherwise a total of 10 attempts are made to write the data and when these fail the operation is abandoned and the failure is reported to the user. The error caused by the circuit necessary to the DS2438 was 14.6mA for both of the current monitors.

5.2. Designing A Suitable Baseline Test

There are a number of key factors which need to be taken into account when designing a test using the logging daemon and the current monitor. One of these factors is the affect on power consumption when the logging daemon is running. The act of querying the current monitor will consume power which means the more often you query the current monitor the more power will be consumed by the process over time. However if you query the current monitor with a longer interval the results might not show any current changes which happen in a small amount of time such as current spikes.

The logging daemon records the results to a file on the internal file system the act of creating a file and then constantly writing to it will consume current even if only a small amount and will still be reflected in the results.

The DS2438 takes a current reading every 27 milliseconds, and it is not possible to align the queries for when the logger takes a reading, as there is no way of knowing what stage the DS2438. If it was switching between two values due to inaccuracies, then you could end up with the same value for most of the results.

Another factor, is the additional processes which could be running on the platforms during the logging and could result in an occasional higher current usage which would affect the overall results. This would require disabling all the non necessary services when the logging begins.

The 1-Wire modules disable IRQs when the DS2438 is queried which affects the other devices on the platforms as they are unable to generate an interrupt for any events that happen. This means the current usage could be recorded as lower than it actually is due to these events being ignored when the reading is taken.

After all these factors are taken into account the tests themselves will need to record the power consumption of the platforms without any peripherals connected and then slowly introduce different peripherals in the different tests. As connecting an Ethernet cable will cause the Ethernet interface to consume power and as of yet there isn't the means to disable it from within Linux the only way disabling the Ethernet interface is to not connect a cable.

Each test will run from boot with a wait time of 20 seconds to ensure the boot process has finished with a 100ms reading interval for 1000 times which means each test will run for a 100 seconds. Initial test showed that a 100ms interval was fast enough to catch any sudden changes in current usage without being too fast that the readings overran their intervals. Running for a 1000 times ensures any repeating patterns in current usage are represented in the results without ending up with too many data points.

5.3. Issues With The Raspberry Pi

The "R-Pi Troubleshooting" wiki page in the "Troubleshooting power problems" section mentions two test points on the Pi which are used to read a voltage using a multimeter set on the 20V range. The voltage reading should be in the range of 4.75V to 5.25V which indicates the Raspberry Pi has enough voltage to function correctly. If the voltage is not within the range then this indicates a problem with the power supply.[22]

The Lipo Rider Pro is a 5v USB power supply which can uses either a Lithium Polymer or Lithium Ion battery to provide the power. It also preforms the charging of the battery using either a solar panel or a Mini USB power supply and provides a maximum current output of 1A. The Lipo Rider Pro was used with a charged Lithium Polymer 3.7V 2000mAH when testing it with the Raspberry Pi. On connecting the Lipo Rider Pro to the Raspberry Pi, the Raspberry Pi appeared to boot but after checking the two test points the voltage reading was 4.66V which indicates there isn't enough voltage for the Pi to function correctly.

The Maplin 2 Port USB Charger Model N13KG is mains powered and provides an output of DC 5v 2.1A. On connecting the USB charger to the Pi, the Pi booted successfully and the voltage reading from the test points was 4.83V which indicates there is enough voltage for the Pi to function correctly.

Due to the requirement of a powered USB hub for high powered USB peripherals the Dynamode USB-H40-A2.0 was chosen because it has 4 USB ports and a DC jack socket. It also comes with a DC 5V 1000mA mains power supply but the USB hub will be powered by the DC jack on the current monitors. On connecting a high powered USB peripheral such as a 3G modem to the hub while the hub was connected to the Raspberry Pi. The Raspberry Pi's test points voltage read 4.79V which is within the range and indicates the Raspberry Pi will function correctly. However this reading was taken when the 3G modem was in an idle state and the voltage can drop further when the 3G modem is connected and transferring data over the network this is because the Raspberry Pi and the USB hub will share the same power supply.

The powered USB hub provides power to the host which causes an issue with the Raspberry Pi as it attempts to boot from the powered USB hub. The result of this is the Raspberry Pi being in a state of limbo, when the actual power supply is attached the Raspberry Pi doesn't continue to boot. The only solution to this is to disconnect both the power supply and the powered USB hub from the Raspberry Pi. Then connect the power supply first and then the powered USB hub, even then the Raspberry Pi's USB hub can crash and results in no USB peripherals including the powered USB hub being detected. This is until the Raspberry Pi is reset by disconnecting and connecting the power supply.

6. Testing Results

6.1. TP-Link TL-WR703N

6.1.1. Idle Average Current Usage

The TP-Link TL-WR703N was tested using the current monitor by individually adding and enabling different network interfaces and monitoring the current while the interfaces were connected to a network but not sending or receiving data. This gives an idea of the idle current draw of the network interfaces compared with the TL-WR703N with no interfaces connected or enabled. The tests ran for 100 seconds with a reading taken every 100 milliseconds. The current readings were averaged to produce figure 6.1.

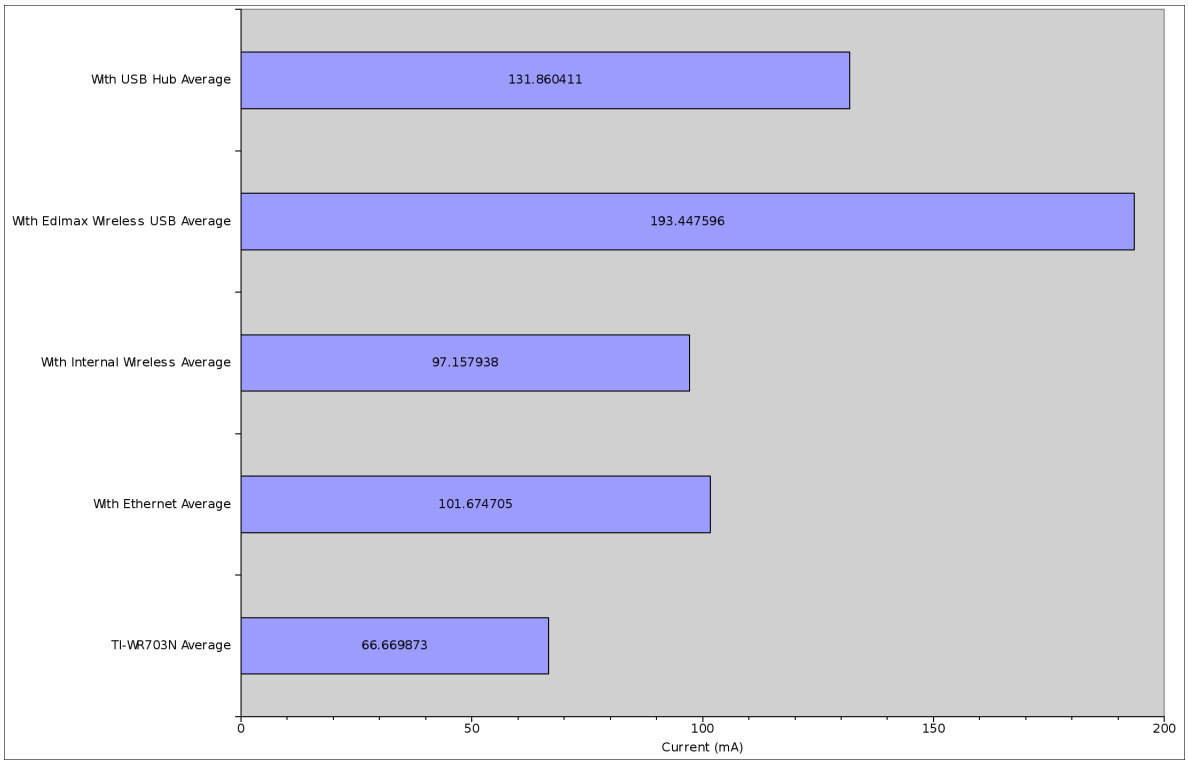


Figure 6.1.: TL-WR703N Idle Average Current Usage

Figure 6.1 shows the TL-WR703N was drawing 67mA when the platform was idle and with no network interfaces enabled or connected. The current draw increased to 102mA when an Ethernet cable was connected. This shows the Ethernet interface caused an increase of 35mA. With the internal wireless interface enabled the current draw increased to 97mA which is an increase of 30mA. This shows the Ethernet interface draws 5mA more than the wireless interface.

The largest current increase occurred when the Edimax EW-7711UAn wireless USB dongle was connected. The current reached 193mA which is an increase of 126mA when compared with the TL-WR703N and a 96mA increase when compared with the internal wireless interface. Both of the wireless interfaces were configured as access points with power saving

enabled because it is enforced when the wireless interface is in AP mode.

6.1.2. Scenario Tests

The TL-WR703N was tested in different usage scenarios which are routing packets from the internal wireless interface to the Edimax EW-7711UAn USB wireless dongle, sending and receiving data from the internal wireless interface and sending and receiving data from the Ethernet interface.

These tests also include monitoring the CPU usage of the TL-WR703N to determine the amount of current being drawn by the CPU. This is because iperf is being used to generate the data being sent and log the data being received. Monitoring the current draw of the platform when the CPU is under different amounts of load will give an idea of the amount of current iperf is consuming.

During each test the iperf bandwidth will be doubled each time up to 64Mbps. The transport protocol used will be UDP because iperf only allows UDP when setting the bandwidth. This means that packet loss between the iperf client and the server will be an issue.

Each wireless scenarios will be tested both with and without encryption enabled. This is to monitor if the encryption scheme causes an increased CPU usage and a greater current draw. The encryption scheme chosen for the tests is WPA2-PSK because it is the most widely used encryption scheme and doesn't require a lot of configuration.

The internal wireless interface was configured as an AP using the Wireless N standard with a theoretical maximum bandwidth of 72Mbps.

Each test will run for 100 seconds with a reading interval of 100 milliseconds. The network interface will be sending or receiving for 60, seconds, starting at 20 seconds and finishing at 80. This will produce a current graph showing the entire 100 seconds of the test with the current rising and dropping due to the network activity.

The average current and CPU usage networking activity graphs will be produce from the average of the 60 seconds the network interface was active.

The CPU graphs are produced from the same setup as each scenario with the exception that iperf isn't running. Instead an application called cpulimit which limits an application to a certain amount of CPU usage. Cpulimit will be running a test application which wastes CPU cycles by counting down. Each CPU graph will include the idle current usage for the network interface being tested.

6.1.2.1. Sending data wireless

The following set graphs show the TL-WR703N sending data to another TL-WR703N using the internal wireless interface.

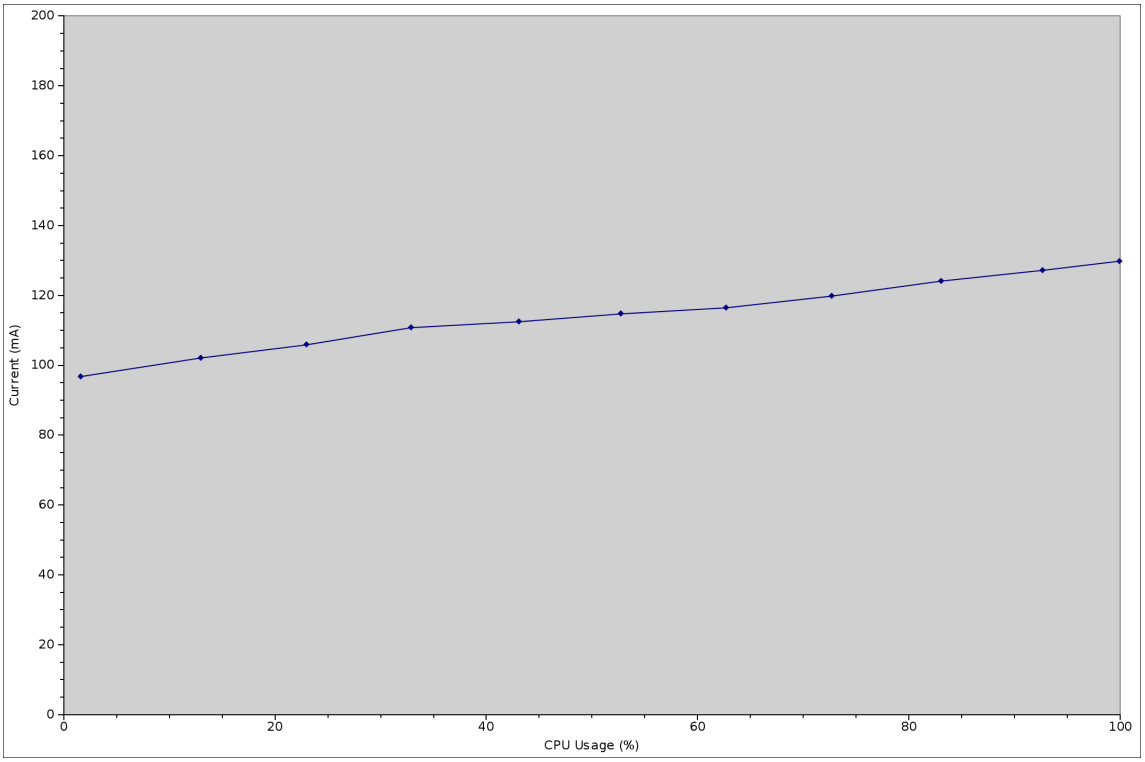


Figure 6.2.: TL-WR703N Wireless Open CPU Percentage Current Usage

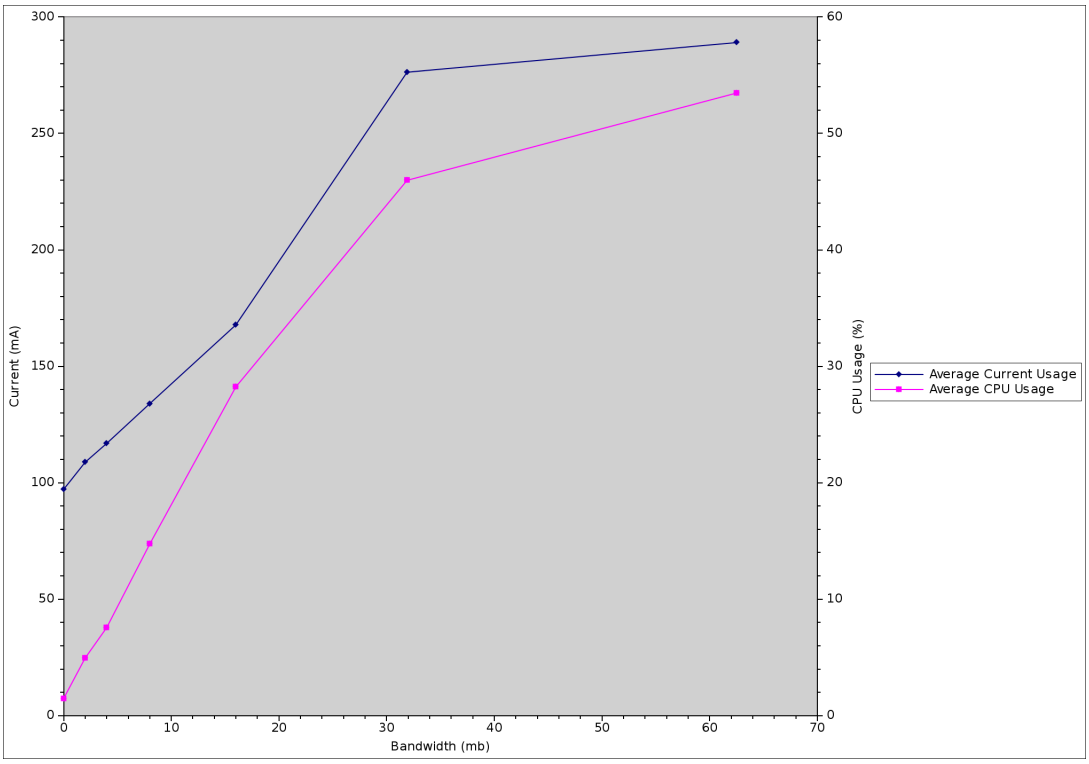


Figure 6.3.: TL-WR703N Wireless Open Sending Data

Figure 6.3 shows the maximum bandwidth of the internal wireless interface was 62.5Mbps when iperf was set at 64Mbps. At this point the TL-WR703N was drawing 289mA of current and using 53% of the CPU. By using figure 6.2 we can see the TL-WR703N draws 115mA at

53% CPU usage. This means the internal wireless interface was drawing 174mA of current while sending data at 62.5Mbps.

At 31.9Mbps the TL-WR703N was drawing 276mA and using 46% of the CPU. The platform draws 113mA at 46% which means the wireless interface draws 163mA of current. The current draw at 16Mbps is 168mA and the CPU usage is 28%. The platform uses 108mA when the CPU is at 28%. The current draw of the wireless interface is 55mA at 16Mbps.

Comparing the differences in current increase when the bandwidth is doubled shows the internal wireless interface began reaching the maximum amount of current draw at 31.9Mbps. This is because the difference between 31.9Mbps and 62.5Mbps is 11mA compared with 16Mbps and 32Mbps which is 108mA.

The difference in CPU usage between 16Mbps and 31.9Mbps is 18% and between 31.9Mbps and 62.5Mbps its 7%. This shows the CPU isn't working as hard even so the bandwidth was doubled. This is because the internal wireless interface couldn't send any faster which means the CPU wasn't doing as much work.

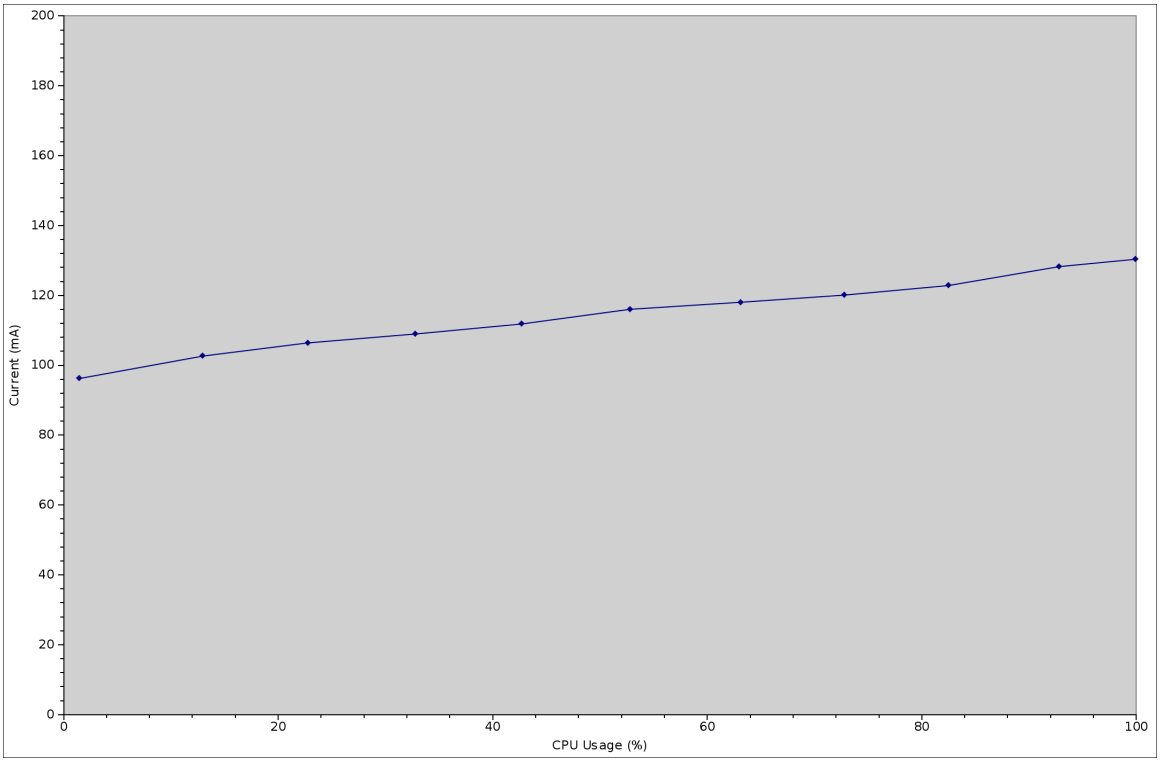


Figure 6.4.: TL-WR703N Wireless WPA2-PSK CPU Percentage Current Usage

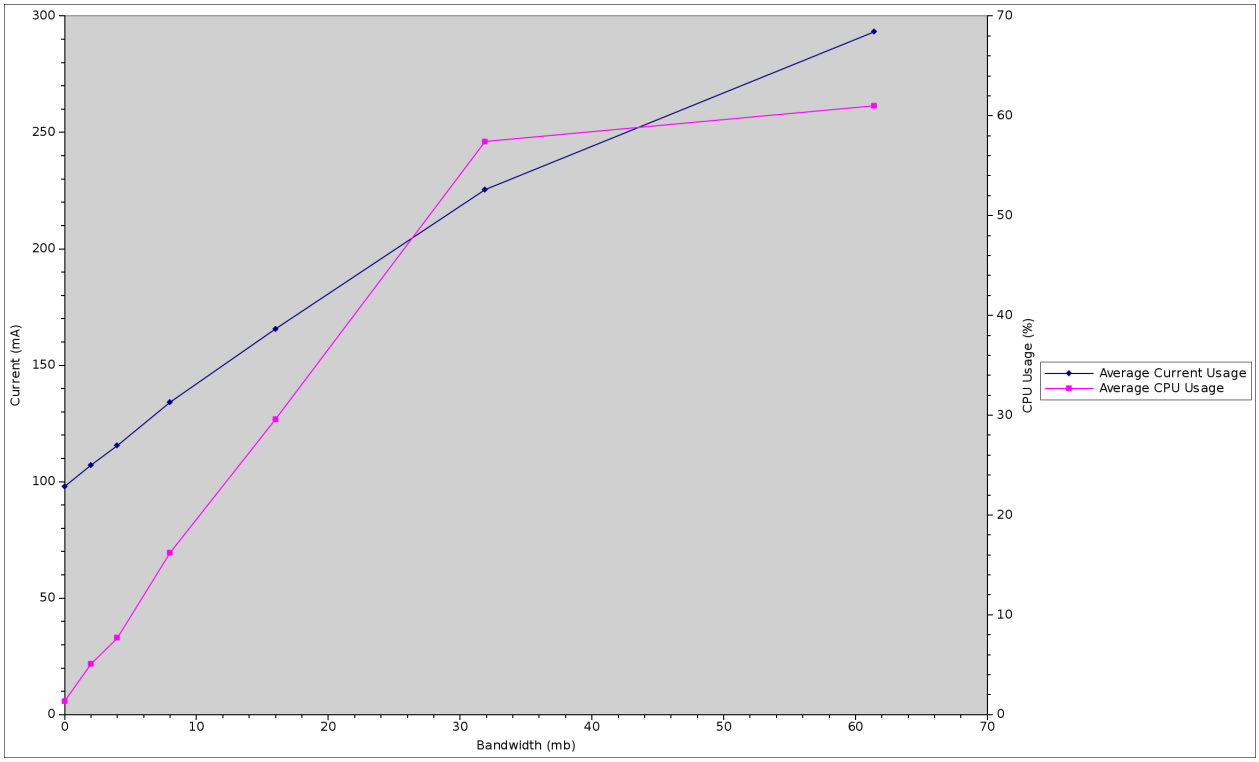


Figure 6.5.: TL-WR703N Wireless WPA2-PSK Sending Data

Figure 6.5 shows the maximum bandwidth the internal wireless interface achieved was 61.4Mbps with WPA2-PSK was enabled. This is a slight drop when compared with the unencrypted test.

At 61.4Mbps the current draw is 293mA with 61% CPU usage. Comparing the CPU usage with figure 6.4 this shows the TL-WR703N draws 117mA at 61%. This means the wireless interface is drawing 176mA which is a 2mA more than the unencrypted test even so the wireless interface is transferring at a slightly slower speed.

The current draw is 225mA at 31.9Mbps with a CPU usage of 57%. The TL-WR703N is using 117mA at 57% CPU which is the same amount of current as 61% CPU usage. The internal wireless interface is using 108mA. Comparing the current usage of the wireless interface at 31.9Mbps with the current usage at 61.4Mbps it's a difference of 68mA.

When the internal wireless interface is transferring at 16Mbps the current draw is 165mA and the CPU usage is 30%. The TL-WR703N draws 108mA at 30% CPU usage which means the wireless interface is drawing 57mA. When comparing the CPU usage between the 61.4Mbps and the 31.9Mbps bandwidth it's a difference of 4%. Between the 16Mbps and the 31.9Mbps bandwidth the CPU usage difference is 27% which shows the CPU is doing less work when the bandwidth is doubled from 32Mbps.

Unlike the unencrypted test the difference between 61.4Mbps and 31.9Mbps is greater than the difference between 31.9Mbps and 16Mbps. This is because the WPA2-PSK is adding an overhead which causes the internal wireless interface to require more current when transferring at faster speeds.

However like the unencrypted test the CPU usage difference has decreased because the internal wireless interface can't transfer any faster and therefore the CPU isn't doing as much work.

Comparing figure 6.2 with figure 6.4 shows that overall the TL-WR703N was drawing more current with WPA2-PSK enabled.

6.1.2.2. Receiving data wireless

The next set of graphs show the TL-WR703N receiving data from another TL-WR703N using the internal wireless interface.

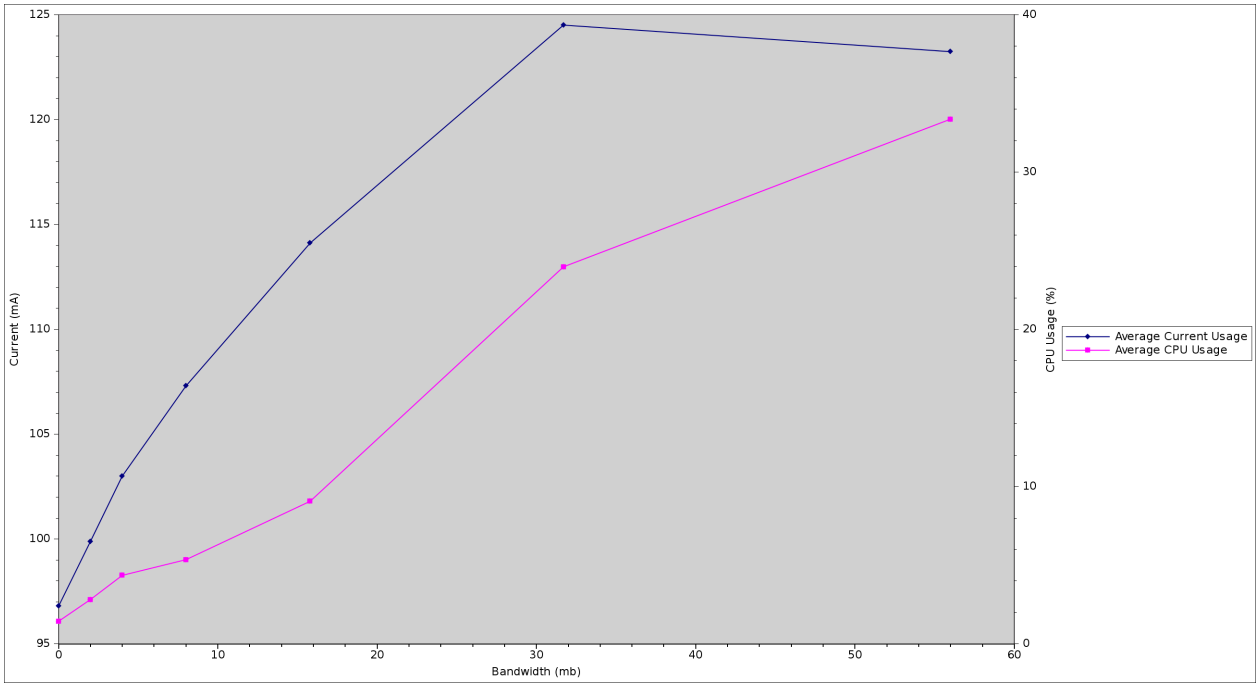


Figure 6.6.: TL-WR703N Wireless Open Receiving Data

When the client was sending at 63.9Mbps the TL-WR703N was receiving at 56Mbps and the current draw was 123mA with the CPU usage at 33%. The TL-WR703N draws 110mA at 33% CPU usage based on figure 6.2 which shows the wireless interface was drawing 13mA to receive at 56Mbps.

When the TL-WR703N was receiving at 31.7Mbps it was drawing 125mA and using 24% of the CPU. The TL-WR703N uses 106mA when the CPU is at 24%. This indicates the internal wireless interface is using 19mA. The current draw dropped when receiving at 56Mbps because the client was sending at 63.9Mbps. The packet loss increased to 12% from 1.1% at 31.7Mbps. This shows the wireless interface was performing less work because it couldn't handle the transfer speed and was dropping packets.

The CPU usage carries on increasing because the iperf server is logging the packets being received as well as the ones missing and generating a report to send back to the client.

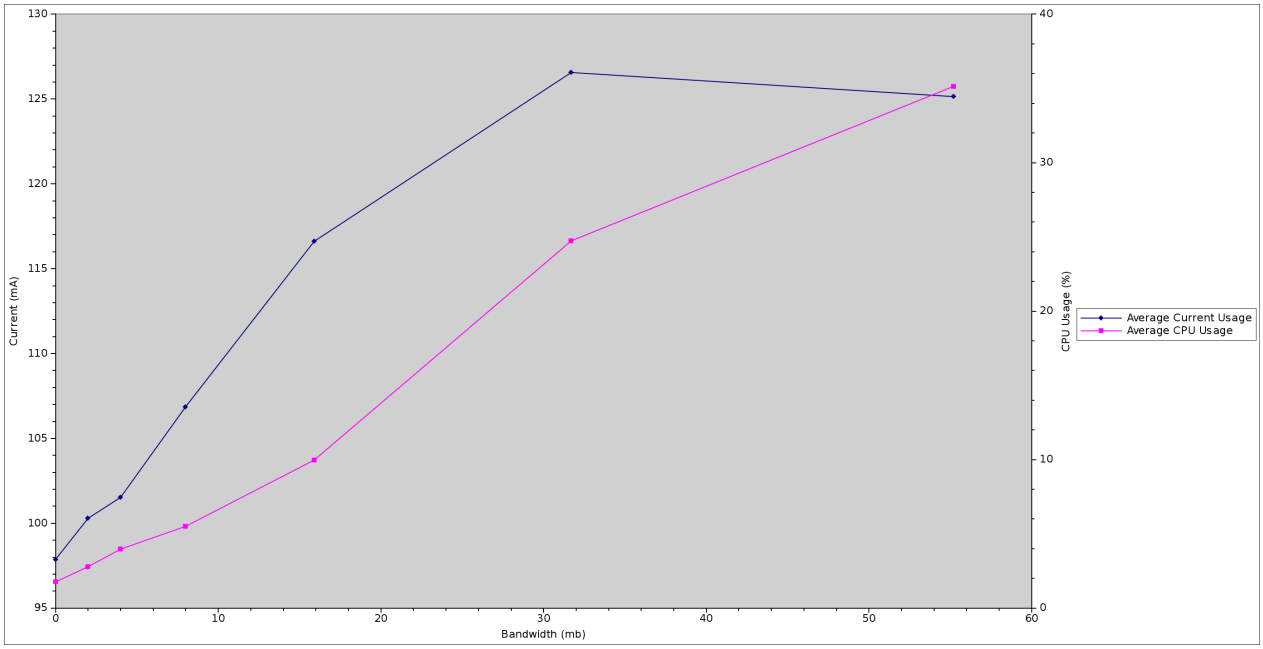


Figure 6.7.: TL-WR703N Wireless WPA2-PSK Receiving Data

When WPA2-PSK is enabled the maximum speed the internal wireless interface was able to receive at is 55.2Mbps with a current draw of 125mA and a CPU usage of 35%. The client was sending at 63.6Mbps which results in a packet loss of 13%. Figure 6.4 shows the TL-WR703N uses 109mA at 35% CPU usage this indicates the internal wireless interface was drawing 16mA of current.

When the TL-WR703N was receiving at 31.7Mbps the client was sending at 62Mbps which is 0.83% packet loss. The current was 127mA and the CPU usage was 25%. When the CPU usage is at 25% the TL-WR703N draws 107mA of current. This means the internal wireless interface is drawing 20mA of current.

This is a current drop of 4mA from 31.7Mbps to 55.2Mbps. Again like the unencrypted test the CPU usage is increasing because of iperf but the overall maximum received bandwidth is 55.2Mbps which is slower than the 56Mbps.

6.1.2.3. Routing data wireless

The following graphs show the TL-WR703N routing data between two TL-WR703Ns using the internal wireless interface and the Edimax wireless USB dongle. One of the TL-WR703Ns is the iperf server and connected to the Edimax wireless interface and the other one is the iperf client and connected to the internal wireless interface.

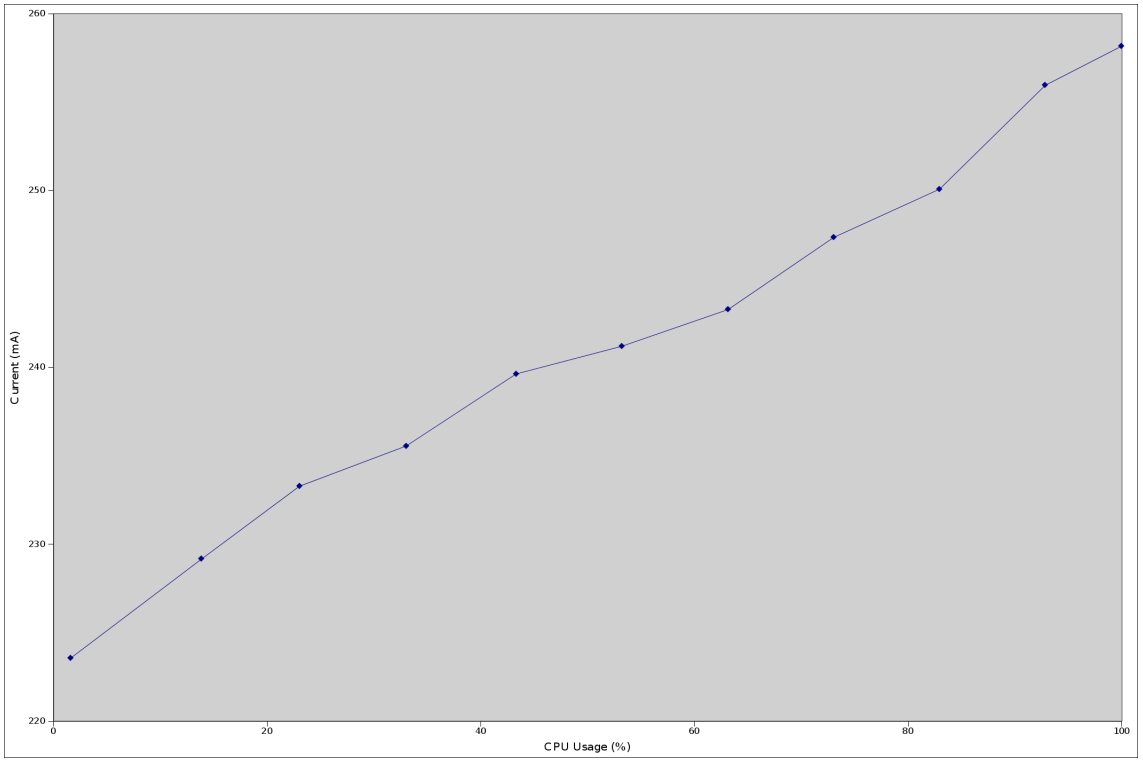


Figure 6.8.: TL-WR703N Wireless Open Routing CPU Usage Percentage

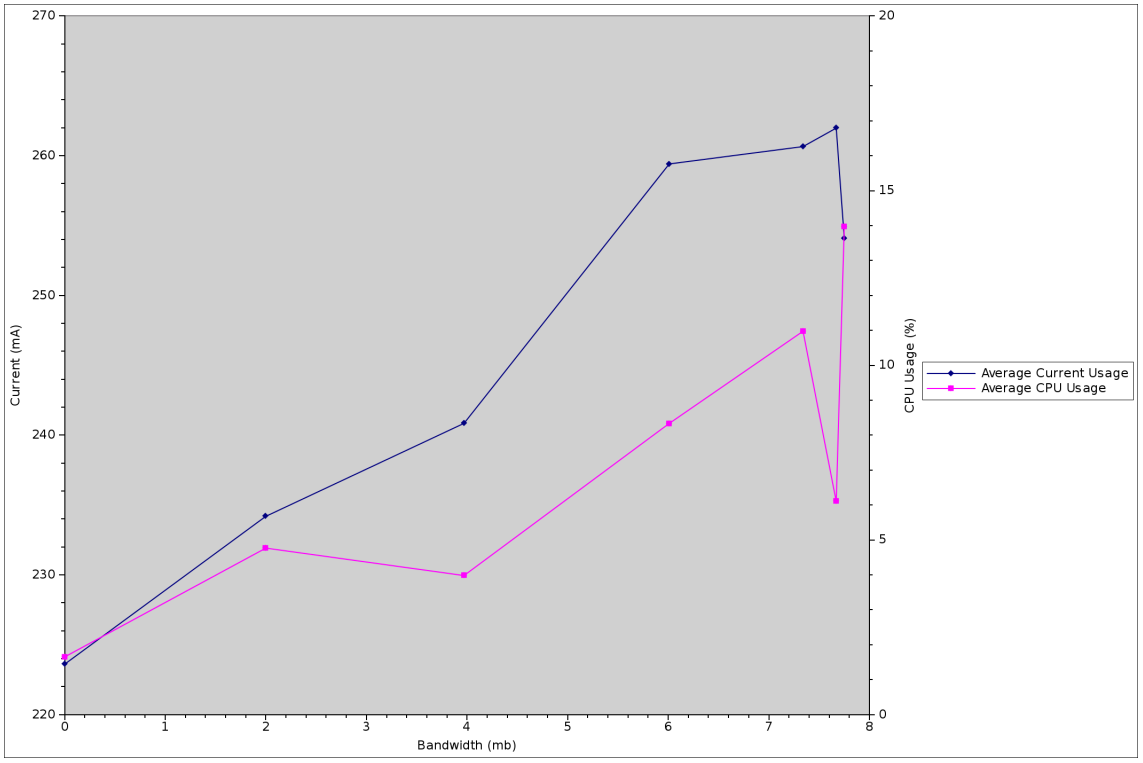


Figure 6.9.: TL-WR703N Wireless Open Routing Between Two Interfaces

The highest bandwidth which reached the server was 7.75Mbps when the client was sending at 8Mbps with a packet loss of 2.8%. The current draw was 254mA and the CPU usage was 14%. Figure 6.8 shows the TL-WR703N uses 229mA at 14% CPU usage. This means both

of the wireless interfaces are using 25mA of current.

The second highest bandwidth was 7.67Mbps when the client was set to 64Mbps with a packet loss of 88%. The current draw was 261mA and the CPU usage was 6%. The TL-WR703N draws 225mA at 6%. Both of the wireless interfaces are drawing 36mA.

The CPU usage dropped from 11% at 7.34Mbps to 6% at 7.67Mbps and increased back to 14% at 7.75Mbps. This is different compared to the sending and receiving graphs because the higher the bandwidth usually means the higher the CPU usage. The CPU usage flickering could indicate how many packets are being routed from one interface to another. There is an opportunity for packets to be lost in the air between the two pairs of wireless interfaces.

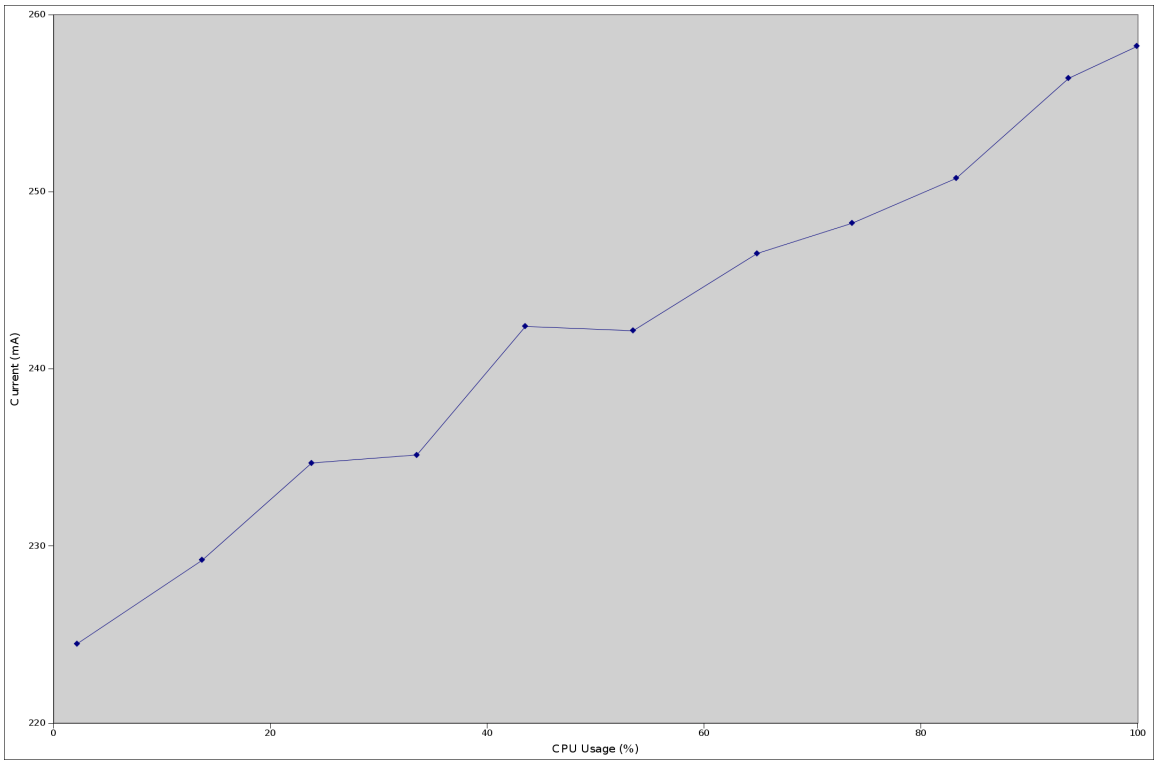


Figure 6.10.: TL-WR703N Wireless WPA2-PSK Routing CPU Usage Percentage

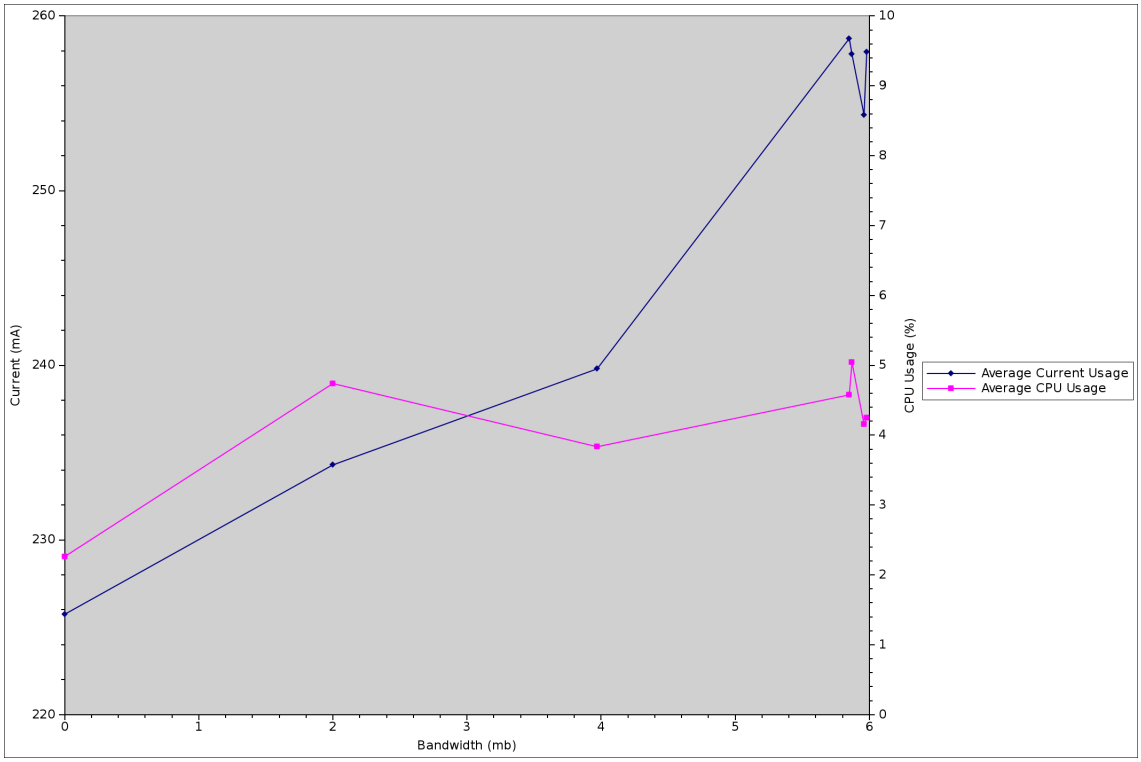


Figure 6.11.: TL-WR703N Wireless WPA2-PSK Routing Between Two Interfaces

When WPA2-PSK is enabled the maximum bandwidth reaching the server is 5.98Mbps when the client is sending at 32Mbps which is a packet loss of 82%. The current usage is 258mA and the CPU usage is 4%. Comparing the CPU usage with figure 6.10 shows the TL-WR703N draws 225mA of current at 4% CPU usage.

This means both the wireless interfaces are drawing 33mA. The next highest bandwidth is 5.96Mbps when the client sending at 64Mbps which indicates a packet loss of 92%. The current draw is 254mA and has a CPU usage of 4%. Both of the wireless interfaces are drawing 29mA.

6.1.2.4. Sending data Ethernet

The next graphs show the TL-WR703N sending data to another TL-WR703N using the Ethernet interface.

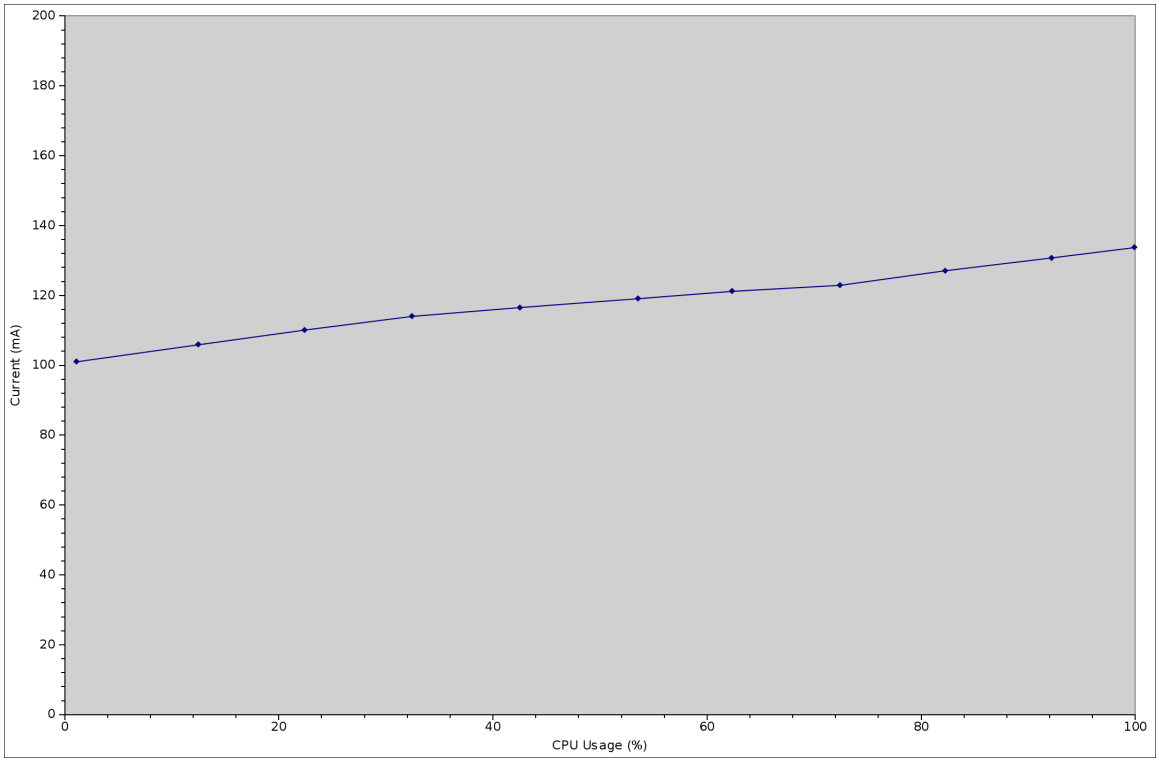


Figure 6.12.: TL-WR703N Ethernet CPU Usage Percentage

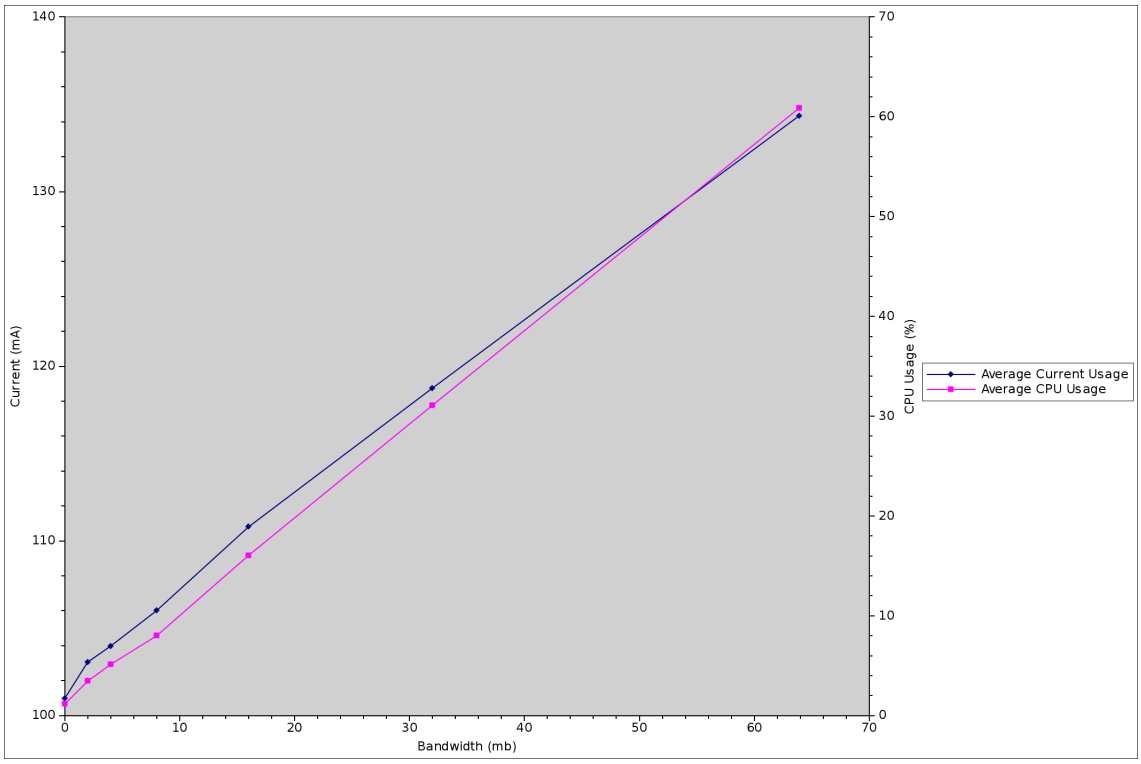


Figure 6.13.: TL-WR703N Ethernet Sending Data

Figure 6.13 shows that at 63.9Mbps the current draw was 134mA and the CPU usage was 60%. The TL-WR703N uses 120mA at 20% CPU usage which is shown by figure 6.12. The Ethernet interface was using 14mA of current when transferring at 63.9Mbps. At 32Mbps

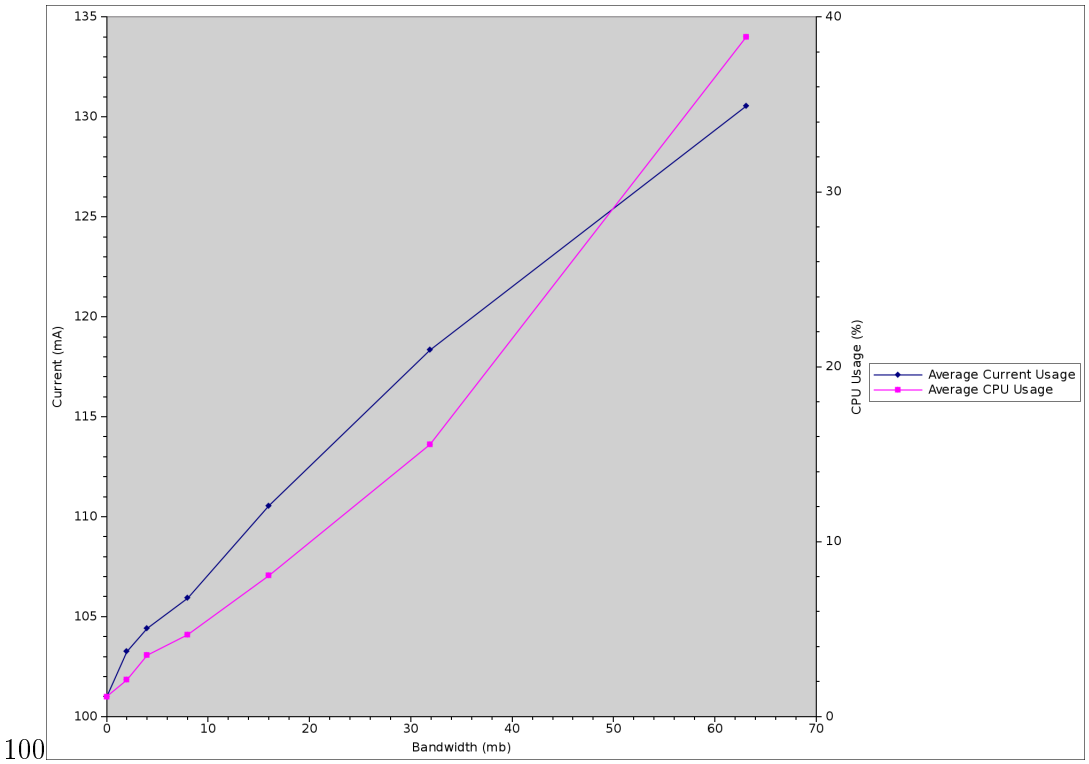
the CPU usage was 31% and the current draw was 119mA.

The TL-WR703N draws 113mA at 31% CPU usage which means the Ethernet interface was drawing 6mA at 32Mbps. When the Ethernet interface was transferring at 16Mbps the current draw was 111mA with a CPU usage of 16%. At 16% CPU usage the TL-WR703N draws 107mA. This indicates the Ethernet interface is drawing 4mA of current. The current difference between 16Mbps and 32Mbps is 2mA and between 32Mbps and 63.9Mbps is 8mA.

The 4Mbps bandwidth current usage is 104mA and the CPU usage is 5%. The TL-WR703N uses 104mA at 5% CPU usage. This means at 4Mbps the CPU is causing the current usage and the Ethernet interface isn't drawing more than its idle current.

6.1.2.5. Receiving data Ethernet

The next graph shows the TL-WR703N receiving data from another TL-WR703N via the Ethernet interface.



100

Figure 6.14.: TL-WR703N Ethernet Receiving Data

At 61.9Mbps the client was sending at 63.6Mbps and the current usage was 130mA with a CPU usage of 38%. This is shown by figure 6.14. Figure 6.12 shows the TL-WR703N uses 115mA at 38% CPU usage. This means the Ethernet interface is using 15mA of current to receive at 63.1Mbps.

When the Ethernet interface was receiving at 31.9Mbps the current draw was 118mA with a CPU usage of 16%. At 16% CPU usage the TL-WR703N uses 107mA. This means the Ethernet interface is using 11mA of current to receive at 31.9Mbps.

At 16Mbps the current usage was 111mA and the CPU usage was 8%. The TL-WR703N draws 104mA at 8% CPU usage which means the Ethernet interface is drawing 7mA to receive at 16Mbps.

6.2. Raspberry Pi

6.2.1. Idle Average Current Usage

The Raspberry Pi was tested by connecting a powered USB hub and then attaching network interfaces to the hub. The current monitor then monitored the current while the network interfaces are connected to a network but not sending or receiving data.

This gives an idea of the current draw while the network interfaces are idle compared to the Raspberry Pi with no network interface attached or enabled. The idle current tests ran for 100 seconds with readings taken every 100 milliseconds. The current readings were averaged and the result produced figure 6.15.

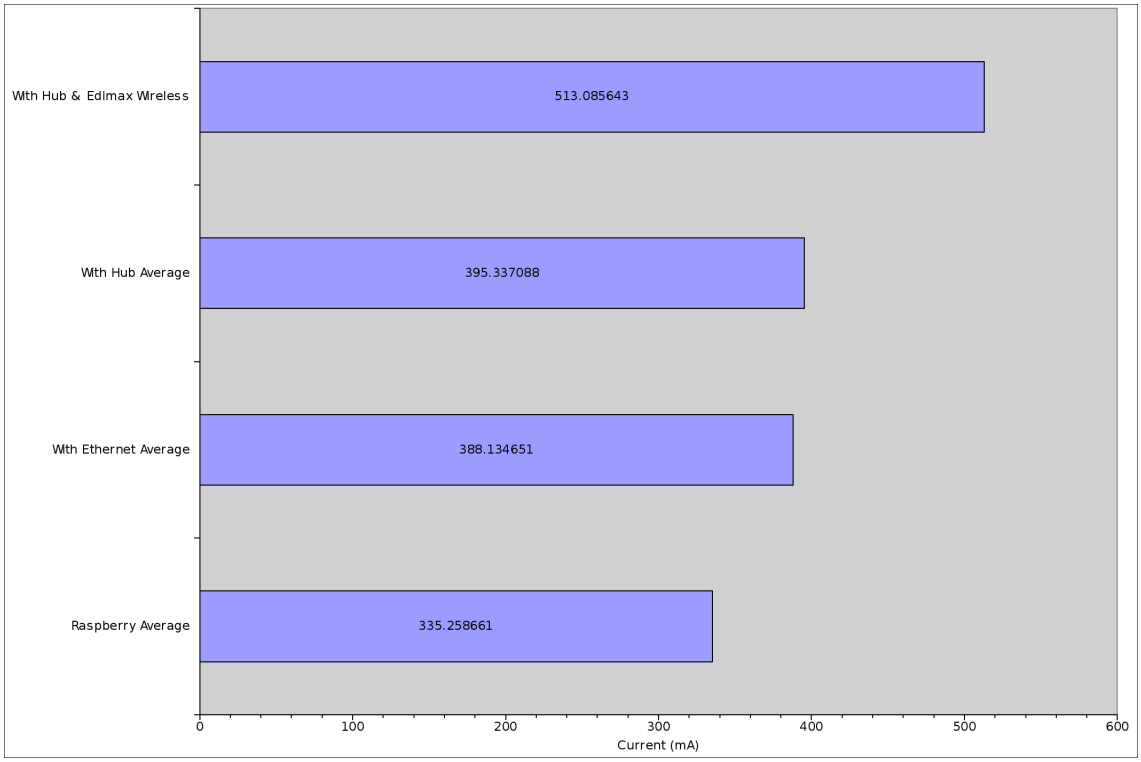


Figure 6.15.: Raspberry Pi Idle Average Current Usage

Figure 6.15 shows the idle average current for the Raspberry Pi is 335mA with no network interfaces connected or enabled. When the powered hub is attached, the current draw increases to 395mA which means the hub is drawing 60mA. With the Edimax wireless USB dongle attached to the hub the current increases to 513mA.

This is an increase of 118mA when compared with the hub reading which means the Edimax wireless USB dongle is drawing 118mA of current. With just an Ethernet cable attached the current draw is 388mA which is an increase of 53mA over the idle Raspberry Pi reading. This means the Ethernet interface is drawing 53mA.

6.2.2. Scenario Tests

The Raspberry Pi was tested in the similar scenarios to the TP-Link TL-WR703N. Taking into account the Raspberry Pi has no internal wireless interface and requires a powered USB hub. The scenarios are routing packets between two Edimax EW-7711UAn USB wireless dongle, sending and receiving data using one Edimax wireless USB dongle and sending and receiving data from the Ethernet interface.

The CPU usage is also being monitored because iperf is being used to generate the data being sent and received. The bandwidth was be doubled each time up to 64Mbps. The transport protocol used is UDP because iperf enforces the protocol when the bandwidth is set.

Each wireless scenario was tested both with and without encryption using the WPA2-PSK encryption scheme. Both of the Edimax wireless USB dongles were configured to use the Wireless N standard with a theoretical maximum bandwidth of 72Mbps.

Each test ran for 100 with a reading taken every 100 milliseconds. The network interface will be sending or receiving data for 60 seconds starting at 20 seconds and finishing at 80. This is the same setup as the TL-WR703N and will produce the same amount of readings and provide a graph showing a sudden increase and drop in current when the network interface is active.

The CPU usage graphs are produced in the same manner as the TL-WR703N with cpulimit being used to limit the amount of CPU usage the countdown application can use. The graphs also include the idle current of the network interface.

Unfortunately there was a strange anomily when the CPU tests were being executed which caused the CPU usage to be 20% higher after the 60% mark which means at 60% CPU limiting the CPU usage was recorded at 80%.

When running the wireless scenario tests the Raspberry Pi became very unstable and would often crash. The reading from the test points was 4.75V which is the low boundary for the test points voltage range. If the Raspberry Pi was to start performing lots of tasks the voltage would drop below the boundary and possible cause the Pi to crash. A sudden surge of current caused by connecting or disconnect the wireless dongle or the Ethernet cable could also cause the Pi to crash.

6.2.2.1. Sending data wireless

The following set of graphs show the Raspberry Pi sending data to a TL-WR703N using the Edimax wireless USB dongle.

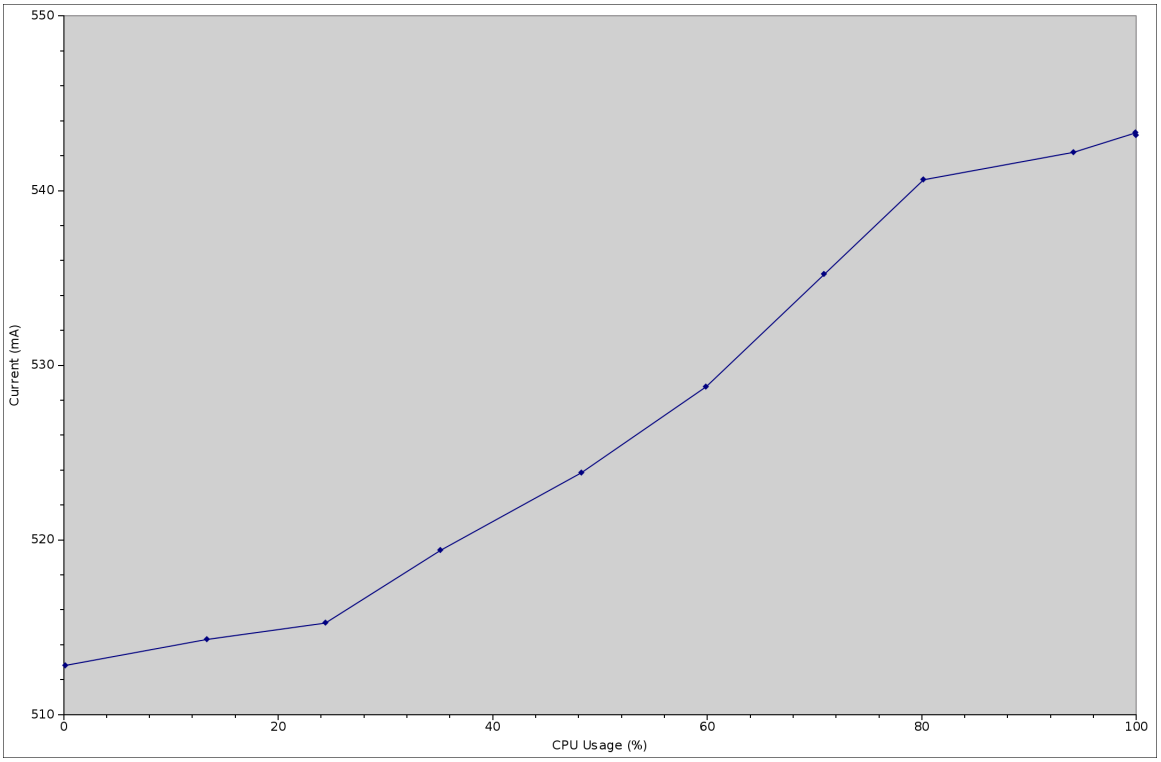


Figure 6.16.: Raspberry Pi Wireless Open CPU Percent Current Usage

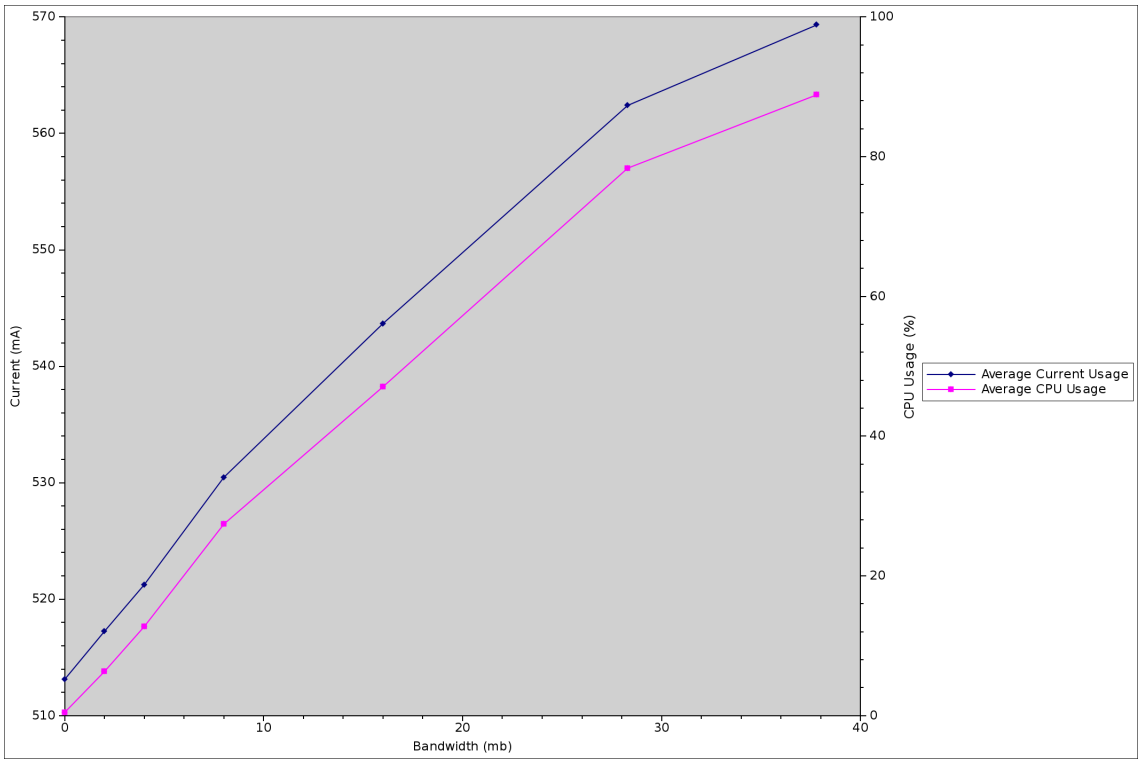


Figure 6.17.: Raspberry Pi Wireless Open Sending Data

Figure 6.17 shows the maximum bandwidth the wireless interface could transmit at is 37.8Mbps with a current draw of 569mA and a CPU usage of 89%. Comparing the CPU usage with figure 6.16 shows the current draw of the Raspberry Pi when the CPU usage is at

89% is 542mA. This means the wireless interface was drawing 27mA to transfer at 37.8Mbps.

At 28.3Mbps the CPU usage was 78% with a current draw of 562mA. The Raspberry Pi's current draw at 78% CPU usage is 539mA. This means the wireless interface was drawing 23mA at 28.3Mbps.

When the wireless interface was transferring at 16Mbps the current draw was 544mA with a CPU usage of 47%. The Raspberry Pi draws 523mA at 47% CPU usage which means the wireless interface was drawing 21mA.

The difference between 16Mbps and 28.3Mbps was 2mA and between 28.3Mbps and 37.8Mbps is 4mA. This is a small increase even so the bandwidth was doubled each time. This shows most of the current was consumed by the CPU running iperf to generate data to send.

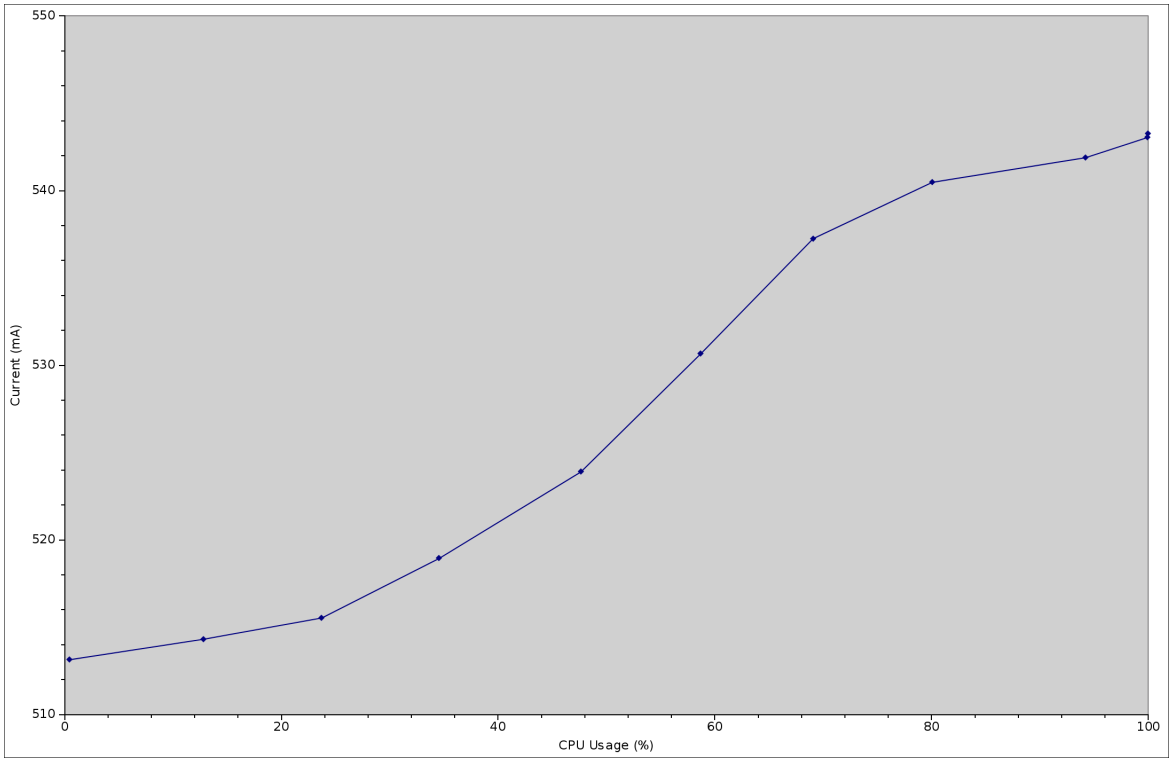


Figure 6.18.: Raspberry Pi Wireless WPA2-PSK CPU Percent Current Usage

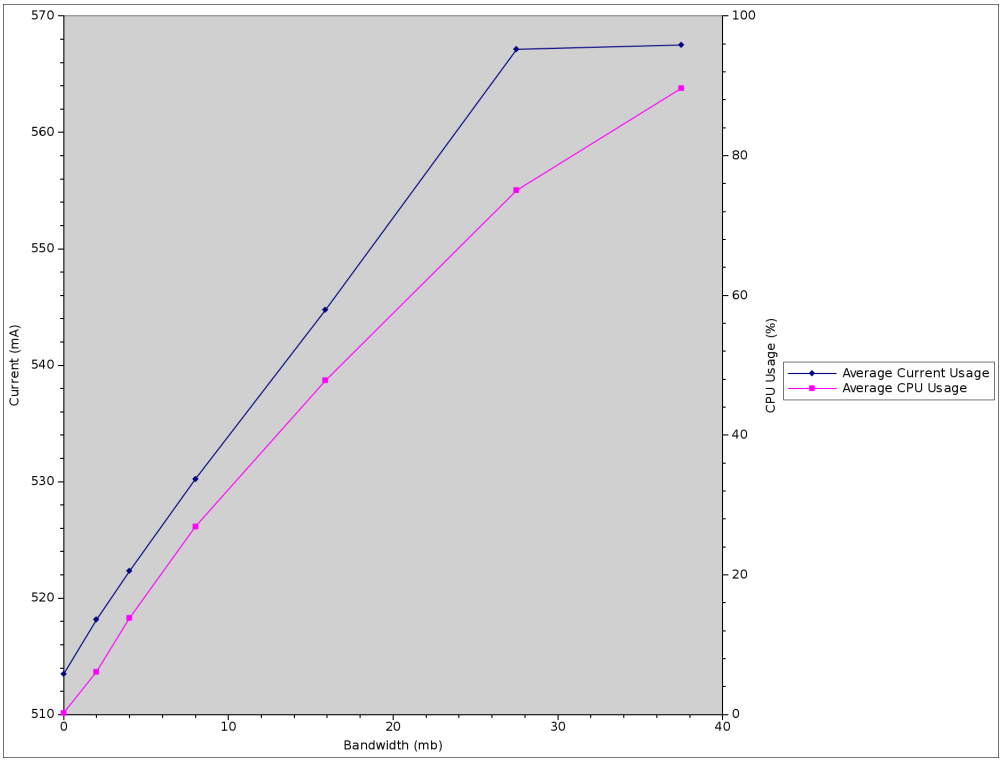


Figure 6.19.: Raspberry Pi Wireless WPA2-PSK Sending Data

Figure 6.18 shows the maximum transfer speed with WPA2-PSK enabled is 37.5Mbps. The current draw was 568mA and the CPU usage was 90%. The Raspberry Pi uses 541mA at 90% CPU usage this was shown by figure 6.18. The wireless interface is using 27mA to transfer at 37.5Mbps.

When the wireless interface was transferring at 27.5Mbps the current draw is 567mA and the CPU usage was 75%. At 75% CPU usage the Raspberry Pi draws 539mA of current which means the wireless interface is drawing 28mA of current.

At 15.9Mbps the CPU usage was 48% and the current draw was 548mA. When the Raspberry Pi was at 48% CPU usage the current draw is 524mA which means the wireless interface is drawing 24mA.

The difference between 15.9Mbps and 27.5Mbps current draw is 4mA and between 27.5Mbps and 37.5Mbps was a 1mA decrease from 27.5Mbps. The overall current draw leveled of at 27.5Mbps while the CPU usage keeps climbing.

Comparing this with unencrypted version of the test shows the WPA2-PSK encryption hasn't added an overhead to the Raspberry Pi. The maiximum bandwidth was only 0.2Mbps different from the unencrypted test with 1% more CPU usage.

6.2.2.2. Receiving data wireless

The next graphs show the Raspberry Pi running the iperf server receiving data from a TL-WR703N running the iperf client using the Edimax wireless USB dongle.

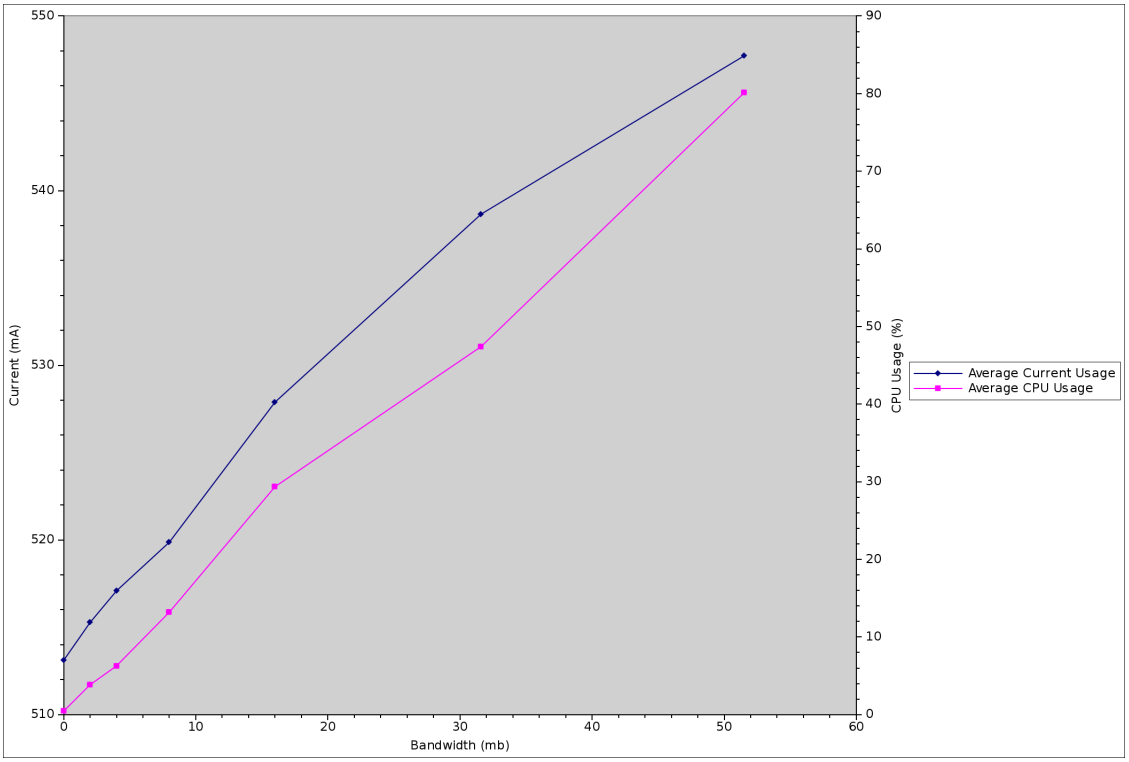


Figure 6.20.: Raspberry Pi Wireless Open Receiving Data

Figure 6.20 shows the maximum received bandwidth was 51.5Mbps with a current draw of 548mA and a CPU usage of 80%. Comparing the CPU usage to figure 6.16 shows the Raspberry Pi uses 541mA at 80%. This means the wireless interface is using 7mA of current to receive at 51.5Mbps.

At 31.6Mbps the current draw is 539mA and the CPU usage was 47%. The Raspberry Pi draws 523mA at 47% CPU usage which means the wireless interface is drawing 16mA.

At 16Mbps the CPU usage is 29% and the current draw was 528mA. When the Raspberry Pi is at 29% the current draw was 517mA. This means the wireless interface is using 11mA. The wireless interface draws the most current at 31.6Mbps and then drops by 9mA at 51.5Mbps.

This is because the packet loss was 19% at 51.5Mbps and only 1.2% at 31.6Mbps. This means the wireless interface isn't receiving packets for a larger proportion of the time the network interface was active. This resulted in a lower current draw average.

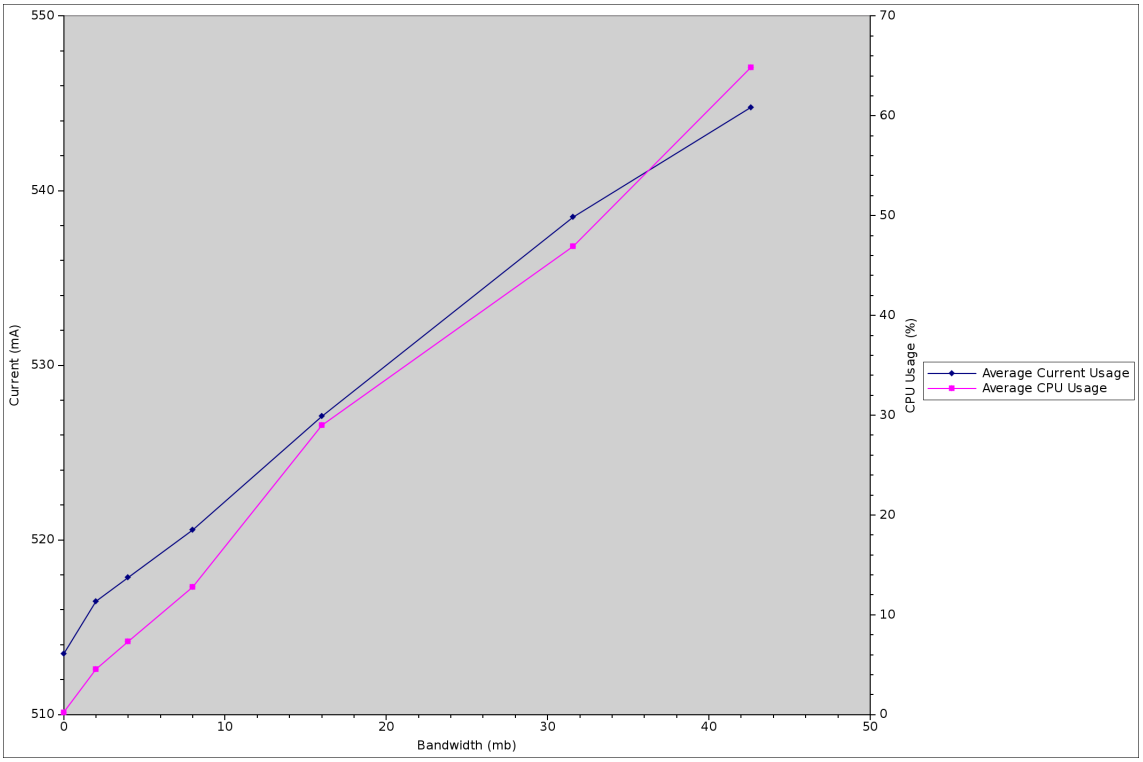


Figure 6.21.: Raspberry Pi Wireless WPA2-PSK Receiving Data

When WPA2-PSK is enabled the maximum received bandwidth is 42.6Mbps shown by figure 6.21. The current draw was 545mA and the CPU usage was 65%. When the CPU usage is compared with figure 6.18. The Raspberry Pi draws 535mA at 65% which means the wireless interface draws 10mA at 42.6Mbps.

When the wireless interface was receiving at 31.6Mbps the CPU usage was 47% and the current draw was 538mA. At 47% CPU the Raspberry Pi draws 524mA which means the wireless interface was drawing 14mA of current.

At 16Mbps the current draw is 527mA and the CPU usage was 29%. When the Raspberry Pi was using 29% of the CPU the current draw is 517mA. This means the wireless interface was drawing 10mA of current. The packet loss at 42.6Mbps was 33% compared with 1.5% at 31.6Mbps. This caused a lower average current draw even so it was receiving faster than 31.6Mbps.

6.2.2.3. Routing data wireless

The following set of graphs show the Raspberry Pi routing traffic between two Edimax wireless USB dongles with a TL-WR703N connected on each one of them. One of the TL-WR703Ns is the iperf server and the other one is the iperf client.

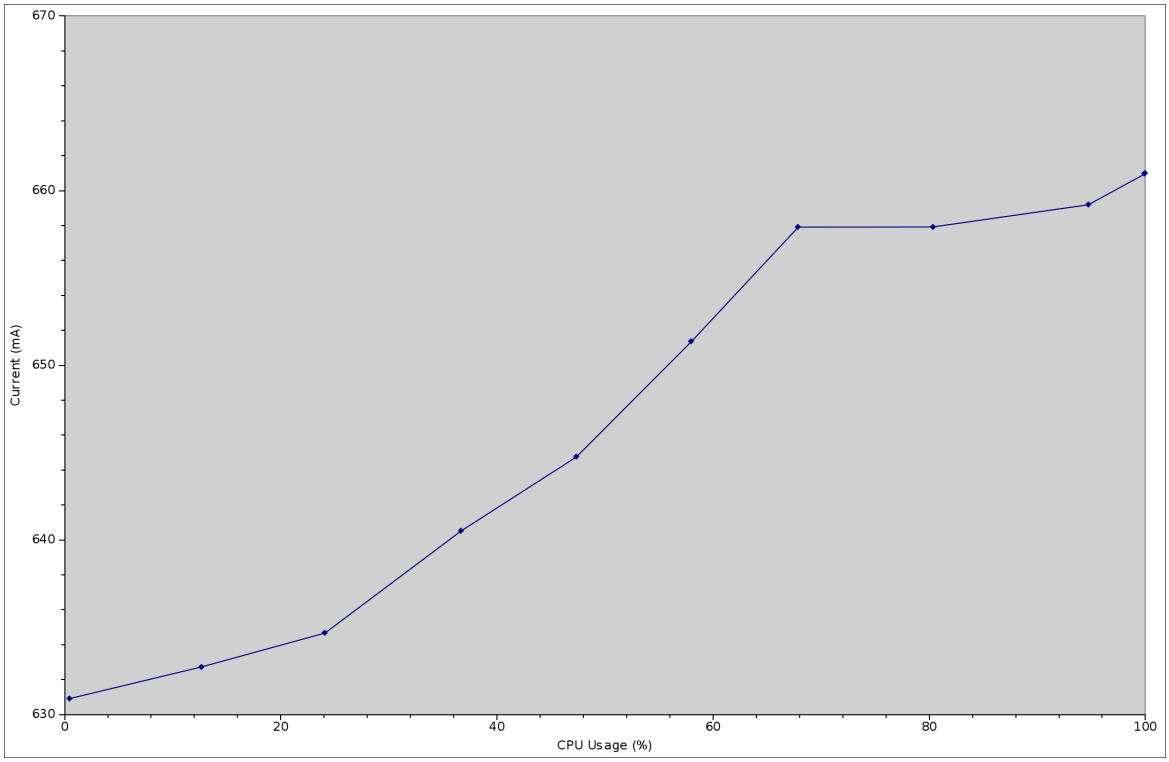


Figure 6.22.: Raspberry Pi Wireless Open Routing CPU Usage Percentage

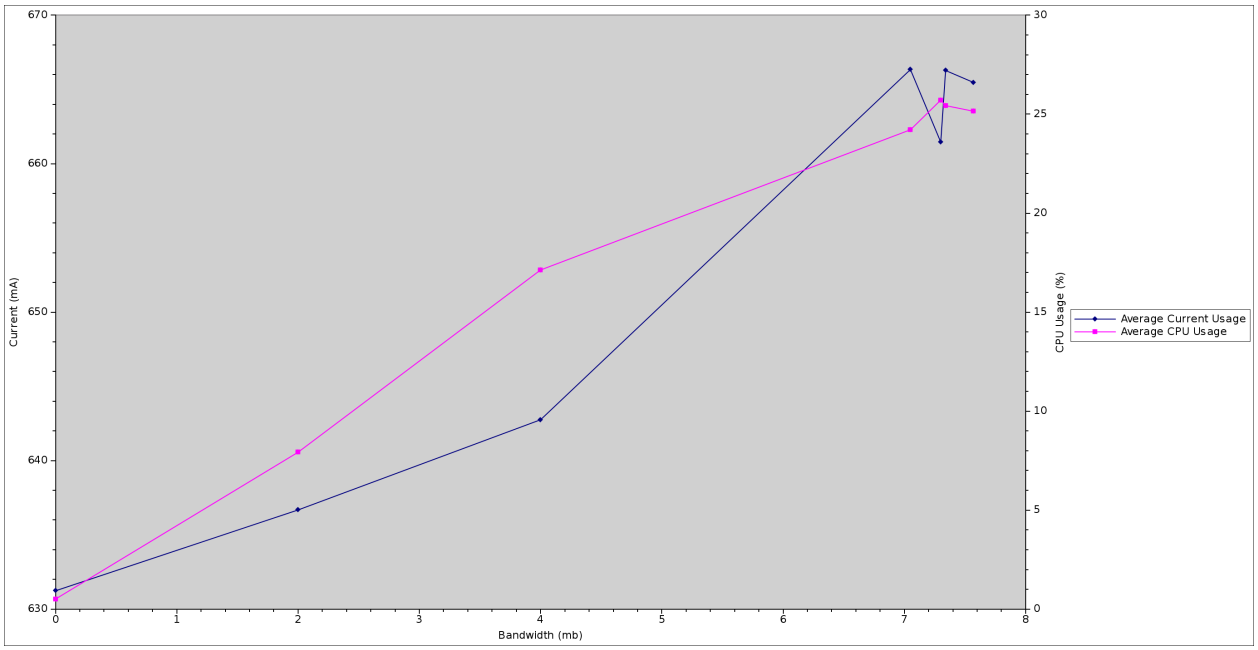


Figure 6.23.: Raspberry Pi Wireless Open Routing Between Two Interfaces

Figure 6.23 shows the maximum bandwidth to the server was 7.57Mbps when the client was sending at 32Mbps with a CPU usage of 25% and a current draw of 665mA. The Raspberry Pi's current draw at 25% CPU usage was 635mA which is shown by figure 6.22. This means both of the wireless interfaces are drawing 30mA.

The next fastest bandwidth achieved was 7.34Mbps when the client was sending at 16Mbps. The current draw was 666mA with a CPU usage of 25%. This indicates that both the wire-

less interfaces are drawing 31mA. At 7.3Mbps the client was sending at 8Mbps and the current draw was 661mA and the CPU usage was 26%. When the CPU usage was at 26% the current draw is 636mA which indicates that both of the wireless interfaces are drawing 25mA.

When the client was sending at 64Mbps the throughput of the Raspberry Pi was 7.05Mbps. The CPU usage was 24% and the current draw was 666mA. At 24% CPU usage the Raspberry Pi draws 635mA of current. This means both of the wireless interfaces are drawing 31mA.

The current drops by 6mA between 7.05Mbps and 7.3Mbps even so the bandwidth was higher this is because the client was sending at 8Mbps with a packet loss of 9.9% compared with the client sending at 64Mbps with a packet loss of 90%. The slower transmission rate from the client allowed the Raspberry Pi to receive and route packets to the server which is why there was an increase in CPU usage.

The current drop indicates there was less work on both of the wireless interfaces part as less data was being sent over the same amount of time.

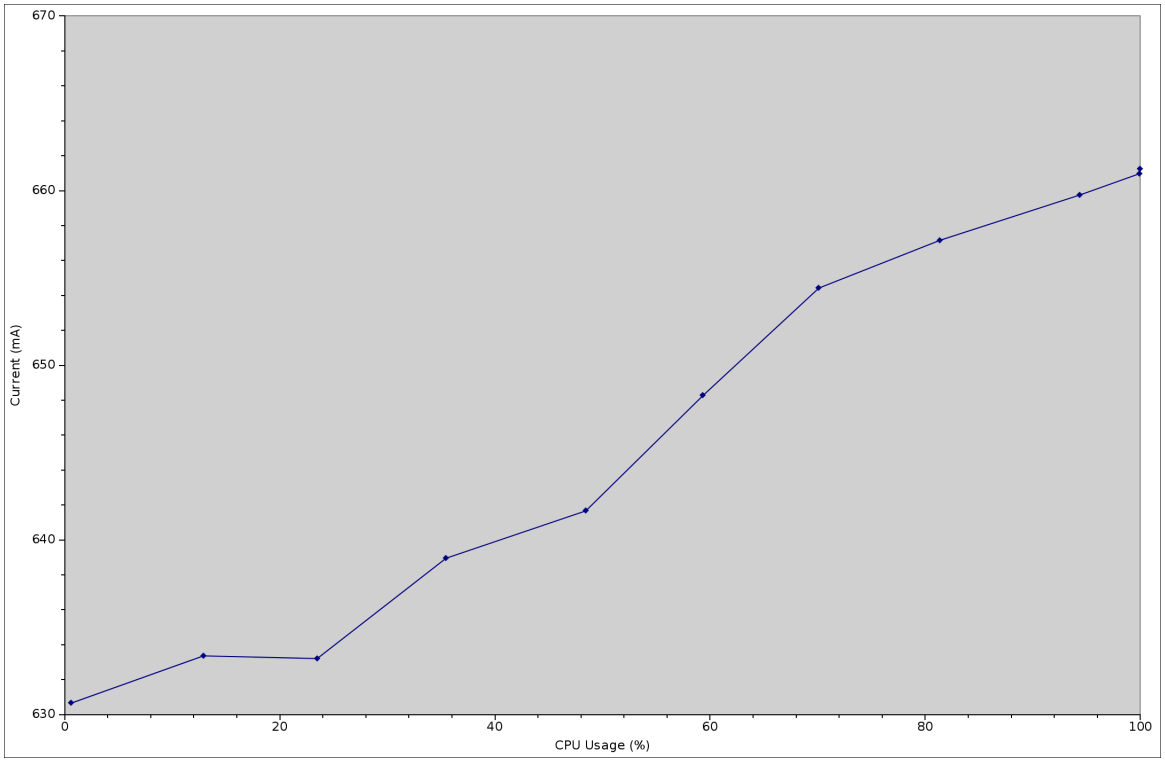


Figure 6.24.: Raspberry Pi Wireless WPA2-PSK Routing CPU Usage Percentage

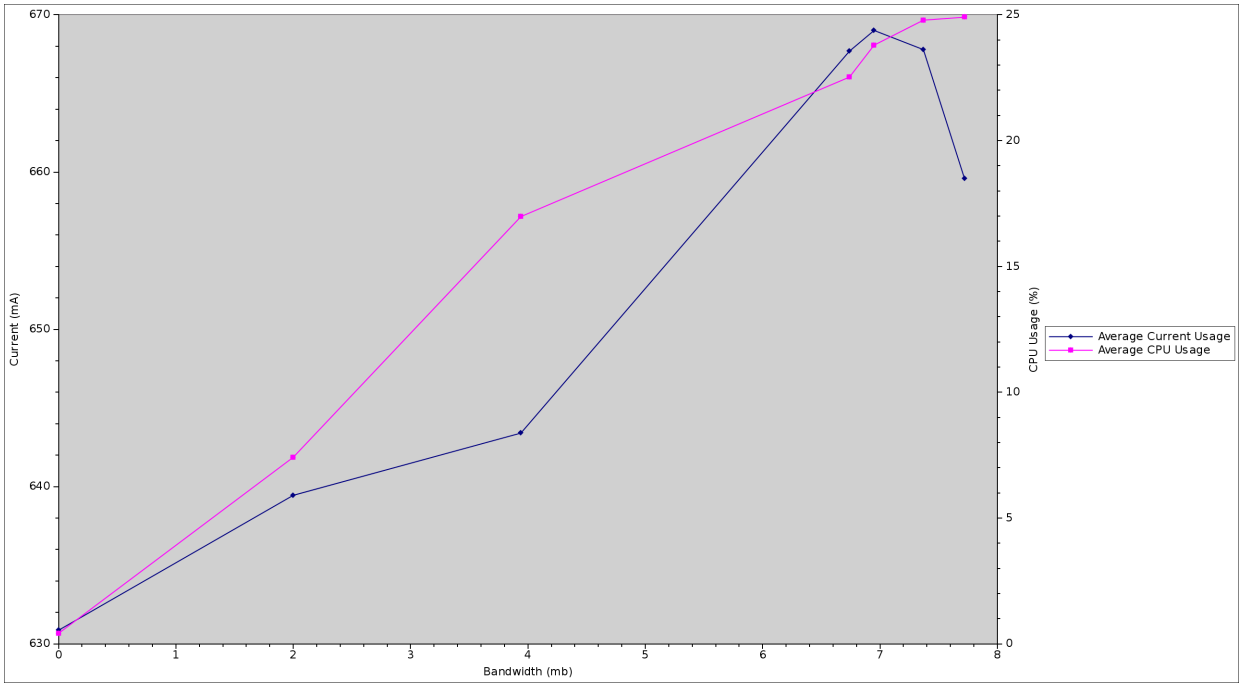


Figure 6.25.: Raspberry Pi Wireless WPA2-PSK Routing Between Two Interfaces

Figure 6.25 shows the maximum bandwidth achieved with WPA2-PSK enabled is 7.72Mbps when the client was sending at 8Mbps. This indicates a packet loss of 4.9%. The current draw was 660mA and the CPU usage was 25%. Figure 6.24 shows the Raspberry Pi draws 634mA when the CPU usage is 25% which indicates that both of the wireless interfaces draw 26mA.

At 7.35Mbps of throughput the client was sending at 32Mbps. The CPU usage was 25% and the current draw was 668mA. When the Raspberry Pi was using 25% of the CPU the current draw was 634mA. This means both of the wireless interfaces are drawing 34mA.

When the client was sending at 64Mbps the router was routing traffic at 6.95Mbps with a current draw of 669mA and uses 24% of the CPU. At 24% CPU usage the Raspberry Pi draws 633mA which means both of the wireless interfaces are drawing 36mA.

At 6.74Mbps the current draw was 668mA and the CPU usage was 23%. The Raspberry Pi draws 633mA at 23% CPU usage which means both of the wireless interfaces draw 35mA of current. The current drops at 7.72Mbps because the client was sending at 8Mbps. At 7.35Mbps the client was sending at 32Mbps. Receiving at 8Mbps is less work for the wireless interface than receiving at 32Mbps.

6.2.2.4. Sending data Ethernet

The next set of graphs show the Raspberry Pi sending data over the Ethernet interface to a TL-WR703N.

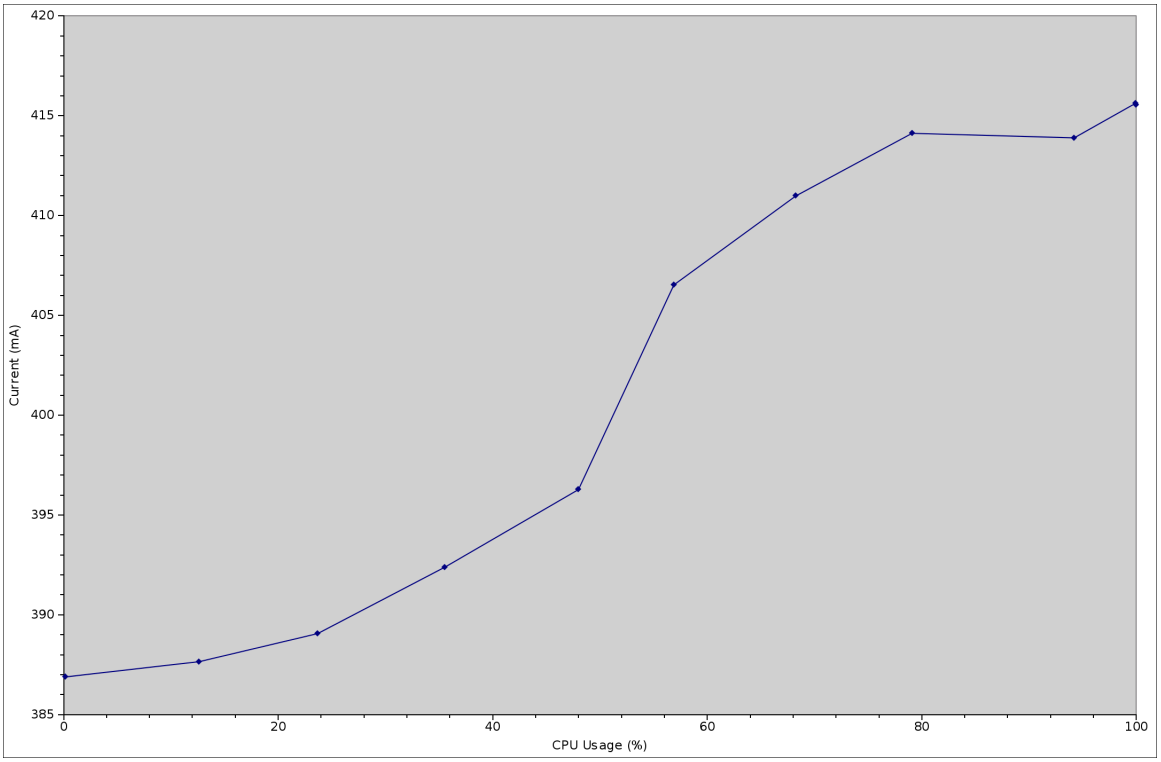


Figure 6.26.: Raspberry Pi Ethernet CPU Usage Percentage

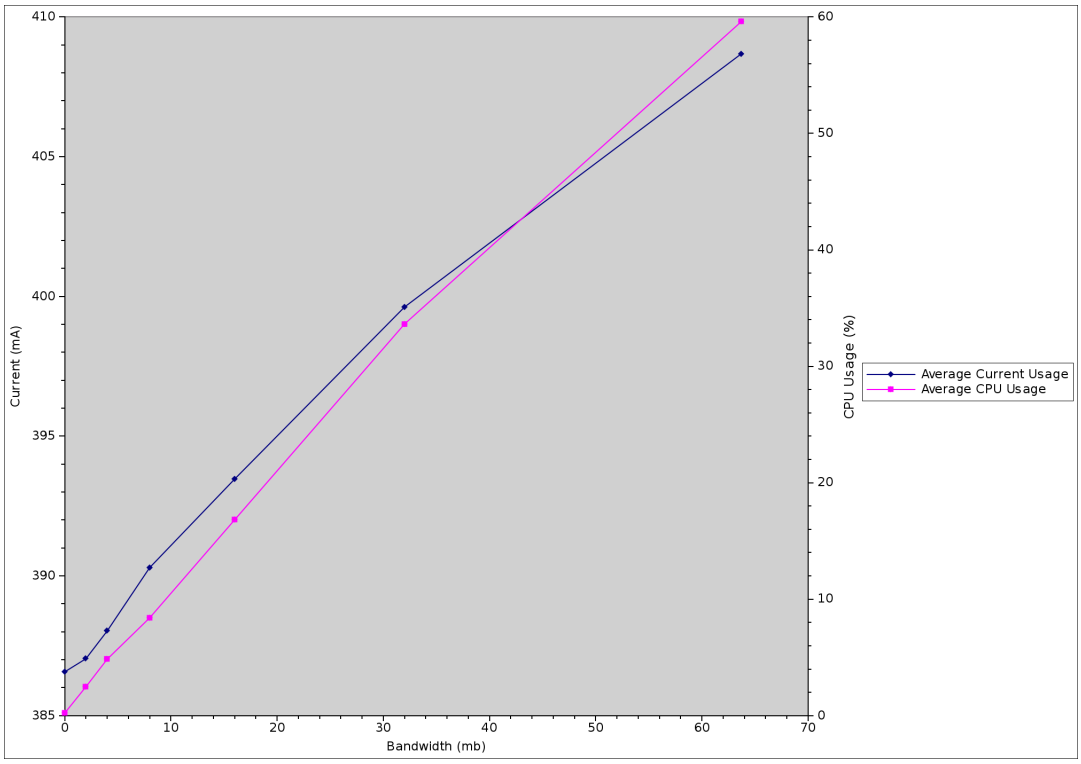


Figure 6.27.: Raspberry Pi Ethernet Sending Data

Figure 6.27 shows the Raspberry Pi draws 409mA with a CPU usage of 60% at 63.7Mbps. The Raspberry Pi draws 408mA of current when the CPU usage was 60%. This is shown by figure 6.26. This means the Ethernet interface was only drawing 1mA of current.

At 32Mbps the CPU usage was 34% and the current draw was 400mA. The current draw at 34% was 392mA which shows the Ethernet interface was drawing 8mA of current. At 16Mbps the current draw was 393mA with a CPU usage of 17%. The CPU usage of the Raspberry Pi at 17% was 388mA which shows the Ethernet interface was drawing 5mA of current.

6.2.2.5. Receiving data Ethernet

The next graph shows the Raspberry Pi receiving data from a TL-WR703N via the Ethernet interface.

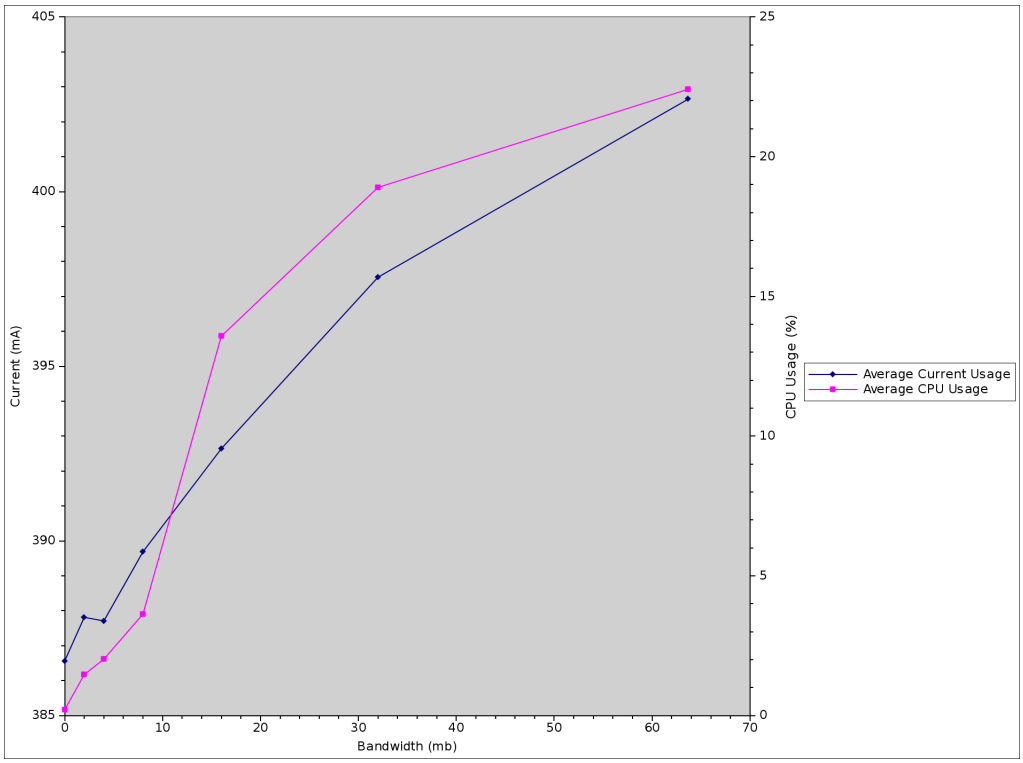


Figure 6.28.: Raspberry Pi Ethernet Receiving Data

Figure 6.28 shows that at 63.7Mbps the current draw was 403mA and the CPU usage was 22%. The Raspberry Pi draws 389mA at 22% CPU usage which means the Ethernet interface was drawing 14mA.

When the Ethernet interface was receiving at 32Mbps the CPU usage was 19% and the current draw was 398mA. When the Raspberry Pi was at 19% CPU usage the current draw was 388mA. The Ethernet interface was drawing 10mA.

At 16Mbps the current draw was 393mA and the CPU usage was 14%. When the Raspberry Pi's CPU was at 14% the current draw was 388mA. The Ethernet interface draws 5mA of current at 16Mbps.

6.3. Comparing the platform results

After preforming the current monitoring and CPU usage tests with both platforms in different scenarios I have determined the TP-Link TL-WR703N uses less current overall than the Raspberry Pi. The idle current for the TL-WR703N was 67mA compared to 335mA for the Raspberry Pi which is an increase of 268mA over the TL-WR703N.

When the platforms are used in a wireless scenario the Raspberry Pi requires a powered USB hub which adds an overhead of 66mA plus the current cost of the wireless USB dongle which was 118mA. This adds a total of 184mA on top of the idle current which creates a total of 519mA current draw. The TL-WR703N integrated wireless interface draws 30mA when its idle which is a total of 97mA when the wireless is active and the platform is idle.

The Edimax wireless USB dongle managed a top data rate of 37.5Mbps and drew 27mA on top of the 118mA idle which is a total of 145mA when sending data. Using figure 6.3 shows that if the TL-WR703N was sending at 37.5Mbps the entire platform would draw 278mA comparing the 47% CPU usage to figure 6.2 shows the TL-WR703N draws 113mA at 47% CPU usage which means the wireless interface is drawing 165mA. Adding the 30mA idle current to the 165mA brings the total to 195mA.

This shows the TL-WR703N's internal wireless interface draws 50mA more than the Edimax wireless USB dongle. With the current draw of the required powered USB hub for the Raspberry Pi this brings the total current draw to 211mA which is 16mA over the TL-WR703N.

During the receiving data scenario the Edimax wireless USB dongle managed to receive at 51.5Mbps with a current draw of 7mA. When this is added to the 118mA idle current this becomes 125mA for the Edimax wireless USB dongle and with the hub this is increased to 191mA.

The TL-WR703N draws 124mA when transferring at 51.5Mbps with a CPU usage of 32% as shown by figure 6.6. When the TL-WR703N is at 32% CPU usage the current draw is 110mA shown by figure 6.2. This means the wireless interface is drawing 14mA to receive at 51.5Mbps and with the idle current the total is 44mA. Compared against the Edimax wireless USB dongle this is a difference of 81mA.

If we ignore the idle current draw we can see the TL-WR703N's internal wireless interface is doing more work to receive the data at 51.5Mbps then the Edimax wireless USB dongle. This is because the Edimax wireless USB dongle has a high gain wireless antenna as opposed to the TL-WR703N's antenna which is a copper strip on the PCB.

The maximum transfer rate of the TL-WR703N's internal wireless interface while sending data was 62.5Mbps compared with the 37.5Mbps of the Edimax wireless USB dongle connected to the Raspberry Pi. This is a decrease of 24.7Mbps.

The limit of the Edimax wireless USB dongle transfer speed could be caused by the USB 2.0 bus even so the theoretical maximum bandwidth of USB 2.0 is 480Mbps. However the wireless interface is connected through two USB hubs, the one built into the Raspberry Pi and the powered USB hub.

USB hubs add an extract delay due to the recalculating of the time left before the next Start-Of-Frame packet.[28] This delay wouldn't exist if the wireless interface was directly connected to the Raspberry Pi's USB host controller built into the BCM2835 SoC. The TL-WR703N's internal wireless interface is built directly into the SoC and most likely connected to the processor using a proprietary bus.

When WPA2-PSK is enabled the maximum transfer speed of the TL-WR703N's internal wireless interface achieved while sending was 61.4Mbps with a CPU usage of 61%. On the Raspberry Pi the maximum transfer speed of the Edimax wireless USB dongle while sending with WPA2-PSK enabled was 37.5Mbps with a CPU usage of 90%. This is a CPU usage increase of 8% on the TL-WR703N and 1% on the Raspberry Pi compared with 62.5Mbps at 53% CPU usage on the TL-WR703N and 37.5Mbps at 89% on the Raspberry Pi. At 31.9Mbps on the TL-WR703N the CPU usage was 57%.

The Raspberry Pi at 27.5Mbps has 75% CPU usage. On the TL-WR703N this is a increase of 11% and on the Raspberry Pi it's a decrease of 2% compared with 46% CPU usage at 31.9Mbps on the TL-WR703N and the 78% CPU usage at 28.3Mbps on the Raspberry Pi. Based on these results WPA2-PSK adds an overhead to the TL-WR703N and not the

Raspberry Pi.

7. Evaluation & Conclusion

7.1. Evaluation

This project originally intended to produce a power management system which could by used on mobile devices which run Linux. The idea was to gather information about the existing power consumption of the TP-Link TL-WR703N and the Raspberry Pi and then make changes and investigate if any improvements have been made. It was a requirement that the power management system was intelligent and had the ability to inform the platform on how much power it was consuming.

When I researched current monitoring I couldn't find an existing device which met the requirements. Instead I came across the battery monitoring ICs which provide an interface to access the readings. When developing the current monitor I ran into problems with the Raspberry Pi such as the logic level difference between the GPIOs of the TL-WR703N and the Raspberry Pi.

Once the current monitor was finished I then developed the logging software which I ran into minor issues with such as the 'rw' file descriptor not reading or writing to the device after so many reads and writes have been performed which took a small amount of time to debug. The logging software was designed to run at boot so the logging process would begin without requiring user intervention as this would increase the amount of current being used. This is due to the fact I would be connected via the Ethernet or wireless interface.

The next step was to test different peripherals such as a 3G USB modem, USB webcam, USB temperature and humidity sensor and USB wireless dongles. However when attempting to connect a USB 3G modem to the Raspberry Pi I discovered the modem would connect and disconnect from Pi. After researching the Raspberry Pi's issues with USB peripherals I discovered the 100mA USB device limit and the two voltage test points which indicate if the Raspberry Pi is being powered correctly.

A powered USB hub was purchased to allow devices which require more than 100mA of current to be connected but modification had to be made to the current monitor to allow it to include the draw of the hub and the devices connected to the hub.

Investigating the Raspberry Pi's test point voltage revealed the power supply that was currently being used wasn't providing enough voltage because it couldn't provide the current for the Raspberry Pi. After trying a number of different power supplies the Maplin 4 port USB charger was found which could provide enough voltage to power the Raspberry Pi.

When I began testing with the current monitor I realised that I needed to add the ability to monitor the CPU usage to the logging application. This is because the activity of the CPU causes the platforms to draw more current. It's also required to separate how much current individual peripherals are drawing from the current cost of the CPU using them.

The test I decided to preform using the current monitor show the platforms as nodes in a network sending and receiving data and their ability to route traffic from one network to another. This relates the results back to the Backpack Router project which is the origin of this project.

Unfortunately I only managed to test the wireless USB dongle. If I had more time I would begin testing the other peripherals in different scenarios such as streaming a USB webcam over the wireless interface. The networking tests required the platforms to be configured with the ability to support wireless USB dongles which meant compiling the kernel modules.

With the TP-Link TL-WR703N this wasn't an issue as all the required packages are built into OpenWRT. However with the Raspberry Pi the kernel is quite old. This required the compat-wireless package which allows the latest wireless kernel drivers to work with older kernels by compiling a compatibility module along with the modded wireless kernel modules.

During testing the Raspberry Pi struggled with the wireless USB dongles as any slight change to the setup would cause the Raspberry Pi to crash. This includes connecting an Ethernet cable or removing a wireless USB dongle. The only solution to reset the Raspberry Pi was to disconnect it from the power supply.

The test points voltage on the Raspberry Pi was 4.75V which is the low boundary of the acceptable voltage range. Connecting and disconnecting the Ethernet cable or peripheral would have caused a current surge which would bring the voltage down further.

When gathering CPU usage at different usage amounts on the Raspberry Pi I ran into an issue with the CPU usage slowly increase to a point when it became 10% more than the limit. This behavior didn't occur on the TL-WR703N as the usage was always 3% more than the limit.

Some of the CPU usage results on the Raspberry Pi have been corrupted as they have been recorded as "NaN". This didn't occur on the TP-Link TL-WR703N.

The Ethernet sending data scenario results on the Raspberry Pi has an invalid result. The current the Ethernet card was drawing at 63.7Mbps is 1mA but at 32Mbps it was 8mA. This doesn't follow the usual trend as the Ethernet card sends at a higher bandwidth the current increases. I would except a 1mA increase over the idle current at 2Mbps. If I had more time I would test the scenario again and investigate how this could have happened.

When I was researching power management on Linux for the platforms I discovered there wasn't any implementations to allow for the Raspberry Pi and TP-Link TL-WR703N to enter into a low power state. However there was a method of automatically placing USB devices into suspend. This feature was disabled by default on the platforms but if I had more time I would have enabled and tested if this feature could have been used reliable to reduce power consumption.

7.2. Future Work

This project didn't reach the end goal of producing a power management system. Instead its set the foundations for a project to start developing the means to intelligently monitor power consumption based on the current draw.

A future project could build upon this work and develop a battery management system based upon the current monitor. This battery management system would allow the platform to monitor the charge and discharge state of the battery. The platform can then make a decision as to whether or not to turn the connected peripherals off.

The battery will be trickle charged using solar panels which means the battery management system has to be able to monitor the current going into and out of the battery. Another DS2438 IC is required to monitor the charging current ans will be connected to the charging side of the circuit. This will allow the battery management system to provide two readings. These readings allow the platform to monitor how much current is going into the battery and how much current is being consumed powering the platform. The battery management system would only require 1 GPIO as more than one 1-Wire bus device can be on the same bus.

If the platform is consuming more current than the solar panel is placing into the battery then the battery isn't being charged and all the charging current is going towards the platform. The platform can then change its behavior and reduce its current consumption to less than the charging current this would allow some of the current to go towards the battery.

Research could be preformed into finding ways to handle the lack of power management implementations for the platforms and finding a way of placing connected peripherals into a low power state.

7.3. Conclusion

One of the primary objectives of the project was to reduce the power consumption of the TP-Link TL-WR703N and the Raspberry Pi using power management interfaces built into Linux. However after discovering both platforms are missing the necessary Linux support to utilise these interfaces. Reducing power consumption would require more research and time than this project can cover.

The current monitor provides the ability to record the current consumption of the TL-WR703N and the Raspberry Pi which allows the comparison of both platforms to be preformed. However it needs further development before it can be used as part of a power management solution.

The tests I decided to preform on both platforms give a basic idea of how each platform preforms as a node on a network. However I didn't design or run any tests using different USB peripherals.

After comparing the power consumption of the TL-WR703N and the Raspberry Pi based on the preformed tests. I have determined the TL-WR703N is best suited for the Backpack Router project.

The idle current of the TL-WR703N is lower than the Raspberry Pi. The TL-WR703N has a built in wireless interface which is a lot faster than using a wireless USB dongle on the Raspberry Pi. The internal wireless interface on the TL-WR703N requires more power to send data than the Edimax wireless USB dongle which was used on the Raspberry Pi.

The TL-WR703N does require more CPU usage to transfer at higher data rates with WPA2-PSK enabled. This could cause a problem if the router was preforming other tasks while transferring on an encrypted connection.

The Raspberry Pi became unstable when a wireless USB dongle was connected which means it would be unreliable to be used as the Backpack Router.

Although this project didn't meet the main objective of producing a power management system it has created some fundamental foundations into exploring the power consumption of embedded platforms. The current monitor is compatible with any platform that has support for the 1-Wire bus and a spare GPIO. The logging application requires Linux and would need compiling for the architecture.

Bibliography

[1] DS2438 Datasheet - http://datasheets.maxim-ic.com/en/ds/DS2438.pdf - 06/09/12

[2] RPi Low Level Peripherals - http://elinux.org/RPi_Low-level_peripherals - 06/09/12

[3] (TL-WR703n / GPIO / Misc) AR9331 pinouts? - Post 5 - https://forum.openwrt.org/viewtopic.php?pid=166716 - 06/09/12

[4] LKML: Markus Franke: [PATCH 2/2] w1: Disable irqs during 1-wire bus operations, extend 1-wire reset pulse - https://lkml.org/lkml/2012/3/16/466 - 06/09/12

[5] Linux-Kernel Archive: [PATCH] w1-gpio: add external pull-up enable callback - http://lkml.indiana.edu/hypermail/linux/kernel/0905.0/01519.html - 06/09/12

[6] #1458 (configure kmod params in config files) - OpenWrt - https://dev.openwrt.org/ticket/1458 - 06/09/12

[7] Using the DS2438 Battery Monitor on Crossbow's Stargate - Vladislav Petkov - http://users.soe.ucsc.edu/~cintia/batmon.pdf - 06/09/12

[8] UMA Wiki - Routers - http://uma-wiki.network-mobility.org/index.php5?title=Routers - 06/09/12

[9] 11km Wireless Link to a Remote Site - http://wireless.ictp.it/school_2006/wiki/pmwiki~45.html - 06/09/12

[10] Developing a Portable aP Wireless LAN Kit - Panagiotis Georgopoulos, Ben McCarthy and Christopher Edwards

[11] Advanced Configuration and Power Interface Wikipedia - http://en.wikipedia.org/wiki/Advanced_Configuration_and_Power_Interface - 06/09/12

[12] Advanced Power Management Wikipedia - http://en.wikipedia.org/wiki/Advanced_power_manageme - 06/09/12

[13] Embedded Linux power management - http://free-electrons.com/services/power-management/ - 06/09/12

[14] Android Power Management - http://www.netmite.com/android/mydroid/development/pdk/docs/pow - 06/09/12

[15] PM Quality Of Service Interface - http://lxr.free-electrons.com/source/Documentation/power/pm_qos_interface.txt - 06/09/12

[16] Wireless usage of pm-qos - http://linuxwireless.org/en/developers/Documentation/pm-qos - 06/09/12

[17] Dynamic power save - http://linuxwireless.org/en/users/Documentation/dynamic-power-save - 06/09/12

[18] Understanding DTIM Period - http://www.juniper.net/techpubs/software/junos-security/junos-security10.0/junos-security-swconfig-wlan/wlan-ax411-access-point-dtim-period-understanding.html - 06/09/12

[19] Power Savings on IEEE-802.11 - http://linuxwireless.org/en/developers/Documentation/ieee80211/pow savings - 06/09/12

[20] Runtime Power Management Framework for I/O Devices - http://www.mjmwired.net/kernel/Documentation/power/runtime_pm.txt - 06/09/12

[21] Power Management For USB - http://www.mjmwired.net/kernel/Documentation/usb/power-management.txt - 06/09/12

[22] RPI Troubleshooting - http://elinux.org/R-Pi_Troubleshooting - 06/09/12

[23] DS2762 Datasheet - http://datasheets.maxim-ic.com/en/ds/DS2762.pdf - 06/09/12

[24] DS2745 Datasheet - http://datasheets.maxim-ic.com/en/ds/DS2745.pdf - 06/09/12

[25] BQ2631Datasheet - http://www.ti.com/lit/ds/symlink/bq26231.pdf - 06/09/12

[26] Stargate Datasheet - http://bullseye.xbow.com:81/Products/Product_pdf_files/Wireless_pdf/Stargat - 06/09/12

[27] Understanding and Using Cyclic Redundancy Checks with Maxim iButton Products - http://www.maximintegrated.com/app-notes/index.mvp/id/27 - 06/09/12

[28] USB 2.0, Hi-Speed USB FAQ - http://www.everythingusb.com/hi-speed-usb.html - 06/09/12

[29] OpenWRT Wiki TP-Link TL-WR703N - http://wiki.openwrt.org/toh/tp-link/tl-wr703n - 06/09/12

[30] Raspberry Pi Wikipedia - http://en.wikipedia.org/wiki/Raspberry_Pi - 06/09/12

A. Linux SoC Power Management Project Proposal