WiCM Software by Schafer Corporation

A Technical Guide by Nathan Wharton

Overview

The WiCM software by Schafer Corporation runs on single board computers and turns them into nodes to build a mesh network. The links in a mesh network can be either wired or wireless over a variety of frequencies. Each node can have multiple links so it can act as a bridge between different meshes.

Each node can provide a LAN to attach Ethernet devices (wired and/or wireless) that are not running WiCM software. Standard Internet Protocol routing is used to route between the local networks, so the attached Ethernet devices do not need to know about how the mesh works. The attached devices merely use the nodes as standard IP gateways.

In addition to Ethernet devices, a WiCM node can bring into a mesh a variety of sensors. The interfaces of sensors that have been integrated into a WiCM mesh so far have been USB, serial, i2c, SPI, CANbus, and analog voltage. This guide covers network topology, building of the software for a WiCM node, installation of the software to a single board computer, configuration of the software once it is on the single board computer, technical details of the WiCM software, and instructions on how to add sensors to a WiCM node.

Network Topology

All WiCM nodes are peers. There is no central node that manages routes or addresses. This means that the nodes require some preset way of telling each other apart. There is no DHCP server assigning mesh nodes IP addresses, so they have to be static. Also, since the nodes each have a possible LAN that they provide and all these LANs need to be able to talk to each other, the LANs also need to be statically assigned per node. To simplify this assignment, all of this information is calculated from the unique node ID. The node ID consists of 2 numbers, each between 1 and 255 inclusive. The first number is referred to as the group number (G), the second as the node number (N).

Any Ethernet interfaces that the node is going to use to mesh are assigned the IP address 172.x.G.N, x being between 16 and 31 inclusive. This gives the restriction that any certain node can have up to 16 mesh interfaces.

The LAN that the node provides for standard Ethernet devices is 10.G.N.0. The node takes 10.G.N.1 as its interface on the LAN, and provides DHCP addresses for 10.G.N.100 through 10.G.N.199. The node acts as the default gateway for the LAN, and the netmask of the LAN is 255.255.255.0. The IP addresses 10.G.N.2 through 10.G.N.99 are available for devices that need to be assigned static addresses.

Ethernet ports can be assigned to the LAN that the node provides. They can also be put in one of three other modes. They can be used to create a wired mesh. They can be used as access to a higher level network like the Internet. They can also be used to create a transparent Ethernet bridge over the mesh to another Ethernet port on another node. If an existing control system utilizes an Ethernet over radio connection, this connection can be replaced by a transparent Ethernet bridge over the mesh. This is the method used to control the iRobot PackBot over a WiCM mesh.

Serial ports on the nodes can be used either to attach sensors to the node or to provide a transparent serial port bridge over the mesh to another serial port on another node. If an existing control system utilizes a serial over radio connection, this connection can be replaced by a transparent serial bridge over the mesh. This is the method used to control the QinetiQ Dragon Runner over a WiCM mesh.

CANbus ports on the nodes can be used either to attach sensors to the node or to provide a transparent CANbus bridge over the mesh to another CANbus port on another node. If an existing control system utilizes a CANbus connection, this connection can be replaced by a transparent CANbus bridge over the mesh, but the deterministic properties of the CANbus are lost. This is the method used to control the HDT robotic arm over a WiCM mesh.

Building

WiCM runs on the Linux operating system, built under the OpenWRT distribution. The version of Linux is 2.6.32, and the version of OpenWRT is SVN-21880. WiCM is implemented as a package in OpenWRT. If OpenWRT runs on a certain platform, installing the WiCM package turns it into a WiCM node. OpenWRT supports many low resource platforms. WiCM has run on platforms with a 275 MHz processor and 16 MB of memory.

The OpenWRT build environment provides a cross compiler to build software to run on the node. It uses this compiler to build all the open source packages that the node needs to run. These packages are either libraries used by other packages, kernel modules used to interface to hardware, or executables. The mesh router OLSRD is an example of an OpenWRT package. It is an executable that handles setting linux routing tables based on what it hears on a network interface. It depends on a package called libpthread that provides threading capabilities to a process. OLSRD needs Ethernet interfaces to mesh, and these interfaces depend on kernel module packages to talk to the hardware. OpenWRT handles dependencies between the packages.

OLSRD manages the path and the hops to route data between different nodes in the mesh. OLSRD runs on every WiCM node and broadcasts and receives data from OLSRD running on other WiCM nodes. From these conversations, OLSRD modifies the operating system IP routing table. OLSRD does not move any TCP or UDP unicast data through the mesh. Once OLSRD has set up the routing table, standard IP routing is used to move the unicast data through the mesh.

Multicast data is moved through the mesh by OLSRD. OLSRD listens on the LAN of each node for multicast packets and moves them through the mesh to the LAN on other nodes. To prevent circular routing that might happen in the mesh by the forwarding of these multicast packets, the packets are not forwarded if the exact same packet has already been forwarded in the last 20 seconds. So, in order to request that all multicast packets from one LAN get to a LAN on another node, one must ensure that the packets are unique.

OpenWRT Packages

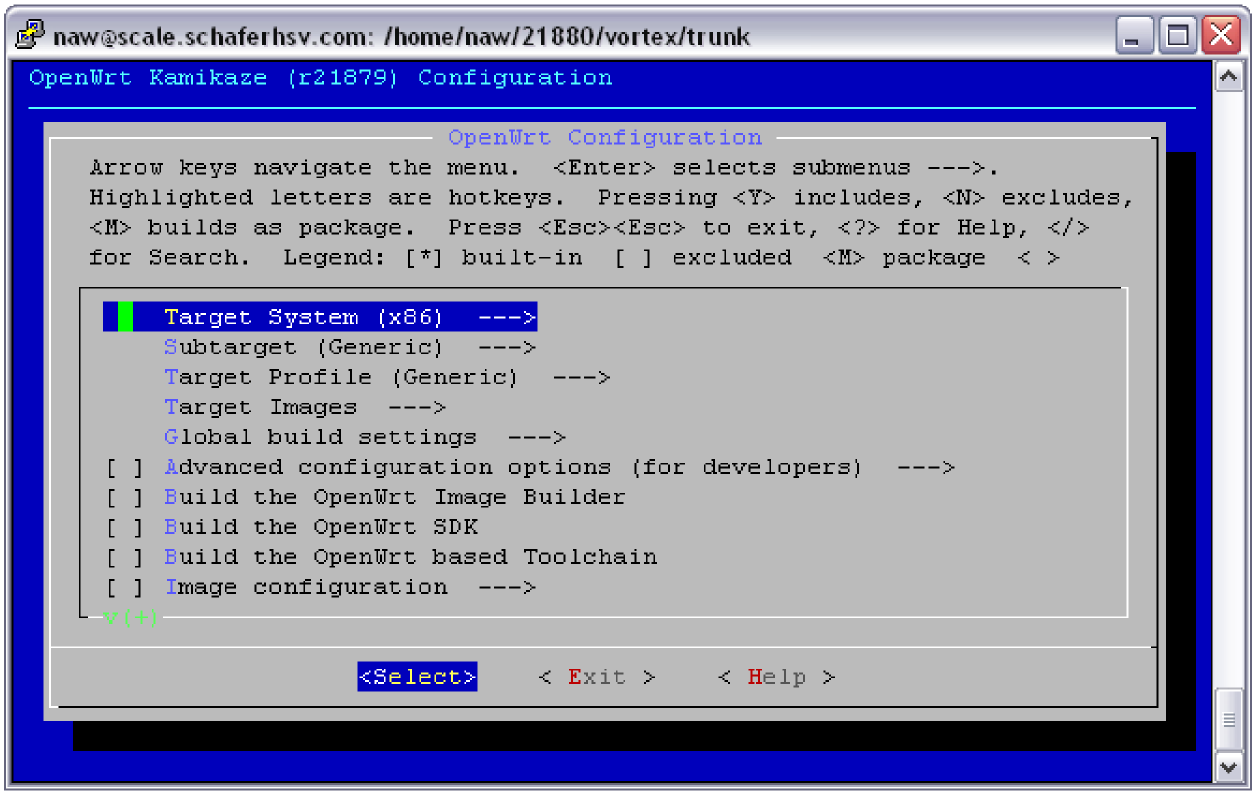
There are many packages even in a minimal OpenWRT install. Here is a sample of some of them and their purposes:

alsa-lib - 1.0.21a-1	Sound library
base-files - 49-r21879	Base files needed for any OpenWRT installation
bc - 1.06.95-1	Command line calculator used in shell scripts
busybox - 1.16.2-1	Basic linux commands like cp, ls, mkdir, grep, tar
dnsmasq - 2.55-1	Combination DHCP server and DNS relay
dropbear - 0.52-5	SSH Server for secure remote logins
e2fsprogs - 1.41.11-1	Programs for formatting and checking e2fs filesystem partitions
fdisk - 2.13.0.1-4	Parition editor
firewall - 2-6	Firewall scripts
gpsd - 2.90-1	GPS server to provide data from a locally connected GPS to the network
grub - 0.97-3	Unified bootloader used on x86 platforms
iperf-mt - 2.0.4-2	Network performance tester
iptables - 1.4.8-1	Executable to modify linux kernel firewall rules
kmod-3c59x - 2.6.32.14-1	Kernel module for 3com ethernet interfaces
kmod-8139too - 2.6.32.14-1	Kernel module for Realtek ethernet interfaces
kmod-e100 - 2.6.32.14-1	Kernel module for Intel ethernet interfaces
kmod-e1000 - 2.6.32.14-1	Kernel module for Intel gigabit ethernet interfaces
kmod-ipt-core - 2.6.32.14-1	Kernel module for linux firewall
kmod-madwifi - 2.6.32.14+r3314-4	Kernel module for Atheros wifi cards
kmod-natsemi - 2.6.32.14-1	Kernel module for National Semiconductor ethernet interfaces
kmod-ne2k-pci - 2.6.32.14-1	Kernel module for Tulip based ethernet interfaces
kmod-r8169 - 2.6.32.14-1	Kernel module for Realtek ethernet interfaces
kmod-sis900 - 2.6.32.14-1	Kernel module for SIS ethernet interfaces
kmod-sound-core - 2.6.32.14-1	Kernel module for sound
kmod-tg3 - 2.6.32.14-1	Kernel module for Broadcom Tigon-3 ethernet interfaces
kmod-tun - 2.6.32.14-1	Kernel module for ethernet tunneling
kmod-usb-acm - 2.6.32.14-1	Kernel module for USB modems such as the Verizon USB stick
kmod-usb-audio - 2.6.32.14-1	Kernel module for USB audio devices
kmod-usb-cm109 - 2.6.32.14-1	Kernel module for the audio device on a Roboard

kmod-usb-core - 2.6.32.14-1	Kernel module for all USB operations
kmod-usb-ohci - 2.6.32.14-1	Kernel module for OHCI USB interfaces
kmod-usb-serial - 2.6.32.14-1	Kernel module for all USB serial ports
kmod-usb-serial-cp210x - 2.6.32.14-1	Kernel module for the USB to serial chip on the Oceanserver compass
kmod-usb-serial-ftdi - 2.6.32.14-1	Kernel module for the USB to serial chip on Xbee radios
kmod-usb-serial-pl2303 - 2.6.32.14-1	Kernel module for the USB to serial chip on some GPS radios
kmod-usb-storage - 2.6.32.14-1	Kernel module for USB memory sticks
kmod-usb-uhci - 2.6.32.14-1	Kernel module for UHCI USB interfaces
kmod-usb2 - 2.6.32.14-1	Kernel module for EHCI USB interfaces (high speed USB)
lame - 398-2-3	MP3 encoder
lame-lib - 398-2-3	Library for MP3 encoder
libc - 0.9.30.1-49	Basic C Library
libgcc - 4.1.2-49	Basic Gnu C Library
libgsm - 1.0.13-1	Audio compression library
libmad - 0.15.1b-3	MP3 decoder library
libogg - 1.1.4-2	Audio compression library
libopenssl - 0.9.8o-1	Encryption library
libpcap - 1.0.0-2	Packet capture library
libpthread - 0.9.30.1-49	Thread library
librt - 0.9.30.1-49	Realtime library
libstdcpp - 4.1.2-49	Standard C++ library
libusb - 0.1.12-2	USB library for raw access to USB bus
libvorbis - 1.2.3-1	Audio compression library
madplay - 0.15.2b-3	MP3 decoder
maradns - 1.3.07.09-1	Lightweight DNS server
microcom - 1.02-1	Terminal emulator to talk to serial devices
minicom - 2.3-1	Terminal emulator to talk to serial devices
mtd - 13	Flash memory utilities
ntpdate - 4.2.6-4	Network Time Protocol client
olsrd - 0.6.0-1	Mesh routing daemon
olsrd-mod-bmf - 0.6.0-1	Plugin to mesh routing daemon to forward multicast packets
olsrd-mod-dot-draw - 0.6.0-1	Plugin to mesh routing daemon to draw network topology
olsrd-mod-dyn-gw-plain - 0.6.0-1	Plugin to mesh routing daemon to share internet gateway
olsrd-mod-httpinfo - 0.6.0-1	Plugin to mesh routing daemon to provide web page of status
openssh-sftp-server - 5.5p1-1	Plugin to dropbear to allow file copying over ssh
openssl-util - 0.9.8o-1	Utilities to manage SSL certificates
openvpn - 2.1.1-1	VPN provider
openvpn-easy-rsa - 2.1.1-1	Scripts for setting up SSL certificates for VPN

opkg - 528-1	Package manager
pciutils - 3.1.7-2	Utility to list PCI devices and information about them
phidget21 - 2.1.8.20110322-1	Library for accessing phidget sensors
run-firetable - 1	Executable to ignite air bag initiators wired to an Acessio board
scale-mesh - 1.1-1	Package containing scripts to set up interfaecs and run the WiCM mesh
socat - 1.7.1.3-3	Tool for plumbing data between various sources
tcpdump - 4.0.0-2	Tool for dumping packet information from an ethernet interface
uclibcxx - 0.2.2-3	Small C++ library
usbutils - 0.86-2	Utility to list USB devices and information about them
wireless-tools - 29-4	Tools for configuring Atheros radios
wpad-mini - 20100418-2	Daemon to provide WPA encryption to wireless clients
zlib - 1.2.3-5	Compression library

Most of these packages don't need to be selected, installed, or even configured manually due to the way OpenWRT handles them. A configuration menu system is used to select the main items that are important, and those selections cause their dependencies to be included. The menu is accessed by running 'make menuconfig' in the main OpenWRT directory.



```
naw@scale.schaferhsv.com: /home/naw/21880/vortex/trunk
OpenWrt Kamikaze (r21879) Configuration

                         OpenWrt Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> builds as package.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> package  < >

        Target System (x86)  --->
        Subtarget (Generic)  --->
        Target Profile (Generic)  --->
        Target Images  --->
        Global build settings  --->
    [ ] Advanced configuration options (for developers)  --->
    [ ] Build the OpenWrt Image Builder
    [ ] Build the OpenWrt SDK
    [ ] Build the OpenWrt based Toolchain
    [ ] Image configuration  --->
    v(+)

              <Select>    < Exit >    < Help >
```

(Intro screen to 'make menuconfig')

Building for X86

To build a WiCM image for an x86 platform:

1) Log into the fedora virtual machine
2) mkdir x86 ; cd x86
3) tar xzf /media/sf_dl/backfire_10.03.1.tgz
4) cd backfire_10.03.1
5) ln -s /media/sf_dl dl
6) make package/symlinks

At this point you will see a menu system that looks like the one above. Use the cursor up and down to select items. You will want to change the target system on the first line by pressing enter. Use the cursor down key to go to the bottom and select x86 by pressing enter on it.

After x86 is selected, make sure Subtarget and Target Profile both are generic.

In the Target Images section, don't select a ramdisk image or any of the root filesystem archives. In the root filesystem images, only select ext2. The last thing to change for now is the serial port baud rate to 115200. Use exit on the bottom of the screen twice and then save your changes.

This adds to the build tree options to build many of the open source projects available on the internet. The build environment will download the source code for the packages automatically and apply patches that the OpenWRT group might have had to make to get the software to work with their distribution.

This process takes a while because it is doing a SVN checkout of all the current packages for OpenWRT. Since it is doing a fresh checkout, all the most recent packages are retrieved. Since we want to make sure we have a certain version of them, we need to go to this newly checked out directory and revert their versions to versions we use for WiCM. Do this by typing 'cd feeds/packages ; svn up -r 21888'.

The version of socat that is included in all these packages needs to be updated to a version that has all the capabilities that WiCM requires. To update the makefile in that package to get the newer version, type 'cd net/socat/ ; cp -f /media/sf_dl/Makefile.socat Makefile'.

This version had some problems building under OpenWRT, so I created a patch to fix that. To install the patch, type 'cp /media/sf_dl/503_crdly_tabdly_csize.patch patches/'

The MP3 encoder 'lame' in the base version of all the packages does not compile right. OpenWRT fixed it in a later version of the package for lame. To update it, type 'cd ../../sound/lame ; svn up -r 26195'.

Now, go back to the trunk directory with 'cd ../../../..' and:

1) cp /media/sf_dl/vortex_ide.patch target/linux/x86/patches-2.6.32/101-vortex_ide.patch
2) cd package
3) cvs -d /CVS co scale-mesh
4) cd ..

5) make menuconfig

From here we will start adding open source software packages to the image. OpenWRT only has a minimal amount of packages by default.

'stty' is a command to set serial port parameters. It is needed to set the baud rate on serial ports according to the needs of the sensors hooked up to them. It is not installed by default in OpenWRT, but it is available in the busybox package that OpenWRT installs. Busybox is a package that combines several common unix utilities into one binary and creates symbolic links to the binary with their names. For example, 'ls' is a symbolic link to the busybox executable. When the executable runs, it sees that the command line used to call it (argv[0] in C terms) is 'ls', so it runs the 'ls' function. 'make menuconfig' allows you to add several functions to busybox that are available but not included in order to save space. So, to add the 'stty' command, navigate to 'Base System', 'Busybox', 'Core Utilities', and then enable 'stty'. In the future, if any additional minor unix utilities are needed, it is good to check to see if you can simply enable them in busybox. If you can, and the function in busybox hasn't optimized out the functionality you need, it is the most optimum way to add the function to OpenWRT.

After enabling 'stty', exit out back to the main entry point in 'make menuconfig' by selecting 'Exit' until you see 'OpenWRT Configuration' at the top of the screen. Enable the package under 'Schafer' called scale-mesh. Make sure that it is built into the image by marking it with a '*'. That goes for all the packages and options in the configuration menus. Marking it with an 'M' merely builds the package to be installed on a running system later, and does not put it in the image. This package contains the main code that we have developed to put WiCM functionality into OpenWRT. It also causes other packages to be included so they don't have to be manually picked in 'make menuconfig'.

Under Network/SSH, install the openssh-sftp-server. This allows files to be moved on and off the node via scp, or winscp.

Next, install OLSRD. It is under Network and maybe under Routing and Redirection. In addition to marking it with a '*', press enter on it to configure the modules. Install the modules bmf, dot-draw, dyn-gw-plain, httpinfo.

At the same level as OLSRD, you will find tcpdump. This is a useful utility for debugging.

Back at the top level, enter the Kernel Modules directory. This is where device drivers are. To enable USB, enter the USB directory and enable ohci, uhci, and usb2. These are the USB interface drivers. You can enable storage if you want to use USB flash drives. 'acm' is the driver needed for the Verizon USB stick. 'cm109' is the audio driver on the Roboard. 'serial' is for many sensors that hook up to USB that are made with USB to serial chips. Enable it and you will get options to install 'cp210x' (Oceanserver Compass), 'ftdi' (XBee), and 'pl2303' (Deluo GPS).

The Roboard has an audio chip on it, so to make use of it, go back to the Kernel Modules section and enter Sound Support. Enable sound-core and then usb-audio under that.

Back under the main menu, the ‘Sound’ section has some sound applications that you can install. ‘sox’, ‘madplay’, and ‘lame’ are good points to start with getting sound on and off a node.

Since development of some code using sensors is in C++, add libstdcpp under Base System.

To use the serial ports on the Roboard, you will have to change their port addresses and IRQs. You need the ‘setserial’ program to do this. It is under Utilities/Terminal.

Now, exit the whole system, using the option to save at the end, and type ‘make’. This will build the image and all the cross compile tools needed. The first time will take the longest since it has to compile all of the tools. It will take about 2 hours.

After this first build, the kernel will be available. Hardware drivers that are not available in the kernel modules section will have to be built into the kernel. To modify the drivers built into the kernel, use ‘make kernel_menuconfig’.

The Roboard uses an Ethernet driver that does not have an OpenWRT package. To add it to the kernel, select Device Drivers/Network Device/Ethernet (10 or 100)/ RDC R6040 Fast Ethernet Adapter support.

The Micro-SD device on the Roboard requires a driver to be built into the kernel as well. To enable it, select Device Drivers/Serial ATA/ IT8211-2 PATA support.

Since the Roboard has extra serial ports, the linux kernel has to know this. To enable them, set the number of ports to 4 for both Maximum number and number to register at boot time under Device Drivers/Character Devices/Serial Drivers/.

After this is all done, exit and save the config file. Run ‘make’ one more time and you should have an image ready to run.

Building for Gateworks

The instructions for building a Gateworks image are about the same. In the beginning, you change the target to ixp4xx instead of x86 with a default target profile. You have to disable the apex boot loader if it is in the configuration and turned on. In the target image section, enable jffs to get an image you can install in onboard flash.

There is no onboard sound on the Gateworks, so you can omit the steps above that involve sound.

Building for Mikrotik

Installation

Installation procedures depend on the platform on which WiCM is being installed. The architectures WiCM has been installed on so far are X86, Gateworks, and Mikrotik. Procedures for installing on other platforms can be manufactured with help from OpenWRT's documentation on how to install it on that platform.

X86 Boards

On x86 type boards, an image is made that can be written to a compact flash or an SD flash card. This image makes the flash have 2 partitions. The first partition is where the boot information resides, and the second partition is the root filesystem. The image also contains the boot loader GRUB. GRUB is what the BIOS on the board loads to boot the operating system.

The image after a successful build of OpenWRT for x86 is in trunk/bin/x86/openwrt-x86-generic-combined-ext2.img.gz. You can gunzip this and copy it to a windows machine to write it to a flash using Win32DiskImager. Once the flash is ready to boot, hook up a serial port to COM1 on the board through a null modem cable (female to female) to a windows machine running PuTTY. Set the baud rate to 115200, serial port parameters to '8 none 1', no flow control, and turn on the board. You should see a lot of text flow by as the linux system boots. After it is all done, pressing enter should give a prompt that says 'root@SCALE-1-254:/#'. From here you are ready to configure the node.

Gateworks Boards

On Gateworks boards, WiCM can either be installed in onboard flash or on compact flash (if the board has a compact flash slot). GRUB is not used because Gateworks boards use a boot loader called Red Boot.

Gateworks Onboard Flash

To get the WiCM images into onboard flash, a machine running TFTP will need to be connected over Ethernet. See Appendix A for information on installing a TFTP server under Windows. Connect the left Ethernet port on the Gateworks boards to the Ethernet port set up with TFTP on a Windows machine.

The images that need to be served via TFTP are in trunk/bin/ ixp4xx/. These are the files that need to be copied to the tftp directory:

```
openwrt-ixp4xx-generic-jffs2-128k.img
openwrt-avila-zImage
openwrt-cambria-zImage
```

Connect a serial cable from a Windows machine running PuTTY. It can be the same machine that is running the TFTP server. The Gateworks boards use a straight through male to female serial cable to connect to a windows machine. Set the baud rate to 115200, serial port parameters to '8 none 1', no flow control, and (while ready to press control-c) turn on the board. As soon as you see the phrase "Trying…" press control-c. You should get a RedBoot prompt.

Gateworks makes two boards that we use so far. Look on the board for the words Avila or Cambria. The boards also have different amounts of memory and flash, so the exact commands are going to vary. So, they will be listed in the order from most memory to least memory, and just use the first one that works.

First the flash needs to be unlocked so WiCM can be installed. If the board is a Cambria, do whichever works:

1) fis unlock -f 0x50080000 -l 0x1f60000
2) fis unlock -f 0x50080000 -l 0x1f40000

If the board is an Avila, 'fis unlock -f 0x50080000 -l 0xf60000'.

Next, initialize the flash with 'fis init –f'.

Set the IP address of RedBoot with 'ip_address -l 172.16.1.2/12 -h 172.16.1.1'.

If you have a Cambria, type 'load -r -b 0x80000 openwrt-cambria-zImage'. If it is an Avila, type 'load -r -b 0x80000 openwrt-avila-zImage'.

Put the kernel in flash with 'fis create linux'.

Next, load the root filesystem into ram with 'load -r -b 0x80000 openwrt-ixp4xx-generic-jffs2-128k.img'.

If you have a Cambria, do the first of these that work:

1) fis create -l 0x1e80000 rootfs
2) fis create -l 0x1e60000 rootfs
3) fis create -l 0x1e40000 rootfs

If it is an Avila, do the first of these that works:

1) fis create -l 0xe80000 rootfs
2) fis create -l 0xe60000 rootfs
3) fis create -l 0x660000 rootfs

At this point, both the kernel and the root filesystem are in flash. RedBoot now just needs to be configured to run them. Run fconfig and change the boot script to be:

```
fis load linux
exec -c "root=/dev/mtdblock2 rootfstype=jffs2 noinitrd console=ttyS0,115200"
```

After that is saved, type 'reset' and WiCM should boot. After it is done, pressing enter should give a prompt that says 'root@SCALE-1-254:/#'. From here you are ready to configure the node.

Gateworks Compact Flash

You have to be logged in as root to write to a compact flash under Linux. On the scale machine, compact flashes show up as /dev/sdb when they are inserted.

The flash has to be partitioned into a boot partition and a filesystem partition. The boot partition can be 128 MB, and the filesystem partition can take the rest of the flash. Use the Linux utility fdisk to do this.

After the flash is partitioned, the partitions have to be formatted. Use mkfs.ext2 on /dev/sdb1 and /dev/sdb2.

Copy the kernel bin/ixp4xx/openwrt-avila-zImage to the first partition, and name it just zImage.

```
mkdir temp
mount /dev/sdb1 temp
cp bin/ixp4xx/openwrt-avila-zImage temp/zImage
umount temp
```

Copy the filesystem bin/ixp4xx/openwrt-ixp4xx-generic-rootfs.tar.gz to the second partition.

```
mount /dev/sdb2 temp
tar xzCf temp bin/ixp4xx/openwrt-ixp4xx-generic-rootfs.tar.gz
umount temp
```

Before you pop the flash out, run 'sync'. This will make sure that any edits to the flash are done.

RedBoot needs to be configured to boot off of compact flash. If the board was not already booting off of flash, connect the serial port via a straight through cable to a Windows machine and run PuTTY. Set the baud rate to 115200, serial port parameters to '8 none 1', no flow control, and (while ready to press control-c) turn on the board. As soon as you see the phrase "Trying…" press control-c. You should get a RedBoot prompt. Type 'fconfig' and change the boot script to:

```
load -m disk -r -b 0x01600000 hda1:zImage
exec
```

After that is saved, type 'reset' and WiCM should boot. After it is done, pressing enter should give a prompt that says 'root@SCALE-1-254:/#'. From here you are ready to configure the node.

Mikrotik Boards

On Mikrotik boards, the only option is to install WiCM into onboard flash. The boot loader on the Mikrotik boards does not have the same features as Red Boot, so a minimal OpenWRT distribution has to be network booted first, then a full version with WiCM is installed using the minimal OpenWRT. A machine running a DHCP server, a TFTP server, and an FTP server is needed to install a Mikrotik board. For information on how to set one up see the Appendixes.

First, connect a null modem serial cable to the Mikrotik board from the upgrade machine. Open PuTTY with a COM connection and set the baud rate to 115200, serial port parameters to '8 none 1', no flow control, and (while ready to press enter) turn on the board. As soon as you see the phrase "Press any key within 2 seconds to enter setup" press enter. You should get a prompt.

Next, connect an Ethernet cable to the Mikrotik board from the upgrade machine. To configure the board to boot from the upgrade machine:

```
press p
press 2
press o (lower case letter)
press e
press x
```

This will cause the Mikrotik to boot into openwrt off of the upgrade machine. Press enter when you see "nf_conntrack version 0.5.0"

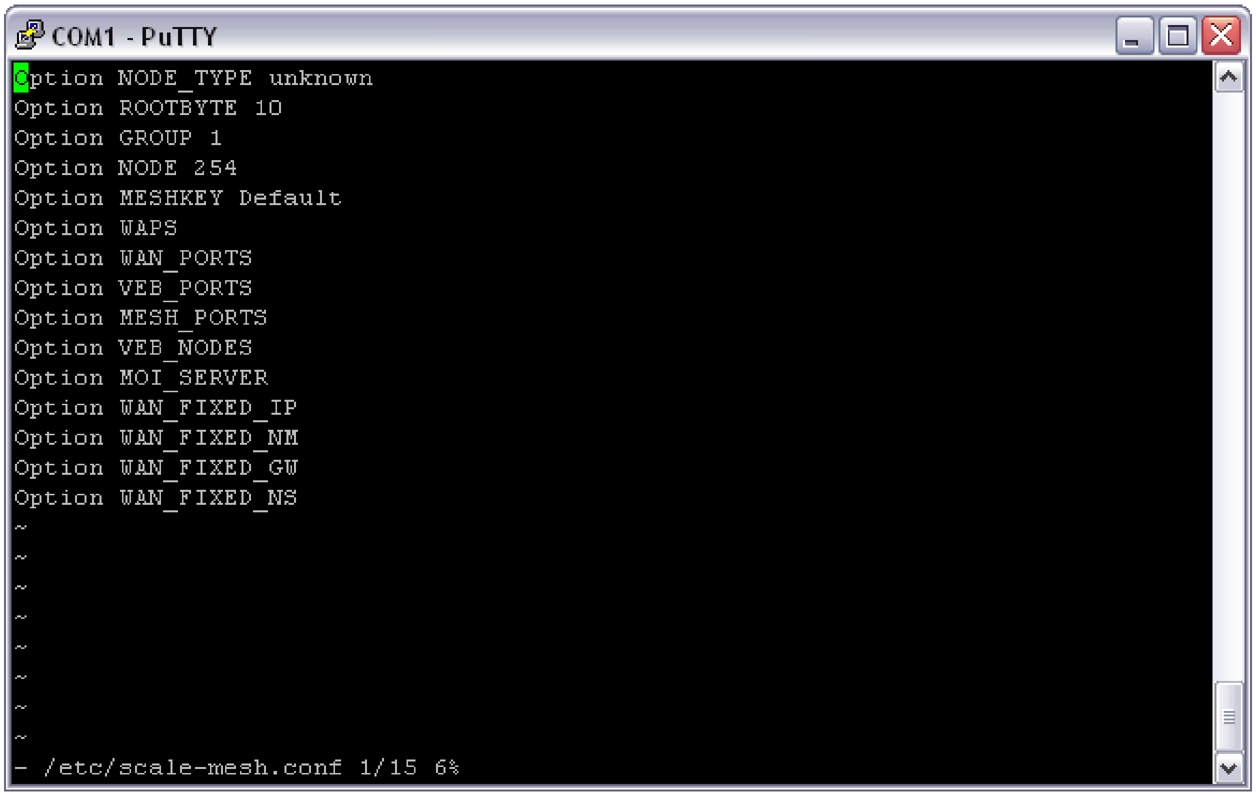
At the openwrt prompt type:

```
 ifconfig br-lan 172.16.1.2
wget2nand ftp://172.16.1.1/
reboot (get ready to press enter)
press enter as soon as power comes on and says "Press any key within 2 seconds to enter setup"
press o
press o
press x
```

After booting twice this time, press return when you see "device br-lan entered promiscuous mode" and you should get SCALE prompt.

Configuration

The default group number for a node is 1, and the default node number is 254. To change this, edit /etc/scalemesh.conf using vi. The file looks like this:



```
Option NODE_TYPE unknown
Option ROOTBYTE 10
Option GROUP 1
Option NODE 254
Option MESHKEY Default
Option WAPS
Option WAN_PORTS
Option VEB_PORTS
Option MESH_PORTS
Option VEB_NODES
Option MOI_SERVER
Option WAN_FIXED_IP
Option WAN_FIXED_NM
Option WAN_FIXED_GW
Option WAN_FIXED_NS
~
~
~
~
~
~
~
~
- /etc/scale-mesh.conf 1/15 6%
```

NODE_TYPE is part of the string that is broadcast to the rest of the network so that client software on windows can display to the user what kind of device it is. Examples are MARCbot, DragonRunner.

ROOTBYTE is not used anymore.

GROUP and NODE are what need to be made unique for this node. Each can be 1-254.

MESHKEY is a string that is used to generate a key for AES encryption. 'Default' skips this generation and uses WEP encryption with the key that was used for MARCbots.

WAPS is a comma delimited list of the radios that are to be used as standard wireless access points. Example radio names are SR4, XR5, XR2, XR4, XR9.

WAN_PORTS is a comma delimited list of the Ethernet ports that are to be connected to a higher level network. An example is eth0. By default dhcp is used to get an IP address from that network. If dhcp is not available, WAN_FIXED_* can be set to a static ip, netmask, gateway, and name server.

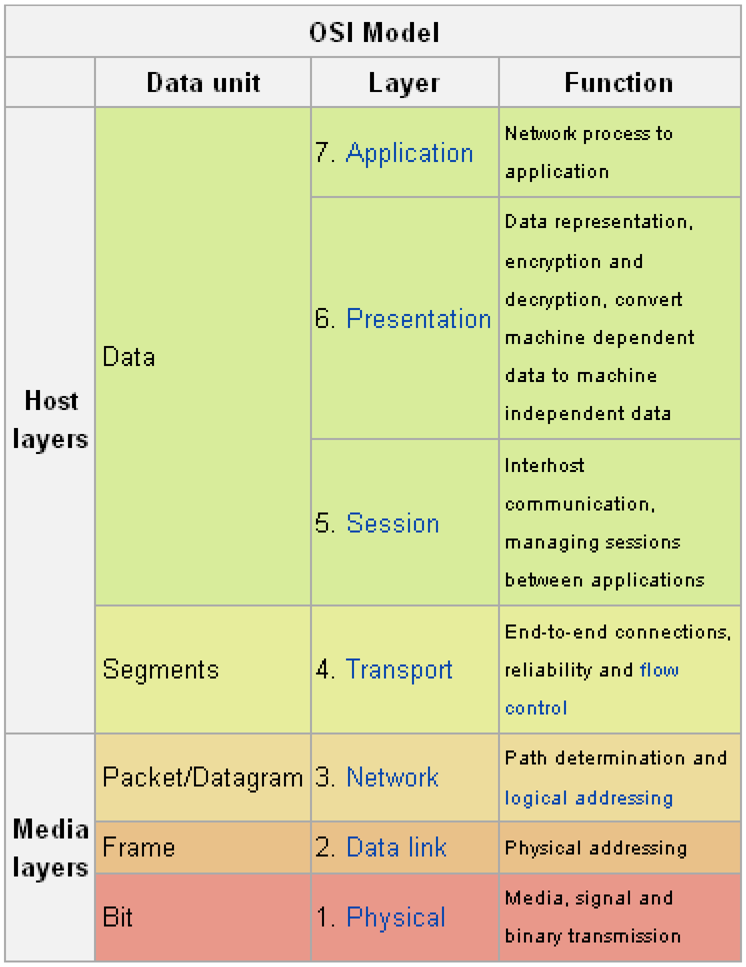
VEB_PORTS is a comma delimited list of the Ethernet ports that are to be bridged to Ethernet ports on other nodes. VEB_NODES are the other nodes to bridge to. VEB_NODES is also a comma delimited list of the group and node numbers such as 12:3,5:1,1:4.

MESH_PORTS are the Ethernet ports that are to be used to make a wired mesh between the nodes.

MOI_SERVER is the Mesh Over Internet server. This is how several nodes all with internet connections in separate locations can mesh together over the internet. If it is 'me', then this node is set up to serve the mesh over internet. Otherwise it is the IP address of the server.

These variables are used in various WiCM scripts, the most important one being /etc/init.d/scale-mesh. Their values can be imported into a borne shell script with the command 'eval `awk '/^Option/{printf $2"="$3"\n"}' /etc/scale-mesh.conf`'. To make sure you get the right apostrophe characters, it is best to copy and paste the line from /etc/init.d/scale-mesh. Backwards apostrophes are used to executing a command and inserting the standard out of that command onto the command line. Regular apostrophes are for quoting a string so that the shell does not preprocess it. The 'eval' command processes the line again without the eval after the sub command has run.

How WiCM operates in the OSI Model



OSI Model

	Data unit	Layer	Function
Host layers	Data	7. Application	Network process to application
		6. Presentation	Data representation, encryption and decryption, convert machine dependent data to machine independent data
		5. Session	Interhost communication, managing sessions between applications
	Segments	4. Transport	End-to-end connections, reliability and flow control
Media layers	Packet/Datagram	3. Network	Path determination and logical addressing
	Frame	2. Data link	Physical addressing
	Bit	1. Physical	Media, signal and binary transmission

OSI Network Layers and what we do at each layer:

1) Physical – The radios in the Atheros chipset based boards handle the physical layer. The only configuration we do at this point is the antenna cable and the antenna.

2) Data Link- Through the Atheros chipset driver called madwifi, settings controlling the data link layer can be modified. The ‘iwconfig’ commands modify this lowest level of transmission configuration.
3) Network- This is the layer at which the mesh protocol runs. OLSRD transmits packets that other nodes within listening range hear. Kernel routing tables are modified based on what everybody hears.
4) From this layer up to layer 7, a WiCM node looks like a standard IP router.

Adding Sensors

How to add a camera

Any IP based camera can be plugged into a node and accessed over the mesh. Up to this point, only ACTi cameras have been integrated into the SCALE software that runs on the laptop. ACTi has a proprietary active-X plugin that allows viewing the video from their cameras from within an application.

Most IP cameras have a web page that shows the video from the camera. Simply opening a web browser to the IP address of the camera from a laptop on the mesh network will allow you to see that video. First you need to know the IP address of the camera, though. Most cameras will come set up to request an IP address from the node using DHCP. The node assigns an address and makes a note of the address in /tmp/dhcp.leases. The process that sends out the SCALE hello packet (/usr/bin/scale_hello) looks at this file to announce what the IP address is for any ACTi cameras connected to it that have requested an IP address via DHCP. In /usr/bin/scale_hello, there is a line that says ‘CAMERA=`awk '/00:0f:7c/{print $3 ; exit}' /tmp/dhcp.leases`’. 00:0f:7c is the MAC header assigned to ACTi. If you wish to change to a different camera company, change the 00:0f:7c to the header of the new camera. You will be able to find out what that is in /tmp/dhcp.leases when the new camera is hooked up and you can see its entry.

If you do this, however, the SCALE software will not know whether the IP for the camera on a node is and ACTi or not. So, additional code will have to be added to SCALE to detect what type of camera is there and do whatever is appropriate to view the different types of cameras. Other cameras have active-X plugins like ACTi, so they might be able to be integrated like the ACTi cameras were. Otherwise you could simply open a web page to the camera. This is what happens currently anyway when you hold the shift key while pressing ‘Video’ in the SCALE software.

How to add a sensor

Adding a sensor depends a great deal on what kind of interface the sensor has. The general approach has been to obtain or write a program to get the data off of it and send the output to the standard output file descriptor (stdout). In other words, if you are logged in via PuTTY, have the data print out to your terminal. Once you get the data there, it is easy to get it across the network using the utility ‘socat’. ‘socat’ can either listen for requests for data and send the data to the requestor, send the data to a predetermined IP address, or broadcast it via multicast. Since the node will forward multicast packets to the rest of the mesh, you can use that to get data from that sensor to all other points in the mesh. Clients that then want to see that data only have to listen on the correct multicast address and port number to hear the data.

Something you have to remember when sending to a multicast address, however, is how OLSRD forwards the data. In order that it doesn't send packets around in circles, OLSRD keeps a record of all the multicast packets it has forwarded for the last several seconds. If the packet matches any of these it will be discarded. So, if your sensor is not constantly changing, either your clients need to assume that no data means no change, or you need to add to the data stream a timestamp.

For an example, let's assume you have a sensor that has a serial interface. If your node has a serial interface that is compatible (make sure that both interfaces are at the same voltage level, either TTL or RS232), then hook the sensor up to that interface if it is not already being used by linux as the console port. Then you can use the stty command to set up the baud rate and the serial port parameters. If the port the sensor is connected to is COM3, then an example stty command to set up the port might be 'stty -F /dev/ttyS2 4800 igncr –echo', where /dev/ttyS2 refers to COM3 (linux starts the com port count at 0), 4800 is the baud rate, igncr tells the port to discard carriage returns (linux uses just newlines to determine end of line), and '-echo' tells linux not to echo the data coming in from the sensor back to the sensor. The default serial port format is 8 data bits, no parity, 1 stop bit. This works for most sensors, but if you happen to have one that is different, other stty flags will change those parameters.

If you have a serial sensor but only a USB port on the node, then you can use a USB to Serial adapter. Ones made by FTDI work well, and you can get cables made to provide the voltage needed to run the sensor in many cases. If you go this route, use the same directions as above, except use /dev/ttyUSB0. If more than one sensor is on a USB to Serial adapter, then there will be /dev/ttyUSB1, /dev/ttyUSB2, etc. and you will have to figure out what order the sensors are in.

If everything goes right, then you should be able to 'cat /dev/ttyS2' to see that data on your screen. If that looks good, then to get it to the network, use 'socat –u /dev/ttyS2 udp-sendto:224.0.0.100:1234'. That command multicasts the data from that sensor to the address 224.0.0.100 on the port 1234. Those numbers can be changed as needed.

/etc/init.d/scale-mesh:

The file /etc/init.d/scale-mesh is the main startup script for a WiCM node. It is installed with the scale-mesh package. OpenWRT by default is set up to have configuration web pages for each package. This is nice if you want to set up one node and be able to log into the web page and change settings for each package. In WiCM, the plan was to set the configuration in one place (/etc/scale-mesh.conf) and have the configuration of all the other packages be generated from those settings. So, one of the first things that this startup script does is see if the startup scripts for some other packages exist, and if they do, remove them and reboot. /etc/init.d/scale-mesh will take care of starting up those other packages with the configuration that WiCM needs.

The very first thing the script does is remount the root partition with the filesystem option of noatime turned on. Since OpenWRT is running out of flash on every platform so far, we do this to minimize wear on the flash. Normally Linux keeps track of the last time a file was accessed in the file system. For it to keep track of this, it writes to the filesystem to update this record even if a file is just being read. Remounting root with noatime stops writes to flash when files are only being read.

After that, the flash is checked for a third partition. The flash starts out with two partitions: One for the booting and one for the main filesystem. If a third exists, it is assumed that it is for data storage. If it is found, it does a filesystem check on it and mounts it under /data.

Next, the script creates three Ethernet bridges. One bridge is for the LAN ports, one is for the WAN ports, and one is for the Virtual Ethernet Bridge ports. Ethernet devices later in the script are looped over and put into one of these bridges as determined by reading /etc/scale-mesh.conf.

The /etc/scale-mesh.conf file is not removed or changed when the scale-mesh package is updated. This is so any modifications to this file won't be lost. A newer scale-mesh package might add options that are available to be configured in the conf file. In order for possible new options in a newer package to be made available, /etc/scale-mesh.conf.default (which is updated when packages are updated) is checked each boot to see if it contains new options that aren't in /etc/scale-mesh.conf. If so, it will create lines in /etc/scale-mesh.conf to reflect the new options.

At this point, custom kernel modules are installed. Some kernel modules are not available as packages in OpenWRT. You can use the cross compiler to build kernel modules outside of the OpenWRT build system. The startup script can automatically install these modules if they are put in the expansion of '/scale_modules/`uname -m`/`uname -r`/'. On a Roboard, this expands to /scale_modules/i586/2.6.32.14/. The modules in that directory are added in alphabetical order, so name them in the order that they need to be installed. For example, 01_first_module.ko, 02_second_module.ko, etc.

After the modules are installed, the init script starts to set up all the network interfaces. In order for every node to have the address 172.16.G.N, a tap interface is made with that address and it is used as a mesh interface. That way OLSRD will tell the rest of the mesh of that IP address

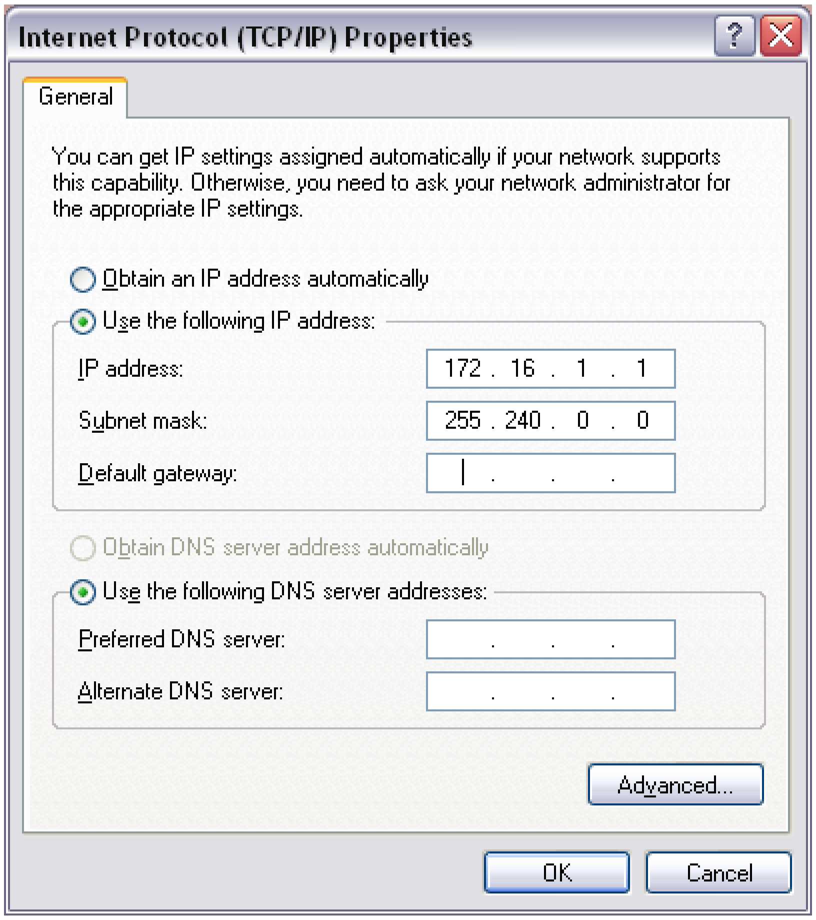
(more explanation to follow)

Appendix A: Installing a TFTP server under Windows

A TFTP server is needed to provide WiCM images over Ethernet to be installed on a single board computer. A machine to be used for this purpose should have an available Ethernet port and serial port. For this section, anytime an Ethernet port is referred to, it is the previously unused Ethernet port that is to be used to install WiCM.

Set the IP address of the unused Ethernet port to 172.16.1.1 with a netmask of 255.240.0.0. This is done by opening Control Panel, switching to classic view if not in that mode, and opening Network

Connections. Right click on the Ethernet port and select properties. In the scrolled list of items, double click on Internet Protocol (TCP/IP). After changing the IP address, the window should look like:


Internet Protocol (TCP/IP) Properties

General

You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

Obtain an IP address automatically [✓ Use the following IP address:]

IP address: 172 . 16 . 1 . 1
Subnet mask: 255 . 240 . 0 . 0
Default gateway: . . .

Obtain DNS server address automatically [✓ Use the following DNS server addresses:]

Preferred DNS server: . . .
Alternate DNS server: . . .

Press ok in this window and the previous window.

Install the freely available windows software package OpenTFTPServerMTInstallerV1.63.exe. In the installation windows, just pick the default options. In the start menu under all programs, there should be an ‘Open TFTP Server’ menu. Select ‘Configure’ under that menu. This will bring up an editor. Change the contents of the file to:

```
[LISTEN-ON]
'172.16.1.1
[HOME]
c:\tftp
[LOGGING]
[ALLOWED-CLIENTS]
[TFTP-OPTIONS]
```

In the control panel, open 'Administrative Tools', then 'Services'. In the services window, right click on 'Open TFTP Server, MultiThreaded' and click restart. The window machine is now ready to provide WiCM images via TFTP. The images should be placed in the C:\tftp directory.

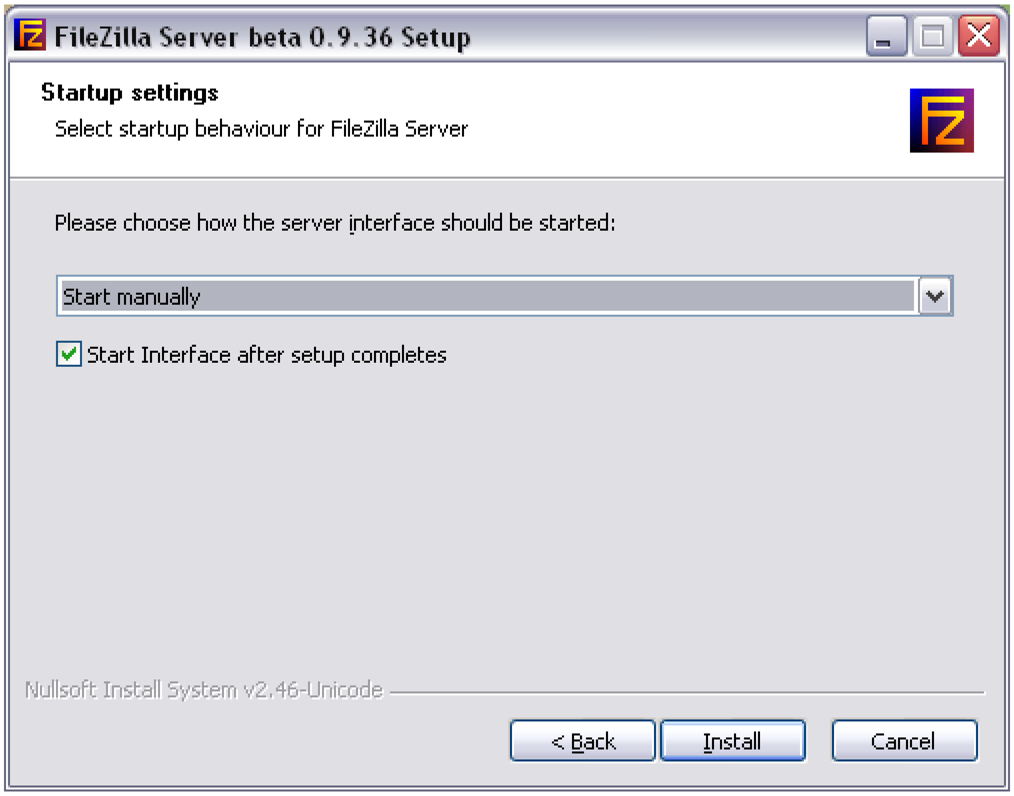
For the tftp server to be reachable, the firewall either needs to be turned off, or it needs to be set up to allow tftp connections. For information on how to do this, see Appendix C.

Appendix B: Installing FTP and DHCP Server

To install WiCM on a Mikrotik board, you will need a machine that is running a DHCP server and an FTP server. The Mikrotik will ask the DHCP server for an IP address, and what image to run. It will download the image over tftp. Then, once that image it is running, a final image to burn to flash will be downloaded over ftp.

First install DualServerInstallerV6.72.exe with default options.

Next, install FileZilla_Server-0_9_36.exe with default options until choosing how the server interface should be started. Then pick Start Manually.


FileZilla Server beta 0.9.36 Setup

Startup settings

Select startup behaviour for FileZilla Server

Please choose how the server interface should be started:

Start manually

☑ Start Interface after setup completes

Nullsoft Install System v2.46-Unicode

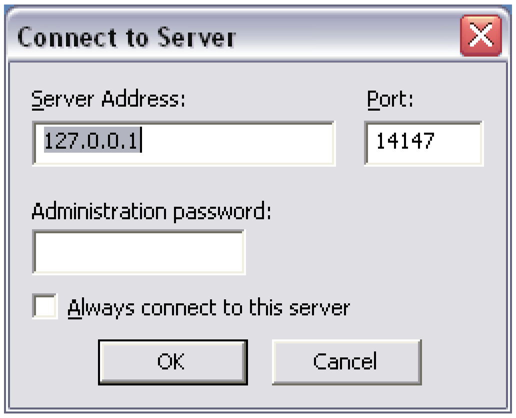
From the start menu, select DualServer/ Configure. It will bring up configuration in notepad.

Change to:

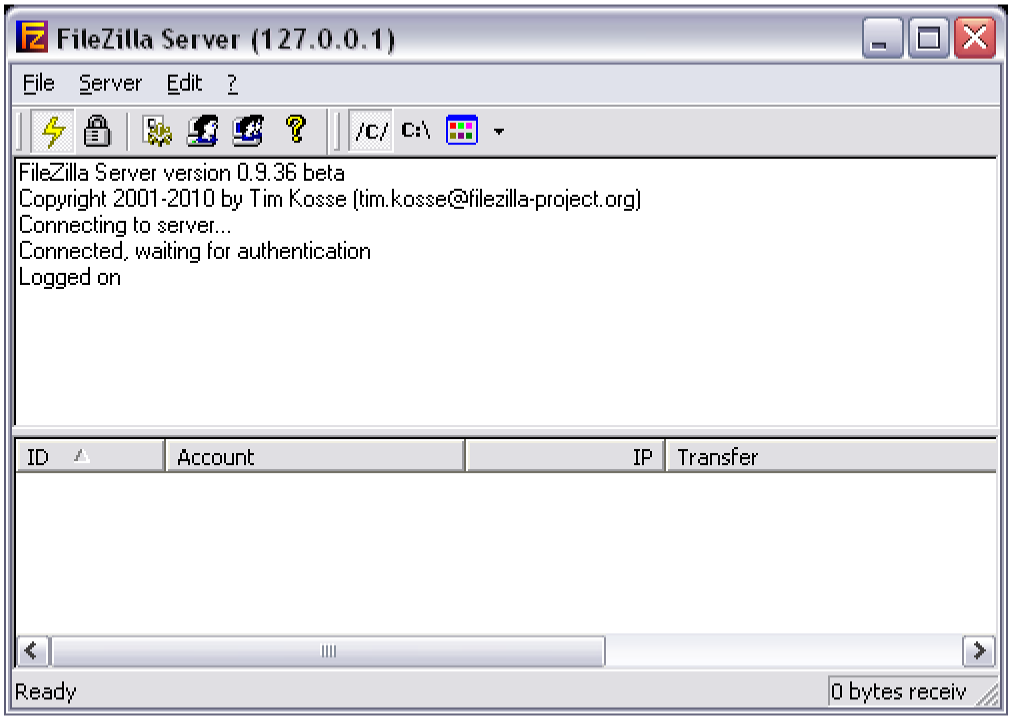
```
[SERVICES]
[LISTEN-ON]
172.16.1.1
[LOGGING]
[DHCP-RANGE]
DHCP_Range=172.17.1.1-172.17.1.254
Subnet_Mask=255.240.0.0
Next_Server=172.16.1.1
Boot_File=rb411
```

In control panel, Administrative tools, Services: Select Dual DHCP DNS Service and the click on restart. Leave the Service window open; it will be used again below.

From start menu, select Filezilla Server/Filezilla Server Interface, click ok to log in without password.

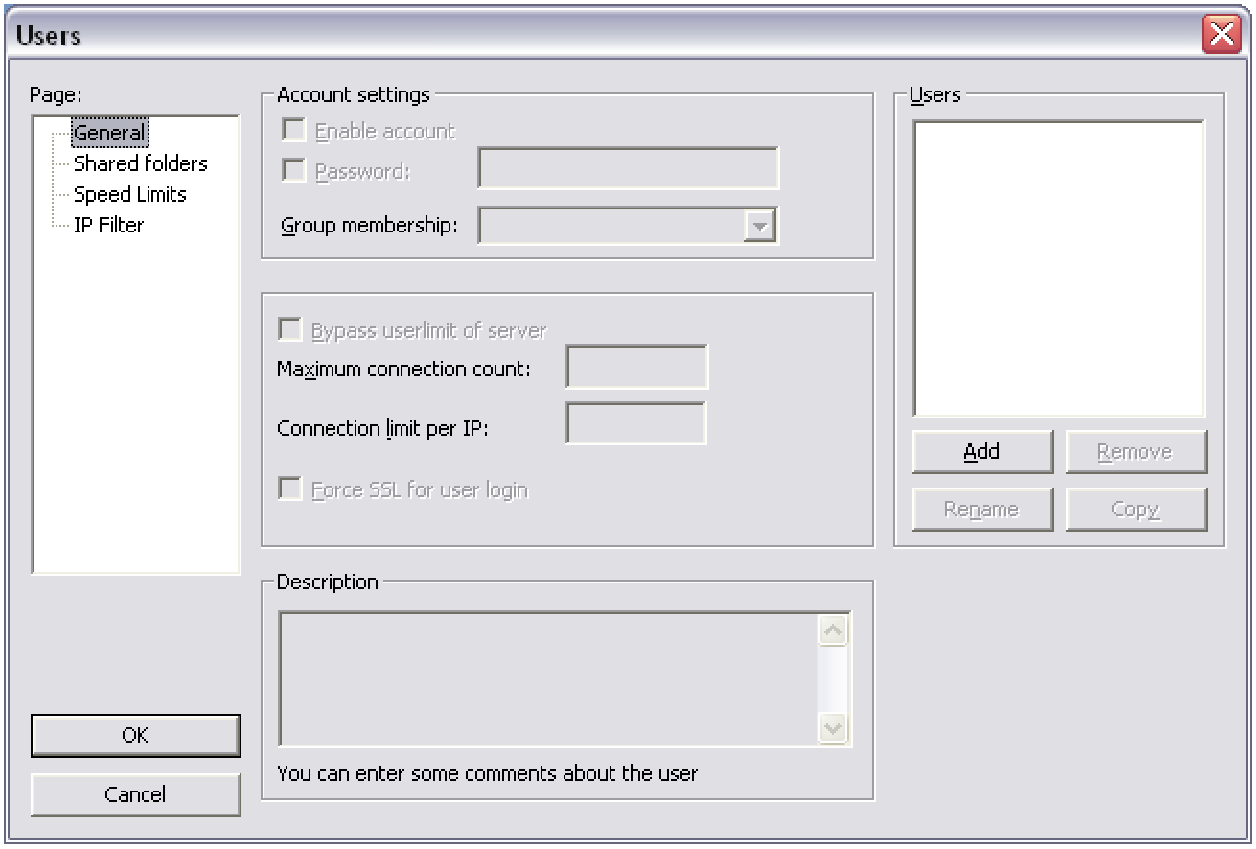


Server Address: 127.0.0.1
Port: 14147



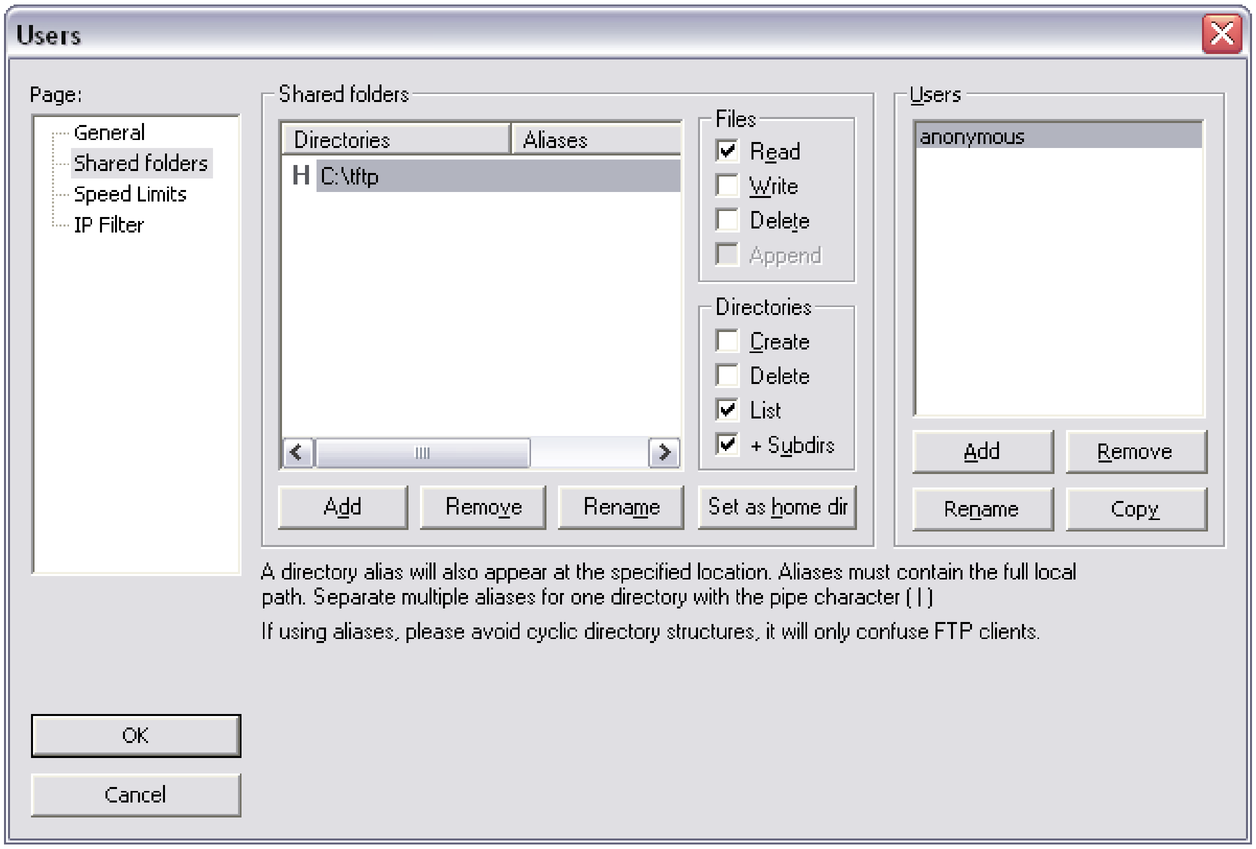
```
FileZilla Server version 0.9.36 beta
Copyright 2001-2010 by Tim Kosse (tim.kosse@filezilla-project.org)
Connecting to server...
Connected, waiting for authentication
Logged on
```

From the menu, Edit users.



Add user named 'anonymous' left in group <none> and leave password blank and unchecked.

Under shared folders add c:\tftp as home.



Users

Page: General, Shared folders, Speed Limits, IP Filter

Shared folders

Directories	Aliases
H C:\tftp	

Files: Read [✓], Write [ ], Delete [ ], Append [ ]

Directories: Create [ ], Delete [ ], List [✓], + Subdirs [✓]

Users: anonymous

A directory alias will also appear at the specified location. Aliases must contain the full local path. Separate multiple aliases for one directory with the pipe character (|)

If using aliases, please avoid cyclic directory structures, it will only confuse FTP clients.

Press ok and close the previous window.

For the servers to be reachable, the firewall either needs to be turned off, or it needs to be set up to allow connections. For information on how to do this, see Appendix C.

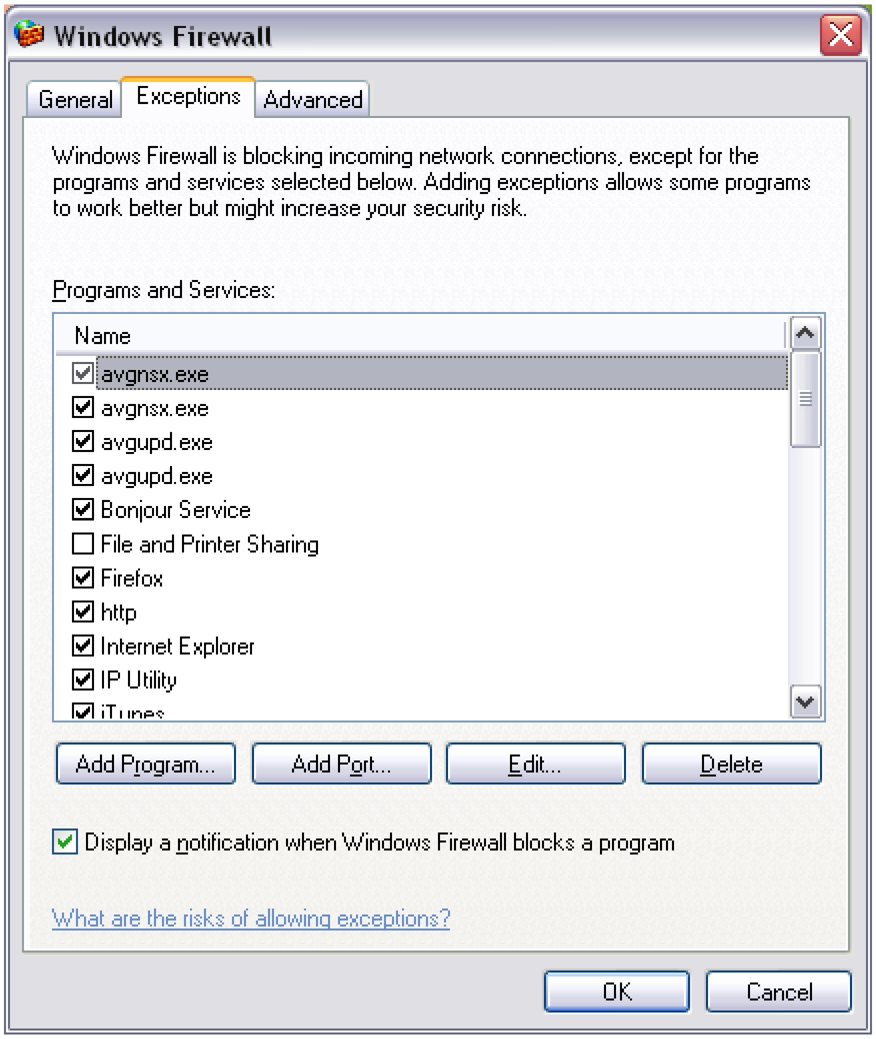
Appendix C: Configuration of Firewall

The servers set up in Appendixes above require the firewall under windows to either be disabled or set to allow connections on those servers. To allow connections, open control panel, windows firewall. Click on the exceptions tab and Add Program. Then click browse. The 3 programs are:

c:\program files\DualServer\DualServer

c:\program files\FileZilla Server\FileZilla server

c:\program files\OpenTFTPServer\OpenTFTPServerMT



Windows Firewall is blocking incoming network connections, except for the programs and services selected below. Adding exceptions allows some programs to work better but might increase your security risk.

Programs and Services:

Name
✓ avgnsx.exe
✓ avgnsx.exe
✓ avgupd.exe
✓ avgupd.exe
✓ Bonjour Service
☐ File and Printer Sharing
✓ Firefox
✓ http
✓ Internet Explorer
✓ IP Utility
✓ iTunes

✓ Display a notification when Windows Firewall blocks a program

What are the risks of allowing exceptions?

Appendix D: Building OpenWRT under Mac OSX Lion

You might be able to build OpenWRT under OSX. To do so, first use Disk Utility to create a partition that is case sensitive. You just have to resize your current partition to make room, add a new one, and format it to be case sensitive.

You also have to install XCode at least 4.1 under OSX 10.7. It is available in the App store.

Fink is the project that ports open source software to Apple. It is needed for some of the tools to build OpenWRT that would normally be in Linux. It is available from http://finkproject.org/. Download it and change to the directory where your new case sensitive partition is mounted (under /Volumes).

```
 tar xvzf ~/Downloads/fink-0.31.6.tar.gz
 cd fink-0.31.6/
./bootstrap
/sw/bin/pathsetup.sh (After this quit the terminal and open again)
```

```
fink selfupdate-rsync
fink index –f
fink install wget-gnutls-idn
fink install gawk
fink install findutils
fink install coreutils
```