UNIVERSITY OF FREIBURG

DEPARTMENT OF COMPUTER SCIENCE

BACHELOR THESIS

DELAY TOLERANT NETWORKING: A REALIZATION ON THE GUMSTIX VERDEX PLATFORM

Author:
Thomas Mayer

Supervisor:
Prof. Dr. Schindelhauer
Dipl. Inf. Rührup

September 18, 2008

Abstract:

orem ipsum dolor sit amet, consectetuer sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Here is space for the basic idea

Chapter 2

Delay Tolerant Networking

Space for some stuff to get to chapter 2

2.1 A Historical Perspective

2.2 Theory

2.3 An Architectural Overview

Chapter 3

Building the Environment

In order to test the Delay Tolerant Networking (DTN) implementation on the gumstix verdex platform a properly set up environment is needed. The setup is covered in this section and will consist of several parts. The goals are:

- Having a environment in which programs for the gumstix verdex platform can be built and compiled into images that can be applied to the chip
- Having ad-hoc mode working on a linux platform.
- Configuring the gumstix so that it can function as a dhcp server for the end devices aswell as doing a fully automatic configuration as soon as the power supply is connected.

The first thing that will be covered is the setup of the linux platform and afterwards the environment will be used to set up the gumstix.

3.1 The Platforms

The platforms presented here are used for all the tests in chapter 3. All the setups described in this work were done on these platforms.

The Gumstix Verdex

The End Device

The end device for all the tests was a Lenovo Thinkpad x61t Notebook with a x86 2.6.22 kernel and an Intel PRO/Wireless 4965 AGN chipset. This chipset is typical for most notebooks that are sold at the moment.

3.2 Building the Linux Environment

There are basically 2 things that need to be done to have a fully working linux environment to run the tests described in the next chapter. Buildroot needs to be

present so that images for the gumstix can be built and deployed and an ad-hoc enabled wireless lan needs to be configured in order for the notebook to be able to act as an end device.

3.2.1 Buildroot

The first thing is to start by choosing a linux distribution to install. The steps that will be covered here are based on a Ubuntu 7.10 Gutsy Gibbon distribution. At the time being the latest version of Ubuntu is 8.04 but this distribution comes with a 2.6.24 kernel which does not work for ad-hoc mode combined with an intel wireless chipset, as will be shown later in the workarounds section.

The distribution needs to be installed according to the installation instructions provided by the homepage of Ubuntu. After having done that, the first thing to do is to instasll additional software to extend the environment so that building packages will be possible. This can be done via the shell command in Figure 3.1.

```
1 sudo apt-get install build-essential libncurses-dev
2 bison flex texinfo zlib1g-dev gettext openssl-dev
3 autoconf wget
```

Figure 3.1: Extra Programs

Also a subversion client needs to be installed to check out the gumstix repository[1]. Afterwards one has to check out the revision of the repository that corresponds to the version installed on the gumstix platform. A newer version may be used, but it should be used only if there are new packages in the newer revision that are needed, since some of the revision aren't stable and may not compile. The revision that is used on the gumstix can be found in the `/etc/gumstix-release` file on the gumstix itself.

It must be made sure that Ubuntu uses the bash as its standart shell and not the dash shell, which is the default under this distribution. This can be checked with the command in Figure 3.2 line 1 and it can be changed with the command in line 4. The command in line 2 can also be used to set the shell back.

```
1 ls -l /bin/sh
2 lrwxrwxrwx 1 root root  /bin/sh -> dash
3 #if this is the case then it can be changed with
4 sudo dpkg-reconfigure dash
```

Figure 3.2: Changing the shell

[1] At the time present this is http://svn.gumstix.com/gumstix-buildroot/trunk as the user "root"

If the desired revision is checked out the actual configuration of the buildroot environment can be done. Buildroot is the packet manager of gumstix. Software can either be cross compiled an loaded onto the gumstix platform by hand hand or it can be included as a package in the buildroot which does the cross compiling. The advantage of using buildroot is that, it is actually a management system, which means that software can be installed and can cleanly be deinstalled. So the few extra steps of building packages should be taken. As a further advantage, buildroot compiles all the software right into an image, so if another gumstick is to be added to the network, the image can simply be put onto the chip.

The base for buildroot to generate the image is located in the folder $BUILDROOT/build_arm_nofpu/root. Files that are copied or changes that are made inside this folder will be compiled into the image. The configuration of components should be done here so that the final configuration is included inside the filesystem image.

To build the environment the commands in Figure 3.3 have to be executed within the trunk directory of the gumstix release source. The command in line 2 is only necessary if it is not the first try building the environment and is only there to make sure the configuration will be written from scratch.

```
1 rm .config
2 make defconfig
3 make
```

Figure 3.3: Building the environment

A few questions about the platform will be asked. For the test platform described above they should be answered with:

- Architecture: ARM
- Target architecture variant: 12. iwmmxt
- Processor speed: 400MHz

writing a new line and filling in the apropriate line in the stop procedure.

The buildroot should be ready now and can be accessed via the `make menuconfig` command. It will look like the screen shown in Figure 3.4.

The entires and the changes that need to be done to the entries are listed here:

- **Target Architecture:** This should be already set to ARM.
- **Target Architecture Variant:** This should be already set to iwmmxt.
- **Target ABI:** The needed value is EABI.
- **Build Options:** No changes to be done here.

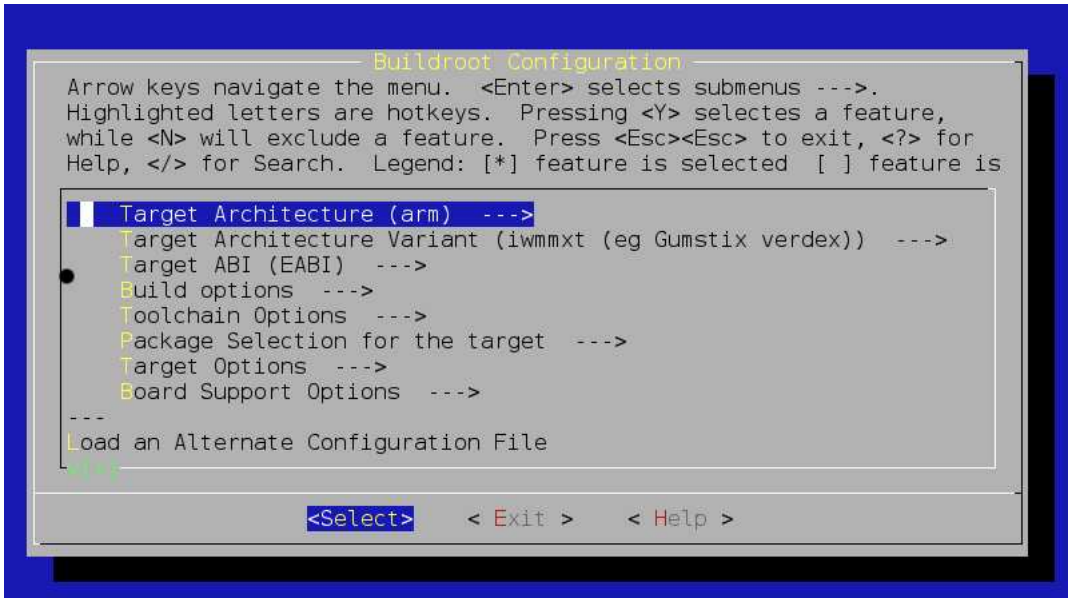

Figure 3.4: Buildroot Main Menu

- **Toolchain Options:** No changes to be done here.
- **Package Selection for the Target:** The list of packages that need to be selected in addition to the base system will be discussed in the gumstix section of this chapter.
- **Target Options:** There need to be done some changes here or else the compiled image will not fit onto the gumstix memory. The "Pad Output" option needs to be enabled and the value need to be set to 0xEC0000 which is roughly about 15,5 MB. The filesystem image, the kernel and the bootloader altogether are not allowed to exceed 16MB which is the memory available on the chip.
- **Board Support Options:** The processor speed can be found here, but since this was configured earlier, no changes need to be applied here.

Once these settings are done, packages can be selected via the respective main menu entry and the image can be compiled with the `make` command.

3.2.2 Ad-hoc Mode

If the configuration is like mentioned above, the ad-hoc should now work out of the box. If a newer distribution 2.6.22+ has been used, theres still the possibility to have a new linux installed and use an old kernel for developement; this can be

done by installing the old kernel as an upstream kernel beneath the normal kernel. This can be done, but will not be covered here.

3.3 Building the Gumstix Environment

The gumstix can be contacted via ssh. This can be done using the ethernet port, or a bluetooth serial connection if this has been configured beforehand. At the end of this section the platform will be able to go into ad-hoc mode right after the startup and give IP addresses to devices that connect to the network.

3.3.1 Package configuration

There are different programs that can act as a dhcp server. The standart way to go is to use DNSBind, which is one of the most powerfull, yet also one of the most complicated to configure and use tools. In addition the normal DNSBind package would be too large for gumstix platform that only has 16 MB of flash memory. So the choice for this test scenario will be dnsmasq. Dnsmasq is a very lightweighted dns forwarder and dhcp server that is fairly easy to configure.

The next thing that will be needed is an environment to run perl scripts, since the scripts for the test cases aswell as the GumDaemon that is described later in chaper 2 are written in perl. Basically it should be possible to run the GUmDaemon on a normal gumstix platform which could make a gumstix platform an end device instead of a notebook.

So the additional packages that should be selected in buildroot are the “dnsmasq” and the “microperl” packages.

3.3.2 Flashing to the new Image

There are several ways one can add a new Image to the verdex platform. The way that will be discussed here is to flash the gumstix by establishing a SSH connection and preparing the platform beforhand and afterwards rebooting to the new image.

After compilation via buildroot has terminated successfully there are 3 new files in the buildroot trunk directory, these are:

- `u.boot.bin`: This is the bootloader. The bootloader lies on a extra partition of the flash memory. It doesn’t need to be flashed everytime the filesystem is replaced. It only needs to be reflashed if the kernel version has changed and thus the links, that the bootloadr needs to find the kernel are not valid anymore. If this happens the gumstix needs to be sent in to customer care. To be sure to not make this mistake the boatloader should be replaced with the new bootlader buildroot generated everytime the filesystem is replaced.

- `uImage`: This is the kernel.

- `rootfs.arm_nofpu.jffs2`: This is the actual filesystem containing the linux and the installed packages. Before flashing to the new image it should be made sure that this image has beed padded correctly. 15466496 Bytes is the size of the image after padding for the 16MB flash memory verdex platform. If the above steps in configuring buildroot have been done it should have been padded to this size by default.

The three files mentioned above need to be copied via SCP to the `/tmp` directory on the gumstix. A SSH connection can be done either by a serial Bluetooth, an ethernet or a wireless connection. Afterwards the mountpoints of the different partitions (filesystem and bootloader) have to be located. This can be done via the `mount` command on the SSH shell as described in Figure **??**.

```
1 mount | grep mtd
2 /dev/mtdblock1 on / type jffs2 (rw,noatime)
```

Figure 3.5: Finding the root filesystem

In this case the filesystem is on the mountblock 1. U-Boot is always one partition below the filesystem, so in this case it's located in mountblock 0. An alternative way to find this out is to use the `df` command which lists the space on all available partitions that are mounted. One can search there for the 15.8 MB partition which is the root filesystem. If this is done, the actual flashing process can begin.

```
1 modprobe sa1100_wdt margin=1
2 mount -o remount,ro /
3 flash_unlock /dev/mtd0
4 flashcp -v /tmp/u-boot.bin /dev/mtd0
5 echo && flashcp -v /tmp/rootfs.arm_nofpu.jffs2
6         /dev/mtd1 && flashcp /tmp/uImage /dev/mtd2
7         && echo > /dev/watchdog
```

Figure 3.6: Flashing to the new image (note that line 5-7 is a single command)

The first line in figure 3.6 loads the watchdog module into the kernel that is used in the last command to reboot the platform. Afterwards the root filesystem is mounted read-only in line 2. In line 3 the partition where the booloader resides is beeing unlocked for flashing and in line4 the actual flashing of the new image is done. The last command starting in line 5 flashes the new root filesystem and finallly the kernel to the chip. The output is piped into the watchdog device loaded before so it can automatically trigger a reboot if the command has been completed successfully.

After reboot the new software should be available on the platform.

3.3.3 Adding a Autostart Script

Up until now the wireless lan has to be enabled manually. Now a script will be added so the wireless lan interface and all the other services like dnsmasq will be started automatically so that after powering the platform up, no additional actions need to be taken to make the device fully functional.

An autostart script in embedded linux needs to be simply placed into the `etc/init.d` directory of the gumstix. The naming scheme of these scripts is always S <TwoDigitNumber> <ScriptName> where the two digit number determines the sequence in which the scripts are called. In this case it is called S60Autostart, since the network interfaces need to be powered on before the script can take effect. Such a script need to support the `start|stop|restart` interface, where in most cases it is sufficient to call `stop && start` for the restart procedure. More programs can be added to the autostart simply by copying this script and changing the code for the start and stop procedures. It should not be forgotten to make the script executable via `chmod`. This should be done in the $BUILDROOT/build_arm_nofpu/root/etc/init.d folder so that it is executable right from the start and not does not have to be made executable on each gumstix.

3.4 DTN on the Gumstix Platform

There are 2 possibilities to bring the dtn2 framework to the gumstix platform. The first and less desireable one is to compile the sourcecode from scratch and copy the compiled binaries directly. This is not a god solution since the copy procedure has to be done on each node separately. The second alternative to build a buildroot package for the dtn2 implementation and integrate it into the buildroot environment. So it will be compiled directly into the chip image and doesn't need to be copied seperately.

The buildroot package is available on the internet[1] but there are some changes that need to be done in order to include the package into buildroot and do a successful compile and build of the chip image. In addition to the package which contains only meta-data the actual sourcecode of the version 2.40 is required. The package would download the sourcecode automatically, but there have to be done some changes so the sourcecode needs to be downloaded and manipulated. The three essential parts of the system are the tcl library, a backend database like berkeley db and the dtn2 system itself. For the dtn package to compile the tcl and berkeley db need to already be compiled on the host computer.

- **tcl:** While dtn2 commpiles it searches for the tcl header files and libraries on the host computer. That means that tcl hast to be compiled with buildroot before dtn2 is compiled. After compiling tcl into the chip image, the folders `/usr/local/tcl-arm/include` and `/usr/local/tcl-arm/lib`

[1] http://fs-prisms.blogspot.com/2007/08/dtn-overview-and-instructions.html

on the local machine have to be linked to the arm-compiled headers and libraries of the buildroot, that can be found in `BUILDROOT/build_arm_nofpu/root/include` and `BUILDROOT/build_arm_nofpu/root/usr/lib`. After the image is deployed on the platform `/lib/tcl8.4` needs to be linked to `/usr/lib/tcl8.4` or else the tcl installtion will not be found by the dtn2 framework.

- **berkelydb:** In the buildroot packages directory the berkeley makefile has to be edited since it does not compile arm-mutex support by default. The changes described in Figure 3.8 have to be made to the file, so the mutex option will be compiled with the database.

Now the dtn2 package can be copied into the `BUILDROOT/package` directory and a entry has to be made inside the Config.in for the package so it can be seen in the MainMenu that appears when doing a `make menuconfig`. In addition a small change has to be made to the sourcecode of dtn2. In the `DTN2/oasys` folder the flag `-I/usr/local/tcl-arm/include` has to be added to the LDFLAGS found in the Rules.in file. Else the tcl won't be found and it will not compile successfull.

After doing this, the image can be recpmpiled and deployed as described above.

3.5 DTNPerf

```
1  #!/bin/sh
2  #
3  # Autostart of services needed
4  #
5
6  start() {
7          echo "Starting adHoc..."
8          adHocInit GumNode01
9  }
10 stop() {
11         echo -n "Stopping adHoc..."
12         ifconfig mwlan0 down
13         iwconfig mwlan0 mode managed
14         ifconfig mwlan0 up
15 }
16 restart() {
17         stop
18         start
19 }
20
21 case "$1" in
22   start)
23         start
24         ;;
25   stop)
26         stop
27         ;;
28   restart)
29         restart
30         ;;
31   *)
32         echo $"Usage: $0 {start|stop|restart}"
33         exit 1
34 esac
35
36 exit $?
```

Figure 3.7: Autostart Script for the Gumstix Platform

```
1 --with-mutex=ARM/gcc-assembly \
2  CFLAGS=-fPIC \
```

Figure 3.8: Changes to the berkeley makefile

Chapter 4

Deployment

Thoughts on deoployment issues

4.1 The GumDaemon

For the DTN to work, the end device needs to be connected to the Gumstix network. Since the end devices are moved around, a solution has to be found that does an autoconnect to the network, as soon as it is reachable. This can be done by running a Daemon that scans the available networks and thus finding Gumstix networks. For the tests the networks are simply identified by a special naming pattern in their .
The is a daemon written in perl that can solve the problem of autoconnecting to available networks. It is basically a finite state machine that describes the process of setting up wireless connections and exchanging data using the DTN system. An overview of the state machine can be seen in Figure 4.1. Each state is able to return if the operations performed were successful or not, in addition each state is able to throw a fatal error that willl terminate the state machine. The fatal error is used in cases when for example the driver cannot be contacted or the privileges are set wrong, so the daemon cannot access all the ressources that are needed. In order to work properly the daemon needs root privleges so it can load into the to do certain workarounds for intel chipset drivers, which will be described later on in this chapter. The .gumrc also needs to conatin the correct system specific values. This will be covered in the "Configuration" section.

4.1.1 The statemachine

The state machine starts with the initial "Start" state and ends with the "Terminate" state. If nothing special happens, the machine always loops the states between the "Searching" state and the "Connected" state, which resembles the process of finding networks, transmiting data while the end device is connected and searching for a new network while the device isn't connected. We will now look at the states

in greater detail.

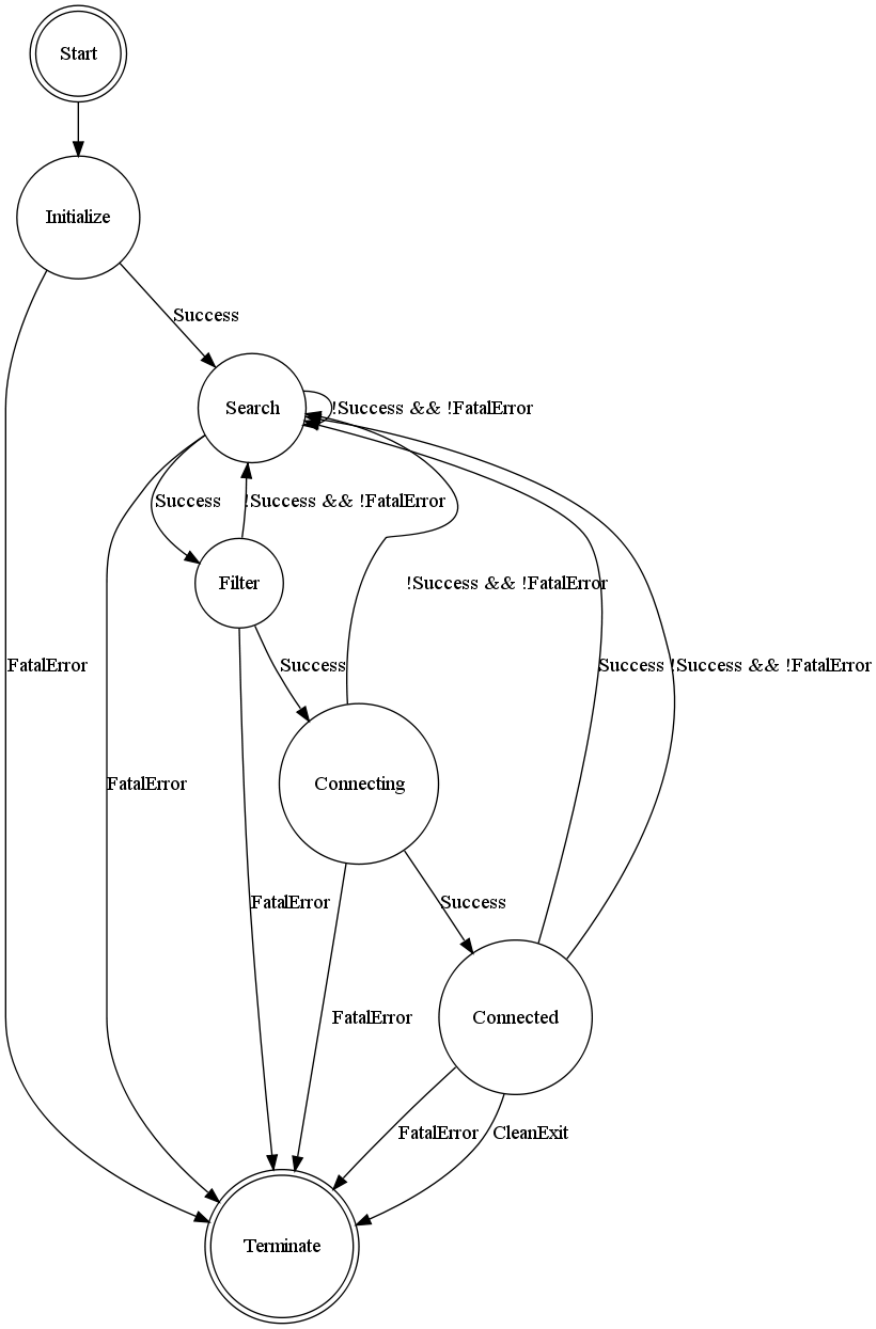

Figure 4.1: The GumDaemon Automata

- **Start:** At the program start all the state machine itself is initialized and the

.gumrc config file is processed, aswell as the debug output and levels.

- **Initialize:** In this state, the driver is put into ad-hoc mode, if it isn't already. This is one of the steps that need root privileges since putting a wireless driver in ad-hoc mode under linux requires the interface to be shut down. How this can be done is shown in Figure 4.2. If this succeeds the state terminates with a success, else the communication with the driver couldn't be established and the state throws a fatal error which will trigger the daemon to terminate.

- **Search:** Each time this state is entered, a scan of the surrounding networks is produced and a pre-filtering is done, meaning that only those networks are included in the results given to the next state, which match a certain naming pattern[1]. If at least one network is found, the success flag is set and the following state will be the filter state. Else the search state is entered again. There is also the possibility for a fatal error here, when the driver is not responding while scanning.

- **Filter:** A restult array is given to this state and a filter method is applied to the array. As a rsult only a single network will be returned and given to the next state. The default filter method that is used here is that the network with the highest link quality is chosen. Link quality calculates from the signal and noise of the target network. Basically it's signal devided by noise and scaled to a value between 0 and 100. The filter succeeds if there is one network left in the results of which the ESSID will be given. If no networks are left after applying the filter, the followup state will be the search state again.

- **Connecting:** In this state, the filtered ESSID is applied to the provided network adapter. Afterwards the adapter is assigned a IP, either this is a static IP or a dynamic IP, which depends on the configuration of the daemon. The state succeeds if a connection is established, this is tested by sending a ping to the Gumstix Node that has beeen found earlier. If the ping doesn't return, it is assumed that the node isn't available anymore and thus the daemon will start looking for new nodes.

- **Connected:** This is the state where the actual work is done. Here the DTN daemon is advised to send or retreve data into or from the network. The connection is checked frequently via pings so that a conection interruption can be early determined and hopefully the connection can be reestablished. Independent from the result, whether it was succesfull or not, the followup state is the search state again, so the daemon can find other networks. Nevertheless the success flag is still used to make a transition to the search state, so that a reaction is basically possible. Furthermore this is the only state that can terminate the deamon without throwing an error, and advise it to do a

[1] In the test cases this is /GumNode\ d\ d/

clean exit if for example all files that needed to be send are sent, or if all the files that the daemon wanted to recieve have arrived.

```
1 ifconfig wlan0 down
2 iwconfig wlan0 mode ad-hoc
3 ifconfig wlan0 up
```

Figure 4.2: Putting a driver into ad-hoc mode

4.1.2 Configuration

The configuration of the GumDaemon is done in the file. This file should be in the home directory of the user that executes[1] the GumDaemon. When the GumDaemon starts it uses the file as an input which means, that not only some configuration variables can be set there, it is possible to include methods that are executed by the daemon befor the actual startup. This means also that one can override the methods called in the states in order to execute another filter for example, without changing the actual code. An example of a .gumrc can be seen in Figure 4.3.

```
1  #This is an example .gumrc
2
3  $interface = "wlan0";
4  $scanRate = 3;
5  $ownIP = "192.168.2.13";
6  $dhcpCommand = "dhclient";
7  $driverModule = "iwl4965";
8  $activateIntelWorkarounds = 1;
9  $gumStickIP = "192.168.2.1";
10 $pingTimeout = "2";
11 $debugLevel = 1;
12 $debugMethod = "Console";
13 $logfile = "/var/log/gumDaemon.log";
```

Figure 4.3: Example .gumrc

Here is a basic overview about the parameters that can be altered:

- `interface`: This is the network interface, of the end device. This is "wlan0" on an average system. The interface can be seen if the command `iwconfig` is run on a shell.

[1] This should be root in most cases and thus the file would be placed in /root/.gumrc

- `scanRate`: This is the time in seconds between scanning intervalls for new gumstix nodes is done The interval is applied whenever a scanning process has failed, no node has been found, or if an action in the connected state has finished and the searching state is entered again.

- `ownIP`: This is the IP address that will be assigned to the adapter after a connection has been established. The value can be either an IP address or it can be set to "dynamic" which will trigger the dhcp-client to automatically obtain an IP address from the connected node.

- `dhcpCommand`: The command to be run to obtain an IP from a client.

- `driverModule`: The name of the driver module that is loaded on the kernel, the name can be determined by entering the command `lsmod` in a shell.

- `activateIntelWorkarounds`: If this flag is set, the daemon resets the driver before doing certain steps, this should always be true for intel class drivers since the implementation of the ad-hoc mode is rather buggy.

- `gumStickIP`: This is the IP of the gumstix node, it is used for sending out pings to ensure the connection has been properly established. For our test cases with a single node this IP is set to a fixed value, while when working with more than one node the last part of the IP will resemble the node number of the ESSID.

- `pingTimeout`: The time in seconds after which the ping that was sent out to test the connection will be considered as having a timeout.

- `debugLevel`: The debug level determines the verbosity of the output. Currently there are 4 levels of debug output implemented:

  - Level 0: Only system messages are displayed and absolutely no debug output.
  - Level 1: The output of the state machine is displayed, which states are going to be entered and the status with which the previous state has finished
  - Level 2: Status information about what is being done inside the states is displayed in addition to the normal state machine output.
  - Level 3: The output of the major varibles will be displayed too in certain states, like for example the exact values of the filtered networks.

- `debugMethod`: This can be one of the following values

  - Console: The debug messages will only be printed into the console.

  - Log: The debug messages will be written in the provided logfile. If no logfile is specified it will be written in the standart logfile in /var/log/gumDaemon.log
  - Hybrid: The debug messages will be displayed and be written into the logfile

- `logfile`: A logfile can be specified here, normally this isn't recommended since all linux logfiles are in /var/log, but in certain cases this can be necessary.

In addition to these basic variables, one can put arbitrary perl code in the .gumrc. Since it is a rc file it is directly included and run after the initialization but before the actual execution of the daemon.

If new methods are needed, like for example a new filtering method, it can be added as an override of the normal filter method. So the code doesn't need to be touched in order to exchange the called methods inside the states. This also doesn't have an effect on the state machine itself, because the state machine code is kept seperate from the code that is executed inside the states. The only thing that needs to be addressed when overriding given methods is, that the `$success` and `$fatalError` variables need to be set inside the overridden methods for the state machine to work properly.

4.1.3 Workarounds

Depending on the hardware that is used, several workarounds need to be done since under Linux systems with certain kernels and certain drivers ad-hoc mode will not, or most likely will not work properly.

Most of the notebooks come with an intel chipset for wireless networking and thus will use the standart intel linux driver for thier adapter, in most cases this will be the "iwl4965" driver module. At the time being, the most common kernel that ships with the distributions is the 2.6.24 kernel. With the combination of this kernel and the intel driver wireless networking in ad-hoc mode will not work. Up to now (kernel 2.6.26 are the lates sources) this bug hasn't been fixed.

The solution is to use an older kernel. Ad-hoc mode worked successfully on a 2.6.22 kernel with the latest driver with some limitations. The first one is, that the driver module needs to be reset after each connection. This can be done with the `rmmod <ModuleName>` to remove the driver from the kernel and the `modprobe <ModuleName>` to load the driver into the kernel again. This will reset the whole driver. The second limitiation is that during scan, the link quality aswell as signal and noise values are visible, but if the connection has been established those values are not forwarded to the driver, which means that when a query is made to driver about the quality of the current connection, it will always say the quality is 0. Therefore the daemon need to sond out pings in order to determine if the connection is still there and can't simply watch over the link quality provided

by the driver. This behaviour should be changed as soon as this driver bus is fixed.

Chapter 5

Conclusion

cute stuff??

5.1 Results

5.2 Outlook

Glossary

ad-hoc

Lorem ipsum dolor sit amet, consectetuer sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. I

Delay Tolerant Networking

Lorem ipsum dolor sit amet, consectetuer sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. V, 3

List of Acronyms

DTN Delay Tolerant Networking

Index