STRATEGIC PLATFORM FOR INTELLIGENT TRAFFIC SYSTEMS

GCDC Communications Stack

Building and installing the CALM FAST router

Jan de Jongh

FOR PUBLIC RELEASE

Describes Version 2 of the GCDC Communication Stack

Document reference :	**not assigned**
Authors:	**Jan de Jongh**
Document version:	**2.0**
Document status:	**Concept**
Date:	**23/03/2013**

Contents

Version History (of this document)

Date	Version	Status	Author	Comments
27-08-2010	1.0	Draft	Jan de Jongh	Initial version from GCDC R&T v1.0 (not released).
29-10-2010	2.0	Concept	Jan de Jongh	Updated to use of CALM FAST router by Eric Koenders.

References

[1] Koenders, E., CALM FAST router, design, v0.1, SPITS Deliverable, 06-03-2010.

[2] GCDC Interaction Protocol, version 2, 25-10-2010.

[3] GCDC Technical Architecture, to appear.

[4] GCDC Rules and Technology Document, version 1.0.

[5] GCDC Technology Workshop 2011/Q1, to appear on Nov 1st, 2010.

[6] GCDC Event 2011 – Technical Aspects, to appear.

[7] GCDCCommStackV2.tgz, distributed through http://www.gcdc.net.

[8] The Liux Wireless wiki, mac80211 kernel module, http://wireless.kernel.org/en/developers/Documentation/mac80211.

[9] The Linux Wireless wiki, http://linuxwireless.org/.

Glossary

BO	Back Office
C2C-CC	Car-2-Car Communications Consortium
GCDC	Grand Cooperative Driving Challenge
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
ITS	Intelligent Transportation Systems
LLC	Logical Link Control
MAC	Medium Access Control
OBU	On-board Unit
RSU	Road Side Unit
SPITS	Strategic Platform for ITS
TCP	Transport Control Protocol
UDP	User Datagram Protocol

1 Introduction

1.1 Purpose and Organization of this Document

This document describes the (updated) communications stack for the the 1st Grand Cooperative Driving Challenge (GCDC), to be held in 2011, and provides documentation for support code for a reference implementation. It deals with all aspects of the communications stack, except for the actual payload (message content), which is described in [2].

By the end of 2009, the GCDC organization became formally part of the SPITS (Strategic Platform for ITS) project [1], while retaining the original objectives, open development attitude, and basic technical architecture of the GCDC. This explains the use of the SPITS logo and document templates. The SPITS project plays a vital role in the organization, roadside deployment and financial structure of the GCDC.

This document supersedes the information provided in [5].

1.2 Other Relevant Documents

The following documents provide additional information on wireless communications (and technology in general) in the GCDC:

- **CALM FAST router, design, v0.1, by Erik Koenders [1]:** The reference document on the CALM FAST router and the open-source IEEE 802.11p driver (included with the GCDC Communication Stack distribution, and used with permission).
- **GCDC Interaction Protocol [2]:** This document describes the so-called *interaction protocol* of the 1st Grand Cooperative Driving Challenge (GCDC), to be held in 2011. The interaction protocol describes the format of the payload of the messages exchanged between vehicles and between vehicles and infrastructure during the GCDC, as well as their semantics and expected sequence.
- **GCDC Technical Architecture [3]:** This document provides an up-to-date technical framework for the GCDC. It ties together all relevant GCDC (and SPITS) technical components.
- **GCDC Rules and Technology Document [4]:** This document serves as a starting point for the Grand Cooperative Driving Challenge, not only for its technical aspects, but also for the organizational, promotional and legal aspects. It also describes the scenarios in more detail. Please note that as both the GCDC 2011 workshop and event approach rapidly now, more and more technical aspects in [4] are obsolete, as it is no longer feasible to keep [4] up-to-date unless by unreasonably short revision lifetimes.
- **GCDC Technology Workshop 2011/Q1 [5]:** This document provides in-depth technical details of the technology workshop to be held in Q1 of 2011. If you want to know how we intend to test the communications stack, interaction protocol, positioning, and other technical issues related to the workshop, this is the document to read. In [5] we also provide operational details like address (block) assignments, radio frequencies for RTK-GPS, vehicle Ids, test scripts, etc.
- **GCDC Event [6]:** This document has the same purpose as [5], but relates to the actual GCDC event in May 2011. It is an update of [5].

1.3 Historical Background

The current section provides our motivation for the choices made for the GCDC communications stack and the GCDC interaction protocol. You may want to skip this section if you just want to "make things work".

The origins of the Cooperative Driving Challenge go back to 2008 when Egbert-Jan Sol at TNO launched the idea of an open, technical competition for cooperative systems stakeholders like car manufacturers, first-tier suppliers, academic and research organizations and anyone else interested in the field. The main objective was (and still is) to catalyze the design, realization and deployment of aforementioned systems. Another implicit objective is to demonstrate the potential of cooperative systems for improved safety, efficiency, environment and comfort.

It was soon decided that GCDC would follow an open, multi-vendor approach towards the communications (OSI layers 1 through 5) and interaction protocols (OSI layer 6). After all, cooperative systems can only work in practice if cars from different vendors "speak the same language", at least to some extent. The direct implication was that the GCDC organization would have to set the rules for communications (modulation, frequency, channel coding, etc.) and interaction (message content). Naturally, the organization would have to waive any Intellectual Property Rights on the specifications.

That left us with only one remaining problem: which protocols to choose?

For the lower levels (layers 1+2) of the protocol stack, this seemed easy at first sight, as IEEE 802.11p was rapidly becoming the de facto standard (albeit with a Draft Status) for ITS communications, and frequencies in the 5.9 GHz band were becoming available, at least in the US and Europe. Moreover, the FP6 project CVIS had realized an open-source implementation of IEEE 802.11p based upon a Debian-Linux distribution, and its sister project Safespot had worked upon defining message sets for the higher layers (they did much more than that, actually). Both projects had been running for a few years and their results seemed usable for GCDC.

However, there were serious complications.

The CVIS implementation relied heavily upon dedicated hardware that was expensive and not widely available. In fact, there were serious indications that the hardware would no longer be produced by the time we were specifying the protocol stacks. Moreover, CVIS relied on closed-source components (a Hardware Abstraction Layer) that could not be distributed as open-source nor modified to suit our particular needs. Also, CVIS did far more than just IEEE 802.11p. It implemented a fully functional ISO CALM protocol stack, which implied the addition of many CALM-specific modules to the kernel. It appeared very difficult to "extract" the changes from the CVIS kernel and port it to other Linux distributions, let alone to other, better available (and cheaper) WLAN hardware. The CALM framework also mandates many changes to the default, de facto Linux communications stack (e.g., CALM management functions), as well as a dual-host implementation (Host + Router) for ITS applications.

With respect to message definitions, things did not look much better: We could not distribute the Safespot message definitions except to Safespot consortium members. A more severe objection was that the Safespot message set was simply deemed too complicated for use in GCDC. Since a GCDC-specific message set seemed feasible, it was decided to do things ourselves, resulting in the GCDC Interaction Protocol [2].

For the lower levels, we were lucky to team up with Eric Koenders of Peak Traffic who shared our concerns about widening the availability of IEEE 802.11p (already during the CVIS project), and he pioneered a generic, light-weight, open-source IEEE 802.11p implementation for Linux [1], based upon the original CVIS implementation. The implementation still relies on parts of ISO CALM though, in particular CALM FAST in favor of IPv4 which was prescribed in the original GCDC specifications [4]. Since using CALM FAST is in line with current activities in standardization bodies like ISO, and in particular ETSI, we decided to adopt Eric's approach for GCDC, at the expense of a rather radical change from the original specification in [4].

1.4 Acknowledgments and Trademarks

In light of the previous section, it goes without saying that the specification of the GCDC Communication Stack relies heavily upon the pioneering work done by others. Even though the list of relevant contributions is probably way too large, we make a futile attempt anyway, acknowledging the work of and thanking the following people:

- The developers, contributors and maintainers of the (various flavors of) the Linux operating system (in particular, Linus Torvalds) for providing and continuously improving an open-source operating system and application framework, greatly assisting the development of new, innovative applications and services.
- Eric Koenders at Peak Traffic for his pioneering work on a (true) open-source IEEE 802.11p and CALM FAST Linux driver.
- The CVIS and Safespot projects for their pioneering work on realizing platforms for cooperative systems.
- The developers of the ath* and MadWiFi WLAN-drivers for Linux.
- The standardization activities in ISO, ETSI, IEEE, and C2C-CC.
- The GCDC visionaries, managers, brainstormers, architects, designers, and programmers, past and present, without whom the GCDC could never take place. In particular, the work of Egbert-Jan Sol, Jan Hein van Ras and Anton Gerrits from TNO is greatly acknowledged. But also that of Otto Baijer, Eric den Breejen, Maarten Ditzel, Miodrag Djurica, Bart Driessen, Stefanie de Hair-Buijssen, Bastiaan Krosse, Kerry Malone, Arnoud van Neerbos, Bart Netten, Rine Pelders, Jeroen Ploeg, Ronald in 't Velt, Diana Vonk Noordergraaf, Thijs Versteegh, Harry Wedemeijer, Nico Zornig.
- Last, but not least, the GCDC participants for their continuous contributions and interest in the GCDC, and for their efforts in realizing the GCDC technical components.
- Anyone I've forgotten and should be severely blamed for...

2 GCDC Communications Protocol Stack

This section described the GCDC protocol stack in an implementation-independent manner. It should provide you with enough information to completely implement the protocol stack while not using the reference implementation described in Section 3.

2.1 Scope

This section describes ISO-OSI/RM networking layers 1 through 5 (Physical Layer through Session Layer) of the GCDC.

2.2 Overview

The figure below summarizes the GCDC Communications Protocol Stack and its reference implementation.

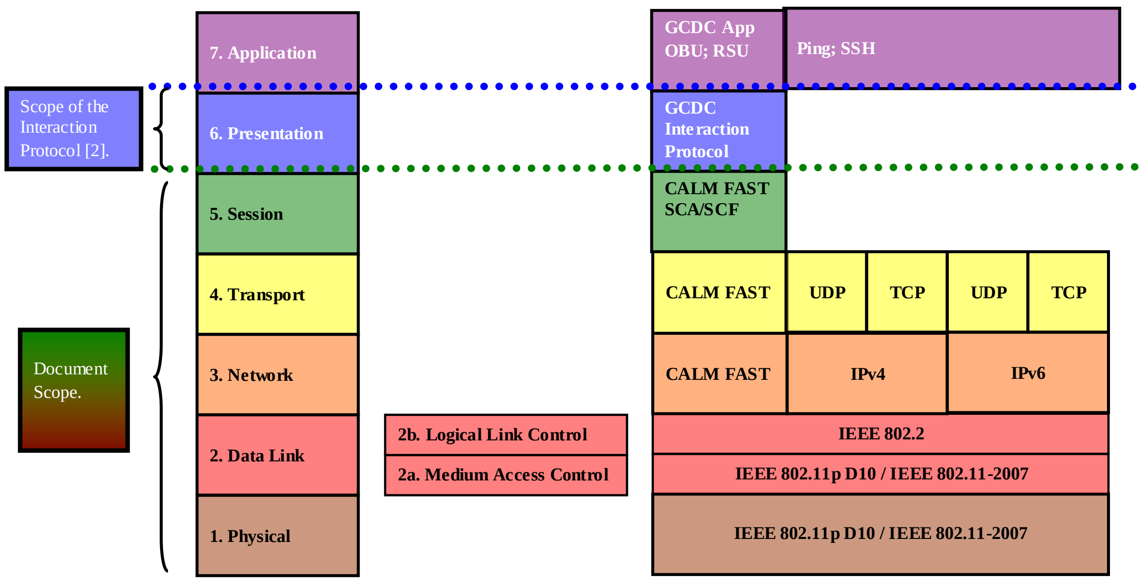

Figure 1: The GCDC Communications Stack

This document only covers layers 1 through 5 of the OSI stack.

2.3 PHY and MAC Layers

The PHY and MAC layers comply with the IEEE 802.11p Draft 10.0 or newer specifications. For GCDC, only (MAC-level) broadcast addressing is needed; there is no ACK, RTS, and CTS. As a result, there is no NAV administration (no *virtual transmissions*).

MAC-level addressing is based upon the unique, hard-coded, 48-bit vendor-supplied MAC addresses.

All transmissions take place in the highest-priority access class (formerly known as IEEE 802.11e AC).

Each GCDC vehicle operates in the ITS Control Channel (CCH) with 10 MHz bandwidth, but must be prepared for evasion to a so-called Service Channels (SCH) so we keep open the option of using a different channel in case the traffic load is too high. Note that frequency evasion will not take place during the event, and will be announced well in advance.

2.4 LLC Layer

The LLC (Logical Link Control) Layer complies with IEEE 802.2.

2.5 NET Layer (Mandatory)

The mandatory NET layer consists of the over-the-air specification of CALM FAST and IPv6. Network and transport addresses are fixed, and specified in [5] and [6].

2.5.1 CALM FAST

The GCDC uses CALM FAST for the Network Layer. Network and transport addresses are fixed, and specified in [5] and [6]

2.5.2 IPv6

In GCDC, IPv6 over IEEE 802.11p must be supported, but is not used during the actual challenge itself. Network addresses are fixed, and specified in [5] and [6]. It is needed for ICMP echo requests (connectivity tests) and remote SSH access.

2.6 NET Layer (Optional)

2.6.1 IPv4

Optionally, IPv4 is supported at the Network Layer. Network and transport addresses are fixed, and specified in [5] and [6]. It can be used for ICMP echo requests (connectivity tests) and remote SSH access.

2.7 Transport Layer (Mandatory)

2.7.1 CALM FAST

The GCDC uses CALM FAST for the Transport Layer. Network and transport addresses are fixed, and specified in [5] and [6].

2.7.2 ICMP, UDP and TCP over IPv6

All GCDC OBUs must support ICMP (echo request/reply), UDP and TCP over IPv6. Transport addresses (‘ports’) are fixed, and specified in [5] and [6].

2.8 *Transport Layer (Optional)*

2.8.1 TCP and UDP over IPv4

The use of TCP and/or TDP over IPv4 is optional (as is IPv4 itself). Transport addresses ('ports') are fixed, and specified in [5] and [6].

2.9 *Session Layer (Mandatory)*

2.9.1 CALM FAST SCA/SCF

All GCDC nodes (vehicles and roadside) must support the periodic broadcast and/or exchange of CALM FAST SCA and SCF frames.

2.10 Mandatory IP Tools

A GCDC OBU must support the following IP tools:

- ICMP echo request/reply over IPv6 ('ping6').
- SSH over IPv6 (for remote access).

SSH access is required, but you do not have to supply the necessary credentials (passwords, certificates) to the GCDC organization. However, the capability to remotely access the OBU in your vehicle is deemed necessary for debugging, monitoring and logging purposes.

2.11 Address, Port, and Service ID Summary

For the detailed assignment of addresses and IDs for the GCDC Workshop 2011/Q1 and the GCDC (1st) event, refer to [5] and [6], respectively.

Note: Your system must allow for run-time changes in these addressing schemes (i.e., without recompilation).

3 Organization of the GCDC Reference Communication Stack

3.1 Introduction

This section describes the reference implementation of the GCDC Communications Stack, as implemented on the roadside equipment at the Helmond Test Site, the main location of the GCDC 2011 event. The communications stack is based upon the pioneering work by Eric Koenders at PeakTraffic, who created an open-source IEEE 802.11p kernel driver (for Ateros-AR-5414-based WLAN cards, aka IEEE 802.11a cards), and a user-space implementation of CALM FAST on top of the driver.

The driver and user-space daemon were the starting point for the communications stack of the entire Helmond Test Site (i.e., for all projects and demonstrations that would take place, like the Shockwave Damping experiment), and were extended quite heavily with Java-based code over the course of 2010. For GCDC, we decided to 'branch off' the original code by Eric Koenders, and distribute it separately to the GCDC participants through the GCDC web site [8].

Although GCDC does not mandate the use of this software, participants are strongly encouraged to study it carefully, and at least test the interoperability of this software with their own communications stack.

This section describes the main components and internal structure of the software distribution, but does not go into detail on building, installing and deploying the software, which is described in Sections 4. Section 6 provides details on the process of bug-fixing, change requests, version control and repository management. Not all details in that section may be relevant to GCDC participants, but we provide them anyway in order to keep the information related to the GCDC communications stack packed in a single document.

The reference implementation of the GCDC Communications Stack is mostly taken from Eric Koenders CALM FAST router implementation. The main reference is [1], which is considered essential reading for the remainder of this document. The reader is assumed to have knowledge on the Linux kernel structure and the kernel build process (for his or her favorite Linux distribution), and some knowledge on Linux wireless kernel drivers.

3.2 Notational Convention

We use `Courier` typeface for source code, commands, and file names. We follow the usual convention of pre-pending a command with a `$` prompt if it can (should) be executed as a regular user, and a `#` prompt if root privileges are required (either through login, or with the `su` or `sudo` commands).

3.3 Main Components

The distributed software consists of the following main components:

- Updated kernel modules for IEEE 802.11p support:
  - `ath5k`,
  - `mac80211`, and

   - wireless.
- Several required (modified) CALM and CALM-FAST related support libraries from the CVIS project and from Eric Koenders:
   - libcalmfast,
   - libcalmmedia,
   - libcam, and
   - libfast.
- An update of the user-space iw tool for runtime configuration of the modified wireless driver.
- The CALM FAST router designed and realized by Eric Koenders.
- Documentation, including the present document.

3.4 *Unpacking the Distributed TarBall*

The software is distributed as a gzip'ed TAR file [7], which can be unpacked in the current working directory on any reasonable Linux distribution with the 'tar' command:

```
$ tar -xzvf GCDCCommStackV2.tgz
```

Note that you may have to change the version number in the filename.

3.5 *Annotated Structure of the Distributed TarBall*

The distributed tarball [7] has the following structure:

doc/

Documentation.

CALM-FAST-router-design.odt

Eric Koenders' documentation [1].

CALMFAST-Service-ID-list.txt

Assigned CALM FAST Service ID's for the SPITS Test Site.

GCDC-Comm-Stack.doc

The present document.

src/

Source code.

calmd/

CALM FAST router source code by Eric Koenders (see Eric Koenders' document [1], Section 3 for details, and Section 4.5 in this document for building the daemon).

iw-0.9.19/

Modified user-space tool for `nl80211` interface (see 4.4.1).

`kernel/`

Modified kernel modules.

`ath/`

Modified `ath` kernel driver for Ateros-based WLAN cards (see Eric Koenders' document [1], Section 2 for details, and Section 4.3.1 in this document for building the driver module).

`mac80211/`

Modified `mac80211` kernel driver (see 4.3.1). For details on the driver, see [9] from which the following section is verbatim copied:

'mac80211 is a framework which driver developers can use to write drivers for SoftMAC wireless devices. SoftMAC devices allow for a finer control of the hardware, allowing for 802.11 frame management to be done in software for them, for both parsing and generation of 802.11 wireless frames. Most 802.11 devices today tend to be of this type. mac80211 implements the cfg80211 callbacks for SoftMAC devices, mac80211 then depends on cfg80211 for both registration to the networking subsystem and for configuration. Configuration is handled by cfg80211 both through nl80211 and wireless extensions.'

`wireless/`

Modified `wireless` kernel driver (see 4.3.1).

`lib/`

Support libraries.

`libcalmfast/`

The CALM FAST library as designed and realized by Eric Koenders [1]. It specifies (in ASN.1) the client-side protocol of the CALM FAST router daemon. The main targets of this directory are `libcalmfast` itself and its supporting header file `calmfast.h`. The CALM FAST router in `calmd/` depends on this library.

`libcalmmedia/`

A library for CALM media support.

`libcam/`

A library for support of Cooperative Awareness Messages (CAM), as specified in the Safespot project, and currently being standardized by ETSI.

`libfast/`

A library for the FAST protocol, especially the message formats.

`linux/`

According to the README file in this directory: 'Some linux header files from the CVIS build. These are needed to be able to build `libcalmmedia` and `libfast`.'

`wireless-crda/`

The agent for regulatory domain enforcement.

`wireless-regdb-2009.11.25/`

Toolset for creating and installing a self-signed regulatory database.

3.6 Repository Details and Version Control

All files described in the previous section are under strict `SubVersion` version control. The table below provides the repository details. Access to the repository is restricted to authorized members of the Spits project, though.

URL	**Purpose**
https://www.spits-project.com/repos/Spits/wp04/Comm Stack	SPITS-wide comm. stack.
https://www.spits-project.com/repos/Spits/wp04/GCDC/GCDCCommStack	GCDC branch.

3.7 Legal Issues

All software described in this document is provided AS IS, without any explicit or implicit warranty or fitness to a particular purpose. Using this software is completely at you own risk and TNO and all affiliate organizations hereby deny all liability related to the use of this software.

We have gone through substantial effort to ensure that we have the legal right to distribute this software and its accompanying documentation, and to respect trademarks and copyright claims. Please contact the author in case of omissions or infringements.

4 Building, Installing and Deploying the GCDC Reference Communications Stack

4.1 Introduction

In this section we provide information on how to build the GCDC communications stack, more or less in a log-like fashion as a result of trying the build ourselves. The approaches outlined in this section may not appeal or even apply to everyone, as solving build errors is pretty much a matter of personal taste. However, I had some constraints during the builds, most notably the constraint that I was working on a Fedora-13 production machine, and did not want to mess up any part of its installation. I also wanted to complete the build without full kernel recompilation (while still using the kernel headers), since kernel recompilation was not an option. I suspect for most participants, the situation is different, as we expect many dedicated kernels on embedded OBU platforms.

My personal preference is to build software bottom-up as far as possible if a top-level build fails (as it did in my case). I therefore started with the libraries, then moved into the kernel modules and ended up with the user-space daemon from Eric and the tools.

Another personal preference is to postpone *any* installation until the build succeeds completely, or at least postpone it as long as possible. Most `Makefile`s do a proper job at supplying a reasonable install target, but removing installed targets from a partially succeeded build can be somewhat of a hassle. I usually do a single `make install` from the top-level directory after the build, and capture the output into a file so I know where they went. Another argument for taking this approach is that installing software requires in general much higher privileges than building it.

Again, these are just personal preferences, and we are open to suggestions for other approaches so we can include them into this documentation.

Please note that I did *not* commit any of the proposed changes during the build to the original software supplied by Eric Koenders, since for the moment, I want to keep both software trees in sync for as long as possible.

4.2 Libraries

4.2.1 libcalmfast

Building libcalmfast is done through

```
$ make
```

in the `lib/libcalmfast` directory. The build completes successfully apart from a few seemingly harmless warning on signedness of some variables related to encoding (this warning relates to `asn1c` supplied code).

The regen.sh script allows you to regenerate the ASN.1 encoding/decoding source files. Please be advised to only run the script if you are sure you have a working asn1c compiler, since I have seen several warnings and patches for asn1c in this and in other projects. Also, the script starts off by deleting all generated asn1c output, and you may have to reinstall them from the distribution.

We are interested in your experiences with asn1c.

4.2.2 libcalmmedia

A first attempt at building libcalmmedia is to simply invoke

```
$ make
```

in the lib/libcalmmedia directory. The build fails because there is an unresolved include directive for <linux/calm.h>. There are at least five conceivable approaches towards fixing this, in descending order of (my) preference:

1. Modify the Makefile in order to make the compiler look for the proper directories, minding the prefixes (like linux/) already present in the relevant include directive.
2. Create appropriate symbolic links and/or subdirectories if needed in the build directory. I tried this but it failed, because the compilation does not look in the current working directory in order to find system-wide include files (a very reasonable strategy, by the way). Other than that, it is a strategy I often use as many builds have **-I.** in the compiler argument list.
3. Copy the lib/linux header files under linux to the system-wide include directory, most likely /usr/include. An alternative is to use /usr/local/include. I did not try this as I did not want to add or overwrite anything there (it is under yum package management anyway).
4. Copy and paste the command executed last as a result of make, insert appropriate command-line arguments for finding the missing include files. This approach has the severe disadvantages that you have to repeat this action on every build. Moreover, this approach is not very suitable for recursive builds from a higher-level directory.
5. Modify the offending include directive in the source file, but this is something you do not want to do as it makes the build process highly location dependent, and impedes any updates that might arise.

I chose the first approach and added **-I..** to the CFLAGS line in the Makefile. I had to make sure that -I.. was added before all other -I directives; otherwise the build would use the system-wide <linux/sysctl.h>, and compilation still failed in my case. (I did not understand why that happened, but the sysctl.h in the Eric Koenders' distribution is clearly different from the one on my Fedora 13 system.)

4.2.3 libcam

A quick glance at its Makefile reveals that building libcam should be as easy as

```
$ make
```

in the `lib/libcam` directory. Unfortunately, this does not work straight away and compilation fails because of missing header files:

- `calm/calmmedia/mininl.h`, and
- `linux/calm.h`.

The first file is part of `lib/libcalmmedia`. The latter file is supplied in the `lib/linux` directory, and the related problem is easily solved by pre-pending `-I..` to the `CFLAFGS` section in the `Makefile`, as described in the previous section. The first problem occurs as a result of not yet installing the `libcalmmedia` header files in their proper places. Because the build already looks in the parent directory for system include files, a somewhat quick-and-dirty solution is to create the path to `mininl.h` relative to the parent directory (make sure the first command puts you into the `src/lib` directory):

```
$ cd ..
$ ln -s . calm
$ ln -s libcalmmedia calmmedia
$ cd libcam
$ make clean
$ make
```

The build now succeeds. Obviously, you are left with two symbolic links in the `src/lib` directory that were not there in the first place. However, since they are symbolic links pointing to directories nearby, hopefully they are easily recognized as hacks for the build process. Leave them in place for now, since we need them for the next library as well.

Please note the warning on the `regen.sh` script in Section 4.2.1.

4.2.4 libfast

In order to build `libfast`, we must again prepend `-I..` to the `CFLAFGS` section in the `Makefile`, as described in the previous sections. Also, you need the symbolic links in the parent directory `src/lib` as described in Section 4.2.3 (of course you kept them in place). A successful build now merely requires

```
$ make
```

in the `lib/libfast` directory.

4.2.5 Installing the libraries

By now, we have reached the first milestone: We have successfully compiled all the required libraries completely local. All it took were minor changes to three out of four `Makefile`s, and two symbolic links in the `src/lib` directory.

However, we did not install the libraries yet. This step is required in order to build most of the remaining software. Make sure to execute the following command as root in each of the library subdirectories:

```
# make install
```

This will install the libraries and the accompanying header files under `/usr/local`. You can check this beforehand with:

```
# make -n install
```

4.3 Kernel Modules

Building the kernel modules actually depends on whether you have a 2.6.31 kernel, or a newer one. In the latter case, building the source tree does not work, proceed to Section 4.3.2. If you are still running a kernel older that version 2.6.31, our advice is to upgrade to 2.6.31, though the approach for 2.6.31 might still work. If you need additional information on the original source of the drivers, the best starting place is the Linux Wireless wiki [9].

We strongly recommend running a 2.6.31 kernel, since the approach for newer kernels is not a success story yet!

4.3.1 ath*, mac80211, and wireless for 2.6.31 kernels

For a 2.6.31 kernel, building the kernel modules (for the *running kernel*!) is easy. From the `src/kernel` directory, enter

```
$ make
```

to build the modules and

```
# make install
```

to install them.

4.3.2 ath*, mac80211, and wireless for kernels more recent than 2.6.31

If you have a kernel more recent that 2.6.31, building the modules as described in the previous section fails, because the kernel headers and the driver software are inconsistent.

Note: This section is quite experimental, has a relatively low rate of success, and requires more advanced Linux kernel building skills. If at all possible, stick to a 2.6.31 kernel until we have solved the build problems.

Our approach is to first create patch files using the `makepatch.sh` script in the `src/kernel` directory (*do not perform this step unless you have a working copy of the SubVersion repository; otherwise use the patch files in the distribution*):

```
$ ./makepatch
```

which creates three patch files, `its-ath.patch`, `its-mac80211.patch`, and `its-wireless.patch`. These patch files contain the changes in diff format that Eric Koenders made to the original drivers from http://linuxwireless.org, and our strategy is an attempt to apply these patches to a more recent version of the drivers. Note that the `its-mac80211.patch` is (at this time) empty.

Note to developers: please do not commit these patch files into the repository!

Next, follow the instructions on http://linuxwireless.org/en/users/Download/stable/ and select a suite of wireless drivers that best matches your kernel version. Store the downloaded `compat-wireless-2.6.x.y.tar.bz2` and unpack it with the tar command:

```
$ tar -xf compat-wireless-2.6.x.y.tar.bz2
```

Since you only need the ath5k driver; use the `scripts/driver_select` script in the distribution to select it:

```
$ scripts/driver_select ath5k
```

Next, make sure the software compiles properly:

```
$ make
```

However, do not install the modules (i.e., do not yet execute `make install` as suggested on the web page)!

Now, try to apply the patches to the downloaded drivers. First, copy `its-ath.patch` to the directory in the distribution holding the `ath` directory, that is `compat-wireless-2.6.x.y/drivers/net/wireless`, and issue a dry-run of the `patch` command in that directory:

```
$ patch -p0 --dry-run -i its-ath.patch
```

(the -p argument is a zero.) If this works, apply the path:

```
$ patch -p0 -i its-ath.patch
```

Notes:

- Even though the patch file seem to correctly reference the right files to be patched, I was sometimes raised the question which file to patch. However, from the output immediately above that question, it was quite clear which file was to be patched, so the questions could easily be answered.
- ***In general: do not proceed if the patch fails, but try a different kernel version.***
- ***The procedure fails on a 2.6.34 kernel and compat-wireless-2.6.34.***

Now apply the same procedure to

- `its-mac80211.patch` in `compat-wireless-2.6.x.y/net` (where the `mac80211` subdir is) and
- `its-wireless.patch`, again in `compat-wireless-2.6.x.y/net` (where the `wireless` subdir is).

If all went well, re-enter the `compat-wireless-2.6.x.y` directory and build and install the modules:

```
$ make
```

followed by

```
# make install
```

You have now completed building and installing the modified ath5k drivers suitable for IEEE 802.11p for the running kernel. Well done!

In the table below we have shown which `compat-wireless-2.6.x.y` versions can be patched and compiled against different kernels (kernel refers to vanilla Fedora kernels; these have a somewhat questionable reputation of not including everything from the kernel source git tree).

OS	Kernel	Compat-wireless	Patch	Compiles	Runs	Note
Fedora 13	2.6.34.7-56	2.6.32.16	OK	FAIL	FAIL	(1)
Fedora 13	2.6.34.7-56	2.6.33.6	OK	OK	UNKNOWN	
Fedora 13	2.6.34.7-56	2.6.34.1	FAIL	FAIL	FAIL	(2)

Note 1

```
make -C /lib/modules/2.6.34.7-56.fc13.i686/build
M=/mnt/hc/jan/SoftDev/TeamProjects/Spits/SubVersion/Spits/wp04/GCDC/GCDCCommStackBranc
h/src/kernel/test/compat-wireless-2.6.32.16 modules
make[1]: Entering directory `/usr/src/kernels/2.6.34.7-56.fc13.i686'
  CC [M]
/mnt/hc/jan/SoftDev/TeamProjects/Spits/SubVersion/Spits/wp04/GCDC/GCDCCommStackBranch/
src/kernel/test/compat-wireless-2.6.32.16/drivers/net/wireless/ath/main.o
In file included from
/mnt/hc/jan/SoftDev/TeamProjects/Spits/SubVersion/Spits/wp04/GCDC/GCDCCommStackBranch/
src/kernel/test/compat-wireless-2.6.32.16/include/net/compat.h:6,
                 from <command-line>:0:
/mnt/hc/jan/SoftDev/TeamProjects/Spits/SubVersion/Spits/wp04/GCDC/GCDCCommStackBranch/
src/kernel/test/compat-wireless-2.6.32.16/include/linux/compat_autoconf.h:15:2: error:
#error Compat-wireless requirement: CONFIG_WIRELESS_EXT must be enabled in your kernel
make[4]: ***
[/mnt/hc/jan/SoftDev/TeamProjects/Spits/SubVersion/Spits/wp04/GCDC/GCDCCommStackBranch
/src/kernel/test/compat-wireless-2.6.32.16/drivers/net/wireless/ath/main.o] Error 1
make[3]: ***
[/mnt/hc/jan/SoftDev/TeamProjects/Spits/SubVersion/Spits/wp04/GCDC/GCDCCommStackBranch
/src/kernel/test/compat-wireless-2.6.32.16/drivers/net/wireless/ath] Error 2
make[2]: ***
[/mnt/hc/jan/SoftDev/TeamProjects/Spits/SubVersion/Spits/wp04/GCDC/GCDCCommStackBranch
/src/kernel/test/compat-wireless-2.6.32.16/drivers/net/wireless] Error 2
make[1]: ***
[_module_/mnt/hc/jan/SoftDev/TeamProjects/Spits/SubVersion/Spits/wp04/GCDC/GCDCCommSta
ckBranch/src/kernel/test/compat-wireless-2.6.32.16] Error 2
make[1]: Leaving directory `/usr/src/kernels/2.6.34.7-56.fc13.i686'
make: *** [modules] Error 2
```

Note 2

```
patching file ath/ath5k/phy.c
Hunk #1 succeeded at 580 (offset 1 line).
Hunk #2 succeeded at 1097 with fuzz 2.
Hunk #3 succeeded at 1550 (offset 95 lines).
Hunk #4 succeeded at 1856 (offset 95 lines).
Hunk #5 succeeded at 2294 (offset 96 lines).
Hunk #6 succeeded at 3131 (offset 96 lines).
patching file ath/ath5k/attach.c
Hunk #1 succeeded at 117 (offset -9 lines).
patching file ath/ath5k/base.c
Hunk #1 succeeded at 318 (offset 42 lines).
Hunk #2 succeeded at 603 (offset 63 lines).
Hunk #3 succeeded at 947 (offset 74 lines).
Hunk #4 succeeded at 987 (offset 74 lines).
Hunk #5 succeeded at 1003 (offset 74 lines).
Hunk #6 succeeded at 1021 (offset 74 lines).
Hunk #7 succeeded at 2199 (offset 94 lines).
Hunk #8 succeeded at 2287 (offset 100 lines).
Hunk #9 succeeded at 2797 (offset 122 lines).
Hunk #10 succeeded at 2809 (offset 122 lines).
patching file ath/ath5k/base.h
```

```
Hunk #1 succeeded at 110 (offset 7 lines).
Hunk #2 succeeded at 162 (offset 2 lines).
patching file ath/ath5k/ath5k.h
Hunk #1 succeeded at 686 (offset -7 lines).
Hunk #2 succeeded at 696 (offset -7 lines).
Hunk #3 succeeded at 1041 with fuzz 2 (offset 7 lines).
Hunk #4 FAILED at 1310.
1 out of 4 hunks FAILED -- saving rejects to file ath/ath5k/ath5k.h.rej
patching file ath/ath5k/reset.c
Hunk #1 FAILED at 62.
Hunk #3 succeeded at 908 (offset 4 lines).
1 out of 3 hunks FAILED -- saving rejects to file ath/ath5k/reset.c.rej
patching file ath/ath5k/caps.c
patching file ath/ath5k/pcu.c
Hunk #1 FAILED at 185.
Hunk #2 FAILED at 198.
Hunk #3 FAILED at 216.
Hunk #4 FAILED at 229.
4 out of 4 hunks FAILED -- saving rejects to file ath/ath5k/pcu.c.rej
patching file ath/ath5k/qcu.c
Hunk #1 FAILED at 524.
Hunk #2 FAILED at 540.
2 out of 2 hunks FAILED -- saving rejects to file ath/ath5k/qcu.c.rej
```

4.4 Tools

4.4.1 iw

Building iw tool is done through:

```
$ make
```

in the `src/iw-0.9.19` directory. The tool builds without a flaw, but requires `libnl` and its development files (on Fedora, these are supplied through packages `libnl` and `linbnl-devel`, respectively).

Next, install the `iw` tool using

```
# make install
```

Please verify that you have installed the new `iw` over a possible old version from your Linux distribution. Unless too many dependencies are removed with it, it may be wise to remove the existing `iw` package first.

4.4.2 wireless-crda

In order to build `wireless-crda`, my development system (Fedora 13) needed some additional packages, viz. `m2crypto` and `libgcrypt-devel`. On Debian-like systems, the packages are named `python-m2crypto` and `libgcrypt11-dev`, respectively, and should be installed with `apt-get`. The `README` file in the distribution provides additional details. Once these package are installed, building the tools through

```
$ make all_noverify
```

succeeds. The argument `all_noverify` tells the build process to not check for the existence of the `regulatory.bin` file. On Fedora 13, it is located in a different directory (`/lib/crda`) than the location assumed by the build scripts.

Most likely, there is already a `regulatory.bin` file present on your system; the build script expects this file in `/usr/lib/crda`. To check this, simply run

```
$ make verify
```

You must replace the existing file with a suitable `regulatory.bin` file that allows the use of the ITS frequencies in your country *and* in the Netherlands (NL). Be sure to first backup the old regulatory file.

Be careful with the next step, as the wireless CRDA tools are likely to be installed already on your system, and properly hooked into your `udev` subsystem. I strongly suspect that reinstalling them is unnecessary, but I still have to check this with Eric Koenders.

In order to install the wireless CRDA tools, use

```
# make install
```

in the `src/wireless-crda` subdirectory. *Note that this does not install a new* `regulatory.bin` *file (see the next section for that).*

4.4.3 wireless-regdb-2009.11.25

The `wireless-regdb` tool creates a self-signed `regulatory.bin` file allowing the use of ITS frequencies in the Netherlands (NL). Some additional details are present in the `README` file in the distribution. Just type

```
$ make
```

in the `src/wireless-regdb-2009.11.25` subdirectory. In order to properly install (the scripts check for multiple distributions) the file and the corresponding `.pem` key(s), use:

```
# make install
```

As mentioned in Section 4.4.2, the location of the `regulatory.bin` file, as well as that of the keys, is distribution-dependent.

4.5 *CALM-FAST Router (user-space daemon)*

Once the libraries are properly built and installed (see Section 4.2.5), building the CALM FAST router `calmd` is as easy as:

```
$ make
```

in the `src/calmd` directory. It will create a single `calmd` executable that you can copy to your favorite daemon system directory. Subsequently, you should create a suitable `init` script for the `calmd`, and activate it. This, however, is distribution dependent. On Fedora 13, these scripts reside in `/etc/init.d/`.

4.6 *Deployment*

By now you have hopefully succeeded in building and installing all required software for the GCDC Communications Stack. It is now time to deploy the software. After a reboot with a proper Atheros card inserted, you should see –depending on your distribution– the `ath5k`, `mac80211` and `wireless` modules in the output of

```
$ lsmod
```

For further configuration and use of the supplied tools, we refer to [1], Section 2.

5 GCDC Application and Interfacing with the CALM FAST router

5.1 Introduction

This section provides information in addition to [1, Sections 2.5 and 3] on how to use the CALM FAST router `calmd` for GCDC. It assumes that the relevant kernel modules are loaded properly, and that `calmd` is already running. We start with an architectural overwiew of the status of the system at this point, then proceed with skeleton C/C++ and Java implementation of the GCDC (OBU) application.

5.2 Architecture

This section will be provided in a future release.

5.3 Skeleton Implementation in C

This section will be provided in a future release.

5.4 Skeleton Implementation in Java

This section will be provided in a future release.

6 Review and Bug-Fix Process

6.1 Introduction

This section provides some details on the review and bug-fix process after release of the first GCDC Communications Stack.

6.2 Timeline

At the time of writing, both the Interaction Protocol (IP) and the Communications Stack (CS) have been distributed in concept form to the participants of the GCDC. A rough timeline for the next couple of months is as follows:

- In November 2010, we expect quite some feedback on the IP and, especially, on the CS documents and software. We will work hard on bug-fixing and re-releasing.
- Meanwhile, we will work at the precise specification of the test scripts for IP and CS we intend to use at the January 2011 GCDC Technology Workshop.
- At the beginning of December of 2010, we will assess the status of both GCDC technology components. Perhaps we must change (partially) the scope of the workshop from “testing and verification” to just “getting things to work”. At the end of December of 2010, we will finalize the scope and the program of the workshop.
- The first two weeks of January 2011, all of us will be busy making preparations for the workshop.
- After the workshop, we will again assess the status of the IP and CS components, and plan required activities for the GCDC event in May 2011.

6.3 Reporting Problems

The best way to report problems is to post a message on the GCDC web site forum (www.gcdc.net). An alternative is to send an email to the author directly. We kindly request you to not contact Eric Koenders directly, and allow us to filter out problems we can solve ourselves. Note that Eric is not part of the GCDC organization.

6.4 Bug Fixes

Bug fixes of the GCDC Communications Stack will be released regularly, most probably at a once-a-week frequency (if needed).

It is essential that you keep us informed through the GCDC web site forum on problems you encounter with the GCDC Communications Stack and its reference implementation.

6.5 Frequently-Asked Questions

The following table provides answers to frequently-asked questions related to the GCDC Communications Stack.

#	Q+A

1	**Q: Why did you switch to CALM FAST instead of just sticking to 'good-old' IPv4?** **A:** Well, first, rest assured that we were quite reluctant to make that switch, and there were no a priori technical reasons for doing so. More the contrary: despite its higher overhead, GCDC would have worked fine over UDP/IPv4 broadcasts, and these technologies are highly mature and well-known. But, since the first specification of the GCDC protocol stack, the ITS world has moved a lot towards CALM FAST (ISO and, in particular, ETSI/C2C-CC), and these technologies were soon chosen in the SPITS project as the core of the road-side units network at the Helmond Test Site. Admittedly, for very good reasons. Recall that GCDC is now under the hood of SPITS, resulting in an entirely different organization, different stakeholders, many more vehicles and more applications of an increasingly diverse nature. Moreover, IPv4 is generally considered obsolete for future ITS deployments; it is hardly if at all mentioned in the ITS standards (ISO TC204; ETSI TC278). Finally, there was serious doubt whether we would have the resources to fully support a dual-stacked or even triple-stacked road-side network. In the end, we had to choice but to make the switch to CALM FAST.
2	**Q: Why did you wait so long to make the switch and release the software?** **A:** We could only release the software and this document if we were absolutely sure that the protocol stack in the roadside equipment would be stable enough to make that switch. Also, we needed to gain confidence in the software as realized by Eric Koenders. We now have the confidence that the specification is stable, and that the software –despite hurdles still to be expected in getting it functional– will also work for GCDC. We simply had to wait for activities to finish successfully within the SPITS project.
3	**Q: Do you anticipate more switches in the specification?** **A:** No, just improvements, bug fixes, and clarifications. And working code…
4	**Q: Why is it so hard to compile and install the reference implementation of the GCDC Communications Stack?** **A:** In short, because the required modifications to IEEE 802.11 software modules to facilitate IEEE 802.11p have not reached mainstream Linux developers yet. The user community is still relatively small, so it is not on the developers' agenda yet. The result is that we have to patch existing code, and for instance cannot supply DEB or RPM packages covering all major distributions. Another important fact is that until recently, an essential part of the `ath*` code (in fact, of `MadWiFi`) was closed-source due to frequency regulations. This is still a point of concern, though: With the distributed code, anyone can start transmission on the 5.9 GHz ITS frequencies.
5	**Q: Will you send the driver modification for IEEE 802.11p upstream?** **A:** Yes, we will gladly make the offer, but if the code is not accepted, we will not actively pursue it.
6	**Q: Can we run IPv4/IPv6 over the modified ath5k driver?** **A:** Yes, you can, even simultaneously; use `ifconfig` for that.
7	**Q: Can a system equipped with a GCDC Communications Stack communicate with a CVIS router?** **A:** Yes, it can, due to its CALM-FAST support.

7 Final Remarks

In this document we described the GCDC Communications Stack and its reference implementation for the GCDC 2011 event. Starting off with the pioneering work of Eric Koender [1] we hope that this document will contribute to a widely-available, highly-accessible, open-source implementation of IEEE 802.1p and CALM FAST for future ITS application deployment,