Final Report

Project: freeNAV
Status: Final draft
Last revision: December 15, 2004
Contact: Morten Vold <morten.vold@ntnu.no>

Table of contents

1 Introduction

1.1 What initiated this project

Although a tremendous amount of work was put into the tigaNAV [1] project of 2003, we still weren't able to push forward a NAV version stable enough for production use. It was an early realization that any NAV activity in 2004 would need to concentrate on stabilizing the code base so that we might produce a final release version of NAV 3.

1.2 The project objective

Project freeNAV is the 2004 NAV development project. The focus is to make NAV version 3 available for wide deployment as a proven network management solution.

With project tigaNAV development of NAV v3 was finalized. There are still some rough edges, and some parts to complete, but the primary focus in freeNAV is on testing, bug-fixing and documentation.

[1] http://metanav.ntnu.no/tigaNAV/

1.3 Project tasks and hours

freeNAV Project Hours

		Task	Person hours Mo	JM	K	S	HJ	Ma	GA	Total hours
1		Alpha test	80	20	40	20	20	10	60	250
2		Beta test	127		80		20		120	347
3		Development of unfinished features								
	3.1	*Messages*				160				160
	3.2	*Device management*					100			100
	3.3	*Collection system*			523					523
	3.4	*Event engine*			120					120
	3.5	*Network explorer and network load map*			130					130
	3.6	*Statistics*		40						40
	3.7	*User interface*						50		50
	3.8	*Prefix matrix*				30				30
4		License, policy	120							120
5		Build system	150							150
6		Documentation	150	10					0	160
7		New features								
	7.1	*Switch port to room*					50			50
	7.2	*Currently active ports*						0		0
	7.3	*Integrated tftp solution*	0							0
8		General improvements / bug fixing	520	10	220	20	20		2	792
9		Project administration	150	10	90	10	9	10	10	289
		Hours used	1297	90	1203	240	219	70	192	3311
		Hours projected / estimated	1300	90	1300	220	190	140	270	3510

Mo Morten Vold
JM John Magne Bredal
K Kristian Eide
S Sigurd Gartmann
HJ Hans Jørgen Hoel
Ma Magnar Sveen
GA Gro-Anita Vindheim

Cost per hour	330
Total cost	1,092,630
UNINETT share of 50%	546,315

2 Alpha testing

Task no.: 1

People: Everyone

Status: Completed

The alpha testing phase was done in the January/February timeframe. We did, however, discover some serious problems/bugs related to the collection system (gDD) and to the event engine which halted testing of depending parts. We never completed the formal tests as set up in the alpha plan [2],. Instead we decided to move on and introduce the beta testers, while in parallel focusing on solving the problems at hand.

We have at later stages, when relevant, returned to the alpha stage testing themes. For example, an extensive testing of the module monitor was performed in October.

At the time of writing most of the tests have been run, but not as extensible as we would have liked. Our current strategy is to leave testing to the real world and resolve bugs as they are reported.

[2] http://metanav.ntnu.no/tigaNAV/planalfa.txt

3 Beta testing

Task no.: 2

People: Everyone

Status: Ongoing

We shared the first beta releases with UiTø and HiMolde as planned. This was beta1 through beta4 (released in the March - May timeframe). This gave us valuable input for further bugfixing.

In the same period the formal decision on releasing NAV under the GNU GPL was made. On May 11 we made NAV 3 beta5 generally available to "the world". This opened for a lot more beta testers. Many university colleges and universities took the challenge. To our knowledge the following have (or have attempted) a beta installation of NAV 3 at the time of writing:

- UNINETT
- Norwegian University of Science and Technology
- University of Bergen
- University of Oslo
- University of Tromsø
- University College of Gjøvik
- University College of Molde
- University College of Narvik
- University College of Nord-Trøndelag
- University College of Telemark
- University College of Volda
- University College of Ålesund

3.1 Beta releases

At the time of this writing, 8 beta releases have been made, with *beta 4* being the first public GPL release:

March 10	beta 1
March 11	beta 2
March 24	beta 3
May 11	*beta 4*
May 27	beta 5
June 5	beta 6
July 21	beta 7
November 8	beta 8

Starting with beta 7, a changelog [3] was kept and published with each successive release. The changelog summarizes the changes since the previous release, and helps in making the end users informed of what has actually been fixed.

3.2 Testing at NTNU

As of October 18, the ITEA Networks group decided to participate more extensively in the testing of NAV 3, by vowing to maintain the current NAV 3 beta installation in parallel with the current production installation of NAV 2. Since they are actual users of NAV at NTNU, we believed it was pertinent to let them test NAV

[3] http://metanav.ntnu.no/moin.cgi/ChangeLog

3 and discover the bugs that we were unable to see ourselves. This decision was also part of a longer-running strategy to migrate from NAV 2 to NAV 3.

Also, The ITEA test lab was set up with a couple of switch stacks (HP, 3Com) to test the ModuleMon plugin of getDeviceData - this is the component that detects module up/down status. Numerous problems were discovered and fixed, but some problems still remain and are still being worked on at the time of this writing.

4 Development of unfinished features

Task no.: 3

People: John-Magne, Kristian, Sigurd, Hans Jørgen, Magnar, Gro-Anita

Status: Partially completed

4.1 Messages

The work done in tigaNAV on the messages system (then known as eMotd) was incomplete. The system had many bugs and omissions, and much work has been put into stabilizing the system. The maintenance component did not work at all, but has now been implemented as a cron job. This job monitors the maintenance schedule and posts the appropriate maintenance on/off events to the eventq accordingly.

The web interface has also been made more user friendly, in effect narrowing down the functionality. The maintenance functionality has been parameterized, so that other web tools (such as the IP Device center) can offer direct links to place specific equipment on maintenance through the messages tool.

In addition to this, an automated RSS feed of active messages is now available (as originally planned in tigaNAV).

4.2 Device Management

The idea behind the Device Management tool is to provide an interface to the device inventory and to manage the event history of any physical device registered in the inventory (each device being identified by its serial number). tigaNAV left off some rough edges that freeNAV has improved:

- Device history now shows the complete list of events that are related to the selected device.
- A lot of details regarding input of serial numbers, dates etc. have been changes. More consistent and logical "form flow".
- It is now possible to register RMAs.
- The underlying set of events sent through the event engine by the Device Management tool has been appropriately restructured.
- The user interface is improved (with menu tabs, and a consistent NAV look and feel).

4.3 The collection system

The collection system is at the heart of NAV. Any problems experienced by this system will have a great effect upon other parts of NAV. Therefore, a great deal of attention was given to this system during freeNAV. Not only were there many yet unimplemented features, but also quite a few bugs or other problems to be solved. 550 work hours were estimated for this task, 570 were spent.

If we were to go into every detail of what changed in the collection system in freeNAV, this section would be immense. Instead, we try to summarize the changes in the following list:

- Work to enable gDD to run 24/7: Automatically detect when devices are added/removed from the database via editDB.
- Detect when the type of existing devices change and re-collect the netbox data when it happens.
- Support for sending (when network changes) / receiving events (respond to requests, e.g. collect data from a device immediately) (events documented in javadoc).
- DNSCheck plugin to verify that DNS matches device sysname.
- Device tracking via serial - we can follow a device as it moves around the network by collecting serial number of all devices (chassis / modules).

- Collect device uptime and detect restarts (a coldStart event is sent when a restart is detected) based on uptime changes.
- Complete re-write of the Gwport-plugin to correctly collect VLAN information without relying on NTNU-specific conventions (netident).
- Support for collecting static route information.
- General support for HP stacks, which are accessed quite differently from Cisco and 3Com stacks.
- Interpret submodule info correctly for modules which use this.
- Support nettype auto-determination of elink.
- ModuleMon support for 3Com PS40, 3Com SuperStack and HP (including stacking).
- New OIDs were needed to fully support Cisco 45xx (cL3Serial, cL3Model, cL3HwVer, cL3SwVer OIDs).
- Plugin CiscoModule collects info about modules found in Cisco devices using oidkeys phys*, cCard*, cL3* and catModule*. We now only create modules for actual physical modules; no more "virtual" modules. This gives a much more accurate view of devices.
- OID tester detects whether an IP device has a separate chassis or is just a collection of normally independent devices.
- Complete rewrite of Database package to support multiple threads in a general way.
- Support UNINETT router interface descriptions.
- Detect which device in a stack is assigned the IP and use this when doing module monitoring.
- Detect duplicate switch ports (e.g. if modules are moved around, causing ifindexes to be renumbered/recycled).
- Collect CDP names we do not recognize (not found in netbox table). These are used to set the correct netident for elinks (external links).
- While no specific support was required (the CAM collector has been generalized to be completely type-independent) it is worth mentioning that WLAN APs are fully supported; we can trace clients connected and which access point they have been connected to at any point in time.

4.3.1 New OIDs

Added OIDs:

- sysUpTime and dnscheck OIDs
- hpSerial oid
- support for Cisco 45xx (cL3Serial, cL3Model, cL3HwVer, cL3SwVer OIDs)
- ipNetToMediaPhysAddress OID for mapping ifindex and IP to physical MAC
- the catModule* OIDs
- cChassisId
- the phys* OIDs
- the vtpVlanState OID
- ipRoute* OIDs
- physParentRelPos OID
- hpStackStatsMemberOperStatus

In particular, ENTITY-MIB support has been added, consisting alone of 12 OIDs. These MIBs give a comprehensive description of the various sub-components of the device in question; it is hierarchical, but as of now we only collect data for the first level below the chassis (most devices only have one level, however). A drawback to using these OIDs is that they don't reflect changes in the hardware configuration until the next reboot, and, as already mentioned, much effort has gone into handling this case well.

In total about 50 new OIDs have been added in 2004, giving a total of 154 OIDs. The number of device types supported has also been greatly increased as the support of a new OID works for all types supporting said OID, not just those for which it has been programmed.

4.3.2 Problems

Here we also try to summarize some of the problems we experienced in the development of the collection system:

- Apparently, many devices report incorrect serial numbers (e.g. *'0'*, *'1234'* or other useless values).
- It is a complex problem to handle module data collection correctly in all cases and in a general way, as different versions of Cisco software are often not consistent. This is especially troublesome when the same data can be collected from different sources, where often only one source gives "good" data, but this sources varies depending on device type and software version. Often heuristics have to be applied to make this determination, and much time has been spent with trial and error to find good heuristics.

  This work pays off when new device types are introduced and NAV automatically detects them correctly without further work on the part of the developers (this is also important as the developers only have access to a subset of available hardware to test on).
- Some devices support hotplugging of modules. However, not all SNMP data is updated to reflect changes made and will only be correct after a reboot. Often this is not desireable and we need to handle this case correctly even in the face of incorrect data.
- The device uptime reported by devices is often not very accurate as the clock can "drift" as compared to an accurate timer, often by several seconds each hour. This has to be taken into account if we are to avoid false device restart alerts, and also to keep the database up-to-date without doing constant updates.

  The uptime counter is also only 32 bits and will wrap after 497 days, and this must be taken into account. Finally, some devices report a constant upticks value and this must be detected and the value ignored.
- For some versions of Cisco software collection of static routes create a high load on the SNMP engine which sets off alarms. To avoid this we collect the data slowly, with a delay between requests.
- At the start of the NAV 3 development hardware was significantly slower than today and the database software less mature. Realizing this, a design decision was made to make getDeviceData read the database upon startup and keep the data in-memory to reduce database accesses and thus improve performance. Unfortunately, this created a need for keeping the in-memory cache and the database contents in sync and much effort was spent toward this goal. However, even the slightest error, for example a database updating failing, would ruin the synchronization, and this has been a frail point for some time.

  Today the reasoning behind the design decisions made is mostly no longer valid, as hardware has become more powerful and database software has matured. We have thus decided to alter getDeviceData to no longer keep an in-memory cache and instead work directly with the database. This is a significantly more robust solution and should increase the stability of the data collection over time.
- All throughout the project we have needed to make adjustments to the database tables. Many constraints have been added to ensure the data is consistent at all times and fields have been added to accommodate "new" features. The OID database was changed to store supported OIDs per netbox instead of per type as different firmware versions within the same type can change the supported OIDs. A new table was created to store VTP (Vlan Trunking Protocol) VLANs per netbox.
- getDeviceData used to delete a netbox whenever its serial number or type changed; this would signify that the netbox had been replaced with a different netbox and the data should be deleted to ensure no stale information about the old netbox was left. This has turned out to have a number of disadvantages, however, most importantly that deleting a netbox is a slow operation since a large number of other tables reference the netbox table, and also all systems in NAV have to be prepared that the data it is working on can disappear at any moment. This has now been changed, and only the modules of a netbox are deleted, which gives faster and more robust operation while still achieving the goal of not leaving stale data behind.
- The current OID database lacks a mechanism for administrating the collection frequency of OIDs. Since complex dependencies can exist between OIDs (if one is collected, several more might have to be collected at the same time as well) it is non-trivial to change the frequency. We have proposed a solution to this problem (described in detail on *nav-dev*) by the addition of another table which encodes the dependencies into functional groups and allows for changing the frequency of each group from a web interface. Implementation of this proposal should be planned for 2005.

4.3.3 Further work

A few recommendations on what should be worked on after project freeNAV:

- NAV 2 collects information about installed memory and software versions in a number of devices. NAV 3 does not yet support this for all devices supported in NAV 2.
- Currently the ARP and CAM collectors run independently of the main collection system (getDeviceData) and are scheduled to run at regular intervals by the cron daemon. This has the drawback of slowing down collection speed to the slowest device as collection has to complete before a new collection run can be started. By converting the CAM and ARP collectors to getDeviceData plugins, collection can be scheduled independently, and it is also possible to increase or decrease the collection frequency for a particular type of device depending on your needs.
- Currently NAV 3 uses Cricket for collection of traffic load data. Unfortunately Cricket does collection in a serial manner, which limits the number of devices from which data can be collected within the given time limit. By doing traffic data collection in a getDeviceData plugin this process can be parallelized and only be limited by the hardware on which NAV runs.

4.4 The event engine

This was done with the event engine:

- Send warning messages before declaring boxes down.
- Support for ModuleMon events.
- Handle events deviceActive, deviceState, deviceNotice.
- Support arbitrary (varchar) subid to enable the event system to be used from a greater number of sources (UNINETT systems).
- Improve alertMsg.conf parsing (and enable use of _ in variable names).
- Make more database fields available as variables for *alertMsg.conf*.
- Add alerttype field to alertq. This makes it possible to also filter on alerttype (e.g. boxDown, boxShadow) in the alert profile.

4.5 Network explorer and Traffic map

4.5.1 Traffic map

This was done with the Traffic map (aka. vlanPlot) in freeNAV:

- Integrated with the new Python authentication system, no *vP* signon.
- Scale icons to fit better on screen.
- Better support for groups or containers. A container can now be displayed as a single icon and thus enabling *vP* to scale to hundreds of routers in a nice way.
- Use a queue for icon drawing to work correctly on all JVMs.
- Links to IP Device center and Network explorer in context menu.
- Various small bugfixes to improve layout and displayed popups etc.
- Print error message if data cannot be fetched from server, instead of the previously blank screen.

4.5.2 Network explorer

- Improved web layout to make the display cleaner and simpler to understand.
- Add MAC display and search (data from camlogger). This makes it easy to visualize exactly where in the network a given MAC or IP is located.
- Links to IP Device center.

4.5.3 Further work

A few recommendations on what should be worked on after project freeNAV:

- The Traffic map has not yet been completely ported to the NAV 3 database layout. The lowest level, switch traversal, remains to be ported.
- The program providing the Traffic map applet with traffic load data has not yet been ported to NAV 3. A new solution is being prototyped and tested at the time of writing.
- The Traffic map could provide status information by color-coding shown devices if it, or any devices below, is/are unreachable.

4.6 Statistics

Server statistics are now supported; makecricketconfig will generate the appropriate Cricket configuration for collecting SNMP data from servers, based on whether NAV deems the server to be in the Windows or Unix category.

Unfortunately, John-Magne has had very little extra time to devote to NAV development this year, and so sorted statistics remain unimplemented in NAV 3.

4.7 User interface

Magnar has done much with the user interface, with respect to:

- Tool icons
- Front page structure
- Machine tracker design
- IP Device center design

Unfortunately, the user administration panel has not been prioritized, meaning only some of its pages have been redesigned. The user administration panel is still usable, though.

4.8 Prefix matrix

Completed as planned.

5 License, policy

Task no.: 4

People: Morten

Status: Completed

In March, various licensing models for NAV were evaluated, resulting in a recommendation [4] for using the GNU GPL [5] license. This recommendation was presented to the IT leaders at NTNU and Uninett on April 2. NTNU concurred on April 13, Uninett on April 20.

After these decisions, the NAV source code was prepared for a GPL release in compliance with the guidelines of GNU. In short, this work mostly consisted of:

- Adding a GPL header to all source files.
- Include licensing/copyright information in the source code tree.

On May 11, *NAV 3.0 beta 4* was made available for public download under the GPL license, although the announcement was only posted on the *nav-users* mailing list. A NAV project page [6] was reserved on Sourceforge [7], which will later be used to inform of NAV to a broader public.

[4] nav-lisens-okonomi.pdf

[5] http://www.gnu.org/licenses/licenses.html#GPL

6 Build system

Task no.: 5

People: Morten

Status: Completed

Most of the new build system was completed for the *Alpha 3* release made on February 27, although subsequent fixes were made up until the first public GPL release of *Beta 4* on May 11. Since its completion, the build system has been continually maintained in sync with changes in the code. Most of the still active developers have been educated in maintaining their own Makefiles.

As a result of the new build system, the code repository has been completely restructured from the old NAV 2 layout, and all hardcoded paths have been removed in favor of build configurable variables. The build system has been proven to simplify the packaging of new NAV releases, as we have easily provided Redhat RPM packages with each successive alpha/beta release. Example *.spec*-files are provided within the source code repository.

On July 6 we were finally able to claim compliance with Apache 2 and provide an example Apache 2 configuration for using with NAV 3. Using Apache 2 simplifies the installation process of NAV 3 further, as it removes the requirement of installing a second Python interpreter having threading support disabled (this was a requirement when using Apache 1.3).

Uninett is already benefiting from the new build system in their ongoing work to create a Debian packaged release of NAV 3 for their Samson 3 platform, in which the configurable build has provided a simple means to relocate NAV into a Debian-specific directory structure. Also, the build system has made it possible for UiTø to submit NAV to the FreeBSD ports collection [8].

7 Documentation

Task no.: 6

People: Morten, John-Magne, Kristian, Gro-Anita

Status: Partially complete

Again, we've had less time than we hoped for writing user documentation, in between all the work on the other tasks. Still, we've tried to help as best we can on the *nav-users* mailing list, guiding new users having problems with their installation. The user documentation that *has* been produced consists, among other things, of:

Installation log We've provided a log of NTNUs installation procedure on Red Hat 9. The procedure consists of installing and configuring all the prerequisites, whether it be through RPM and apt-get or through manual compilation of software. The log was also greatly revised/simplified after Apache 2 compliance was achieved. It has served as a basis for installation on other platforms as well. A similar log for a SuSE Linux Enterprise Server 9 will be published shortly.

Getting started guide The getting-started guide goes through setting up your NAV installation after the prerequisites and NAV itself has been installed to the system.

We also hope the new MetaNAV wiki will help inspire even NAV users to contribute documentation tidbits.

When it comes to developer documentation, it might not be easily accessible, but it exists. The Java based systems are fairly well documented through Javadoc [9], and also Python allows for nice integration of developer documentation within the source code.

An online context help system included in the web interface was discussed on some of the early meetings. However, this was never implemented, due to limited resources. It may be a good idea for future improvements.

[6] http://www.sourceforge.net/projects/nav

[7] http://www.sourceforge.net/

[8] http://www.freebsd.org/cgi/cvsweb.cgi/ports/net-mgmt/nav/

[9] http://metanav.ntnu.no/javadoc/

7.1 A new MetaNAV website

Many hours were spent putting content into and evaluating Plone as a content management system for a new MetaNAV web site. Unfortunately, during this work, ITEA dropped their plans to offer Plone services at NTNU and shut down the Plone system. As a result, new effort had to be made to find a simple system to publish NAV related information.

This has caused a delay in launching a new MetaNAV site. Various Wiki clones have been evaluated as replacements for the MetaNAV site, and a MoinMoin [10] based MetaNAV was launched shortly before the finishing of this report. We hope that a MetaNAV wiki will inspire other NAV users to contribute information and lead to more NAV collaboration between Uninett members (and other NAV users). See [11] for the new web site.

8 New features

Task no.: 7

People: Morten, Hans Jørgen, Magnar

Status: Partially complete

8.1 Switch port to room

The edit database tool now supports entry of cabling and patch information into the NAV database. We now have the means to track an IP/MAC address all the way down to a physical office room. An interface to track down to this level is not yet implemented in the Machine Tracker, but is considered a minor task that can be postponed.

We would like to mention at this point a project at one of NTNU's institutes, concering tracking of IP phones. This project already receives this type of tracking data from a proprietary script installed on our NAV 3 server.

8.2 Currently active ports

Postponed in favor of other, more urgent matters. The needed functionality in getDeviceData has actually been implemented, but the IP Device center interface doesn't support it yet. The interface will need to post an event to the event queue, on which gDD will react and collect port information on-the-fly. A reply event will be posted by gDD when it finishes, which means the web interface needs to poll the event queue in an asynchronous manner to know when the results may be retrieved from the database.

8.3 Integrated tftp solution

Also postponed, due to a lack of resources. The solution already exists as a proprietary extension to NAV 2 at NTNU, so porting to NAV 3 should be trivial when time allows.

9 General improvements / bug fixing

Task no.: 8

People: Morten, Kristian, Sigurd, Hans Jørgen

Status: Always an ongoing task

We have discovered and fixed a great amount of bugs. Unfortunately we have have discovered several problems that seem to be a result of insufficiencies in the design & analysis phases. Future NAV development could still use much improvement in this area.

Many of the problems were encountered in the collection and event systems, and have been described in these chapters, although many of the hours have been assigned to this task.

Some of the smaller improvements include:

[10] http://moinmoin.wikiwikiweb.de/

[11] http://metanav.ntnu.no/moin.cgi

- Syslog Analyzer was ported/rewritten from NAV 2 (then known as *navlog*).
- As we no longer store supported SNMP OIDs per. device type, but instead per. IP device, Cricket's hierarchical configuration properties has become useless to us. Makecricketconfig had to be rewritten to produce a flat Cricket configuration file.
- In March, we added the ability to externally authenticate users of the web interface through LDAP. This is used today to authenticate NAV 3 users at NTNU using NTNU's central user database.
- Added *dump.py*, a script to dump the 'seed' contents of the NAV database into a format suitable for bulk import in the Edit Database web interface.
- Added *navclean.py*, a utility script that can be used to delete old ARP and CAM records to save disk space in the database. Typically, this script would be run as a cron job by the NAV administrator.
- The threshhold monitor has received several fixes and now runs as a standard cron job.
- The service start/stop system has been revised. A Python API is now available for starting/stopping and querying the status of NAV daemons and cron jobs.

9.1 Database package rewrite

All Java components of NAV 3 use a shared library package for easy access to the database. This extra layer means improvements to the database access layer will automatically benefit all components using the database. The new data collection component (getDeviceData) in NAV 3 is heavily threaded to make it scalable to collect data from hundreds, or even thousands of network devices, while the old Database package was essentially single-threaded. This turned out to cause performance problems, and made necessary a rewrite of the package.

The new package is fully threaded. Its API has been further generalized, and while still retaining backwards compatibility with the old API, it allows for simultaineously accessing several databases, and hides the complexity of using multiple threads. Transactions are also fully supported. The new database package works well and provides a scalable, high-performance database abstraction for all Java-based NAV 3 components.

9.2 Ghost in the Machine Tracker

During our test runs in October, we discovered a design fault in the Machine Tracker. At the time the Machine Tracker component was developed (during the tigaNAV [1] project), we still did not have reliable tracking data collection in NAV 3, rendering us unable to do useful testing of the Machine Tracker.

We found the Machine Tracker to be using an SQL statement that was intended to automatically correlate CAM and ARP records, but instead it would cross join ARP records with any CAM record having a matching MAC address. The symptom was seemingly arbitrary results in many search cases.

We spent quite a bit of time trying to design a new SQL statement which would do what was originally intended. A new SQL statement was implemented, and at first it seemed to give more consistent results. While examining the problem more closely, though, we realized how extremely complex a problem it is to correlate these records automatically and reliably for every conceiveable scenario. Instead of spending even more time on this problem, we recommend that we go back to the "old way" of manually correlating the ARP and CAM records by returning to separate ARP and CAM searches (as it was done in NAV 2). This work should commence shortly after the conclusion of freeNAV.

10 Other improvements

Beyond what was part of the project plan, several other, mentionable, improvements have been made - not only by NTNU, but also Uninett. These include:

- Uninett has created MailIn [12] - an add-on that enables the NAV event system to receive external events through e-mail.
- NAV provides a patch to Cricket which adds functionality such as graph zooming in the user interface, and fixing a locking problem when updating RRD files. The patch was originally applicable to Cricket 1.0.3, but has been fixed so it works for 1.0.5. Morten Werner Olsen of Uninett has submitted the locking part of this patch to the upstream Cricket authors, who seem interested in incorporating it into the Cricket codebase.

11 Project administration

Task no.: 9

People: Everyone

Status: Completed

Regular project meetings were held during the first half of 2004. After this time, most of our part-time employed developers were no longer working for us, Kristian Eide being the one exception. Sine then, most project coordination has taken place using phones, e-mail and instant messaging.

The *nav-dev* mailing list has, as always, been our arena of technical discussions on NAV development. Mailing list traffic has exceeded 1600 postings during freeNAV. Over 60% of the postings originated from either of Morten, Kristian and Vidar (Uninett).

12 Summary and conclusion

12.1 The path ahead

A few issues still need to be ironed out in the near future. Some features still remain to be implemented or fixed before we can push forward a release candidate of NAV 3.0. Some features may be postponed til post-3.0, even some that already exist in NAV 2 but remain unimplemented from NAV 3.

Before we can push forward a release candidate of NAV 3.0, we need to do the following:

- Machine Tracker needs to be reverted to the old-style, separate ARP and CAM searches, as described in Ghost in the Machine Tracker.
- Color coded traffic load information needs to be implemented/ported into the Traffic map.
- Drill-down to switch layer in the Traffic map (as in NAV 2)
- Collection of data related to memory and software version.
- Make sure that we receive accurate alerts (according to active alert profile) when IP devices and modules go down and up. This means more testing of the basic detection of box/module state (up/down/shadow) and, accordingly, the correct passing of events and alerts.

Several of these tasks are being worked on as this is being written.

Still, the most important goal of the 3.0 release is stability, with a known basic feature set. At this point, we can forego certain features that will only serve to delay the release further. This also means that some of the features seen in NAV 2, and some of the planned features from freeNAV will need to be postponed, but implemented ASAP, post-3.0:

- A sorted statistics query page (as in NAV 2 NTNU local addition)
- A refresh button in the IP Device center, to trigger force immediate retrieval of port status.
- Similarly, a view in the IP Device center to display which switch ports that have had any activity during the last N days.

For NTNU's own sake, we will need to spend resources in 2005 on porting our local NAV additions from NAV 2 to NAV 3 as we migrate to the new version. Some of these local additions are also planned features for inclusion in the official NAV code base, so this will be our primary focus after the 3.0 release.

The *ITEA Networks group* has taken to evaluating NAV 3 in its current beta state, and have committed themselves to keep it as updated as they do the current production installation of NAV 2. We believe this is the best way to discover and further reduce the number of bugs before the release of 3.0.

Probably, the most important activity that should be taken up during 2005 can be summarized in two words: *Knowledge transfer*. NAV suffers from the fact that each developer is mostly the only one to know his/her own code. If developers leave the project (either because of unexpectedly getting hit by the bus, or just not being interested in renewing their part-time contracts), no one is able to maintain their code without considerable resource usage, at great cost to the further development of NAV. A big example of this would be the entire collection and event systems. They are all Java programs, and were all written by one developer. If transfer of

[12] http://drift.uninett.no/~andreas/mailin/

knowledge does not take place, we will have a huge problem whenever Kristian decides he has better things to do than to work for us.

For more information about planned features and/or ideas for new features, visit the new MetaNAV [13] pages.

12.2 Conclusion

We have followed the project plan to the best of our efforts. Many more hours than at first anticipated had to be devoted to tracking down and eliminating bugs and design problems, some of which should have been discovered or thought of as far back as project NAVMore in 2002. This goes to show that future NAV development projects still need to improve on the analysis and design phases, of which too little emphasis has been given in previous projects.

Also, it has become apparent to us that the continued use of part-time employees for a software project of this magnitude mostly serves to reduce the continuity of the software development. Our part-time developers have been competent students, producing high quality work and contributing great ideas. The problems with hiring part-time employees are, however:

- Each developer has been given to much freedom in his/her choice of software development tools - with little regard to integration with existing or future subsystems of NAV.
- Each developer has had a strong ownership to his/her own code, and as their part-time employment with us ends, no one else knows the code they've produced. This creates significant problems in maintaining the code in the future.
- When urgent problems are discovered, the response time of part-time employed students varies greatly, from a matter of minutes to a matter of weeks. Sometimes it is a matter of being occupied with studies, other times it may just be a lack of responsibility towards an employer one has no foreseeable future of working full-time for.

During project freeNAV, we've lost three part-time developers from our crew: Sigurd, Hans Jørgen and Magnar. Although their departures were planned, we've noticed the stress of being only two active developers. While John-Magne and Gro-Anita have been preoccupied with other assignments this year, Kristian and Morten have been the only active developers for almost all of the second half of 2004, as no other resources have been available to us. Fortunately, Vidar (Uninett) has been with us and contributed a substantial administrative and inspirational effort, even after leaving NTNU to work for Uninett.

Despite the problems we've experienced during freeNAV, we believe we have come a long way towards releasing a final 3.0 version of NAV. Although it is not complete at the time of this writing, we believe we will have a release candidate in the January-February time frame, provided that we have at least the same amount of resources available to us. Having made available beta releases and development versions to the world, we've also received a lot of positive feedback from other Uninett members, and we feel it has helped to alleviate the users' demand for an imminent 3.0 release.

[13] http://metanav.ntnu.no/NAVPlanned