Praxissemesterbericht

Prototypische Entwicklung einer VoIP–Telefonanlage unter Einsatz von Linux und eingebetteten Systemen

Phil Sutter

SS06/07

Inhaltsverzeichnis

1. Einleitung

Telefonanlagen sind in Unternehmen meist in Form dedizierter Hardware zu finden, deren Administration einer gesonderten Ausbildung bedarf. Innovation ist hier selten zu finden. Doch gerade in der Telekommunikationsindustrie sollte Innovation auf der Tagesordnung stehen, vermischt sie sich doch zunehmend mit der EDV, wobei nicht nur am Beispiel von Voice over IP positive Resultate zu beobachten sind.

Zwar wird Voice over IP gerne als der Nachfolger herkömmlicher Telefonie bezeichnet, jedoch gibt es Argumente, die eher auf einen parallelen Einsatz beider Technologien schließen lassen. Beispielsweise sind IP–Netze nicht echtzeitfähig und somit potentiell zu unzuverlässig, um als gute Grundlage für die Telefonie zu gelten. Dennoch wird es als günstige und flexible Alternative genutzt.

Unternehmensseitig liegen die Anforderungen an die eigenen Telefoniedienste etwas höher. Erreichbarkeit auf mehreren Nummern, interne Telefonate und Konferenzschaltungen sind nur einige Beispiele für solche Anforderungen, die selbst manche proprietäre Telefonanlage nicht leisten kann oder sie auf teils umständlichen Wegen erreicht (z.B. interne Gespräche über den externen S0–Bus).

Am Beispiel der Entwicklung von softwarebasierten Telefonanlagen ist zu beobachten, dass eindeutig Bedarf nach Alternativen herrscht. Die vollständige Realisierung der Anwendungslogik in Software schafft hohe Flexibilität und Ausbaufähigkeit.

Da herkömmliche Telefonanlagen bereits ab Design auf Langlebigkeit ausgelegt sind, muss die Hardware für eine Software–Telefonanlage diesen Anforderungen ebenso gerecht werden. Aufgrund des Fehlens rotierender Teile bieten sich eingebettete Systeme an, die aufgrund geringer Preise und Grössen auch das Vorhalten eines Zweitgerätes ermöglichen. Zudem können sie problemlos auf dem Postweg transportiert und ohne Fachkraft vom Kunden montiert werden.

Im Rahmen meines zweiten Praxissemesters erhielt ich bei der Firma AurISP IT Consulting die Möglichkeit, eine solche Telefonanlage zu entwickeln und exemplarisch für meinen Arbeitgeber einzurichten. AurISP IT Consulting ist eine junge Firma aus Bonn, die IT–Lösungen auf Linux–Basis für kleine und mittelständische Unternehmen konzipiert und implementiert. Zudem verfügt sie über großes Know–How im Embedded–Linux–Sektor.

In diesem Bericht wird zunächst auf die Anforderungen an das spätere Produkt eingegangen. Anschliessend werden die technischen Voraussetzungen erläutert, wobei Soft- und Hardware jeweils ein eigenes Kapitel gewidmet wird. Nachdem im folgenden Kapitel auf die Integration der Soft- und Hardware eingegangen wurde, wenden sich die nächsten zwei Kapitel ganz der Telefonie zu. Abschließend werden Testergebnisse präsentiert und gewichtet.

2. Anforderungen

Die Anlage sollte genügend Leistungsreserven besitzen, um bis zu zehn gleichzeitige Gespräche zu verwalten. Um diesen Wert einschätzen zu können, muss bekannt sein, wie hoch die relative Dichte der telefonierenden Personen in einem bestimmten Unternehmen ist. Dann lässt sich berechnen, ob die Telefonanlage unter- oder überdimensioniert ist. Da die Telefoniererdichte allerdings stark vom Unternehmen abhängt, existieren hierzu sehr verschiedene Meinungen. Jedoch ergibt sich unter Annahme eines Maximums von 50% und eines Durchschnitts von 20% eine Unternehmensgröße von 20 bis 50 Personen – je nach Beschäftigungsfeld. Wobei anzumerken ist, dass Branchen wie Callcenter, in denen die Dichte gerne auf 80% und mehr geschätzt werden kann, hier keine Beachtung finden. Diese Unternehmen haben ohnehin Bedarf nach mehr Ressourcen als eine Telefonanlage nach hier gezeigten Maßstäben bieten könnte.

Der Fokus auf kleine und mittelständische Unternehmen setzt natürlich auch preisliche Grenzen. Da gerade der Preis meist ein Argument gegen die herkömmliche Anlage ist, sollten die Anschaffungskosten 400 Euro nicht überschreiten.

Unternehmensseitig liegen die Anforderungen an die eigenen Telefoniedienste recht hoch. Neben Erreichbarkeit auf mehreren Nummern, internen Telefonaten und Konferenzschaltungen, Weiterleitungen und Anrufbeantworterfunktionalitäten sollten beispielsweise auch das Heranholen von Anrufen auf einer anderen Durchwahl oder die Signalisierung belegter Endgeräte am eigenen Telefon zum Repertoire der Anlage gehören. Die große Herausforderung liegt für den Dienstleister vorallem darin, die Telefonanlage in die bestehenden EDV–Systeme des Unternehmens zu integrieren und auf dessen Wünsche hin anzupassen.

Telefonanlagen müssen ausfallsicher und wartbar sein. Eine gewisse Ausfallsicherheit ist bei einem eingebetteten System a priori vorhanden und die Möglichkeit der Lagerung eines Zweitgeräts vor Ort erhöht diese weiter. Wartbarkeit wird bei einer herkömmlichen Telefonanlage durch austauschbare Komponenten erreicht, eingebettete Systeme werden einfach komplett ausgetauscht.

Die Unterstützung von ISDN ist unverzichtbar. Abgesehen von der höheren Zuverlässigkeit gegenüber VoIP sind beispielsweise Faxdienste noch immer gefragt. Manche ISDN–Karten erlauben sogar die Anbindung von ISDN–Endgeräten wie beispielsweise Telefone oder eine interne Telefonanlage.

Eine weitere, wünschenswerte Fähigkeit ist die Möglichkeit zur Steuerung der Telefonanlage von einem Endgerät aus. Dies kann zum Beispiel genutzt werden, um Rufumleitungen einzurichten oder den Anrufbeantworter–Text zu verändern. Hierzu gibt es an den meisten neueren Telefonen Zusatztasten, welche mit einer Status–LED ausgestattet sind. Das Ansprechen dieser LEDs von der Telefonanlage aus rundet die Interaktion zwischen Anlage und Endgerät ab.

3. Betriebssystem

Das FreeWRT–Projekt ist eine Linux–Distribution für eingebettete Systeme, vornehmlich Router aus dem Low–End–Bereich. Der Name lehnt sich am ersten unterstützten System, dem Linksys WRT54G, an. Da FreeWRT komplett vom End–Nutzer übersetzt wird, bietet es hervorragende Möglichkeiten, um eine Linux–Distribution nach eigenen Maßstäben zum dedizierten Einsatz zusammenzustellen.

Die FreeWRT–Quellen sind nicht nur frei - da GPL, sondern auch gratis. Der Quellbaum beinhaltet ein Framework aus Makefiles und Shell–Skripten, welche eine Konfigurationsoberfläche im Stil der Linux–Kernelkonfiguration bieten. Desweiteren automatisieren sie das Herunterladen und Übersetzen aller benötigter Programme. Dies umfasst bis auf wenige Ausnahmen sowohl auf dem Hostsystem benötigte Werkzeuge, als auch Kernel und Programme für das Zielsystem. Das erzeugte System wird schließlich in einem Image[1] zusammengefasst.

Die Verwendung eines Images begründet sich darin, dass das Schreiben in den ROM–Speicher eines eingebetteten Systems als kritischer Vorgang zu sehen ist. Der Zustand der meisten Systeme nach einem fehlerhaften Schreibvorgang ist undefiniert und für einen Laien oft nicht kontrollierbar. Daher werden Daten in einem Image erst vorbereitet und später in einer atomaren Operation geschrieben. Dieser Vorgang wird allgemein als Flashen bezeichnet. Das FreeWRT ADK beinhaltet Skripte, die den Anwender bei der Installation des Images unterstützen.

Das FreeWRT Appliance Development Kit dient zur Übersetzung von Programmen für verschiedenste Architekturen, wenngleich bisher lediglich Geräte mit MIPS–Architektur unterstützt werden. Die gute Skalierbarkeit zeigt sich darin, dass ohne grössere Probleme Patches[2], Konfigurationen und Makefiles für neue Zielsysteme integriert werden können.

Während meiner Arbeit am Projekt wurde das FreeWRT Release 1.0 veröffentlicht. Über die Neuigkeit wurde im Heise–Open–Newsticker[3] und Linux Magazin[4] berichtet. Das Ergebnis meiner Mitarbeit wird zum grössten Teil in das Release 1.1 einfliessen.

Als FreeWRT–Developer konnte ich weitere Verbesserungen des Appliance Development Kits realisieren:

- Bisher wurde pro Kernel–Version eine Datei vorgehalten, welche die notwendigen

[1] Eine Datei, welche die Daten eines bestimmten Mediums eins–zu–eins enthält, also ein eindeutiges Abbild.

[2] Ein Patch bezeichnet ein anwendbares Delta zweier Dateien oder Verzeichnisse. Gewöhnlich wird ein Patch zwischen der alten und der neuen Version einer Datei gemacht und verteilt, sodass die alte Datei leicht auf den neuen Stand gebracht werden kann. Patches werden mit den Unix–Kommandos `diff` und `patch` erstellt und angewendet.

[3] http://www.heise.de/open/news/meldung/print/83858

[4] http://www.linux-magazin.de/news/router_flash_mit_freewrt

Informationen beinhaltet, um die im Kernel gebauten Module in Packages zusammenzufassen. Aufgrund der großen Gemeinsamkeiten wurden diese von mir zu einer einzelnen zusammengefasst.

- Zur Optimierung der Makefiles für die einzelnen Pakete auf Größe und Komplexität wurden die internen Makefiles zur Paketgenerierung umgestaltet. An der folgenden Anpassung der über 300 Paketmakefiles war ich maßgeblich beteiligt.

- Da die Informationen, die für jedes zu übersetzende Paket vorgehalten werden, verschiedenen Subsystemen zur Verfügung stehen müssen, liegen diese oft redundant vor. Um zu ermöglichen, dass alle Informationen pro Paket in einer einzelnen Datei angegeben werden können, wurde von mir ein Parser und Interpreter geschrieben, welcher diese Dateien einliest und die Informationen den einzelnen Subsystemen in nativer Form zur Verfügung stellt.

Gerade letzteres Projekt ist noch nicht beendet und wird in der Zukunft weitergeführt werden.

4. RouterBoard

Das RouterBoard 532 wird vom Hersteller Mikrotik[1] als robuste Hardwareplattform für das Routing hoher Bandbreiten via LAN und WLAN angepriesen. Es besitzt drei Netzwerkkarten, zwei MiniPCI–Slots, einen CompactFlash–Slot und einen seriellen Port. Bei der CPU handelt es sich um einen MIPS Prozessor von IDT. Auf der Platine sind 32MB Arbeitsspeicher, sowie 128MB NAND–Flash aufgelötet. Die CPU lässt sich stufenweise hochtakten, von minimal 200MHz über 266MHz und 300MHz bis hin zu 400MHz.

Besonders reizvoll an der Hardware ist, dass die Spezifikationen offen sind. Auf der IDT–Homepage können Referenzimplementierungen der CPU für Linux Kernel 2.6 heruntergeladen werden, Mikrotik bietet Interessierten ein Debian–Image als Referenzbetriebssystem und die vorgenommenen Kernel–Patches auf der eigenen Seite[2] an.

Weitere Argumente sind die Leistung der CPU und die Steckplätze für Speicher- und Peripheriekarten. Ein Großteil der auf dem Markt populären Hardware–Router müssen mit der halben Taktfrequenz (wenn überhaupt) auskommen. Dies genügt für normales Routing kleinerer Firmen, in den meisten Haushalten sollten diese Leistungsreserven auch einer Telefonanlage genügen. Doch gerade bei mehreren, gleichzeitigen Gesprächen,

[1] http://www.mikrotik.com
[2] http://www.routerboard.com

die zudem eine Umsetzung des Codecs benötigen, ist die Prozessor–Leistung ein entscheidender Faktor.

Mit Hilfe des CompactFlash–Slots kann der interne Flash–Speicher stark erweitert werden, was auch den Betrieb von speicherintensiven Applikationen ermöglicht. Wenigere Schreibvorgänge in den internen Flash–Speicher lassen diesen zudem langsamer verschleissen, was wiederum die Lebensdauer des Gerätes erhöht. Bezogen auf den Einsatz als Telefonanlage könnten beispielsweise Anrufbeantworter–Aufnahmen auf der CompactFlash–Karte abgelegt werden.

Ausser den zwei MiniPCI–Slots befindet sich zur Erweiterung mit PCI–Hardware noch ein Daughterboard–Steckplatz auf der Platine. Mikrotik bietet beispielsweise eine Erweiterungskarte mit vier zusätzlichen MiniPCI–Slots und sechs zusätzlichen Netzwerkkarten an. Gerade diese Erweiterungsmöglichkeiten sind ein entscheidendes Kriterium bei eingebetteten Systemen. Viele alte Accesspoints hätten eine neue Bestimmung als Printserver, wenn es möglich wäre, einen USB–Controller nachzurüsten.

5. Portierung und Integration

Eine erste Aufgabe bestand darin, einen 2.4er Linux–Kernel soweit vorzubereiten, dass er auf dem RouterBoard bootet. Als Ausgangsbasis diente ein von Mikrotik veröffentlichter Patch für Kernel 2.4.30. Darin enthalten waren nicht nur der benötigte Programmcode, sondern auch sehr viele unrelevante Änderungen, wie z.B. Unterstützung für Ethernet Bridging Tables oder exotische IPTables–Module. Der erste Schritt bestand also darin, die essentiellen Kerneländerungen zu finden, und zu extrahieren.

Besonders hinsichtlich späterer Integration ins FreeWRT Buildsystem musste der resultierende, kleine Patch an die zu dem Zeitpunkt aktuelle Kernelversion 2.4.33 angepasst werden. Diese Tätigkeit war eine gute Schulung, Patches an leicht veränderten Programmcode anzupassen, was in Software–Projekten immer wieder getan werden muss. Das Beispiel eignete sich daher ganz gut, weil die Veränderungen nie schwerwiegend waren, dafür aber sehr häufig auftraten.

Die Basis für die Integration in den Linux–Kernel 2.6 bildeten Patches vom OpenWrt–Projekt. Ursprünglich wurden sie der Referenzimplementierung von IDT entnommen und auf Kernel 2.6.17 angepasst. Für das FreeWRT–Projekt mussten sie an den damals aktuellen Kernel 2.6.18 angepasst werden. Im Zuge der Vorbereitung zur Einsendung an das Linux Subprojekt für MIPS–Architekturen erfolgte später eine weitere Anpassung an Version 2.6.19. Da sich im Kernel 2.6.19 Teile der Memory Technology Devices (MTD) API geändert haben, war gerade letztere Anpassung sehr aufwendig.

Um erste Tests mit dem neuen Kernel zu machen, empfiehlt es sich eine Netzboot–Umgebung einzurichten. Klassisches Netzboot (nicht zu verwechseln mit PXE) funktioniert folgendermassen:

1. Das bootende System erfragt folgende Informationen per `DHCP` oder `BOOTP`:
   - Die eigene IP Adresse, Netzmaske und Defaultroute.
   - Die IP Adresse des TFTP–Servers, welcher die Kerneldatei ausliefert. (Deren Name auch erfragt wird.)
   - Die IP Adresse des NFS–Servers, der ein Root–Dateisystem zur Verfügung stellt. (Evtl. auch den Namen des Shares, per Default */tftpboot/IP_Adresse/*.)
2. Der Kernel wird vom TFTP–Server heruntergeladen und direkt gebootet.
3. Wenn die Kernelinitialisierung abgeschlossen ist, wird das Root–Dateisystem per NFS gemountet und der Bootvorgang im Userspace fortgeführt.

Da Kernel und Root–Dateisystem serverseitig abgelegt sind, können Änderungen sehr schnell vorgenommen werden. Der vermeintlich standardisierte Ablauf des Bootvorgangs

verspricht zwar ein Vermeiden fehlerhafter Installationen, jedoch ist die korrekte Einrichtung einer Netzboot–Umgebung nicht gerade trivial, und stellt sich daher wiederum als eigene Fehlerquelle dar.

Nachdem erste Laufzeittests keine schwerwiegenden Probleme mit dem Kernel aufgewiesen hatten, konnte mit der Integration der notwendigen Patches in das FreeWRT ADK begonnen werden. Abgesehen davon, dass zunächst die notwendige Verzeichnisstruktur für ein neues Zielsystem eingerichtet werden musste, verkomplizierte ein weiterer Umstand die Situation: alle anderen unterstützten Geräte erlauben ein Flashen im herkömmlichen Sinn - vom TFTP–Server wird ein Dateisystem–Image heruntergeladen, der Flash–Speicher damit beschrieben, und abschließend neu gestartet. Das Routerboard unterstützt diesen Vorgang nicht, sodass als erstes Zieldateisystem *nfsroot* definiert wurde. Nach Abschluss des Buildvorgangs wird in der Console auf Dokumentation und Beispielkonfigurationen zur Einrichtung der Netzboot–Umgebung hingewiesen, und ein Tarball erzeugt, welcher Rootdateisystem und Kernel beinhaltet.

In der darauf folgenden Zeit wurde die Installation auf eine CompactFlash–Karte als weiteres Zieldateisystem definiert und integriert. Allerdings gelang auch dies nicht ganz problemlos: da dieses Medium für keines der anderen, unterstützten Geräte verfügbar ist, fehlten die notwendigen Werkzeuge, um ein sachgemässes Image zu erstellen. Der Flash–Speicher wird in der Regel vom Kernel partitioniert, d.h. es gibt keine Partitionstabelle auf dem Medium, sondern im Kernel sind Offset, Grösse und Name jeder Partition eingetragen. Das Anlegen einer Partitionstabelle war also das erste Problem. Ein weiterer Punkt ist, dass auf Flash–Speicher spezielle Dateisysteme zum Einsatz kommen. Da normalerweise sehr wenig Speicherplatz zur Verfügung steht, fallen diese sehr minimalistisch aus. Praktisch wird lediglich die Erase–Size des Flash–Speichers benötigt, um ein Image mit einem gültigen Dateisystem zu erstellen. Dieses Image kann dann unabhängig von seiner Grösse in eine Flash–Partition geschrieben werden. Dies ist mit dem auf der CompactFlash–Karte eingesetzten Ext2–Dateisystem nicht so leicht: beim Erstellen des Images muss die Größe genau bekannt sein und exakt mit der Größe der Partition übereinstimmen, in die später geschrieben wird. Ist sie größer, bleibt Speicherplatz ungenutzt. Ist sie kleiner, kommt es zu Datenverlust und Dateisystemkorruption.

Die Unterstützung für den internen NAND–Flash war problematisch. Die ersten RouterBoards hatten nur 64MB Flash, mit 512Byte Page–Size. Ab einer gewissen Revision wurden die Boards mit 128MB Flashbausteinen mit 2048Byte Page–Size bestückt. Zu einem Zeitpunkt, als noch unklar war, ob der Speicher vom Kernel überhaupt korrekt angesprochen wird, wurden Tests mit MTD–Utilities gemacht, die mit der großen Page–Size nicht umgehen konnten. Ein BIOS–Upgrade brachte dann eine Option zur Formatierung des Flashs mit sich, wodurch auf die MTD–Utilities vorerst verzichtet werden konnte. Als Dateisystem sollte YAFFS2[1] verwendet werden, da es speziell für den Einsatz auf NAND–Flash geschrieben wurde. YAFFS2 ist noch nicht in Linus' Kernelquellen eingeflossen, sodass es nachträglich hinzugepatcht werden muss. Hierfür wurden anfangs die von Mikrotik zur Verfügung gestellten Patches benutzt, die auch nur bis zu 512Byte Page–Size unterstützten. Mit aktuellen YAFFS2–Quellen konnte der Flash–

[1] http://www.aleph1.co.uk/

Speicher dann auch genutzt werden.

Das YAFFS2–Dateisystem besitzt die Eigenheit, dass zur Suche einer Datei der gesamte Dateisystembaum durchsucht werden muss. Besonders beim Laden des Kernels vom Flash–Speicher ist dies störend, da sich der Bootvorgang unter Umständen erheblich verzögert. Die Entwickler bei Mikrotik teilten den Flash–Speicher im Kernel daher in eine kleine Partition für den Kernel und eine große für den restlichen Speicher auf.

Seitens der YAFFS2–Entwickler wurde allerdings eine Technologie entwickelt, um das Dateisystem einer Partition für schnelle Suche zu indexieren. Hierzu werden einige Blöcke am Anfang der Partition zur Haltung der Meta–Daten reserviert, und als belegt markiert. Bei einem Defaultwert von zehn reservierten Blöcken zu 2048Byte (Page–Size) in Kombination mit einer 4MB großen Partition führt dies zum Erscheinen eines initial halbvollen Dateisystems. Verwirrend an diesem Umstand ist, dass sich der Flash–Speicher ganz ähnlich verhält, wenn er vor dem Mounten nicht vollständig gelöscht wurde.

Um das Editieren der Kernelquellen unnötig zu machen, wurde die Anzahl reservierter Blöcke von mir in eine Konfigurationsoption umgewandelt, zusammen mit der Möglichkeit, das sogenannte Checkpointing ganz zu deaktivieren. Die YAFFS2–Entwickler arbeiten zwar derzeit an einer Logik zur automatischen Berechnung der Anzahl an benötigten Blöcken in Abhängigkeit zur Partitionsgröße, jedoch wurden meine Erweiterungen akzeptiert und in deren CVS–Baum aufgenommen.

Da FreeWRT grundsätzlich als unprivilegierter Benutzer gebaut wird, ergeben sich für die Imageerzeugung bzw. Installation einige Probleme:

- Besitzer und Gruppe von Dateien und Verzeichnissen können nicht korrekt gesetzt werden.
- Es können keine Gerätedateien erzeugt werden.
- `mount` kann nicht verwendet werden.

Das FreeWRT ADK liefert Programme zur Erzeugung von Flash–Images mit, die auch Gerätedateien erzeugen oder Besitzer und Gruppe setzen können. Da einerseits das YAFFS2–Dateisystem von diesen Programmen noch nicht unterstützt wurde und andererseits die zur Erstellung einer Partitionstabelle für CompactFlash–Karten benötigten Werkzeuge fehlten, wurden kleine Skripte erstellt, die bei der Installation assistieren.

Viele der von FreeWRT unterstützten Geräte besitzen Taster und Leuchtdioden, die aus dem Betriebssystem heraus abgefragt bzw. gesteuert werden können. Wenn vorhanden, werden sie benutzt, um das Betriebssystem zu Rettungszwecken in einer Standardkonfiguration zu starten. Da auch das RouterBoard über Taster und LED verfügt, wurde ein Treiber geschrieben, der die Ansteuerung über Dateien im Proc–Dateisystem ermöglicht.

Da die Integration der Kernelunterstützung (sowohl 2.4 als auch 2.6) und aller Dateisysteme komplett durch mich erfolgt ist, stellten die in diesem Kapitel genannten Tätigkeiten einen sehr großen Teil meines Praxissemesters dar.

6. Asterisk

Asterisk ist eine Software–Telefonanlage auf Open Source Basis. Neben der ausgereiften Unterstützung von Voice over IP zeichnet es sich durch eine hohe Anzahl an Funktionalitäten aus. Durch Nutzung einer ISDN–Karte oder eines Mobiltelefons (welches per Bluetooth angesprochen wird) kann Asterisk ans Mobilfunk- bzw. Festnetz angebunden werden.

Die hohe Flexibilität in der Anwendung erreicht Asterisk vorallem durch modulare Struktur des Quellcodes. So können Entwickler definierte Schnittstellen nutzen, um neue Funktionalitäten einzubinden, die dann als separate Objektdateien angelegt werden. Asterisk erlaubt dem Administrator, in einer Datei anzugeben, welche dieser Funktionalitäten zur Laufzeit benötigt werden und welche nicht.

Ein Kritikpunkt seitens der ISDN–Unterstützung von Asterisk ist die geringe Vielfalt bei der Wahl der ISDN–Karte. Dass auf das RouterBoard nur Karten im MiniPCI–Format passen, grenzt den Rahmen weiter ein. Hinzu kommt, dass Primary Rate Interface Karten, wie sie im High–End–Bereich eingesetzt werden, zu teuer für die angesprochene Zielgruppe sind. So bleiben letztendlich nur die preiswerten Basic Rate Interface Karten mit HFC–Chip von Cologne Chip [1] übrig. Treiber für diese Karten zur Verwendung mit Asterisk sind leider jedoch weder im Linux–Kernel noch in Asterisk fest integriert, sodass Software externer Projekte zusätzlich benötigt wird.

Derzeit erfährt Asterisk ein Versionsupgrade. Seit Ende Dezember 2006 ist die Version 1.4 verfügbar, welche die bisherige Version 1.2 ablösen wird. Initial wurde Version 1.4 portiert und in das FreeWRT ADK integriert, jedoch zeigten sich bei Tests Probleme mit der neuen Version: die Übertragung von frei/besetzt–Signalisierungen einzelner Endgeräte an Sip–Telefone, sogenannte *hints*, funktionierten nicht. Mit dieser Funktionalität kann auf einem Telefon angezeigt werden, wenn mit einem anderen gerade telefoniert wird. Da nicht abzusehen war, wie lange die Entwicklergemeinde für einen Fix benötigen werde, wurde parallel Asterisk 1.2 portiert und integriert.

Der Asterisk–1.4–Quellcode enthielt einen Designfehler, welcher Probleme bei der Integration ins FreeWRT ADK machte. Dieser wurde behoben und die Korrekturen als Bug bei Digium eingereicht[2]. Die Änderungen wurden vom Entwicklerteam verifiziert und in Asterisk–1.4 integriert.

Zur Gestaltung der Telefonanlage für die Firma AurISP IT Consulting wurden zunächst die Funktionalitäten der bisherigen Lösung festgehalten:

Endgeräte vier VoIP–Telefone, optional lauschen mehrere Telefone auf einer Durchwahl.

[1] http://www.colognechip.com
[2] http://bugs.digium.com/view.php?id=8814

Amtsanbindung ISDN–Mehrgeräteanschluss mit zehn MSN, zudem VoIP über SipGate[3].

Anrufbeantworter pro Endgerät, nicht deaktivierbar.

Rufumleitung per Kurzwahl steuerbar, Zielrufnummer wählbar.

Faxanbindung Faxversand über ISDN ist möglich.

Anschließend wurde eine Liste mit Funktionalitäten erstellt, um welche die obigen Grundfunktionen erweitert werden sollten:

Konferenzen Konferenzschaltungen sollten sowohl rein intern, als auch unter Miteinbeziehung externer Anrufer vom Telefon aus schaltbar sein.

Anrufe heranholen Anrufe auf anderen Durchwahlen sollen am Telefon (per Leuchtdiode) signalisiert werden, Heranholen des Anrufes auf den eigenen Apparat muss per Tastendruck oder Kurzwahl möglich sein.

Rufumleitung Der Status der Anrufweiterleitung soll per Leuchtdiode am Endgerät signalisiert werden.

Rufumleitung 2 Weiterleitung bei Nichtannahme des Anrufs sollte möglich sein.

Die Umstellung der Anlage sollte genauso in zwei Schritten erfolgen: nach Abbildung und Test der bisherigen Funktionalitäten in der neuen Anlage sollte der Wechsel erfolgen. Nach einer kurzen Phase des Tests sollte die Konfiguration dann schrittweise um die neuen Funktionalitäten erweitert werden.

Aufgrund der im folgenden Kapitel erläuterten Verzögerungen und Probleme mit der ISDN–Unterstützung blieb leider weder Zeit die Konfiguration vorzubereiten, noch die Umstellung der Anlage vorzunehmen. Daher wird im Folgenden lediglich jene Testkonfiguration erläutert, welche unter Verwendung von `mISDN` zu zufriedenstellenden Ergebnissen geführt hat.

/etc/asterisk/sip.conf In dieser Datei werden alle SIP–Teilnehmer eingetragen. Dazu gehören nicht nur VoIP–Telefone, sondern auch etwaige SIP–Provider im Internet. Die Konfiguration im Anhang definiert lediglich IP–Adresse und Port, auf dem Asterisk für SIP–Verbindungen lauschen soll, und ein Telefon. Um die spätere Konfiguration der Durchwahlen zu erleichtern, wird der Name des Endgeräts mit der Durchwahl gleichgesetzt (Zeilen 8 und 19). Eine weitere, wichtige Angabe ist Zeile zehn abzulesen, die Variable *context* bestimmt, von welchem Teil des Wählplans ausgehende Anrufe des Telefons abgehandelt werden.

/etc/misdn-init.conf Diese Datei enthält die nötigen Informationen zu vorhandenen ISDN–Karten. Zeile eins definiert die erste (und einzige) Karte im System als HFC–Karte, welche über PCI angesprochen wird. Zeile zwei gibt an, dass der erste Anschluss Amtsseitig eingesetzt wird, und ein Mehrgeräteanschluss ist.

[3] http://www.sipgate.de

/etc/asterisk/misdn.conf Hier wird die eigentliche Konfiguration des `mISDN`–Subsystems vorgenommen. Nach Deklaration einiger Default–Werte wird ein ISDN–Kanal namens *Extern* definiert. Zeile 27 bestimmt die zugehörigen ISDN–Anschlüsse, Zeile 29 die zulässigen MSN für eingehende Anrufe. Auch hier kommt eine Variable *context* zum Einsatz, welche den Einsprungpunkt in den Wählplan für eingehende Gespräche von ISDN bestimmt.

/etc/asterisk/extensions.conf Dies ist der Wählplan, und somit das Herzstück der Asterisk–Konfiguration. Unter *general* können allgemeine Einstellungen vorgenommen werden, unter *globals* werden in der Regel statische Variablen zur späteren Verwendung initialisiert. Des weiteren existieren die beiden, zuvor genannten, Einsprungpunkte *intern* und *misdn-in*.

Wählplanregeln beginnen stets mit „*exten =>* ". Der Rest der Zeile setzt sich aus drei kommaseparierten Angaben zusammen: die gewählte Nummer, eine fortlaufende Nummerierung aller Regeln pro Einsprungpunkt und gewählter Nummer und der eigentlichen Aktion (eine sogenannte *Application*). Wie in Zeilen sieben und acht zu sehen, kann die gewählte Nummer auch als regulärer Ausdruck formuliert werden. In diesem Fall gelten die Regeln für alle Nummern, die mit einer Null beginnen und länger als eine Ziffer sind. Für diese Nummern soll über den ersten `mISDN`–Kanal gewählt werden. Die Variable `EXTEN` enthält die gewählte Nummer, die Modifikation „`:1`" schneidet die führende Null ab. Nach Beendigung des Anrufs wird dann in jedem Fall aufgelegt.

Anrufe, die per `mISDN` hereinkommen, werden ganz ähnlich behandelt, jedoch werden nur Anrufe auf eine einzelne MSN akzeptiert, der rest ignoriert. Der `NoOp`–Befehl gibt lediglich die angegebenen Parameter als String auf der Asterisk–Console aus, und kann somit zur funktionalen Betrachtung ignoriert werden. Die beiden verbleibenden Einträge starten die Weiterleitung des Anrufs an den SIP–Teilnehmer mit dem Namen 13. Nach Beendigung des Anrufs oder einem 60 sekündigen Timeout wird aufgelegt. Der letzte Parameter des `Dial`–Befehls löst das Läuten des Zielgerätes explizit aus und kann wahrscheinlich weggelassen werden.

7. ISDN

Für erste Tests stand eine HFC–Karte von Cameronet (http://www.cameronet.com) zur Verfügung. Da ein Quarz auf der Platine nachträglich neu angelötet worden war, konnte über die Funktionalität allerdings keine Aussage gemacht werden. Eine herkömmliche ISDN–Einwahl mit Hilfe des `hisax`–Treibers und der `isdn4k--utils` sollte als simpler Funktionstest der Hardware dienen.

Da bei ausgiebigen Tests keine Erfolge hinsichtlich einer erfolgreichen ISDN–Einwahl verzeichnet werden konnten, wurde eine zweite Karte nachbestellt, die allerdings aufgrund von Lieferschwierigkeiten sehr lange auf sich warten ließ. Da jedoch auch mit der neuen Karte keine Erfolge verzeichnet werden konnten, erfolgten weitere Tests auf einem x86–basierten, eingebetteten System mit MiniPCI–Slot. Da auf diesem System keinerlei Einwahlprobleme auftraten, wurde ein x86–PC mit einer ebenfalls HFC–basierten PCI–ISDN–Karte ausgestattet und als Testsystem aufgesetzt, sodass auftretende Probleme stets gegengeprüft werden konnten.

Die Wahl des Treibers zur Integration der ISDN–Unterstützung in Asterisk fiel zunächst auf den von Herrn Junghanns geschriebenen `zaphfc`–Treiber[1]. Dieser erfordert allerdings veränderungen an den Asterisk–Quellen, sodass sich die Wahl der eingesetzten Asterisk–Version auf die vom Autor vorgesehene beschränkt, was ein weiterer Grund für die parallele Portierung von Asterisk–1.2 war. Jedoch verhielt sich dieser Treiber sowohl auf dem Routerboard als auch auf dem x86–Testsystem sehr instabil. Obwohl sowohl eingehende als auch ausgehende Anrufe erkannt wurden, kam es nur eingehend zum Läuten des Endgerätes. Gespräche konnten über das Routerboard nicht aufgebaut werden, über das x86–System nur sporadisch.

Die Analyse der aufgetretenen Probleme verzögerte das Projekt zu dem zeitlich schon recht fortgeschrittenen Stadium erheblich, wobei zu beachten ist, dass die Anbindung ans ISDN–Netz einerseits rudimentär für die Fertigstellung des Produkts, andererseits als Funktionalität von der Telefonanlage für meinen Arbeitgeber erwartet wurde.

Als Fortschritte bei Tests an einem anderen ISDN–Anschluss verzeichnet werden konnten wurde das NTBA, eine sogenannte ,,Starterbox" von Arcor, ausgetauscht, mit dem Effekt, dass zumindest sporadisch ausgehende Gespräche über das x86–Testsystem geführt werden konnten. Ein Telefonat mit dem Provider Arcor ergab, dass mit angeschlossener Starterbox drei Volt Fremdspannung auf der ISDN–Leitung anlägen. Da jedoch auch mit dem alternativen NTBA keine Gespräche übers Routerboard geführt werden konnten, schien parallel ein Treiberproblem zu existieren, zumindest auf der MIPS–Architektur.

In einem letzten Ansatz wurden alternative Treiber auf dem x86–System getestet,

[1] http://www.junghanns.net

um sie bei erfolgreichem Test nach MIPS zu portieren und auf dem Routerboard zu testen. Die Wahl fiel hierzu zunächst auf `vISDN`[2], einer kompletten Implementierung des ISDN–Stacks für Linux von Daniele Orlandi. Da sich der Treiber auf dem x86–System als nicht–funktional erwies, wurde er verworfen. Die nächste Wahl fiel auf `mISDN`[3], dem selbsternannten Nachfolger des `hisax`–Treibers. Zwar befinden sich die zum Download zur Verfügung stehenden Tarballs in einem eher schlechten Zustand, jedoch konnten bei Einsatz der aktuellen Quellen aus dem CVS–Baum erfolgreich ein- und ausgehende Gespräche über das x86–System geführt werden.

So ergab es sich, dass ich meine letzten Tage bei der Firma AurISP IT Consulting mit der Portierung und dem initialen, erfolgreichen Test von `mISDN` auf dem Routerboard verbrachte. In einem etwa einstündigen Laufzeittest konnten mehrere, ein- wie ausgehende Telefonate geführt werden, wobei die Sprachqualität als tadellos empfunden wurde. Auch die Systemlast von etwa 10-20% bei 400MHz CPU–Takt lag weitgehend im Rahmen der Erwartungen.

[2] http://www.visdn.org/
[3] http://www.misdn.org/

8. Tests

Getestet wurde in zwei Etappen: nachdem die Portierung beider Kernel und die Unterstützung für die Zieldateisysteme abgeschlossen war, erfolgten diverse Performance- und Stabilitätstests. Da sich der zweite Teil der Tests auf Asterisk als Kernkomponente bezog, wurde mit diesem bis nach Abschluss der Asterisk–Portierung gewartet.

Stabilitäts- und Performancetests

Zum Test von Performance und Stabilität der Kernel–Portierung wurde eine Reihe von Testumständen definiert und in Kombination jeweils auf Kernel–2.4 und 2.6 angewendet. Getestet wird das Netzwerk–Subsystem als zentrales Element, da dieses auch im späteren Einsatz eine zentrale Rolle spielt. Die Testumstände sind:

CPU–Last künstlich erzeugt oder nativ

Kommunikationskanal verschlüsseltes VPN oder direkt

CPU–Last kann mit dem Unix–Tool `stress` erzeugt werden, welches so konfiguriert wurde, dass es acht Threads Primzahlen berechnen lässt, während zwei weitere kontinuierlich Hauptspeicher allokieren und wieder freigeben.

Für das VPN–Netz kam OpenVPN mit einem statischen Schlüssel zum Einsatz. Das Besondere an dieser Art der Lasterzeugung ist, dass sie in direkter Verbindung zu den gefahrenen Netzwerktests steht. Um Anfragen aus dem Netzwerk zu erhalten und zu beantworten ist jeweils eine, wenn auch kleine, Berechnung notwendig.

Als wertegenerierende Werkzeuge wurden verwendet:

ping im *flood*–Modus (sendet den nächsten PING direkt nach erhaltenem PONG), 100.000 Pakete

netio sowohl per UDP– als auch TCP–Protokoll, kleine und große Pakete

Mit `ping` kann die Reaktionszeit des Systems getestet werden. Die hohe Zahl gesendeter Pakete wurde gewählt, um die Messergebnisse zu stabilisieren.

Mit Hilfe von `netio` können Aussagen über die zur Verfügung stehende Bandbreite getroffen werden. Für eine bessere Relativierung der Ergebnisse erfolgte der Test sowohl auf UDP als auch TCP, zudem wurden kleine (1KB) und große (32KB) Pakete verwendet. `netio` ist eine Client–Server–Applikation, für alle Tests agierte das RouterBoard als Server, während ein zweiter Rechner als Client (und somit Initiator der Testläufe) auftrat.

		2.4	2.6
plain	total	25,310ms	25,593s
	min/avg/max (ms)	0,158 / 0,226 / 6,305	0,174 / 0,229 / 0,839
vpn	total	201,124s	197,124s
	min/avg/max (ms)	1,835 / 1,978 / 32,995	1,762 / 1,938 / 28,758

Abbildung 8.1.: Flood–Ping, idle

Wie in Abb. 8.1 zu erkennen ist, verschlechtern sich die Pingzeiten bei Nutzung von VPN in etwa um den Faktor acht. Bemerkenswert ist, dass Kernel–2.6 im unbelasteten Zustand zwar etwas langsamer reagiert, über VPN aber schneller antworten kann als Kernel-2.4, was auf bessere Performance im Lastbetrieb hindeutet.

		2.4	2.6
plain	total	27,310s	25,378s
	min/avg/max (ms)	0,169 / 0,245 / 10,384	0,167 / 0,227 / 0,698
vpn	total	-	199,440s
	min/avg/max (ms)	-	1,769 / 1,956 / 26,740

Abbildung 8.2.: Flood–Ping, busy

Abb. 8.2 bestätigt die ersten Annahmen, im Lastbetrieb antwortet Kernel-2.6 etwas schneller über die unverschlüsselte Leitung als Kernel–2.4. Dass zum Volllastbetrieb über VPN für Kernel–2.4 keine Werte angegeben sind, liegt daran, dass neben sehr hohen Latenzzeiten auch Paketverluste auftraten, was die Messergebnisse weiter verschlechtert. Da Kernel–2.4 ohnehin nicht im Produktivbetrieb eingesetzt werden sollte, wurde dieses Problem, welches vermutlich auf die schlechte Qualität der Kernelportierung zurückzuführen ist, nicht weiter beachtet.

Die Ergebnisse der Tests mit `netio` sind mir nicht schlüssig. Es ist zwar unklar, ob es sich hierbei um Messfehler oder ein tatsächliches Problem handelt, deutet die scheinbar nicht vorhandene Struktur der Performance–Einbrüche jedoch eher auf Messfehler hin.

Test der Asterisk–Integration

Nachdem die Portierung von Asterisk 1.4 abgeschlossen war, wurde es intensiv getestet. Zum einen wurde eine Liste an Funktionalitäten aufgestellt, die im Versuchsaufbau abgebildet wurden. Zum anderen wurde die Systemlast bei laufenden Gesprächen betrachtet, und (mangels Telefonen) in einer Hochrechnung die gleichzeitig maximal möglichen Gespräche ermittelt. Bemerkenswert ist hierbei, dass es große Unterschiede macht, ob zwischen zwei Teilnehmern eine Transkodierung des Sprachcodecs erfolgt oder nicht, weswegen beide Fälle separat betrachtet wurden.

Die abgeprüften Funktionalitäten sind im folgenden:

		2.4 (TX / RX)	2.6 (TX / RX)
plain, idle	1k, tcp	5M / 4M	3M / 4M
	1k, udp	11M / 6M	1M / 4M
	32k, tcp	9M / 7M	9M / 6M
	32k, udp	12M / 12M	10M / 12M
plain, busy	1k, tcp	584K / 353K	713K / 299K
	1k, udp	11M / 549K	478K / 1M
	32k, tcp	1M / 615K	2M / 511K
	32k, udp	12M / 1M	5M / 3M
vpn, idle	1k, tcp	434K / 428K	450K / 433K
	1k, udp	305K / 252K	71K / 157K
	32k, tcp	440K / 428K	458K / 438K
	32k, udp	7K / 206K	73K / 123K
vpn, busy	1k, tcp	63K / 24K	153K / 23K
	1k, udp	43K / 43K	18K / 28K
	32k, tcp	46K / 26K	92K / 27K
	32k, udp	5K / 39K	5K / 24K

Abbildung 8.3.: Messungen mit `netio`

Echotest mit Ansage Test von Sprachqualität der Ansage, Latenz des Echos und Erkennen der Rautetaste zum Beenden.

Wechselseitiger Rufaufbau mit und ohne Codec Umwandlung Test von Sprachqualität, Verzögerung nach dem Abheben, Verbindungen zwischen den verschiedenen Technologien.

Abhören von Musiconhold Prüfung auf ruckelfreie und qualitativ hochwertige Wiedergabe.

Weiterverbinden von Gesprächen Test auf erfolgreiches weiterverbinden mit Wartemusik.

Voicemail Aufsprechen Test von Aufnahme und Richtigkeit der Ansagen.

Voicemail abhören Test von Wiedergabe und Speichern/Löschen der Aufnahme, Möglichkeit zum Ändern der Ansagetexte.

Aufsprechen einer Ansage Test erfolgreicher Aufzeichnung.

Konferenz mit mindestens drei Teilnehmern Test auf Funktion, Audioqualität und Latenz.

Authenticate Test der Authentifikationsmöglichkeiten.

Enum Lookup Test erfolgreicher Lookups.

Asterisk DB Test erfolgreichen Einfügens, Auslesens und Entfernens von Einträgen aus der Asterisk–eigenen Datenbank.

Presence / Subscriptions / Hints / Directed Chan Pickup Test diverser Benachrichtigungsmechanismen.

Diese Liste wurde in konkreterer Form für spätere Regressionstests vorbereitet, das heißt es existiert eine Asterisk–Konfiguration und für jeden Test wurde genau festgehalten wie er durchzuführen, und worauf bei der Wertung zu achten ist.

Die Tests wurden einheitlich bei 200MHz CPU–Takt durchgeführt, und die Hochrechnungen auf die damals maximalen 400MHz durch Stichproben erfolgreich geprüft.

Die Messungen ergaben, dass bei 200MHz 14% der CPU–Zeit pro Gespräch benötigt wird, bzw. 25% wenn Transkodierung erforderlich ist. Das entspricht 28 bzw. 50MHz und somit sieben bzw. vier gleichzeitigen Gesprächen. Hochgerechnet auf 400MHz ergeben sich im besten Fall 13, im schlechtesten acht gleichzeitige Gespräche.

Da es wenig verlässliche Aussagen über den durchschnittlichen Anteil telefonierender Personen in einem Unternehmen gibt, ist es schwierig, die oben genannten Ergebnisse zu werten. Als Bewertungsmassstab könnte jedoch die bisherige Situation in Unternehmen dienen, die lediglich ISDN einsetzen. Somit könnte angenommen werden, dass eine Firma mehr als drei ISDN–Anschlüsse nutzen muss, um an die Grenzen der Anlage zu gelangen. (Bei Verbindungen zu ISDN kann immer von Transkodierung ausgegangen werden.)

Diese Aussage relativiert sich jedoch wieder, da bei den Tests keine ISDN–Hardware zum Einsatz kam. Die hierbei notwendige Ansteuerung der Hardware lässt auf eine höhere Ressourcenbeanspruchung im Betrieb schließen. Wird die Telefonanlage jedoch rein auf IP betrieben, und ein Provider zur Anbindung ins Festnetz genutzt, liegen die Kapazitäten der Anlage um ein Vielfaches höher.

Genauere Aussagen werden sich wohl erst durch Langzeittests im Produktivbetrieb machen lassen. Die Installation der Anlage bei der Firma AurISP IT Consulting war als der Start eines solchen Tests gedacht, was allerdings durch die fehlende ISDN–Funktionalität noch nicht erfolgen konnte.

Jedoch hat sich die Portierung des Gesamtsystems im gesamten Testbetrieb als sehr stabil erwiesen, es traten keine Systemabstürze und lediglich konfigurationsbedingte Abstürze von Asterisk auf.

9. Fazit

Eine Internetrecherche bestätigt, dass Telefonanlagen der hier gezeigten Art Mangelware auf dem Markt sind. Die Liste an bestehenden Alternativen teilt sich auf in Produkte für Privatkunden, die in der Regel mit ein bis zwei gleichzeitigen Gesprächen auskommen, und Produkte für Firmen, die gleich ein bis mehrere hundert Gespräche gleichzeitig führen wollen. Bei einer vorsichtigen Schätzung von 50% telefonierenden Personen ergibt sich für ersteren Fall eine Familiengröße von vier Personen, im letzteren Fall eine Firmengröße von mindestens 200 Mitarbeitern. Dabei sind es gerade junge und kleine Unternehmen, die Bedarf an einer preiswerten Telefonanlage haben. Auch das Kapital für eine ,,Investition in die Zukunft" fehlt hier in der Regel - von der Sinnfrage einer Investition in eine Telefonanlage abgesehen. Ein weiterer Aspekt ist in meinen Augen die Vertragsbindung. Alle namhaften Hersteller bieten ihre Hardware im Bundle mit einem ISDN–, VoIP– oder DSL–Vertrag an, am besten mit allen dreien. Es ist eine besondere Leistung, ein Produkt zu einem erschwinglichen Preis ohne vertragliche Bindung anbieten zu können.

Persönlich hatte ich sehr großen Spaß bei der Arbeit. Während der gesamten Zeit herrschte ein lockeres und kollegiales Klima, was durch die Vertrautheit der Mitarbeiter nicht ganz unbegründet ist. Die Arbeitszeiten waren sehr flexibel. Obwohl dies selbständige Arbeitsweise erforderlich macht, wurde dennoch auf strukturierte und intensive Betreuung geachtet.

Zweiwöchentlich fanden Treffen zwischen allen Beteiligten statt. Hierfür wurden von mir kleine Präsentationen vorbereitet, welche einerseits den aktuellen Stand der Dinge, andererseits einen Blick auf den Gesamtverlauf des Projekts beinhalteten. Letzterer wurde mit Hilfe eines Diagramms visualisiert, welches die vergangene Zeit auf der Abszisse und die einzelnen Stationen des Projekts auf der Ordinate abbildet. Zudem wurden die geplanten Aktivitäten der folgenden zwei Wochen festgehalten.

Die Kombination aus Selbständigkeit und Betreuung, welche in diesem Unternehmen sehr gut getroffen wurde, ermöglicht einen hohen Lerneffekt bei der eigenen Arbeit. Hinzu kommt eine großzügige Ausstattung mit benötigten Lehrmitteln und Testgeräten. Gerade in den Bereichen der Kernelprogrammierung, Portierung von Software auf andere Architekturen und Telefonanlagen (speziell Asterisk) konnte ich große Fortschritte erzielen.

Der Hersteller des Routerboards, Miktrotik, veranstaltete Ende Januar 2007 das sogenannte ,,Mikrotik User Meeting" in Krakau, welches ich zusammen mit meinen Arbeitgebern besuchen durfte. Die Teilnahme an dem Treffen erwies sich als eine sehr gute Entscheidung, denn in einem persönlichen Gespräch mit dem Sales Manager konnte das gegenseitige Interesse bestätigt werden. So wurde die engere Zusammenarbeit zwischen den Entwicklern beider Parteien besprochen, auch der Verweis zu FreeWRT als alter-

natives Betriebssystem für RouterBoard auf Miktrotiks Homepage wurde in Erwägung gezogen. Ferner berichteten Kunden wie Mitarbeiter in Vorträgen über Einsatzszenarien, Erfahrungen und Pläne mit den Mikrotik–Produkten.

Die Ergebnisse meiner Arbeit hinterlassen nicht nur beim FreeWRT–Projekt einen bleibenden Eindruck. Weiterentwicklungen in bestimmten Bereichen wurden den jeweils zuständigen Personen zum Review zugesandt, was die gängige Methode in Open Source Projekten ist, eigene Änderungen ins Projekt einfließen zu lassen. So wurde die RouterBoard–Kernelunterstützung für Kernel–2.6 beispielsweise an Ralf Bächle, den Verantwortlichen für das Linux–MIPS–Projekt, geschickt.

Meine Anpassungen des YAFFS2–Codes zur Unterstützung von Kernelversionen größer 2.6.18 und die Erweiterungen zum besseren Einsatz in sehr kleinen (kleiner 5MB) Partitionen wurden an Charles Manning, einem der YAFFS2–Kernentwickler geschickt, und von ihm integriert. Auch benötigte Anpassungen am Quellcode von Asterisk–1.4.0 wurden der Entwicklergemeinde übermittelt und von dieser integriert.

Dieses Projekt befindet sich eigentlich erst am Anfang. Vorallem die Fertigstellung der ISDN–Anbindung steht noch aus, danach müssen Langzeittests folgen. Somit wird meine Mithilfe am Projekt auch in Zukunft noch benötigt werden, welcher nachzukommen ich gerne bereit bin. Auch über eine weitere Zusammenarbeit mit der Firma AurISP IT Consulting würde ich mich sehr freuen.

A. Referenzen

A.1. RouterBoard-Kernelmodul

```
/*
 * rb-diag.c - button and led driver for Mikrotik's RouterBoard
 *
 * Copyright (C) 2006 Phil Sutter <n0-1@FreeWRT.org>
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 *
 */

#include <linux/module.h>
#include <linux/init.h>
#include <linux/proc_fs.h>
#include <asm/rc32434/gpio.h>
#include <asm/uaccess.h>

/* S1 is at GPIO pin1 */
#define GPIO_S1_BIT (1 << 1)

/* defined in drivers/mtd/nand/rbmipsnand.c */
#define LO_FOFF (1 << 5)
#define LO_ULED (1 << 7)

static struct proc_dir_entry *proc_diag_dir;
static struct proc_dir_entry *proc_reset_file;
static struct proc_dir_entry *proc_led_file;

extern void changeLatchU5(unsigned char, unsigned char);
extern unsigned char getLatchU5State(void);

static GPIO_t rbgpio = (GPIO_t)GPIO_VirtualAddress;
```

```
/* The S1 button state is provided by GPIO pin 1. But as this
 * pin is also used for the serial port the operational modes
 * must be switched first (by changing apropriate bits in gpiocfg
 * and gpiofunc). Therefore the serial port has to be disabled
 * using the changeLatchU5 ksym.
 * This function does the trick and resets everything to it's
 * original values afterwards.                                    50
 */
static unsigned char sense_reset_button(void) {
        unsigned char funcval, cfgval;
        unsigned char value = 255;

        changeLatchU5(0, LO_FOFF);

        /* backup and overwrite */
        funcval = rbgpio->gpiofunc & GPIO_S1_BIT;
        cfgval = rbgpio->gpiocfg & GPIO_S1_BIT;                    60
        rbgpio->gpiofunc &= ~GPIO_S1_BIT;
        rbgpio->gpiocfg &= ~GPIO_S1_BIT;

        /* read reset state */
        value = (rbgpio->gpiod & GPIO_pin1_m) >> 1;

        /* restore values */
        rbgpio->gpiofunc |= funcval;
        rbgpio->gpiocfg |= cfgval;
                                                                   70
        changeLatchU5(LO_FOFF, 0);

        return value;
}

static int proc_read_reset(char *buf, char **start, off_t offset, int size, int *peof, void *data) {
        int ret;
        unsigned char value;

        value = sense_reset_button();                              80
        /* the actual value is inverted here for compatibility to broadcom-diag */
        if(value == 0 || value == 1) value = (value + 1) % 2;

        ret = snprintf(buf, size, "%i\n", value);
        *peof = 1;
        return ret;
}

static int proc_read_led(char *buf, char **start, off_t offset, int size, int *peof, void *data) {
        int ret;                                                   90
        unsigned char val;

        val = (getLatchU5State() & LO_ULED) >> 7;

        ret = snprintf(buf, size, "%i\n", val);
        *peof = 1;
```

```
        return ret;
}
                                                                                100
static int proc_write_led(struct file *inst, const char __user *userbuf, unsigned long count, void *data) {
        char *kbuf;
        int bytes_missing;

        kbuf = kmalloc(count, GFP_KERNEL);
        if(!kbuf) return -ENOMEM;

        bytes_missing = copy_from_user(kbuf, userbuf, count);

        changeLatchU5(0, LO_FOFF);                                              110

        if(strncmp("0", kbuf, 1) == 0)
                changeLatchU5(0, LO_ULED);
        else if(strncmp("1", kbuf, 1) == 0)
                changeLatchU5(LO_ULED, 0);

        changeLatchU5(LO_FOFF, 0);

        kfree(kbuf);
        return count - bytes_missing;                                           120
}

static int __init rbdiag_init(void) {
        proc_diag_dir = proc_mkdir( "diag", proc_root_driver);
        proc_reset_file = create_proc_read_entry("reset", S_IRUGO, proc_diag_dir, proc_read_reset, NULL);
        proc_led_file = create_proc_entry("led", S_IRUGO | S_IWUGO, proc_diag_dir);
        if(proc_led_file) {
                proc_led_file->read_proc = proc_read_led;
                proc_led_file->write_proc = proc_write_led;
                proc_led_file->data = NULL;                                     130
        }
        return ((proc_reset_file && proc_led_file) ? 0 : 1);
}

static void __exit rbdiag_exit(void) {
        if(proc_reset_file) remove_proc_entry("reset", proc_diag_dir);
        if(proc_led_file) remove_proc_entry("led", proc_diag_dir);
        if(proc_diag_dir) remove_proc_entry("diag", proc_root_driver);
}
                                                                                140
module_init(rbdiag_init);
module_exit(rbdiag_exit);

MODULE_AUTHOR("Phil Sutter");
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Make RouterBoard's extra hardware accessible via /proc.");
```

A.2. Asterisk-Testkonfiguration

/etc/asterisk/sip.conf

```
[general]
context=default
bindport=5060
bindaddr=0.0.0.0

[authentication]

[13]
accountcode=3
context=intern
type=friend
secret=xxx
language=de
dtmfmode=rfc2833
host=dynamic
disallow=all
allow=ulaw
allow=alaw
callerid="Phil Sutter" <13>
```

/etc/misdn-init.conf

```
card=1,hfcpci
te_ptmp=1
```

/etc/asterisk/misdn.conf

```
[general]

misdn_init=/etc/misdn-init.conf
debug=4
bridging=no

[default]
context=misdn-default
language=de
senddtmf=yes
nationalprefix=
internationalprefix=
rxgain=0
txgain=0
te_choose_channel=no
pmp_l1_check=no
```

```
reject_cause=16
dialplan=0
localdialplan=0
cpndialplan=0
early_bconnect=yes
hdlc=no

[Extern]
ports=1
context=misdn-in
msns=*
```

/etc/asterisk/extensions.conf

```
[general]

[globals]

[intern]

exten => _0X.,1,Dial(MISDN/1/${EXTEN:1})
exten => _0X.,2,Hangup()

[misdn-in]

exten => 9616457,1,NoOp(before dial)
exten => 9616457,2,Dial(Sip/13,60,R)
exten => 9616457,3,NoOp(after dial)
exten => 9616457,4,Hangup()
```