RBTemp
Verteilte Temperaturmessung

Ulrich Schrittesser
Februar 2011

Embedded Software Engineering
Univ.-Prof. Dr. Christoph Kirsch
VP im Wintersemester 2011
Institut für Computerwissenschaften Universität Salzburg

Verteilte (Echtzeit)-Temperaturmessung in einem Wlan-Netzwerk auf Embedded-Linux-Basis

1. RBTemp

Beim Projekt RBTemp werden Temperaturen mittels Sensoren über eine serielle Schnittstelle an einem Embedded-Board erfasst. Für die Datenerfassung kommt die frei verfügbare Digitemp-Anwendung samt Digitemp-Sensor[1] und zwei Mikrotik-Routerboards mit dem Opensource Betriebssystem Openwrt[2] zum Einsatz. Ein Board dient der Datenerfassung und ein zweites Board als Wlan-Schnittstelle für den verteilten Datenzugriff. Zusätzlich wird die Messarchitektur auf Real-Time Anforderungen und Funktionen untersucht bzw. erweitet.

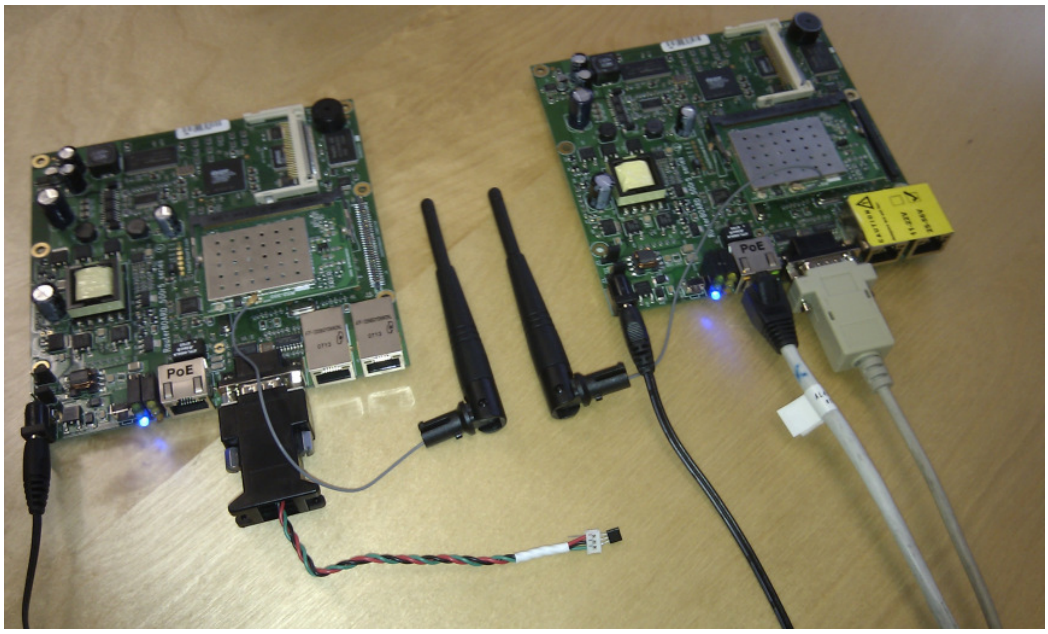


Abbildung 1: Hardware Aufbau

Abbildung 1 zeigt den Hardware-Aufbau, respektive zwei Routerboards mit jeweils einer Wlan-Karte, Lan-Netzwerkzugang und dem Temperatursensor.

2. Hardware

Im folgenden Abschnitt wir die verwendete Hardware genauer beschrieben.

Routerboard (RB)[3]

CPU:	MIPS 32, 400 MHz
Speicher:	64 MB DDR Onboard-Chip
Bootloader:	RouterBOOT
Datenspeicher:	128 MB Onboard-Nand-Chip CF Steckplatz
Ethernet:	3 x 10/100 Mbit/s
Abmessungen:	14 x 14 cm

MiniPCI; Watchdog; RS232; PoE;
Stromversorgung 24 V DC;

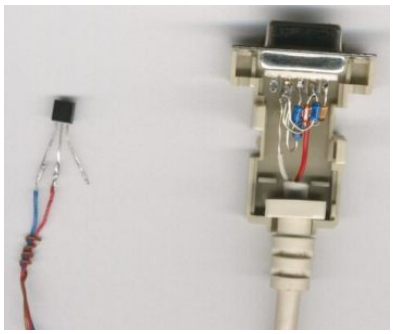
1 http://www.digitemp.com
2 http://openwrt.org
3 http://www.routerboard.com/rb500.html

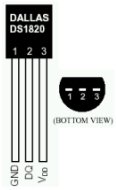
Beim Mikrotik Routerboard 532A handelt es sich um eine voll integrierte Kommunikationsplattform für Hochgeschwindigkeits-, Ethernet- als auch Drahtlos-Netzwerke. Es bietet einerseits 3 LAN-Schnittstellen und andererseits einen Steckplatz für WLAN-Karten im 2,4/5 GHz-Bereich. Zusätzlich besteht eine Erweiterungsmöglichkeit über das Mikrotik Tocherboard 502.

Routerboard Interface R52

Das R52-Wireless-Modul ist eine 802.11a/b/g miniPCI-Karte für Mehrband-Hochgeschwindigkeits-Anwendungen. Hierbei findet das Atheros-AR5414-Chipset mit Turbo / SuperG (108Mbps über den 802.11g Standard bei 2.4 GHz) Verwendung. Das R52 sendet sowohl im Frequenzbereich von 2.312-2.499 GHz als auch von 4.920-6.100 GHz.

Digitemp Temperatursensor:





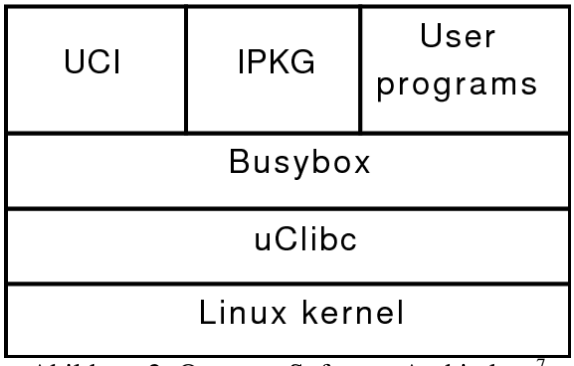
Die Temperaturmessung erfolgt über das digitale Thermometer Maxim Dallas DS1820[4] welches über den Maxim Dallas DS9097U[5] universal 1-Wire COM Port Adapter abgegriffen und an die serielle Schnittstelle angeschlossen ist. Der Sensor erfasst laut Datenblatt Temperaturen zwischen - 55°C und 125° C. Im Netz sind mehrere Bauanleitungen für diesen preiswerten Temperatursensor, speziell für den Bau des Adapters verfügbar. In der RBTemp-Anwendung wird der DS9097U-009 Adapter, an dem das digitale Thermometer über eine Zweidrahtverbindung angeschlossen ist, verwendet.

3. Software

Betriebssystem

Das Mikrotik Routerboard bittet die Wahl zwischen zwei Betriebssystemen, welche beide auf einem Embedded-Linux-System basieren. Zum einen das proprietäre Mikrotik-RouterOS[6], welches eine Vielzahl an Protokollen sowie Tools bereitstellt. Jedoch bietet diese Variante wenige Zugriffsmöglichkeiten für Eigenentwicklungen. Zum anderen besteht die Möglichkeit der Verwendung des Open-Source-Projektes Openwrt, welches speziell als Entwicklungsumgebung (SDK) und Linux-Betriebssystem für Wlan-Router im Netz bereitgestellt wird.

4 http://www.maxim-ic.com/datasheet/index.mvp/id/3021
5 http://www.maxim-ic.com/datasheet/index.mvp/id/2983/t/do
6 http://www.mikrotik.com



Abildung 2: Openwrt Software Architektur[7]

Die Openwrt Community stellt sowohl die einzelnen Kernel- und Filesystemimages, als auch den dazugehörigen Software-Entwicklungs-Satz (SDK Software-Developement-Kit) und einen Image-Builder, für rund ein Dutzend Embedded-Boards im Web bereit.
Genau genommen werden seit 2004 drei Versionen namentlich *White Russian*, *Kamikaze* und eine *Backfire* Version als OS zum Download bereitgestellt. Im RBTemp-Projekt fällt die Entscheidung auf letztere Version[8].

Digitemp

Für den Zugriff auf den Temperatursensor kommt die im Netz freiverfügbare Digitemp-Software[9] zum Einsatz. Diese Linux-Anwendung kümmert sich um den Zugriff auf die Serielle oder USB-Schnittstelle und verspeichert die Daten inklusive Zeitstempel in einem Textfile. Des Weiteren besteht die Möglichkeit mehrere Sensoren an einem Bus zu verwenden und auszulesen.

4. Engineering

Entwicklungsumgebung

Hierbei wurde ein PC mit Linux Debian 4.0 als Entwicklungsrechner (EWR) verwendet. Um Images auf ein Embedded Device zu laden, mussten für den Ersteinstieg über die serielle Schnittstelle ein Terminal (Minicom), sowie ein TFTP- als DHCP-Server am EWR installiert bzw. konfiguriert werden.

Überspielung Openwrt Images

Zuerst wird das vorhandene Betriebssystem „RouterOS“ über den Bootloader vom Speicher des RB gelöscht. Dann muss ein bootfähiges Kernel- samt Filesystem-Image über die Ethernet-Verbindung vom EWR auf das RB nachgeladen werden. Zu guter letzt ladet der Entwickler die auf dem Openwrt-Server bereit gestellten Images[10] permanent in den Speicher des RB.

7 Fainelli F.: The OpenWrt embedded development framework, 2008.
8 http://downloads.openwrt.org/backfire/
9 http://www.digitemp.com/software.shtml
10 http://downloads.openwrt.org/backfire/10.03/rb532/

Installation OPKG-Pakete

Openwrt nutzt verschiedene Embedded Linux-Tools wie uClibc, Busybox, Shell-Interpreter und bietet eine Hardware-Abstraktionsschicht sowie einen Paket-Manager.
Für die OW-Backfire Version bietet einen OPKG Packet-Manager die Möglichkeit verschiedene Anwendungen aus dem Netz auf das Board nachzuladen. Im konkreten Fall wurden das *ssh* (Verteilter Zugriff), *nano* (Editor), *digitemp* (Temperaturmessung) und der *ntpclient* (Zeitsynchronisierung) auf das Board installiert.

Test Digitemp Sensor

Das aus den Openwrt-Sourcen zur Verfügung stehende digitemp_DS9097-Packet funktioniert für den Temperatursensor mit dem Maxim DS9097U Adapter nicht. Da auf eingebetteten Systemen, wie auch im Falle von RBTemp meist kein Compiler zur Verfügung steht, muss der Sourcecode neu cross-kompiliert werden. Hierzu werden die Digitemp-Sourcen auf den EWR geladen und über die Opwrt-Toolchain (Crosscompiler, Bibliotheken, etc.) die neue Digitemp-Anwendung[11] erstellt und auf das Board kopiert.
Beim erneuten Test der Temperaturmessung wurde die beschriebene Antwortzeit von 1 Sekunde in periodischen Messungen nicht erreicht. Der Sensor benötigte teilweise über 40 Sekunden für eine Rückmeldung der Messung. Nebenbei traten einige Fehlmessungen hinzu. Diese Problematiken dürften auf die Eigenschaften des 1-Wire-Bus Systems zurückzuführen sein.

Ein vollständiges Howto, beginnend bei der Einrichtung des EWR bis hin zu Temperaturmessung ist in einem angefügten Pdf-Dokument[12] aufgeführt.

Entwicklungsumgebung

Um eigene Kernel- bzw. Fileimages zu erstellen, kann die Openwrt-SDK zu Hilfe genommen werden. Mittels dieser Umgebung werden die jeweiligen Kernel- oder auch andere Paket-Sourcen auf den EWR geladen, womit für ausgesuchte Embedded Devices beliebige Images erstellt werden können. Die SDK bietet zusätzlich eine Toolchain als Crosscompilierung-Umgebung für weitere Sourcen.
Nach Zusammenstellung bzw. Konfiguration der Kernel- und Rootfilesystems sind sowohl über das Netzwerk boot bare, als auch im Speicher des RB permanente Images[13] erstellt und auf dem Board getestet worden.

RT-Patch für OW-SDK

Im Zuge der Real-Time-Thematik aus der Embedded-Software-Engineering-Vorlesung (ESE) wurden die RT-Eigenschaften der OW-Lösung näher untersucht. Um Linux RT-fähig zu betreiben, sind entweder ein RT-Linux (preemtable Kernel) oder ein Standard Kernel mit RT-Patch notwendig.

---

11 Digitemp-Anwendung: `RBTemp_Digitemp`

12 RBTemp-Howto: `RBTemp_Openwrt_Digitemp_HowTo.pdf`

13 RBTemp-Images: `openwr_kernel_librt; openwrt-rb532-rootfs_librt.tgz`

Das Openwrt-OS bietet grundsätzlich keine besondere Echtzeitfunktionalität. Darum wurde zuerst versucht Kernel-Images mit dem im Netz verfügbaren RT-Patch[14] zu kompilieren. Dies misslang jedoch nach mehreren Versuchen am EWR, aufgrund der unterschiedlichen Kernel-Sourcen der beiden Opensource-Anwendungen zueinander.

- RT-Patch Kernelversionen: 2.6.24.7; 2.6.26.8; 2.6.29.6; 2.6.31.12; 2.6.33.7
- Openwrt-Backfire Kernelversionen: 2.6.30; 2.6.32

Als zeitsparende Weiterentwicklung ist der verwendete Kernel in der Backfire-Version auf seine RT-Eigenschaften untersucht worden.

Allgemein bedarf es für Linux-RT-Anwendungen:

- Linux Schedule Policies
- Locking Techniken (Semaphoren, Spinlocks, Mutex..)
- High-Resolution-Timer (HRT)
- POSIX/Linux Möglichkeiten
- Preemption
- Scheduling (Normal; FiFo; RR); Scheduling Policies
- verbesserte Speicherallozierung

Die im RT-Patch-Projekt[15] gewonnenen Erkenntnisse sind fortlaufend in die Entwicklung des Mainline Linux-Kernel seit der Version 2.6. mit eingeflossen.

Kernel 2.6.
- O(1) Scheduler; deterministischer Scheduler (Normal; FiFo; RR)
- Kernel Preemption
- Verbesserte Posix

2.6.18
- Prioritätsvererbung (Priority Inheritance) für Mutexe

2.6.21
- Hochauflösender Zeitgeber (HRT, High-resolution timers)

2.6.30
- Threaded interrupts, IRQ-Threads

2.6.33
- Spinlock annotations

Die OW-Backfireversion beruht auf der Kernelversion 2.6.32. Infolgedessen wurden neue Images inklusive libpthread, librt bzw. der Scheduling-Anwendung chrt[16] kompiliert und am Board getestet.

```
$ chrt -m

SCHED_FIFO min/max priority     : 1/99
SCHED_RR min/max priority       : 1/99
SCHED_OTHER min/max priority    : 0/0
```

14 http://www.kernel.org/pub/linux/kernel/projects/rt/
15 http://rt.wiki.kernel.org
16 http://linux.die.net/man/1/chrt

Mit *chrt* konnten laufenden bzw. zukünftigen Prozessen verschiedene Priortitäten von 0 bis 99 und drei Scheduling-Verfahren zugeteilt werden. Zum Ende des RBTemp-Projektes wurden verschiedenen Prozessen, als auch der Digitemp-Anwendung konkrete Prioritäten zugeteilt und das Schedulingverhalten beobachtet. Als Resultat entstanden etwaige weitere Anwendungsszenarien, wie in Punkt 5 näher beschrieben. Nebenzu wurden Thread gesteuerte Beispiel-Anwendungen compiliert und am Board getestet. Betreffend des HRT konnte die *nanosleep*-Funktion am Board nicht ausgeführt werden. Diese ist in der *uclib*-Variante nicht enthalten. Hierfür müssten Images mit der *glibc*-Variante kompiliert werden.

5. Messung

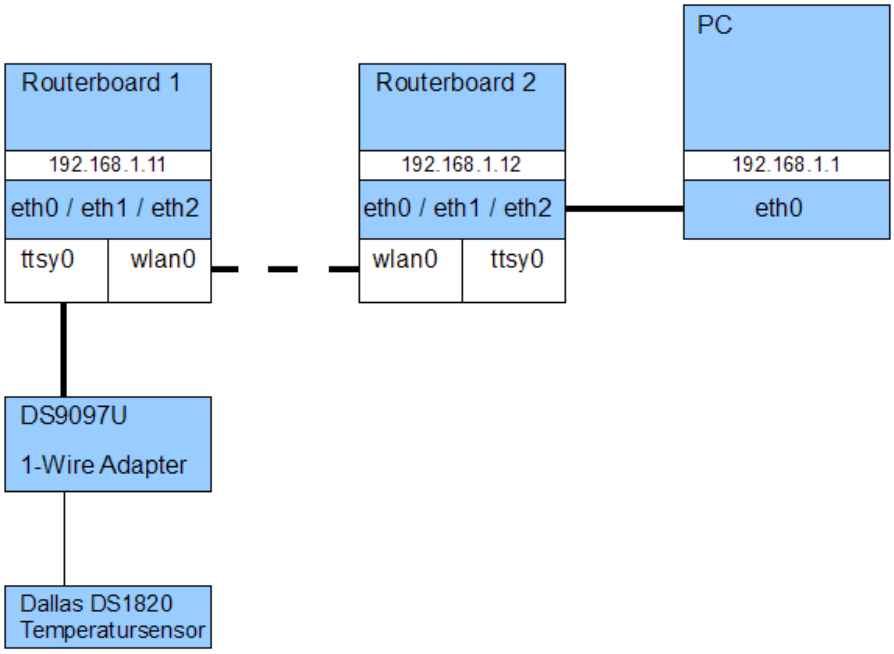


Abbildung 3: Messarchitektur

Abbildung 3 zeigt die verteilte Messarchitektur zur Temperaturmessung. Die RB's kommunizieren über Wlan im 5GHz-Frequenzbereich[17]. Das RB2 und ein PC mit Ssh-Client ist über ein LAN-Kabel verbunden. Am RB1 wird über die serielle Schnittstelle mittels des DS9097U 1-Wire Adapters der Temperatursensor abgegriffen. Das Messszenario ist in einzelnen Büroräumen und einem Hörsaal der Universität Salzburg angewendet worden.
Ein Test ergab, dass bei einer periodischen Messung die Digitemp-Anwendung nach ca. 8 Stunden aus nicht bekannten Gründen abricht. Für eine 24 Stunden-Messung[18] im Minutentakt wurde die Linux eigene Cronjob-Anwendung verwendet.

[17] Netzwerkeinstellungen: `RBTemp_Network_Config.txt`

[18] 24-Stunden Temperatur Messung: `RBTemp_24h_Messung.txt`

6. Fazit

Abschließend ist zu überlegen, welche Notwendigkeit bezüglich RT-Anforderungen in einer verteilten Temperaturmessung bestehen. Speziell in Szenarien einer Temperaturüberwachung von Systemen muss das RT-Verhalten des Systems konkret durchdacht sein. Sowohl das Zeit als auch die Priorisierung verschiedener Aufgaben, speziell für die Übertragung von Alarmwerten sollte stabil lauffähig sein.

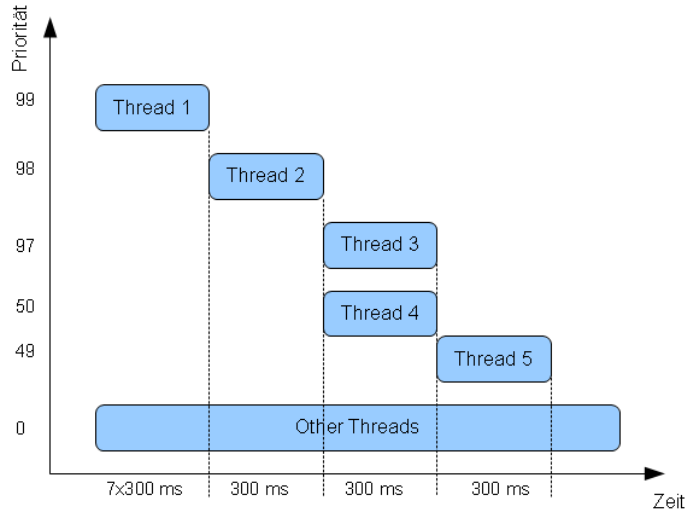


Abbildung 4: Prioritäts-Scheduling für Temperaturmessung

Thread 1: Liest Sensor 1,2,3, …
Thread 2: Werte vergleichen
Thread 3: Alarm zum Server senden
Thread 4: Schreibt Temperaturdaten in ein Textfile
Thread 5: Liest Temperatur aus dem Textfile (Server Anfrage)

Abbildung 4 zeigt einen möglichen Ablauf für eine priorisierte Temperaturmessung betreffend einer Fortsetzung für das RBTemp-Projekt. Erreicht man eine im Gegensatz wie in Punkt 4 beschrieben optimierte Antwortzeit von einer Sekunde für den Temperatursensor, so könnten verschiedene Situationen in periodischen Zeitschlitzen von 300 ms z.B. eine konstante 3 Sekunden Periode für zum Beispiel das Lesen, Vergleichen, Schreiben und Lesen der Daten vonstatten gehen. Mit einem 7 mal 300 ms langen Zeitschlitz inklusive Reserve sollte für die angegebene Responsetime von 1 Sekunde des Digitemp-Bausatzes das Auslangen gefunden werden.
Bezüglich einer Weiterentwicklung von RBTemp speziell für den RT-Einsatz, wird es notwendig sein, die frei verfügbare Digitemp-Anwendung und damit das 1-Wire Bus System auf seine Stabilität zu untersuchen. Hierbei stellt sich aber die Frage, ob diese kostengünstige Lösung generell für Real-Time- bzw. die gesamte RBTemp-Architektur (Hard-)Real-Time-tauglich sein kann?