(S)NTP: Implementierung auf einem AVR

Andreas Dittrich – dittrich@informatik.hu-berlin.de
Jon Kowal – kowal@informatik.hu-berlin.de

Abstract: Diese Arbeit beschäftigt sich mit dem Network Time Protocol (NTP) und der Implementierung dessen auf der AVR Plattform, welche durchaus sehr eingeschränkt genannt werden kann, was die Ausstattung mit Programmspeicher, Arbeitsspeicher und Rechenleistung betrifft.

Nach einem Teil, der das Network Time Protocol im Allgemeinen diskutiert, wird sich diese Arbeit mit den Problemen der Implementierung von NTP – und zu Teilen auch UDP/IP – auf einer Plattform auseinandersetzen, die kein Betriebssystem, dafür aber Stack Overflows und vieles andere zu bieten hat, mit dem man in heutigen Hochsprachen auf PCs sonst kaum noch zu tun hat.

1 Network Time Protocol (NTP)

1.1 Motivation

Für viele Anwendungen ist es von Bedeutung, dass verschiedene Computer eine gleiche, absolute Sicht auf die Zeit haben. Schon allein wenn mehrere Rechner auf einem zentralen Datenspeicher arbeiten, sollte eine synchrone Sicht auf die Zeit gegeben sein, Programme wie *make* kommen sonst durcheinander.

Es kann als unmöglich angenommen werden, eine beliebige Zahl von Computeruhren manuell synchron zu halten, weshalb ein automatisierte Mechanismus her musste.

Als man sich in den 80er Jahren hierüber Gedanken machte, gab es in den USA eine kleine Zahl genauer Uhren, die entweder selbst Atomuhr, oder per Funk oder Draht direkt von einer solchen synchronisiert wurden. Das damals noch junge Internet bzw. *ARPAnet* war ein ideales Medium Computer auch über große Entfernungen zu synchronisieren.

1.2 NTP Geschichte

1.2.1 Erster Vorschlag

Im September 1985 reichten *Mills* und *Linkabit* einen ersten Vorschlag für ein Network Time Protocol als Request for Comments (RFC), welches eher zaghaft mit folgendem Satz beginnt:

> ”This RFC suggests a proposed protocol for the ARPA-Internet community,

> and requests discussion and suggestions for improvements. Distribution of this memo is unlimited."

Das vorgeschlagene Protokoll legt ein Nachrichtenformat für Uhrensynchronisation fest, welches fast unverändert noch in aktuellen NTP Versionen Anwendung findet. Es ist ein symmetrischer Modus vorgesehen, der es verschiedenen Hosts erlaubt sich gegenseitig zu synchronisieren. Das Protokoll ist so angelegt, dass Verzögerungen durch die Übertragung heraus gerechnet werden können und Besonderheiten wie Schaltsekunden sind ebenfalls berücksichtigt. Dennoch wird nur die Übertragung von Daten spezifiziert, keine Algorithmen zur Uhrensynchronisation.

Grundlage dieses ersten NTP Protokolls ist ein 64 Bit Timestamp, dessen erste 32 Bit die Sekunden seit dem 1. Januar 1900 zählen und die zweiten 32 Bit die Nachkommastellen, auf die Protokollbestandteile werden wir aber später noch genauer eingehen.

1.2.2 Entwicklung der Spezifikation

Auf diesen NTP Vorschlag folgten weitere RFC, eine Übersicht mit Angabe der Seitenzahlen soll die rasche Entwicklung des Protokolls ausdrücken.

1985 Network Time Protocol, RFC 958 14 Seiten
1988 Network Time Protocol (Version 1), RFC 1059 58 Seiten
1989 Network Time Protocol (Version 2), RFC 1119 64 Seiten
1992 Network Time Protocol (Version 3), RFC 1305 120 Seiten

Der Umfang von NTP3 trieb die Entwickler dazu, ein vereinfachtes, aber dennoch zu NTP kompatibles Protokoll zu spezifizieren, das Simple Network Time Protocol (SNTP).

1992 Simple Network Time Protocol (Version 1), RFC 1361 10 Seiten
1995 Simple Network Time Protocol (Version 2), RFC 1769 14 Seiten
1996 SNTP (Version 4) for IPv4, IPv6, OSI, RFC 2030 18 Seiten

SNTP Version 4 wurde so genannt, um Konformität zur NTP Entwicklung anzudeuten, welche also auch schon mindestens seit 1996 bei Version 4 steht, nur wurde zu NTP Version 4 noch kein RFC herausgegeben. Aktuelle Implementierungen liegen jedoch längst in Version 4 vor.

1.3 NTP – Architektur

1.3.1 Komponenten eines NTP Netzes

Die NTP Architektur sieht einige wesentliche Komponenten eines Netzwerks vor, welches per NTP synchronisiert wird.

Primäre Zeitserver Diese sind im Besitz einer korrekten Zeit, welche sie über eine direkte Verbindung zu einer genauen Uhr erhalten. NTP sieht vor, dass eine Reihe solcher Server im Netz existieren, die breit verfügbare Ressourcen, wie Backbone Gateways, angeschlossen sind. Ziel von NTP ist es die Zeitinformation der primären Zeitserver im Netz zu verteilen und auch über Checks der Server untereinander Fehlinformation auszuschließen.

Sekundäre Zeitserver Lokale Hosts oder Gateways beziehen ihre Zeit von einem oder mehreren primären Zeitservern und stellen diese als Sekundäre Zeitserver bereit. Dies geschieht vornehmlich, um Ressourcen zu sparen und die Last des gesamten Internets von den primären Zeitservern fernzuhalten.

Sonstige lokalen Hosts Das sind â wie die Bezeichnung schon vermuten lässt â alle anderen Hosts im Internet, die ihre Uhr per NTP synchronisieren wollen. In der Regel sind diese Hosts Endknoten, die ihre Uhrzeit nicht weiter zur Verfügung stellen.

Um die Zuverlässigkeit des Systems zu erhöhen wird explizit vorgeschlagen, dass einige â mit günstigen, aber etwas ungenaueren Funkuhren ausgestattete â Server als Backup zur Verfügung stehen, sollten Primäre oder Sekundäre Server nicht erreichbar sein.

1.3.2 Definitionen/Nomenklatur

Für das weitere Verständnis der Bemühungen um das NTP Protokoll ist das Verständnis einiger Begriffe unabdingbar. Da diese in den RFC natürlich englisch benannt werden, werden wir auch hier die englischen Begriffe erklären, um Missverständnissen aus nicht eindeutigen Übersetzungen vorzubeugen.

stability Unter Stabilität (stability) einer Uhr versteht man die Konstanz der Frequenz.

accuracy Eine Uhr ist korrekt (accurate), wenn die Uhrzeit stimmt.

precision Die Genauigkeit (precision) beschreibt wie gut die obigen Werte eingehalten werden, sprich: wie korrekt ist die Uhr und wie stabil.

offset Die Zeitdifferenz bzw. der zeitliche Abstand zweier Uhren wird mit offset bezeichnet.

skew Die Frequenzdifferenz zweier Uhren wird skew genannt und kann auch als erste Ableitung des offset nach der Zeit verstanden werden.

drift Unter drift versteht man wiederum die Veränderung der skew über die Zeit, also die erste Ableitung von skew bzw. die zweite Ableitung des offset. In NTPv3 wird die drift aber als 0 angenommen.

1.3.3 Ergebnisse von NTP

NTP produziert 3 Ergebnisse, die genügen, um Uhren zu synchronisieren.

clock offset Die absolute Abweichung von der Referenzuhr. Das ist die Uhr zu der ein Host sich synchronisieren möchte. Weiß er das clock offset, kann der Host seine Uhr entsprechend vor oder nach stellen, um die lokale Zeit zu korrigieren. Später wird klar, dass für unsere schwachen Anforderungen an die Genauigkeit auf dem AVR der clock offset genügt, um die Uhr zu stellen.

roundtrip delay Dies wird benötigt, um eine Nachricht so zu senden, dass sie zu einer bestimmten Zeit bei der Referenzuhr ankommt. Es ist die Zeitdifferenz zwischen dem Zeitpunkt an dem ein Host eine Nachricht zur Referenzuhr schickt und dem Zeitpunkt an dem er die Antwort erhält.

dispersion Die maximale Abweichung von einer Referenzuhr. Dies ist wichtig möchte man wissen, wie ungenau eine Uhr maximal geht. Vor allem wenn Zeit über mehrere Hosts hinweg synchronisiert wird, möchte man noch wissen wie groß die maximale Abweichung zur ursprünglichen primären Uhr ist. Zum Beispiel kann auf Grund des Wissens über die dispersion ein Host sich für den einen oder anderen Zeitserver zur Synchronisierung entscheiden, je nachdem welcher die kleinste maximale Abweichung aufzuweisen hat.

1.3.4 Genauigkeit von NTP

Die Frage nach der möglichen Genauigkeit ist sicherlich eine, die die Architektur von NTP nicht direkt beantworten kann. Sie beschreibt aber einige Kriterien, die man beachten muss um je nach Präferenz eine mehr oder weniger genaue Uhr zu erhalten.

Es ist sehr schnell möglich, eine Uhr mit einer korrekten Zeit zu versehen, nur ist diese sicherlich nicht von Dauer, wenn beispielsweise skew nicht weiter betrachtet wird. Nur über einen längeren Stellzeitraum kann gewährleistet werden, dass ein Host seine eigene Ungenauigkeit beobachten und korrigieren kann.

Die Auflösung des Protokolls erlaubt mit seinen 32 Bit für Nachkommastellen immerhin eine Genauigkeit bis hin zu ca. 200 Picosekunden ().

1.3.5 Implementationsmodell

NTP sieht seit Version 1 zwei verschiedene Betriebsmodi vor, je nachdem welche Hosts sich darüber synchronisieren möchten. Für einen NTP Host kann es durchaus genügen nur

einen Modus zu implementieren.

Client/Server In diesem sehr einfachen Modus schickt ein Host (Client) eine Anfrage an einen Zeitserver, welcher daraufhin seine Zeit zurückschickt. Der Client hat kein Interesse und auch keine Möglichkeit den Server mit seiner Zeit zu aktualisieren. Die Behandlung der Client Anfragen auf Seite des Servers ist denkbar einfach, muss dieser im ankommenden Paket nur ein paar Felder ausfüllen und schickt es sodann gleich zurück an den Client.

Symmetrischer Betriebsmodus Im großen verteilten Netz des Internet erlangt NTP aber nur im symmetrischen Modus seine volle Stärke. Hier kann sich eine Menge von Hosts dynamisch organisieren, ohne fest vorgegebene Client/Server Strukturen zu kennen. Die Entscheidung wer wessen Uhr aktualisiert wird daran festgemacht, wer den âkürzestenâ Draht zu einem primären Zeitserver hat. Die Hosts organisieren sich in einer hierarchischen Struktur, auch Ausfälle einzelner Hosts werden vom Protokoll abgefangen.

Der symmetrische Betriebsmodus verlangt nach ausgefeilten Algorithmen für Assoziationsmanagement, Datenmanipulation und Uhrenkontrolle.

1.3.6 Netzwerkkonfiguration

Die hierarchische Anordnung verschiedener Hosts im Netzwerk geschieht über das so genannte Stratum. Das Stratum – das bedeutet Schicht – drückt im Falle von NTP die Entfernung eines Host vom Primären Zeitserver aus. Primäre Zeitserver haben das Stratum 1, alle mit ihnen synchronisierten Hosts das Stratum 2, usw.

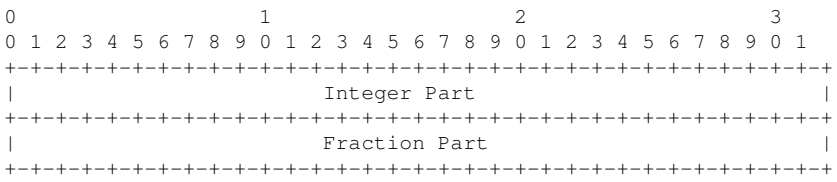
Das Netz über das sich die Hosts synchronisieren entsteht aus den Minimal Weight-Spanning Trees, deren Wurzeln die Primären Zeitserver sind. NTP sieht die Verwendung einer Variante des *Bellman-Ford-Algorithmus* vor, um diese Bäume zu finden. Metrik (für die Kantenwichtung) ist das Stratum plus die Synchronisationsentfernung, wobei letztere sich aus Dispersion und ein halb dem âabsolute delayâ zusammensetzt.

Das Ergebnis sind quasi selbst organisierte Master/Slave Beziehungen zwischen den Hosts im Subnetz, welche immer die genauest mögliche Zeit produzieren, selbst wenn einzelne Primäre oder Sekundäre Zeitserver bzw. die Netzwerkpfade dahin ausfallen sollten.

1.4 NTP – das Protokoll

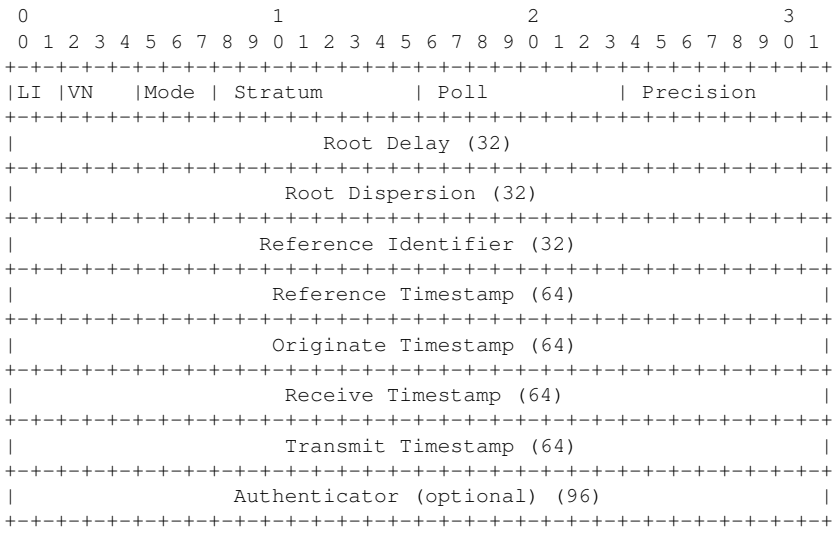
1.4.1 Datenformate

Alle Daten werden als Zweierkomplement im BigEndian Format übertragen. An dieser Stelle noch die schematische Darstellung des 64 Bit NTP Timestamps, der schon im Kapitel zur NTP Geschichte angesprochen wurde.



Der ganzzahlige Teil drückte ursprünglich die Sekunden seit dem 1. Januar 1900 aus, inzwischen wird aber dem Überlauf des Zahlenformats Rechnung getragen und so ist es auch erlaubt die Zeit ab 2036 zu zählen, wenn das erste Bit 0 ist. Ist das erste Bit 1, werden Sekunden ab 1968 angenommen. So kann der Überlauf des Timestamps, mit dem ca. alle 68 Jahre zu rechnen ist, sanft abgefangen werden.

1.4.2 Paketaufbau



1.4.3 Paketvariablen

LeapIndicator (LI) Zeigt Schaltsekunden an. Ist der LeapIndicator gesetzt, so weiß der Empfänger, dass eine Schaltsekunde in der letzten Minute des Tages hinzuzufügen ist (LI=01), oder eine Schaltsekunde abzuziehen ist (LI=10). Im letzteren Fall bedeutet dies, dass 23:58 die letzte Sekunde des Tages ist. Desweiteren wird dieses Feld auch als Statusfeld genutzt, und auf 11 gesetzt, um dem Synchronisationspartner mitzuteilen, dass die eigene Uhr nicht synchronisiert ist.

Version NTP Version, sollte bei aktuellen Implementationen 100 (=4) sein.

Mode Diese Variable legt den Modus fest, über den ein Host mit dem anderen kommunizieren möchte. So gibt es die Modi symmetrisch aktiv, symmetrisch passiv, Client, Server, Broadcast und ControlMessage.

Symmetrisch und Client/Server entsprechen den oben diskutieren Modi, wobei symmetrisch aktiv genau der Host ist, der sich synchronisieren möchte, während symmetrisch passiv der Empfänger einer Nachricht von einem symmetrisch passiven Host ist, allerdings nur wenn das Stratum des Empfängers kleiner ist als das des Senders. Broadcast ist ein zusätzlicher Modus, der es einem Server (vorzugsweise im LAN) erlaubt Clients per Broadcast zu synchronisieren, ohne dass diese selbst anfragen müssen.

ControlMessages sollen eine Möglichkeit bieten Managementfunktionen zu implementieren, welche SNMP o.ä. in Netzen ersetzen können, wenn dieses nicht vorhanden ist. Diese stellen aber strenggenommen ein eigenes Protokoll dar und sollen an dieser Stelle nicht weiter diskutiert werden.

Stratum In diesem Feld gibt ein Host sein Stratum an, bzw. 0, so er denn nichts über sein Stratum weiß.

Poll Interval Ein vorzeichenbehafteter 8 Bit Integerwert, der die maximale Zeit als Zweierpotenz angibt, die zwischen zwei erfolgreichen Synchronisationsnachrichten vergehen darf. Ist Poll Interval = 6 bedeutet dies also, dass 64 Sekunden zwischen den Nachrichten vergehen dürfen.

Precision Vorzeichenbehafteter 8 Bit Integer, der Uhr des Senders gerundet zur nächsten Zweierpotenz ausdrückt. Bsp. Precision = -5 impliziert eine Genauigkeit von 1/32s oder 31,25ms.

Root Delay Drückt das totale roundtrip delay zum Primären Zeitserver aus. Die Angabe der Zeit erfolgt über eine 32 Bit Festkommazahl, deren Nachkommastellen mit dem 16. Bit beginnen.

Root Dispersion Maximale Abweichung von der Uhr des Primären Zeitservers. Das Zeitformat ist ähnlich dem vom Root Delay, ausser dass für die Dispersion nur positive Zahlen erlaubt sind.

Reference Clock ID Gibt den Typ bzw. die Adresse der Uhr an, von der synchronisiert wurde. Typen können etwa *ATOM* (für Atomuhr), oder *TSP* (für Digital Time Service) sein. Hat die Referenzuhr ein Stratum von Zwei oder größer, wird die Internetadresse der Uhr angegeben.

Reference Timestamp Zeit zu der die lokale Uhr zuletzt synchronisiert wurde, bzw. Null, wenn diese noch nie synchronisiert wurde.

Originate Timestamp Zeit, zu der die letzte NTP Nachricht von der Gegenstelle gesendet wurde. Ein NTP Server setzt hier in seinen Antworten den Transmit Timestamp des Client ein.

Receive Timestamp Zeit, zu der die letzte NTP Nachricht von der Gegenstelle empfangen wurde.

Transmit Timestamp Zeit, zu der die Nachricht den Sender verlassen hat.

Authenticator Enthält Authentifizierungsinformation, so der Authentifizierungsmechanismus implementiert ist. Hierauf wird in diesem Dokument nicht weiter eingegangen, da dieser Mechanismus (für unsere Verhältnisse) sehr speicheraufwendig ist und auf einer so beschränkten Plattform wie dem AVR nicht unbedingt notwendig ist.

2 SNTP

SNTP – oder Simple Network Time Protokoll – ist ein Subset von NTP. Es wurde entwickelt, nachdem NTP Version 3 schon ziemlich komplex geworden war und sich für kompakte Geräte einfach nicht mehr eignete. SNTP beschränkt sich dabei hauptsächlich bei den Synchronisationsalgorithmen. Es ist ansonsten vollständig kompatibel zu NTP, was das Protokoll- und Paketformat angeht. Dies geht soweit, dass es für eine NTP-Host nicht erkennbar ist, ob der Kommunikationspartner SNTP anwendet und umgekehrt. Für uns war aufgrund der extremen Hardwarebeschränkungen der Plattform SNTP die erste Wahl. Zur Berechnung der Zeitdifferenz wird folgender Algorithmus angewandt:

$$d = ((T2 - T1) + (T3 - T4))/2 \tag{1}$$

T1: Originate Timestamp (Client)
T2: Receive Timestamp (Server)
T3: Transmit Timestamp (Server)
T4: Empfangszeit (Client)

Es wird also bei beiden Rechnungsteilen die Zeit des Clients von der des Servers abgezogen. So rechnet sich die Zeit für den Paketweg raus. Bei Annahme gleicher Dauer für den Pakethin- und Rückweg bleibt nur noch die Differenz der beiden Zeitskalen von Client und Server über. Da davon nicht immer auszugehen ist – in der Realität ist dies nie zu erreichen – wird die Rechnung mehrmals in kurzen Abständen durchgeführt

3 Hardware

3.1 Boardaufbau

Das Entwicklungs-Board wurde in Eigenregie angefertigt und befindet sich jetzt im Zustand eines stabilen Prototyps. Die Stromversorgung von konstant 5V wird über einen Spannungsregulierer mit davorgeschalteter Graetzbrücke gesichert. Ein Atmega32 steuert eine handelsübliche NE2000-kompatible Netzwerkkarte für den ISA-Bus. Neben einem parallelen und einem seriellen Interface ist noch ein 16Mhz Quartz vorhanden, der den Systemtakt vorgibt. Die beiden externen Interfaces sind Relikte aus frühen Entwicklungsstadien des Boards, mittlerweile wird ein JTAG-Adapter zum programmieren und debuggen benutzt. Das komplette Board ist von Hand gelötet. Dies führte im Laufe der Arbeit an diesem Projekt immer wieder zu nichtdeterministischem Fehlverhalten, welches erst durch mühsames Ausmessen der einzelnen Leitungen erroiert werden konnte. Ein nächster Schritt wird auf jeden Fall die Fixierung des Layouts und daraus dann das tzen eines Boards.

3.2 ATmega

Der Atmega32 stammt aus der AVR-Familie von *Atmel*. Dies ist eine gut skalierende 8-Bit RISC-Mikrocontroller Familie, welche recht preisgünstig für Endanwender zu erwerben ist. Er besitzt 32KB Programmspeicher und 2KB Arbeitsspeicher. Zur Kommunikation stehen ihm 32 programmierbare E/A-Kanäle zur Verfügung, mit acht davon steuern wir die ISA-Karte an, welche sich auch im 8-Bit-Modus betreiben lässt.

4 Programmierung

Der knappe Speicher gestaltete sich als das Hauptproblem bei der Entwicklung von Software, welche standardkonform im Internet agieren soll. Besonders der Arbeitsspeicher, in dem zur Laufzeit alle Variablen plus Stack untergebracht werden müssen, läuft schnell über. Versucht man dies durch intelligente Code-Anpassung zu umgehen wird irgendwann auch der Programmspeicher zu klein. So mussten bei der Programmierung viele Kompromisse eingegangen werden, um den zur Verfügung stehenden Speicher im Sinne der Standards optimal zu nutzen.

Da wir auf keinem Betriebssystem aufbauen, haben wir – abzüglich des dynamischen Stacks – den gesamten zur Verfügung stehenden Speicher unter unserer Kontrolle. Wir haben ein grosses, globales *Byte-Array* allokiert, in das ein komplettes Paket eingelesen werden kann. Die einzelnen Schichten greifen dann an vordefinierten Grenzen auf ihren Bereich, und nur auf diesen, zu. Kommunikation zwischen den Schichten geschieht über Zeiger auf die jeweiligen Bereiche. Somit sind die Schichten zwar physisch nicht getrennt, logisch jedoch bleibt die Struktur erhalten.

Die Kontrolle des Programmflusses findet in Ermangelung eines echten Schedulers fast vollständig durch Interrupts und daran angehängte Routinen statt. Hat die Karte ein Paket empfangen, bekommen wir das durch einen Interrupt mitgeteilt, ebenso ist die lokale Zeit mittels Interrupts realisiert: Ein spezielles Timer-Register wird alle 1024 Zyklen des Prozessors inkrementiert. Geschieht ein berlauf, so bekommen wir einen Timer-Interrupt; durch geschicktes Zurücksetzen dieses Registers lässt sich erreichen, daß das Register alle 4ms überläuft. Dies ist gleichzeitig die Auflösung unserer Uhr.

Die Programmierung gestaltet sich relativ komfortabel. Es existiert ein *avr-gcc* und damit eine *avr-libc*, in der fast alle Befehle des ATmega als C-Routinen enthalten sind. Es muss also kaum auf Assembler zurückgegriffen werden. Dennoch muss ausgepasst werden: Der Compiler hat aufgrund der vielen Interrupt-Routinen wenig Möglichkeiten, den echten Programmfluss zu erkennen – das Keyword *volatile* sei hier als vielbenutzte Hilfe erwähnt – und einige Optimierung erreichen im Endeffekt das Gegenteil bzw führen im schlimmsten Fall zu schwer zu findenden semantischen Fehlern.

5 Paketweg

5.1 Hinweg

Zunächst beginnen wir mit der Anfrage des Clients, unseres ATmegas. Der ATmega bootet, merkt, dass seine Zeit noch nicht aktualisiert wurde und schickt eine Anfrage an den ihm bekannten Zeitserver. Dessen Adresse wurde vorher vom DHCP-Server übergeben oder kann auch zur Übersetzungszeit festgelegt werden. Hier beginnt eine Schleife, da wir wie oben schon erwähnt nicht davon ausgehen können, daß die Paketwege immer genau gleich lang sind. Wir durchlaufen diese Schleife wie in der Standard-NTP-Implementation vier mal.

5.1.1 NTP

NTP geht folgendermassen vor: Es legt eine *struct* über den zugewiesenen Speicherbereich im globalen *Byte-Array*. Diese Struct enthält alle für ein gültiges NTP-Paket nötigen Felder. Es baut sich also innerhalb dieses Speicherbereichs den eigenen Teil zum Abschicken fertig zusammen. Eingetragen werden die Parameter der lokalen Uhr, welche vorab berechnet wurden und die momentane lokale Uhrzeit. Dann wird mit dem Aufruf von `UDP_Send()` die nächste Schicht beauftragt. Übergeben werden Absender- und Empfängeradresse.

5.1.2 UDP

UPD handelt ähnlich wie NTP, es legt die eigene *struct* über den UDP-Bereich im *Byte-Array* und füllt die nötigen Felder für einen korrekte UDP-Header. Anschliessend folgt noch eine Prüfsummenberechnung, bevor zum Schluss die nächstuntere Schicht – IP –

mittels `IP_Send()` aufgerufen wird.

5.1.3 IP

IP erhält von UDP Zeiger auf zu sendende Daten, ebenso auf die IP-Adressen, an die gesendet werden soll. Daraus erstellt es wie schon zuvor den eigenen IP-Header, auch hier muss die obligatorische Prüfsumme berechnet werden. Vor dem Abschicken prüft IP, ob die Empfängeradresse, also die des Zeitservers, schon im ARP-Puffer enthalten wird. Falls nicht – dies ist beim ersten NTP-Paket sehr wahrscheinlich – wird ARP beauftragt, eine MAC-Auflösung zu machen. Wichtig hierbei ist, dass der IP-Puffer während einer MAC-Auflösung gesperrt sein muss, da sonst ein ungültiger Header entstünde, falls während dem Warten ein weiteres IP-Paket einginge. Aus demselben Grund sind IP- und ARP-Header getrennte Einheiten. Schliesslich muss ARP ja trotzdem noch Antworten erhalten. Sobald IP eine MAC-Adresse bekommen hat wird die letzte Schicht aufgerufen: `Ethernet_Send()`.

5.1.4 Ethernet

Die Ethernetschicht geht grundsätzlich genauso vor wie die anderen Schichten. Interessant ist die Kommunikation mit der ISA-Karte. So muss diese zuerst initialisiert werden. Dafür wird in entsprechenden Registern auf der Karte der Schreibvorgang vorbereitet und die Länge des folgenden Pakets mitgeteilt. Danach werden die von IP übergebenen Puffer nach und nach in die Ausgangspuffer der Karte übertragen. Zuletzt wird mit einem Padding terminiert und über den Eintrag in ein Register der Sendevorgang auf der Karte angestossen.

5.2 Rückweg

Der Netzwerkverkehr auf unserem ATmega geht nun seinen gewohnten Gang – er steht also still – bis erneut ein Paket eintrifft. Dies kann ein beliebiges für andere implementierte Protokolle bestimmtes Paket sein. Schließlich ist unser Netzwerkstack komplett implementiert. Wir gehen hier nur auf das nächste für NTP wichtige Paket ein.

5.2.1 Ethernet

Der ATmega bekommt einen Interrupt von der Netzwerkarte. Darauf springt die Empfangsroutine in der Ethernetschicht an und liest über eine Schleife alle im Empfangspuffer der ISA-Karte liegenden Pakete aus. Schritt für Schritt werden diese dann von den Schichten weiterverarbeitet. Ethernet schaut im Header des Paketes nach, welchen Typs es ist, dies ist typischer Weise IP oder ARP und im Fall von NTP sicher IP, daraufhin wird `IP_Receive()` aufgerufen.

5.2.2 IP

Zuerst muss der IP-Puffer auf *block* gesetzt werden, damit bei einem Interrupt der Netzwerkkarte nicht das aktuelle von einem neuen IP-Paket überschrieben wird. IP liest dann den eigenen Header ein und prüft diesen auf Korrektheit. Stimmen die entscheidenden Felder nicht überein oder ist die Prüfsumme fehlerhaft wird das Paket kommentarlos verworfen, schließlich ist IP zustandslos. Im korrekten Fall wird das Paket an die darüber liegende Transportschicht übergeben, im Fall von NTP wird also `UDP_Receive()` aufgerufen.

5.2.3 UDP

UDP bekommt alle wichtigen Daten üebergeben, also die IP-Adressen von Quelle und Ziel sowie einen Zeiger auf den Rest des Paketes – den eigenen Puffer. Es legt wie auch schon die Schichten vorher also die eigene *struct* darüber und prüft die einzelnen Felder, abschließend noch die Prüfsumme. Es liest den Port aus und, falls ein Dienst für diesen Port implementiert ist – im Fall 123 (NTP) ist dies der Fall – wird die Payload an diesen Dienst weitergegeben, also `NTP_Receive()` aufgerufen.

5.2.4 NTP

NTP schließendlich liest den Rest des Paketes aus und berechnet nach der unter SNTP angegebenen Methode das Offset. Dieses wird auf die aktuelle lokale Zeit angerechnet. Sind unsere vier Schleifendurchgänge noch nicht komplett, wird nun der ganze Durchlauf erneut durchgeführt.

6 Abschließendes

Grundsätzlich gilt: Der Quelltext erklärt die Implementation wesentlich besser, als es im Rahmen dieses Arbeitspapiers möglich ist. Die von uns geschriebene Software ist unter der *GPL* veröffentlicht und auf dem Server `pallas.informatik.hu-berlin.de` für lokale Benutzer via *CVS* zugänglich. Der Name des Moduls ist `AVRWeb`. Interessierte ohne Zugang zu diesem *CVS*-Server können sich gerne per Email mit uns in Verbindung setzen.

Die Autoren freuen sich über konstruktive Anregungen und Kritik. Jede Weiterentwicklung der Software wird begrüßt, wir erbitten in diesem Fall lediglich eine Benachrichtigung über den Fortschritt. Für Erläuterungen und erweiterte Dokumentation stehen wir gerne und jederzeit zur Verfügung.

7 Quellen

- `RFC 1305`: Network Time Protocol (Version 3) Specification, Implementation
- `RFC 2030`: Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI