CHALMERS | GÖTEBORGS UNIVERSITET

Hummingbird
- Tweet till elektronisk dörrskylt

Kandidatarbete inom Data- och informationsteknik

ANDREAS ÅKESSON
ANTON SVENSSON
FREDRIK BROSSER
JAKOB KALLIN
KIM BURGESTRAND
LARS TIDSTAM

Institutionen för Data- och informationsteknik

CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET

Göteborg, Sverige 2012
Kandidatarbete/rapport nr 2012:43

Sammanfattning

Den här rapporten beskriver utvecklingen av en prototyp till ett system, Hummingbird, som låter användaren uppdatera en trådlös dörrskylt varifrån som helst, via mikrobloggtjänsten Twitter. Systemet har utvecklats som ett kandidatarbete vid Data- och IT-institutionen vid Chalmers Tekniska Högskola, Göteborg. Projektgruppen har bestått av sex teknikstuderande från Chalmers och Göteborgs Universitet.

Resultatet är ett fungerande system bestående av två delar: en dörrskylt med display samt en basstation som hämtar meddelanden från Twitter. Skylten och basstationen kommunicerar via en radiolänk.

Projektets huvudsakliga fokus ligger på energieffektivitet hos skylten. Även användarvänlighet och robusta kommunikationsprotokoll har prioriterats. Som en del i projektets fokus på energieffektivitet har tekniker för strömsnåla displayer, radiokommunikationsmoduler och microcontrollerplattformar undersökts och använts. Användarvänligheten ligger i skyltens trådlöshet, enkla konfigurering, samt dess fysiska dimensioner som möjliggör enkel montering på en dörr eller vägg. Slutligen ges systemet robusthet av pålitliga kommunikationsprotokoll för nätverk och radiolänk mellan basstation och skylt.

Abstract

This report aims to describe the prototype development process of a system, Hummingbird, designed to allow its user to update a wireless door sign from anywhere, using the popular microblogging service Twitter. The system has been developed as a Bachelor's Thesis project at Chalmers University of Technology. Behind the project are six technology students from Chalmers and Göteborgs Universitet.

The result of the project is a well functioning system, composed of two units: an electronic door sign with a display for displaying messages, and a base station tasked with fetching messages from Twitter. A radio link provides wireless communication between the two units.

The main focus points of the project have been to construct an energy efficient, simple-to-use and robust system. As a part of the project's energy efficiency goal, technologies for low power displays, radio communication devices and embedded computing platforms have been studied and used. The simplicity of the system is in the wireless design of the door sign unit, as well as its physical dimensions, which allow for easy wall or door mounting. Finally, robustness is implemented by the use of reliable communication protocols for network and radio communication between the base station and door sign unit.

Begreppslista

A/D-Omvandlare	Omvandlar en analog signal till digital.
API	Application Programming Interface
Arduino	Datorplattformsprojekt bestående av utvecklingsmiljö och hårdvara.
AVR	Microprocessorarkitektur som används av Arduinoplattformen.
Basstation	Sändare med internetåtkomst som skickar data trådlöst till skylten.
Coordinator	Basenheten i ett ZigBee-nätverk, koordinerar andra enheter.
DHCP	Protokoll för att automatiskt ge datorer deras nätverksinställningar.
Displaysträng	Den textsträng som skickas från basstationen till skylten.
End Device	Ändenhet i nätverket, förmedlar data uppåt i nätverkshierarkin.
Escape-tecken	Tecken som gör att efterföljande tecken tolkas annorlunda.
Ethernet	En samling metoder för nätverkskommunikation.
Hashtag	Etikett en användare kan sätta på tweets för att markera och organisera
I/O	Input/Output, in- eller utgångsanslutning på microcontroller.
IP	Internet Protocol. Protokoll för routing och addressering.
Kontrollsumma	Kontrolldata för överförda paket i syfte att upptäcka fel.
Lånetid	Tid som en dator fått låna en IP-address från en DHCP server.
MCU	Förkortning för microcontroller.
PAN	Personal Area Network, ett ZigBee-nätverk.
PAN-ID	Ett (helst unikt) Identifikationsnummer för ett PAN.
Radio-modul	XBee och den gränssnittskod som interagerar med XBee.
Radiolänken	Den trådlösa kommunikationslänken mellan basstation och skyltmodul.
SD	Secure Digital. Ett digitalt minneskortsformat.
Sekvensnumrering	Numrering av paket som skickas över nätverket.
Skylt	Display med trådlös mottagare som visar data skickad från basstationen.
SPI	Hårdvarugränssnitt för att kommunicera mellan MCU och periferienheter.
Systemet	Kombinationen av basstation och skylt, hela konstruktionen.
TCP	Transmission Control Protocol. Förbindelseorienterat dataprotokoll.
Tweet	Ett meddelande bestående av maximalt 140 tecken som skickats via Twitter.
UDP	User Datagram Protocol. Förbindelselöst dataprotokoll.
U.FL	Kompakt ytmonterad koaxialanslutning för antenner på mönsterkort.
XBee-AT	Application Transparent, simpelt point-to-point-läge för XBee.
XBee-API	avancerat operationsläge för XBee.
XBee-modul	Hårdvaran som ansvarar för sändning och mottagning av radiosignaler.
ZigBee	Det nätverksprotokoll som XBee bygger på och använder.

Tabell 1: Begreppslista

Innehåll

1 Inledning

1.1 Bakgrund och sammanhang

Den ursprungliga idén till projektet kommer från en artikel i IEEE Spectrum av Erico Guizzo (Guizzo, 2011), där författaren beskriver en elektronisk dörrskylt som visar upp meddelanden från hans Twitter-konto. Målet med projektet är att vidareutveckla samma idé, bland annat genom att göra den mindre otymplig och mer energieffektiv.

Syftet med dörrskylten i den ursprungliga artikeln var att kunna lämna meddelanden på sin kontorsdörr även om man inte är på plats, exempelvis för att man har valt att arbeta hemifrån. Den möjliggör även en centralisering av statusmeddelanden som användaren vill sprida till kollegor och andra i sitt arbete: de visas både på webben och på dörrskylten.

Dörrskylten är relevant för de som har ett väldigt rörligt arbete, oberäkneligt arbetsschema eller behov av att centralisera sin kommunikation genom att lägga ut den på nätet. Den har även ett egenvärde för teknikfantaster som vill ha de senaste intressanta prylarna.

Projektet skulle kunna relateras till den större diskussionen om sammanflätandet av verklighet med sociala medier, och om ett ökat beroende av dem är önskvärt. Det ligger även nära diskussioner om det papperslösa samhället, där all skriftlig kommunikation sker digitalt (även om man i detta fall vanligtvis inte ersätter papper utan whiteboard). Slutligen kan det skapa en rent teknisk diskussion kring hur ett sådant system kan skalas upp för att kostnadseffektivt kunna produceras för användning på en hel arbetsplats eller i andra sammanhang.

1.2 Problembeskrivning och syfte

Projektet syftar till att utveckla en prototyp av en elektronisk dörrskylt för uppvisning av meddelanden från Twitter, så kallade tweets (härefter meddelanden). Användaren ska enkelt kunna koppla skylten till sitt Twitter-konto. Användaren ska också kunna ange kriterier för hämtningen av meddelanden så att enbart vissa meddelanden visas på dörrskylten. Skylten ska vara helt sladd- och trådlös, samt strömsnål så att dess batteri sällan behöver laddas.

En viktig del av projektet är energieffektiviteten och strömsnålheten hos systemet. Detta undersöks i projektet ur ett tekniskt och ingenjörsmässigt perspektiv, men skulle också kunna diskuteras från en miljö- och samhällssynpunkt. Projektet syftar även till att undersöka vilka olika tekniska lösningar som finns för displayer, samt hur modern, energieffektiv displayteknik

kan användas för att konstruera energieffektiva system för informationsvisning.

Ett ytterligare projektmål är att systemet skall vara användarvänligt och lättanvänt, för att kunna användas i kontorsmiljö med minimal ansträngning från användaren.

Systemet ska bestå av två delar: en skylt och en basstation. Skylten ska trådlöst hämta sina meddelanden från basstationen och visa upp dem på sin display. Basstationen ska i sin tur kontinuerligt hämta meddelanden genom användarens vanliga internetuppkoppling. Skylten ska vara helt fristående från basstationen så att monteringen av den blir så enkel som möjlig och inte kräver några sladdar. Tyngdpunkten i energieffektivitetsmålet ligger på skylten, då den till skillnad från basstationen är batteridriven.

Systemet ska erbjuda användaren följande funktionalitet:

- Välja vilket Twitter-konto som skylten ska visa upp tweets från.
- Välja vilka tweets från det valda kontot som ska visas på skylten.
- Automatisk konfigurering av basstationens internetuppkoppling.
- Enkel ihoplänkning av basstation och skylt.
- Felindikation till användare i händelse av misslyckad kommunikation mellan skylt-basstation, basstation-Twitter, eller andra felscenarion.
- Statusindikatorer som visar systemets hälsa och eventuella fel.
- Monteringsmöjlighet för skylten på vägg eller dörr.
- Uppladdningsbart batteri för skylten.

Den hårdvara som utgör skylt och basstation ska baseras på en lämplig microcontrollerplattform. På skylten ska också finnas en fysisk knapp som gör att mottagaren direkt aktiveras och hämtar in det senaste meddelandet. Denna behövs för de gånger då det är viktigt att skylten uppdateras omedelbart.

Konfigureringen av systemet ska bestå av att ange namnet på det offentliga Twitter-konto vars meddelanden man vill visa upp, samt eventuellt de kriterier som meddelanden ska hämtas utifrån. Användaren ska sköta konfigureringen genom att med sin dator redigera en enkel textfil på ett micro-SD-kort, som sedan läses genom basstationens SD-kortsläsare.

1.3 Användare och användningsmiljö

Den användargrupp som projektets slutprodukt främst är tänkt för är kontorsanställda med ett behov av att kunna informera kollegor om eventuella möten, sjukdagar, schemaändringar eller liknande via en dörrskylt på kontorsdörren. Ett användarexempel är en stressad Chalmersstudent som letar efter sin (något impulsive) kandidatarbeteshandledare. Denne har tydligen bestämt sig för att jobba hemifrån för dagen, men som tur är informeras studenten via den twitteruppkopplade elektroniska dörrskylten på handledarens kontorsdörr. Situationen är räddad.

Den miljö som systemet är utformat för att verka i är den typiska moderna kontorsmiljön med flera olika typer av trådlösa nätverk i luften, vilket är en faktor att ta hänsyn till under utvecklingen av produkten. Vidare anses det rimligt att anta att skylten endast kommer att användas inomhus, med vanlig kontorsbelysning, samt endast under vanlig arbetstid. Detta är relevant då det tillåter eliminering av bakgrundsbelysning på skyltens display.

1.4 Begränsningar

Ett antal tekniska och produktionsrelaterade begränsningar har pålagts projeket. Syftet är att rama in och förtydliga uppgiften, samt att ställa upp realistiska mål, givet projektets resurser och tidstillgång. De huvudsakliga begränsningarna som är relevanta för utvecklingen är:

- Systemet är en prototyp, och behöver inte vara kostnadseffektivt för massproduktion. Istället kan systemet ses som ett test av ett nytt koncept. Resultatet, den färdiga prototypen, kan sedan anpassas för serieproduktion genom specialutformade mönsterkort och mekaniska komponenter (inbyggnadslådor och monteringsdetaljer), men detta ges ingen särskild uppmärksamhet under projektets gång.
- Systemet behöver inte ha stöd för att styra flera skyltar från en och samma basstation.
- Skylten behöver enbart ha stöd för ASCII-tecken samt de svenska bokstäverna Å, Ä och Ö.

2 Metod

2.1 Konstruktionsmoment

Under projektets inledande fas identifierades de huvudsakliga konstruktionsmomenten, samt gruppmedlemmarnas olika intresseområden. Eftersom projektidén och de olika momenten var ganska klart definierade tidigt i projektet, kunde en uppdelning göras redan under första projektveckan.

- Konstruktion av hårdvara, stödkretsar och mekanik:
  - Arduinoplattformar (färdiga kort)
  - Expansionskort till Arduino (färdiga kort)
  - Batteriladdning och batteristatusavläsning
  - Temperaturövervakning
  - Tryckknappar och statusindikatorer
  - Mellankopplingar och konverteringar, kablage
  - Avkoppling och kondensatorbanker
  - Inbyggnadslådor, mekaniska detaljer
- Hårdvarugränssnitt, mjukvara som kommunicerar med och styr hårdvaran:
  - Radiolänk
  - Läsning av SD-kort
  - Ethernet
  - Display
  - Hantering av fysiskt interface
  - Batteriavläsning
- Programlogik
  - Kommunikation med Twitter
  - DHCP
  - Parsing av konfigurationsfil
  - Parsing av Twitterdata
  - Formatering av Tweets
- Systemintegration
  - Kommunikation mellan mjukvarumoduler
  - Övergripande programstruktur

2.2 Arbetsuppdelning

Projektgruppen bestod av fyra datateknologer och två datavetare. En naturlig uppdelning, som även passade bra med gruppmedlemmarnas egna intresseområden, var att datavetarna blev huvudansvariga för den webbprogrammeringsrelaterade delen av arbetet, det vill säga hämtning, parsing och formattering av meddelanden. Mer hårdvarunära programmering delades upp på datateknologerna, som även skötte konstruktionsmoment relaterade till hårdvara och elektronikkonstruktion. En mer detaljerad beskrivning av arbetsuppdelningen inom gruppen ges i Appendix G.

2.3 Utvecklingsmetod

Det mesta arbetet utfördes i grupper om två, dock flexibelt och anpassat efter de uppsatta konstruktionsmomenten samt efter rådande projektstatus. Vissa uppgifter utfördes av enskilda gruppmedlemmar, där det ansågs lämpligt.

En viktig strategi som togs fram redan i början av projektplaneringen var att etablera en gemensam bild av systemet inom gruppen och definiera tydligt avgränsade konstruktionsmoment, samt att formulera en tidsplanering. Vidare lades mycket fokus på kommunikation mellan och inom (de vid tidpunkten aktuella) arbetsparen, och på kontinuerlig utvärdering av utfört arbete. Utvärderingarna relaterades till de gemensamt uppsatta målen och till tidsplaneringen.

Kommunikationen skedde genom ett antal olika kanaler; de viktigaste var Google Groups, Git (versionshanteringssystemet) och gruppmöten. Gruppmöten hölls varje vecka för att kunna samordna gruppen och ta större gemensamma beslut gällande till exempel komponentval och projektets övergripande riktning. Handledaren kunde följa arbetet via den loggbok som fördes under arbetets gång, men medverkade också vid varannat gruppmöte för kontinuerlig kontakt. Gruppens loggbok fungerade även som ett stöd under rapportskrivningen.

2.4 Versionshantering

Versionshantering av projektets programkod togs upp tidigt. Hantering av kod när det finns flera personer som jobbar på samma kodbas är komplex, inte minst när det sker ändringar i samma stycke kod samtidigt, som sedan behöver kombineras till det slutgiltiga resultatet. Situationer där kod fungerar som den ska den ena dagen men inte längre några veckor därefter är inte ovanliga. Den klart svåraste delen med att lösa generella buggar i programkod är att hitta felet, och då underlättar möjligheten att spåra just vilken ändring som införde buggen. Detta är något som ett bra versionshanteringssystem underlättar.

Versionshantering innebär ofta en central plats att lagra sin kod. För att underlätta sammanflätningen av jobb som utförts parallelt så spårar ett versionshanteringssystem alla ändringar i koden, och kan intelligent föreslå strategier för att slå samman det arbetet som utförts. Som programmerare är det också användbart att kunna se vilka ändringar som har gjorts under tidens gång av sina medarbetare, utan att själv behöva leta upp förändringarna och jämföra den existerande koden med sin egen.

Av ovan nämnda anledningar valde gruppen därför att använda sig av ett versionshanteringssystem. Det finns en mängd olika, bland annat CVS, SVN, Git, Bazaar, och Mercurial. SVN och CVS har använts väldigt länge inom industrin, och hör till de centraliserade versionshanteringssystemen. Git, Bazaar och Mercurial är något yngre (cirka 10 år), och hör till de decentraliserade versionshanteringsystemen. Då några i gruppen sedan tidigare har erfarenhet med SVN, och några i gruppen föredrar Git, stod valet mellan dem.

Jämfört SVN med Git så är SVN långsammare överlag, då alla operationer på projektbasen kräver att de kommunicerar via nätverk med den server där projektbasen ligger. I Git utförs dock majoriteten av alla kommandon enbart lokalt, vilket ger Git en stor hastighetsfördel över SVN (Git, 2012a). Att Git arbetar lokalt innebär också att alla i gruppen kan arbeta på kodbasen och versionshantera sin kod utan att ha tillgång till internet eller någon central server. Av bland annat ovan nämnda anledningar valdes Git som versionshanteringssystem för projektet.

2.4.1 Git och GitHub

Git är som tidigare nämnt ett decentraliserat versionshanteringssystem. Git används genom att först göra det arbete som är tänkt, och därefter spara det lokalt i sitt projekts kodbas tillsammans med ett meddelande som beskriver varför ändringen gjordes och vad den gör. Efter en viss tid ska ens egna arbete delas med sina medarbetare, och det görs genom att *pusha*, ladda upp, sin kod till en plats som alla medarbetare kan nå. Om någon inte har hunnit ladda upp upp sin kod är det sedan varje individuell persons uppgift att slå ihop ändringarna och bestämma vad som ska finnas i slutresultatet.

Som plats att spara koden på användes GitHub. GitHub är en tjänst som har funnits sedan 2008, och ämnar att göra programmering och kod till en mer social företeelse än vad som tidigare varit möjligt. Tjänsten är gratis, oavsett hur många projekt man har och hur stora de är, förutsatt att man delar med sig av sin kod till allmänheten. GitHub har vuxit explosionsartat sedan dess lansering, och har sedan maj 2012 över en och en halv miljon användare och över två och en halv miljoner publika projekt (GitHub, 2012a).

Under arbetets gång kunde alltså varje medlem i projektet publicera sina ändringar på gruppens projekt som finns tillgängligt på GitHub, där de andra medlemmarna i projektet sedan kunde inspektera ändringarna, kommentera på enstaka rader, rapportera buggar och även göra mindre ändringar i koden från webben i de fall man inte kunde klona projektet till sin dator och jobba lokalt.

3 Bakomliggande tekniker och tjänster

3.1 Twitter

Twitter är en hemsida som tillhandahåller gratis mikrobloggar där varje inlägg är högst 140 tecken långt. Twitter lanserades 2006 och har idag över 140 miljoner aktiva medlemmar världen över (Twitter, 2012c).

Twitters syfte är att låta användare förmedla korta meddelanden till antingen inbjudna vänner eller hela webben. Dess användningsområde sträcker sig från SMS-liknande kommunikation mellan individer till spridning av nyheter och politisk aktivism. Den fick ett stort medialt genomslag under 2010 och 2011, då många beskrev tjänsten som en bidragande faktor till den *Arabiska våren* (The National, 2011).

3.1.1 Terminologi

Twitter utgörs av ett stort antal användare. En användare identifieras av ett användarnamn som är högst 15 tecken långt.

En tweet (härefter *meddelande*) är ett inlägg på Twitter. Det tillhör en användare och består av upp till 140 Unicode-tecken. Detta meddelande kan innehålla så kallade hashtags, som ingår i meddelandets metadata. En användares samling av meddelanden kallas för tidslinje.

En hashtag är ett ord inuti ett meddelande som föregås av ett nummertecken (#, hash). En hashtag används för att kategorisera meddelandet, som sedan kan hittas genom en sökning på denna hashtag. En hashtag kan vara upp till 139 tecken lång och ett meddelande kan innehålla så många hashtags som ryms i dess 140 tecken.

En *retweet* är en kopia av ett existerande meddelande, återpublicerat av en annan användare. En retweet tillhör den andra användarens tidslinje. Den innehåller en kopia av det fullständiga originalmeddelandet och inleds med strängen *RT @user:* , där user är den ursprungliga författarens användarnamn. Detta innebär att en retweet kan innehålla upp till 161 tecken: 140 för det ursprungliga meddelandet, 15 för ett användarnamn av maximal längd och 6 för formateringen runt användarnamnet.

3.1.2 API

Twitter erbjuder ett HTTP-baserat API som kan användas av vem som helst för att hämta meddelanden, användaruppgifter och annan information från tjänstens databas (Twitter, 2012a). Autentisering krävs enbart för att

komma åt dold information (exempelvis privata meddelanden) och för att publicera nya meddelanden.

Den allmänna informationen i Twitters databas returneras som svar på HTTP-förfrågningar till URL:er som dokumenteras på Twitters hemsida. Svaren skickas i antingen XML- eller JSON-format, se avsnitt 5.4.

3.2 Arduino

Arduino är en microcontroller-plattform vars mönsterkortsdesign och källkod publiceras under öppen licens (Arduino, 2012a). Det finns ett antal varianter av plattformen, i olika storlek och komplexitetsgrad. Originalversionen av Arduino tillverkas av ett italienskt företag, men det finns många tredjepartsalternativ och varianter. Utöver grundplattformarna finns ett stort antal tillbehör och utbyggnadskort som lägger till funktionalitet.

På grund av sin enkelhet och flexibilitet är Arduino en populär plattform för små prototyper och hobbyprojekt, men den erbjuder också relativt avancerad funktionalitet. Då Arduino är utformad för att kunna användas av nybörjare inom microcontrollersystem är basfunktionaliteten relativt simpel, men en utvecklare kan med hjälp av utbyggnadskort (Arduino, 2012a) och kreativ programmering få ut tillräcklig prestanda för många icke-triviala tillämpningar. Dessa egenskaper är stora bidragande faktorer till att Arduino valdes som plattform för systemet, men även att det finns ett stort community på internet och många färdiga mjukvarubibliotek.

Andra alternativ som övervägdes i projektets planeringsfas var STM32-baserade utvecklingskort från ARM och Cerebot II från Digilent. STM32 erbjuder bättre prestanda än övriga alternativ, men är sämre lämpad i övrigt för projektet, med avseende på kostnad, smidighet och tiden det tar att komma igång med utvecklingen. Cerebot II har liknande specifikationer som Arduino, men det finns färre färdiga tillbehör av den typ som behövs för projektet, och skulle bli otympligare fysiskt. Utöver detta saknas till stora delar det community och de färdiga bibliotek som finns till Arduino.

3.2.1 Historia

Initiativet till Arduino togs vid den italienska högskolan *Interaction Design Institute Ivrea*. Det föddes ur det liknande projektet *Wiring* från samma högskola. Wiring bygger i sin tur på projektet *Processing* från Massachusetts Institute of Technology (Wiring, 2012).

Samtliga projekt har haft som utgångspunkt att bygga på källkod och hårdvara publicerad under öppen licens. Öppenheten hos hårdvaran gäller dock endast de mönsterkort och små stödkretsar som utformats inom projektet, och inte de övriga kommersiella kretsar som ingår, såsom microcontrollern från Atmel. Skillnader som kan nämnas mellan projekten är att Processing från början inte var kopplat till en specifik hårdvaruplattform, utan var mer fokuserat på programmeringsspråk och utvecklingsmiljö. Processing byggde på Java, snarare än på C/C++ som Wiringprojektet och senare Arduino (Processing, 2012).

3.2.2 Hårdvara

Arduino bygger på microcontrollers ur Atmels AVR-familj (de flesta varianterna av Arduino använder ATMega-serien). I sin grundform erbjuder Arduino en relativt omodifierad AVR-microcontroller och grundläggande kringelektronik för att stödja och programmera denna. Dessa kringkretsar består bland annat av en spänningsregulator, en oscillator och i vissa fall externt minne för att komplettera AVR:ens on-chip-minne (Arduino, 2012a). Dessutom finns grundläggande in- och utgångar på microcontrollern utdragna till fysiska anslutningar för att förenkla för användaren vid inkoppling av tillbehör eller övrig elektronik som ska användas i projektet.

De flesta Arduino-implementationer följer samma standard för de fysiska anslutningarna, för att vara kompatibla med varandra och tillåta smidig anslutning av utbyggnadskort. Dessa utbyggnadskort kallas i Arduinosammanhang för *Shields*. Många Arduino-varianter har även en USB-port för programmering och kommunikation med en PC.

3.2.3 Community

Eftersom Arduino anses användarvänlig och är en öppen plattform har den ett stort community av hobbyister och även mer avancerade användare på internet. Många bloggar och forum är dedikerade till elektronikprojekt genomförda med Arduino som utgångspunkt. Vidare finns ett stort antal mjukvarubibliotek tillgängliga, i många fall skrivna av community-medlemmar, och det finns goda möjligheter att få hjälp. Detta var ett av skälen till att Arduino valdes som plattform för arbetet.

3.2.4 Programmering

Arduino inkluderar en utvecklingsmiljö för den mjukvara som ska köras på hårdvaruplattformen. Programmeringsspråket som används är C/C++. Det finns vissa mindre modfikationer, främst i form av tillägg för att underlätta hårdvarunära eller elektronikrelaterade operationer, exempelvis att skriva

till en digital utgång. Detta sker genom färdiga funktioner och basbibliotek, samt viss modifiering av koden innan kompilering. Utvecklingsmiljön använder kompilatorn *avr-gcc* för att kompilera kod som ska köras på Arduinons AVR-microcontroller, och programmeraren kan utan problem använda sig av AVR-anpassad kod skriven i *ANSI-C* (Arduino, 2012a).

3.3 Energieffektiva Displaytekniker

3.3.1 E-Paper

E-Paper är en typ av displayteknologi som är utvecklad för att vara energieffektiv, och för att efterlikna en tryckt boktext. Tekniken använder ingen bakgrundsbelysning, utan fungerar genom att reflektera snarare än att sända ut ljus (Epaper Central, 2012). Detta gör att E-Paper inte kan läsas i mörker. Den egenskap som gör E-Paper attraktivt är i först hand att tekniken inte använder någon energi för att visa en statisk bild, utan endast för att byta tillstånd. Detta gör att E-paper kan behålla och visa en bild utan matningsspänning ansluten, vilket gör tekniken användbar i en del applikationsområden där energieffektivitet är högt prioriterat. Exempel som kan nämnas är busskurer, smarta kreditkort, E-boksläsare och elektroniska prisskyltar i affärer. I fallet med E-boksläsare är dessutom likheten med boktext en eftertraktad egenskap. Displayer som tillverkas med E-paper-teknik kan även göras böjbara och ytterst tunna.

De uppenbara fördelarna med E-paper är att energiförbrukningen för textuppvisningen minskar dramatiskt och att behovet av bakgrundsbelysning frångås. Utöver detta finns det andra, mer subjektiva, fördelar, såsom att läsupplevelsen kan uppfattas som mer naturlig och bekväm då E-paper liknar en tryckt bok utseendemässigt, och inte behöver uppdateras kontinuerligt.

En stor nackdel som vanligen nämns i samband med E-paper är att tekniken inte klarar av snabba uppdateringar, vilket omöjliggör mer avancerade menysystem och användargränssnitt. Dessutom är E-paper relativt dyrt i dagsläget, och de flesta kommersiellt tillgängliga displayerna med tekniken stödjer endast svart-vitt eller motsvarande. Färgdisplayer är dock på väg ut på marknaden (TechOn, 2009).

3.3.2 Bistabil LCD

Bistabil LCD är baserad, som vanlig LCD-teknik, på flytande kristaller. Kristallerna organiseras i lager ovanpå varandra, och deras riktning kan styras, vilket gör att man kan släppa genom eller reflektera inkommande ljus genom att positionera de olika lagren av flytande kristall i förhållande till varandra (Gu, 2006). Tekniken möjliggör att en display kan behålla en bild även utan matningsspänning och kräver ingen bakgrundsbelysning, även om många

tillverkare av bistabila LCD valt att bygga in energieffektiv sådan.

Bistabila LCD delar många fördelar och nackdelar med E-paper. En nackdel som båda teknikerna har är den relativt, jämfört med konventionell displayteknik, långsamma uppdateringshastigheten. Det rör sig dock om två nya tekniker som fortfarande utvecklas.

4 Systembeskrivning, Hårdvara

4.1 Moduluppdelning

Systemet är konstruerat med moduläritet i åtanke, både för mjuk- och hårdvara. På en övergripande nivå är systemet uppdelat i två delar: basstation och skylt. Basstationen och skylten är i sin tur uppbyggda av flera olika submoduler för att hantera olika funktioner.

Basstationen och skylten är båda baserade runt microcontrollerplattformar. Till dessa plattformar finns anslutna kringmoduler och kringkretsar som ger de önskade I/O-funktionerna. De båda systemdelarna kommunicerar trådlöst via en radiolänk och är utrustade med varsin radiotranceiver.

4.1.1 Skylt

Skylten är baserad runt en Arduino Pro. Plattformen ger basfunktionalitet i form av ett komplett microcontrollersystem med I/O, spänningsreglering och klockkrets. Till microcontrollern ansluts submoduler:

- Display för uppvisning av meddelanden och systeminformation
- XBee-modem för radiokommunikation med basstationen
- Interfacekort med tryckknappar och statusindikatorer
- Batteri och batteriavläsningskrets

Skylten utför inget avancerat logikarbete. Tyngdpunkten för skyltens design ligger istället på att så enkelt och strömsnålt som möjligt ta emot meddelanden och visa upp dem. För att göra systemet så energieffektivt som möjligt befinner sig skylten i ett strömspar- eller sovläge större delen av tiden. Skylten aktiveras var femte minut och skickar en förfrågan till basstationen om ny data. Intervallet fem minuter valdes som en bra kompromiss mellan energieffektivitet och snabba uppdateringar för användaren. En ny förfrågan kan också skickas genom att användaren trycker på en knapp på skylten.

Skyltens display har plats för 160 tecken: 140 tecken från själva meddelandet och 20 tecken ytterligare information i form av tid och datum. Tecknen fördelas över 8 rader med plats för 20 tecken vardera.

4.1.2 Basstation

Basstationen är konstruerad runt en större variant av Arduino, Arduino Mega 2560, på grund av dess extra RAM, vilket behövs för hantering av tweets. Basstationen är ansvarig för att hämta och formattera meddelanden. Efter att texten för ett meddelande har hämtats återstår följande justeringar:

- Tecken som inte stöds av skyltens display rensas bort eller ersätts.
- Texten ska avstavas så att för långa ord i slutet av en rad antingen delas upp eller flyttas ned på nästa rad.

Basstationen är alltid aktiv och lyssnar kontinuerligt efter inkommande förfrågningar. Internetuppkopplingen konfigureras automatiskt genom DHCP och använder sig av DNS för att kunna adressera Twitter i de HTTP-anrop som görs till dess API.

4.2 Arduinoplattformar

4.2.1 AVR och energieffektivitet

Arduino bygger på microcontrollers ur Atmels AVR-serie. AVR är relativt enkla att bygga med och att programmera, samtidigt som de erbjuder den prestanda och konfigurerbarhet som behövs i många projekt som baseras runt microcontrollers. Just i detta projekt är även energieffektiveten av stor vikt, och AVR erbjuder många möjligheter att spara energi. Detta sker främst genom att under körtid försätta hela eller delar av systemet i sömnläge. Exempelvis kan programmeraren samoptimera energiförbrukning och prestanda genom att sätta exakt klockfrekvens. Enligt databladet för AVR ATMega328 (som exempel) kan AVR ge en exekveringshastighet på nära 1 MIPS per MHz, vilket är mycket effektivt (Atmel, 2012a).

Som exempel på moduläriteten och energieffektiviteten hos AVR kan nämnas att systemutvecklaren har frihet att stänga ner oanvända delar av microcontrollern, såsom inbyggda A/D-omvandlare eller extraklockor (Atmel, 2012c).

4.2.2 Skylt

Arduino Pro är en nedskalad variant av Arduino, och har endast grundläggande funktionalitet och fysiska kontakter. Varianten är byggd för att klara batteridrift och för att ta liten plats i konstruktionen, egenskaper som passar bra in på den produkt arbetet syftar till att framställa. Som namnet antyder är Arduino Pro något mindre lättanvänd då den saknar USB-anslutning och kräver att användaren klarar av att bygga egna anslutningar. Den Arduino Pro som valts för arbetet bygger på en ATMega328, baserad

på 8-bit-arkitekturen AVR, är klockad till 8MHz och använder 3,3V (Arduino, 2012b). ATMega328 är utrustad med 32 kByte flashminne och kan maximalt stödja 23 I/O-pinnar (Atmel, 2012a).

4.2.3 Basstation

Arduino Mega är en större variant av Arduino, baserad på ATMega2560 som har mer on-chip-minne och fler I/O-pinnar än exempelvis ATMega328 som många andra Arduino-varianter bygger på (Arduino, 2012c). Vidare erbjuder Arduino Mega fler inbyggda hårdvaruserieportar och smidigare utdragningar av matningsspänning och jord till färdiga anslutningar. Plattformens dimensioner, minnesstorlek och många anslutningar gjorde den till ett bra val för arbetet. Basstationen är baserad på en Arduino Mega. Arduino Mega har 256 kByte flashminne, betydligt mer än de mindre kretsarna i samma familj, och stödjer upp till 86 I/O-pinnar (Atmel, 2012b).

Basstationen använder en Ethernet Shield för att ansluta till internet. Ethernet Shield är ett utbyggnadskort till Arduino som utökar dess funktionalitet genom att lägga till möjlighet till nätverksuppkoppling via 10/100 Mbit/s Ethernet med en vanlig RJ45-anslutning. Kortet drar matningsspänning från sitt värdkort. Kommunikationen med värdplattformen sker via SPI. På Ethernet Shield sitter en krets, WIZnet W5100 (Arduino, 2012e). Kretsen implementerar en TCP/IP-stack vilket innebär att detta inte behöver skötas i mjukvara (WIZnet Co. Ltd., 2012). Utöver RJ45-anslutning har Ethernet Shield även plats för ett micro-SD-kort som Arduino kan läsa och skriva till.

XBee Shield är ett utbyggnadskort för att lägga till funktionalitet för XBee-modem för användning tillsammans med Arduino. Utbyggnadskortet är relativt okomplicerat, och dess största bidrag är spänningsreglering och fysiskt smidig anslutning till resten av systemet. Basstationen använder en XBee-shield, medan skylten använder en mindre, nedskalad variant (Arduino, 2012d).

4.3 Radiolänk

En central del i projektet är den trådlösa kommunikationen mellan skylt och basstation, som möjliggör kringflyttande av skylten och gör den smidig att använda och montera utan sladdar. Energieffektiviten i systemet kopplar starkt till radiolänken, då trådlös kommunikation är en stor energiförbrukare i sammanhanget. Därför bör användningen av aktiv kommunikation hållas till ett minimum. Fokus läggs på enkel ihoplänkning av basstation och skylt, robusthet och dynamisk anpassning till rådande flora av radiosignaler i systemets omgivning. Radiolänken består av två delar:

- *Radiohårdvara och -sändare*: Trådlösa nätverksmoduler som kan kommunicera med microcontrollerplattformarna samt erbjuda sändning och mottagning av radiosignaler.
- *Nätverket*: Den mjukvara och det protokoll som bygger upp nätverket som använder hårdvaran.

4.3.1 Uppgift

Huvuduppgiften för det trådlösa nätverket är att överföra textdata över en radiolänk. Datan hämtas från Twitter och behandlas av basstationen, som sedan skickar den vidare trådlöst via radiolänken till skylten, som slutligen visar upp datan på displaymodulen.

Trådlösa nätvek är av naturen mer opålitliga än trådburna lösningar (Dell, 2012), av flera olika anledningar. Det är svårare att garantera och kontrollera att data har kommit fram, eftersom data kan förloras helt eller delvis över radiolänk (där luften utgör ett delat medium) lättare än i en kopparledning (eller liknande). Vidare finns det potentiellt många liknande trådlösa nätverk och radiolänkar i närheten som kan konkurrera om samma kanaler och nätverksidentifierare. Radiolänken behöver alltså vara både *robust* för att garantera tillförlitlig dataöverföring, och *dynamisk* för att upptäcka andra nätverk i närheten och anpassa sig för att kunna samexistera utan att orsaka kollisioner mellan paket som skickas över de olika närliggande nätverken.

Kravet på robusthet innebär att radiolänken måste implementera ett protokoll som stödjer felkontroll för skickade paket, såsom i form av sekvensnummrering, kontrollsummor och omsändning.

4.3.2 Nätverk

Dynamisk ihopkoppling och nätverksuppsättning är ett mål med radiolänken. Detta innebär praktiskt att basstationen söker genom tillgängliga sändningskanaler för att hitta en lämplig kanal, det vill säga med låg energinivå. Basstationen skapar ett nätverk genom att sätta ett nätverksidentifikationsnummer (PAN-ID), och sänder ut information om det nyformerade nätverket till skyltmoduler i närheten, samt öppnar nätverket för nya enheter att ansluta sig. Skyltmodulen behöver kunna söka efter basstationer i närheten, för att sedan gå med i en basstations nätverk om ett sådant finns tillgängligt, är öppet och matchar skyltens PAN-ID.

För den trådlösa kommunikationen valdes standarden ZigBee, eftersom den:

- Är en välkänd standard (bygger på IEEE 802.15.4), god dokumentation.

- Är billig och relativt enkel att implementera, vilket är lämpligt för mindre projekt. Den används av hobbyelektronikentusiaster, vilket ger värdefulla kunskapskällor i form av bloggar och tidningsartiklar, där liknande problem som projektet behandlar tas upp.
- Stödjer en nätverksstruktur som passar projektet, där det finns en basstation som koordinerar nätverket (Coordinator i ZigBee-terminologi) och en eller flera slutenheter (End Devices).
- Är utvecklad för att vara *energieffektiv*.
- Stödjer *tillförlitlig* överföring av data.
- Kan konfigureras så att radiomodulerna ger respons på mottagna kommandon.

(ZigBee, 2012), (Sensor Networks, 2010), (EE Times, 2010), (Embedded Computing, 2011)

ZigBee stödjer överföringshastigheter upp till 250 kbit/s och upp till 240 enheter i samma nätverk, vilket är mer än tillräckligt för projektet. Radiosignalerna använder det fria 2,4 GHz-bandet, som är uppdelat i 16 kanaler (ZigBee, 2012). Varje kanal motsvarar ett smalt band (5 MHz) i det frekvensspektrum som stöds. Inom varje kanal kan flera nätverk samexistera, dock under kravet att de har ett inom kanalen unikt PAN-ID. Detta presenterar inga större begränsningar för projektet, då varje kanal kan innehålla över 16000 unika PAN-ID:n (ZigBee, 2012). För större nätverk eller nätverk med höga prestandakrav är ZigBee dock inget rimligt alternativ.

ZigBee-nätverk kan innehålla tre olika typer av enheter: *Coordinator*, *router* och *end-device*. Ett nätverk måste innehålla en (och endast en) Coordinator, som fungerar liksom namnet antyder som koordinator och basstation för nätverket, med ansvar för att samla in data från och kontrollera övriga enheter (ZigBee Alliance, 2012). Coordinator-enheten kan starta nya nätverk och har kontroll över att tillåta nya enheter att ansluta sig. I systemet som beskrivs i denna rapport motsvaras Coordinator-enheten av basstationen. Coordinator-enheten måste alltid vara aktiv, och får alltså inte gå ner i sömn- eller strömsparlägen.

En router fungerar som en länk mellan Coordinator (eller en annan router) och en eller flera end-devices eller andra routers. Dessa är inte aktuella för användning i det här projektet, då det inte finns något behov för en sådan trädstruktur på nätverket.

End-devices är något enklare, och fungerar oftast som simpla insamlare eller mottagare av data. Eftersom end-devices inte har ansvar för att skicka vidare

data eller koordinera andra enheter, kan de för att spara energi med fördel försättas i strömsparläge under större delen av tiden (Digi, 2012a). Detta är idealt för skyltmodulen i systemet, då denna behöver ha lång batteritid. I det system som beskrivs här motsvarar skyltmodulen en end-device, och det finns som mest en end-device i nätverket, det vill säga att nätverket består endast av två enheter, en Coordinator och en end-device. På grund av den relativa enkelheten hos en end-device finns det goda möjligheter att utöka systemet med fler skyltar (end-devices), men det ligger utanför ramarna för detta projekt.

4.3.3 Hårdvara

ZigBee-standarden stöds av många olika hårdvaruplattformar och tillverkare (bland andra Atmel och Freescale), men efter undersökning av utbudet av hårdvara som stödjer standarden valdes XBee från tillverkaren *Digi*, eftersom XBee-modulerna jämfört med andra radiotransceivers i samma storlek:

- Är kompatibla med Arduinoplattformen och relativt billiga.
- Är relativt energieffektiva.
- Fungerar med en minimal uppkoppling och inte kräver mycket extra hårdvara.
- Erbjuder god räckvidd.
- Går att programmera om och är relativt lätta att styra via kommandon.

(Digi, 2012a,b)

Radiosändarna/mottagarna finns tillgängliga med fasta stavantenner (monopol), med chipantenner (integrerade på kretskorten) och med små koaxialanslutningar (U.FL) för extern antenn. Chipantenner utgör det smidigaste alternativet eftersom de tar lite plats och är ytmonterade direkt på kretskortet, men ger sämre signalräckvidd. Bäst räckvidd fås med förhållandevis stora, externa antenner som monteras via koaxial-anslutningarna (Digi, 2012c). Dessa är dock för otympliga för projektet. Av dessa skäl valdes varianten med stavantenner som en kompromiss, då dessa erbjuder bättre räckvidd än chipantennerna och samtidigt är smidigare än externa antenner.

XBee-modulerna har två operationslägen (xbee-arduino, 2011). AT-läget (Application Transparent) är ett enkelt läge för punkt-till-punkt-kommunikation, och kan ses som en trådlös ersättning för en seriellkabel (Digi, 2012a). AT-läget har fördelen att det är mycket enkelt att konfigurera och erbjuder en

simpel och snabb lösning för grundläggande radioöverföring. Nackdelarna med AT-läget är att mycket av den mer avancerade funktionaliteten går förlorad. Som nämnt ovan behövs robusthet i överföringen, och ihopparningen som den beskrivs ovan är över nivån för vad som är det tänkta användningsområdet för AT-läget.

Det andra läget som stödjs är API-läget (Application Programming Interface) och erbjuder den funktionalitet som saknas i AT-läget, med robusta överföringsmetoder och tillåter mer komplexa nätverksstrukturer. API-läget kräver mer konfiguration och mjukvarudetaljer, men tillåter implementering av de funktioner som eftersträvas inom projektet. API-läget valdes för att det:

- Erbjuder feedback på kommandon som har skickats till radiomodulerna.
- Har säkrare dataöverföring och mer kontrolldata (men alltså även mer overhead) genom att datapaket packas in i frames som sedan skickas över radiolänken.
- Ger nätverkskoordinatorn (här basstationen) möjlighet att dynamiskt söka av lediga kanaler och nätverksidentifierare, för att sätta upp nya nätverk utan att störa befintliga.
- Ger programmareren bättre kontroll över nätverket och den data som skickas.

(Digi, 2012a)

Radiomodulerna drar i aktivt läge, det vill säga vid sändning eller mottagning, i sammanhanget stora strömmar, 45-55mA (Digi, 2012a). Detta gör radiolänken till en stor energiförbrukningspunkt sett till hela systemet. Utrymme för energieffektivisering utgörs här främst av möjligheten att försätta radiosändarna (XBee-modulerna) i sömn- eller strömsparlägen när de inte aktivt används. Vidare ska radioöverföring användas så sparsamt som möjligt. Som förklaras nedan är strömsparfunktionerna i första hand aktuella på skyltmodulen, och det är också här de behövs som mest, i syfte att spara batteri.

Grundidén är att skyltmodulen är försatt i sömnläge större delen av tiden, och endast vaknar och är aktiv under korta perioder med regelbundna intervall, eller då användaren interagerar med det fysiska gränssnittet (knappsatsen). Då skyltmodulen vaknar, skickar den en förfrågan till basstationen om att hämta ny data, då sådan finns. Basstationen lyssnar konstant efter dataförfrågningar, utan att aktivt sända någon data utan att ha explicit

blivit tillfrågad.

Radiomodulerna har inbyggda funktioner för att söka av sin närmiljö efter andra nätverk. Sökningen fungerar genom att modulerna först söker igenom de tillgängliga frekvenskanalerna inom operationsbandet. Modulerna känner av och registrerar energinivåerna på de olika kanalerna, och på så sätt skapas en bild över vilken kanal där det finns minst traffik. En lämplig kanal väljs ut baserat på den insamlade informationen för att undvika kollisioner. Då en kanal valts sänds signaler ut på nätverket för att upptäcka Coordinators (modulerna ansvariga för de enskilda nätverken), som svarar med att sända sitt PAN-ID för att markera sin närvaro. En Coordinator anpassar sig till den insamlade information genom att välja att skapa ett nytt nätverk på en ledig kanal och PAN-ID-plats. Övriga nodtyper i nätverket (End-Devices eller Routers) kan med samma information göra ett informerat val om vilket nätverk som är mest lämpligt att ansluta till.

4.3.4 Implementering

Målet i projektet har varit att i så stor utsträckning som möjligt nyttja de funktioner som finns inbyggda i ZigBee-protokollet och de mekanismer som stöds av XBee-modulerna. Den funktionalitet som specificerats under projektets uppstart är: Pålitlig överföring av data (Tweets) till skyltmodulen, möjlighet till god energieffektivitet samt ett robust och dynamiskt sätt att forma nätverk och göra en ihopparning mellan skylt och basstation. ZigBee ger möjligheterna att implementera detta(EE Times, 2010), (Embedded Computing, 2011).

All data skickas över radiolänken som datapaket, snarare än som en enkel byteström. Datapaketen är uppbyggda av nyttodata, metadata samt överföringsdata. Överföringsdatan hjälper till med den rena överföringsbiten och består av kontrollsumma, sekvensnummer samt start- och stopptecken. Metadata ger information om nyttodatans egenskaper. Paketen är märkta med vilken typ av data det rör sig om. De olika pakettyperna är:

- Data: Inkommande eller utgående data till annan ZigBee-nod.
- Kommandosvar: Svar eller bekräftelse på kommando till lokal radiomodul.
- Leveransstatus: Bekräftelse på att ett utgående paket har skickats (eller inte).
- Modemstatus: Statussignal från lokal radiomodul.

(Digi, 2012b)

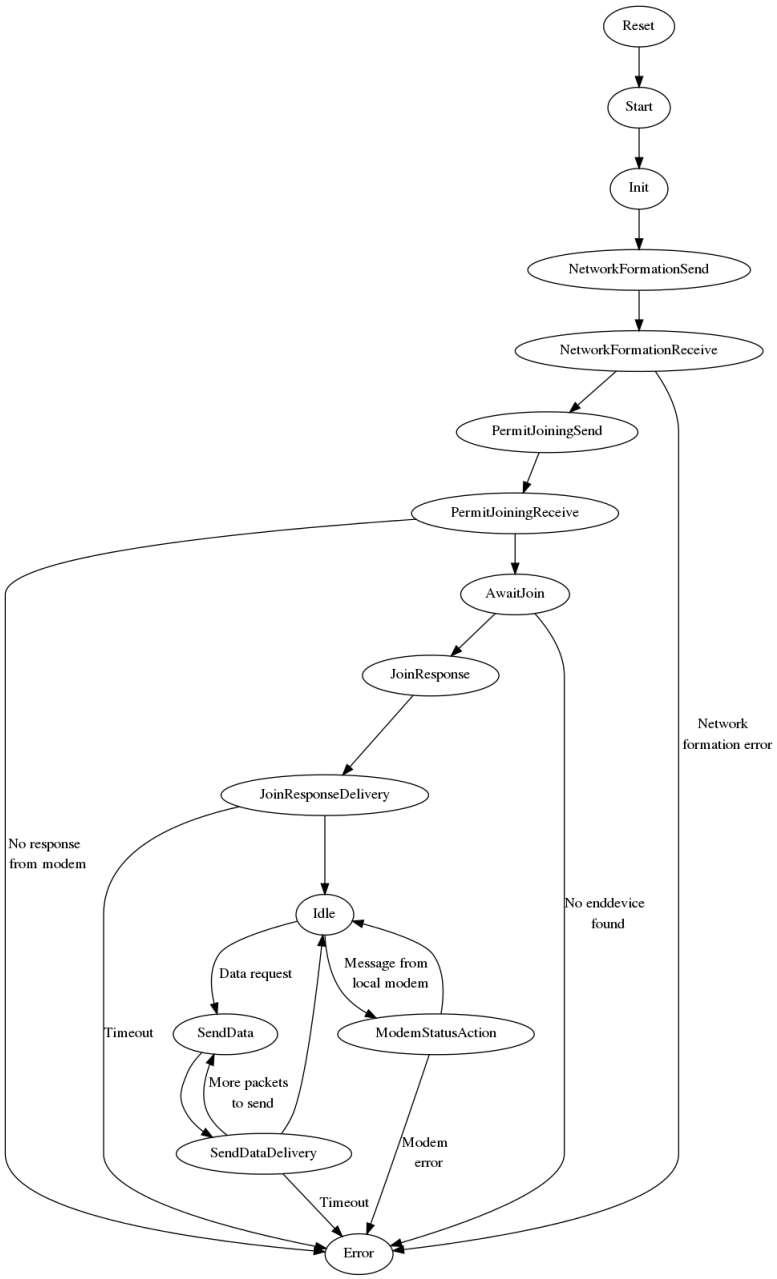
Nyttodatan är den data som ska överföras, exempelvis i det aktuella projektet textdata från basstation till skylt eller statuskoder från modemen för att signalera nätverksstatus.

Ett paket kan maximalt innehålla 72 bytes av nyttodata (Digi, 2012a), som motsvarar maximalt 72 tecken, vilket är mindre än antalet tecken ett meddelande kan innehålla. Detta gör det nödvändigt att dela upp längre meddelanden i flera olika paket, som sedan skickas numrerade över radiolänken. Ett meddelande kan maximalt delas upp i tre paket. För att möjliggöra smidig paketuppdelning används på basstationsidan en databuffert, som även underlättar för programlogiklagret och minimerar risken för felaktiga överföringar.

Ihopparningen av moduler sker genom att basstationen sätter upp ett nätverk, och sedan signalerar att den är redo för att end-devices ansluter sig till nätverket. Basstationen är ansvarig för att sätta upp reglerna för enheter som ansluter sig, och nya enheter tillåts endast ansluta under en begränsad tidsperiod. Skyltmodulen söker efter nätverk i närheten, och ansluter sig. Detta koordineras genom att användaren trycker på fysiska knappar på basstation och skylt för att starta ihopparningen.

Skyltmodulen ansluter sig till basstationen genom att skicka en simpel förfrågan. Basstationen svarar med ett svarsmeddelande. När basstationen har verifierat skyltens anslutning sparas dess adress för framtida kommunikation.

Arduinoplattformarna kommunicerar med sina respektive XBee-modem genom överföring av kommandon och XBee-modulernas svar på dessa kommandon. Överföringen sker på ett liknande sätt som dataöverföringen via radiolänken, men på ett mer uppstrukturerat och väldefinierat sätt, då det endast finns en begränsad uppsättning kommandon och svar. Då de ges vissa kommandon svarar XBee-modemen med data, och på vissa andra endast med ACK-meddelanden. Dessa kommandosvar ger arduinoplattformarna bättre kontroll över XBee-modemens exakta status, och hjälper även vid felsökning under utvecklingen.

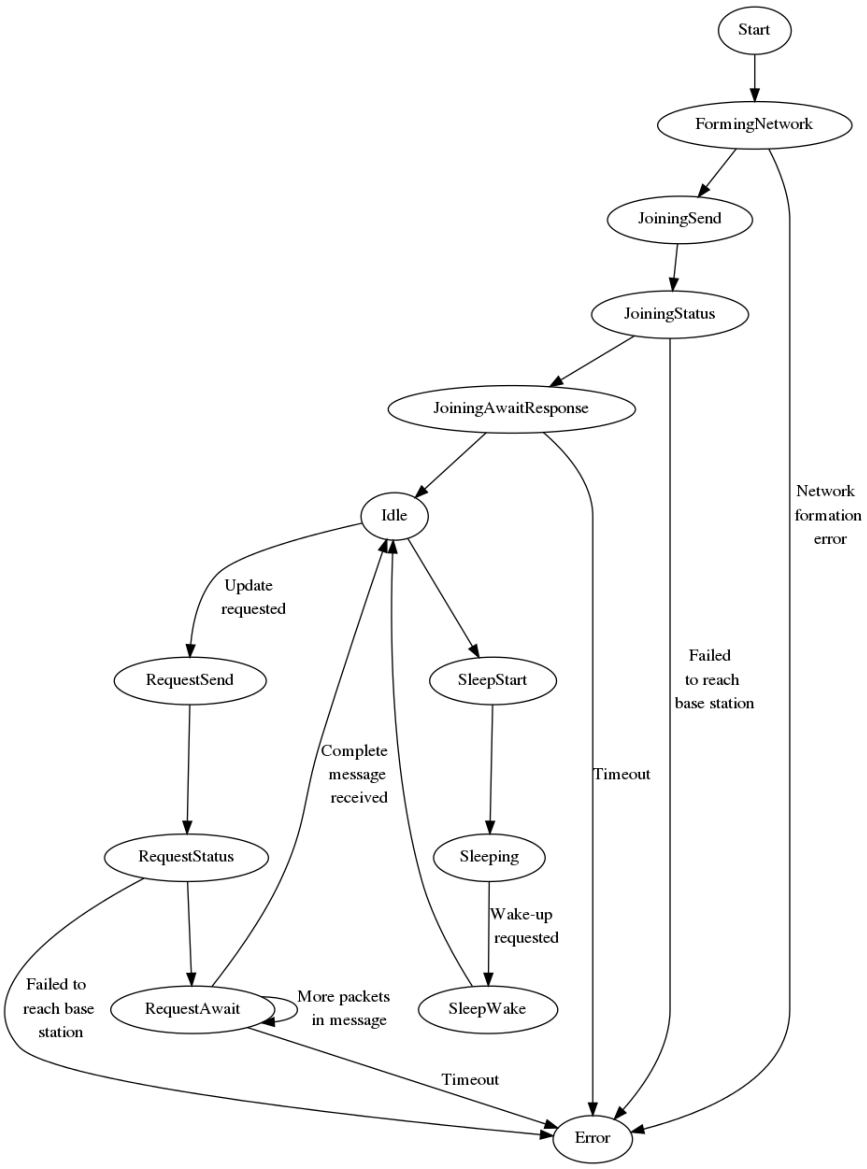


Figur 1: Flödesschema över radio i basstation

Radiolänk-delen i implementeringen av basstationen är internt uppbyggd runt en tillståndsmaskin, se figur 1. Syftet med detta är att ge programmeraren kontroll och översikt över exakt vilka kommandon och svar som förväntas, och att programmet endast kan befinna sig i ett av ett antal väldefinierade tillstånd. Basstationen börjar sin livscykel i ett reset-tillstånd,

och går vidare till att starta upp Coordinator-funktionen och nätverket, sedan lyssna efter anslutande skyltmoduler. Då en skyltmodul anslutit sig går basstationen in i ett passivt lyssningstillstånd. I detta läge väntar basstationen på inkommande statusmeddelanden från det lokala XBee-modemet eller på dataförfrågningar från skylten. Användaren kan även påverka basstationen genom det fysiska gränssnittet under lyssningstillståndet.

Då en dataförfrågan mottas, sänder basstationen över den aktuella textdatan via radiolänken och väntar på att skylten ska verifiera överföringen. Då överföringen verifierats återvänder basstationen till sitt passiva lyssningstillstånd. Om basstationen vid något tillfälle avviker från de väldefinierade tillstånden, får ett oväntat och ohanterbart svar eller måste vänta för länge på svar, går den in i ett felläge och signalerar detta till omvärlden. Felläget beskrivs i figur 1 som *Error*, och systemet stannar i felläget tills användaren manuellt trycker på systemomstarts-knappen.



Figur 2: Flödesschema över radio i skylt

Skyltmodulens radio är uppbyggd på ett liknande sätt som basstationen, och en beskrivning ges av figur 2. Skyltmodulen börjar vid omstart att initiera sin radio, och försöker sedan ansluta sig till en basstation. Sammankopplingen mellan skylt och basstation genomförs genom att skylten skickar en förfrågan om att gå med i nätverket, som sedan identifieras av basstationen. Om förfrågan lyckas sparar basstationen undan skyltmodulens hårdvaruadress för framtida kommunikation och skickar tillbaka ett svarsmeddelande. I det här läget har basstation och skylt anslutit sig till samma nätverk och

kan börja kommunicera. Då skylten är i vila är den försatt i ett strömsparläge, och radion är avstängd. Med jämna mellanrum kommer skylten att vakna upp, aktivera sin radiomodul och skicka en dataförfrågan till basstationen. Basstationen svarar med den senaste hämtade textdatan, som skylten sedan tar emot och visar upp på sin display. Detta är det normala operationsläget för skylten. Om skyltens tillståndsmaskin går utanför detta beteende kommer en felhanteringsmetod att anropas.

4.4 Display

För att kunna visa upp meddelanden på skyltmodulen krävs det att den har en display. Till detta har en ChLCD (Cholesteric Liquid Crystal Display) från Kent Displays valts. Det är en grafisk bistabil LCD som har en upplösning på 240x160 pixlar med en bildyta på 61x41 mm. Pixlarna på kanten runtom bildytan är större än de andra pixlarna för att enkelt kunna göra en dekorativ ram runt innehållet som visas på displayen (Kent, 2010).

Displayen skall kunna visa upp 160 tecken som beskrivits i avsnittet (Moduluppdelning - skylt), vilket ger utrymme att använda 240 pixlar per tecken. Med exempelvis tecken som är 12 pixlar breda och 20 pixlar höga kan man utnyttja hela displayen och det finns gott om pixlar till att visa upp tydliga tecken. Notera att tomt utrymme mellan tecken ingår i de 240 pixlarna.

Det är viktigt att tecknen är lätta att läsa även på avstånd eftersom typisk användning innebär att man kan vilja se om meddelandet uppdateras genom att kasta en snabb blick när man går förbi skyltmodulen, och den fasta monteringen innebär att det är omöjligt att få skyltmodulen i ögonhöjd för alla som kan tänkas vilja läsa från den.

4.4.1 Displayens energieffektivitet

För att maximera skyltens batteritid är viktigt att displayen är energisnål. Eftersom den valda displayen endast kräver ström då den uppdateras passar den bra med resten av skyltmodulen som spenderar större delen av tiden i ett lågenergiläge och inte kräver uppdateringar av displayen.

För att avgöra hur energisnål displayen är jämförs den med LCD-displayer av konventionell typ. Se tabell 2. Den alfanumeriska displayen är av en enklare typ som dock klarar av att visa lika många tecken. Den grafiska displayen är av liknande storlek och upplösning som Kent-displayen.

Strömförbrukningsvärdena har hämtats från respektive displays datablad (DISPLAY Elektronik, 2009), (DISPLAY Elektronik, 2011). Kent-displayens genomsnittsförbrukning togs fram med antagandet att en uppdatering tar

Modell	**Statisk**	**Uppdatering**	**Genomsnitt**
Kent	0	97	0,41
Alfanumerisk 160 tecken	1,75	1,75	1,75
Grafisk 240x128 pixlar	10,9	10,9	10,9

Tabell 2: Jämförelse av effektförbrukning i ett urval av displayer. [mW]

1,27 sekunder, vilket anges som en typisk uppdateringstid vid rumstemperatur i displayens datablad (Kent Displays Inc., 2010). I exemplet räknas med att uppdatering utförs var 5:e minut, den högsta uppdateringsfrekvensen som uppnås vid normalt användande av skylten.

Trots att den alfanumeriska displayen är av en enklare modell och under antagandet att displayen uppdateras så ofta som möjligt så kräver Kent-displayen en fjärdedel så mycket energi. Vid normalt användande med färre uppdateringar skulle Kent-displayen kräva ännu mindre energi medan de andra skulle ligga kvar på samma nivå.

Andra displayer baserade på bistabila tekniker var i åtanke under projektet, framförallt de av typen E-paper. Deras långa ledtider och dåliga tillgänglighet ledde dock till att de fick räknas bort.

4.4.2 Displayens uppbyggnad och användningssätt

Kent-displayen består utav en ChLCD-panel och en styrenhet. Styrenheten sköter uppdateringen av panelen och innehåller ett 32 kByte stort minne för att lagra data som kan visas på panelen. Displayen styrs genom att kommandon skickas till styrenheten via ett SPI gränssnitt. Ett flertal olika kommandon finns, men till de viktigaste hör att skriva till och läsa från minnet, uppdatera hela eller delar av displayen med data från minnet och att försätta displayen i ett lågenergiläge.

För att visa data på displayen måste först ett kommando utföras som överför datan till minnet, sedan ytterligare ett kommando som faktiskt visar datan på displayen. Datan överförs som minst en byte i taget till minnet i displayen. Då datan skall visas upp på panelen kan man välja att uppdatera 1 till 80 pixelrader eller hela panelen på en gång.

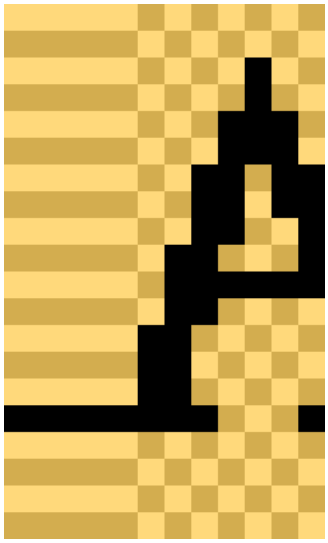
Med uppdateringskommandot anges vilken adress i minnet datan skall börja läsas ifrån. Den mest signifikanta biten på den byte som adressen pekar på utgör den första pixeln på den första raden som skall uppdateras. Nästa bit representerar nästa pixel, nästa byte representerar således 9:e till 16:e pixeln. De första 30 byten utgör hela den första raden. Om fler rader skall uppdateras utgör de följande 30 byten raden under. Detta mönster följs för

alla rader som skall uppdateras.

4.4.3 Hantering av tecken och teckensnitt

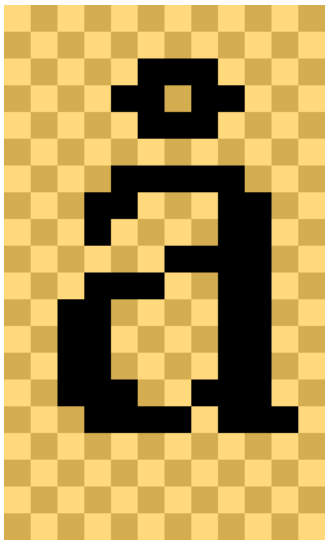
Displayen har inga inbyggda funktioner för att visa upp text, utan Arduinon i skyltmodulen måste översätta texten som skall visas upp på displayen till pixelrepresentation utav texten. Därför lagras ett typsnitt i programminnet på skyltmodulsarduinon där den ska slå upp tecken och få ut en pixelrepresentation av tecknet. Typsnittet som valdes består av tecken med samma storlek som i exemplet i början av detta delkapitel, 12x20 pixlar. Alla tecken har således samma storlek vilket gör dem lätta att hantera. Det är en modifierad version av ett 10x18 pixlars typsnitt från Linux-kärnan, som innehåller alla tecken som skall kunna visas upp på displayen enligt begränsningarna för projektet. För att anpassa typsnittet från 10x18 pixlar till 12x20 pixlar lades tomt utrymme höger och nedanför tecknet till.

Med den första anpassningen av typsnittet hade vissa tecken färgade pixlar ända ut till kanten på utrymmet avsett för tecknet. Om dessa tecken skrevs nära kanten på displayen överlappade tecknet de större rampixlarna och gav ett oönskat utseende. Se figur 3. Därför gjordes anpassningen om där tecken centrerades på deras teckenområde, vilket gör att de yttersta pixlarna på varje teckens teckenområde aldrig är aktiva. Således undviks problemet med att vissa tecken skrivs på ramen.



Figur 3: Del av ett A från tidig version av typsnittet: notera rampixlarna i vänstra sidan av figuren

Ett exempel från det slutgiltiga typsnittet kan ses i figur 4. Varje normalstor pixel på displayen är 0,26 mm bred och hög vilket ger ett teckenområde (hela orangea området) på 3,4 x 5,5 mm. För att sätta detta i perspektiv är tecknen på displayen ungefär lika stora som de i typsnittet *Times New Roman* i storleken 16 punkter.



Figur 4: Ett litet å från det slutgiltiga typsnittet

Översättningen från tecken till pixlar görs genom att varje rad i texten behandlas var för sig. Pixelrepresentationen av en textrad består av 20 pixelrader, eftersom alla tecken är 20 pixlar höga. Pixelraderna skapas och överförs en och en till displayens minne. En pixelrad byggs upp genom att textraden gås igenom och för varje tecken slås motsvarande pixelrad i tecknet upp för att sedan konkateneras med resterande teckens pixelrader.

Om till exempel en textrad består av tecknen AB så kommer första pixelraden att innehålla första raden i tecknet A följt av första raden av tecknet B. Nästa pixelrad består av andra pixelraden i A och andra pixelraden i B. Då samtliga 20 pixelrader har skickats iväg till displayens minne påbörjas arbetet med eventuella efterföljande textrader.

4.5 Övriga uppkopplingar

Systemet använder en uppsättning handbyggda kretsar för små uppgifter såsom användargränssnitt. Dessa kretsar är relativt simpla och befinner sig i konstruktionens utkant. Detta kapitel beskriver de stödkretsar som byggts inom ramarna för projektet.

4.5.1 Avkoppling och kondensatorbank

XBee-modulerna på basstationen och skylten matas med 5V och 3,3V, från sina respektive arduinoplattformar. För att buffra mot eventuella störningar på matningsspänningen används avkopplingskondensatorer av olika storlek, som rekommenderat i databladet för XBee-modulerna.

Skyltens display matas med 3,3V och jord från arduinoplattformen. Mellan display och arduino finns en kondensatorbank inkopplad, med en i sammanhanget stor elektrolytkondensator på 470μF, och två mindre, snabbare 10μF-kondensatorer.

4.5.2 Knappar och Statusindikatorer

De knappar som används är enkla tryckknappar med pull-up-motstånd och små kondensatorer. Då en knapp trycks ned kopplas signalen ned till jord. Innan signalen skickas vidare som insignal till målarduinoplattformen skickas den genom en 7414 Schmitt-trigger med inverterarverkan, vilket gör insignalen från knappen aktiv hög. Vidare används LEDs som statusindikatorer, kopplade till digitala utgångar på arduinoplattformarna. Samma princip för tryckknappar och statusindikatorer används på både basstation och skyltmodul.

4.5.3 Batteri

Batteriet som driver skylten är ett en-cells Litium-Polymerbatteri på 2000mAh, ger 3,7V och har inbyggt skydd mot överspänning ($> 4,25V$), för stort strömuttag och skydd för minimumspänning ($< 2,75V$).

Batteriet laddas via en kontakt på basstationen. Det kort som används för att ladda batteriet tar sin matningsspänning och jord från Arduinoplattformen (5V) och reglerar ner spänningen till lämplig nivå för att ladda Litium-Polymer-batteriet. Övriga komponenter på kortet är kondensatorer enligt rekommendationerna för spänningsregulatorn, samt resistanser som fungerar som strömbegränsande för lysdioder. Laddningsstatus visas på en LED.

4.5.4 Batterinivåavläsning

Batterinivån på skyltmodulen avläses genom att en analogingång på skyltens arduinoplattform kopplas till en spänningsdelare. Spänningsdelaren slås till och från via en NMOS-transistor, och microcontrollern läser av spänningen via en A/D-omvandlare.

4.5.5 Inbyggnadslådor och mekanik

Delsystemen (skylt och basstation) är inbygga i ABS-plastlådor där hål borrats ut för kontakter, knappar och statusindikatorer. Alla mekaniska delar är byggda för att kunna plockas isär och sättas ihop igen smidigt. M3-skruv har använts genomgående. En kylfläns har lagts till för att kyla spänningsregulatorn på basstationens microcontrollerplattform, då det upptäcktes att den blev varm vid systemdrift med nätadapter.

5 Systembeskrivning, programlogik

Med begreppet programlogik avses den del av systemets kod som behandlar problemdomänen. Problemdomänen är en modell av de uppgifter som systemet ska utföra, fri från hänsyn till implementationsdetaljer.

Vid uppstart av skylten går den in i en evig loop och skickar var femte minut en förfrågan till basstationen om vilket meddelande som ska visas upp. När ett svar har tagits emot visas meddelandet och dess metadata upp på displayen.

Vid uppstart av basstationen läses en konfiguration in för att ställa in vilka meddelanden som ska hämtas in från Twitter och vilken tidszon som datum ska visas i.

Efter att inställningarna har lästs in går basstationen in i en evig loop som utför två uppgifter. Dels hämtar den varje minut in, formaterar och lagrar det senaste meddelandet från Twitters API som går igenom användarens filter, och dels svarar den på förfrågningar om meddelanden från skylten genom att skicka det lagrade meddelandet till skylten.

Om något i programmet går fel så meddelas detta på två olika sätt, beroende på var felet uppstår. Om felet uppstår i basstationen så lyser en diod upp på basstationen. Om felet uppstår i skylten så lyser en diod upp på skylten och felmeddelande visas på displayen.

5.1 Konfigurering av systemet

En viktig funktion är att kunna konfigurera basstationen med vilket Twitterkonto den ska hämta meddelanden från, då det inte är speciellt användarvänligt att hårdkoda detta. Användaren av den elektroniska Twitter-skylten ska inte behöva editera i källkoden eftersom det blir onödigt omständigt samt en risk för kompileringsfel. Lösningen är en konfigurationsfil lagrat på ett micro-SD-kort där användaren kan ange vilket Twitter-konto som skall

användas samt en söksträng.

Systemet konfigureras vid uppstart av basstationen genom att läsa av det micro-SD-kort som sitter i basstationens Ethernet Shield. Minneskortet förväntas innehålla en textfil med namnet config.txt. Den första strängen i filen ska innehålla namnet på den Twitter-användare vars tidszon datumet ska visas i. Därefter följer en söksträng med vilken användaren kan specificera vilken meddelanden som ska visas på skylten. Om söksträngen saknas används istället en standardsöksträng. Användandet av söksträngar beskrivs i avdelningen Twitters API.

5.1.1 Inläsning

För att läsa från SD-kortet användes Arduinos standardbibliotek för SD-kort som i sin tur använder biblioteket *sdfatlib* (Arduino, 2012f). Konfigurationsfilens namn måste följa *8.3-konventionen* eftersom sdfatlib begränsas av det (Arduino, 2012f). Detta innebär att filnamnet får ha ett högst åtta tecken långt namn följt av en filändelse på maximalt tre tecken (Microsoft, 2012).

I konfigurationsfilen ska varje inställning separeras med mellanslag, tab eller nyradstecken. Först kommer användarnamnet därefter söksträngen. När filen har öppnats allokeras ett minnesutrymme som är lika många bytes som filen är stor. Sedan läses hela filen in i minnet med undantag av blanktecken före och efter användarnamnet. När inläsningen är klar är det känt hur många signifikanta tecken som är inlästa. Om det är färre inlästa tecken än antalet i filen så allokeras ett nytt minnesutrymme som endast har plats för adekvata tecken. Minnesutrymme som tidigare bestod av vittecken kan på så vis frigöras. Slutligen exponeras inställningarna för systemet genom metoder som returnerar dessa.

5.2 Förnyelse av IP-adress via DHCP

För att kommunicera över internet krävs det att basstationen har en IP-adress (Cisco Systems, 2003). Tilldelningen kommer att ske med hjälp av nätverkets DHCP-server, som har en samling med lediga adresser. För att DHCP-servern inte ska få slut på adresser så är det brukligt att en lånetid sätts, det vill sig hur länge enheten kan vara säker på att ha den givna adressen. Arduinos Ethernet-bibliotek (v1.0) har inte implementerat stöd för att uppdatera lånetiden av IP-adressen, vilket kan leda till att kontakten med internet förloras.

Då DHCP initeras försöker basstationen att få IP-adress av DHCP-servern genom att skicka DHCP Discover meddelanden (Cisco Systems, 2003). Om den inte får svar inom en minut så slutar den försöka. Om det däremot

kommer ett erbjudande om IP-address från servern så svarar basstationen på erbjudandet och accepterar adressen. Basstationen har då IP-adressen under den period som servern har bestämt.

5.2.1 Implementation

I enighet med DHCP ska förnyelseförsök av IP-adress ske vid halva lånetiden, $T1$, och vid sju åttondelar av lånetiden, $T2$, om det inte lyckades första gången. Första förfrågan sker endast till DHCP-servern som lånar ut IP-adressen till klienten, medan den andra förfrågningen går ut till alla på nätverket genom broadcast. En förfrågan ska gå till så att klienten skickar sitt meddelande med längre och längre intervall till dess att den får svar eller att 60 sekunder har passerat (Droms, 1997). Får klienten svar uppdateras dess lånetid och eventuella IP-adressändringar genomförs.

Hantering av lånetidsuppdatering kan implementeras med:

1. Timer som ger avbrott
2. En funktion som anropas för tidavläsning

Fördelen med alternativ 1 är att uppdateringen sker automatiskt när timern genererar ett avbrott. Nackdelarna är att timern i basstationens mikrocontroller slår över alltför ofta relativt till hur långa lånetider på IP-adresser det oftast är, vilket kräver en räknare för att utöka tidsrymden. Dessutom skulle det behövas totalt tre timerar för $T1$, $T2$ och lånetiden.

I alternativ 2 används *millis*, en funktion som finns i Arduinomiljön och som räknar antalet millisekunder modulo 2^{32} sedan enheten startades och kan läsas av samtidigt som tiden fortskrider. Anrop till millis används för att jämföra aktuell tid med tiden då IP-adressen blev lånad för att avgöra om lånetiden behöver uppdateras. Funktionaliteten implementeras i en funktion som ska anropas med jämna mellanrum. Fördelen med alternativ 2 är att med enkla jämförelser kan det avgöras när förnyelse ska ske. Nackdelen är att tiden måste läsas av varje varv i programloopen, att det inte kan styras exakt när kontrollen av lånetiden görs och det kan hända att IP-adressen används efter lånetiden löpt ut.

Den slutliga lösningen blev alternativ 2, eftersom programloopen ska köras ofta och att millis redan är implementerad. Vidare valdes alternativ 1 bort eftersom ATmega2560 endast har fyra 16-bitars timers vilka kan behövas i andra delar av programmet. Implementationen bryter mot DHCP specifikationen på några punkter:

- Om lånetiden är mindre än åtta minuter blir det problem om förnyelse misslyckas eftersom den efterföljande förfrågan kan ta upp till 60

sekunder. Under den tiden kan lånetiden redan passerat och basstationen använder IP-adressen felaktigt.

- Då millis slår över går det inte att kontrollera om lånetiden är slut. I det fallet görs ett försök att låna om IP-adressen omedelbart även om inte T1 har passerats.

Eftersom basstationen kommer vara ansluten till ett trådbundet nätverk kommer första punkten inte vara något problem, eftersom den minsta lånetiden i praktiken är 15 minuter (Windows Enterprise Networking, 2007). Andra punkten är endast en rekommendation som bryts. Detta görs eftersom det blev alltför besvärligt att hantera millis överslag, som sker en gång ungefär vart 50:e dygn. Den effektivaste lösningen är således att begära förnyelse av IP-adressen vid överslag på millis.

5.3 HTTP

För att hämta information från Twitter använder sig systemet av HTTP, som är det kommunikationsprotokoll som ligger till grund för webben. HTTP används för att skicka och ta emot resurser utifrån klient/server-modellen: En klient skickar en förfrågan till en server, som skickar tillbaka ett svar. En resurs är ett abstrakt begrepp som i IETF:s specifikation definieras som något med en identitet. De resurser som systemet efterfrågar från Twitter är användaruppgifter samt meddelanden och deras metadata.

HTTP är textbaserat och kommunicerar över TCP. Både klienter och servrar använder sig av valfria portnummer, men för servrar är det vanligtvis port 80 (Fielding et al., 1999).

5.3.1 Förfrågan

En HTTP-förfrågan består av fem delar:

1. En metod, som anger hur resursen ska behandlas.
2. En URL, som anger vilken resurs som ska behandlas.
3. Ett versionsnummer, som anger vilken version av HTTP som ska användas. (Systemet använder version 1.1.)
4. En lista med headers, som anger diverse alternativ och metadata.
5. En meddelandekropp, som anger eventuell data som ska skickas till servern.

HTTP har stöd för ett antal olika metoder, däribland GET, POST, PUT och DELETE. I systemet används enbart förfrågningar med GET-metoden, som begär att resursens innehåll skickas tillbaka som svar.

5.3.2 Svar

Ett HTTP-svar liknar en HTTP-förfrågan och består av fyra delar:

1. Ett versionsnummer.
2. En statuskod följt av en statusbeskrivning.
3. En lista med headers.
4. En meddelandekropp.

5.4 JSON

JSON är ett textbaserat format för överföring av data. Det är ett av två format som används i svaren på förfrågningar till Twitters API; det andra är XML. Behandling av JSON kräver generellt mindre processorkraft och mindre minnesutrymme än behandling av XML (Nurseitov et al., 2009). Av denna anledning tar systemet emot data från Twitters API i JSON-format.

5.4.1 Syntax och datatyper

JSON tillåter värden av följande datatyper:

- Nummer
- Sträng
- Booleanvärde
- Fält
- Objekt
- Null

Av dessa datatyper är fält och objekt sammansatta: De innehåller andra värden av valfri typ. I fält är dessa värden indexerade med heltal i stigande ordning från noll. I objekt är dessa värden indexerade med strängar som är unika inom det objektet (nycklar).

En JSON-fil består av antingen ett objekt eller ett fält, som i sin tur kan innehålla ytterligare värden enligt reglerna för respektive datatyp.

5.4.2 Avläsning och intolkning

Vanligtvis tolkas JSON genom att läsa av hela filen och lagra den i en datastruktur i minnet som tillåter godtycklig navigering. Problemet med parsingen när den ska utföras på en Arduinoplattform är att mängden RAM-minne är begränsad till 8 kByte. Risken är således stor att ett fullständigt inläst JSON-svar från Twitters API tillsammans med övrig programdata tar slut på systemets minne.

För att lösa detta problem används en *händelsebaserad strömparser*. Till skillnad från en konventionell parser lagrar en händelsebaserad strömparser inte JSON-datan i minnet. Istället signalerar parsern händelser varje gång den påträffar ett värde medan den läser av JSON-filen. Dessa händelser hanteras sedan av en uppsättning olika funktioner som anropas av parsern genom funktionspekare. Detta tillvägagångssätt innebär att minnesåtgången är minimal eftersom enbart det värde som precis har lästs av behöver sparas i minnet.

Den parser som används i systemet heter Yajl. Den signalerar en händelse varje gång som den påträffar något av följande:

- Nullvärde
- Booleanvärde
- Heltal
- Decimaltal
- Nummervärde
- Sträng
- Början på objekt
- Objektnyckel
- Slut på objekt
- Början på fält
- Slut på fält

Dessa motsvarar en händelse för varje datatyp med följande undantag:

- Händelsen för nummervärden anropas för både heltal och decimaltal.

- Ett värde av en sammansatt datatyp ger upphov till flera händelser: en för dess början, en för dess slut samt en för varje nyckel som ingår i det (om det är ett objekt). Varje värde som ingår i objektet eller fältet ger dessutom upphov till händelser som vanligt utifrån dess datatyp.

Ett av problemen med en händelsebaserad parser som Yajl är att dess sekventiella inläsning gör det svårare att deklarativt hämta in data från JSON-filen genom att exempelvis ange en sökväg. Yajl har visserligen inbyggt stöd för att hämta in data genom att ange en sökväg (*simplified tree interface*), men detta kräver att hela JSON-filen läses in i minnet på samma gång och fungerar enbart med värden inuti objekt, inte värden inuti fält. Av denna anledning behöver inhämtningen av data implementeras på imperativ väg genom de funktionspekare som anropas vid olika händelser.

5.4.3 Algoritm

Den inhämtning av data som används i systemet är baserad på en väldigt enkel princip: En lista med objektnycklar matas in och för varje objektnyckel sparas det första värde som förknippas med den objektnyckeln, oavsett var i JSON-filen objektnyckeln befinner sig.

Avläsningen av JSON-filen inleds med följande algoritm:

1. En tabell definieras med objektnycklar och variabler, där varje objektnyckel förknippas med en variabel.
2. Parsern läser av JSON-filen och signalerar en händelse så fort någon av objektnycklarna i listan påträffas.

Så fort en händelse signaleras utförs denna algoritm:

1. Nästa värde som läses in av parsern sparas i den variabel som förknippas med den nyligen inlästa objektnyckeln.
2. Den nyligen inlästa objektnyckeln och variabeln som den är förknippad med tas bort från tabellen.

Algoritmen skulle kunna implementeras så generellt som den är beskriven ovan, men eftersom systemet hämtar in så pass få olika värden (tre stycken) är den istället implementerad genom individuell hantering av varje enskilt fall; slutresultatet är samma.

Denna metod har två begränsningar:

1. Endast ett enda värde kan läsas in för varje objektnyckel som anges, så det är omöjligt att läsa in separata värden som befinner sig i olika delar av JSON-filen men har samma objektnyckel.

2. Endast det första värdet som förknippas med en angiven objektnyckel kan läsas in, så det är omöjligt att komma åt värden vars objektnycklar har förekommit tidigare i JSON-filen.

Dessa begränsningar är dock inga problem i systemet eftersom inga av de objektnycklar som behöver läsas in förekommer mer än en gång i något av de svar som tas emot från Twitters API. Eftersom Twitters API dessutom är versionsangivet bör risken vara minimal att beteendet förändras i framtiden, och om en godtycklig förändring mot förmodan sker så skulle också alla andra algoritmer för avläsning potentiellt kunna misslyckas.

5.5 Twitters API

Twitters API består av ett antal URL:er som kan efterfrågas med HTTP för att komma åt stora delar av Twitters innehåll. Systemet använder sig av detta API för att komma åt två olika delar av Twitters innehåll: användaruppgifter och meddelanden.

5.5.1 Användaruppgifter

Varje meddelande ska visas på skylten tillsammans med datum och tid, men samtliga meddelanden som skickas från Twitters API har datum och tid angivet i UTC (GMT-tidszonen). För att visa rätt datum och tid måste systemet komma åt användarens tidszon. Denna finns tillgänglig i form av en UTC-offset i användarens offentliga uppgifter. Dessa uppgifter skickas som svar på förfrågningar till följande URL, där namn är namnet på den användare vars meddelanden ska visas: `api.twitter.com/1/users/show.json?screen_name=namn`. Användarnamnet erhålls av systemet genom konfigurationsfilen.

Svaret innehåller ett objekt med flera olika värden. Användarens UTC-offset har nyckeln *utc_offset* och representeras som ett heltal med tecken där värdet anger skillnaden i sekunder mellan användarens tidszon och UTC. Denna offset hämtas in en enda gång när systemet startar och lagras i programmets minne för att senare användas vid infogandet av metadata i användarens meddelanden.

Värt att notera är att denna UTC-offset inte inkluderar eventuell tidsskillnad på grund av sommartid. Eftersom basstationen inte själv innehåller någon klocka så hanterar systemet inte heller denna tidsskillnad, utan det krävs att användaren själv ändrar inställningarna på sitt Twitter-konto för att rätt tid ska visas på skylten. En möjlig lösning på problemet är att låta användaren själv ange sin tidszon i konfigurationsfilen, men detta valdes bort då det skulle leda till en mer komplicerad implementation och ett mer komplicerat användargränssnitt.

5.5.2 Meddelanden

Den sökträng som användaren har angivit i konfigurationsfilen används för att hämta de meddelanden som ska visas på skylten. De meddelanden som matchar söksträngen skickas som svar på förfågningar till följande URL, där där sökträng är den angivna söksträngen: `http://search.twitter.com/search.json?q=sökträng&result_type=recent&rpp=1`. De parametrar utöver söksträngen som ingår i URL:en har följande verkan:

- *result_type=recent* sorterar de matchande meddelandena efter publiceringsdatum i fallande ordning.
- $rpp = 1$ begränsar antalet resultat till ett. *rpp* står för *return per page* (Twitter, 2012b).

Söksträngar som ingår i förfrågningar till Twitters sök-API använder samma syntax som den vanliga sökfunktionen på Twitters hemsida. Denna syntax finns dokumenterad på Twitters hemsida (Twitter, 2012a).
När en sökträng saknas i konfigurationsfilen används istället söksträngen *from:namn +exclude:retweets*, där namn är det användarnamn som har angivits i konfigurationsfilen. *from:namn* matchar enbart de meddelanden som har skrivits av användaren med det angivna användarnamnet. *+exclude:retweets* filtrerar bort de meddelanden som är retweets. Anledningen till detta är att retweets på grund av ett särskilt prefix kan bli längre än 140 tecken och därför inte garanterat kan visas på displayen.

5.6 Formatering

Efter att ett meddelande har tagits emot från Twitters API men innan det visas upp på skylten behöver det formateras. Formateringen innefattar fyra områden:

1. Omkodning: Att hantera tecken som inte stöds av skyltmodulens font genom att ersätta dem mot andra.
2. Normalisering: Att radera överflödiga blanksteg i början av, i slutet av och inuti meddelandet.
3. Justering: Att fördela orden i meddelandet över bildskärmens rader så att så få ord som möjligt behöver brytas upp över två rader.
4. Metadata: Att lägga till metadata till meddelandet i form av ett numeriskt publiceringsdatum.

Resultatet av formateringen är en displaysträng som kan skickas från basstationen till skylten för att visa upp meddelandet och dess metadata.

Innan någon formatering görs trunkeras meddelandet till 140 tecken eftersom det i vissa fall kan vara längre än 140 tecken. Som tidigare har beskrivits gäller detta bland annat för retweets. När denna trunkering har gjorts finns det ingen risk att meddelandet överfyller någon av de buffertar som har allokerats för ändamålet, eller att det blir för långt för att visa på displayen.

5.6.1 Omkodning

De meddelanden som tas emot från Twitters API innehåller Unicode-tecken kodade i UTF-8. Eftersom displayen bara har stöd för 255 olika tecken, som alla skulle kunna representeras annorlunda på skylten än i UTF-8, behöver tecknen kodas om innan de visas. Då både engelska och svenska meddelanden ska kunna visas lades stöd in för ASCII-tecken samt bokstäverna Å, Ä och Ö.

Eftersom Arduino inte har inbyggt stöd för konvertering till och från UTF-8 så omkodar basstationen meddelandena från Twitters API enligt följande regler:

- Samtliga ASCII-tecken samt Å, Ä och Ö kodas om från UTF-8 till den representation som de behöver på displayen.
- Övriga tecken ersätts med tilde-tecken (∼).
- Varje UTF-8-tecken som består av flera bytes ersätts med ett enda tilde-tecken, inte ett för varje byte som de upptar.

5.6.2 Algoritm, omkodning

UTF-8 är en teckenkodning som tillåter varierande antal bytes för att representera enstaka tecken. En fördel med UTF-8 gentemot många andra teckenkodningar är att UTF-8 även är ASCII-kompatibel, vilket innebär att alla tecken som är giltiga ASCII-tecken också är giltiga UTF-8-tecken. Det är dessutom något som gör algoritmen för att koda om UTF-8-meddelanden till skyltens teckenkodning markant enklare (Yergeau, 2003).

Konverteringen består av två huvudkomponenter: gruppering av bytes till tecken, och uppslagning av tecken i en teckentabell för att representera ett tecken på skylten i passande kodning. Denna tabell tillåter uppslagning av ett godtyckligt antal tecken, och de tecken som inte har en översättning får således översättas med ett ersättningstecken.

Givet en början med textmarkören i strängens början konverteras strängen till önskad teckenkodning på följande sätt.

1. Skapa en tom sträng med plats för lika många tecken som originalsträngen.
2. Sök genom strängen byte för byte.
   (a) Om den byte som är under nuvarande markör är ett 7-bitars ASCII-tecken görs en uppslagning i översättningstabellen och det konverterade tecknet läggs till i resultatsträngen. Markören flyttas därefter ett steg framåt.
   (b) Om den byte som är under nuvarande markör har sin åttonde bit satt är tecknet ett multibyte-tecken. Räkna antalet aktiva bitar bland de fyra högre bitarna av nuvarande byte, och flytta markören lika många gånger framåt i strängen. Detta är så många bytes som tecknet under markören bestod av. Uppslagning görs av denna sekvens av bytes i översättningstabellen och läggs till i resultatsträngen.
3. Om hela strängen har behandlats, avsluta och returnera resultatsträngen. I annat fall återgå till punkt två.

5.6.3 Normalisering

För att kunna formatera meddelandet så enkelt och konsekvent som möjligt är det önskvärt att normalisera det så att det har samma struktur som alla andra meddelanden. I detta fall innebär det behandling av meddelandets blanksteg. När meddelandet ska justeras och visas upp blir logiken enklare om vi kan utgå från att varje ord skiljs åt med ett och endast ett mellanslag. På en abstrakt nivå vill vi kunna behandla varje meddelande som en lista av ord där blanktecken (*whitespace*) inte behöver tas i åtanke.

En sådan radering av blanktecken innebär att en användare inte själv kan formatera sina meddelanden genom att infoga extra utrymme i form av blanktecken. Detta är dock konsekvent med Twitters beteende i webbläsaren, där upprepade blanktecken alltid slås samman (collapse) till ett enda blanksteg.

Ett meddelande med blanktecken precis i början och i slutet av texten kan publiceras till Twitter, men det är enbart blanktecken inuti texten som returneras av Twitters API. Detta innebär att radering av såna blanktecken inte behöver ingå i algoritmen för normalisering.

5.6.4 Algoritm, normalisering

Utgå från att strängen består av en mängd tecken, som alla kan kategoriseras att vara whitespace (blanksteg, nyrad, tab, etc) eller en del av ett ord. För

att veta vad som ska returneras gås strängen igenom två gånger: en gång för att räkna ord, och en gång för att extrahera ord.

1. Sätt nuvarande status till WHITESPACE, och ordräknaren till noll.
2. Traversera strängen tecken för tecken.
   (a) Om vår nuvarande status är WHITESPACE.
      i. Om det tecken som är under markören är en whitespace, gör ingenting.
      ii. Om det tecken som är under markören inte är en whitespace, öka ordräknaren och sätt nuvarande status till WORD.
   (b) Om vår nuvarande status är WORD.
      i. Om det tecken som är under markören är en whitespace, sätt nuvarande status till WHITESPACE.
      ii. Om det tecken som är under markören inte är en whitespace, gör ingenting.
3. Allokera utrymme för returvärdet så att det rymmer det antal strängar som ordräknaren står på. Sätt nuvarande status till WHITESPACE.
4. Traversera strängen tecken för tecken. Flytta markören till första whitespace. Flytta markören till första icke-whitespace. Skapa en ny sträng av alla tecken från markören som inte är whitespace. Lägg strängen på första lediga plats i minnet för vårt returvärde.

5.6.5 Justering

Systemet använder sig av två olika sätt att justera ett ord som överskrider längden på en rad:

1. Nedflyttning: Att flytta ned hela ordet till början av nästa rad.
2. Avstavning: Att låta första halvan stanna på samma rad medan andra halvan flyttas ned till nästa rad.

Nedflyttning leder i allmänhet till bra läslighet men kräver mer utrymme än avstavning, i synnerhet när väldigt långa ord flyttas ned.

Avstavning leder till ett bättre utnyttjande av displayen men är svårare att implementera, inte minst för att avstavningsregler är olika beroende på vilket språk som meddelandet är skrivet på. Olika approximationer kan göras men ett bra resultat kan endast uppnås med hjälp av bibliotek för språkbehandling. Eftersom sådana kräver betydande utrymme för diverse språkdatabaser kan de inte användas i detta sammanhang.

Även utan algoritmer för korrekt avstavning kvarstår möjligheten att låta ett ord gå ut till kanten av en rad och flytta ned resten av ordet till nästa rad. Systemet avstavar ord på detta sätt.

Avstavning kan även göras med att infoga ett bindestreck i slutet av ordets första halva. Bedömningen gjordes dock att en sådan naiv avstavning inte är nämnvärt bättre än att låta ordet brytas upp över två rader utan något bindestreck.

5.6.6 Algoritm, justering

Den algoritm som vi tog fram för justering är en relativt simpel girig algoritm: Den flyttar ned ord så långt det går och bryter ord över två rader (utan bindestreck) så fort en nedflyttning skulle orsaka platsbrist. Algoritmen tar som argument en lista med ord.

För att snabbare kunna testa olika lösningar implementerades algoritmen först som en prototyp i skriptspråket JavaScript.

1. För varje ord i listan:
   (a) Om ordet får plats på nuvarande rad
      i. placera ordet på nuvarande rad.
   (b) Annars:
      i. Beräkna det minsta utrymmet som krävs för att placera de kvarvarande orden med början på nästa rad.
      ii. Om utrymmet räcker till:
         A. Placera ordet på nästa rad och fortsätt justeringen därifrån.
      iii. Annars (om ordet måste brytas upp över två rader):
         A. Dela upp ordet i två nya ord: det första så långt som antalet oanvända tecken kvar på raden, det andra bildat av de återstående tecknen.
         B. Placera det första ordet på nuvarande rad.
         C. Infoga det andra ordet på nuvarande plats i listan.

5.7 Metadata

Eftersom displayen har rum för 160 tecken finns det möjlighet att visa mer text än enbart ett meddelande, som högst är 140 tecken långt. Under projektets planeringsfas diskuterades vilken eventuell metadata som skulle visas. Främst diskuterades publiceringsdatum och användarnamn.

Publiceringsdatum är önskvärt att visa på displayen eftersom det ger meddelandet ett sammanhang. Många av de meddelanden som sätts upp på dörrskyltar handlar om tillfällig frånvaro, och då är det ofta viktigt för läsarna att veta när meddelandet publicerades. Användarnamn ger också ett sammanhang, men eftersom en dörrskylt redan är fysiskt monterad på användarens kontorsdörr blir informationen vanligtvis överflödig. Vi bestämde oss således för att enbart ta med meddelandets publiceringsdatum som metadata.

Datumet visas på formatet *dd/mm tt:ss*, där uttrycken syftar på följande information i meddelandets publiceringsdatum:

- dd: Dagen på månaden, utan inledande nolla (1-31).
- mm: Månadens ordningsnummer, utan inledande nolla (1-12).
- tt: Timmen på dygnet, med inledande nolla (00-23).
- ss: Minuten på timmen, med inledande nolla (00-59).

Datumet visas i det nedre högra hörnet av displayen och föregås av ett bindestreck följt av ett mellanslag (- *dd/mm tt:ss*).

En komplikation som uppstår när metadata ska infogas i displayens nedre högra hörn är att algoritmen för avstavning potentiellt kan ha förbrukat detta utrymme genom att flytta ned ord, som i följande exempel med ett 124 tecken långt meddelande.

Meddelande:

a bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb ccccccccccccccccccccccccccccccccccccccccccc ddddddddddddddddddddddddddddddddddddddddddddd

a
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
ccccccccccccccccccccccccccccccccccccccccccc
ddddddddddddddddddddddddddddddddddddddddddddd

Tabell 3: Meddelandet efter avstavning och högermarginaljustering

I exemplet har ordet *bbb...* flyttats ned istället för att avstavas eftersom detta lämnar kvar tillräckligt med utrymme för resten av meddelandet. Problemet är att utrymmet i det nedre högra hörnet nu är förbrukat, vilket gör det omöjligt att infoga ett datum. Detta går att lösa genom att justera algoritmen för justering, men detta gör den mer komplicerad och kräver att den har kunskap om hur den justerade strängen ska användas, vilket ökar

antalet beroenden mellan funktioner.

Istället löses detta problem genom att en 13 tecken lång platshållarsträng infogas i slutet på den lista av ord som ska justeras. Eftersom denna platshållare är lika lång som den metadata som ska infogas i meddelandet (ett datum samt det bindestreck och mellanslag som föregår det) kommer algoritmen för avstavning att garantera att det alltid finns plats över för att infoga metadatan. Infogandet av den faktiska metadatan består sedan av två steg: att ersätta platshållarsträngen med metadatan samt att infoga nyrader och blanksteg så att metadatan visas i displayens nedre högra hörn.

Den platshållarsträng som används är *0123456789abc*. Vilken sträng som helst skulle kunna användas så länge den består av 13 tecken och inte innehåller några mellanslag, men den aktuella strängen har en fördel: inga tecken förekommer två gånger. Detta används när metadatan ska placeras på rätt plats i meddelandet, då platshållarsträngens början identifieras genom att söka efter den första nollan från meddelandets slut.

5.7.1 Algoritm, metadata

Datumet läggs till på meddelandet efter att det har justerats. Eftersom det alltid ska visas det nedre högra hörnet av displayen kräver detta viss logik för att implementera, då längden på meddelandet och antalet nyrader i det är okänt.

1. Räkna antalet nyrader i strängen.
2. Infoga nyrader tills det finns sju nyrader i strängen.
3. Räkna antalet tecken som föregår platshållaren på den sista raden.
4. Infoga blanksteg före platshållaren tills den sista raden är fylld med tecken.
5. Ersätt platshållaren med metadatan.

6 Resultat

6.1 Resultatsammanställning, mätdata och produktegenskaper

6.1.1 Funktionalitet

Systemet i sin helhet är idag fullbordat, enligt planeringen. Basstationen hämtar nya meddelanden från Twitter via HTTP, där den därefter läser ut svaret ur Twitters respons genom en egenskriven HTTP-parser. Under tiden

som meddelandet hämtas strömmas det igenom en JSON-parser baserad på Yajl, som i sin tur signalerar att parsningen är färdig först när önskad data har hittats och extraherats ur i texten.

De meddelanden som hämtas går igenom en process där icke-önskade tecken städas bort, och där alla andra tecken ersätts med sina respektive representationer som krävs för visning på systemets display. Därefter städas meddelandet ytterligare genom att dela upp texten i ord, som sedan justeras efter bästa förmåga i syfte att kunna visa texten på optimalt vis, uppdelat i det antal rader som displayen kräver.

Algoritmen för justering utvärderades genom att testa den på 100 huvudsakligen svenska meddelanden från Twitters API och granska resultatet för hand. Algoritmen fungerade förvånansvärt bra på dessa godtyckligt utvalda meddelanden. I de flesta av meddelandena kunde samtliga ord flyttas ned istället för att brytas över två rader, vilket ledde till bra läslighet. Faktum är att praktiskt taget de enda ord testet som behövde brytas över flera rader var URL:er, som lider av samma problem i nästan alla andra medier.

6.1.2 Begränsningar i Twitters API

Systemets användning av Twitters API lyder under vissa begränsningar från Twitters sida som systemet inte kan påverka. Dessa begränsningar påverkar dels hur pålitliga API-svaren är och dels hur snabbt systemet kan hämta in användarens senaste meddelande.

Twitters sök-API kan bara användas för att hitta meddelanden som är *ungefär en vecka* gamla (Twitter, 2012e). Dessutom fokuserar API:et på *relevans, inte fullständighet*, vilket innebär att vissa meddelanden inte är tillgängliga (Twitter, 2012e). Den testning som gjordes under utvecklingen av systemet stötte inte på några av dessa problem, men de förtjänar att nämnas eftersom de skulle kunna påverka pålitligheten hos en färdig produkt baserad på systemet.

Twitters sök-API kan inte heller användas för att ta emot meddelanden i realtid. Den tjänst som Twitter erbjuder för detta ändamål (Streaming API) används inte av systemet eftersom den kräver autentisering, vilket skulle göra implementationen och användargränssnittet mer komplicerat (Twitter, 2012f). Istället måste systemet regelbundet skicka förfrågningar till API:et. Twitter begränsar dessa förfrågningar till 150 per timme per IP-adress (Twitter, 2012g), vilket innebär att systemet kan skicka förfrågningar maximalt ungefär två gånger per minut. Detta är inget problem för systemet, som skickar en förfrågan per minut, men det skulle orsaka problem om frekvensen ökades.

En ytterligare detalj är att systemet i sin användning av Twitters sök-API förlitar sig på en operator som inte är dokumenterad: *+exclude:retweets*. Syftet med detta beskrivs i avsnittet Twitters API. Konsekvensen om operatorn skulle sluta fungera vore att även retweets, som kan vara längre än 140 tecken, skulle visas på skylten. Eftersom samtliga meddelanden trunkeras till 140 tecken påverkar detta dock inte systemets stabilitet utan kan i värsta fall leda till att slutet av en sådan retweet inte får plats på displayen.

6.1.3 Energieffektivitet

För att mäta energieffektivteten hos skyltmodulen har ett antal mätningar, beräkningar och uppskattningar gjorts. Strömförbrukningen för varje delmodul har så långt det gått uppmätts dels i isolation och dels som en del av det kompletta systemet. Syftet med metoden är att kunna peka ut systemets flaskhalsar. Mätningarna har gjorts vid matningsspänning på 3,7V (från batteriet), som regleras ned till 3,3V för systemet. Alla mätningar har skett vid vanlig rumstemperatur och i övrigt under normala kontorsförhållanden.

Arduino Pro 3,3V (aktivt läge)	$11,4mA$
Arduino Pro 3,3V (strömsparläge)	$16\mu A$
Spänningsregulator (idle)	$1mA$
Då Arduinoplattformen är aktiv, gäller även:	
Radiotransceiver (avlyssning/mottagning)	$51mA$
Radiotransceiver (sändning)	$45mA$
Display (uppdatering)	$40mA$
Display (aktiv)	$0mA$
Batteriavläsning (inaktiv)	*obetydlig*
Batteriavläsning (aktiv)	$4\mu A$
Per aktiv LED	$2mA$
Per aktiverad knapp	$30\mu A$

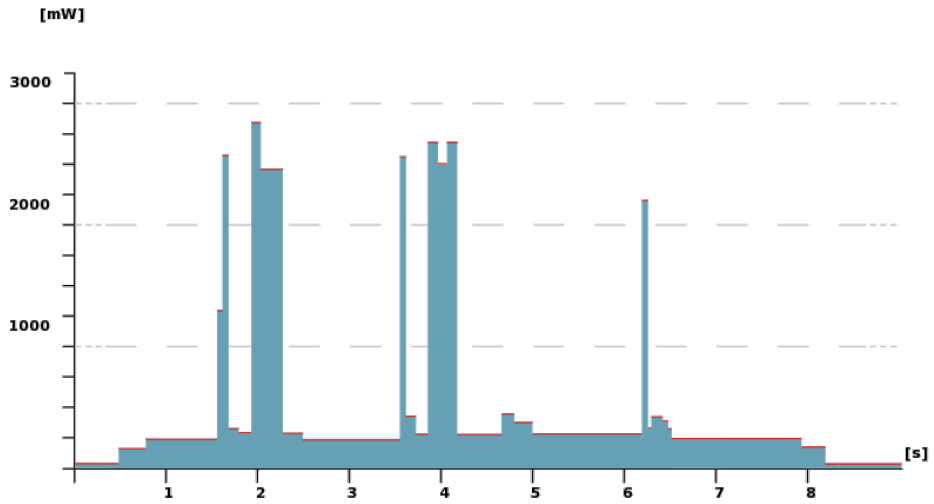
Tabell 4: Strömförbrukning, enskilda moduler

Den slutsats som direkt kan dras är att spänningsregulatorn är den största källan till energiförlust, utslaget över en längre period. Detta beror på att resistiva förluster (i form av värme) fås i regulatorn då spänningen regleras ned till $3,3V$ för systemet. Resultatet blir även att regulatorn blir varm. Lite efterforskning ger dock resultatet att den regulator som används är väl anpassad för systemet och batteriet. Det finns dyrare regulatorer som skulle kunna ge bättre energieffektivitet (Dimension Engineering, 2012).

Med uppdateringsintervall på 5 minuter och uppdateringstid på 10 sekunder för en komplett uppdatering av skylten (med display och radiokommunikation) ger detta en batteritid på strax över 690 timmar ($\approx 29 dagar$) pessimistiskt räknat. I denna beräkning har antagits att radion är aktiv hela tiden och maximalt antal omsändningar görs. Beräkningen tar dock inte hänsyn till förluster i ledningar, regulatorn för XBee-modulen eller temperaturförändringar. Av dessa uppskattas dock endast XBee-modulens spänningsregulator ha en mätbar inverkan, och även den bör vara mycket liten.

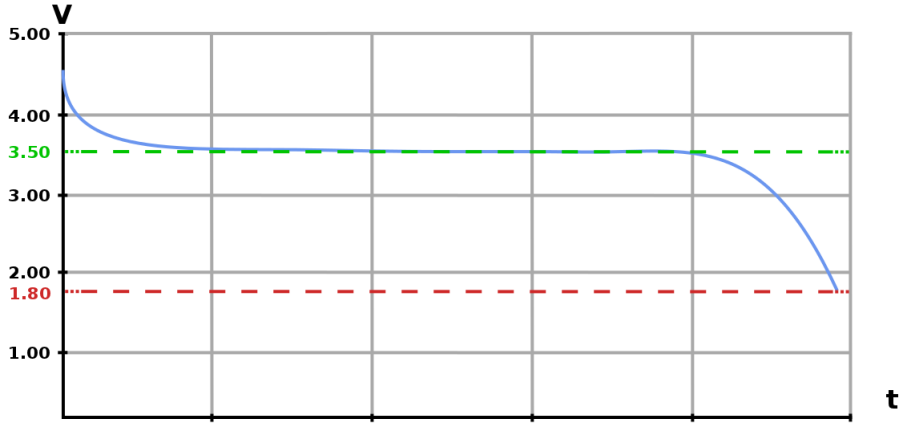
Uppskattningsvis och optimistiskt räknat, fås en maximal batteritid på strax under 1200 timmar ($\approx 50 dagar$). En noggrannare och rimligare uppskattning av driftstiden för systemet ligger runt $800 - 860$ timmar ($32 - 36 dagar$) på ett fulladdat batteri, inräknat alla mätbara faktorer och under antagandet att användande av skylten sker enligt det normala användarscenario som beskrivs tidigare i rapporten. Som väntat är det alltså spänningsregulatorerna som är de största källorna till förluster. Med perfekta spänningsregulatorer beräknas systemet kunna ha en batteritid på uppemot 2400 timmar.

Arduinoplattformen använder sig av en linjär spänningsregulator, vilket innebär att överbliven effekt går förlorad som värme. Den överblivna effekten kommer av differensen i spänning mellan vad som fås ut av batteriet och systemets driftsspänning. Ett alternativ skulle vara att använda en så kallad switchande spänningsregulator. Dessa kräver dock mer avancerade uppkopplingar och är dyrare (Dimension Engineering, 2012).



Figur 5: Effektanalys för skylt

Systemet är försatt i strömsparläge den allra största delen av driftstiden. Aktiv tid uppgår till maximalt 3%, typiskt 1% av tiden, beroende mestadels på radiokommunikationen. Under sömnperioden är effektförbrukningen relativt liten, men under en längre period. Som motsats är uppdateringsperioden kort, flera effektförbrukningsspikar uppträder under uppdateringen. Dessa kommer av radiokommunikation (sändning och mottagning), samt uppdatering av display. Displayen uppdateras mycket snabbt, och sändning av data går också idealt snabbt, dock med risk för omsändning vid misslyckande. Mottagning av och avlyssning efter data på radiolänken är den aktivitet som bidrar mest till energiförbrukningen under uppdateringscykeln. Figur 5 visar effektförbrukning i hela systemet under en uppdateringscykel, där två datapaket tas emot från basstationen. De spikar som noteras kommer från radion, först sändning, sedan mottagning, samt från display.



Figur 6: Urladdningsförlopp för LiPo-batteri

Batteriet som valdes visade sig vara ett mycket bra val, då det ger en lång batteritid, men fortfarande är litet och smidigt nog att bygga in i produkten. Eftersom det rör sig om ett LiPo-batteri så har dock den avläsningsmetod som används i projektet visat sig vara otillräcklig. LiPo-batterier har en relativt konstant spännings-tid-kurva, undantaget relativt skarpa förändringar i början och slutet av urladdningscykeln.

Batteriavläsningen tar för närvarande endast hänsyn till den spänning som fås ut från batteriet, vilket inte fungerar som en bra indikator för laddningsstatus. Figur 6 visar att spänningen som fås från batteriet hålls relativt konstant under urladdningscykeln, fram till en skarp nedåtsväng mot slutet. En alternativ lösning för batteristatusavläsning togs fram, där systemet konstant håller reda på strömmen som dras från batteriet, och utför en integration av strömmen. Med kompletterande information om batteriets initiala

laddning kan med denna metod därefter kvarvarande laddning räknas ut. Vidare fås detaljerad information om hur batteriet används, som kan användas till att räkna fram en uppskattning av återstående batteritid.

Denna lösning är dock inte implementerad i systemet, då det skulle innebära mycket extra beräkningskraft som går åt till att beräkna batteritid, samt en del extra hårdvara. Bland annat skulle en avläsningsbar strömprob behöva kopplas till microcontrollern, samt att externa minneskretsar skulle behöva läggas till för att spara datan relaterat till batterihanteringen. Inom projektet togs snabbt beslutet att detta skulle vara mycket onödigt, på grund av extra kostnader och platsbrist, både fysiskt i enheten och i form av I/O-pinnar på microcontrollern. Vidare skulle det bidra relativt lite till användarupplevelsen. Med nuvarande lösning ger systemet en varning om att batterinivån är låg då spänningen från batteriet börjar sjunka, genom att en LED på skyltmodulen tänds. Då vet samtidigt användaren om att det verkligen är dags att ladda batteriet, då batterinivåvarningen kommer relativt sent i urladdningscykeln.

6.1.4 Radiolänk och nätverk

Radiolänken har testats i kontorsmiljö (Instutitionen för Data- och informationsteknik på Chalmers), och fungerar bra. Ihopparning av modulerna går relativt snabbt (ca 5 sekunder), och inga konflikter med närliggande nätverk märks av. Vid systemstart kan basstationen ta upp till en minut på sig att ansluta sig till nätverksuppkopplingen via DHCP, men oftast går det snabbare.

Radiokommunikationen har testats även över längre avstånd inomhus, och verkar fungera bra upp till ungefär två våningar upp eller ned i ett vanligt kontorshus. Även cirka 40-50 meter i sidled. Detta ger en radie runt basstationen som skylten kan operera inom. Operationsområdet för skylten begränsas av väggar, annan störande elektronisk utrustning och andra fysiska hinder. Sammantaget bedöms radiokommunikationen väl uppfylla de krav på räckvidd och samexistensförmåga med andra nätverk som finns i en vanlig kontorsmiljö, och därmed uppfyller radionätverket även den specifikation som ställdes upp i början av projektet. Test har inte utförts utomhus, då det är utanför skyltens tänkta användningsområde.

Robustheten i radiokommunikationen är relativt god. Vid längre avstånd gör systemet fler omsändningar av data över radiolänken, på grund av paketförluster. Dessa bidrar till längre uppdateringstider för skylten, men är acceptabla. Små avstånd (upp till 10 meter) ger mycket goda förhållanden mellan misslyckade och framgångsrika paketsändningar. Vid enstaka tillfäl-

len händer det att många paket i följd tappas, vilket får som resultat att systemet går in i ett felhanteringsläge. Detta sker dock sällan, och endast vid långa avstånd med fysiska hinder för radiokommunikationen.

6.1.5 Fysiska dimensioner

Modul	**Längd**	**Djup**	**Höjd**
Basstation	150	80	50
Skylt	150	80	20

Tabell 5: Fysiska dimensioner [mm]

Systemets något stora och klumpiga fysiska dimensioner beror på att inbyggnadslådorna och elektroniken är konstruerad för hand och med vanlig labbutrustning och vanliga hem-verkstadsverktyg. Skylten är smidig nog att hänga upp på en dörr, och basstationen är relativt lätt att ställa undan obemärkt på ett vanligt kontor. Ingen av enheterna gör ljud eller har någon nämnvärd värmeutveckling som begränsar placeringsmöjligheterna. Även vikten hos enheterna är låg. På detta sätt kan de fysiska dimensionerna anses uppfylla kravspecifikationen, men vid serieproduktion skulle inbyggnadslådorna och elektroniken behöva göras mindre och smidigare.

7 Diskussion och slutsatser

7.1 Utvärdering av projektupplägg

7.1.1 Kommunikation och beslutsfattande

Kommunikation skedde kontinuerligt genom e-post och veckomöten och bedöms ha ha fungerat utmärkt, med endast mindre missförstånd. Veckomötena ägnades åt beslutsfattande, diskussion kring projektets status och planering inför de kommande veckornas arbete. Beslut togs genom omröstningar, men i praktiken ledde diskussionerna inför varje beslut alltid till konsensus inom gruppen. De problem som stöttes på under projektet hade främst att göra med konstruktion, elektronik och mjukvarubibliotek, och samtliga kunde lösas.

Gruppens handledare medverkade vid ungefär varannat gruppmöte och kommunikationen med honom fungerade bra. Vidare kunde handledaren kontaktas via mail då det behövdes, och kunde följa projektets gång via loggboken. Handledaren bidrog med värdefullt stöd och bra feedback under hårdvaruinköp samt under rapportskrivande.

7.1.2 Arbetsfördelning

Upplägget med att fördela konstruktionsmoment till par av två gruppmedlemmar var lyckat. På detta sätt kunde medlemmarna inom ett arbetspar stödja varandra på ett bra sätt, samtidigt som man kunde arbeta på tre parallella konstruktionsmoment samtidigt inom gruppen, genom tre grupper av två medlemmar vardera. En detalj som kunde ha justerats är att en av arbetsgrupperna arbetade mycket på distans, vilket i sig inte är ett problem, men ibland var det önskvärt att ha en gruppmedlem från varje arbetspar på plats i arbetsrummet. Samtliga gruppmedlemmar bidrog till projektet, och utförde de uppgifter som man hade kommit överens om utan problem. Samarbetet i gruppen fungerade mycket bra, utan några egentliga problem.

7.1.3 Hårdvarubeställningar

Inköpen av hårdvara under projektet har fungerat smidigt via handledare och ansvariga på institutionen. Det har inte varit problem från Chalmers sida att få den hårdvara gruppen velat ha. Ett misstag från gruppens sida som gjordes tidigt i projektet var dock att vänta för länge med beställningen av display. Då gruppen väl beslutat att använda E-Paper, visade det sig vara för lång leveranstid på sådana displayer. Istället fick man använda sig av en bistabil LCD från Kent, vilket i efterhand dock visade sig vara ett fullt jämförbart alternativ till E-Paper. Anledningen till att gruppen sköt upp beställningen av display var att tillräcklig information saknades, att enstaka exemplar till försäljning var sällsynt, och att E-Paper ansågs som något av en risk på grund av dess höga komplexitet jämfört med en normal LCD.

Av en tillfällighet har gruppen fått dubbla uppsättningar av många komponenter i projektet, något som har visat sig vara mycket användbart under konstruktionsfasen. Med dubbel uppsättning hårdvara har gruppen kunnat koppla upp testuppkopplingar som varit till stor hjälp under debugging och felsökning.

7.2 Sidospår och problem under projektet

7.2.1 Operationslägen för XBee

Ett tidigt sidospår inom projektet var designvalet att basera radiolänken på XBee:s AT-läge. AT-läget är ett simpelt kommunikationsläge som kan ses som en trådlös ersättning av en serielänk, och har till skillnad från API-läget inte stöd för några avancerade protokollfunktioner som omsändning och paketinkapsling. Efter att ha arbetat med AT-läget och byggt upp en fungerande radiolänk, togs valet att gå över till API-läget på grund av den

robustare kommunikationen och bättre kontrollen över radiolänkens status. Vidare behövde meddelanden delas upp i flera paket som dessutom behövde samordnas mellan varandra, vilket är olämpligt att implementera med XBee:s AT-läge.

7.2.2 Kanalsökning för radio

Under ett tidigt stadie i utvecklingen fick gruppen problem med XBee:s förmåga att söka genom radiomiljön efter energinivåer i alla de olika frekvenskanalerna inom operationsområdet. Problemet bestod i att det tog för lång tid att söka genom alla kanaler, och det var inte hållbart att ta den första tillgängliga. Problemet löstes genom att radion konfigurerades att använda fasta inställningar för PAN-ID, och klarar nu av att ställa upp en fungerande radiolänk på kort tid.

7.2.3 Kondensatorbank för display

Basstationen drivs med 12V från nätspänning via en transformator, och behöver inga extra kondensatorer. Skylten däremot drivs endast av ett en-cells LiPo-batteri. Detta ger problem vid systemstart då skyltens display startas upp och ska uppdateras.

Problemet upptäcktes då gruppen skulle föra över skylten från en testuppkoppling på breadboard till den hårdvara som ingår i den slutgiltiga prototypen. Systemet startade om sig upprepade gånger, och displayen blinkade utan att skriva ut någonting. Efter mycket felsökning och försök till problemisolering föll misstanken på avkopplingskondensatorerna, som antogs vara för små. Ytterligare experiment genomfördes, och slutsatsen blev att systemet fungerade endast om det kopplades upp med matningsspänning via en breadboard. Mer felsökning och tester följde, och resultatet var förvånande: ledningarna i breadboarden fungerade som kapacitanser, vilket gjorde att testuppkopplingen på breadbord, men inte den slutgiltiga prototypen, klarade av att uppdatera displayen.

Lösningen på problemet blev att lägga till extra avkopplingskondensatorer mellan spänning och jord för hela systemet, samt en extra stor kondensatorbank mellan matningsspänning och jord till displayen. Flera olika kapacitanser testades innan det avgjordes att en stor elektrolytkondensator (470μF) behövdes. Till detta lades två mindre (10μF) och snabbare kapacitanser för att ta bort högfrekventa störningar och hjälpa till att snabbare ladda upp elektrolytkondensatorn.

Dessa kondensatorer behövs för att hantera den strömspik som displayen ger upphov till då den startas och uppdateras. Strömspiken kommer av display-

ens strömförsörjningssystem och dess kondensatorer som skall laddas upp. Utan kondensatorbanken faller spänningen i systemet till cirka halva matningsspänningen vid displayuppdatering. Upp- och urladdningsförloppen i kondensatorbanken går mycket snabbare än displayens uppdateringstid, och utgör därmed inget problem tidsmässigt.

7.2.4 Problem med Arduinos bootloader

I projektets slutfas slutade programmeringen av Arduino Mega:n (basstationen) att fungera. Elektrisk felsökning genomfördes på hårdvaran, men utan att något problem hittades. Efterforskningar på Arduinos forum gjordes, och det visade sig vara ett fel i konstruktionen av Arduinons bootloader, som gör att om vissa tecken eller sekvenser finns i den källkod som laddas upp, kan bootloadern gå in i ett felaktigt läge och fastna. Bootloadern är den del av Arduinos källkod som behövs för att starta igång den användarprogrammerade koden.

Gruppen övervägde flera alternativ, bland annat att försöka bygga basstationen runt en annan typ av Arduino, men detta valdes bort på grund av den minnesbrist det skulle medföra. Efter en del arbete lyckades gruppen lägga in en ny bootloader på Arduino Mega:n, och arbetet kunde fortsätta. Den nya bootloadern hämtades från Arduinoprojektets hemsida och är en uppdaterad version av den som ursprungligen användes på Arduino Mega:n. Det vanliga sättet att lägga in bootloaders på Arduinokort, via Arduinos utvecklingsmiljö, fungerade inte. Istället fick de så kallade Fuse-bits sättas manuellt, vissa hårdvaruregister skrivas över, och en hex-fil föras över med hjälp av en extern programmeringskrets.

7.2.5 Svårigheter med externa bibliotek

Tidigt i projektet valdes Arduino-plattformen då den var välkänd av medlemmarna i gruppen och ansågs vara ett populärt projekt där mycket hjälp kunde fås från communityt. Som en del i det antagandet antogs även att mängden öppen källkod att ta del av, från allt såsom HTTP-klienter till JSON-parser, var både rik och väletablerad. Så visade sig dock inte vara fallet.

Under arbetets gång började gruppen med existerande lösningar för både HTTP-anrop och parsning av JSON. Biblioteken som användes hette aJSON, (GitHub, 2012b) respektive HTTPClient (GitHub, 2012c). Biblioteken är skrivna av samma författare, och i användningen av biblioteken stötte gruppen på problem.

HTTPClient, ett bibliotek för att göra HTTP-anrop, användes en kort stund

i början av projektet. Under användning märkte gruppen att biblioteket gav tillbaka korrupt data, vilket ledde till en närmre granskning av bibliotekets källkod för felsökning. Gruppen fann dock inga uppenbara indikationer på varför responsdatan returnerades som godtyckliga tecken.

aJSON är ytterligare ett bibliotek från författaren till HTTPClient, som är ämnat att kunna läsa in JSON på Arduino. Biblioteket stödjer inte i skrivande stund UTF-8 — det specificeras dock inte vad det innebär — vilket arbetades runt genom att strippa vår data innan parsning. Det visade sig dock att även efter detta så fick aJSON vår Arduino att crasha. När både aJSON och HTTPClient orsakade problem som var svåra att lösa ansåg gruppen att det skulle ta mindre tid att skriva egna ersättare för ovan nämnda bibliotek, vilket också senare gjordes.

För de egengjorda biblioteken bestämde sig gruppen ändå för att försöka använda öppen källkod för att minimera det jobb som behövde göras, och därför valde gruppen att använda en något modifierad version av Yajl som JSON-parser (GitHub 2012d). Modifieringarna innebar att definiera en del konstanter som finns i standard-C, men inte AVR-C.

7.2.6 Minnesbrist vid parsing

Projektet stötte på ett problem när JSON-data ifrån Twitter skulle parsas. Då JSON strängen innehöll escape-tecken så fungerade inte parsningen med parser biblioteket Yajl och basstationen startade helt plötsligt om. Det misstänktes det att åtgången av RAM var för stort och att detta orsakade omstarten. Efter undersökning av parserns källkod upptäcktes det att en buffer allokerades när den stötte på ett escape-tecken. Hur mycket minne som parsern försökte allokera styrdes av ett standardvärde som var inställt på 2 kByte, vilket kan jämföras med basstationens totala RAM-minne på 8 kByte. Standardvärdet ansågs var för högt inställt och när det minskades flöt parsningen på som förväntat.

7.2.7 E-Paper

Originaltanken var att basera skylten runt en E-paper-display. Detta ändrades först under projektets slutfas, då E-paper fanns vara för dyrt och svårtillgängligt med bland annat controllerkort och leveranstider på produkterna. Istället valdes en ChLCD, en annan energieffektiv displayteknik. Såsom marknaden ser ut i dagsläget väntas inte E-paper eller ChLCD kunna konkurrera på den vanliga displaymarknaden under de närmaste åren, på grund av avsaknad av bra färgdisplayer och långa uppdateringstider (Epaper Central, 2012).

7.2.8 Defekt ChLCD

Under projektets slutfas slutade skyltens display att fungera. Styrelektroniken verkar fortfarande fungera och displayen ger rätt svar på kommandon. Även läsning och skrivning till minnet på displayen verkar fungera. Däremot uppdateras aldrig själva displayytan, av okänd anledning. Tänkbara fel kan vara konstruktionsfel eller ESD-skador, men båda dessa förefaller osannolika. ESD-skador borde ha skadat styrelektroniken på displayen, medan konstruktionsfel är mycket ovanliga enligt tillverkaren. Tillverkaren, Kent Displays, kontaktades via mail (kontaktperson: Tony Emanuele, 2012-05-10), men inte heller de kunde ge någon förklaring till problemet. En ersättnings-display fick beställas.

7.3 Jämförelser med liknande projekt

En enkel Google-sökning ger en handfull projekt som involverar Twitter och Arduino. Det främsta exemplet på ett liknande projekt är även det som gav grundtanken till detta projekt, Erico Guizzo:s artikel Send a Tweet to Your Office Door i IEEE Spectrum, juni 2011 (Guizzo, IEEE Spectrum, 2011). Guizzo:s projekt skiljer sig dock på en del viktiga punkter, det är inte trådlöst eller har energieffektivitetsfokus (bland annat används en mindre energieffektiv display), samt att slutprodukten är relativt otymplig. Guizzo:s projekt är även tekniskt enklare och har mindre inbyggd robusthet och felhantering.

Ett annat projekt, Arduino LCD Twitter Display, implementerar liknande funktioner, men har liknande nackdelar som Guizzo:s projekt, i form av energiineffektivt, icke-trådlöst och otympligt. Här behöver Arduino-enheten även direkt koppling till en PC (Instructables, 2011).

Inget annat projekt erbjuder en twitteransluten, trådlös dörrskylt med energieffektiv display och radio, eller lång batteritid. Vidare finns inget annat projekt som erbjuder samma robusta kommunikationslänk eller användarvänlighet. Hummingbird kräver inte heller någon extern PC för att koppla upp sig mot internet, utan använder egen nätadapter för matningsspänning, och går att koppla direkt till en router via vanlig Ethernet.

7.4 Expansionsmöjligheter

En expansionsmöjlighet till projektet, som diskuterades tidigt under planeringsfasen, är att lägga till möjligheten för flera skyltar att ansluta sig till samma basstation. Vidare finns vidareutvecklingspotential för radion och sättet på vilket den anpassar sig till närliggande nätverk.

Andra tänkbara vidareutvecklingar är tillägg av möjlighet för användaren att komma åt och konfigurera basstationer via ett webb-interface, eller att kontrollera hela nätverk av basstation-skyltpar via sitt lokala nätverk eller via internet.

Rent produktionstekniskt skulle systemet kunna anpassas för serieproduktion med relativ lätthet, givet industriell utvecklingsutrustning. Med för ändamålet specialutvecklade mönsterkort och ytmonterade komponenter skulle de fysiska dimensionerna på enheterna kunna krympas avsevärt. Vidare skulle kostnaderna per enhet sjunka dramatiskt vid serieproduktion. Användarvänligheten hos systemet skulle kunna förbättras genom att göra det lättare att komma åt SD-kortet samt att byta och ladda skyltens batteri.

Referenser

Atmel, 2008-2012,

[a] *ATMega328P.* http://www.atmel.com/devices/ATMEGA328.aspx (2012-01-24)

[b] *ATMega2560.* http://www.atmel.com/devices/ATMEGA2560.aspx (2012-01-24)

[c] *ATMega328P Datablad.* http://www.atmel.com/Images/doc8271.pdf (2012-02-15)

.

Arduino, 2009-2012,

[a] *Arduino.* http://www.arduino.cc/ (2012-01-19)

[b] *Arduino Pro.* http://arduino.cc/it/Main/ArduinoBoardPro (2012-01-19)

[c] *Arduino Mega.* http://arduino.cc/it/Main/ArduinoBoardMega (2012-01-19)

[d] *XBee Shield* http://arduino.cc/it/Main/ArduinoXbeeShield (2012-01-24)

[e] *Ethernet Shield* http://arduino.cc/en/Main/ArduinoEthernetShield (2012-01-19)

[f] *Arduino SD* http://arduino.cc/en/Reference/SD (2012-04-14)

.

Cisco Systems, 2003, *CCNA 3 and 4 Companion Guide (Cisco Networking Academy Program), ss. 14, 391-392.*. Cisco Press, Indianapolis, USA, Upplaga 3.

Dell, 2012, *Wireless vs. Wired*, http://www.dell.com/content/topics/topic.aspx/global/learn/network/plan_vs?c=us&l=en&cs=19. (2012-04-27)

Digi, 2012,

[a] *XBee Series 1*, http://ftp1.digi.com/support/documentation/90000976_J.pdf (2012-01-24)

[b] *XCTU*, http://www.digi.com/support/productdetail?pid=3352&osvid=57&type=cabling (2012-02-23)

[c] *Choosing an XBee Antenna*, ChoosinganXBeeAntennahttp://www.digi.com/technology/rf-tips/2007/08 (2012-01-27)

.

Dimension Engineering, 2012, *A beginner's guide to switching regulators*, http://www.dimensionengineering.com/info/switching-regulators . (2012-04-29)

DISPLAY Elektronik GmbH., 2009, *LCD MODULE DEM 240128D FGH-PW. DISPLAY Elektronik GmbH.*, http://www.display-elektronik.de/DEM240128D_FGH-PW.pdf . (2012-05-07).

DISPLAY Elektronik GmbH., 2011, *LCD MODULE DEM 40491 SYH-LY. DISPLAY Elektronik GmbH.*, http://www.display-elektronik.de/DEM40491SYH-LY.PDF . (2012-05-07).

Drew Gislason, EE Times, 2011, *ZigBee Applications*, http://www.eetimes.com/design/embedded-internet-design/4201087/ZigBee-applications--Part-1-Sending-and-receiving-data/ . (2012-02-25)

Embedded Computing, 2011, *Smart energy apps making the move to ZigBee: Q&A with Oyvind Strom, PhD, Senior Director of Wireless Microcontrollers, Atmel Corporation*, http://embedded-computing.com/smart-microcontrollers-atmel-corporation-2#ixzz1Xs7COh4m . (2012-02-05)

Epaper Central, 2012, http://www.epapercentral.com/ . (2012-03-29)

Erico Guizzo, IEEE Spectrum, 2011, *Send a tweet to your office door*, http://spectrum.ieee.org/geek-life/hands-on/send-a-tweet-to-your-office-door. (2012-01-17)

R. Fielding; UC Irvine; J. Gettys; Compaq/W3C; J. Mogul; Compaq; H. Frystyk; W3C/MIT; L. Masinter; Xerox; P. Leach; Microsoft; T. Berners-Lee, 1999, *Hypertext Transfer Protocol – HTTP/1.1. RFC 2616*, http://www.w3.org/Protocols/rfc2616/rfc2616.html . (2012-05-10)

Instructables, 2011, *Arduino LCD Twitter Display*, http://www.instructables.com/id/Arduino-LCD-Twitter-display/ . (2012-01-17)

Droms, R., 1997, *Dynamic Host Configuration Protocol. The Internet Engineering Task Force.*, http://www.ietf.org/rfc/rfc2131.txt . (2012-04-05)

Git, 2012, *Git - Small and fast*, http://git-scm.com/about/small-and-fast .

GitHub, 2012, *Data at GitHub*, https://github.com/blog/1112-data-at-github .

Hilaiel, L., 2012, *Yajl*, http://lloyd.github.com/yajl/, . (2012-05-10)

Kent Displays Inc, 2012,

[a] *Kent Displays*, http://www.kentdisplays.com/ . (2012-04-05)

[b] *1/8 VGA Cholesteric Display Module with SPITM-Compatible Interface. Kent Displays | Resources.*, 2012, http://www.kentdisplays.com/services/resources/datasheets/25075h_240x160_SPI_datasheet.pdf

. (2012-03-31)

Microsoft, 2012, *Naming, Files, Paths, and Namespace*, http://msdn.microsoft.com/en-us/library/aa365247.aspx . (2012-04-03)

M. Gu., 2006, *The World of Liquid Crystal Displays - Cholesteric Liquid Crystal*, http://www.personal.kent.edu/~mgu/LCD/chlc.htm . (2012-04-19)

Nurseitov N. et al., 2009, *Comparison of JSON and XML Data Interchange Formats: A Case Study*, http://www.cs.montana.edu/izurieta/pubs/caine2009.pdf . (2012-02-08)

O.D. Lavrentovich, M.Gu, et al., 2008, *Electro-Optical Effects in Liquid Crystals with Dielectric Dispersion*, Proc. SPIE, Vol. 7050, 70500S .

Processing, 2012, *Processing*, http://www.processing.org/ . (2012-03-30)

Sensor Networks, 2010, *Triple Security in ZigBee: Link, Network and Application layer Encryptions*, http://sensor-networks.org/index.php?page=1010510536 . (2012-02-05)

TechOn, 2009, *Entire Surface of Handset Becomes LCD Display*, http://techon.nikkeibp.co.jp/english/NEWS_EN/20090609/171529/ . (2012-03-30)

The National, 2011 *Facebook and Twitter key to Arab uprisings: report*, http://www.thenational.ae/news/uae-news/facebook-and-twitter-key-to-arab-spring-uprisings-report . (2012-05-08)

Twitter, 2012,

[a] *How to Use Advanced Twitter Search*, https://support.twitter.com/articles/71577, (2012-02-06)

[b] *Twitter turns six*, http://blog.twitter.com/2012/03/twitter-turns-six.html, (2012-02-06)

[c] *REST API Resource*, https://dev.twitter.com/docs/api, (2012-02-06)

[d] *Using the Twitter Search API*, https://dev.twitter.com/docs/using-search, (2012-02-06)

[e] *Streaming API*, https://dev.twitter.com/docs/streaming-api, (2012-02-06)

[f] *Rate Limiting*, https://dev.twitter.com/docs/rate-limiting, (2012-02-06)

Windows Enterprise Networking, 2007. *Configuring lease time.*, Microsoft Windows DHCP Team Blog, http://blogs.technet.com/b/teamdhcp/archive/2007/02/07/configuring-lease-time.aspx. (2012-04-03)

Wiring, 2012, *Wiring*, http://wiring.org.co/ . (2012-03-30)

WIZnet Co., Ltd., 2012, *W5100. WIZnet.*, http://www.wiznet.co.kr/Sub_Modules/en/product/Product_Detail.asp?cate1=5&cate2=7&cate3=26&pid=1011 . (2012-04-04)

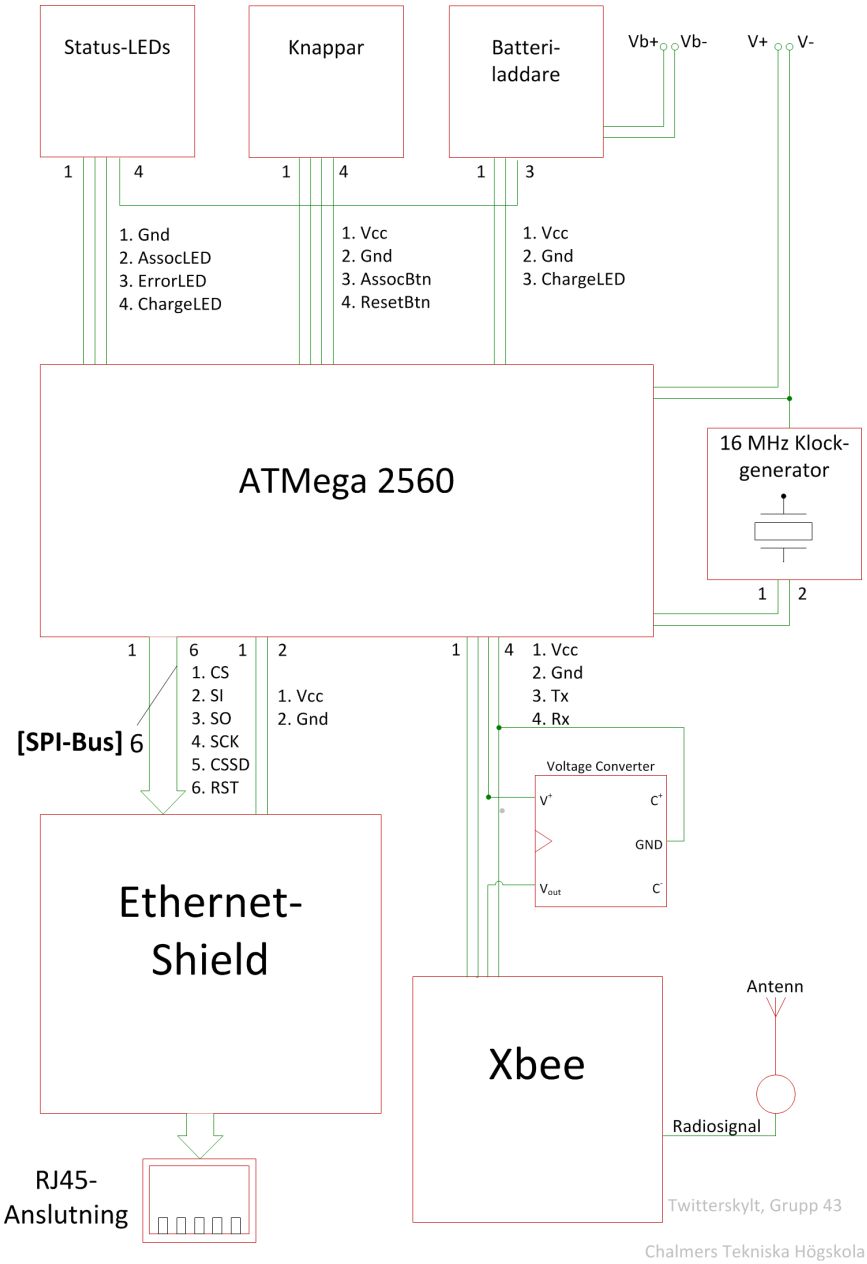
xbee-arduino, 2011, *xbee-arduino*, http://code.google.com/p/xbee-arduino/. (2012-02-01)

Yergeau, F., 2003, *UTF-8, a transformation format of ISO 10646. STD 63. RFC 3629.*, http://tools.ietf.org/rfc/rfc3629.txt. (2012-05-10)

ZigBee Alliance, 2012, *ZigBee*, http://www.zigbee.org/Specifications/ZigBee/Overview.aspx. (2012-01-24)

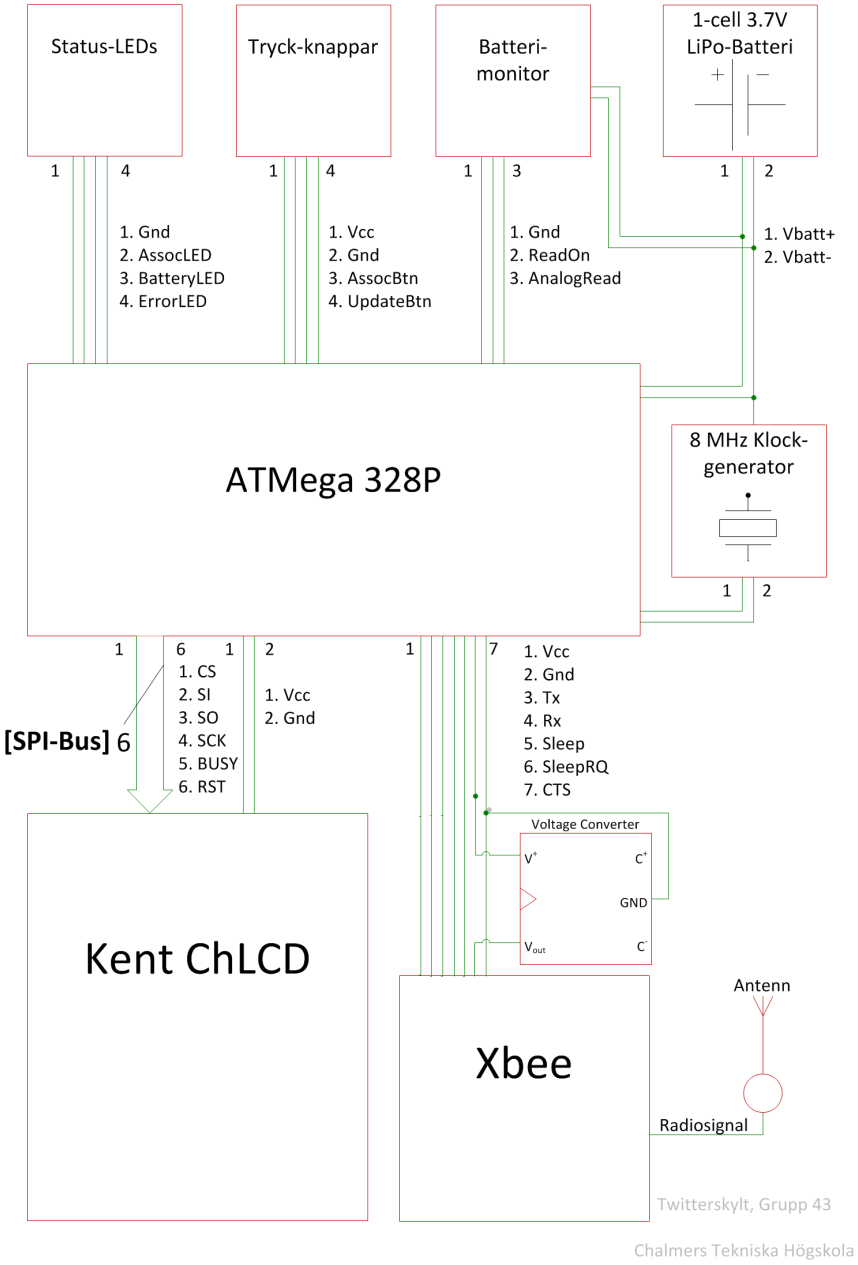
Appendix

A Blockschema, basstation



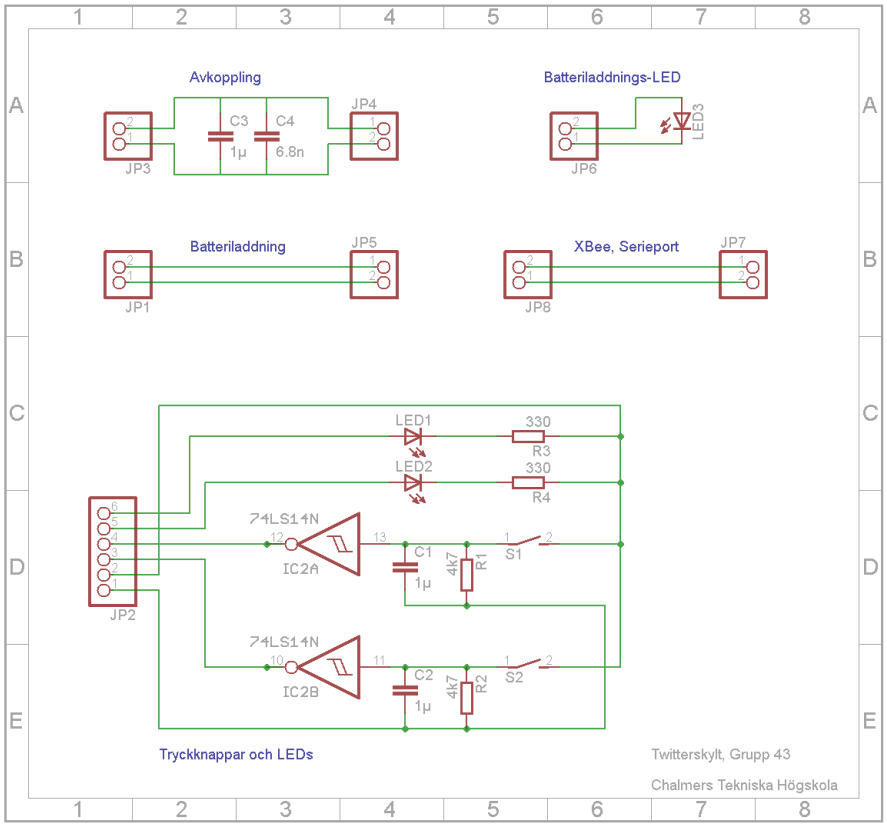
Figur 7: Blockschema, basstation

B Blockschema, skylt



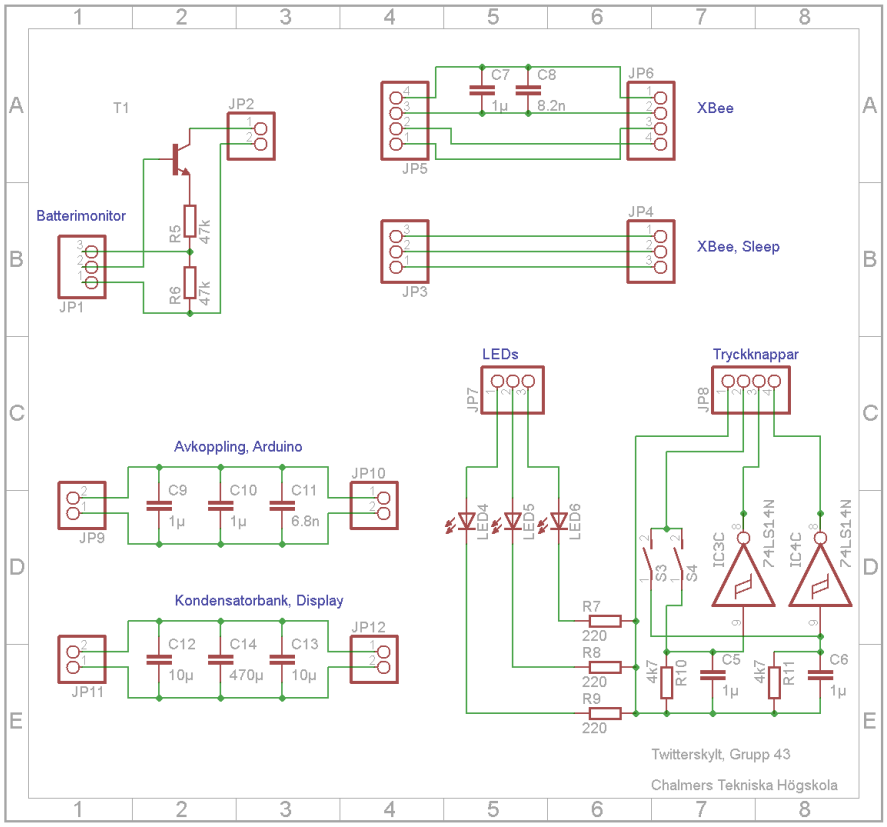
Figur 8: Blockschema, skylt

C Kopplingsschema för stödkretsar i basstation



Figur 9: Kopplingsschema, bas

D Kopplingsschema för stödkretsar i skyltmodul



Figur 10: Kopplingsschema, skylt

E Komponentlista, stödkretsar

Namn	**Värde**	**Typ**
C1-C3	1μ	Kapacitans
C4	6,8n	Kapacitans
C5-C7	1μ	Kapacitans
C8	8,2n	Kapaictans
C9-C10	1μ	Kapacitans
C11	6,8n	Kapacitans
C12-C13	10μ	Kapacitans
C14	470μ	Kapacitans
IC1	-	7414-Schmitt
IC2	-	TMP36
IC3-IC4	-	7414-Schmitt
LED1-LED3	-	LED (5mm)
LED4-LED6	-	LED (3mm)
R1-R2	4k7	Resistans
R3-R4	330	Resistans
R5-R6	47k	Resistans
R7-R9	220	Resistans
R10-R11	4k7	Resistans
S1-S4	-	Tryckknapp
T1	-	2N4124-NPN-TO92-CBE

Tabell 6: Komponentlista, stödkretsar

F Komponentlista, huvudsystem

Komponent/Kort	**Tillverkare**
Arduino Mega 2560	Arduino
Arduino Ethernet Shield	Arduino
XBee Shield	Arduino
XBee Series 1 Whip Antenna 1mW	Digi International
USB LiPoly Charger Single Cell	Sparkfun Electronics Inc.

Tabell 7: Komponentlista, basstation

Komponent/Kort	**Tillverkare**
Arduino Mega 2560	Arduino
Arduino Ethernet Shield	Arduino
XBee Shield	Arduino
XBee Series 1 Whip Antenna 1mW	Digi International
USB LiPoly Charger Single Cell	Sparkfun Electronics Inc.

Tabell 8: Komponentlista, skylt

G Arbetsfördelning

Nedan redovisas de gruppmedlemmar som varit huvudansvariga för de olika områdena under projektet. Till viss del överlappar områden, och gruppmedlemmar har varit inblandade i andra områden än de som står listade nedan. Under rapportskrivningen har gruppmedlemmarna fokuserat på sina expertisområden, men även läst genom de andras text och utbytt synpunkter och förslag.

Konstruktion

1. Tweethämtning - *Kim*
2. Parsing - *Jakob*
3. Formattering - *Kim, Jakob*
4. Användarkonfigurering (SD-kort) - *Andreas, Anton*
5. Ethernet - *Andreas, Anton*
6. Radiolänk - *Lars, Fredrik*
7. Display - *Lars, Anton*
8. Övrig elektronik och mekanik - *Fredrik*

Presentation

1. Halvtidsredovisning - *Kim, Andreas*
2. Slutpresentation - *Jakob, Lars, Fredrik*
3. Opponering - *Kim, Andreas, Anton*
4. Utställningsaffisch - *Jakob*

H Användarmanual

Hummingbird kopplar din kontorsdörr till Twitter. Dina tweets visas på en skylt som du enkelt monterar utanför ditt kontor. Skylten är helt trådlös och hämtar sina meddelanden från en basstation vilken du kopplar till ett vanligt nätverksuttag. Batteritiden är beräknad till minst 800 timmar för ett 2000mAh-batteri av den typ som följer med.

H.1 Montering

Basstationen skall ges tillgång till spänning och nätverksuppkoppling. Skylten är helt trådlös och monteras lämpligen på en dörr eller en vägg. Basstation och skylt behöver befinna sig inom räckvidd för varandra. Räckvidden beror på mängden radiotrafik i omgivningen och på antalet hinder i vägen, såsom väggar, större elektrisk utrustning. En tumregel i vanlig kontorsmiljö kan vara två våningar i höjdled eller 50 meter i sidled.

H.2 Konfigurering

1. Anslut ett micro-SD-kort till kortläsaren i din dator.
2. Kör det konfigureringsskript som är anpassat för ditt operativsystem: *config-windows.bat* för Windows eller *config-linux.sh* för Linux.
   (a) Ange ditt Twitter-användarnamn och tryck *Enter*.
   (b) Om du inte vill visa alla dina tweets på skylten, ange den söksträng som ska användas för att välja ut tweets och tryck *Enter*. Tryck annars bara *Enter*. Syntaxen för söksträngar dokumenteras på Twitters hemsida: `https://support.twitter.com/articles/71577`
3. Ta ut micro-SD-kortet från din dator och anslut det till kortläsaren i basstationen.

H.3 Användning

1. Förse basstationen med nätverksuppkoppling via RJ45-kontakten.
2. Starta basstationen genom att ansluta en spänningskälla (5V via USB alternativt 12V med transformator). Båda status-LEDs på basstationens ovansida tänds för att visa att systemet har spänning. De släcks igen då basstationen fått en IP-adress via DHCP och är redo för att anslutas till en skylt.
3. Starta skylten genom att ansluta LiPo-batteriet.

4. Tryck först på *Associate-knappen* på basstationen och sedan på *Associate-knappen* på skylten. Vänta i ett par sekunder. När nätverksindikatorerna på båda modulerna lyser fast grönt är skylten ihopkopplad med basstationen och kommer automatiskt att hämta in din senaste tweet. Om ihopparningen av modulerna misslyckas, försök igen eller följ anvisningarna under *Felsökning*.

När skylten är ihopkopplad med basstationen kommer den automatiskt att var femte minut hämta in din senaste tweet. Om du vill hämta in din senaste tweet direkt, tryck på knapp B på skylten. Om du precis skrev din tweet kan det dröja upp till en minut innan den kan hämtas in av Hummingbird.

H.4 Statusindikatorer och knappar

LED	**Fast ljus**	**Blinkande**
Röd	Fel eller timeout	-
Grön	Ansluten till skylt	Söker efter skylt
Röd (sidan)	Batteriet laddas	-

Tabell 9: LEDs på basstation

Knapp	**Funktion**
A	Ihopparning med skylt
B	Systemomstart

Tabell 10: Knappar på basstation

LED	**Fast ljus**
Röd	Fel eller timeout
Gul	Låg batterinivå
Grön	Ansluten till basstation

Tabell 11: LEDs på skylt

Knapp	**Funktion**
A	Uppdatering
B	Ihopparning med basstation

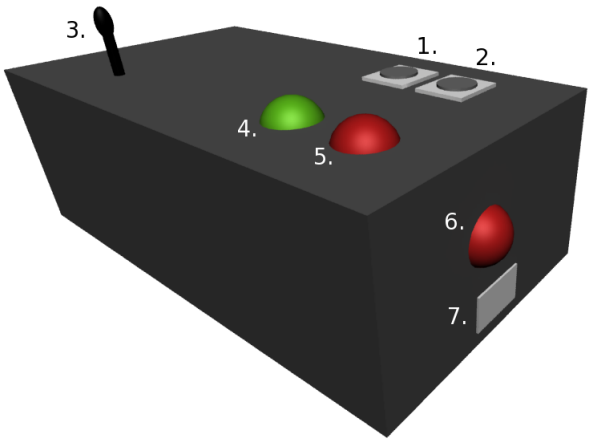
Tabell 12: Knappar på skylt

H.5 Felsökning

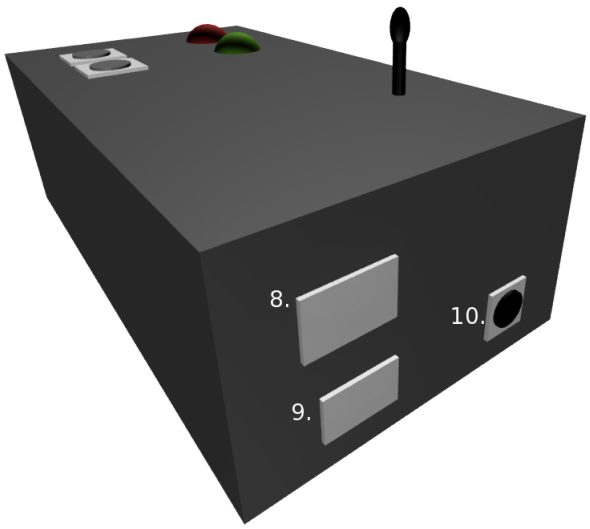
Om den röda lysdioden på basstationen lyser, testa först att starta om systemet genom att trycka på Knapp A på basstationen.
Om felet uppstår igen, följ anvisningarna nedan:

- Säkerställ att systemet är korrekt monterat: Följ anvisningarna under *Montering*.
- Säkerställ att din internetuppkoppling fungerar: Öppna webbläsaren i en dator som är kopplad till ett nätverksuttag i ditt kontor och säkerställ att hemsidor kan laddas.
- Säkerställ tillräcklig matningsspänning: För basstationen behövs en matningsspänning på 5V via USB eller 12V via medföljande transformator, som skall anslutas till nätspänning. För skylten skall medföljande batteri användas (en-cells LiPo).
- Säkerställ att du har en giltig konfigurationsfil: Anslut basstationens SD-kort till en dator och säkerställ att det innehåller en fil med namnet config.txt där den första raden innehåller ett giltigt Twitter-användarnamn och där den andra raden antingen är tom eller innehåller en giltig Twitter-söksträng. Syntaxen för söksträngar dokumenteras på Twitters hemsida.

I 3D-Modell av basstation



Figur 11: Basstation, sidovy

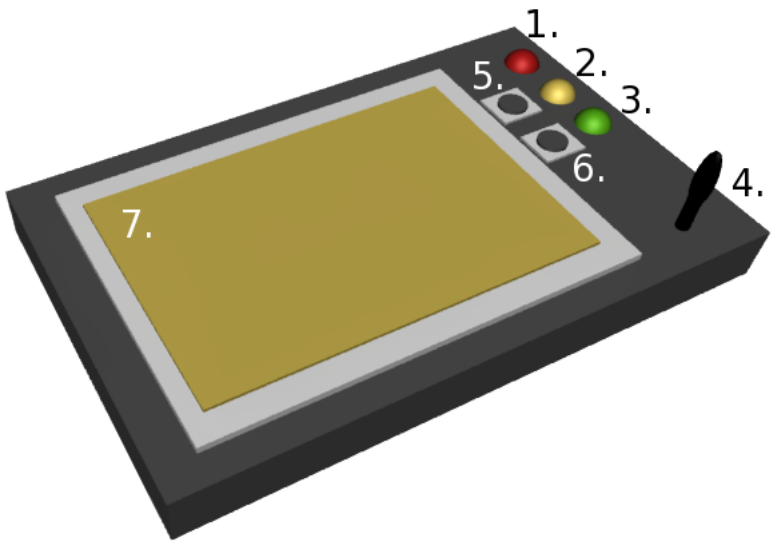


Figur 12: Basstation, andra sidan

I.1 Sifferförklaringar

1. Knapp A: Ihopparning
2. Knapp B: Systemomstart
3. Antenn
4. Associate-LED (grön)
5. Error-LED (röd)
6. Batteriladdnings-LED (röd)
7. Batteriladdningsanslutning för LiPo-batteri
8. Ethernet-port
9. USB-port
10. 12V-matning

J 3D-Modell av skylt



Figur 13: Skyltmodul, ovanifrån

J.1 Sifferförklaringar

1. Error-LED (röd)
2. Batteri-LED (gul)
3. Associate-LED (grön)
4. Antenn
5. Knapp A: Uppdatering
6. Knapp B: Ihopparning
7. Display