10. Ethernet shield

Hva er et ethernet shield

Et ethernet shield er et Arduino shield som git deg muligheten til å koble på et vanlig nettverk via kabel, og kommunisere med vanlige datamaskiner. Veldig ofte er nettverk koblet sammen med Internett. En Arduino tilkoblet et slikt nettverk kan gjøre som andre datamaskiner; delta i chat, lese internettsider og lignende.

I tillegg har disse ethernet shieldene en SD kortleser. Dette gir deg muligheten til å få Arduino til å levere ut mye større sider enn den kan holde i minnet. Eller det er mulig å kun bruke SD kort-egenskapen, og se helt vekk fra nettverksdelen av kortet.

Mer utfyllende kan finnes på Arduino sine sider, her og her. Det er også mange andre ressurser åpent på nett for den som har lyst til å lete.

Ulike bruksområder

Ganske mange av de ulike bruksområdene er godt dokumentert fra Arduino sin side.

For eksempler og forklaring på bruk av SD kort, har Arduino et innebygd bibliotek for dette med medfølgende hjelesider. Når du installerte Arduino, ble disse eksemplene også installert.

For eksempler for ethernet delen, har Arduino et innebygd bibliotek for dette med medfølgende hjelpesider. Her vises det fram eksempler på blant annet hvordan en web klient kan lages, en chat server (og videre chat klienter), en veldig enkel Web server (som kun kan skrive ut en enkelt side) og fler. Det anbefales å se litt på de forskjellige eksemplene.

Raskt om teknologien bak Internett

Her beskriver vi veldig raskt og overfladisk noe av teknologien bak Internett. Denne introduksjonen er på ingen som helst måte fullstendig, NTNU tilbyr egne fag som går mer i dybden på de forskjellige teknologiene. **Det er på ingen som helst måte nødvendig å ha full kontroll på dette for å bruke ethernet shieldet**, men det kan være greit å vite litt om web teknologi. For den interesserte leser legger vi også inn noen linker til Wikipedia hvor du kan lese mer om de forskjellige temaene.

OSI modellen

OSI modellen (engelsk her) er en måte å beskrive nettverksprotokoller på. De nederste nivåene i denne stacken beskriver de fysiske tilkoblingene, eksempelvis ethernet og WiFi. Naturlig nok finnes det flere bunnlag enn disse, men i denne sammenhengen er lagene opp til hit ikke så viktige. Når vi ser på teknologiene fokuserer vi på ethernet siden av teknologien.

MAC adresse

For å unikt identifisere et nettverkskort, har alle nettverkskort en MAC adresse. Denne er på Arduino ethernet shield oppgitt som seks grupper á to siffer med heksadesimale tall, totalt 6 byte (eller 48 bit). Denne adressen brukes for å adressere beskjeder mellom enheten og den nærmeste ruteren. For å få Arduino shieldet til å virke, må du skrive inn denne adressen i koden din.

TCP/IP

I OSI modellen tilhører TCP (Transmission Control Protocol) transport laget, mens IP (Internet Protocol) tilhører nettverkslaget. Disse protokollene nevnes ofte sammen siden blant annet HTTP baserer seg på en stabil tilknytning (som TCP tilbyr). Alternativer til TCP inkluderer UDP, en protokoll som sender enkeltbeskjeder uten å gjøre noen sjekk på om de kommer fram.

IP inneholder en adresse, IP adressen, som unikt identifiserer en maskin på Internett. Denne protokollen finnes i flere versjoner, men den du trenger å forholde deg til er IPv4. IPv4 har adresser på 4 byte (fire tall mellom 0 og 255 skilt med punktum). For å få tildelt en IP adresse automatisk, finnes en teknologi som heter DHCP (Dynamic Host Configuration Protocol). Heldigvis har Arduino ethernet biblioteket støtte for denne protokollen, så du trenger ikke å konfigurere IP adressen selv.

Sitter du på NTNU nettverket kan IP adressen du får tildelt automatisk brukes direkte som Internettadresse (skriv inn den tildelte adressen i adressefeltet i nettleseren direkte). Sitter du på et hjemmenettverk brukes veldig ofte en teknologi som kalles NAT (Network Address Translation), noe som gjør det nært umulig starte kontakt med enheten fra utsiden uten å konfigurere ruteren din. Inne på hjemmenettverket vil du ikke merke at dette skjer.

En siste ting vi skal nevne i denne sammenhengen er porter. Siden en IP adresse kan brukes til forskjellige ting (hoste en web side, mail server, spillserver og lignende), trengs et system for å skille de ulike tjenestene som tilbys. Her kommer noe som heter porter inn. Du kan selv bestemme hvilken port som skal benyttes til hva, men vær klar over at ulike tjenester har ofte standardiserte porter. For eksempel er vanlige web sider å finne på port 80. Du kan legge en webserver på annen port, men da må du også spesifisere porten når du skriver inn adressen. Eksempelvis hvis du starter webserveren på port 8080, og IP adressen er 127.0.0.1, kan du finne siden ved å skrive inn 127.0.0.1:8080 i adressefeltet i nettleseren. Hadde den vært på port 80, hadde du sluppet kolonet og det som kom etter.

HTTP

HTTP (Hypertext Transfer Protocol) er en tekstbasert overføringsprotokoll for Internett. Den har forespørsler (request) og svar (response), som består av en header og kropp.

En header i en forespørsel forteller blant annet hvilken metode den vil benytte (GET, HEAD, POST etc.) og hvilken ressurs den etterspør. Enkelte av disse metodene, som POST, har en kropp som inneholder data. GET, som er den forespørseltypen du oftest bruker når du etterspør en Internettside inneholder ikke kropp.

Svaret består av en header som forteller om statusen til forespørselen, var den vellykket? Eventuelt: Hvilken feil oppstod? Disse statusene har tresifrede feilkoder, for eksempel *404 Not Found* (ikke funnet). I tillegg finnes annen meta informasjon i headeren, som hvilken type data (MIME type) svaret inneholder. I kroppen til svaret finner vi dataene som ble etterspurt, som for eksempel en Internettside skrevet i HTML.

Web sider

For å vise web sider, trengs det en måte å beskrive dokumenter på som "alle" er enige om at er bra å bruke. Teknologiene blir standardisert av W3C. Det finnes et utall av tutorials for de forskjellige teknologiene ute på nettet, en av de som samler mange av de viktigste er W3Schools. **Merk:** W3Schools er ikke en del av W3C, så det kan være informasjon som er ikke i henhold til standarden der.

Følgende er noen av de viktigste teknologiene som brukes på nettet:

HTML

HTML (Hypertext Markup Language) brukes for å beskrive strukturen til en nettside. Dette brukes for å lage beholdere som tekst, bilder eller annet kan pakkes inn i. W3Schools har tutorial på denne teknologien.

CSS

CSS (Cascading Style Sheets) brukes for å beskrive design, layout (gjerne responsiv; reagerer på størrelsen på skjermen), farger, enkle animasjoner og lignende. W3Schools har tutorial på denne teknologien. Det finnes også en del rammeverk som kan gjøre forskjellige ting (som responsivt design) enklere.

Javascript

Javascript er et programmeringsspråk. Dette språket kan kjøres på nettsider for å kontrollere det aller meste som skjer i nettleseren. W3Schools har en tutorial på denne teknologien. I tillegg finnes det et utall av rammeverk som kan gjøre veldig mye enklere. Eksempler er JQuery, Bootstrap, AngularJS... Det finnes mange fler.

Et undersett av Javascript er JSON (Javascript Object Notation), en teknologi som brukes for lagring og overføring av data.

XML

XML (Extensible Markup Language) brukes for å beskrive data på en strukturert måte. HTML er i stor grad (men ikke fullstendig) en spesiell versjon av XML. XML brukes i stor grad av Web services for utveksling av informasjon. Eksempelvis tilbyr yr.no gratis værdata i XML format.

Web server

Web server er en tjener som behandler HTTP forespørsler. For å sette opp en ekstremt enkel web server på Arduino med ethernet shield, kan Arduinos WebServer eksempel brukes. Dette eksempelet svarer alle forespørsler med en fastprogrammert side. Mere avanserte tutorials som bruker SD kortet på ethernet shieldet eksisterer. Disse har dessverre en del begrensninger. Vi har hatt problemer med å finne kode som er veldig generell, gir feilmeldinger når filer ikke blir funnet og som tar nok høyde for Arduinos svært begrensede ressurser. Derfor har vi lagd ett bibliotek som gjør dette for deg!

Utfordringer på Arduino

Den største utfordringen for Arduino som web server er utvilsomt minnet. Arduino UNO har 2048 byte ram, noe som tilsvarer 2048 bokstaver (om alt minnet brukes til lagring av bokstaver). I tillegg til masser av tekst (HTTP headere og HTML kode for feilmeldinger) trengs det også en del ram for å lagre andre variabler. Det finnes flere metoder som kan lagre tekst som gjør at det ikke er i ram hele tiden. En metode er å lagre det i programminnet til vanlig, og overføre det til vanlig ram når det trengs. Dette er avansert, og brukes kun når du må. Det er denne metoden vi har brukt i biblioteket vårt, noe som gjør at du slipper.

Identifisering av MIME type og fortelle klienten om dette kan også være litt vrient. Vi har ikke funnet noen tutorials som gjør dette enkelt å forstå eller enkelt å reprodusere, derfor har vi bakt dette inn i biblioteket vårt, basert på filnavnet som etterspørres.

Biblioteket

Vi har lagt ut et bibliotek som skal gjøre det mye enklere å lage en web server som gir ut filer. Filene leses i utgangspunktet rett fra SD kortleseren, men vi har lagt med muligheten for å filtrere og endre forespørsler og svar. Vi har prøvd å minimere minnebruken til biblioteket, men bruker allikevel rundt 1300 byte. Dette gjør at du bare har 200-400 byte du kan bruke i tillegg til biblioteket. Brukes noe særlig mer enn dette blir Arduino ustabil, den henger, får uforklarlige omstarter eller lignende.

Biblioteket vårt er beregnet til å starte en web server som gir ut filer rett fra et SD kort. I tillegg kan du filtrere forespørsler, eller til og med bruke filene som templates hvor du endrer deler av teksten som blir levert til klienten. Eksemplene vi har lagt ut er:

- PLabExamples/examples/17.Web/BasicHttpFileServer - Setter opp en server med port 80 som deler ut alle filene som er på SD kortet.
- PLabExamples/examples/17.Web/HttpFileServerFilters - Setter opp en server som gir all debug informasjon via filter. Fungerer som dokumentasjon på alle filtreringsmuligheten du har med dette biblioteket.
- PLabExamples/examples/17.Web/HttpFileServerGetQuery - Viser hvordan du kan lese ut informasjon som har blitt sendt med GET metoden. Slike forespørsler forekommer fra blant annet AJAX og HTML form (når metoden er definert som GET).
- PLabExamples/examples/17.Web/HttpFileServerFilterTest - Leter etter teksten "test" i alle filer som blir etterspurt, og legger til " INSERTED BY FILTER " rett etter denne teksten. Dette eksempelet kan brukes som utgangspunkt hvis du vil lage en template. For eksempel kan du velge ut ett eller flere tegn (eller kombinasjoner) som ikke er naturlig i noen av filene (for eksempel er "$" lite brukt) og sette inn egen tekst på dette punktet. Bruker du en slik strategi kan det være greit å følge med hvilken fil(type) du svarer: når bilder oversettes til tekst kan det bli mange spesielle tegnkombinasjoner av seg selv.

I biblioteket har vi lagt inn støtte for følgende feilmeldinger, en liste som kan utvides:

Kode	Beskjed	Beskrivelse
400	Bad Request	Klienten hadde syntaksfeil i forespørselen.
404	Not Found	Fant ikke ressurs, for eksempel fantes ikke filen på SD kortet
414	Request URI Too Long	Klarte ikke å holde hele den etterspurte adressen. Som standard i biblioteket vårt skjer dette når den etterspurte URI-en er over 79 tegn.
500	Internal Server Error	Når det oppstår en uforutsett feil i serveren.
501	Not Implemented	Alle forespørsler som ikke bruker GET eller HEAD metode vil trigge denne.
505	HTTP Version Not Supported	HTTP finnes i flere varianter. Denne serveren støtter kun 1.x.

Filtyper, og da innhold, gjenkjennes av biblioteket basert på filnavnavslutningen. Det vil si det som kommer etter siste punktum i filen (heter filen index.html, snakker vi da om html). De MIME typene biblioteket gjenkjenner er:

Filetternavn	MIME type
htm	text/html
html	text/html
xml	application/xml
css	text/css
js	application/javascript
json	application/json
jpg	image/jpeg
jpeg	image/jpeg
png	image/png
gif	image/gif
svg	image/svg+xml