Opdrachten week 5 lab 1 – Wifi (sockets)

Tot nu toe heb je onder andere geleerd om de TI-drivers te gebruiken en om TI-RTOS te gebruiken.

Bij dit lab wordt er weer gebruik gemaakt van de SimpleLink™ Wi-Fi® CC3220S LaunchPad™. Je gaat leren:

- hoe de wifi-software in de SimpleLink SDK is opgebouwd;
- hoe je gebruik kunt maken van het HTTP-protocol om je LaunchPad via wifi gegevens van het World Wide Web op te laten halen;
- hoe je gebruik kunt maken van het ICMP-protocol om de wifi verbinding van je LaunchPad te testen;
- hoe je sockets kunt gebruiken om je LaunchPad met behulp van het TCP-protocol via wifi met een ander device op je WLAN te laten communiceren.

De netwerkprocessor (NWP) van de CC3220S bevat alle functionaliteit om succesvol verbinding te maken met een wifi-netwerk en hierover internetcommunicatie op te zetten. De Simplelink SDK voorziet in de benodigde functies om de NWP aan te sturen en correct te configureren en dit is wat we gaan gebruiken in de komende opdrachten. Eerst ga je zien hoe de Launchpad een protocol uit de applicatielaag, namelijk HTTP[1], kan gebruiken. Vervolgens leer je hoe een protocol uit de netwerklaag, namelijk ICMP[2], gebruikt kan worden. Tot slot wordt uitgebreid ingegaan op een protocol uit de transportlaag, namelijk TCP[3], waarmee we verbindingen op gaan zetten tussen hosts en over deze verbindingen data gaan versturen.

5.1.1 Voordat je verder gaat is het belangrijk dat je het project `Network_Terminal` hebt ingeladen in het flash geheugen van jouw LaunchPad zoals uitgelegd in opdracht 1.1.3. Dit zorgt ervoor dat er geen overbodige wifi-netwerken zijn, en dat de LaunchPad standaard als station[4] staat ingesteld, in plaats van als accesspoint. Als dit toen niet is gelukt, is het zaak dit nu opnieuw te proberen. Wanneer dit is gelukt kun je verder.

[1] Zie eventueel `https://nl.wikipedia.org/wiki/Hypertext_Transfer_Protocol`.

[2] Zie eventueel `https://nl.wikipedia.org/wiki/Internet_Control_Message_Protocol`.

[3] Zie eventueel `https://nl.wikipedia.org/wiki/Transmission_Control_Protocol`.

[4] Een station is een apparaat dat verbinding maakt met een accesspoint

5.1.2 We beginnen dit lab met een voorbeeldproject uit de SDK. Importeer het project `httpget_CC3220S_LAUNCHXL_tirtos_ccs`[5] vanuit de SDK map. Dit project kun je niet meteen uitvoeren, er moeten eerst een aantal wijzigingen worden aangebracht. Eerst zal uitgelegd worden hoe dit project in elkaar steekt:

A Maak de Debug configuratie Active en delete vervolgens de MCU+Image configuratie. Verwijder `image.syscfg` en `MCU+Image` uit het project.

B Open het bestand `main_tirtos`. Je ziet dat hierin één *pthread* wordt aangemaakt. Deze thread voert de functie `mainThread` uit. Deze functie is op regel 52 **`extern`** gedefinieerd en bevindt zich dus in een ander bestand.

C Open het bestand `platform.c`. Dit bestand bevat naast de functie `mainThread` nog een aantal andere functies en definities. De code in dit bestand zorgt ervoor dat er een wifi-verbinding wordt opgezet en het is niet nodig om deze code geheel te begrijpen. Bepaalde delen moet je echter wel begrijpen en aan kunnen passen.

- Op regel 52 wordt de `SSID_NAME` gedefinieerd. Dit is de naam van het wifi-netwerk waarmee de LaunchPad probeert te verbinden. Wijzig deze naam `"CiscoK3637"` in de naam van jouw wifi-netwerk[6].
- Op regel 55 wordt `SECURITY_TYPE`, de beveiliging van het wifi-netwerk, gedefinieerd. Als jouw wifi-netwerk een open netwerk is, dan hoef je niets te doen. Als jouw wifi-netwerk beveiligd is met een wachtwoord, dan moet je `SL_WLAN_SEC_TYPE_OPEN` wijzigen in `SL_WLAN_SEC_TYPE_WPA_WPA2` en het wifi-wachtwoord invullen in de nu nog lege string bij `SECURITY_KEY` op regel 58.

Voor mijn thuisnetwerk gebruik ik, bijvoorbeeld, de volgende definities:

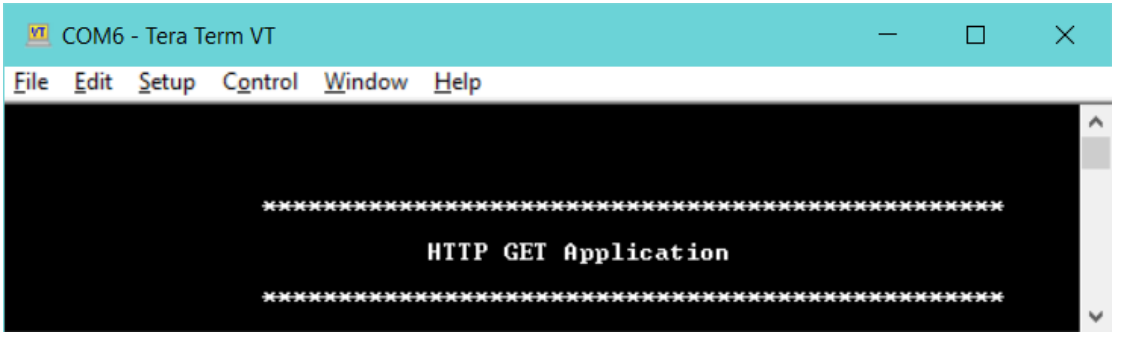
```
#define SSID_NAME "Broeders"
#define SECURITY_TYPE SL_WLAN_SEC_TYPE_WPA_WPA2
#define SECURITY_KEY "Groot Geheim"
```

[5] `C:\ti\simplelink_cc32xx_sdk_6_10_00_05\examples\rtos\CC3220S_LAUNCHXL\demos\httpget\ti-rtos\ccs`

[6] Thuis gebruik je uiteraard je eigen wifi-netwerk. Op school maak je gebruik van een wifi-hotspot op een telefoon (zoek eventueel op internet uit hoe dat moet op jouw telefoon). We maken geen gebruik van het wifi-netwerk van de hogeschool (eduroam) omdat je dan je gebruikersnaam en wachtwoord in je code zou moeten zetten en dat willen we om veiligheidsredenen vermijden.

We bespreken nu regel voor regel de werking van de functie `mainThread`:

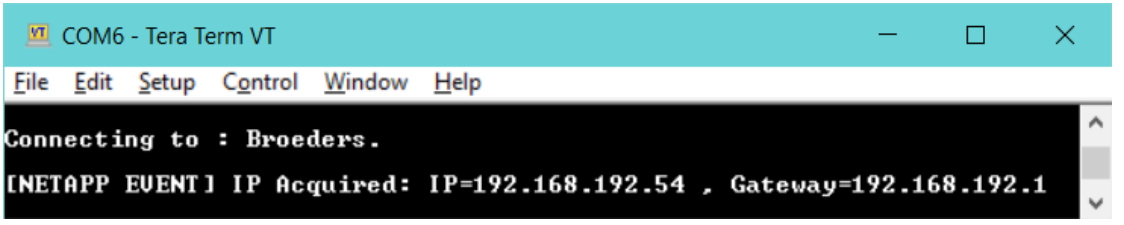
- Op regel 375 wordt `SPI-init()` aangeroepen om de SPI-bus te initialiseren. Dit is nodig omdat de ARM-processor die wij programmeren via de SPI bus met de networkprocessor (NWP) communiceert. Deze netwerkprocessor verzorgt de wifi-communicatie.
- Op regel 376 en 377 wordt een zogenoemde `Display` geinitialiseerd en geopend. De globaal (op regel 64) gedefinieerde handle naar dit `Display` genaamd `dispay` kunnen we in het vervolg van het programma gebruiken om via de UART informatie weer te geven met behulp van de functie `Display_printf`.
- Op regel 388 wordt de functie `DisplayBanner` aangeroepen. Als argument wordt `APPLICATION_NAME` meegegeven. Deze is gedefinieerd op regel 42 en kan naar believen aangepast worden. De functie `DisplayBanner` is gedefinieerd op regel 354 tot en met 366 en verstuurt de in figuur 1 getoonde banner naar de UART (met een baudrate van 115200 Baud).



Figuur 1: Aan het begin van het programma wordt deze banner verstuurd via de UART.

- Op regel 396 wordt een *pthread* aangemaakt. Deze thread voert de functie `sl_Task` uit. Deze functie is gedefinieerd in de SDK en beheert de communicatie met de NWP.
- Op regel 403 wordt de NWP opgestart.
- Op regel 409 wordt gecontroleerd of de NWP in de zogenoemde *Station* mode staat en b.v. niet in de *Access Point* mode. Als de NWP niet in de Station mode is gestart, dan wordt geprobeerd om de mode aan te passen en wordt de NWP opnieuw gestart. Op regel 432 wordt de mode nogmaals gecontroleerd.

- Op regel 436 wordt de functie `Connect` aangeroepen. Deze functie is gedefinieerd op regel 329 tot en met 343. Deze functie gebruikt de eerder gedefinieerde `SSID_NAME`, `SECURITY_TYPE` en `SECURITY_KEY` om een wifi verbinding te openen. Als dit gelukt is wordt vanuit de SDK de call-back functie `SimpleLinkNetAppEventHandler` aangeroepen. Deze functie is gedefinieerd op regel 104 tot en met 154.
- Als het opzetten van de wifi-verbinding is gelukt, dan wordt op regel 128 tot en met 137 het verkregen IP-adres en het IP-adres van de Gateway getoond, zie figuur 2.



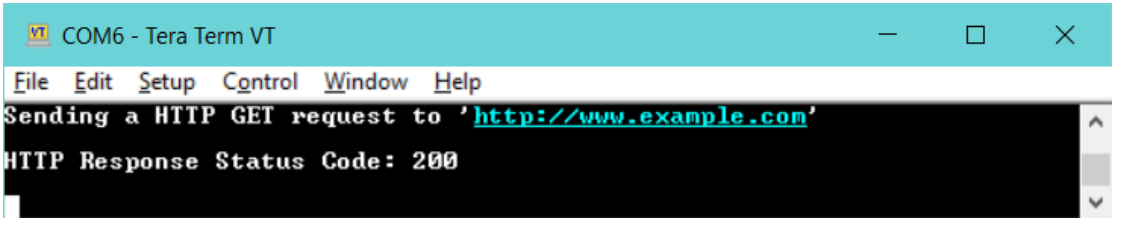
Figuur 2: Het opzetten van de wifi-verbinding is gelukt.

- Tot slot van dit deel van het programma wordt vervolgens op regel 144 een *pthread* aangemaakt. Deze thread voert de functie `httpTask` uit. Deze functie is op regel 66 **`extern`** gedefinieerd en bevindt zich dus in een ander bestand.

D Open het bestand `httpget.c`. Dit bestand bevat naast de functie `httpTask` nog een aantal andere functies en definities. Deze functie gebruikt de wifi-verbinding om met behulp van het HTTP-protocol en webpagina op te halen. Het is niet nodig om deze code geheel te begrijpen. Bepaalde delen moet je echter wel begrijpen en aan kunnen passen.

- Op regel 44 wordt de naam van de webserver, in dit geval `"http://www.example.com"` gedefinieerd als `HOSTNAME`.
- Op regel 45 wordt het *path* naar het op te vragen bestand, in dit geval `"/"` gedefinieerd als `REQUEST_URI`.
- Op regel 70 wordt de functie `HTTPClient_create()` aangeroepen. Hiermee wordt een `HTTPClient_Handle` genaamd `httpClientHandle` aangemaakt die vervolgens gebruikt kan worden om via het HTTP-protocol met een webserver te communiceren.

- Op regel 78 wordt de functie `HTTPClient_setHeader()` aangeroepen. Hiermee wordt de header van het HTTP-request gevuld.
- Op regel 87 wordt de functie `HTTPClient_connect()` aangeroepen. Hierdoor wordt verbinding met de in `HOSTNAME` gedefinieerde webserver gemaakt.
- Op regel 93 wordt de functie `HTTPClient_sendRequest()` aangeroepen. Hiermee wordt het bestand met de in `REQUEST_URI` gedefinieerde path opgevraagd.
- Op regel 106 wordt de HTTP-statuscode[7] naar de UART verzonden, zie figuur 3. De statuscode 200 geeft aan dat de aanvraag correct is afgehandeld door de server.



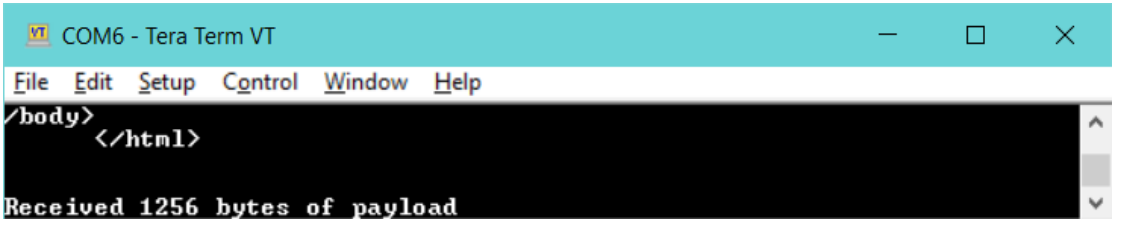
Figuur 3: Het opvragen van de webpagina is gelukt.

- In de **do-while** loop van regel 109 tot en met 120 wordt herhaaldelijk de functie `HTTPClient_readResponseBody()` aangeroepen. Deze functie kopieert steeds een stukje van het opgehaalde bestand in de array `data`. Deze array bevat 256 karakters (de grootte van de array is gedefinieerd met behulp van de constante `HTTP_MIN_RECV` op regel 47). Als deze functie succesvol is uitgevoerd, wordt het aantal gekopieerde karakters teruggegeven. Dit aantal wordt opgeslagen in de variabele `ret`. De functie vult ook de booleaanse variabele `moreDataFlag`, die aangeeft of er nog meer data beschikbaar is. Vervolgens wordt de ontvangen data naar de UART verzonden. Dit wordt herhaald totdat er geen data meer beschikbaar is. De inhoud van het ontvangen bestand is een in HTML[8] gecodeerde webpagina, zie figuur 4.
- Na afloop van de loop wordt het totaal aantal ontvangen karakters weergegeven, zie figuur 5 en wordt de verbinding met de webserver weer afgesloten door de functie `HTTPClient_disconnect()` aan te roepen op regel 124.

[7] Zie https://nl.wikipedia.org/wiki/Lijst_van_HTTP-statuscodes.

[8] Zie eventueel https://nl.wikipedia.org/wiki/HyperText_Markup_Language.

Figuur 4: Het begin van de opgevraagde webpagina.



Figuur 5: Na het einde van de opgevraagde webpagina wordt het totaal aantal karakters weergegeven.

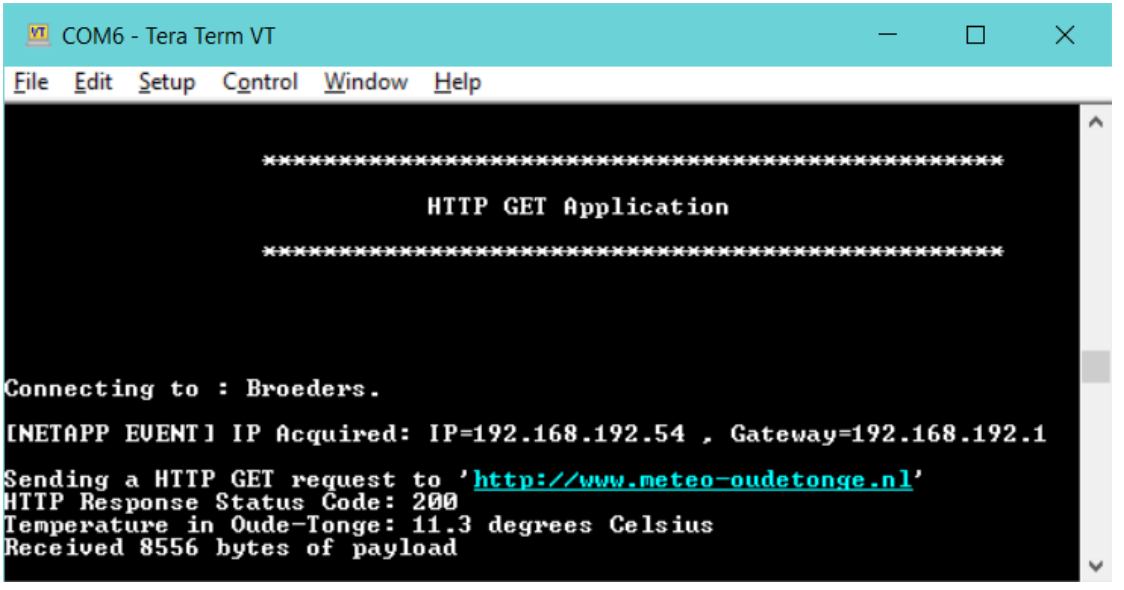
- Tot slot wordt de functie `HTTPClient_destroy()` aangeroepen. Hiermee wordt de regel 70 aangemaakte HTTP-client weer verwijderd.

Het opvragen van een statische webpagina is natuurlijk niet zo zinvol.

Stel dat we een embedded applicatie willen maken die het binnenklimaat regelt, maar daarbij ook rekening houdt met het buitenklimaat. We zouden dan extra sensors aan dit systeem kunnen toevoegen om het buitenklimaat te meten. We moeten er daarbij voor zorgen dat de sensoren juist geplaatst worden en we moeten ook voor een dataverbinding zorgen tussen de buiten geplaatste sensoren en het embedded systeem dat zich binnen bevindt. Het is duidelijk dat dit de kosten van deze applicatie verhoogt en het installatiegemak vermindert. Maar als de embedded applicatie kan beschikken over een internetverbinding (zoals onze CC3220S LaunchPad) dan is het helemaal niet nodig om het buitenklimaat zelf te meten. We kunnen dit dan ook 'opzoeken' op het world wide web. Als voorbeeld gaan we bekijken hoe de actuele temperatuur in Oude-Tonge opgehaald kan worden.

5.1.3 Actuele meteorologische gegevens voor Oude-Tonge kun je vinden op: `http://www.meteo-oudetonge.nl/updates/starttekst.html`. Het is vrij eenvoudig om bijvoorbeeld de actuele buitentemperatuur in Oude-Tonge uit de broncode van deze pagina te halen. Je kunt deze broncode bekijken door het volgende adres in de adresbalk van je

browser te typen: `view-source:http://www.meteo-oudetonge.nl/updates/start-tekst.html`. Vervang nu de code uit het bestand `httpget.c` met de code uit het bestand `httpget.c`. Run dit programma. Je ziet dat het programma de actuele temperatuur in Oude-Tonge weergeeft, zie figuur 6.



Figuur 6: De buitentemperatuur in Oude-Tonge.

De code in de functie `httpTask()` is, zoals je ziet, enigszins aangepast om er voor te zorgen dat de informatie waar we in geïnteresseerd zijn relatief eenvoudig uit de HTML-code kan worden gehaald.

Zo wordt de ingelezen data nu telkens afgesloten met een nul-karakter (`'\0'`), zodat deze als een C-string gebruikt kan worden. Op deze manier kunnen we gebruik maken van de functies uit `string.h`[9], bijvoorbeeld om de data te doorzoeken.

De temperatuur die we uit willen lezen bevindt zich in het volgende HTML-fragment:

```
<TR height=20>
  <TD Width="115"><STRONG><FONT face="Arial" ↩
 ↪ size="2">Temperatuur:</FONT></STRONG></TD>
  <TD Width="275" align=left> <b> <font face="Arial" size="2"> ↩
 ↪ 11.3</font>°C</b></TD>
```

[9] Zie https://en.cppreference.com/w/c/string/byte.

```
</TR>
```

Het programma voert de volgende stappen uit om de waarde van de temperatuur in te lezen in een variabele van het type **float**.

- Op regel 50 wordt de functie `strstr()` aangeroepen om te kijken of en waar de tekst `"Temperatuur:"` zich in de ingelezen data bevindt.
- Als deze tekst gevonden is, dan wordt vanaf die plaatst gezocht naar de tekst `"size=\"2\">"`, door de functie `strstr()` nogmaals aan te roepen op regel 53. Merk op dat de `"`-tekens in de zoektekst weergegeven moeten worden als een escape sequence `"\""`.
- Vervolgens wordt, als ook deze tekst gevonden is, op regel 56 de functie `sscanf()` aangeroepen om de waarde van de temperatuur in te lezen en op te slaan in de variabele `temperature`. De functie `sscanf()` begint te lezen op de positie aangewezen door `p2 + strlen(zoektekst2)`, dat is meteen *na* de gevonden tweede zoektekst.

De waarde van de variabele `temperature` wordt vervolgens met behulp van de functie `snprintf()` afgedrukt in een C-string en deze string wordt met behulp van de functie `Display_printf` naar de UART verzonden.

De data afkomstig van de webpagina wordt in brokjes van 332 karakters verwerkt. Toevallig zit de tekst `"Temperatuur:"` en de bijbehorende waarde van de temperatuur in hetzelfde brokje.

Pas het programma nu zo aan dat, in plaats van de temperatuur, de actuele luchtvochtigheid in Oude-Tonge wordt opgehaald en afgedrukt. De luchtvochtigheid (in %) moet daarbij worden opgeslagen in de variabele `humidity` van het type **int**.

Je hebt nu gezien hoe de SimpleLink SDK gebruikt kan worden om via het HTTP-protocol uit de applicatielaag van het OSI-model[10] een webpagina op te vragen. De SimpleLink SDK biedt ook ondersteuning bij het gebruik van protocollen uit onderliggende lagen van het OSI-model. We gaan nu gebruik maken van het ICMP-protocol[11] om de wifi verbinding van je LaunchPad te testen. Dit protocol wordt ook gebruikt door het bekende ping-commando[12].

[10] Zie eventueel https://nl.wikipedia.org/wiki/OSI-model.

[11] Zie https://nl.wikipedia.org/wiki/Internet_Control_Message_Protocol.

[12] Zie eventueel https://nl.wikipedia.org/wiki/Ping_(netwerk).

5.1.4 Voer de volgende stappen uit:

- Kopieer het project `httpget_CC3220S_LAUNCHXL_tirtos_ccs` in CCS naar een nieuw project genaamd `ping`.
- Hernoem het bestand `httpget.c` naar `ping.c`.
- Zorg ervoor dat in `platform.c` in plaats van de functie `httpTask` de functie `pingTask` wordt uitgevoerd door de pthread die wordt gestart in `SimpleLinkNetAppEventHandler`.
- Verander de `APPLICATION_NAME` naar `"PING"`.
- Vervang nu de code uit het bestand `ping.c` met de code uit het bestand `ping.c`.

Je ziet dat dit programma met behulp van de functie `sl_NetAppPing()` een ping-commando uitvoert. Het ping-commando stuurt een ICMP-pakketje[13] naar een IP-adres en wacht op een echo[14]. Als een antwoord wordt ontvangen dan weet je dat de host[15] benaderbaar is via het IP-adres.

Pas de bestaande code aan om nu niet example.com te pingen, maar het IP-adres van jouw pc. Het IP-adres van jouw pc kun je vinden door een command window te openen (start het programma `cmd.exe`) en het commando `ipconfig` in te typen. Als dit lukt, weet je dat jouw Launchpad en jouw computer beide succesvol zijn verbonden met jouw wifi-netwerk en dat communicatie mogelijk is, zie figuur 7.

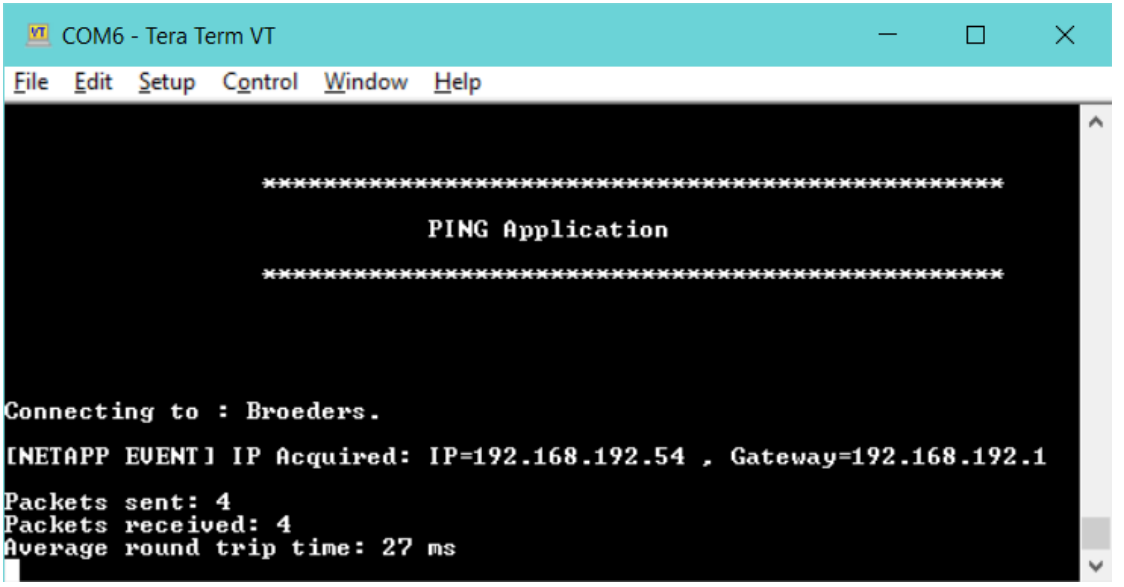
In het vervolg van deze opdracht ga je gebruik maken van het TCP-protocol uit de transportlaag van het OSI-model. Dit protocol kun je gebruiken om data te versturen naar of te ontvangen van een specifieke host op een netwerk.

Bij communicatie met behulp van TCP/IP wordt onderscheid gemaakt tussen servers en clients. De server heeft als taak om op een verbinding te wachten en als een verbinding wordt gemaakt, te zeggen met welke host de client is verbonden. De client heeft als taak om verbinding te maken met een server en het antwoord van de server te laten zien. We maken, voor deze opdracht, de afspraak dat de server op poort 2000 wacht op een verbinding.

[13] ICMP staat voor Internet Control Message Protocol. Het wordt voornamelijk gebruikt door netwerkapparaten, inclusief routers, om foutmeldingen en statusinformatie te verzenden, zie eventueel https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol.

[14] Een ICMP ECHO_REQUEST pakket en ECHO_RESPONSE antwoord.

[15] Elk apparaat met een eigen IP-adres wordt een host genoemd.



Figuur 7: Een succesvol uitgevoerd ping-commando.

Omdat zowel de client als de server met sockets werken, wordt dit eerst uitgelegd. In theorie is een socket een combinatie van een IP-adres en een poortnummer.

In de BSD socket bibliotheek[16] is een socket niets meer dan een beschrijving van de verbinding die eigenlijk altijd in combinatie met een adres-structure[17] wordt gebruikt. Open de documentatie van de Simplelink SDK Socket module. Dit zul je nodig hebben bij deze opdracht. Alle socket-functies van de SimpleLink SDK beginnen met `sl_` en daarna een hoofdletter. De standaard POSIX-versies van deze functies beginnen zonder deze prefix en met een kleine letter.

5.1.5 Voer de volgende stappen uit:

- Kopieer het project `ping` in CCS naar een nieuw project genaamd `socket_server`.
- Hernoem het bestand `ping.c` naar `socket_server.c`.

[16] BSD staat voor Berkeley Software Distribution. Dit is een (oude) UNIX-distributie waarin de sockets voor het eerst geïmplementeerd werden. Tegenwoordig is deze BSD socket library opgenomen in de POSIX standaard, zie eventueel `https://en.wikipedia.org/wiki/Berkeley_sockets`.

[17] Dit wordt verderop uitgelegd, zie listing 1.

- Zorg ervoor dat in `platform.c` in plaats van de functie `pingTask` de functie `socketServerTask` wordt uitgevoerd door de pthread die wordt gestart in `SimpleLinkNetAppEventHandler`.
- Verander de `APPLICATION_NAME` naar `"SOCKET SERVER"`.
- Vervang nu de functienaam `pingTask` in het bestand `socket_server.c` door de naam `socketServerTask` en verwijder alle code van deze functie behalve `return NULL;`.

In deze en de volgende opdrachten gaan we stap voor stap de benodigde code in de `socketServerTask` invullen.

Gebruik de functie `sl_Socket()` om een socket aan te maken met de volgende opties:

- Gebruik `SL_AF_INET` als domain, zodat IPv4 wordt gebruikt.
- Gebruik `SL_SOCK_STREAM` als type en `SL_IPPROTO_TCP` als protocol om een TCP verbinding op te zetten.

De returnwaarde van `sl_Socket()` is een zogenoemde 'handle' waarmee we de socket in het vervolg van het programma kunnen identificeren. Sla de returnwaarde op in een variabele, zodat je deze later kunt gebruiken. Als de returnwaarde negatief is, dan is er een fout opgetreden. Zorg ervoor dat het programma in een oneindige loop terechtkomt als `sl_Socket()` een foutcode teruggeeft. Bij het debuggen kun je dan op pauze drukken als het programma vastloopt en zie je waar de fout is opgetreden en welke foutcode is teruggegeven. In een professioneel programma wil je natuurlijk niet dat het programma vastloopt maar beëindig je het programma met een nette foutmelding.

Bij een socket hoort ook een beschrijving van het ip-adres en het poortnummer. Deze informatie staat in een structure. Voor een IPv4 socket wordt de structure gebruikt die is weergegeven in listing 1.

Omdat poortnummers en IP-adressen door iedereen op het internet begrepen moet worden, is het belangrijk om deze om te zetten naar een standaard formaat. Dan kunnen big- en little-endian[18] systemen nog steeds met elkaar communiceren!

[18] Zie eventueel `https://nl.wikipedia.org/wiki/Endianness`.

```
/* Socket address, Internet style. */

typedef struct SlSockAddrIn_t
{
    _u16        sin_family;  /* Internet Protocol */
    _u16        sin_port;    /* Address port      */
    SlInAddr_t  sin_addr;    /* Internet address  */
    _i8         sin_zero[8]; /* Not used.         */
} SlSockAddrIn_t;
```

Listing 1: Structure zoals gedefinieerd in de SDK.

Deze omzetting vindt plaats door middel van een aantal functies voor het versturen, en ontvangen van poortnummers en IP-adressen:

- Host to network short/long: `sl_Htons(poort)` en `sl_Htonl(ip)`
- Network to host short/long: `sl_Ntohs(poort)` en `sl_Ntohl(ip)`

De short variant moet gebruikt worden voor 16-bits variabelen en de long variant moet gebruikt worden voor 32-bits variabelen.

Gebruik dus bijvoorbeeld:

```
mijnAdres.sin_port = sl_Htons(2000);
```

In eerste instantie ga je de LaunchPad als server programmeren en je pc als client gebruiken.

5.1.6 Maak een variabele `mijnAdres` aan van het in de SDK gedefinieerde type `SlSockAddrIn_t`, zie listing 1 en vul deze in. Omdat we TCP/IP als protocol gebruiken, moet je in het veld `sin_family` de waarde `SL_AF_INET` invullen. Als poortnummer gebruik je `2000`, zoals afgesproken met de client. De variabele `sin_addr` is ook een structure, met een eigen variabele `s_addr`. Omdat jouw programma een server moet opzetten, kies je als IP-adres `SL_INADDR_ANY`. `SL_INADDR_ANY` geeft aan dat inkomend verkeer van elk IP-adres wordt geaccepteerd. Vergeet niet om het poortnummer en het IP-adres om te zetten naar het netwerkformaat.

Gebruik nu de functie `sl_Bind()` om het zojuist gedefinieerde socket-adres `mijnAdres` te koppelen met de al eerder aangemaakte socket. Let er op dat de tweede parameter een pointer is! Dit zorgt ervoor dat je verschillende structures kan doorgeven aan deze functie. Voor IPv6 moet bijvoorbeeld een structure van het type `SlSockAddrIn6_t` wor-

den gebruikt. Omdat het type van de pointer `SlSockAddrIn_t *` die je aan de `sl_Bind`-functie doorgeeft, niet hetzelfde is dan het verwachte parametertype `SlSockAddr_t *` moet de doorgegeven pointer nog gecast worden:

```
sl_Bind(..., (SlSockAddr_t *)&mijnAdres, ...);
```

De lengte van de structure kun je achterhalen met **`sizeof`**`(SlSockAddrIn_t)`. Controleer of de functie `sl_Bind` succesvol is uitgevoerd door de returnwaarde te controleren.

Een server heeft als taak te luisteren naar inkomende verbindingsaanvragen. Hier zijn een tweetal functies voor beschikbaar, die in volgorde moeten worden aangeroepen.

- De functie `sl_Listen()` zorgt ervoor dat er actief op de poort wordt *geluisterd* waardoor clienten een verbinding kunnen aanvragen en hierop antwoord krijgen.
- De functie `sl_Accept()` accepteert een binnenkomende aanvraag.

5.1.7 Gebruik `sl_Listen()` om te gaan luisteren op de socket. De `backlog` parameter bepaalt het aantal clients dat we tegelijk willen accepteren, zet deze op `1`. Controleer ook nu weer de returnwaarde.

De `sl_Accept()` functie zal een nieuwe socket (handle) teruggeven als een client verbinding maakt. Deze nieuwe (lokale) socket is voortaan het verbindingspunt naar de client toe. Iedere client krijgt zijn eigen socket, op deze manier kun je blijven luisteren op poort 2000 naar nieuwe clients!

Gebruik `sl_Accept()` om te wachten op een client en sla de return waarde op in de variabele `_i16 client`. Aan de functie `sl_Accept` kun je een socket-adres meegeven (zoals bij `sl_Bind`). In deze socket-adres structure zal door de functie `sl_Accept` het IP-adres en poortnummer van de client worden ingevuld. Deze informatie hebben we in dit geval niet nodig en daarom kun je als tweede `NULL` en als derde parameter `0` meegeven.

Data ontvangen of versturen gaat met twee functies: `sl_Recv()` en `sl_Send()`. Omdat een (draadloos) netwerk onverwachte vertragingen op kan leveren, werken deze twee functies met buffers. De `sl_Send()` functie dumpt de te versturen data in een verzendingsbuffer. De `sl_Recv()` wacht tot het opgegeven aantal bytes is ontvangen en zet het in de opgegeven buffer.

5.1.8 Gebruik `sl_Send()` om jouw volledige naam (als een C-string) te versturen zodra een client verbinding heeft gemaakt. Sluit daarna de verbinding door de functie `sl_Close()` aan te roepen. Zorg ervoor dat het programma hierna de volgende client accepteert, enzovoorts.

Run het serverprogramma op je LaunchPad en test het door één of meer van de volgende clients te gebruiken:

- Het Python programma gegeven in listing 2 maakt verbinding met de server en print het ontvangen bericht. Vervang het IP-adres met het IP-adres van jouw Launchpad. Het gebruik van het `with`-statement zorgt ervoor dat de socket-verbinding netjes afgesloten wordt als het `with`-statement uitgevoerd is. De ontvangen data moet nog worden omgezet naar de juiste karaktercodering. Bij het gebruik van CCS onder het Windows-OS wordt default de Cp1252 karaktercodering[19] gebruikt. Als het goed is ziet de uitvoer van het programma gegeven in listing 2 er vergelijkbaar uit met figuur 8.

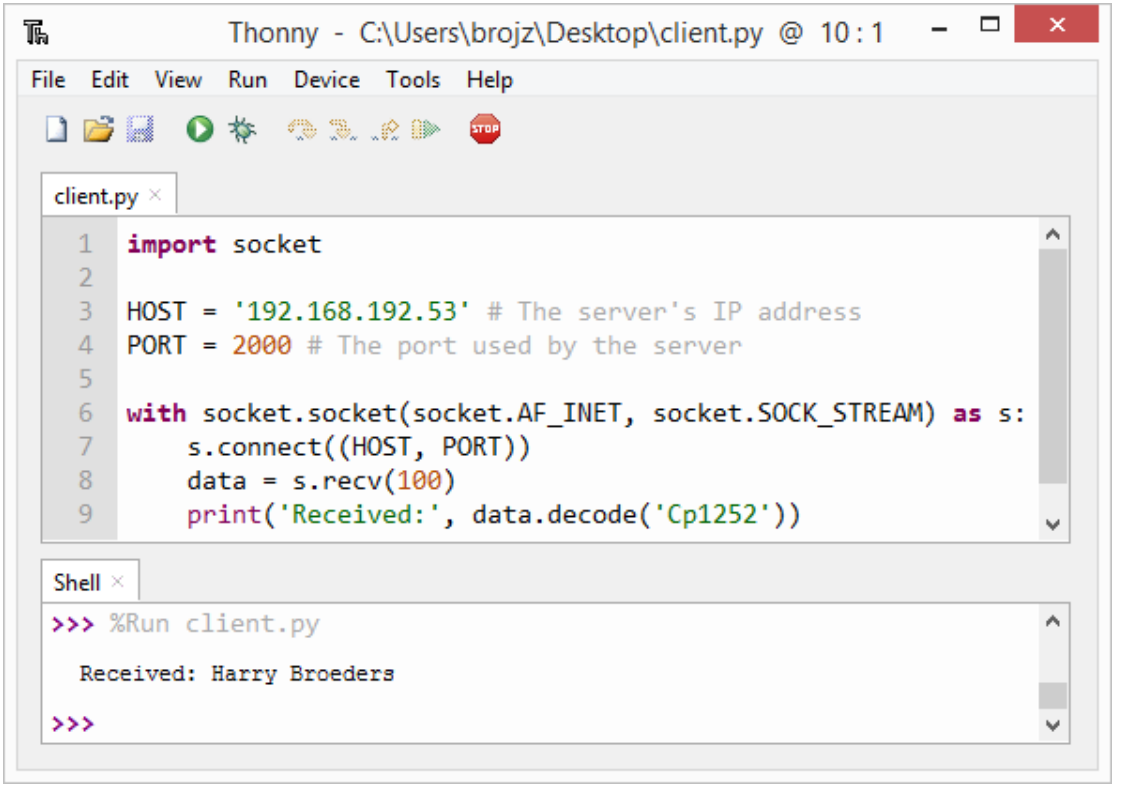
```
import socket

HOST = '192.168.192.53' # The server's IP address
PORT = 2000 # The port used by the server

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    data = s.recv(100)
    print('Received:', data.decode('Cp1252'))
```

Listing 2: Een clientprogramma in Python: `client.py`.

- Je kunt het programma telnet[20] als client gebruiken. Start het programma `cmd.exe` en type het commando `telnet` in. Type vervolgens het telnet-commando `open 192.168.192.53 2000` in, waarbij je uiteraard het IP-adres van jouw LaunchPad gebruikt. Als het goed is ziet de uitvoer er vergelijkbaar uit met figuur 9. Je kunt de server dus via een telnet-programma eenvoudig aanspreken met elk

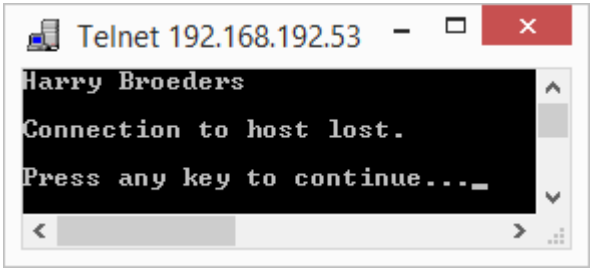
[19] Zie eventueel: https://en.wikipedia.org/wiki/Windows-1252.

[20] Het programma telnet is standaard aanwezig in Windows maar moet nog wel enabled worden. Zie: https://social.technet.microsoft.com/wiki/contents/articles/38433.windows-10-enabling-telnet-client.aspx.



Figuur 8: De uitvoer van het programma gegeven in listing 2.

apparaat dat op je WLAN aanwezig is. Op je Android telefoon kun je bijvoorbeeld het programma ConnectBot[21] gebruiken, zie figuur 10

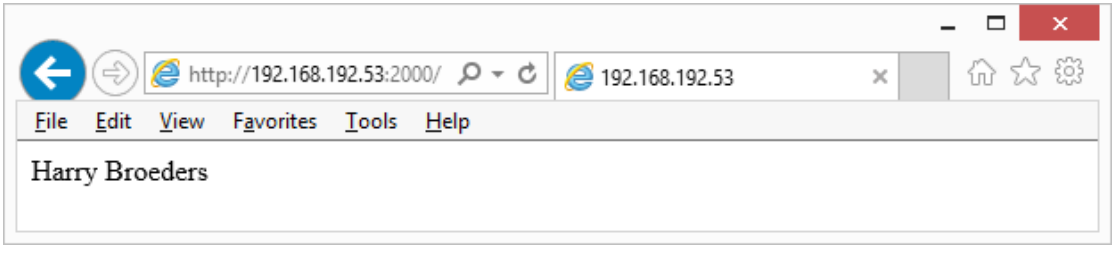


Figuur 9: De uitvoer van het telnetcommando: `open 192.168.192.53 2000`.

[21] Zie https://connectbot.org/.

Figuur 10: Een telnetsessie met poort 2000 van de server op een Android telefoon.

- Je kunt ook sommige webbrowsers[22] als client gebruiken. Open bijvoorbeeld Internet Explorer en type het volgende in de adresbalk: `http://192.168.192.53:2000`, waarbij je uiteraard het IP-adres van jouw Launchpad gebruikt. Als het goed is ziet de uitvoer er vergelijkbaar uit met figuur 11. Een webbrowser verwacht uiteraard een HTTP-response te ontvangen, maar Internet Explorer geeft de ruwe data weer als geen HTTP-response wordt ontvangen. Je kunt de server dus via een webbrowser eenvoudig aanspreken met elk apparaat dat op je WLAN aanwezig is. Dus bijvoorbeeld ook met je mobiele telefoon.



Figuur 11: Adres `http://192.168.192.53:2000` zoals weergegeven in een webbrowser.

Je gaat nu de rollen omkeren. Op de pc ga je het server programma gegeven in listing 3 uitvoeren. De LaunchPad ga je nu als client gebruiken.

[22] Bijvoorbeeld Internet Explorer of Firefox. Edge, Safari en Chrome geven een foutmelding.

5.1.9 Voer de volgende stappen uit:

- Kopieer het project `socket_server` in CCS naar een nieuw project genaamd `socket_client`.
- Hernoem het bestand `socket_server.c` naar `socket_client.c`.
- Zorg ervoor dat in `platform.c` in plaats van de functie `socketServerTask` de functie `socketClientTask` wordt uitgevoerd door de pthread die wordt gestart in `SimpleLinkNetAppEventHandler`.
- Verander de `APPLICATION_NAME` naar `"SOCKET CLIENT"`.
- Vervang nu de functienaam `socketServerTask` in het bestand `socket_client.c` door de naam `socketClientTask` en verwijder de code die specifiek is voor de server: alles vanaf `sl_Bind` tot en met `sl_Close`.

```
import socket

HOST = '0.0.0.0' # accept any IP-adres
PORT = 2000 # Port to listen on

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    while True:
        conn, addr = s.accept()
        with conn:
            print('Connected by', addr)
            conn.sendall(b'Harry Broeders\0')
```

Listing 3: Een serverprogramma in Python: `server.py`.

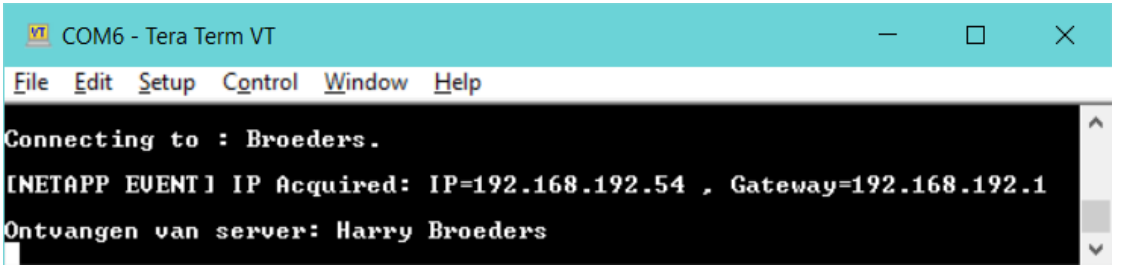
Een client heeft als taak verbinding te maken met een server. Dit kan met de functie `sl_Connect()`. Gebruik `sl_Connect()` om verbinding te maken met de server die je later op je pc gaat uitvoeren. De client moet dus verbinden met het IP-adres van je pc. Gebruik hiervoor de `SL_IPV4_VAL(a,b,c,d)` macro.

Let er op dat de tweede parameter van `sl_Connect()` een pointer is! Dit zorgt ervoor dat je verschillende structures kan doorgeven aan deze functie. Voor IPv6 moet bijvoorbeeld een structure van het type `SlSockAddrIn6_t` worden gebruikt. Omdat het type van de pointer `SlSockAddrIn_t *` die je aan de `sl_Connect`-functie doorgeeft, niet hetzelfde is dan het verwachte parametertype `SlSockAddr_t *` moet de doorgegeven pointer nog gecast worden:

```
sl_Connect(..., (SlSockAddr_t *)&mijnAdres, ...);
```

De lengte van de structure kun je achterhalen met `sizeof(SlSockAddrIn_t)`. Controleer of de functie `sl_Connect` succesvol is uitgevoerd door de returnwaarde te controleren.

Gebruik `sl_Recv()` om een bericht te ontvangen nadat verbinding is gemaakt met de server. Zorg ervoor dat het buffer dat je aan `sl_Recv` meegeeft groot genoeg is om elke mogelijke naam in op te slaan. Een buffer van 100 karakters is meer dan voldoende. Druk de ontvangen C-string af. Sluit daarna de verbinding door de functie `sl_Close()` aan te roepen. Als het goed is ziet de uitvoer van de client en de server er vergelijkbaar uit met figuur 12 en figuur 13.



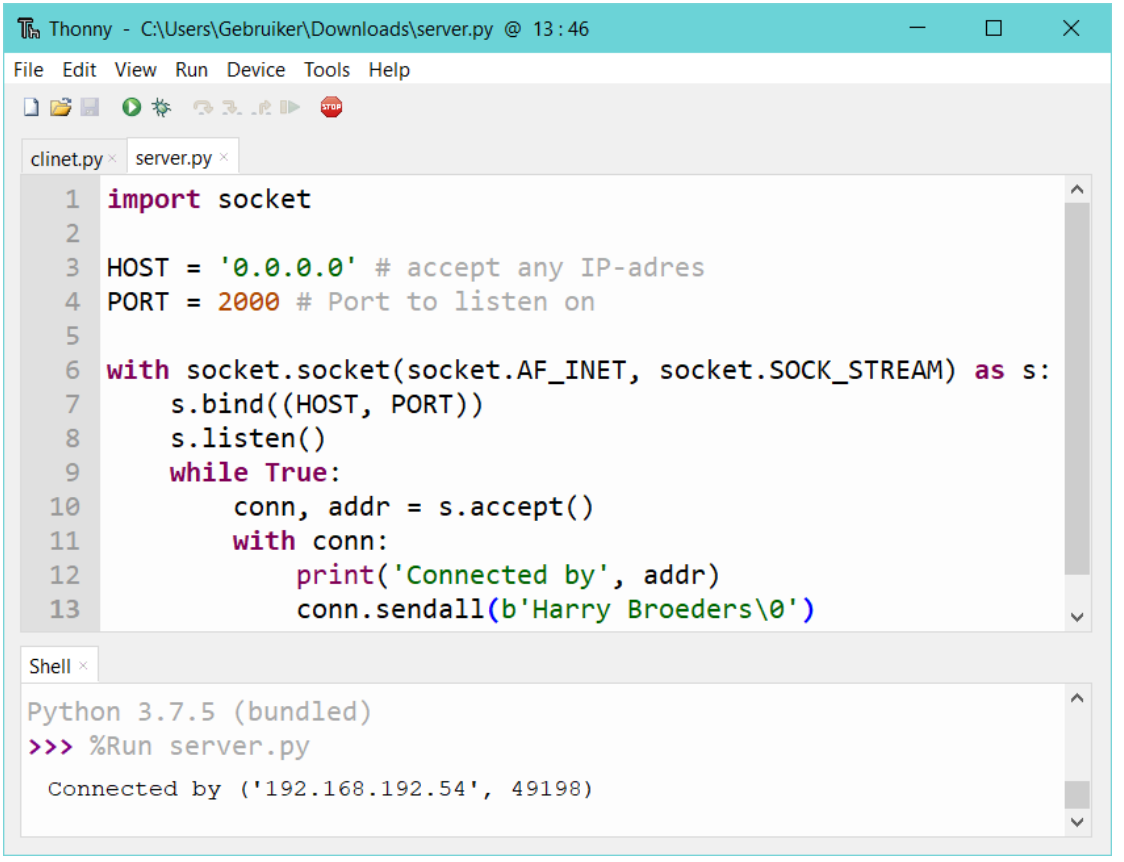
Figuur 12: De uitvoer van het clientprogramma op de LaunchPad.

5.1.10 Als laatste zullen we de code uitbreiden, zodat de client een ledje kan schakelen op de server.

A Pas het client Python programma uit listing 2 aan zodat het karakter `'0'` naar de server wordt gestuurd wanneer een `0` wordt ingetypt op het toetsenbord en het karakter `'1'` naar de server wordt gestuurd wanneer een `1` wordt ingetypt.

B Pas het project `socket_server` zo aan dat gewacht wordt op data, zodra een client verbonden is. Gebruik de functie `sl_Recv` om te wachten op data. Als de ontvangen data gelijk is aan `'1'`, dan zet de server de rode led aan. Als de ontvangen data gelijk is aan `'0'`, dan zet de server de rode led uit.

Je hebt in dit lab gezien hoe een socket kan worden aangemaakt, en hoe je hiermee een verbinding opzet.



Figuur 13: De uitvoer van het programma gegeven in listing 3.

Volgend lab wordt gekeken naar een veelgebruikt IoT-protocol wat draait bovenop de TCP/IP-laag.