**RP2 T13 op maandag 21-6 Intro IoT**

**IoT =** verbinden van elektronische apparaten en data uitwisselen via :

WiFi,

LAN en internet,

maar ook bv via GSM netwerk en LoraWan, Sigfox etc.

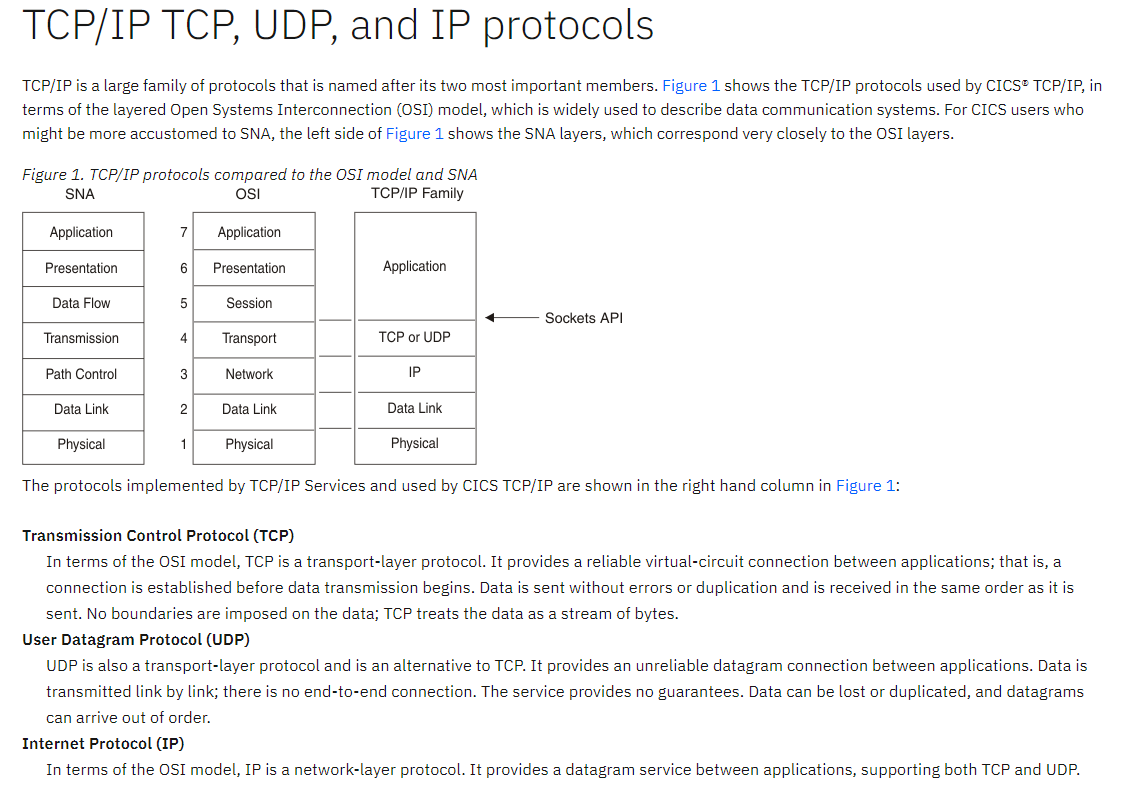
**Elektronische apparaten =**

embedded devices microcontroller gebaseerd ( bv ESP32) ,

App op smart device,

je PC en Laptop,

cloud server(s)

**Client maakt verbinding met server via sockets**

Wat is TCP / UDP?

[VPS](https://www.transip.nl/knowledgebase/tag/7-vps/)[Basis](https://www.transip.nl/knowledgebase/tag/35-basis/)[Netwerk](https://www.transip.nl/knowledgebase/tag/61-netwerk/)

Wanneer je met netwerken werkt, kom je al snel de termen TCP & UDP tegen, maar wat betekenen deze termen?

Zowel TCP en UDP zijn netwerkprotocollen, die gebruikt worden voor het versturen van data bits (packets) over het internet, bovenop het reguliere internet protocol (i.e. in de transpoortlaag/applicatielaag in de IP-stack).

TCP

TCP staat voor Transmission Control Protocol en wordt het meeste gebruikt; als je bijvoorbeeld vanaf je eigen computer op een link op een webpagina klikt, stuurt je browser TCP packets over het internet naar de server die de website host, en de betreffende server stuurt TCP packets terug.

Deze packets krijgen een getal toegewezen, zodat de ontvanger ze in de juiste volgorde ontvangt. TCP controleert daarnaast ook de data die verstuurd wordt, dus de server stuurt ook berichten terug naar de verzender om te bevestigen dat het je packets heeft ontvangen en dat er geen fouten in zitten. Krijgt je computer een onjuist antwoord terug? Dan worden de packets opnieuw gestuurd.

Naast webverkeer wordt TCP bijvoorbeeld ook gebruikt voor downloaden en het (niet live) streamen van video.

UDP

UDP staat voor User Datagram Protocol, waarbij een Datagram hetzelfde is als een packet. UDP werkt hetzelfde als TCP, maar zonder de error controles die TCP uitvoert. Hierdoor is UTP dus sneller dan TCP, maar ook minder betrouwbaar in het versturen van data.

UDP wordt dan ook vooral gebruikt wanneer snelheid belangrijker is dan een 100% foutloze verbinding. Dit is bijvoorbeeld het geval bij het doorsturen van server beschikbaarheid, live voice en video streaming en online gaming en live streams.

TCP verkeer is dus tweerichtingsverkeer waarbij voornamelijk de foutloze overdracht centraal staat en UDP eenrichtingsverkeer, waar snelheid de prioriteit heeft.

**Sockets**

A **network socket** is a software structure within a [network node](https://en.wikipedia.org/wiki/Node_(networking)) of a [computer network](https://en.wikipedia.org/wiki/Computer_network) that serves as an endpoint for sending and receiving data across the network

**Testen van TCP socketconnectie met Hercules :**

**Download en installeer Hercules >** [**https://www.hw-group.com/software/hercules-setup-utility**](https://www.hw-group.com/software/hercules-setup-utility)

**alternatieven :** [**https://www.downloadcloud.com/tcp-test-tool.html**](https://www.downloadcloud.com/tcp-test-tool.html)

**socketTest :** [**https://sourceforge.net/projects/sockettest/files/latest/download**](https://sourceforge.net/projects/sockettest/files/latest/download)

**Test 1:**

Open 2 instances van Hercules op dezelfde computer

Gebruik 1-ste instance als server en luister op bv poort 52000

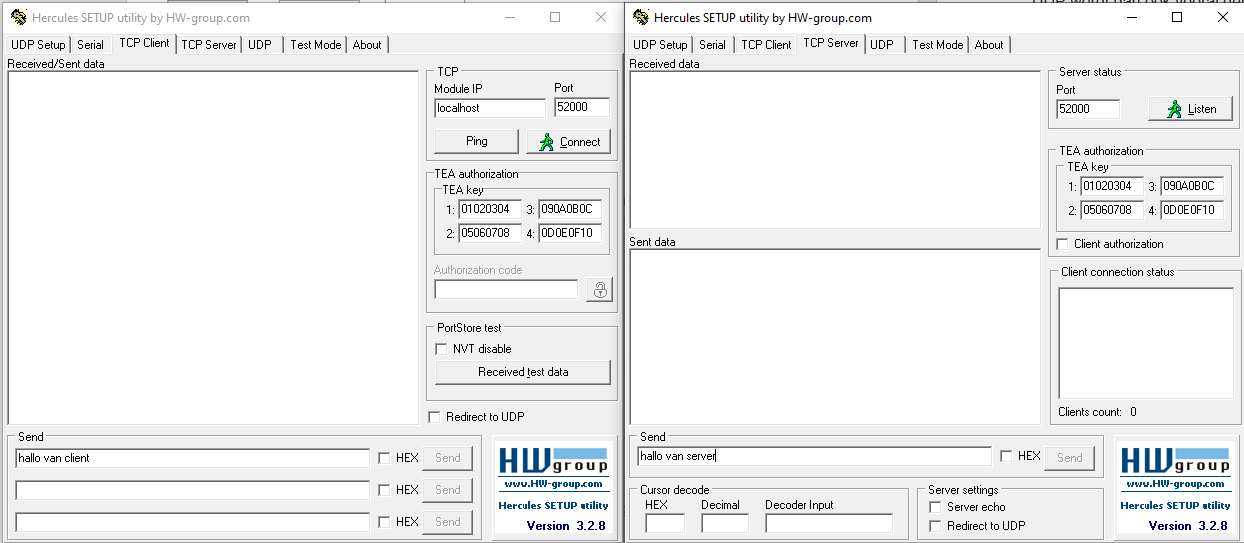
Gebruik 2-de instance als client en connecteer naar je IP van je computer of localhost en poort 52000

IP van je computer via cmd / ipconfig voor windows

Zet de server in Listen mode

Connecteer de client

Zend berichten van client naar server en omgekeerd

****

**Test 2:**

**Kan je meerdere clients connecteren met 1 server?**

**Wat gebeurt er als de server iets terug stuurt? Bij welke client komt het terecht?**

**Is dat altijd zo?**

**Test 3:**

**Zou je een client op je ene computer kunnen starten en een server op je andere computer?**

**Test 4:**

**Zou je bij jou een server kunnen starten die data ontvangt van een client bij een andere cursist?**

**Hoe doe je dat?**

**Test 5 :**

**Wat gebeurd er als je je chrome browser laat connecteren met je Hercules server?**

**Wat zie je allemaal? Hoe noem je dat protocol?**

**Test 6:**

**Kan je hallo van voornaam achternaam terug sturen en tonen in de browser?**

**Indien ja, hoe heb je dat gedaan?**

**Indien nee, laat ons het http protocol eens bekijken op de volgende pagina.**

**HTTP PROTOCOL https://nl.wikipedia.org/wiki/Hypertext\_Transfer\_Protocol**

**Hypertext Transfer Protocol** (**HTTP**) is het [protocol](https://nl.wikipedia.org/wiki/Protocol) voor de communicatie tussen een webclient (meestal een [webbrowser](https://nl.wikipedia.org/wiki/Webbrowser) of een app) en een [webserver](https://nl.wikipedia.org/wiki/Webserver). Dit protocol wordt niet alleen veel op het [wereldwijde web](https://nl.wikipedia.org/wiki/Wereldwijd_web) gebruikt, maar ook op lokale netwerken (we spreken dan van een [intranet](https://nl.wikipedia.org/wiki/Intranet)).

In HTTP is vastgelegd welke vragen (de Engelse term hiervoor is requests) een cliënt aan een server kan stellen en welke antwoorden (de Engelse term is responses) een webserver daarop kan teruggeven. Elke vraag bevat een [URL](https://nl.wikipedia.org/wiki/Uniform_Resource_Locator) die naar een webcomponent of een statisch object zoals een webpagina of plaatje verwijst.

## HTTP-requests

Een HTTP-request bestaat uit de requestsoort, de URL, de headervelden (koptitelvelden) en eventueel een inhoud. Een overzicht van de HTTP-requestmethoden:

* GET – Ontvang het document gespecificeerd door de URL.
* HEAD – Ontvang alleen de headers van het op te vragen document.
* POST – Zend gegevens naar de server.
* PUT – Vervang het document op de server door de verzonden data.
* DELETE – Verwijder het document.
* TRACE – Retourneert de aanvraag zodat een client kan zien welke wijzigingen of aanvullingen zijn gemaakt door tussenstations.
* OPTIONS – Vraag de mogelijkheden op dit niveau aan van de server.
* CONNECT – Vervangt de verbinding door een transparante TCP-/IP-tunnel, om bijvoorbeeld SSL-versleutelde communicatie (HTTPS) via een onversleutelde HTTP proxy te ondersteunen.
* PATCH – Gedeeltelijke modificatie van het document (vervang een deel door de verzonden data).

De belangrijkste headervelden zijn:

* Content-Length: Geeft de lengte van de inhoud aan.
* Location: Indien de server de webbrowser naar een andere pagina doorverwijst wordt hierbij het benodigde pad gevoegd.
* Server: Omvat een beknopte beschrijving van de serversoftware (bijvoorbeeld "Apache/2.2.9 (Win32) PHP/5.2.6").
* User-Agent: Geeft informatie over de aanvrager. Dit is meestal een webbrowser.
* Date: De datum en het tijdstip waarop het document verzonden is.
* Host\*: Omdat het pakket naar een [IP-adres](https://nl.wikipedia.org/wiki/IP-adres) wordt gestuurd weet de server niet op welk domein men aan het surfen is. In HTTP 1.1 is het verplicht om met deze header het domein mee te delen.

GET /wiki/Hoofdpagina HTTP/1.1

Host: nl.wikipedia.org

Connection: close

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; nl; rv:1.8.0.3) Gecko/20060426 Firefox/1.5.0.3

Accept: text/xml,text/html,text/plain,image/png,image/jpeg,image/gif

Accept-Charset: ISO-8859-1,utf-8

Een reactie op bovenstaande aanvraag kan er dan als volgt uitzien:

## HTTP-responses

HTTP/1.1 200 OK

Date: Thu, 08 Apr 2004 18:24:33 GMT

Server: Apache/1.3.29 (Unix) PHP/4.3.4

X-Powered-By: PHP/4.3.4

Content-Language: nl

Content-Type: text/html; charset=iso-8859-1

X-Cache: MISS from wikipedia.org

Connection: close

Content-Type: text/html

Content-Length: 49

<html>

<head>

</head>

<body>

</body>

</html>

Een HTTP-response bestaat uit een resultaatcode, headervelden en een body (de boodschap). De resultaatcode bestaat uit minimaal drie cijfers. Het eerste cijfer is het belangrijkste:

* 1xx: een mededelende boodschap van de webserver die nog gevolgd zal worden door andere data
* 2xx: een boodschap van de server dat de gevraagde actie succesvol is afgehandeld
* 3xx: een "redirect" naar een andere locatie, om wat voor reden ook
* 4xx: een foutboodschap die door de client veroorzaakt is, zoals het verkeerd typen van een URL
* 5xx: een foutboodschap die door de server veroorzaakt is, zoals een fout in een [CGI](https://nl.wikipedia.org/wiki/Common_Gateway_Interface)-[script](https://nl.wikipedia.org/wiki/Scripttaal)
* 6xx: een [proxyfout](https://nl.wikipedia.org/wiki/Proxyserver) is opgetreden

De meest voorkomende resultaatcodes zijn:

* 200 OK – Het gevraagde document is succesvol opgevraagd.
* 304 Not Modified – T.o.v. de versie in de [cache](https://nl.wikipedia.org/wiki/Cache_(tijdelijk_geheugen)) is de pagina niet gewijzigd.
* 400 Bad Request - De gebruiker heeft een fout gemaakt in het verzoek waardoor deze niet verwerkt kan worden.

**Hoe kan je met python een TCP server maken?**

# socket basic server example

import socket

import sys

# Create a TCP/IP socket

sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

#Then bind() is used to associate the socket with the server address. In this case, the address is localhost, referring to the current server, and the port number is 10000.

# Bind the socket to the port

server\_address = ('localhost', 53000)

print ('starting up on %s port %s' % server\_address)

sock.bind(server\_address)

#Calling listen() puts the socket into server mode, and accept() waits for an incoming connection.

# Listen for incoming connections

sock.listen(1)

while True:

# Wait for a connection

print ( 'waiting for a new connection')

connection, client\_address = sock.accept()#accept() returns an open connection between the server and client, along with the address of the client. The connection is actually a different socket on another port (assigned by the kernel).

#Data is read from the connection with recv() and transmitted with sendall().

print ( 'connection from', client\_address)

# Receive the data in small chunks and retransmit it

while True:

try:

data = connection.recv(4096)

print('received "%s"' % data)

if data:

pass

else:

print ( 'client closed the connection', client\_address)

break

except Exception as e:

print("e=",e)

break

# Clean up the connection

connection.close()

print("server closed his connection")

**Hoe kan je met python een TCP client maken?**

import socket

PORT = 53000

FORMAT = "utf-8"

DISCONNECT\_MESSAGE = "disconnect!"

#SERVER= "192.168.0.172"

SERVER= "127.0.0.1"

ADDR=(SERVER, PORT)

client = socket.socket(socket.AF\_INET , socket.SOCK\_STREAM)

client.connect(ADDR)

def send\_to\_server(msg):

message=msg.encode(FORMAT)

message\_length = len(message)

client.sendall(message)

while True:

i=input("Wat wil je naar server zenden? (STOP) to finish>")

if i !="STOP":

send\_to\_server(i)

else:

break

client.close()

**Opmerking :**

Sommige functies blokkeren, ook het werken met meerdere clients vergt enig gepuzzel!

**Test 7 : Stuur vanuit Hercules het commando ON/OFF om een led op je RaspberryPi aan uit te schakelen.**

**Test 8 : Stuur vanuit je RP een commando naar Hercules**

**Test 9: Stuur van je PC ON/OFF via een python client naar een python server op je RP**

**HTML +CSS +JS**

**HTM,L CSS ,JAVASCRIPT basics worden aangeleerd in module 3.**

**Dit laat ons toe om webpagina’s te designen die onze metingen tonen of onze apparaten besturen.**

**Dynamische webpages**

**Ajax**

**Webscokets**

**MIT APP INV extenties**

**TCP/IP**

**MQTT**

**API’s**

**Wat is een API?**

**De betekenis van een API is letterlijk Application Programming Interface. Een API maakt het mogelijk om andere systemen en software een ‘ingang’ tot het systeem te bieden. Met deze ingang kunnen verschillende systemen met elkaar communiceren en gegevens uitwisselen.**

**API met Postman**

**Weer**

<https://openweathermap.org/api>

<https://openweathermap.org/current>

<https://api.openweathermap.org/data/2.5/weather?q=lokeren,be&units=metric&APPID=youkey>

**Mopjes**

<https://icanhazdadjoke.com/api> (**Accept application/json )**

**Covid vaccination**

<https://covid-vaccinatie.be/api/v1/delivered.json>

<https://api.covid19api.com/summary>

**Camera 192.168.0.109:50009**

[http://192.168.0.109:50009/cgi-bin/configManager.cgi?action=setConfig&VideoWidget[0].CustomTitle[1].Text=Hello%20CVO%20FOCUS](http://192.168.0.109:50009/cgi-bin/configManager.cgi?action=setConfig&VideoWidget%5b0%5d.CustomTitle%5b1%5d.Text=Hello%20CVO%20FOCUS)

**Thingsboard**

<https://demo.thingsboard.io/signup>

<https://demo.thingsboard.io/login>

**API**

<http://demo.thingsboard.io/api/v1/iuQgO9akTxZIOOvDc7X5/telemetry>

In Module 3 zullen we de JSON data gebruiken en verwerken, bv tonen op een display.

**Node RED**

**Een grafisch IoT platform, werkt met nodes en extra Javascript functies**

**Gebaseerd op Node-JS** [**https://nodejs.org/en/**](https://nodejs.org/en/)

**Voor windows :** [**https://nodered.org/docs/getting-started/windows**](https://nodered.org/docs/getting-started/windows)

**Demo op digital ocean private cloud server : node-red en CTRL+C**

[**http://yourcloudip:1880/**](http://yourcloudip:1880/)

**Test 10 : Jouw huis op mijn cloud- map zetten**

**Zend naar** [**http://128.199.50.62:51111**](http://128.199.50.62:51111) **JSON data met jouw coördinaten en label**

**Info :**

[**https://flows.nodered.org/node/node-red-contrib-web-worldmap**](https://flows.nodered.org/node/node-red-contrib-web-worldmap)

**Vb:**

**{ "name":"Jason", "lat":51.05, "lon":-1.35 }**

**{"command":{"lat":52.175815, "lon":4.454396 }}**

MQTT

MQTT is a standard messaging protocol for the Internet of Things (IoT). It is designed as an extremely lightweight publish/subscribe messaging transport that is ideal for connecting remote devices with a small code footprint and minimal network bandwidth. MQTT today is used in a wide variety of industries, such as automotive, manufacturing, telecommunications, oil and gas, etc.

### **Lightweight and Efficient**

MQTT clients are very small, require minimal resources so can be used on small microcontrollers. MQTT message headers are small to optimize network bandwidth.

### **Bi-directional Communications**

MQTT allows for messaging between device to cloud and cloud to device. This makes for easy broadcasting messages to groups of things.

### **Scale to Millions of Things**

MQTT can scale to connect with millions of IoT devices.

### **Reliable Message Delivery**

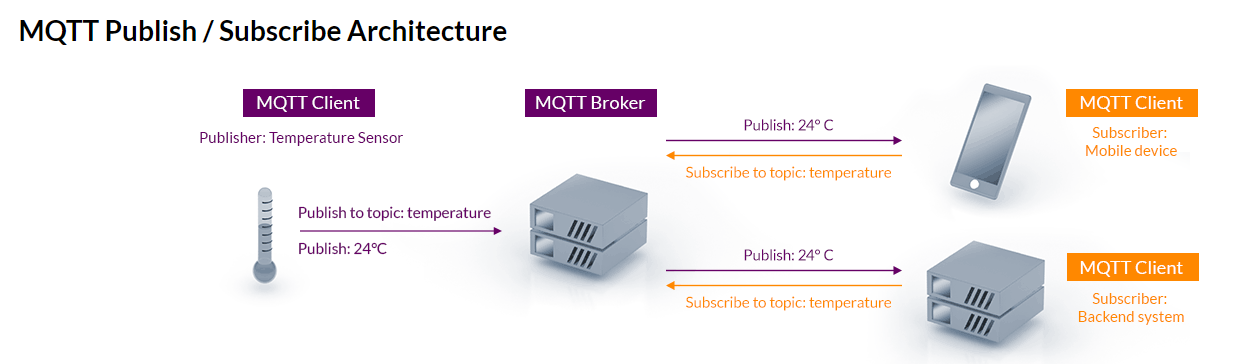
Reliability of message delivery is important for many IoT use cases. This is why MQTT has 3 defined quality of service levels: 0 - at most once, 1- at least once, 2 - exactly once

### **Support for Unreliable Networks**

Many IoT devices connect over unreliable cellular networks. MQTT’s support for persistent sessions reduces the time to reconnect the client with the broker.

### **Security Enabled**

MQTT makes it easy to encrypt messages using TLS and authenticate clients using modern authentication protocols, such as OAuth.



**Chrome**

MQTT webclient

**Understanding wildcards**

**When we analyzed the subscription operation, we learn that an MQTT client can subscribe to one or more topic filters. In case we specify a topic name as a topic filter, we will only subscribe to a single topic. We can take advantage of the following two wildcards to create topic filters that subscribe to all the topics that match the filter:**

**Plus sign (+): It is a single level wildcard that matches any name for a specific topic level. We can use this wildcard instead of specifying a name for any topic level in the topic filter.**

**Hash (#): It is a multi level wildcard that we can use only at the end of the topic filter, as the last level and matches any topic whose first levels are the same as the topic levels specified at the left-hand side of the # symbol.**

**Test 11 : Stuur MQTT messages naar elkaar**

[**http://www.hivemq.com/demos/websocket-client/**](http://www.hivemq.com/demos/websocket-client/)

**subscribe op een topic : cvofocus/voornaamachternaam**

**zend messages naar elkaar, hoe kan ik alle messages zien?**

**Test 12 : Toon in Node-red dashboard MQTT message van de MQTT webclient**

**Test 13 : Zend vanuit Node-red een MQTT message naar de MQTT webclient**

**Ook vanuit onze Raspberry Py en ESP32 kunnen we in python een MQTT Client maken.**

**import paho.mqtt.client as mqtt #import the client1**

**import time**

**############**

**def on\_message(client, userdata, message):**

**print("message received " ,str(message.payload.decode("utf-8")))**

**print("message topic=",message.topic)**

**########################################**

**broker\_address="broker.mqttdashboard.com"**

**#broker\_address="iot.eclipse.org"**

**print("creating new instance")**

**client = mqtt.Client("my\_id\_wim\_verlinden") #create new instance**

**client.on\_message=on\_message #attach function to callback**

**print("connecting to broker")**

**client.connect(broker\_address) #connect to broker**

**client.loop\_start() #start the loop**

**print("Subscribing to topic","cvofocus/wimverlinden")**

**client.subscribe("cvofocus/wimverlinden")**

**print("Publishing message to topic","cvofocus/wimverlinden")**

**client.publish("cvofocus/wimverlinden","My MQTT Paho CLient on my PC sends >> BE/FIN ? Who wins?")**

**try:**

**while True:**

**pass**

**except:**

**client.loop\_stop()**

**print("session finished")**