Netwerk latentie in communicatie tussen microcontrollerbord via TCP en UDP

Bryan Chung

November 2021

Samenvatting

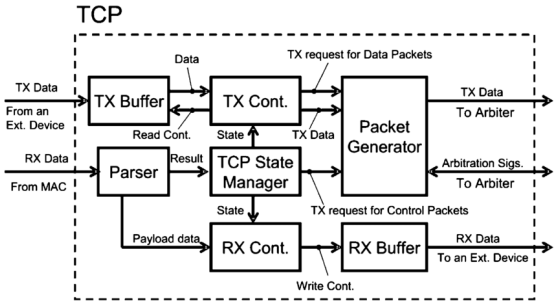
Het doel van dit onderzoek is om het snelheid van communicatie tussen meerdere microcontrollerborden via TCP en UDP te bepalen en een conclusie trekken welke is sneller. Met andere woorden communicatie tussen TCP en UDP wordt gemeten in alle scenerio's en dan met elkaar vergelijkt. Om dat te doen werd eerst vragen geformuleert en dan een experiment opgesteld. De gevonden resultaat van dit onderzoek is dat communicatie via UDP is sneller dan TCP met een factor van 2.5.

1 Inleiding

Netwerk latentie is de tijd die nodig is voordat de gegevens die u aan het ene uiteinde van uw netwerk (node) invoert, aan het andere uiteinde verschijnen. In moderne robotica is een laag netwerk latentie belangrijk, want je kan in een groot afstand een robot besturen met een klein vertraging. Hierdoor kan bijvoorbeeld een chirurg in Rotterdam een patiënt in Nairobi realtime opereren via een chirurgische robot. Alles onder de 100ms telt als laag latentie[3]. Het doel van dit onderzoek is om te bepalen of de netwerk latentie tussen microcontrollerborden laag genoeg is voor praktische applicatie. Om dit te kunnen doen, werd er een experiment opgesteld met twee Arduino Uno en een Rasberry Pi, berichten werden naar elkaar gestuurd via het netwerk van de Rasberry Pi de tijdstip werd gemarkeerd van de Arduino's. Arduino A gaat Arduino B "Pingen". PING biedt twee primaire doelstellingen, nagaan of de gastheer beschikbaar is en meten hoe lang de respons zal duren. PING-commando is een van de meest gebruikte van de verschillende commandoregel interfaces. PING bestaat uit een enkel pakket dat een echo request presenteert. Als de host beschikbaar is, reageert hij met één pakket. De tijdmeting voor PING wordt gekwantificeerd in milliseconden, dit heeft betrekking op de tijd dat het pakket bij de host aankomt en dat de respons teruggaat naar de afzender.

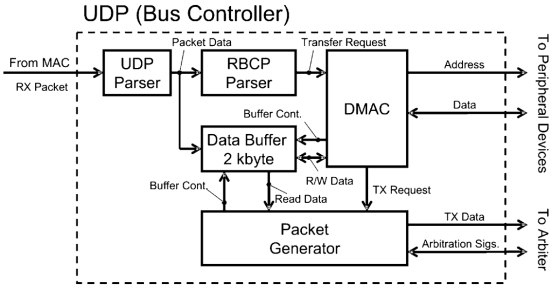
$$Latentie = Tijd_{ontvangen} - Tijd_{zenden} \tag{1}$$

$$Latentie_{gem} = Latentie_{som}/Poging_n \tag{2}$$



Figuur 1: Blok diagram van een TCP blok

Door dit poging te herhalen wordt er een lijst van latentie opgesteld. Door de som van alle latentie in de lijst delen door het aantal pogingen krijg je de gemiddelde latentie (2). Hierdoor wordt de gemiddelde latentie berekend en een conclusie getrokken. Het Transmission Control Protocol (TCP) is een verbindingsgeoriënteerd protocol dat veel gebruikt wordt voor gegevensoverdracht over netwerkverbindingen op het internet en op computernetwerken zoals local area networks en thuisnetwerken.TCP/IP is een IP-netwerkprotocol voor stabiele, betrouwbare netwerkverbindingen en geen verbindingsloos protocol zoals UDP en GRE. TCP heeft als kenmerken dat het gegevens in een datastroom kan versturen, waarbij de garantie wordt geleverd dat de gegevens aankomen zoals ze verstuurd werden, en eventuele communicatiefouten, zowel in de gegevens zelf als in de volgorde van de gegevens kunnen worden opgevangen. Hierdoor hoeft een clientapplicatie die TCP als transmissieprotocol gebruikt, geen rekening te houden met de onderliggende netwerkarchitectuur en eventuele fouten in de communicatie.Figuur 1 toont een hoofd blok van TCP en verwerkt TCP als server of client. Het parsercircuit verwerkt alleen pakketten die vooraf een TCP-tag en een bestemmingspoortnummer hebben[2]. Deze pakketten worden geanalyseerd. Circuits in dit blok, behalve de Parser, worden bestuurd door de manager met een TCP-status. Er wordt een gereduceerde TCP-statusmachine aangenomen. Gelijktijdig openen en sluiten wordt niet ondersteund.Het User Datagram Protocol (UDP) is een van de basisprotocollen van het internet. Het protocol opereert op hetzelfde niveau als TCP. UDP is onbetrouwbaar: het protocol biedt geen garantie dat de gegevens daadwerkelijk aankomen, wat bij TCP wel het geval is. Een aantal protocollen dat via UDP werkt, implementeert zelf een verificatiemethode. Hiermee zorgen ze effectief voor een vervanging van de functionaliteit die TCP heeft op dit gebied. UDP wordt veel gebruikt bij toepassingen waar het snel overdragen van de gegevens en een korte reactietijd zeer belangrijk is, en het minder erg is dat er gegevens verloren kunnen gaan, zoals bij telefonie, videoconferencing, DNS of het online spelen van actievolle spellen, zoals first person shooters. Figuur 2 toont een blokschema van het UDP-blok. De parser filtert UDP-pakketten op basis van hun tags, en stuurt een pakket met een poortnummer dat van tevoren in de UDP header is ingesteld naar de



Figuur 2: Blok diagram van een UDP blok

RBCP-parser. Dit blok verwerkt de RBCP-pakketten één voor één. Wanneer de Data Buffer bezet is door een pakket, worden de ontvangen pakketten verwijderd. De RBCP parser analyseert het pakket en extraheert parameters voor de bustoegang. De schakeling verzoekt om toegang tot de Direct Memory Access Controller (DMAC) met de parameters. De DMAC draagt gegevens over van/naar een bus-target naar/van een gegevensbuffer. Na het einde van de toegang vraagt de DMAC om verzending aan de pakketgenerator. De generator genereert een RACK en draagt deze over aan het Arbiter blok. TCP gebruikt men dus primair als de overdracht zeker en compleet moet zijn (onder andere bij bestandsoverdracht), UDP gebruikt men als de overdracht vooral snel moet zijn (telefoon, video).

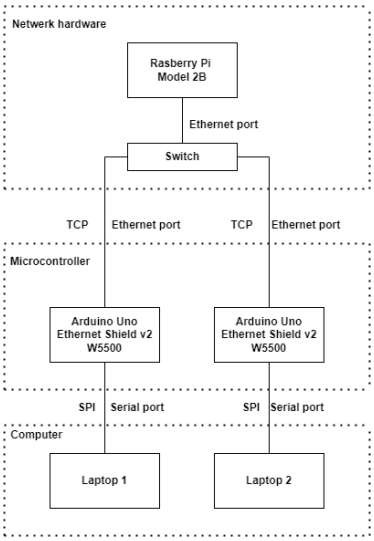
2 Experiment

Doel van dit experiment is om te testen of de netwerk latentie laag genoeg is voor communicatie via TCP en UDP tussen microcontrollerborden. Om dit te kunnen doen wordt er een experiment protocol die bestaat uit vraagstelling, meetopstelling, te verrichten handelingen, vast te leggen data, analyse opgesteld. De reden waarom er een experiment protocol opgesteld is, omdat experimentele protocollen staan centraal voor reproduceerbaarheid [1]. Aan het einde van dit experiment moet ook een conclusie getrokken worden over welke is sneller tussen de UDP en TCP.

2.1 Vraagstelling

Er werd een hoofdvraag opgesteld voor dit experiment:

- Wat is de netwerk latentie in communicatie via TCP en UDP tussen twee Arduino Uno in een best case scenario?
- Wat is de netwerk latentie in communicatie via TCP en UDP tussen twee Arduino Uno in een worst case scenario ?



Figuur 3: Blok diagram van aansluitingen

- Wat is de netwerk latentie in communicatie via TCP en UDP tussen twee Arduino Uno in een average case scenario ?
- Welke is sneller tussen de UDP en TCP in alle situatie?

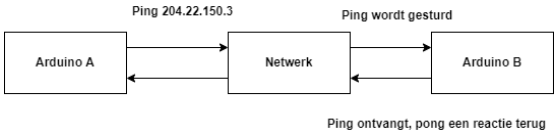
Met een deze vragen wordt het doel van het experiment duidelijker gemaakt.

2.2 Meetopstelling

In dit experiment wordt er gebruik gemaakt van de volgende instrumenten:

- 1x Rasberry Pi model 2B
- 2x Arduino Uno
- 2x Arduino Ethernet shield 2
- 2x Laptop

De Raspberry Pi 2 Model B is de tweede generatie Raspberry Pi. Het is gebaseerd op de BCM2836 system-on-chip (SoC), die een quad-core ARM Cortex-A7-processor en een krachtige GPU bevat. De Arduino Uno is een microcontrollerbord gebaseerd op de ATmega328. Het heeft 20 digitale input/output-pinnen (waarvan 6 kunnen worden gebruikt als PWM-uitgangen en 6 kunnen worden gebruikt als analoge ingangen), een 16 MHz-resonator, een USB-aansluiting, een voedingsaansluiting, een in-circuit systeemprogrammering (ICSP) koptekst en een resetknop. Met het Arduino Ethernet Shield 2 kan een Arduino-bord verbinding maken met internet. Het is gebaseerd op de Wiznet W5500 Ethernet-chip.



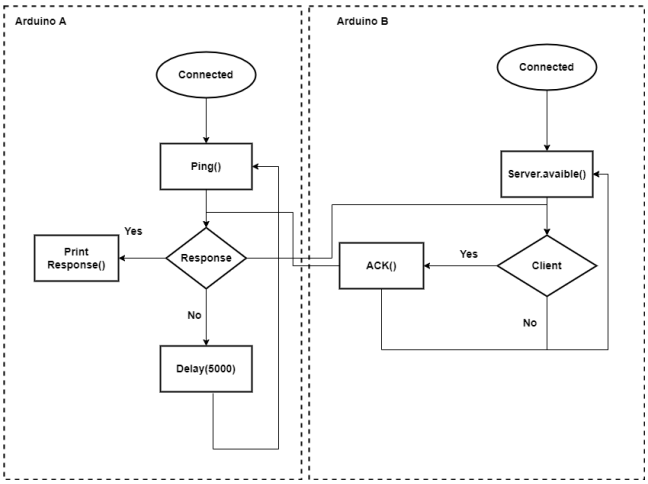
Figuur 4: Blok diagram van pingen

De Wiznet W5500 biedt een netwerk (IP)-stack die zowel TCP als UDP aankan. Het ondersteunt maximaal acht gelijktijdige socketverbindingen. Als eerst moet je de Rasberry pi aansluiten op de switch. Zorg dat de Pi "Ethernet port sharing compatibleïs. Dan moet je zorgen dat beide Arduino is aangesloten aan hun eigen laptop. Vervolgens sluit je beiden Arduino op de switch via de Ethernet port. Figuur 3 is een blokschema van de aansluitingen.

Voor de software van Arduino moet je Ethernet en SPI bibliotheek gebruiken. De SPI werd gebruikt om te communiceren met de Arduino via de serial port van de laptop. SPI is een interfacebus die gewoonlijk wordt gebruikt om gegevens te verzenden tussen microcontrollers en kleine randapparatuur zoals schuifregisters, sensoren en SD-kaarten. Het maakt gebruik van afzonderlijke klok- en datalijnen, samen met een geselecteerde lijn om het apparaat te kiezen waarmee u wilt praten. De Ethernet bibliotheek is ontworpen om te werken met het Arduino Ethernet Shield, Arduino Ethernet Shield 2, Leonardo Ethernet en alle andere W5100/W5200/W5500-gebaseerde apparaten. Met deze bibliotheek kan een Arduino-bord verbinding maken met internet. De Arduino Uno heb zelf geen real time clock (RTC). De centrale klok voor dit experiment wordt dan de real time clock van het laptop. Berichten komen via de serial port tussen de twee Arduino en dan de Arduino print het serial message op de serial monitor. De klok van beide laptops is dus gesynchroniseerd. Maar het lezen van de tijdstip op de serial monitor is natuurlijk helemaal niet nauwkeurig, uit een test bleekt dat het printen van de boodschappen op de serial monitor zou 20 tot 40ms kosten. Dus de foutmarge zit rond de 40ms.

2.3 Te verrichten handelingen

Om het experiment te starten moet je de code op beide Arduino runnen. Hieronder zijn pseudo-codes geschreven in engels van beide Arduino's. Zoals te zien bij figuur 4 Arduino B gaat Arduino B pingen, dan pong Arduino B een reactie terug. Het verschil tussen de tijd van zending end de tijd van ontvangen is de latentie. Arduino A gaat verbinding maken met de server, de server is Arduino B. Als het gelukt is om een verbinding maken print het een bericht op de serial monitor dat het gelukt is. Arduino A zou dan een string sturen naar de server, verderop gaat die een bericht op de serial monitor printen dat het gelukt is om een bericht naar de server te sturen, dan krijg je ook de tijdstip van het sturen van het bericht naar de server. Wanneer Arduino A een response krijgt van Arduino B, print Arduino A die response op de serial monitor, zo krijgt de tijdstip van het ontvangen bericht. Figuur 5 toont een control-flow van de programma.



Figuur 5: Control flow van code

Arduino A:

```
Connect to server

if server is connected, print "connected to server"
else print "failed to connnect to server"

while server is connected:
write string "request" to server
serial print "request" on serial monitor #get timestamp of send message
read response from server
serial print response #get timestamp of reponse
delay 5 seconds #interval of 5 seconds
repeat
```

Arduino B:

```
Host server
Connect to client

if client is available:
read response
serial print response # to confirm message is recieved
write "acknowledge" to client #pong back to client
repeat
```

2.4 Vast te leggen data

De tijdstip wordt vastgelegd in de serial monitor van de Arduino. Vervolgens noteer je met hand de tijdstippen op een tabel. Om de latentie te berekenen gebruik je (1). Latentie noteer je in milliseconde(ms). In een interval 5 seconde gaat Arduino A pingen. Sluit vervolgens meerdere Arduino's op het switch voor een worst case scenario.

2.5 Analyse

Tabel 1: Gemeten resultaten van communicatie via TCP (best case)

Tijdstip zenden	Tijdstip ontvangen	Latentie (ms)
17:58:16.120	17:58:17.144	1024
17:58:22.128	17:58:23.107	979
17:58:28.133	17:58:29.105	972
17:58:34.123	17:58:35.101	978
17:58:40.134	17:58:41.114	980
17:58:46.095	17:58:47.119	1024
17:58:52.100	17:58:54.100	2000
17:58:59.132	17:59:01.134	2002
17:59:06.121	17:59:08.125	2004
17:59:13.106	17:59:14.129	1023
17:59:19.115	17:59:20.092	977
17:59:25.122	17:59:26.103	981
17:59:31.129	17:59:33.130	2001

Tabel 1 zijn de data van het communicatie via TCP. Om de gemiddelde latentie te berekenen gebruiken we (2). Eerst ga je de som van alle latentie berekenen. 1024+979+972+978+980+1024+2000+2002+2004+1023+977+981+2001 = 16945. Er zijn dus 13 berichten dus delen door 13. Dan wordt het

$$16945/13.0 = 1303.5$$

De uitkomst met foutmarge is 1303 ± 40 ms. Dit is de latentie in een best case scenario met 2 Arduino's.

Tabel 2: Gemeten resultaten van communicatie via UDP (best case).

Tijdstip zenden	Tijdstip ontvangen	Latentie (ms)
18:23:04.173	18:23:04.499	260
18:23:09.502	18:23:10.058	556
18:23:15.073	18:23:15.627	554
18:23:20.648	18:23:21.021	373
18:23:26.019	18:23:26.529	510
18:23:31.533	18:23:31.857	324
18:23:36.868	18:23:37.379	511
18:23:42.390	18:23:43.040	650
18:23:48.052	18:23:48.376	324
18:23:53.340	18:23:53.896	556
18:23:58.896	18:23:59.593	697
18:24:04.605	18:24:05.115	510
18:24:10.117	18:24:10.674	557

Tabel 2 zijn de data van het communicatie via UDP in een best case scenario. De som van alle latentie is $260 + 556 + 554 + 373 + 510 + 324 + 511 + 650 + 324 + 556 + 697 + 510 + 557 = 6382$. Er zijn dus 13 berichten dus delen door 13. Dan wordt het

$$6382/13.0 = 491$$

De uitkomst met foutmarge is 491 ± 40 ms

Tabel 3: Gemeten resultaten van communicatie via TCP (worst case)

Tijdstip zenden	Tijdstip ontvangen	Latentie (ms)
20:29:50.825	20:29:52.768	1943
20:29:57.763	20:29:59.428	1665
20:30:04.423	20:30:06.088	1665
20:30:11.085	20:30:12.750	1665
20:30:17.744	20:30:19.274	1530
20:30:24.272	20:30:25.844	1572
20:30:30.840	20:30:32.369	1529
20:30:37.369	20:30:39.129	1760
20:30:44.169	20:30:45.977	1808
20:30:50.972	20:30:52.917	1945
20:30:57.878	20:30:59.681	1803
20:31:04.673	20:31:06.524	1851
20:31:11.516	20:31:13.274	1758

Client lag is de de lag die veroorzaak wordt door het traag verwerken van de data door de ontvanger van de data, soms is de oorzaak een overbelasting van de client Arduino doordat er teveel processen actief zijn. Server lag is de lag die veroorzaakt wordt doordat de server de aangeboden data niet snel genoeg kan verwerken. Dus om een worst case scenario te krijgen hebben wordt er meerdere Arduino's client op het server aangesloten. Tabel 3 zijn de data van

het communicatie via TCP in een worst case scenario. De som van alle latentie is $1943 + 1665 + 1665 + 1665 + 1530 + 1572 + 1529 + 1760 + 1808 + 1945 + 1803 + 1851 + 1758 = 22,494$. Dan wordt het

$$22{,}494/13.0 = 1730.3$$

De uitkomst met foutmarge is 1730.3 ± 40 ms

Tabel 4: Gemeten resultaten van communicatie via TCP (worst case)

Tijdstip zenden	Tijdstip ontvangen	Latentie (ms)
20:57:42.090	20:57:42.691	601
20:57:47.676	20:57:48.185	509
20:57:53.177	20:57:54.057	880
20:57:59.053	20:57:59.750	697
20:58:04.7464	20:58:05.577	831
20:58:10.576	20:58:11.362	786
20:58:16.365	20:58:17.060	695
20:58:22.058	20:58:22.892	834
20:58:27.880	20:58:28.436	556
20:58:33.432	20:58:34.174	742
20:58:39.172	20:58:39.915	743
20:58:44.917	20:58:45.474	557
20:58:50.479	20:58:51.310	831

Tabel 4 zijn de data van het communicatie via UDP in een worst case scenario. De som van alle latentie is $601 + 509 + 880 + 697 + 831 + 786 + 695 + 834 + 556 + 742 + 743 + 557 + 831 = 9262$. Er zijn dus 13 berichten dus delen door 13. Dan wordt het

$$9262/13.0 = 712.4$$

De uitkomst met foutmarge is 712 ± 40 ms

3 Conclusie

Latentie heeft invloed op de algehele snelheid van uw verbinding, maar veel toepassingen hebben manieren gevonden om deze te omzeilen. Videodiensten zoals Netflix bufferen bijvoorbeeld de gegevens die ze downloaden. Dit betekent dat ze een voorsprong hebben op het afspelen en extra videogegevens opslaan, zodat het klaar is voor gebruik zodra het nodig is. Zelfs als er een vertraging of een plotselinge fluctuatie in de stream is, kan de video nog steeds soepel worden afgespeeld zolang er nog informatie in de buffer zit. Andere soorten activiteiten zijn minder vergevingsgezind. Online games zijn afhankelijk van wederzijdse interactie tussen de speler en de server. Latentie vertraagt je reactietijd, ook wel lag genoemd. Lag kan uw acties traag of wankel maken. De doel van

dit onderzoek is om te vinden UDP of TCP sneller is voor communicatie tussen microcontrollerborden. De best case scenario resultaat van TCP is rond 1303ms en worst case is 1730ms, de gemmidelde latentie in een average case scenario is dus $(1303+1730)/2 = 1517ms \pm 40$. Voor UDP is het $(491+712)/2 = 602ms \pm 40$. In dit experiment is communicatie via UDP 252% sneller dan TCP. In alle situaties is de UDP sneller dan de TCP, dat klopt omdat bij UDP laat de protocol constant data binnen zonder te controleren op data loss.

Referenties

[1] Marco Anisetti, Valerio Bellandi, Alberto Colombo, Marco Cremonini, Ernesto Damiani, Fulvio Frati, Joêl T Hounsou, and Davide Rebeccani. Learning computer networking on open paravirtual laboratories. *IEEE Transactions on Education*, 50(4):302–311, 2007.

[2] Tomohisa Uchida. Hardware-based tcp processor for gigabit ethernet. *Nuclear Science, IEEE Transactions on*, 55:1631 – 1637, 07 2008.

[3] Florian Voigtländer, Ali Ramadan, Joseph Eichinger, Claus Lenz, Dirk Pensky, and Alois Knoll. 5g for robotics: Ultra-low latency control of distributed robotic systems. In *2017 International Symposium on Computer Science and Intelligent Controls (ISCSIC)*, pages 69–72. IEEE, 2017.