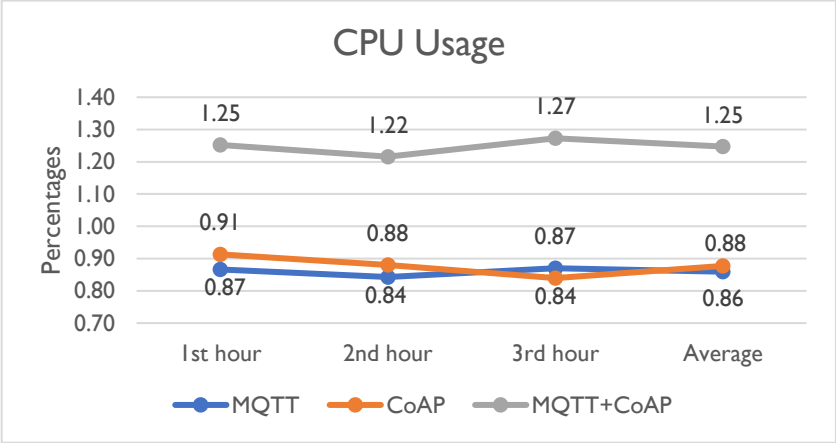
BAB 6 PEMBAHASAN

Pada bab pembahasan ini akan dijelaskan data hasil dari pengujian yang telah dilakukan dan kemudian akan dibahas sehingga dapat menjadi bahan penarikan kesimpulan pada bab selanjutnya.

6.1 Penggunaan CPU dan Memori

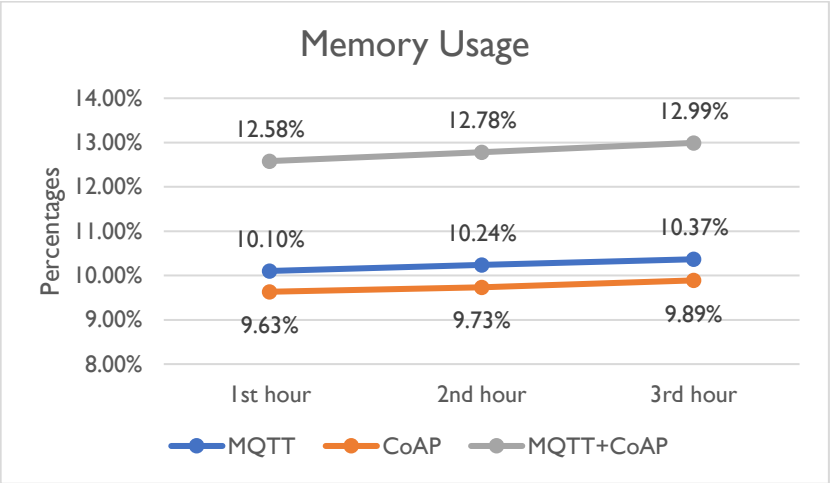
6.1.1 Penggunaan CPU



Gambar 6.1 Grafik CPU *Usage*

Pada Gambar 6.1 menunjukkan hasil pengujian kinerja penggunaan CPU oleh *middleware* pada raspberry pi zero yang dilakukan selama 3 jam. Dari hasil pengujian diperoleh rata-rata penggunaan CPU ketika menggunakan nodeMCU MQTT saja pada satu jam pertama sebesar 0,87%, satu jam kedua menurun menjadi 0,84%, dan satu jam ketiga mengalami kenaikan menjadi 0,87%. Sehingga diperoleh rata-rata secara keseluruhan penggunaan CPU dengan nodeMCU MQTT saja sebesar 0,86%. Penggunaan CPU untuk nodeMCU CoAP saja pada satu jam pertama sebesar 0,91%, satu jam kedua menurun menjadi 0,88%, dan pada satu jam ketiga menurun menjadi 0,84%. Sehingga diperoleh rata-rata secara keseluruhan penggunaan CPU dengan nodeMCU CoAP saja sebesar 0,88%. Sedangkan saat nodeMCU MQTT dan CoAP dijalankan bersamaan penggunaan CPU dari *middleware* mengalami peningkatan menjadi 1,25% pada satu jam pertama, pada satu jam kedua menurun menjadi 1,22%, dan pada satu jam ketiga mengalami kenaikan menjadi 1,27%. Sehingga diperoleh rata-rata secara keseluruhan penggunaan CPU saat nodeMCU MQTT dan CoAP dijalankan secara bersamaam sebesar 1,25%. Secara keseluruhan penggunaan CPU dari *middleware* tergolong kecil karena hanya menggunakan rata-rata 1,25% dari CPU.

6.1.2 Penggunaan Memori



Gambar 6.2 Grafik Memory *Usage*

Pada Gambar 5.2 menunjukkan hasil pengujian kinerja penggunaan memori oleh *middleware* pada raspberry pi zero yang dilakukan selama 3 jam. Dari hasil pengujian diperoleh *range* penggunaan memori ketika menggunakan nodeMCU MQTT sekitar 10,10%-10,37% dan ketika menggunakan nodeMCU CoAP sekitar 9,63%-9,89% sedangkan saat nodeMCU MQTT dan CoAP dijalankan bersamaan penggunaan memori mengalami peningkatan menjadi 12,58%-12,99%.

6.2 Data Transfer dan *Delay*

Hasil yang diperoleh dari pengujian dengan menggunakan 5 nodeMCU MQTT dan 5 nodeMCU CoAP menunjukkan bahwa pengiriman data dengan menggunakan protokol MQTT dan CoAP memiliki tingkat kesuksesan 100% dimana setiap node berhasil mengirimkan sebanyak 120 data dengan rata-rata *delay* sebesar 0,373 detik untuk MQTT dan 0,467 detik untuk CoAP. Nilai *delay* yang diperoleh relatif kecil untuk masing-masing protokol karena ukuran paket yang dikirimkan oleh nodeMCU MQTT dan nodeMCU CoAP kurang dari 1 Kb (269 bytes untuk MQTT dan 287 bytes untuk CoAP).

Tabel 6.1 MQTT Delivery Ratio

MQTT						
Topic	Expected	Actual	Invalid	Loss Rate	Success rate	Average *Delay*(s)
barrack-1	120	120	0	0.00%	100%	0,357
barrack-2	120	120	0	0.00%	100%	0,566
barrack-3	120	120	0	0.00%	100%	0,123
barrack-4	120	120	0	0.00%	100%	0,578
barrack-5	120	120	0	0.00%	100%	0,242
Total	600	600	0	0.00%	100%	0,373

Tabel 6.2 CoAP Delivery Ratio

CoAP						
Topic	Expected	Actual	Invalid	Loss Rate	Success rate	Average *Delay* (s)
kitchen-1	120	120	0	0%	100%	0,776
kitchen-2	120	120	0	0%	100%	0,647
kitchen-3	120	120	0	0%	100%	0,653
kitchen-4	120	120	0	0%	100%	0,128
kitchen-5	120	120	0	0%	100%	0,131
Total	600	600	0	0%	100%	0,467

Tabel 6.3 Webscoket Delivery Ratio

Websocket			
Expected	Actual	Overhead	Average *Delay*
1200	1200	0.00%	0,495

Tabel 6.4 *End-to-end Delay*

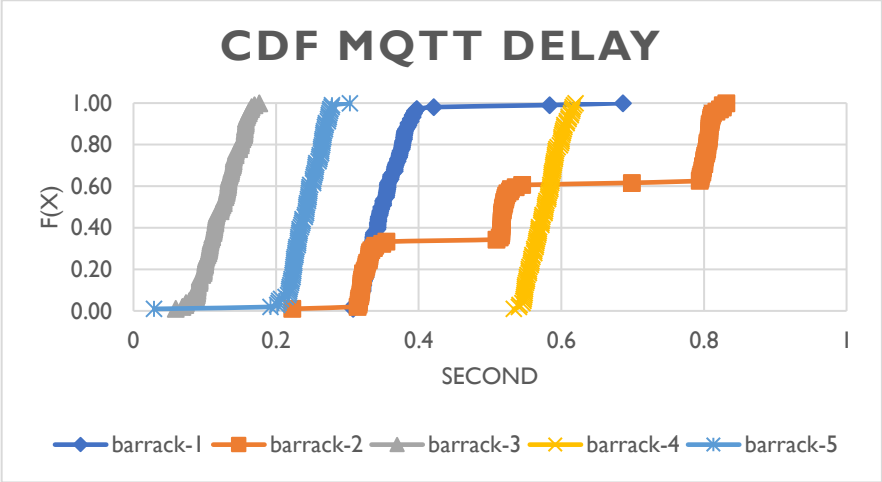
End-to-end Delay (s)				
MQTT	CoAP	Websocket	MQTT -> Websocket	CoAP -> Websocket
0,373	0,467	0,495	0,868	0,962

Pada sisi *subscriber*, data yang diterima sesuai dengan data yang dikirimkan oleh *publisher* dan tidak terjadi *overhead*. Rata-rata waktu yang dibutuhkan *middleware* untuk mengirim data ke datacenter sebesar 0,495 detik. Jadi waktu yang dibutuhkan oleh nodeMCU MQTT untuk mengirim data ke datacenter adalah 0.868 detik sedangkan untuk nodeMCU CoAP 0,962 detik.

Delay yang diperoleh dari masing-masing *publisher* dengan protokol MQTT dan protokol CoAP memiliki nilai yang bervariasi sehingga dapat digambarkan dalam grafik CDF *(Cumulative Distribution Function)* untuk mengetahui bagaimana distribusi kumulatif dari *delay* pada masing-masing topik dan mengetahui nilai minimal *delay* dan nilai maksimal *delay* untuk masing-masing topik.

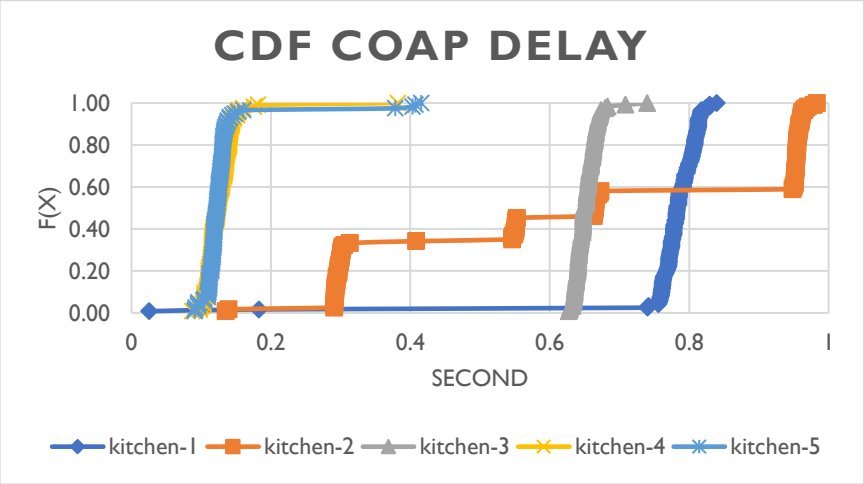
Grafik (Gambar 6.3) CDF *(Cumulative Distribution Function)* menggambarkan distribusi kumulatif dari *delay* pada tiap *range*. Batas bawah menggambarkan nilai *delay* terendah dan batas atas menggambarkan nilai *delay* tertinggi. Berdasarkan Gambar 6.3 dapat disimpulkan bahwa distribusi kumulatif *delay* cukup stabil pada topik barrack-1 dengan *range delay* 0,308-0,686 detik, topik barrack-3 dengan *range delay* 0,059-0,176 detik, barrack-4 dengan *range delay* 0,533-0,62 detik, dan barrack-5 dengan *range delay* 0,029-0,303 detik. Sedangkan pada topik barrack-2

distribusi kumulatif *delay* lebih bervariasi dengan rentang *delay* yang cukup lebar yakni 0,222-0,831 detik.



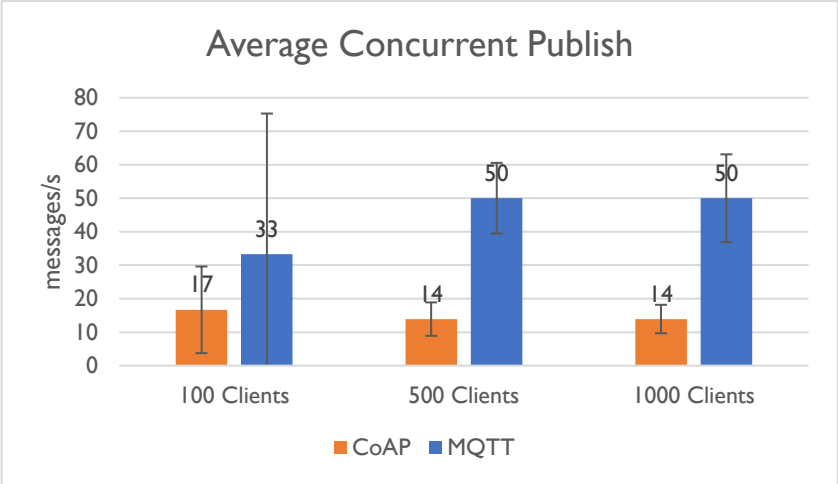
Gambar 6.3 CDF MQTT *Delay*

Grafik (Gambar 6.4) CDF *(Cumulative Distribution Function)* menggambarkan distribusi kumulatif dari *delay* pada tiap *range*. Batas bawah menggambarkan nilai *delay* terendah dan batas atas menggambarkan nilai *delay* tertinggi. Berdasarkan Gambar 6.4 dapat disimpulkan bahwa distribusi kumulatif dari *delay* cukup stabil terjadi pada topik kitchen-1 dengan *range delay* 0,025-0,839 detik, topik kitchen-3 dengan *range delay* 0,627-0,74 detik, topik kitchen-4 dengan *range delay* 0,087-0,382 detik, topik kitchen-5 dengan *range delay* 0,09-0,416 detik. Sedangkan pada topik kitchen-2 distribusi kumulatif *delay* lebih bervariasi dengan rentang *delay* yang cukup lebar yakni 0,135-0,983 detik.



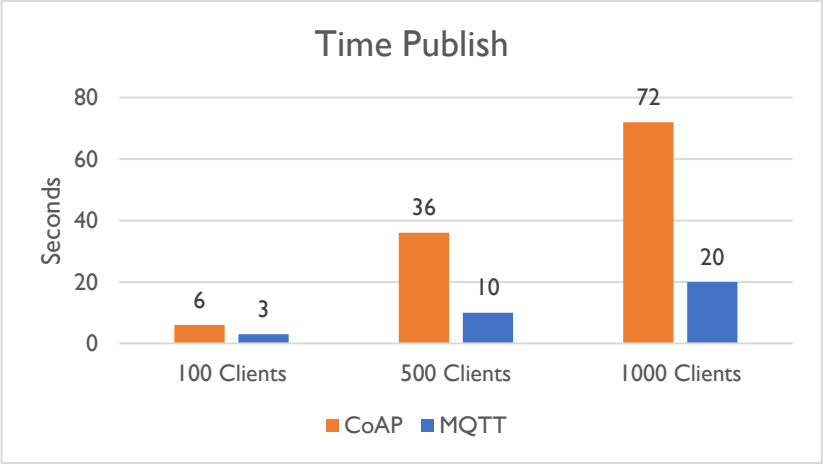
Gambar 6.4 CDF CoAP *Delay*

6.3 Skalabilitas



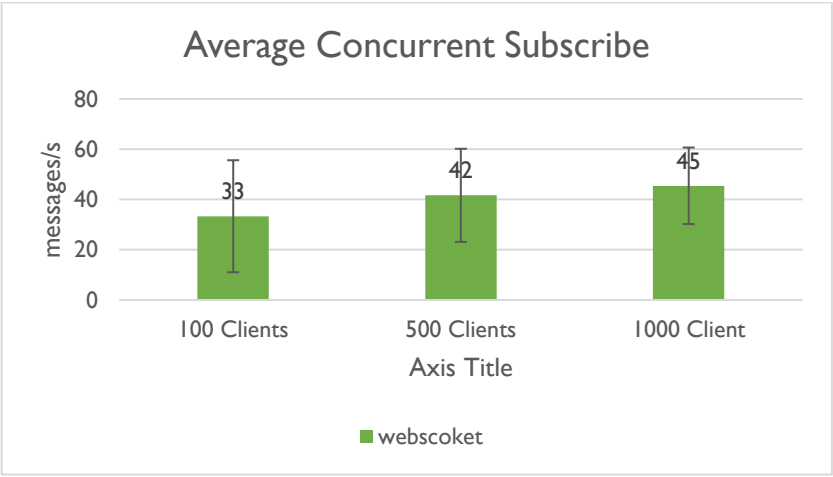
Gambar 6.5 Grafik Average *Concurrent Publish*

Gambar 6.5 menunjukkan rata-rata *publisher* yang dapat ditangani oleh *middleware* dalam satu detik. Jumlah maksimal pesan per detik untuk protokol MQTT adalah 81 dan 41 untuk protokol CoAP. Standar deviasi merupakan nilai statistik yang digunakan untuk menentukan bagaimana sebaran data dalam sampel dan seberapa dekat titik data individu ke rata-rata nilai sampel. Nilai standar deviasi yang lebih besar menunjukkan bahwa data tersebut jauh dari nilai rata-rata. Untuk menghitung standar deviasi, pertama adalah menghitung nilai rata-rata dari semua titik data. Selanjutnya, penyimpangan setiap titik data dari rata-rata dihitung dengan mengurangkan nilai dari nilai rata-rata. Deviasi setiap titik data akan dikuadratkan, dan dicari penyimpangan kuadrat individu rata-rata. Nilai yang dihasilkan dikenal sebagai varians. Standar deviasi adalah akar kuadrat dari varians. Standar deviasi untuk protokol CoAP menurun seiring dengan bertambahnya jumlah klien. Dalam menangani 100 klien, standar deviasi untuk protokol MQTT adalah 41,96 dan 12,94 untuk CoAP. Ketika jumlah klien meningkat menjadi 500, standar deviasi dari MQTT menurun menjadi 10,55 dan untuk CoAP menjadi 4,99. Sedangkan untuk 1000 klien standar deviasi MQTT adalah 13,10 dan CoAP 4,25. Jika dibandingkan dengan CoAP, protokol MQTT lebih baik dengan rata-rata pesan per detik sekitar 50 pesan.



Gambar 6.6 Grafik *Time Publish*

Gambar 6.6 menunjukkan waktu yang dibutuhkan oleh *middleware* untuk menyelesaikan *query publish* untuk masing-masing variasi jumlah klien. Waktu yang dibutuhkan untuk menyelesaikan *query publish* berbanding lurus dengan banyaknya jumlah klien. Semakin banyak jumlah klien semakin banyak pula waktu yang dibutuhkan untuk menyelesaikan *query publish*. Dari sisi klien, protokol MQTT lebih cepat daripada protokol CoAP dalam pengiriman pesan.

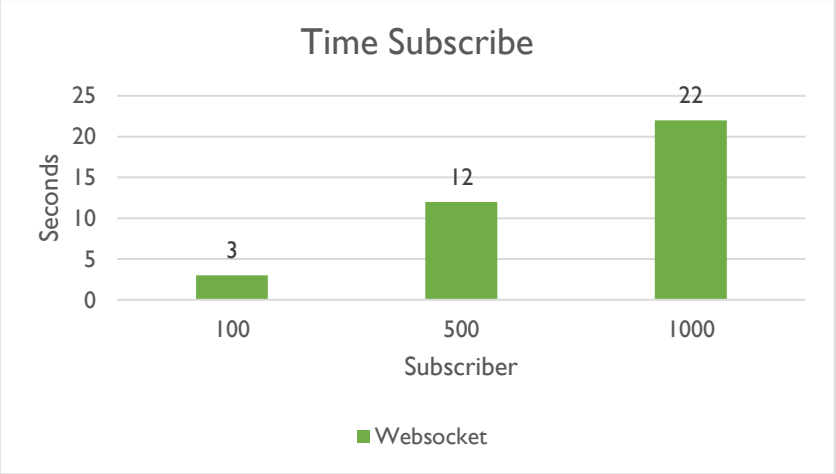


Gambar 6.7 Grafik Average *Concurrent Subscribe*

Pada Gambar 6.7 menunjukkan rata-rata *subscriber* yang dapat ditangani oleh *middleware* dalam satu detik. Jumlah *subscriber* yang dapat ditangani tiap detik meningkat seiring dengan meningkatnya jumlah klien dikarenakan mekanisme kerja dari webscoket dalam hal ini adalah dengan membuat sejumlah koneksi yang sama dengan jumlah *subscriber* dimana biasanya websocket melakukan satu koneksi saja untuk bertukar banyak pesan. Standar deviasi merupakan nilai statistik yang digunakan untuk menentukan bagaimana sebaran data dalam sampel dan seberapa dekat titik data individu ke rata-rata nilai sampel. Nilai standar deviasi yang lebih besar menunjukkan bahwa data tersebut jauh dari nilai rata-rata. Untuk menghitung standar deviasi, pertama adalah menghitung nilai

rata-rata dari semua titik data. Selanjutnya, penyimpangan setiap titik data dari rata-rata dihitung dengan mengurangkan nilai dari nilai rata-rata. Deviasi setiap titik data akan dikuadratkan, dan dicari penyimpangan kuadrat individu rata-rata. Nilai yang dihasilkan dikenal sebagai varians. Standar deviasi adalah akar kuadrat dari varians. Standar deviasi websocket secara berurutan adalah 22,30, 18,28, 15,23. Standar deviasi menurun seiring dengan meningkatnya jumlah klien yang berarti data semakin dekat ke rata-rata nilai sampel. Maksimal jumlah *subscriber* yang dapat ditangani oleh *middleware* dalam satu detik adalah 54.

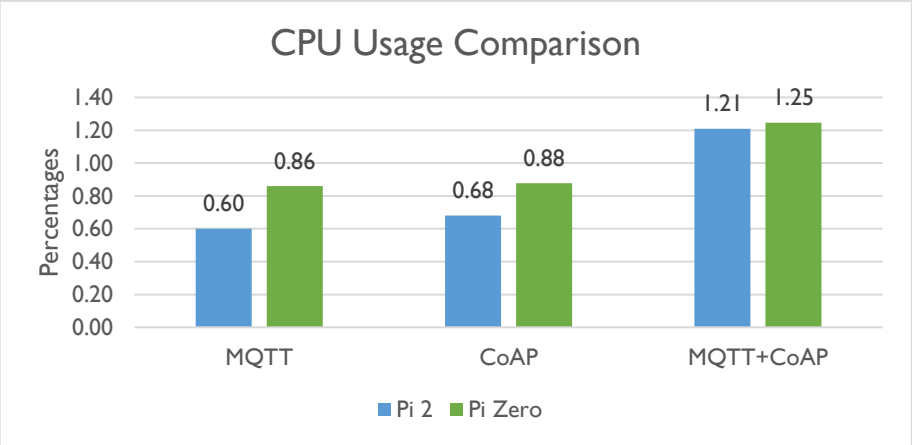
Gambar 6.8 menunjukkan waktu yang dibutuhkan oleh *middleware* untuk menyelesaikan *query subscribe* untuk masing-masing variasi jumlah *subscribe*. Waktu yang dibutuhkan untuk menyelesaikan *query subscribe* berbanding lurus dengan banyaknya jumlah *subscriber*. Semakin banyak jumlah *subscriber* semakin banyak pula waktu yang dibutuhkan untuk menyelesaikan *query subscribe*.



Gambar 6.8 Grafik *Time Subscribe*

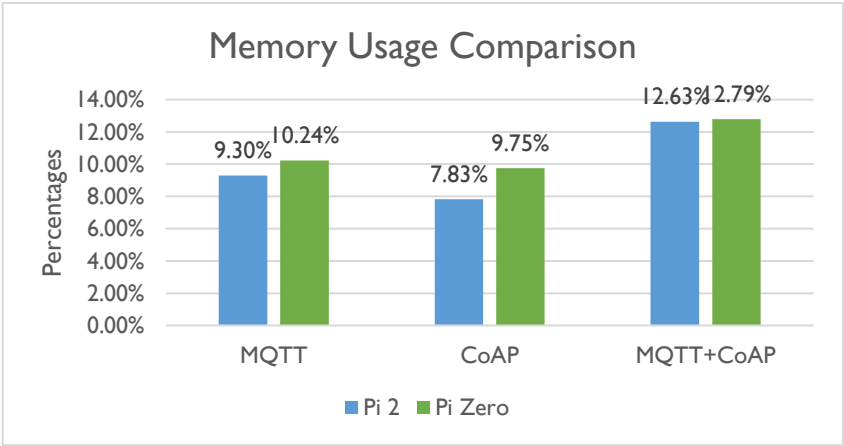
6.4 Perbandingan Hasil Raspberry Pi Zero dengan Raspberry Pi 2

6.4.1 Penggunaan CPU dan Memori



Gambar 6.9 Grafik CPU *Usage Comparison*

Gambar 6.9 menunjukkan kinerja penggunaan CPU oleh *middleware* pada raspberry pi zero mengalami peningkatan jika dibandingkan dengan raspberry pi 2. Penggunaan CPU untuk protokol MQTT pada raspberry pi zero mengalami peningkatan sebesar 0,26%. Untuk protokol CoAP penggunaan CPU meningkat sebesar 0,20%. Sedangkan untuk protokol MQTT+CoAP terjadi peningkatan penggunaan CPU sebesar 0,04%. Peningkatan penggunaan CPU pada raspberry pi zero dikarenakan sumber daya raspberry pi zero lebih kecil jika dibandingkan dengan raspberry pi 2, dimana raspberry pi 2 menggunakan quad-core ARM Cortex-A7 CPU sedangkan raspberry pi zero hanya menggunakan single-core CPU sehingga terjadi peningkatan kebutuhan sumber daya.



Gambar 6.10 Grafik Memory *Usage Comparison*

Seperti ditunjukkan pada Gambar 6.10 kinerja penggunaan memori oleh *middleware* pada raspberry pi zero mengalami peningkatan jika dibandingkan dengan raspberry pi 2. Penggunaan memori untuk protokol MQTT pada raspberry pi zero mengalami peningkatan sebesar 0,94%. Untuk protokol CoAP penggunaan

memori meningkat sebesar 1,92%. Sedangkan untuk protokol MQTT+CoAP terjadi peningkatan penggunaan memori sebesar 0,16%. Peningkatan penggunaan memori pada raspberry pi zero dikarenakan sumber daya yang dimiliki raspberry pi zero lebih kecil jika dibandingkan dengan raspberry pi 2, dimana raspberry pi 2 memiliki 1GB RAM sedangkan raspberry pi zero hanya setengah dari RAM raspberry pi 2 yaitu 512MB sehingga terjadi peningkatan kebutuhan sumber daya.

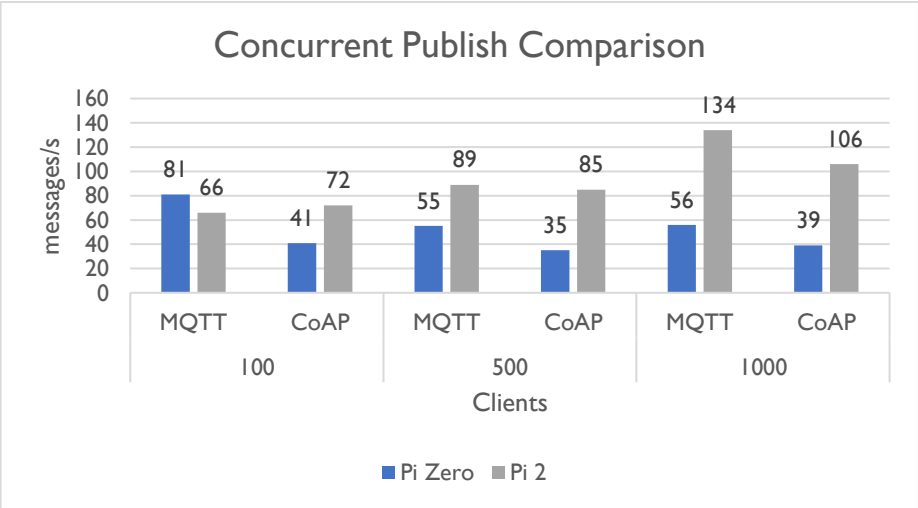
6.4.2 *Delay*

Tabel 6.5 Perbandingan *Delay*

Host	MQTT	CoAP	Websocket	MQTT -> Websocket	CoAP -> Websocket
Pi 2	0,33	0,358	0,506	0,836	0,864
Pi Zero	0,373	0,467	0,495	0,868	0,962

Tabel 6.5 menunjukkan perbandingan *delay* antara *host* raspberry pi zero dan raspberry pi 2. *Delay* MQTT dan CoAP pada raspberry pi zero meningkat menjadi 0.373 dan 0.467 detik. Sedangkan *delay* websocket menurun menjadi 0.495 detik. Nilai dari *end-to-end delay* pada MQTT – Websocket mengalami peningkatan menjadi 0.868 detik dan pad CoAP – Websocket meningkat menjadi 0.962 detik.

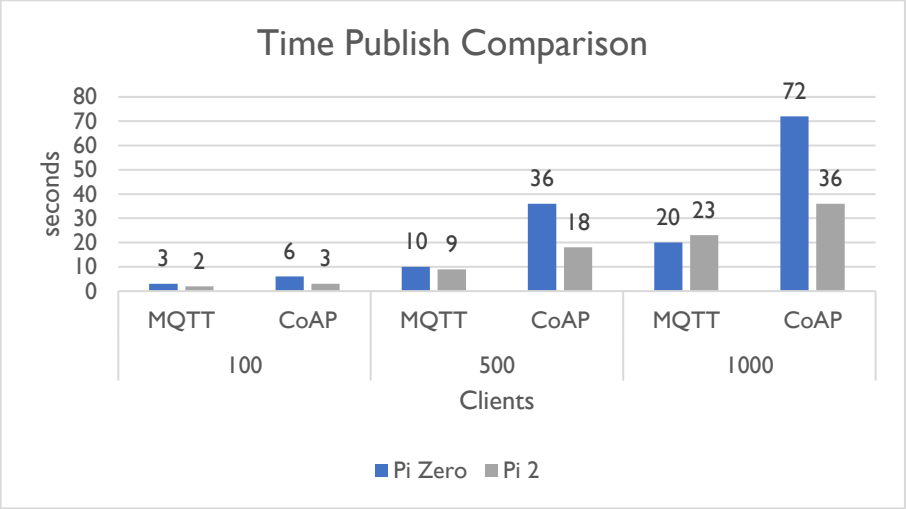
6.4.3 Skalabilitas



Gambar 6.11 Grafik *Concurrent Publish Comparison*

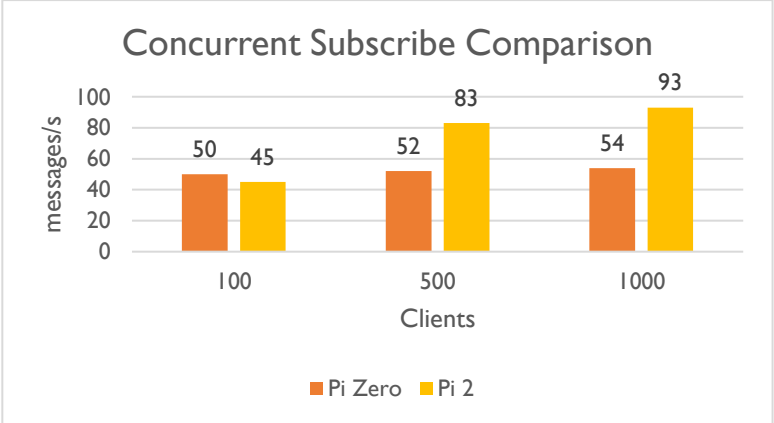
Gambar 6.11 menunjukkan perbandingan jumlah maksimal *publisher* yang dapat ditangani oleh *middleware* dalam satu detik antara raspberry pi 2 dan raspberry pi zero. Pada raspberry pi 2 jumlah *publisher* yang dapat ditangani berbanding lurus dengan peningkatan jumlah klien. Semakin banyak jumlah klien semakin banyak pula jumlah *publisher* yang dapat ditangani. Berbeda halnya dengan raspberry pi zero, dimana dapat menangani secara maksimal hanya ketika jumlah klien adalah 100. Ketika jumlah klien meningkat pada raspberry pi zero,

maka jumlah *publisher* yang dapat ditangani mengalami penurunan. Hal ini dikarenakan perbedaan sumber daya pada raspberry pi 2 dan raspberry pi zero.



Gambar 6.12 Grafik *Time Publish Comparison*

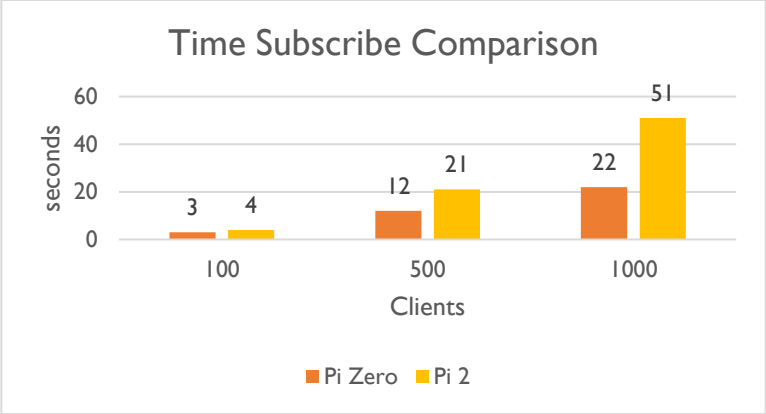
Pada Gambar 6.12 menunjukkan perbandingan waktu yang dibutuhkan oleh *middleware* untuk menyelesaikan *query publish* antara raspberry pi zero dan raspberry pi 2. Waktu yang dibutuhkan oleh kedua *host* untuk menyelesaikan *query publish* berbanding lurus dengan banyaknya jumlah klien. Semakin banyak jumlah klien semakin banyak pula waktu yang dibutuhkan untuk menyelesaikan *query publish*. Jika dibandingkan secara keseluruhan *middleware* dengan *host* raspberry pi 2 lebih cepat dalam menyelesaikan *query publish*. Hal ini dikarenakan sumber daya pada raspberry pi 2 lebih baik dari sumber daya di raspberry pi zero.



Gambar 6.13 Grafik *Concurrent Subscribe Comparison*

Gambar 6.13 menunjukkan perbandingan kemampuan *middleware* dalam mengelola *subscriber* per detik antara raspberry pi zero dan raspberry pi 2. Secara keseluruhan memang *middleware* pada raspberry pi 2 dapat mengelola lebih banyak *subscriber* daripada *middleware* pada raspberry pi zero namun pada raspberry pi zero dengan jumlah 100 klien jumlah *subscriber* yang dapat dikelola

lebih baik daripada raspberry pi 2 dengan perbedaan 5 *subscriber*. Jumlah *subscriber* yang dapat ditangani tiap detik meningkat seiring dengan meningkatnya jumlah klien dikarenakan mekanisme kerja dari webscoket dalam hal ini adalah dengan membuat sejumlah koneksi yang sama dengan jumlah *subscriber* dimana biasanya websocket melakukan satu koneksi saja untuk bertukar banyak pesan.



Gambar 6.14 Grafik *Time Subscribe Comparison*

Pada Gambar 6.14 menunjukkan perbedaan waktu yang dibutuhkan oleh *middleware* untuk menyelesaikan *query subscribe* antara raspberry pi zero dan raspberry pi 2. Waktu yang dibutuhkan oleh kedua *host* berbanding lurus dengan banyaknya jumlah klien. Semakin banyak jumlah klien semakin banyak pula waktu yang dibutuhkan untuk menyelesaikan *query subscribe*. Dalam hal ini *middleware* dengan *host* raspberry pi zero lebih cepat dalam menangani *query subscribe* jika dibandingkan dengan *middleware* pada raspberry pi 2. Hal ini dikarenakan perbedaan sumber daya yang dimiliki oleh kedua *host* untuk *middleware*.