BAB 2
DASAR TEORI

2.1 KAJIAN PUSTAKA

Penelitian [5] membahas tentang pemantauan alat infus pasien berbasis *Internet of Things*. Alat yang digunakan adalah mikrokontroler ESP32 untuk menampilkan hasil pemantauan melalui aplikasi berbasis *web* di ruang perawat. Hasilnya, pemantauan menggunakan sistem memungkinkan pemantauan infus pasien yang lebih baik, dengan hasil pengujian berupa nilai persen dan rata-rata tetes infus dengan selisih rata-rata berat antara sensor *load cell* dan timbangan manual sebesar 18 ml atau 4,86% tegangannya 5,54 volt.

Penelitian [6] mengembangkan sistem pemantauan volume cairan pasien menggunakan NodeMCU ESP8266. Pada penelitian ini, data pembacaan sensor dari NodeMCU ESP8266 diproses dan dikirim ke aplikasi *load cell* melalui jaringan WiFi. Jumlah tetes infus dan volume infus ditampilkan pada LCD dan *load cell.* Jika volume injeksi kurang dari 50ml, alarm akan berbunyi dan garis merah akan muncul di aplikasi *load cell*. Hasil Pengujian *load cell* dengan Drop Volume 0,05ml Hasil pengujian sensor *load cell* menunjukkan bahwa setiap volume drop adalah 0,05ml. Hasil pengujian *droplet* sensor dengan pengukuran 20 TPM dalam 5 kali percobaan memiliki *error* masing-masing sebesar 1,75% dan 3,88% untuk 5 pembacaan yang berbeda dalam 3 kali percobaan. Terjadi *delay* 19,6 detik dalam pengiriman data hasil pengujian aplikasi *load cell*. Namun pada penelitian [5] dan [6], parameter yang dihitung untuk memonitoring infus dilihat dari tetes per menit. Mengacu dari penelitian [5] dan maka penulis melakukan penambahan parameter estimasi kebutuhan cairan tubuh.

Penelitian [7] melakukan perancangan sistem pengawasan infus berbasis IoT. Pada penelitian ini bertujuan untuk meningkatkan kualitas pelayanan rumah sakit bagi pasien rawat inap, sistem yang dibangun diuji dengan menjalankannya. Hasil implementasi sistem *monitoring* infus berbasis teknologi IoT digunakan untuk klasifikasi kelayakan dengan mendapatkan nilai kelayakan sebesar 90% dari nilai kelayakan 100% yang diperoleh dari hasil pengujian sistem *monitoring*

infus jarak jauh. Dari penelitian terdapat saran untuk mengembangkan sistem ke dalam aplikasi android. Mengacu dari penelitian [7] maka penulis melakukan penelitian menggunakan aplikasi android untuk menampilkan data.

Penelitian [4] melakukan analisis kualitas layanan dari protokol MQTT untuk sistem pemantauan perangkat IoT. Seiring bertambahnya ukuran dan jumlah perangkat IoT dan penggunaan IoT menjadi lebih komplek, hal itu menyebabkan gangguan pada kinerja IoT dan perangkat IoT. Memecahkan masalah ini memerlukan manajemen perangkat yang dapat membangun sistem pemantauan untuk perangkat IoT, mendeteksi kegagalan perangkat, serta memantau dan menjaga ketersediaan perangkat IoT. Dengan mengimplementasikan protokol MQTT, MQTT berjalan pada lapisan aplikasi dan menjadi protokol dengan mekanisme *publish*. Protokol MQTT mendasarkan pengiriman dan penerimaan pesan pada topik tertentu, memungkinkan merancang pengiriman dan penerimaan pesan sesuai dengan kebutuhan kontrol dan pemantauan. Berdasarkan hasil pengujian, protokol MQTT terbukti memiliki kebutuhan sumber daya yang lebih rendah daripada protokol UDP dan ukuran paket yang lebih kecil daripada protokol SNMP. Sistem pemantauan perangkat IoT menggunakan protokol MQTT juga memiliki latensi rendah 0,008634 detik dan nilai *throughput* 9,2 Mbit/s. Sehingga sistem dapat berjalan walaupun pada *bandwith* yang rendah [4]. Melihat hasil perbandingan dari penelitian [4], protokol MQTT terbukti lebih baik dibandingkan protokol UDP dan SNMP, sehingga pada penelitian ini digunakan protokol MQTT.

Tabel 2. 1 Kajian penelitian sebelumnya

No	Penulis	Judul	Tahun	Pembahasan	Kelemahan
1	Raden Gumilar Riyansyah, dkk	*Smart Monitoring* Alat Infus Pasien Berbasis *Internet of Things*	2021	Alat yang digunakan adalah mikrokontroler ESP32 untuk menampilkan hasil monitoring melalui aplikasi berbasis *web* di ruang perawat. hasil pengujian berupa nilai	Selisih nilai berat sensor dengan nilai berat timbangan gantung digital yang besar. Nilai *error* yang didpatkan sebesar 4,48%

No	Penulis	Judul	Tahun	Pembahasan	Kelemahan
				persen dan rata-rata tetesan infus dengan rata-rata selisih berat antara sensor *load cell* dan timbangan manual	
2	Alwah Fanah Shinta	Rancang Bangun Sistem *Monitoring* Volume dan Laju Tetes Infus Pasien Menggunakan Nodemcu ESP8266	2020	Parameter yang diuji adalah uji *load cell*, uji *droplet* sensor dari 5 jenis pembacaan data untuk masing-masing 3 percobaan. Terjadi *delay* sebesar 19,6 detik dalam pengiriman data dari hasil pengujian aplikasi *load cell*	Perlu adanya perbaikan sensor tetesan agar dapat membaca tetesan lebih akurat
3	Achmad Herman Wijaya	Rancangan Bangun Sistem Pengawasan Infus Berbasis *Internet of Things*	2020	Hasil implementasi sistem monitoring infus berbasis teknologi IoT digunakan untuk klasifikasi kelayakan dengan mendapatkan nilai kelayakan sebesar 90% dari nilai kelayakan 100% yang diperoleh dari hasil pengujian sistem monitoring infus jarak jauh	Sistem yang dikembangkan masih berbentuk web
4	Zavero Brillianata Abilovani	Implementasi Protookol MQTT untuk Sistem	2018	Berdasarkan hasil pengujian, protokol MQTT terbukti	Tampilan dari *web monitoring* yang kurang menarik

No	Penulis	Judul	Tahun	Pembahasan	Kelemahan
		Monitoring Perangkat IoT		memiliki kebutuhan *resource* yang lebih kecil dibandingkan protokol UDP, dan memiliki ukuran paket yang lebih kecil dibandingkan protokol SNMP.	

2.2 DASAR TEORI

2.2.1 Infus

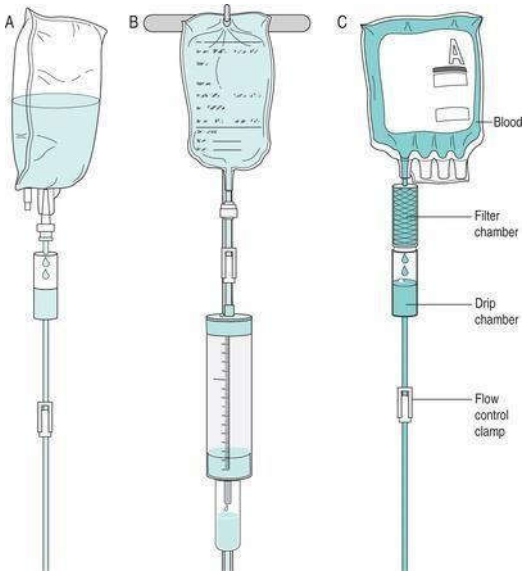
Dalam terapi infus, atau terapi intravena (IV), pasien diinfus melalui jarum ke pembuluh darah vena dari waktu ke waktu dengan cairan, elektrolit, obat intravena, dan nutrisi dalam jumlah terus menerus. Penggunaan cairan infus membutuhkan petunjuk yang akurat dan pemantauan teratur. Jenis dan dosis infus yang diberikan dokter kepada pasien bergantung pada kondisi medis pasien, usia, elektrolit, dan kadar gula darah. Kondisi pasien yang membutuhkan cairan infus sebagai berikut:

1. Dehidrasi
2. Demam tinggi
3. Stroke
4. Koma
5. Gangguan pencernaan
6. Gangguan fungsi organ
7. Infeksi parah
8. Luka bakar
9. Cedera serius
10. Persiapan pra operasi [8].

Prinsip kerja infus mirip dengan cara air mengalir dari tinggi ke rendah di bawah pengaruh gravitasi. Sistem kerjanya memungkinkan pengaturan volume infus menggunakan klem pada tabung infus. Memindahkan klem untuk memberi

tekanan lebih pada selang sehingga memperlambat aliran infus, dan jika klem digerakan melebar maka laju tetes infus semakin cepat. Infus kemudian mengalir melalui selang dari ruang tetes, di mana tetesan mengalir melalui selang ke dalam tubuh. Problem dalam pemberian infus juga menimbulkan risiko tinggi bagi kesehatan pasien, termasuk:

a. Tekanan yang tidak tepat atau tusukan berulang saat memasukkan jarum dapat menyebabkan pecahnya pembuluh darah arteri dan vena, sehingga terjadi pembekuan darah di jaringan tubuh.
b. Darah masuk ke dalam tabung infus atau tabung infus tidak segera diganti karena kehabisan infus.
c. Pembengkakan vena yang disebabkan oleh infus yang terhubung tidak dipantau dengan baik dan akurat.
d. Emboli udara adalah kondisi di mana gelembung udara masuk ke dalam pembuluh darah. Kondisi ini disebabkan oleh infus kosong di mana cairan mengubah gelembung udara.
e. Retensi tetesan yang disebabkan oleh pembekuan darah dan pembekuan darah dapat menyebabkan kematian karena dehidrasi internal pada orang yang mengalami dehidrasi parah. Pada Gambar 2.1 merupakan satu set peralatan infus[7].



Gambar 2.1 Set peralatan infus [7]

Untuk menghitung tetes per menit dapat dicari menggunakan rumus sebagai berikut[7]:

$$\text{Tetes/menit (macro)} = \frac{\text{Jumlah Cairan (ml)}}{\text{lamanya infus (jam)} x\ 3} \quad \textbf{(1)}$$

Kebutuhan cairan pada tubuh berdasarkan berat badan dan usia dapat dicari menggunakan rumus sebagai berikut [9]:

Usia $<$ 17 Tahun

Berat Badan	Kebutuhan Cairan
0 - 10 Kg	100 ml x Berat badan
10 -20 Kg	1000 ml + (50 ml x berat badan)
> 20 Kg	1500 ml + (20 ml x (berat badan - 20))

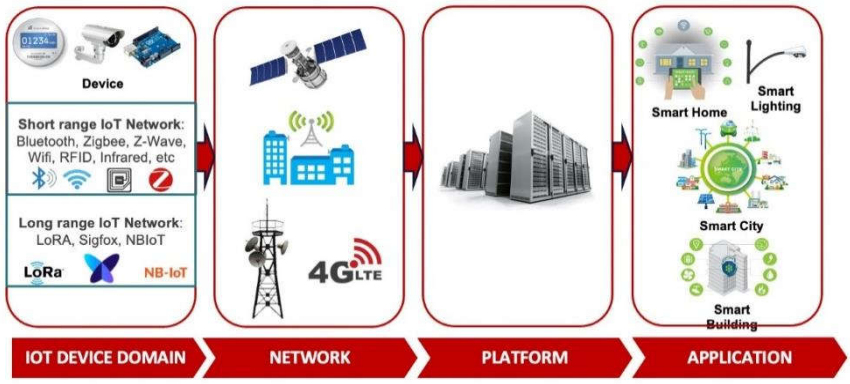
(2)

Usia > 17 Tahun

Kebutuhan Cairan=30ml x berat badan

2.2.2 *Internet of Things* (IoT)

Internet of Things adalah teknologi yang menghubungkan mesin, perangkat, dan objek fisik lainnya dengan sensor dan aktuator jaringan untuk mengumpulkan data dan memungkinkan mereka mengelola kinerjanya sendiri. Ini memungkinkan mesin untuk bekerja sama atau bahkan bereaksi secara independen terhadap informasi yang baru diperoleh [10].



Gambar 2.2 Arsitektur *Internet of Things* [10]

Pada Gambar 2.2 merupakan Arsitektur IoT yang terdiri dari empat aspek, aspek yang pertama yaitu *device* terdiri dari perangkat IoT seperti sensor dan

perangkat yang dapat terkoneksi ke *internet.* Perangkat ini bisa berkomunikasi dengan *network* dan layanan melalui protokol komunikasi. Kemudian aspek yang kedua yaitu *network* terdiri dari jaringan perangkat dan teknologi yang memungkinkan alat untuk berkomunikasi dengan *internet*. Beberapa protokol jaringan yang digunakan dalam IoT termasuk *WiFi*, *Bluetooth, ZigBee, dan LoRaWAN*. Kemudian aspek yang ketiga yaitu platform terdiri dari perangkat lunakdan layanan yang berfungsi mengelola data yang telah dikumpulkan oleh perangkat IoT dan menyediakan layanan seperti manajemen perangkat, manajemen data, analisis data, dan terhubung dengan *cloud.* Kemudian aspek keempat yaitu *application* terdiri dari aplikasi dan layanan yang difungsikan untuk memproses serta menganalisis data pada perangkat IoT sehingga memungkinkan pengguna untuk memantau, mengontrol, dan mengotomatisasi perangkat IoT.

2.2.3 Mikrokontroler

Mikrokontroler adalah *chip* mikrokomputer fisik dalam bentuk IC (*integrated circuit*). Mikrokontroler umumnya digunakan dalam sistem yang kecil, murah, dan tidak memerlukan komputasi yang sangat rumit, seperti aplikasi PC. Mikrokontroler digunakan dalam *oven microwave, keyboard*, pemutar CD, VCR, *remote control,* robot, dan perangkat lainnya. Mikrokontroler terdiri dari bagian-bagian utama: CPU (*central processing unit*), RAM (*random access memory*), ROM (*read only memory*), dan port I/0 (*input/output*). Mikrokontroler perangkat keras yang dapat digunakan untuk berbagai keperluan seperti statistik, komunikasi serial, dan interupsi. Beberapa mikrokontroler memiliki ADC bawaan (konverter analog ke digital), pengontrol USB, CAN (jaringan area pengontrol), dll. Mikrokontroler beroperasi atas dasar program tertanam (perangkat lunak) dan membuat program sesuai kebutuhan. Aplikasi mikrokontroler sering berhubungan dengan pembacaan data dan pengendalian perangkat eksternal [8].

2.2.4 Prinsip kerja sensor *load cell*

Prinsip kerja pada sensor *load cell* adalah mengukur sebuah massa atau tekanan yang mengakibatkan terjadinya perubahan resistansi yang akan dikonversikan menjadi elektrik yang dapat terukur. Jika sensor *load cell* tidak ada

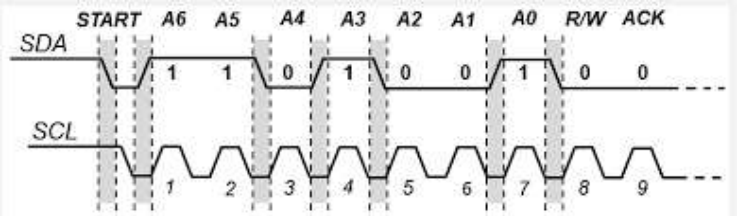
massa atau tekanan, resistansinya akan bernilai sama dengan sisi yang lainnya. Namun jika sensor *load cell* memiliki beban maka nilai resistansinya akan menjadi berat sebelah. Proses ini yang dimanfaatkan menjadi pengukur berat pada suatu benda [10].

2.2.5 Prinsip kerja modul HX711

Prinsip kerja modul HX711 adalah konversi perubahan resistansi menjadi besaran tegangan pada rangkaian. Modul HX711 melakukkan pengolahan data yang ada di sensor *load cell* kepada mikrokontroler. Selanjutnya modul HX711 akan mengolah data yang diperoleh dari sensor *load cell* berupa massa atau beban. Selanjutnya modul HX711 akan menerjemahkan data yang dikirim sensor *load cell* ke NodeMCU ESP8266 [10].

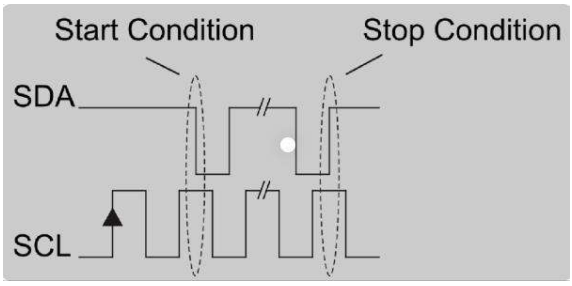
2.2.6 I2C

I2C (*Inter-integrated circuit*) adalah koneksi yang dibuat untuk komunikasi antar perangkat. Komunikasi I2C menggunakan protokol yang hanya menggunakan dua kabel untuk komunikasi: *synchronous clock* (SCL) dan *synchronous data* (SDA). Data dikirim secara berurutan dari *master* ke *slave* dan kemudian dari *slave* ke *master*. *Master* adalah perangkat yang menghasilkan *clock,* memulai dan mengakhiri tranmisi pada sistem. *Slave node* adalah *node* yang menerima *clock* yang merupakan alamat yang dikirimkan oleh *master node*. Baik master maupun *slave* dapat menerima dan mengirim data. I2C adalah protokol komunikasi serial di mana setiap bit data yang dikirimkan pada jalur SDA disinkronkan dengan pulsa clock pada jalur SCL. Ketika *clock* pada keadaan *high*, jalur data tidak dapat berubah. Pada I2C, setiap alamat atau data yang ditransfer harus membentuk sebuah paket dengan panjang 9 bit, di mana 8 bit pertama disimpan oleh *transmitter* pada jalur SDA dan bit ke-9 merupakan *acknowledge* (atau *not acknowledge*) oleh *receiver*. Pada Gambar 2.3 merupakan bentuk data dari protokol I2C.



Gambar 2.3 Bentuk data protokol I2C [11]

Jika pada SDA, terdapat transisi dari *high state* ke *low state* saat SCL *high*, maka terjadi kondisi *start*. Jika SDA terjadi transisi dari keadaan *low* ke keadaan *high* saat SCL dalam keadaan *high*, yaitu terjadi kondisi *stop*. Kondisi *start* dan *stop* selalu ditentukan oleh *master* dan bus dikatakan sibuk pada kondisi setelah *start* dan menganggur setelah *stop*. Gambar 2.4 merupakan gambaran kondisi *stop* dan *start* pada komunikasi I2C [11].

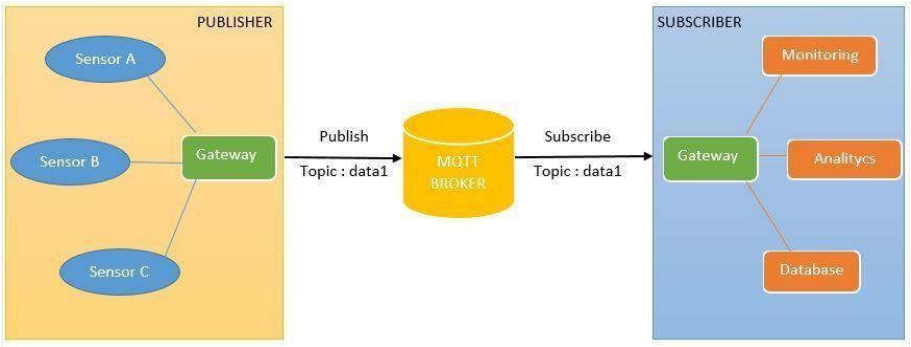


Gambar 2. 4 Kondisi start dan stop pada komunikasi I2C [11]

2.2.7 *Message Queuing Telemetry Transfer Protocol* (MQTT)

MQTT merupakan protokol jaringan lapisan aplikasi yang bekerja di atas *stack* TCP/IP atau protokol lainnya yang bersifat *lossless*. MQTT adalah teknologi untuk transmisi yang dioptimalkan melalui jaringan *bandwidth* rendah, perangkat komputasi hemat energi, yang dikembangkan oleh IBM, untuk berbagai skenario aplikasi *Internet Of Things*. Sistem komunikasi MQTT terdiri dari tiga komponen, yaitu *publisher*, *broker*, dan *subscriber*. *Publisher* (*client* MQTT) adalah perangkat yang terhubung dengan sensor dan mikrokontroler yang fungsinya untuk mengirimkan data dengan topik atau grup data yang berbeda. *Subscriber* (*client* MQTT) bertindak sebagai penerima data yang dikirim dengan memilih data mana yang akan diterima. Proses serah terima data dihubungkan oleh *broker* (*server* MQTT) sebagai perantara, yang mengurutkan semua data yang dikirim oleh penerbit dan meneruskannya ke pelanggan yang relevan. Setelah mendapatkan data

dari sensor di *publisher*, data tersebut akan diolah untuk dimasukkan ke dalam basis data, kemudian akan diproses menjadi sebuah sistem *monitoring* yang tertata. Dalam IoT, data telemetri yang terdiri dari sensor dan aktor berkomunikasi melalui *broker.* Pada gambar 2.5 merupakan ilustrasi dari sistem MQTT[12].



Gambar 2.5 Sistem MQTT [12]

2.2.8 Kualitas Layanan / *Quality of service* (QoS)

Kualitas layanan adalah kemampuan suatu layanan untuk memberikan kinerja dan merupakan parameter yang digunakan untuk mengukur kualitas layanan. Pada penelitian ini parameter yang dianalisa adalah *delay, packet loss, jitter*, dan *throughput.* Kualitas layanan juga merupakan istilah umum untuk keseluruhan dampak kualitas layanan dari sudut pandang pengguna [13].

Quality of Service (QoS) dapat dianggap sebagai istilah yang digunakan untuk mendefinisikan karakteristik layanan untuk memahami baik atau buruknya suatu kualitas layanan[14]. Pada penelitian ini parameter QoS yang dianalisa adalah *delay, jitter, throughput*, dan *packet loss*.

2.2.9 *Throughput*

Throughput adalah ukuran seberapa cepat data dapat dikirim melalui jaringan, dinyatakan dalam bit per detik (bps). Istilah *bandwidth* terkadang digunakan sebagai sinonim untuk *throughput*. *Throughput* juga bisa dikatakan *bandwidth* yang sebenarnya. *Throughput* adalah jumlah total paket yang tiba di tujuan selama interval waktu tertentu dibagi dengan durasi interval waktu tersebut.

Throughput adalah kemampuan suatu jaringan untuk benar-benar mentransmisikan data. Mengukur *throughput* dapat menggunakan persamaan (3) [15].

Gambar 2.6 *Troughput* [15]

Merujuk pada Gambar 2.6 *throughput* diibaratkan sebagai air yang mengalir melalui pipa yang besaranya dibatasi oleh besarnya pipa (*bandwidth*) sehingga *throughput* tidak akan melebihi besar kapasitas *bandwidth*.

$$\text{Throughput} = \left(\frac{\text{Jumlah data yang dikirim}}{\text{Waktu pengiriman data}}\right) \quad \textbf{(3)}$$

Dari perhitungan di atas dapat kita lihat bahwa untuk mendapatkan nilai *throughput*, kita perlu membagi jumlah data yang terkirim dengan waktu pengiriman data.

2.2.10 *Packet loss*

Packet loss akan terjadi pada saat kebutuhan *bandwidth* melebihi *bandwidth* yang tersedia. Hal ini bisa terjadi pada level *network* secara keseluruhan, atau dapat terjad per *user*. Sebagai contoh *user* A dialokasikan *bandwidth* sebesar 10 Mbps, tetapi *user* A membutuhkan *bandwidth* lebih dari 10 Mbps maka akan terjadi *packet loss.* Pada dasarnya *packet loss* merupakan paket yang hilang pada saat pengiriman data berlangsung.

Adapun rumus perhitungan untuk mendapatkan hasil presentase nilai *packet loss* yaitu :

$$\text{Packet loss} = \left(\frac{\text{data yang dikirim-paket data yang diterima}}{\text{paket data yang dikirim}}\right) \text{x } 100\ \% \quad \textbf{(4)}$$

Setelah menghitung rumus di atas, kita bisa mencari nilai *packet loss* dengan cara mengurangkan jumlah data yang dikirim dari paket data yang diterima dan

membagi hasil selisihnya dengan jumlah paket yang dikirim. Kemudian, kalikan hasil pembagian dengan 100 untuk menentukan persentase nilai *packet loss*.

2.2.11 *Delay*

Delay adalah waktu jeda sebuah paket oleh proses pengiriman dari satu titik ke titik tujuan. Delay dipengaruhi oleh media fisik, jarak, kemacetan, atau waktu pemrosesan yang lama. Hal ini disebabkan oleh antrian panjang atau kemacetan lalu lintas. Semakin rendah *delay*, semakin tinggi kualitas jaringan.

$$\text{Rata-rata } Delay = \frac{\text{Total } Delay}{\text{Total paket yang diterima}} \quad \textbf{(5)}$$

Untuk menghitung nilai rata-rata *delay* dengan cara, membagi total *delay* yang diterima dengan jumlah total paket yang diterima. Maka dari itu, membagi *delay* total dengan jumlah total paket yang diterima dapat menghasilkan nilai *delay* rata-rata.

2.2.12 *Jitter*

Jitter adalah variasi *delay* antar paket dalam jaringan IP. Besarnya nilai *jitter* akan sangat dipengaruhi oleh perubahan beban trafik dan besarnya tabrakan antar paket (*congestion*) yang ada dalam jaringan IP. Semakin besar beban trafik di dalam jaringan akan menyebabkan semakin besar pula peluang terjadinya *congestion,* dengan demikian nilai *jitter* akan semakin besar.

Nilai *jitter* yang lebih besar akan menghasilkan kualitas nilai layanan yang lebih rendah. Untuk mendapatkan nilai QoS jaringan yang baik, nilai *jitter* harus dijaga seminimal mungkin.

$$Jitter = \frac{\text{Total Variasi } Delay}{\text{Total paket } Delay - 1} \quad \textbf{(6)}$$

Seperti dapat dilihat dari rumus di atas, nilai *jitter* dihitung dengan membagi nilai total *delay* dengan jumlah total paket yang diterima dikurangi 1[17].

2.2.13 *Software* MIT App Inventor

MIT app inventor adalah platform pemrograman visual berbasis blok yang dirancang untuk membuat dan mengembangkan aplikasi *smartphone* dan tablet melalui situs *web* dan ponsel atau emulator yang terhubung. Fitur-fitur yang termasuk dalam platform memungkinkan untuk membuat aplikasi yang rumit dan terperinci dalam waktu yang lebih singkat daripada aplikasi yang diprogram secara tradisional.

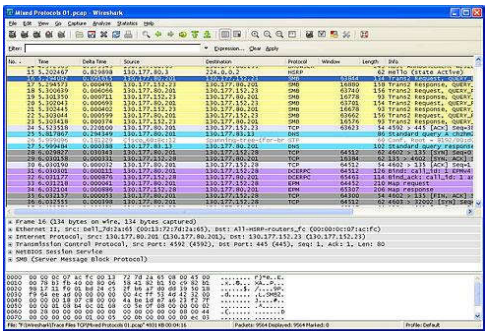
Server yang termasuk dalam MIT app inventor menyimpan pekerjaan sehingga mereka dapat terus bekerja kapan pun dibutuhkan. Platform ini memiliki dua fungsi antarmuka untuk membuat aplikasi, app inventor *designer* sebagai platform untuk memilih komponen atau elemen yang diperlukan dalam aplikasi, dan app inventor *block editor* sebagai platform untuk menggabungkan blok program, blok program ini mengatur komponen kerja dari aplikasi yang telah dipilih sebelumnya. Setelah kompilasi selesai, program aplikasi akan muncul di ponsel satu per satu, sehingga program aplikasi dapat diuji di ponsel. Pada Gambar 2.7 merupakan tampilan dari MIT App inventor[18].

Gambar 2.7 MIT app inventor[18]

2.2.14 *Software Wireshark*

Perangkat lunak wireshark yang bertindak sebagai penganalisa protokol jaringan. Metode pengukuran ini menggunakan wireshark yang berfungsi sebagai *network analyzer*, yang berguna untuk mengubungkan antar *client* saling berkomunikasi agar *client* 1 (pengirim) dapat mengirimkan paket dari ke *client* 2 (penerima) dan sebaliknya melalui server. Pada penelitian ini, penulis

menggunakan wireshark untuk mengukur kualitas layanan dari protokol MQTT . Pada gambar 2.8 merupakan tampilan dari *software* wireshark[19].



Gambar 2.8 *Software Wireshark* [19]

2.2.15 *Software* Arduino IDE

Salah satu perangkat lunak yang digunakan dalam penelitian ini adalah Arduino *integrated development environment* (IDE). IDE didefinisikan sebagai pemograman yang digunakan untuk menulis scrip perintah, memeriksa kesalahan skrip, dan menyertakan program yang dibuat pada *disk* dalam satu applikasi. Bahasa pemograman yang digunakan pada Arduino IDE merupakan turunan dari Bahasa C/C++ . Pada gambar 2.9 merupakan tampilan logo *software* Arduino IDE[20].

Gambar 2.9 *Software* Arduino IDE

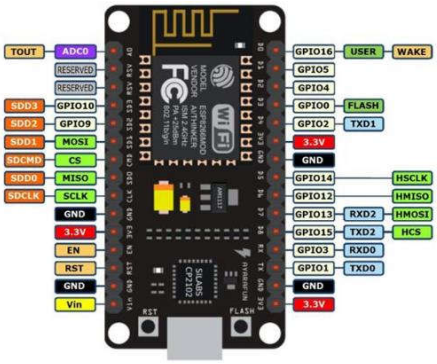
2.2.16 Telkom IoT Platform

Platform IoT memiliki peran penting untuk mengumpulkan data yang dikirim dari perangkat IoT untuk diproses oleh aplikasi IoT melalui protokol yang aman dan andal seperti MQTT. Dibentuk oleh layanan mikro, platform IoT memungkinkan untuk membuat aplikasi IoT untuk pemantauan dan kontrol. Platform IoT menawarkan fitur unik seperti manajemen perangkat, *interoperabilitas* dari berbagai merek perangkat, agnostik ke konektivitas apa pun, dan dasbor pribadi. Pada Gambar 2.10 merupakan logo dari Telkom IoT platform.

Gambar 2.10 Telkom IoT platform

2.2.17 NodeMCU ESP8266

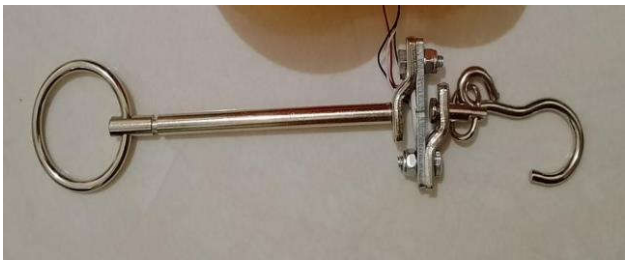
Nodemcu Nodemcu adalah platform IoT *open-source* dan kit pengembangan yang menggunakan bahasa pemrograman untuk memungkinkan pembuat membuat produk IoT dan menggunakan sketsa di Arduino IDE. ESP8266 adalah modul *WiFi* SoC (*System on Chip*), sehingga pemrograman dapat dilakukan langsung di ESP8266. Terdapat beberapa pin I/O sehingga dapat dikembangkan menjadi sebuah aplikasi *monitoring* maupun *controlling* pada proyek IoT. NodeMCU ESP8266 dapat diprogram dengan Arduino IDE. NodeMCU ESP8266 merupakan modul turunan pengembangan dari modul platform loT keluarga ESP8266 tipe ESP-12. Secara fungi modul ini hampir menyerupai dengan platform modul arduino, tetapi yang membedakan yaitu dikhususkan untuk *"Connected to Internet"*. Pada Gambar 2.11 merupakan tampilan pin I/O pada NodeMCU ESP8266[6].



Gambar 2.11 NodeMCU ESP8266 [6]

2.2.18 Sensor Berat Timbangan Digital/ *Loadcell*

Load cell adalah sensor yang menghasilkan keluaran sebanding dengan beban atau gaya. Bentuknya berubah tergantung pada beban atau gaya, dan hambatannya berubah. *Load cell* biasanya digunakan untuk menghitung massa suatu benda. Sensor sel beban terdiri dari konduktor, pengukur regangan, dan jembatan *wheatstone*. Fungsi pengukur regangan adalah untuk mengukur perubahan bentuk dengan perubahan resistansi [21]. Pada Gambar 2.12 merupakan sensor *load cell* yang terdapat pada timbangan digital.



Gambar 2.12 *Load cell* pada timbangan digital

2.2.19 Modul HX711

Modul HX711 merupakan modul amplifier yang biasa digunakan pada rangkaian timbangan digital. Modul HX711 bertindak sebagai *load cell* modul pengubah sinyal analog ke sinyal digital. Modul HX711 menampilkan *input* ADC 24-bit presisi, gain tinggi, yang dirancang khusus untuk berbagai sensor jembatan *wheatstone*. Prinsip kerja modul HX711 adalah sebagai penguat tegangan *load cell* selama pengoperasian *load cell*. Pada Tabel 3.1 merupakan pin yang akan dipakai untuk menghubungkan modul HX711 ke mikrokontroler NodeMCU ESP8266 [6].

Tabel 3.1 Konfigurasi pin dari modul HX711 ke Nodemcu

Pin Nodemcu	Pin HX711
D6	DT
D7	SCK
Vcc	Vcc
Ground	Gnd

2.2.20 Infus

Infus yang digunakan dalam penelitian ini menggunakan Sanbe Hest dengan volume infus 500ml, formulasi infus terapi mempercepat proses penyembuhan pasien dan pencegahan kondisi penurunan volume darah. Sanbe Hest 200 diindikasikan untuk pengobatan hemoragik (pecahnya pembuluh darah), traumatis (kerusakan jiwa akibat peristiwa), syok septik, luka bakar, dan cedera, serta untuk pengobatan dan pencegahan hipovolemia. Pada Gambar 2.13 merupakan infus yang digunakan pada penelitian ini.

Gambar 2.13 Infus