Hak cipta dan penggunaan kembali:

Lisensi ini mengizinkan setiap orang untuk menggubah, memperbaiki, dan membuat ciptaan turunan bukan untuk kepentingan komersial, selama anda mencantumkan nama penulis dan melisensikan ciptaan turunan dengan syarat yang serupa dengan ciptaan asli.

Copyright and reuse:

This license lets you remix, tweak, and build upon work non-commercially, as long as you credit the origin creator and license it on your new creations under the identical terms.

Team project ©2017
Dony Pratidana S. Hum | Bima Agus Setyawan S. IIP

BAB II
LANDASAN TEORI

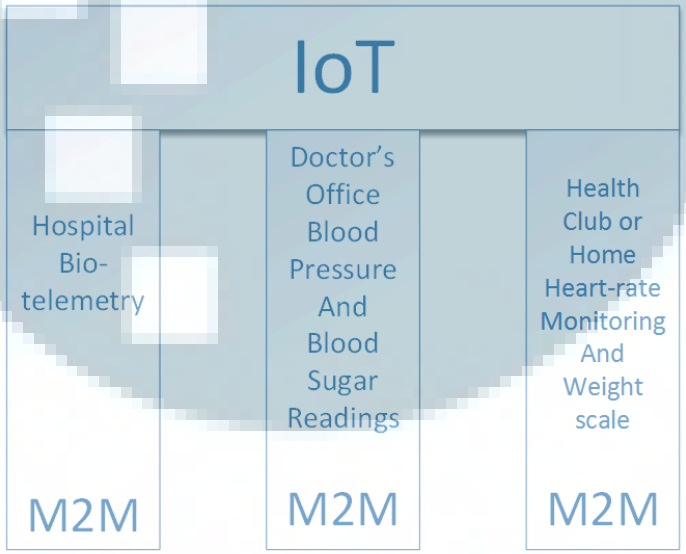
2.1 Internet of Things

Internet of Things (IoT) adalah sebuah skenario di mana suatu obyek, binatang, atau manusia memiliki suatu pengenal yang unik satu sama lain dan mampu mentransmisikan data melalui jaringan tanpa memerlukan interaksi antar manusia maupun manusia dengan komputer. IoT telah berkembang dengan menyatunya teknologi nirkabel, *micro-electromechanical systems* (MEMS) dan Internet [4]. Perangkat dan sensor yang saling terhubung dalam IoT memungkinkan manusia untuk memanfaatkan dan menganalisa data untuk mendapatkan informasi berharga yang selama ini berada di luar jangkauan [5].

Machine-to-Machine (M2M) *communication* adalah sebuah istilah umum yang dapat digunakan untuk mendeskripsikan teknologi apapun yang memungkinkan perangkat-perangkat terhubung ke dalam suatu jaringan lokal atau privat [6], agar dapat saling bertukar informasi dan menjalankan suatu aksi tanpa campur tangan manusia. Komunikasi ini merupakan konsep awal IoT [7].

Sistem M2M biasanya dapat diandalkan, tangguh, dan dirancang untuk tetap bekerja selama bertahun-tahun. Namun, sebagian besar dibangun untuk melaksanakan satu tujuan tertentu dalam satu pengaturan, bergantung pada perangkat keras, lunak dan jaringan yang spesifik. Integrasi vertikal ini menimbulkan ketergantungan pada suatu *vendor*

tertentu dan membatasi ruang untuk pengembangan. Berbeda dengan sistem M2M, IoT dibangun berdasarkan pada komunikasi menggunakan IP (*Internet Protocol*) dan hubungan horizontal antara perangkat keras, lunak dan konektivitasnya, sehingga teknologi-teknologi yang dihasilkan oleh berbagai *vendor* berbeda juga dapat saling terhubung [6]. IoT menghubungkan pilar-pilar vertikal sistem M2M agar dapat saling bertukar informasi dengan sistem lainnya, dengan menggabungkan keandalan M2M dan konektivitas IoT yang luas [8].



Gambar 2.1 Ilustrasi hubungan antara IoT dan M2M [8]

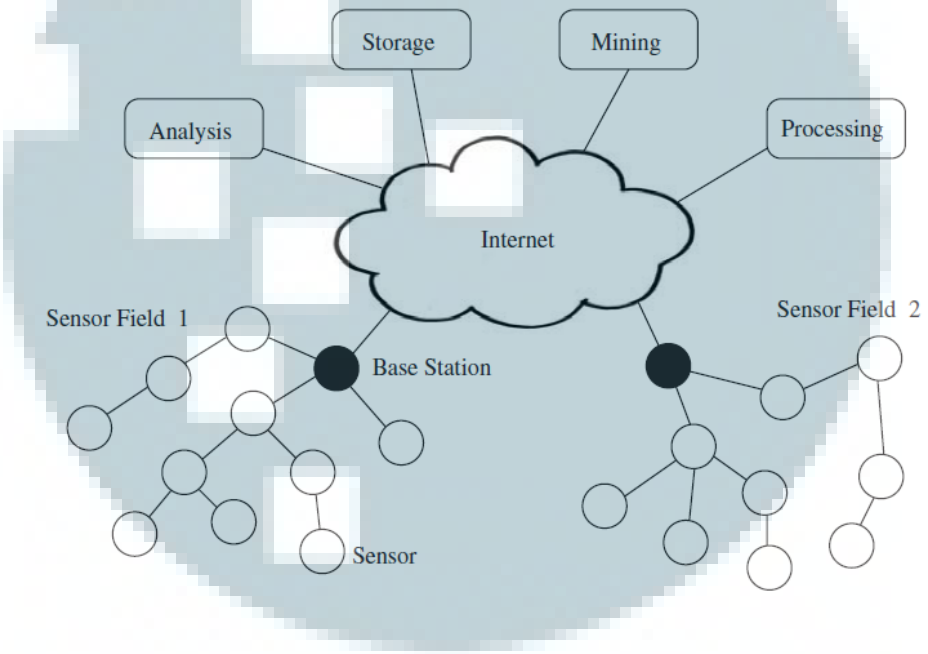
Perangkat pintar merupakan komponen IoT yang saling terhubung melalui teknologi jaringan seperti WiFi untuk berkomunikasi dengan perangkat lainnya, atau dengan *cloud*. Beberapa juga terhubung dengan *central hub* yang berperan sebagai pusat kendali. Secara ideal, pemilik perangkat dapat mengakses *central access point* melalui *smartphone* dan *tablet*, baik di dalam maupun di luar rumah [9].

Agar perangkat-perangkat di dalamnya dapat saling bertukar informasi, IoT tidak terlepas dari *wireless sensor network* (WSN), sekelompok transduser khusus dengan infrastruktur komunikasi untuk mengawasi dan merekam kondisi lingkungan berbeda. Besaran-besaran yang umumnya dipantau adalah suhu, kelembapan, tekanan, arah dan kecepatan angin, intensitas cahaya, getaran, dan suara, tegangan listrik, konsentrasi bahan kimia, dan fungsi-fungsi vital pada tubuh [10].

Sebuah jaringan sensor terdiri dari beberapa stasiun deteksi yang disebut *sensor node*, yang masing-masing berukuran kecil, ringan dan portabel. Setiap *sensor node* dilengkapi dengan transduser, komputer mikro, *transceiver* dan sumber daya. Transduser menghasilkan sinyal listrik berdasarkan efek fisik dan fenomena yang dirasakan. Komputer mikro mengolah dan menyimpan *output* dari sensor. *Transceiver* menerima perintah dari komputer pusat dan mengirimkan data ke komputer tersebut. Daya untuk setiap *sensor node* berasal dari baterai [10].

Terdapat empat komponen dasar dalam sebuah *sensor network*, yaitu (1) kumpulan dari sensor-sensor terdistribusi/terlokalisasi; (2) jaringan yang saling terhubung (biasanya berupa jaringan nirkabel); (3) titik pusat pengumpul informasi; dan (4) sekumpulan sumber daya pengolahan pada titik pusat (atau di belakangnya) yang bertugas untuk mengatur korelasi data, *event trending*, *status querying*, dan *data mining* [11]. Sehingga *sensor node* tidak hanya bertugas untuk melakukan pengumpulan data, tapi juga melakukan berbagai pengolahan data pada data yang dikumpulkan

dan dari *sensor node* lainnya [12]. *Sensor node* tidak hanya berkomunikasi dengan *sensor node* lainnya, melainkan juga dengan *base station* (BS) melalui komunikasi radio. Gambar 2.2. menunjukkan dua buah *sensor field* yang memantau dua kondisi geografis di dua wilayah berbeda dan terhubung ke internet melalui BS masing-masing.



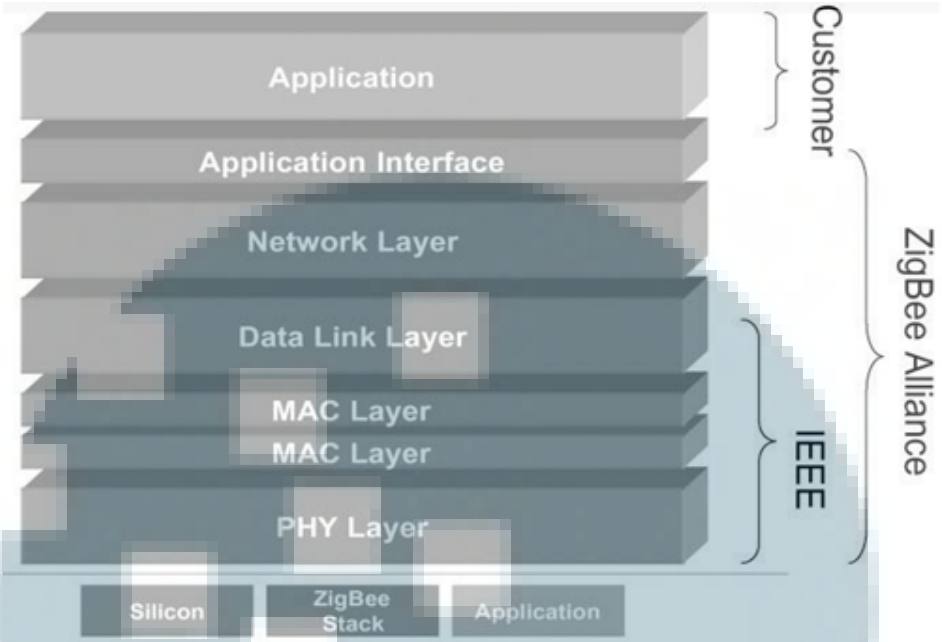
Gambar 2.2 Wireless sensor networks *[12]*

Ada beberapa pilihan komunikasi radio yang dapat diimplementasikan agar suatu WSN di suatu wilayah tertentu dapat berkomunikasi dengan WSN di wilayah lainnya. Komunikasi tersebut dapat menggunakan protokol yang sudah tersedia saat ini yaitu IEEE802.11x (WiFi), dan IEEE802.15.4 (ZigBee) [13].

IEEE802.15.4 merupakan standar yang secara spesifik dikembangkan untuk diaplikasikan pada *wireless sensing*. Standar ini memiliki kecepatan transfer dan frekuensi yang bervariasi. Daya yang

dibutuhkan cenderung rendah dan perangkat kerasnya dirancang agar dapat masuk ke *sleep mode*, yang dapat membantu mengurangi penggunaan daya. ZigBee sendiri merupakan standar yang dibentuk oleh ZigBee Alliance dengan menggunakan protokol IEEE802.15.4 pada lapisan *physical* dan MAC [13]. Karakteristik IEEE802.15.4 adalah sebagai berikut:

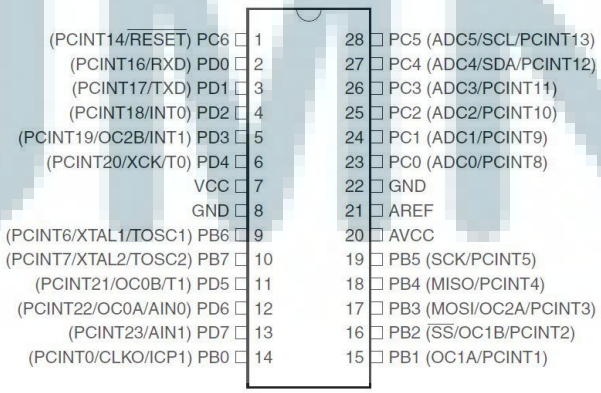
1. Frekuensi transmisi, 868 MHz/902-928 Mhz/2.48-2.5GHz (frekuensi 2.4 GHz merupakan frekuensi bebas lisensi);
2. Kecepatan transfer data 20 Kbps (868 MHz), 40 Kbps (902 MHz), dan 250 Kbps (2.4 GHz), cocok untuk transmisi data bacaan sensor yang ukurannya relatif lebih kecil;
3. Mendukung topologi jaringan *star* dan *peer-to-peer* (*mesh*);
4. Penggunaan AES-128 untuk enkripsi data yang ditransmisikan;
5. Indikasi kualitas hubungan, yang berguna untuk algoritma jaringan *multi-hop mesh*;
6. Menggunakan *direct sequence spread spectrum* (DSSS) untuk menghasilkan komunikasi yang lebih tahan gangguan.



Gambar 2.3 ZigBee stack *[14]*

2.2 Mikrokontroler ATmega328

Atmega328 merupakan mikrokontroler AVR 8-bit yang berbasis RISC (Reduced Instruction Set Computing) dengan memori ISP *flash* 32KB dengan kemampuan membaca-sambil-menulis, 1KB EEPROM (*Electrically Erasable Programmable Read-Only Memory*), dan 2KB SRAM (*Static Random-Access Memory*) [15].



Gambar 2.4 Konfigurasi pin ATmega328 [16]

Arduino Uno [17] pada Gambar 2.2 merupakan *board* yang dirancang berdasarkan mikrokontroler ATmega328. Adapun fitur yang dimiliki *board* ini adalah sebagai berikut:

Tabel 2.1 Fitur pada mikrokontroler board Arduino Uno [17]

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz
Length	68.6 mm
Width	53.4 mm
Weight	25 g

Gambar 2.5 Microntroller board Arduino Uno [17]

Arduino Uno dapat diaktifkan melalui koneksi USB atau dengan catu daya eksternal. Sumber daya dipilih secara otomatis. Sumber listrik eksternal (non-USB) bisa didapatkan dari adaptor AC-DC. Adaptor dapat dihubungkan dengan mencolokkan steker 2.1mm pusat-positif ke colokan

listrik pada *board*. Pin untuk mengakses sumber daya adalah sebagai berikut:

- **VIN**. Tegangan input ke papan Arduino ketika menggunakan sumber daya eksternal (sebagai lawan 5 volt dari koneksi USB atau sumber daya teratur lainnya). Tegangan dapat dialirkan melalui pin ini, atau, jika memasok tegangan melalui colokan listrik, mengaksesnya melalui pin ini.
- **5V**. Pin ini mengeluarkan 5V teratur dari regulator pada *board*, dapat dialiri dengan daya baik dari colokan listrik DC (7 - 12V), konektor USB (5V), atau pin VIN *board* (7-12V).
- **3V3**. Sebuah listrik 3,3 volt yang dihasilkan oleh regulator *on-board*. Maksimum kuat arusnya adalah 50 mA.
- **GND**. Pin *ground*.
- **IOREF**. Pin ini memberikan tegangan referensi yang digunakan oleh mikrokontroler untuk beroperasi. *Shield* yang dikonfigurasi dengan benar dapat membaca tegangan pin IOREF dan memilih sumber tegangan yang tepat atau mengaktifkan penerjemah tegangan pada keluaran yang bekerja dengan 5V atau 3.3V.

Masing-masing dari 14 pin digital pada Uno dapat digunakan sebagai *input* atau *output*, menggunakan fungsi pinMode(), digitalWrite(), dan digitalRead(), dan beroperasi pada tegangan 5 volt. Setiap pin dapat memberikan atau menerima maksimum 40 mA dan memiliki resistor *pull-up*

internal (yang terputus secara default) sebesar 20-50 kOhms. Selain itu, beberapa pin memiliki fungsi khusus:

- **Serial: 0 (RX) and 1 (TX).** Digunakan untuk menerima (RX) dan mengirimkan (TX) TTL data serial. Pin ini terhubung ke pin yang sesuai pada ATmega8U2 USB-to-TTL Serial Chip.
- **External Interrupts: 2 and 3.** Pin ini dapat dikonfigurasi untuk memicu interupsi.
- **PWM: 3, 5, 6, 9, 10, dan 11.** Menyediakan keluaran PWM 8-bit menggunakan fungsi analogWrite().
- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** Pin ini mendukung komunikasi SPI menggunakan SPI *library*.
- **LED: 13.** Terdapat sebuah *built-in* LED yang terhubung dengan pin digital 13. Pada saat nilai pin *HIGH*, LED-nya menyala. Ketika nilai pin *LOW*, LED-nya padam.

Uno memiliki 6 input analog, yaitu A0 sampai A5, yang masing-masing menyediakan 10 bit resolusi.

2.3 Sensor

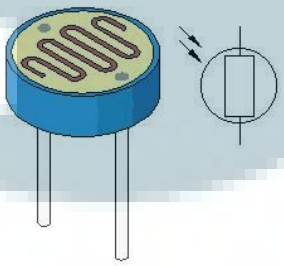
Menurut [18], sensor adalah "transduser yang berfungsi untuk mengolah variasi gerak, panas, cahaya atau sinar, magnetis, dan kimia menjadi tegangan serta arus listrik." Masing-masing sensor memiliki karakteristiknya masing- masing. Berikut ini adalah karakteristik penting mengenai sensor yang harus diperhatikan dalam perancangan suatu sistem [19]:

- ***Transfer function*** – Fungsi transfer menunjukkan hubungan fungsional antara *input* fisik sinyal dan *output* listrik sinyal.
- ***Sensitivity*** – Sensitivitas umumnya merupakan perbandingan antara suatu perubahan pada sinyal elektrik terhadap perubahan pada sinyal fisik. Misalnya, sebuah termometer dapat dikatakan sangat sensitif apabila perubahan suhu yang kecil menyebabkan perubahan tegangan yang besar.
- ***Span or dynamic range*** – Span atau dynamic range adalah rentang sinyal input fisik yang dapat diubah menjadi sinyal listrik oleh sensor.
- ***Accuracy or uncertainty*** – Uncertainty biasanya didefinisikan sebagai error terbesar yang dapat diperkirakan antara sinyal output yang ideal dengan yang sesungguhnya.
- ***Hysteresis*** – Suatu sensor tidak selalu memberikan keluaran yang sama, apabila rangsangan yang dibaca mengalami perubahan naik maupun turun. *Hysteresis* merupakan istilah yang digunakan sebagai perkiraan besarnya *error* dari kondisi tersebut.
- ***Nonlinearity*** – Deviasi maksimum dari sebuah fungsi transfer linear terhadap suatu rentang dinamis yang telah ditentukan.
- ***Noise*** – *Noise* merupakan *output* yang juga dihasilkan oleh sensor, selain sinyal *output*.
- ***Resolution*** – *Resolution* merupakan nilai fluktuasi sinyal terkecil yang dapat dibaca oleh sensor.

- ***Bandwidth*** – *Bandwidth* merupakan rentang frekuensi antara frekuensi *cutoff* tertinggi dan terendah dari suatu sinyal fisik.

2.3.1 Sensor Cahaya – Photo Resistor

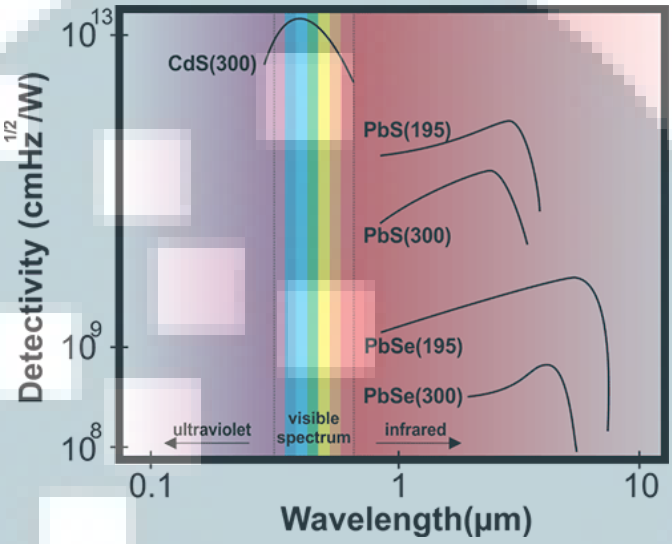
Photo resistor atau *light dependant resistor* (LDR) merupakan perangkat yang peka terhadap cahaya dan pada umumnya digunakan untuk mengindikasikan keberadaan dan ketiadaan cahaya atau mengukur intensitas cahaya [20]. Satuan ukuran cahaya pada LDR adalah *illumination* (Lux) [21]. Resistansi pada *photo resistor* bergantung pada terang gelapnya cahaya. Semakin terang, resistansinya akan semakin berkurang, dan semakin gelap, akan semakin bertambah. Maka, dapat disimpulkan bahwa resistansi *photo resistor* berbanding terbalik dengan intensitas cahaya.



Gambar 2.6 Bentuk fisik dan simbol photo resistor *[22]*

Sensitivitas *photo resistor* bervariasi tergantung pada panjang gelombang cahaya. Apabila panjang gelombang cahaya berada di luar kisaran tertentu, maka tidak akan berpengaruh pada perubahan resistansinya. Panjang gelombang yang dapat ditangkap oleh *photo resistor* bervariasi pada bahan yang digunakan untuk membuat resistor tersebut [20]. Rentang panjang gelombang yang dapat ditangkap oleh *photo resistor* dapat dilihat pada Gambar 2.6. Bahan yang paling umum

digunakan untuk membuat *photo resistor* adalah *cadmium sulfide*. Dengan bahan ini, muatan elektron yang dilepas lebih banyak pada saat terkena cahaya sehingga menyebabkan resistansinya berkurang [22].



Gambar 2.7 Grafik panjang gelombang yang dapat ditangkap oleh photo resistor *[20]*

2.3.2 Sensor Suhu – LM35

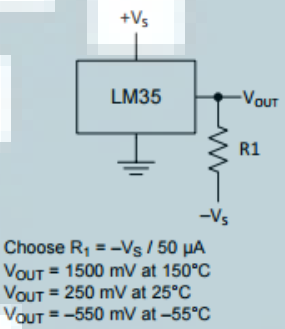
Sensor suhu mengukur besarnya energi panas yang dihasilkan suatu objek atau sistem, yang memungkinkan kita untuk mendeteksi perubahasan fisik dari objek tersebut dengan keluaran analog maupun digital [23].

Hal-hal yang harus diperhatikan dalam memilih sensor suhu adalah sebagai berikut:

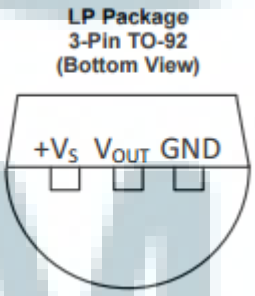
- Jenis sensor yang digunakan, *contact* atau *non-contact*
- Rentang suhu yang akan diukur atau dikendalikan
- Tingkatan perubahan suhu pada sistem

- Seberapa ketat kontrol yang akan dilakukan
- Biaya yang dibutuhkan untuk pemilihan sensor

LM35 adalah sebuah perangkat IC (*integrated-circuit*) pengukur suhu presisi, dengan tegangan *output* yang secara linier sebanding dengan suhu Celcius. Perangkat ini tidak memerlukan kalibrasi eksternal untuk suhu ruangan maupun rentang pengukuran penuh dari -55°C sampai dengan 150°C, dan hanya membutuhkan arus 60 µA dari sumber daya utama [24].



Gambar 2.8 Skema rangkaian LM35 untuk nilai ukur rentang penuh [24]



Gambar 2.9 Tampak bawah dari LM35 [24]

2.4 ZigBee

ZigBee merupakan suatu standar yang dikembangkan oleh ZigBee Alliance untuk kebutuhan *personal-area network* (PAN). Protokol ini dibangun berdasarkan pada standar IEEE 802.15.4, yang mendefinisikan

Media Access Control (MAC) dan *physical layer* untuk *low-cost wireless personal-area network* (LR-WPAN). Standar ZigBee menawarkan *stack profile* yang mendefinisikan *network*, *security*, dan *application layer*. Spesifikasinya sendiri merupakan standar terbuka yang memungkinkan pabrikan untuk mengembangkan sendiri perangkat-perangkat mereka yang membutuhkan biaya dan daya yang rendah [3].

Spesifikasi ZigBee sangat dapat diandalkan. ZigBee menggunakan kombinasi dua buah teknologi, yaitu *Offset-Quadrature Phase-Shift Keying* (O-QPSK) dan *Direct Sequence Spread Spectrum* (DSSS), sehingga dapat tetap bekerja dengan baik dalam lingkungan dengan *signal-to-noise ratio* (SNR) yang rendah. Untuk menambah keandalan, ZigBee menggunakan *Carrier Sense Multiple Access Collision Avoidance* (CSMA-CA). ZigBee hanya akan mentransmisikan pesan apabila kanal yang digunakan bersih, sehingga dapat mencegah kerusakan data [25].

Jaringan yang dibangun menggunakan standar ZigBee memiliki karakteristik sebagai berikut:

- Konsumsi daya baterai rendah. *End device* ZigBee dapat beroperasi tanpa mengganti baterainya dalam jangka waktu beberapa bulan hingga tahun;
- Biaya yang rendah;
- Kecepatan transmisi data yang rendah, dengan maksimum 250Kbps;
- Mudah diimplementasikan;

- *Nodes* yang dapat terhubung ke dalam jaringan mencapai 65.000;
- ZigBee dapat membentuk jaringannya sendiri secara otomatis;
- ZigBee menggunakan paket data berukuran kecil dibandingkan dengan WiFi dan Bluetooth.

Tabel 2.2 menunjukkan perbandingan antara karakteristik WiFi dengan ZigBee.

Tabel 2.2 Perbandingan karakteristik ZigBee, WiFi, dan Bluetooth [3]

	WiFi IEEE 802.11	**ZigBee IEEE 802.15.4**
Application	Wireless LAN	Control and monitor
Frequency bands	2.4GHz	2.4GHz, 868MHz, 915MHz
Battery life (days)	0.1 - 5	100 – 7,000
Nodes per network	30	65,000
Bandwidth	2-100Mbps	20-250Kbps
Range (meters)	1-100	1-75 and more
Topology	Tree	Star, tree, cluster tree, and mesh
Standby current	20 * 10-3 amps	3 * 10-6 amps
Memory	100KB	32-60KB

Secara umum ada tiga jenis perangkat utama untuk membentuk suatu jaringan ZigBee yang perlu diketahui, yaitu:

- ***Coordinator*** [3] – merupakan sebuah FFD (*full-function device*) dan bertanggung jawab atas seluruh manajemen jaringan. Setiap jaringan mempunyai hanya satu *coordinator* yang melaksanakan fungsi-fungsi sebagai berikut:
    - Memilih saluran yang digunakan oleh jaringan

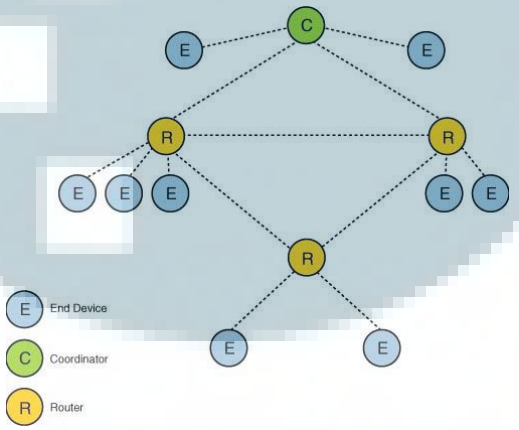
- Memulai jaringan
- Menetapkan alokasi pengalamatan pada *node* atau *router*
- Mengijinkan perangkat lain untuk bergabung atau meninggalkan jaringan
- Menyimpan daftar tetangganya dan *router*
- Mengirimkan paket-paket aplikasi

- ***End device*** [3] – dapat menjadi RFD (*reduced-function device*), yang hanya beroperasi dalam set terbatas pada IEEE 802.15.4 MAC *layer*, sehingga dapat mengkonsumsi lebih sedikit daya. *End device* (*child*) dapat dihubungkan ke *router* atau *coordinator* (*parent*). Pada dasarnya, *end device* hanya mengkonsumsi daya pada saat melakukan transmisi informasi. *End device* menjalankan fungsi-fungsi berikut:
  - Menggabungkan diri ke jaringan atau meninggalkannya
  - Mengirimkan paket-paket aplikasi
- ***Router*** [3] – merupakan sebuah FFD, yang digunakan dalam topologi *mesh* dan *tree* untuk memperluas cakupan jaringan. Fungsinya adalah untuk mencari rute terbaik ke tujuan pada saat melakukan tranfer pesan. *Router* melakukan fungsi yang mirip dengan *coordinator* kecuali dalam hal membentuk jaringan.

Jaringan ZigBee dapat dibentuk dengan menggunakan suatu topologi tertentu. Topologi terbaik yang dapat digunakan untuk melakukan transmisi adalah *mesh*. Topologi ini disebut juga dengan jaringan *peer-to-peer* yang

terdiri dari satu *coordinator*, beberapa *router* dan sejumlah *end device* [3]. Topologi *mesh* memiliki karakteristik sebagai berikut:

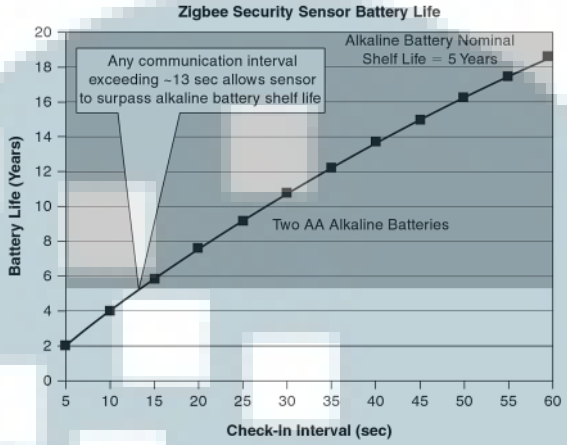
- Merupakan jaringan *multihop*; paket yang dikirimkan melalui sejumlah *hop* untuk mencapai tujuan;
- Ukuran jaringan dapat ditingkatkan dengan cara menambahkan lebih banyak perangkat;
- Dapat menghilangkan *dead zone*;
- *Self-healing*, yang berarti apabila terjadi kegagalan jalur pada saat transmisi, *node* akan mencari jalur alternatif ke tujuannya.



Gambar 2.10 Topologi mesh [3]

Saat sebuah perangkat bergabung ke dalam suatu jaringan ZigBee, *coordinator* atau *router* menetapkan suatu alamat logikal berukuran 16 bit untuk perangkat tersebut. Setiap perangkat juga memiliki alamat IEEE berukuran 64 bit, di mana tidak akan ada perangkat lain yang memiliki alamat yang sama. Alamat logikal 16 bit membantu mengurangi ukuran *frame*, yang berarti waktu yang dibutuhkan untuk mentransmisikan *frame*

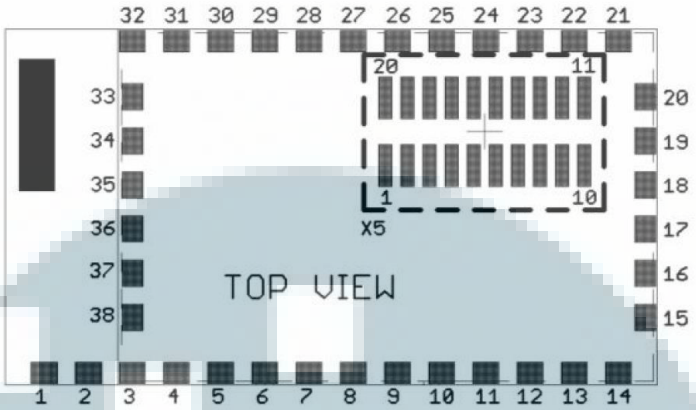
menjadi berkurang. Hal ini memungkinkan perangkat untuk menghemat konsumsi daya [3].



Gambar 2.11 Usia pakai baterai pada ZigBee [25]

Keamanan pada ZigBee menyediakan integritas, autentikasi, *freshness* dan kerahasiaan pesan bagi perangkat ZigBee. Untuk keamanan, ZigBee menggunakan *counter mode* (CTR) dengan AES (*Advanced Encryption Standard*) 128 bit untuk mengenkripsi pesan dan *cipher block chaining* (CBC) dengan AES 128 bit untuk menghasilkan *message integrity code* (MIC) [3].

Modul Telegesis ETRX2 merupakan ISM *band tranceiver* 2,4GHz dengan kebutuhan daya rendah, yang mengacu pada Ember EM250 *single chip* ZigBee. Modul ini dirancang agar dapat diintegrasikan dengan perangkat apapun tanpa harus memiliki pengetahuan dan keahlian dalam bidang RF (*Radio Frequency*) [26]. Gambar 2.8 merupakan konfigurasi pin pada modul telegesis ETRX2.



Gambar 2.12 Konfigurasi pin ETRX2 [26]

Tabel 2.3 merupakan informasi pin pada modul ETRX2.

Tabel 2.3 Informasi pin Telegesis ETRX2 [26]

ETRX2 Pad	Function	EM250 GPIO	ETRX2 Harwin Pin
1	GND	GND	
2	Antenna		
3	GND	GND	
4	I/O9	GPIO 0 (21)	1
5	Vreg {1}		2
6	GND	GND	3
7	Vcc		10
8	GND	GND	3
9	A/D1	GPIO 4 (26)	4
10	A/D2	GPIO 5 (27)	5
11	I/O7	GPIO 3 (25)	6
12	I/O6	GPIO 2 (24)	7
13	I/O5	GPIO 1 (22)	8
14	I/O4 or RTS {3}	GPIO 12 (20)	9
15	GND	GND	
16	SIF CLK	SIF CLK	
17	SIF MISO	SIF MISO	
18	SIF MOSI	SIF MOSI	
19	SIF LOADB	SIF LOADB	
20	GND	GND	
21	I/O8	GPIO 6 (29)	11
22	I/O2 or CTS {3}	GPIO 11 (19)	12
23	I/O3	GPIO 13 (43)	13
24	Reset	(13)	14
25	I/O1	GPIO 14 (42)	15
26	I/O0	GPIO 8 (31)	16
27	TXD {2}	GPIO 9 (32)	18
28	RXD {2}	GPIO 10 (33)	17
29	GND	GND	3
30	I/O10	GPIO 15 (41)	19
31	I/O11	GPIO 16 (40)	20
32	GND	GND	
33	RXTXSW	(11)	
34	GND	GND	
35	GND	GND	
36	GND	GND	
37	GND	GND	
38	N/C {4}	GPIO 7 (30)	

Untuk melakukan pengaturan pada modul ETRX2, digunakan suatu perintah yang dikenal dengan AT *style command*. Setiap perintah harus diawali dengan *prefix* “AT” atau “at” dan diakhiri dengan CR. Perintah-perintah tersebut diikuti *response* opsional termasuk <CR><LF><Response><CR><LF> dan/atau *prompt* <CR><LF><Prompt><CR><LF> di mana *prompt* juga dapat merupakan pesan *error* [27]. Perintah-perintah yang digunakan untuk membuat sistem kulkas pintar adalah sebagai berikut:

a. AT+EN: membentuk jaringan PAN baru (lalu menjadi *Coordinator*).

   Respon:

   1. JPAN:<channel>,<PID>,<EPID>

   OK

   2. ERROR:<error code>

b. AT+JN : bergabung ke jaringan yang ada.

   Respon:

   1. JPAN:<channel>,<PID>,<EPID>

   OK

   2. ERROR:<error code>

c. AT+N? : menampilkan informasi jaringan.

   Respon:

   1. +N=<devicetype>,<channel>,<power>,<PID>,<EPID>

   OK

   2.+N=NoPAN

OK

d. AT+IDREQ:<Address>[,XX]: mengetahui NodeID.

<Address>: dapat merupakan EUI64. Diisi FF jika ingin mengetahui NodeID sendiri.

XX: *optional*. Berisi nomor indeks

Respon

1. OK

2. ERROR:<error code>

Setelah respon di atas kecuali *error code* 00, terdapat *prompt*:

AddrResp:<errorcode>[,<NodeID>,<EUI64>] [nn.<NodeID>]

e. AT+EUIREQ:<Address>,<NodeID>[,XX]: mengetahui EUI64.

<Address>: dapat merupakan EUI64. Diisi FF jika ingin mengetahui NodeID sendiri.

XX: *optional*. Berisi nomor indeks

Respon :

1. SEQ:XX

OK

2. ERROR:<errorcode>

Setelah respon di atas kecuali *error code* 00, terdapat *prompt*: AddrResp:<errorcode>[,<NodeID>,<EUI64>] [nn.<NodeID>]

f. AT+ANNCE: melakukan *broadcast* ID dirinya ke semua *node* di jaringan.

Respon :

1. OK

2. ERROR:<errorcode>

Node lain akan mendapatkan *prompt* di bawah ini tergantung tipe device yang melakukan *broadcast*:

FFD:<EUI64>,<NodeID> [,syy,zz]

MED:<EUI64>,<NodeID> [,syy,zz]

SED:<EUI64>,<NodeID> [,syy,zz]

ZED:<EUI64>,<NodeID> [,syy,zz]

g. AT+BCAST:nn,<data>: broadcast pesan ke semua node di jaringan.

nn: jumlah maksimum *hop* untuk broadcast. Nilai dari 00 sampai 30. Jika bernilai 00, semua *node* akan mendapatkan pesan.

Respon :

1. OK

2. ERROR:<errorcode>

Node lain mendapatkan prompt setelah menerima *broadcast*:

BCAST:[<EUI64>,]<length>=<data>

EUI64 adalah EUI64 pengirim.

h. AT+UCAST:<address>=<data>: mengirimkan *unicast* ke salah satu node di jaringan.

address: alamat tujuan. Dapat berupa EUI64 atau NodeID.

Respon :

1. SEQ:XX

OK

2. ERROR:<errorcode>

Pengirim akan mendapatkan *prompt* di bawah ini untuk mengetahui apakah pesan berhasil dikirim atau tidak.

1. ACK:XX

2. NACK:XX

Node penerima akan mendapatkan *prompt*:

UCAST:[<EUI64>,]<length>=<data>

i. ATSXX[x[x]]=<data>[,<password>] : mengganti nilai di S-Register.

Respon :

1. OK

2. ERROR:<errorcode>

S-Register yang diubah nilainya:

1. S12: mengatur UART. Terdiri dari 4 digit heksadesimal.

2. S0E: menampilkan/menghilangkan *prompt* tertentu.

2.5 Android

Android merupakan suatu kumpulan perangkat lunak yang tidak hanya berisi sistem operasi tetapi juga *middleware* dan *key application*, yang mendukung sejumlah besar aplikasi di dalam *smartphone*. Aplikasi pada Android ditulis dalam bahasa pemrograman java. Android mengacu pada sistem operasi Linux versi 2.6 untuk layanan utama seperti *security*, *memory management*, *process management*, *network stack*, dan *driver model* [28].

Aplikasi pada Android ditulis dalam bahasa pemrograman java. *Code* tersebut di*compile* oleh Android SDK (*Software Development Kit*) *tools* – bersama dengan berbagai data dan *resource file* – menjadi sebuah APK (*Android package*), yang mana menggunakan *suffix* .apk. APK tersebut menyimpan semua *file* yang dibutuhkan saat akan melakukan instalasi aplikasi [29]. Aplikasi Android memiliki dasar sebagai berikut [28]:

1. Aplikasi berbasis Android terdiri dari satu lebih komponen-komponen aplikasi, seperti misalnya *activities*, *services*, *content providers*, dan *broadcast receivers*.
2. Setiap komponen melaksanakan peran yang berbeda dalam perilaku aplikasi secara keseluruhan, dan setiap komponen dapat diaktifkan secara individual.
3. *File* manifestasi harus mendeklarasikan semua komponen aplikasi dan juga dapat mendeklarasikan semua kebutuhan aplikasi, seperti misalnya versi minimum Android dan pengaturan perangkat yang dibutuhkan.
4. Sumber daya aplikasi *non-code* (gambar, *strings*, tampilan *file* dan sebagainya) harus memiliki alternatif untuk perangkat dengan konfigurasi berbeda.

2.6 XML-RPC

XML-RPC merupakan sebuah protokol *Remote Procedure Calling* yang bekerja di Internet [30]. Sebuah pesan XML-RPC merupakan HTTP-POST *request* dengan *request body* dalam bentuk XML [30]. Sebuah

procedure dieksekusi pada *server* dan nilainya dikembalikan dalam format XML. Parameter dari *procedure* dapat berupa skalar, angka, *string*, tanggal, dan sebagainya, dan juga dapat berupa struktur *record* dan *list* yang kompleks.

Berikut ini merupakan contoh dari XML-RPC *request*:

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181

<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
    <param>
      <value><i4>41</i4></value>
    </param>
  </params>
</methodCall>
```

Penjelasan *Header*

- Format URI pada baris pertama *header* tidak ditentukan. Sebagai contoh, isinya dapat dikosongkan atau garis miring, jika *server* hanya melayani XML-RPC *request*. Namun, jika *server* melayani berbagai HTTP *request* yang berbeda, URI digunakan untuk membantu mengarahkan *request* ke kode yang melayani XML-RPC *request*
- *User-Agent* dan *Host* harus dirincikan
- *Content-Type* merupakan *text/xml*
- *Content-Length* harus dirincikan dan benar

Format *payload*

Isi dari *payload* adalah sebuah struktur <*methodCall*> tunggal yang ditulis dalam format XML.

<*methodCall*> harus mengandung sebuah <*methodName*> *sub-item*, serta sebuah *string* yang berisi nama dari *method* yang dipanggil. *String* tersebut hanya boleh mengandung karakter-karakter seperti, huruf kecil maupun kapital, angka, garis bawah, tanda titik, titik dua, dan garis miring. Interpretasi karakter-karakter tersebut diserahkan sepenuhnya pada *server*.

Sebagai contoh, <*methodName*> dapat berupa nama dari sebuah *file* berisi *script* yang mengeksekusi *request* yang masuk. Dapat juga berupa nama dari suatu baris pada sebuah tabel di *database*. Atau dapat juga berupa *path* dari *file* yang terdapat pada suatu hirarki pada *folder* dan *file*.

Jika *procedure call* memiliki parameter, <*methodCall*> harus mengandung sebuah <*params*> *sub-item*. Parameter tersebut dapat berisi sejumlah <*param*>, yang masing-masing memiliki sebuah <*value*>.

Scalar <value>s

<value> dapat berupa skalar. Tipe datanya adalah sebagai berikut:

Tabel 2.4 Tipe data dan tag dalam [30]

Tag	**Tipe Data**	**Contoh**
<i4> or <int>	*four-byte signed integer*	-12
<boolean>	0 (*false*) or 1 (*true*)	1
<string>	*String*	hello world
<double>	*double-precision signed floating point number*	-12.214
<dateTime .iso8601>	*date/time*	19980717T14:08:55

<base64>	*base64-encoded binary*	eW91IGNhbid0IHJlYWQgdGhpcyE=

<struct>s

Sebuah *value* juga dapat berupa <*struct*>. Sebuah <*struct*> berisi <*member*> dan setiap <*member*> berisi sebuah <*name*> dan <*value*>. Berikut ini adalah contoh <*struct*> dengan dua elemen:

```
<struct>
  <member>
    <name>lowerBound</name>
    <value><i4>18</i4></value>
  </member>
  <member>
    <name>upperBound</name>
    <value><i4>139</i4></value>
  </member>
</struct>
```

<*struct*> dapat bersifat rekursif, <*value*> apapun dapat berisi sebuah <*struct*> atau tipe data lainnya, termasuk juga <*array*>.

<array>s

Tipe data sebuah *value* dapat berupa <*array*>. Sebuah <*array*> berisi sebuah elemen <*data*>, yang dapat berisi sejumlah <*value*>. Berikut ini adalah contoh *array* dengan empat elemen:

```
<array>
  <data>
    <value><i4>12</i4></value>
    <value><string>Egypt</string></value>
    <value><boolean>0</boolean></value>
    <value><i4>-31</i4></value>
  </data>
</array>
```

Elemen pada <*array*> tidak memiliki nama. Tipe data yang dikirimkan dapat dikombinasikan, seperti pada contoh di atas.

<array> dapat bersifat rekursif, *<value>* apapun dapat berisi sebuah *<array>* atau tipe data lainnya, termasuk *<struct>*.

Contoh *response*

Berikut adalah contoh *response* dari suatu XML-RPC *request*:

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:08 GMT
Server: UserLand Frontier/5.1.2-WinNT

<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><string>South Dakota</string></value>
    </param>
  </params>
</methodResponse>
```

Format *response*

- *Server* selalu mengembalikan *response code* dan *phrase* 200 OK, kecuali terjadi *low-level error*. *Content-Type* dari *response* adalah *text/xml*. *Content-Length* harus diisi dan bernilai benar.
- *Body* dari *response* berupa sebuah struktur XML, sebuah *<methodResponse>*, yang dapat berisi sebuah *<params>*, yang berisi sebuah *<param>*, yang berisi sebuah *<value>*.
- *<methodResponse>* juga dapat berisi *<fault>* yang berisi sebuah *<value>* berupa *<struct>* yang mengandung dua elemen, yaitu *<faultCode>* yang berupa *<int>* dan *<faultString>* yang berupa *<string>*.

- *<methodResponse>* tidak dapat berisi *<fault>* dan *<params>* pada saat yang bersamaan.

Contoh *fault*

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 426
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:02 GMT
Server: UserLand Frontier/5.1.2-WinNT


<?xml version="1.0"?>
<methodResponse>
  <fault>
    <value>
      <struct>
        <member>
          <name>faultCode</name>
          <value><int>4</int></value>
        </member>
        <member>
          <name>faultString</name>
          <value><string>Too many parameters.</string></value>
        </member>
      </struct>
    </value>
  </fault>
</methodResponse>
```

Langkah-langkah komunikasi antara *client* dan *server* menggunakan XML-RPC adalah sebagai berikut [31]:

1. Client membuat XML *element* *<methodCall>* yang berisi *method* yang akan dipanggil dan parameter untuk menjalankan *method*.
2. Client mengirimkan sebuah POST *request* yang berisi XML *element* yang telah dibuat.
3. *Server* menerima *request* dan menggunakan HTTP *header Content-Length* untuk mengetahui panjang XML *element*.

4. *Server* mem-parse XML *element*, mengambil nama *method*, dan parameter untuk menjalankan *method* dari XML *element*.
5. *Server* mencari *method* yang diinginkan, dan jika ditemukan, menjalankannya dengan parameter yang diterima.
6. Jika *method* berhasil dieksekusi, *server* membuat *return* <*value*> dalam XML *element* <*methodResponse*>, dengan sebuah *params element*, dan mengirimkannya ke *client*.
7. Jika *method* tidak ditemukan atau tidak dapat dijalankan karena alasan tertentu, *server* membuat XML *element* <*methodResponse*> dengan <*value*> dari *element* adalah *fault element*, kemudian mengirimkannya ke *client*.
8. *Client* menerima *response*, mem-parse XML *element* yang dikembalikan *server*, yang dapat berupa <*value*> atau <*fault*> kemudian melaporkannya ke *client user*.