BAB II
TINJAUAN PUSTAKA

2.1 Sistem *Pick-by-Light*

Sistem *pick-by-light* adalah sistem pengambilan barang menggunakan lampu sebagai penanda lokasi penyimpanan. Sistem ini diterapkan pada industri khususnya pergudangan dengan menggunakan sistem terintegrasi antara perangkat lunak sebagai pengolah informasi dan perangkat keras yang terdiri dari lampu sebagai indikator dan tombol sebagai konfirmasi pengambilan barang. Sistem ini dapat menggantikan sistem konvensional dimana operator akan mengambil barang dengan mencari ke tempat penyimpanan tanpa ada indikator pada lokasi barang. Sistem *pick-by-light* membantu operator dalam mengoptimalkan pengambilan barang dan meminimalisir kesalahan. Setelah operator mengambil barang, kemudian melakukan konfirmasi dengan menekan tombol[3]. Contoh sistem *pick-by-light* dapat dilihat pada gambar berikut.

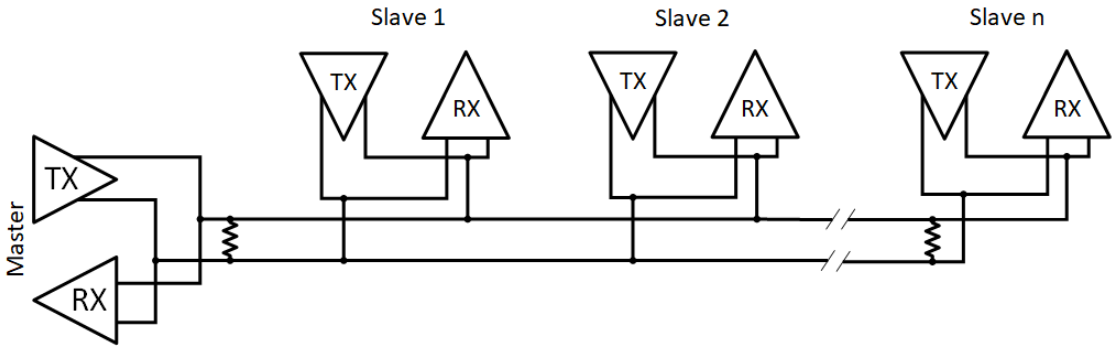
Gambar 2. 1. Contoh Sistem Pick-by-light[5]

Sumber: https://lightningpick.com/products/put-to-light/

2.2 RS-485

TIA/EIA-485-A atau dikenal juga sebagai RS-485 adalah standar komunikasi serial asinkron yang ditetapkan oleh Electronics Industries Association (EIA) pada tahun 1983 dengan nama lengkap *EIA/TIA-485 Standard for Electrical Characteristics of Generators and Receivers for use in a Balanced Digital Multipoint System*[6]. Komunikasi RS-485 dapat mengirimkan data sejauh 1,2 km dan dapat menghubungkan

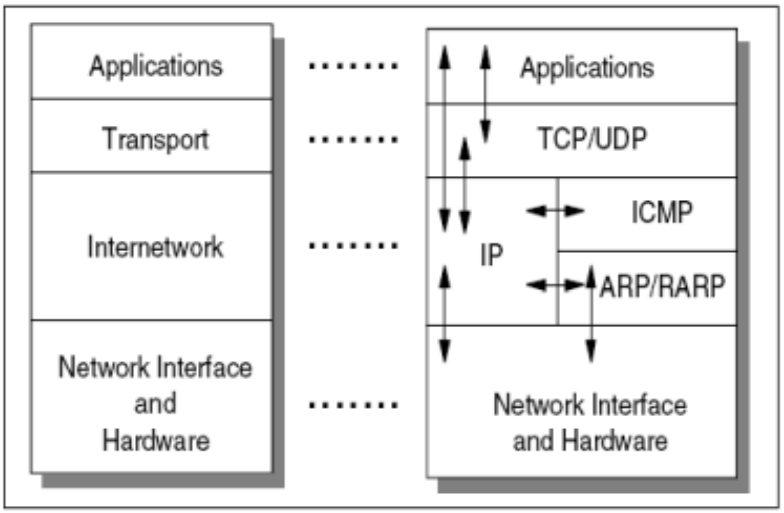
32 perangkat menggunakan dua kabel tanpa referensi *ground* yang sama[7]. RS-485 merupakan pengembangan dari RS-232 dimana hanya dapat menghubungkan satu perangkat secara *point-to-point* dengan jarak maksimum 15 meter. Komunikasi RS-485 dilakukan secara *multipoint* yaitu dapat menghubungkan beberapa perangkat menggunakan satu jalur komunikasi[8]. RS-232 dan RS-485 merupakan standar komunikasi yang digunakan di industri. RS-485 dapat menggunakan sistem komunikasi *half-duplex* yaitu dapat melakukan pengiriman data dua arah tetapi tidak dalam waktu yang sama atau *full-duplex* dengan pengiriman data dua arah. Komunikasi RS-485 menggunakan transmisi diferensial *balanced transmission* yaitu mengubah tegangan TTL menjadi selisih tegangan antara *output* A dan B sehingga meminimalkan efek dari *noise*[9]. Selisih tegangan antara *output* A dan B akan tetap karena interferensi *noise* akan terjadi sekaligus pada jalur *output* A dan jalur *complementary output* B[10]. Rangkaian RS-485 dalam topologi *daisy-chain* dengan *multiple point* dapat dilihat pada gambar 2.2.



Gambar 2. 2. Rangkaian Topologi Daisy-chain RS-485

2.3 TCP/IP

TCP/IP (*Transmission Control Protocol/Internet Protocol*) adalah standar komunikasi data yang digunakan untuk menghubungkan komputer dalam jaringan internet maupun *Local Area Network* (LAN). TCP/IP merupakan sebuah standar jaringan terbuka yang bersifat independen terhadap mekanisme transport jaringan fisik. Protokol ini menggunakan skema pengalamatan yang disebut dengan alamat IP (*Internet Protocol*). TCP/IP dibentuk dalam beberapa lapisan (*layer*) yang bertujuan untuk mempermudah pengembangan dan pengimplementasian. Antar *layer* dapat berkomunikasi dengan suatu penghubung *interface.* Lapisan pada TCP/IP dapat dilihat pada gambar berikut.

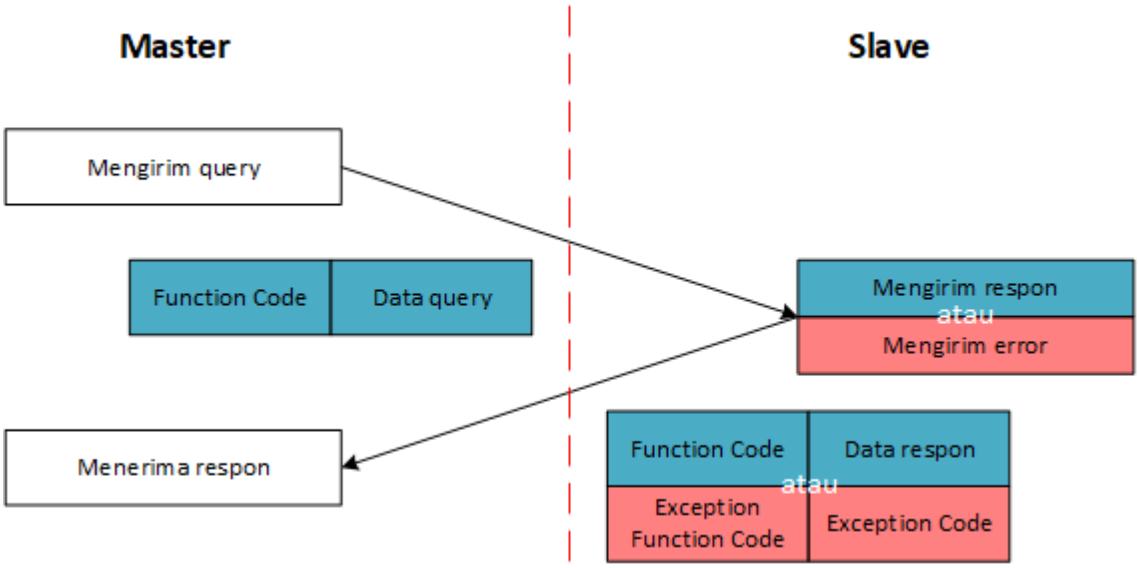


Gambar 2. 3. Lapisan Pada TCP/IP[11]

Sumber: Jaringan Komputer I, 2014

2.4 Modbus

Modbus adalah protokol komunikasi jaringan berstandar internasional yang dipublikasikan oleh Modicon pada tahun 1979 yang digunakan pada *Programmable Logic Controllers* (PLC)[12]. Protokol Modbus berjalan pada *Application Layer* atau lapisan ketujuh model referensi *Open System Interconnection* (OSI). Modbus dikembangkan secara *open source* oleh organisasi Modbus.org. Modbus merupakan protokol yang paling banyak digunakan pada industri otomasi karena kemudahannya. Salah satu metode yang digunakan yaitu menghubungkan setiap perangkat melalui komunikasi serial. Perangkat yang menerima informasi disebut Modbus *master* dan perangkat yang mengirimkan informasi disebut Modbus *slave*. *Master* atau *slave* dapat berupa PLC, *Human Machine Interface* (HMI), mikrokontroler dan lain-lain. *Master* bersifat aktif yaitu memulai komunikasi antara lain membaca data, menulis data, mengetahui status *slave* dan mengirimkan permintaan atau *query* yang terdiri dari *function code* dan data. Sedangkan *slave* bersifat pasif yang hanya merespon jika ada permintaan dari *master*. Jika data *query* dari *master* diterima dengan benar oleh *slave,* maka *slave* akan mengirim pesan respon ke *master* yang terdiri dari *function code* dan *data response.* Sedangkan jika terdapat kesalahan pada *query* yang diterima oleh *slave,* maka *slave* akan mengirim pesan *error* ke *master* yang terdiri dari *exception function code* dan *exception code.* Proses transaksi data dari *master* ke *slave* dapat dilihat pada gambar berikut.



Gambar 2. 4. Proses Transaksi Data Modbus

Protocol Data Unit pada Modbus terdiri dari *function code* dan *data query*. *Function code* yaitu kode perintah yang harus dilakukan oleh *slave*. *Function code* pada Modbus dapat dilihat pada tabel berikut.

Tabel 2. 1. Function Code Modbus

Jenis Data	**Fungsi**			**No awal Register**
	Membaca	**Menulis 1 data**	**Menulis banyak data**	
Coil	01 (01 hex)	05 (05 hex)	15 (0F hex)	00001
Discrete Input	02 (02 hex)			10001
Input Register	04 (04 hex)			30001
Holding Register	03 (03 hex)	06 (06 hex)	16 (10 hex)	40001

Response exception adalah respon dari *slave* saat terjadi kesalahan atau *error* yang akan dikirim oleh *slave* ketika *slave* tidak dapat menangani perintah dari *master*. *Response exception* terdiri dari *exception function code* dan *exception code*. *Exception function code* adalah *function code* dari *master* dengan mengubah *most significant bit* dalam bilangan biner menjadi satu atau *function code* dalam bilangan heksa desimal ditambah dengan 80. Berikut ini adalah tabel *exception function code.*

Tabel 2. 2. Function Exception Code

Function Code in Request	***Function Code in Exception Response***
01 (01 hex) 0000 0001	129 (81 hex) 1000 0001
02 (02 hex) 0000 0010	130 (82 hex) 1000 0010
03 (03 hex) 0000 0011	131 (83 hex) 1000 0011
04 (04 hex) 0000 0100	132 (84 hex) 1000 0100
05 (05 hex) 0000 0101	133 (85 hex) 1000 0101
06 (06 hex) 0000 0110	134 (86 hex) 1000 0110
15 (0F hex) 0000 1111	143 (8F hex) 1000 1111
16 (10 hex) 0001 0000	144 (90 hex) 1001 0000

Exception code adalah kode yang mengindikasikan kesalahan yang terjadi. *Exception code* dapat dilihat pada tabel berikut.

Tabel 2. 3. Exception Code

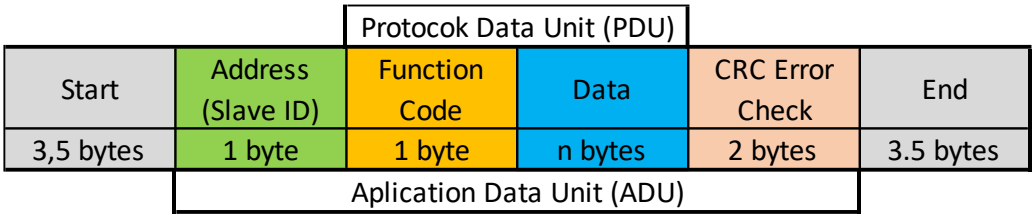
Exception Code	**Nama**	**Arti**
01 (01 hex)	*Illegal Function*	*Function code* salah. Kode ini menunjukkan bahwa *slave* dalam kondisi yang salah untuk memproses permintaan jenis ini, misalnya karena tidak dikonfigurasi dan diminta untuk mengembalikan nilai register
02 (02 hex)	*Illegal Data Address*	Alamat register salah
03 (03 hex)	*Illegal Data Value*	Mengandung nilai data yang tidak diijinkan oleh *slave*
04 (04 hex)	*Slave Device Failure*	*Slave* gagal melaksanakan perintah
05 (05 hex)	*Acknowledgement*	Pemberitahuan ke *master* bahwa pelaksanaan perintah akan memakan waktu yang lama dan dapat menyebabkan *time out*
06 (06 hex)	*Slave Device Busy*	*Slave* sedang memproses perintah *long-duration program. Master* akan mengirim pesan setelah *slave* bebas.

Exception Code	Nama	Arti
07 (07 hex)	*Negative Acknowledgement*	*Slave* tidak dapat menjalankan fungsi program yang diterima dalam *query*. Kode ini dikembalikan untuk permintaan pemrogram yang gagal menggunakan *function code* 13 atau 14 desimal. *Master* harus meminta informasi diagnostik atau kesalahan dari *slave.*
08 (08 hex)	*Memory Parity Error*	*Slave* berusaha membaca *extended memory* atau merekam file tetapi terdeteksi kesalahan *parity memory.*
10 (0A hex)	*Gateway Path Unavailable*	Menunjukkan bahwa *gatewa*y tidak dapat mengalokasikan jalur komunikasi internal dari *port input* ke *port output* untuk memproses permintaan
11 (0B hex)	*Gateway Target Device Failed to Respond*	Menunjukkan bahwa tidak ada respon yang diperoleh dari perangkat target. Biasanya berarti perangkat tidak ada di jaringan.

Sebuah *slave* memiliki alamat yang unik, alamat tersebut digunakan untuk mengirim perintah dari *master* sesuai dengan alamat yang dituju. Hanya alamat yang dituju yang dapat menerima perintah, walaupun perintah tersebut dikirim ke *slave* yang lain. Jaringan Modbus dapat menghubungkan sampai dengan 247 *slave* dengan alamat *slave* yang berbeda pada 1 *master*. Setiap perintah yang dikirimkan mempunyai mekanisme pemeriksaan data untuk memastikan data yang dikirim tidak terjadi kerusakan. Perintah pada Modbus dapat berisi perintah untuk mengubah nilai register, membaca nilai register, maupun membaca *port* I/O. Modbus *master* dapat mengirimkan perintah dalam dua mode, yaitu *unicast* dan *broadcast*. Pada mode *unicast*, perangkat *master* mengirim perintah ke satu *slave*, sedangkan mode *broadcast* perangkat *master* mengirim perintah ke semua *slave*. Modbus memiliki beberapa tipe, diantaranya yaitu:

2.4.1. Modbus RTU

Modbus RTU (*Remote Terminal Unit*) merupakan varian Modbus yang umum digunakan pada komunikasi serial dengan bentuk yang ringkas. Format RTU menggunakan mekanisme *Cyclic Redundancy Check* (CRC) untuk memastikan data yang dikirim lengkap dan benar. Dalam setiap 8-byte data yang dikirim berisi karakter 4-bit heksa desimal. Keuntungan menggunakan Modbus RTU adalah dapat mengirimkan data lebih banyak dengan *baud rate* yang sama. Setiap data yang dikirim bersifat kontinyu dan memiliki waktu tunda pada awal dan akhir pesan antara 1,5 sampai dengan 3,5 karakter dari *baud rate* yang dipakai[13]. Jika waktu tunda kurang dari 1,5 atau lebih dari 3,5 karakter, maka data tersebut termasuk error. Format *frame* data pada protokol Modbus berbeda untuk tiap jenisnya. *Frame* pada Modbus RTU dapat dilihat pada gambar berikut.



Gambar 2. 5. Format Frame Modbus RTU

Sumber: *Modicon Modbus Protocol Reference Guide*, 1996

Fungsi dari setiap blok *frame* Modbus RTU adalah sebagai berikut.

1. *Start*: berfungsi sebagai penanda awal *frame* baru.
2. *Address:* alamat dari *slave*, yaitu dari 1 sampai dengan 247. Alamat 0 digunakan untuk *broadcast.*
3. *Function Code:* merupakan perintah yang harus dikerjakan oleh *slave.*
4. *Data:* berisi alamat register, jumlah data, dan data yang akan ditulis. Jumlah *byte* data bergantung pada jumlah data yang akan dituliskan di *slave.*
5. *CRC Error check:* untuk mendeteksi kerusakan data yang dikirim.
6. *End:* sebagai penanda akhir dari *frame* data.

Data yang dikirim disimpan oleh *slave* pada register yang mempunyai alamat berbeda berdasarkan tipe data dan fungsi yang dikirimkan. Data pada Modbus terdiri dari dua tipe, yaitu *discrete* dan analog. Data *discrete* memiliki nilai data satu bit yaitu 0 dan 1, sedangkan data analog memiliki nilai data 16-bit. Data yang ditampilkan dapat berupa bilangan biner maupun bilangan heksadesimal. Setiap data juga memiliki jenis akses yang berbeda, yaitu *read-write* yang dapat diubah oleh *master* dan *slave* serta

read-only yang hanya dapat diubah oleh *slave.* Register penyimpanan pada Modbus dapat dilihat pada tabel 2.4.

Tabel 2. 4. Register Penyimpanan pada Modbus

Lokasi	Nama Tabel	Nilai Data	Tipe	Keterangan
00001 – 9999	*Discrete Output Coils*	1-bit	*Read-write*	*Master* dan *slave* dapat mengubah data *coil*
10001 - 19999	*Discrete Inputs Contacts*	1-bit	*Read-only*	Data hanya dapat diubah oleh *slave*
30001 – 39999	*Analog Input Registers*	16-bit words	*Read-only*	Data hanya dapat diubah oleh *slave*
40001 – 49999	*Analog Output Holding Registers*	16-bit words	*Read-write*	*Master* dan *slave* dapat mengubah data register

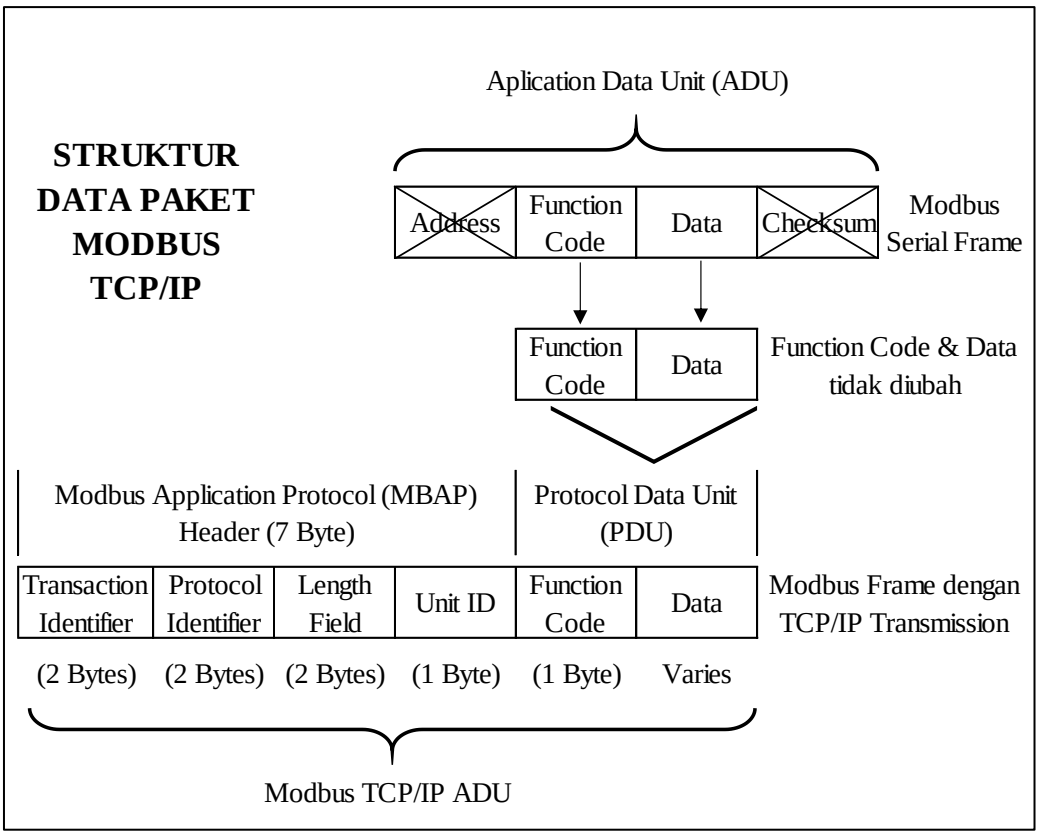
Fungsi dari setiap register adalah sebagai berikut:

1. *Discrete output coils*: pada mulanya data ini digunakan untuk mengaktifkan *coil* relay, register ini berfungsi untuk mengirimkan data digital 0 atau 1.
2. *Discrete input contact*: disebut juga dengan input relay, register in berfungsi untuk membaca data digital.
3. *Analog input registers*: digunakan untuk menyimpan data analog dari 0 sampai dengan 65535.
4. *Analog output holding registers*: digunakan untuk menyimpan nilai analog dengan *range* 0 sampai dengan 65535.

2.4.2. Modbus TCP

Modbus TCP (dikenal juga sebagai Modbus-TCP/IP) adalah protokol Modbus RTU menggunakan antarmuka TCP yang berjalan pada media Ethernet. Modbus TCP adalah protokol yang berjalan pada *layer* aplikasi TCP/IP. Modbus TCP mengkombinasikan jaringan fisik (Ethernet) dengan *networking standard* (TCP/IP) dan metode standar untuk merepresentasikan data (Modbus sebagai protokol aplikasi)[14]. Modbus TCP adalah protokol Modbus yang dienkapsulasi ke dalam Ethernet TCP/IP. Modbus TCP menanamkan *frame* data Modbus standar ke dalam *frame* TCP tanpa

checksum[15]. Selanjutnya, *frame* alamat Modbus digantikan dengan identifikasi unit di Modbus TCP dan menjadi bagian dari *Modbus Application Protocol* (MBAP) *header*. Paket data Modbus kemudian ditambahkan alamat IP dan dikirim dari *client* ke *server* sesuai alamat IP yang diberikan. Struktur *frame* dari Modbus TCP dapat dilihat pada gambar berikut.



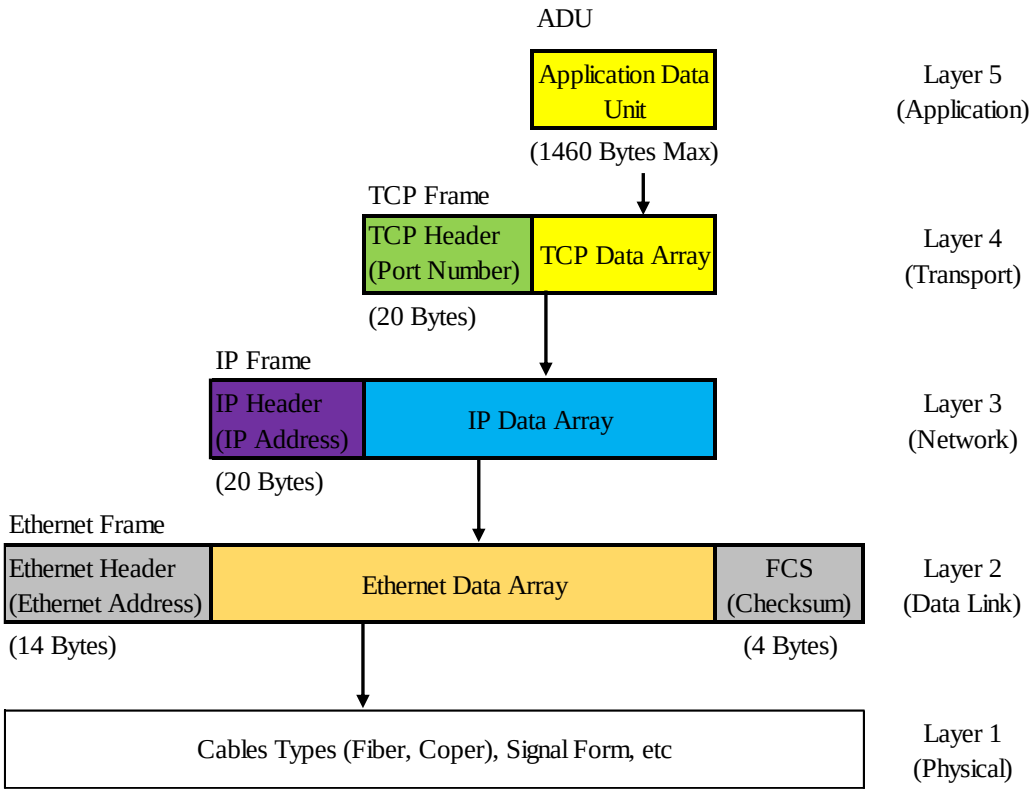
Gambar 2. 6. Struktur Frame Modbus TCP

Sumber: *Introduction to Modbus TCP/IP*, 2005

Berdasarkan Gambar 2.6, MBAP *header* berjumlah 7-byte terdiri dari:

1. *Transaction/Invocation Identifier* (2-bytes): digunakan untuk *pairing* transaksi ketika beberapa pesan dikirim bersamaan dengan koneksi TCP yang sama oleh *client* tanpa menunggu respon sebelumnya.
2. *Protocol Identifier* (2-bytes): *field* ini selalu 0 untuk Modbus dan nilai lainnya dicadangkan untuk ekstensi di masa mendatang.
3. *Length Field* (2-bytes): *field* ini adalah jumlah byte dari *field* yang tersisa dan termasuk *Unit Identifier, Function Code,* dan *field* Data.
4. *Unit Identifier* (1-byte): *field* ini digunakan untuk mengidentifikasi *remote server* yang berada di jaringan non-TCP/IP (sebagai penghubung serial). Pada sebuah aplikasi server Modbus TCP tertentu, unit ID diset ke 00 atau FF.

Saat pengiriman data pada Modbus TCP terjadi perubahan struktur data pada setiap *layer* yang dilewati. Proses perubahan struktur data disebut dengan enkapsulasi. Pada *application layer,* protokol Modbus membangun format *frame* yang disebut dengan *Application Data Unit* (ADU) yang terdiri dari MBAP dan *Protocol Data Unit* Modbus. Pada *transport layer,* ADU kemudian ditambah TCP *Header* untuk dapat dikirimkan pada jaringan. *Frame* pada *layer* ini disebut dengan TCP *frame.* Pada *network layer,* TCP *frame* ditambahkan IP *Header* agar data dapat dikirimkan sesuai dengan alamat IP yang diberikan. *Frame* pada *layer* ini disebut dengan IP *frame.* Pada *data link layer,* IP *frame* ditambahkan Ethernet *header* dan *Frame Check Sequence* (FCS). *Frame* pada *layer* ini disebut dengan Ethernet *frame.* Pada *physical layer,* Ethernet *frame* akan dikirim melalui jaringan fisik yang tersedia, seperti kabel, *fiber* optic, dan lain-lain. Proses enkapsulasi Modbus TCP dapat dilihat pada gambar berikut.

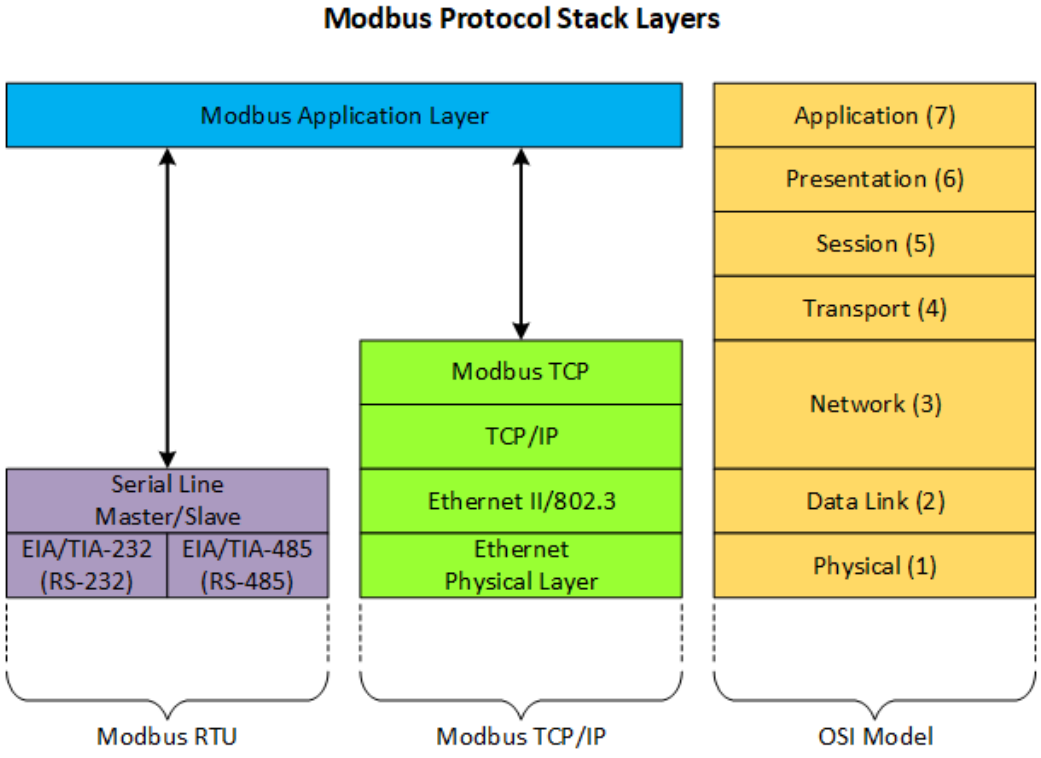


Gambar 2. 7. Proses Enkapsulasi Data Modbus TCP

Sumber: *Introduction to Modbus TCP/IP*, 2005

Modbus TCP menggunakan topologi jaringan *client* dan *server*. TCP adalah protokol berbasis koneksi (*connection-based protocol*) dimana *client* harus selalu terhubung dengan *server* saat pertukaran data. *Client* dan *server* Modbus TCP mengirim dan menerima data Modbus melalui port 502[14]. IEEE 802.3 Ethernet adalah protokol jaringan yang telah diterapkan secara universal di seluruh dunia dan merupakan standar

terbuka yang didukung oleh banyak produsen serta infrastrukturnya tersedia secara luas. Oleh karena itu, protokol TCP/IP digunakan di seluruh dunia dan bahkan berfungsi sebagai dasar untuk akses ke *World Wide Web*. Banyak perangkat yang sudah mendukung Ethernet, tidak heran jika protokol ini digunakan dalam aplikasi industri. Modbus TCP berbagi lapisan fisik dan data link yang sama dengan IEEE 802.3 Ethernet dan menggunakan TCP/IP suite protokol yang sama, sehingga Modbus TCP sepenuhnya kompatibel dengan infrastruktur Ethernet yang sudah ada mulai dari pengkabelan, konektor, *network interface cards*, hub, dan switch. Perbedaan antara Modbus RTU dan Modbus TCP dapat dilihat dari *stack layers protocol* berdasarkan model referensi OSI dimana Modbus RTU yang berjalan pada *Application Layer* OSI langsung terhubung ke *Serial Line Master/Slave* pada *Data Link* OSI model. Sedangkan Modbus TCP yang berjalan pada *Application Layer* OSI langsung terhubung ke *Network Layer* OSI model. Berikut ini adalah gambar *stack layer protocol* Modbus.



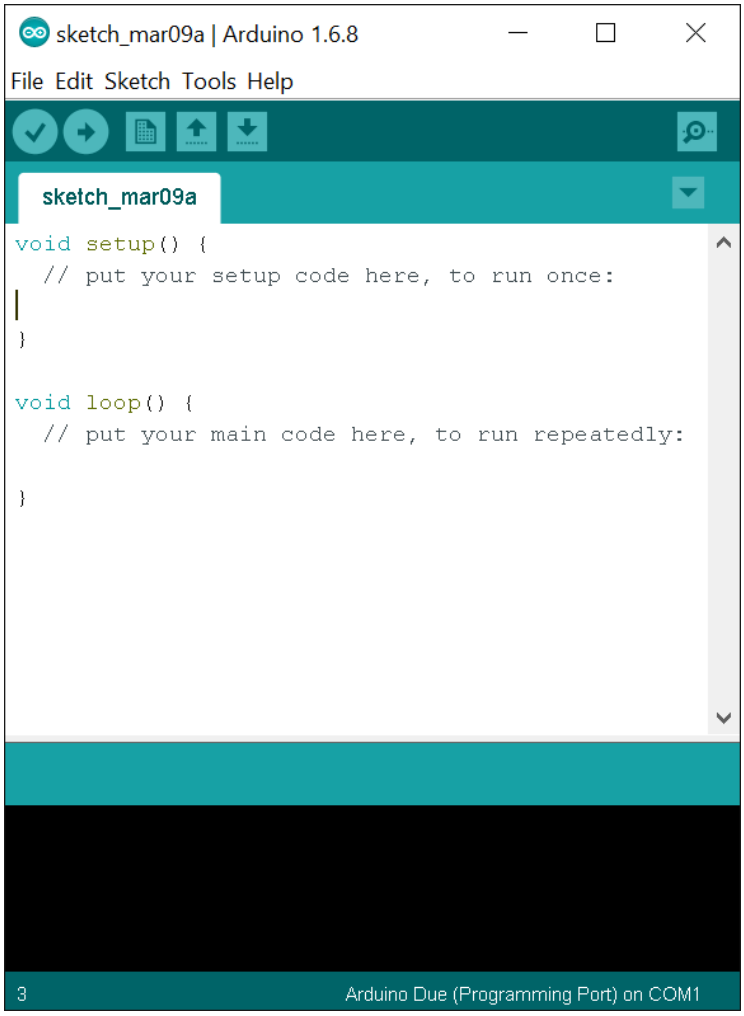
Gambar 2. 8. Modbus Protocol Stack Layers

Sumber: *Introduction to Modbus TCP/IP*, 2005

2.5 Arduino

Arduino adalah sebuah *platform* elektronik yang bersifat *open source* serta mudah digunakan[16]. Ekosistem Arduino terdiri dari *software* dan *hardware* yang merupakan ekosistem pengembangan terkemuka di dunia. Arduino diturunkan dari *Wiring*

Platform yaitu *platform* elektonik *open source* yang terdiri dari bahasa pemrograman, *software Integrated Development Environtment* (IDE) dan perangkat mikrokontroler. Bahasa pemrograman arduino mempunyai kemiripan dengan bahasa C. *Software* IDE Arduino adalah lingkungan pengembangan untuk memrogram mikrokontroler Arduino yang bersifat sumber terbuka artinya setiap orang dapat bebas membuat atau mengembangkannya. IDE Arduino dapat dilihat pada gambar berikut.



Gambar 2. 9. Arduino IDE

Sumber: https://www.digikey.ca/en/maker/blogs/2018/introduction-to-the-arduino-ide

Mikrokontroler Arduino merupakan papan pengendali *single-board* yang artinya perangkat khusus berupa modul elektronik yang bentuk dan komponennya sudah jadi dan siap digunakan. Terdapat berbagai macam mikrokontroler Arduino, diantaranya yaitu Arduino Uno, Arduino Nano, Arduino Mega dan sebagainya.