PARVATHA REDDY BABUL REDDY
VISVODAYA INSTITUTE OF TECHNOLOGY AND SCIENCE's

VITSPACE

For more Materials
☺ Click the link given below ☺

vitspace.netlify.com

Scroll to read your material

UNIT 3 COMMUNICATION IN THE IOT

1. IP ADDRESS

An IP address is an identifier similar to a physical address. It allows a sender and recipient of information to know where to send and exchange data. Every device that connects to the internet has its own IP address. Without it devices could not identify each other and send the required information.

An IP address identifies every device connected to the internet. This enables computers and other internet-connected devices, such as mobile phones and Internet-of-Things (IoT) devices, to communicate over the internet and on local-area networks (LANs).

What is MAC Address?

- MAC address is the physical address, which uniquely identifies each device on a given network. To make communication between two networked devices, we need two addresses: **IP address and MAC address.** It is assigned to the NIC (Network Interface card) of each device that can be connected to the internet.
- It stands for **Media Access Control**, and also known as **Physical address, hardware address, or BIA (Burned In Address).**
- It is globally unique; it means two devices cannot have the same MAC address. It is represented in a hexadecimal format on each device, such as **00:0a:95:9d:67:16.**
- It is 12-digit, and 48 bits long, out of which the first *24 bits are used for* ***OUI****(Organization Unique Identifier),* and *24 bits are for NIC/vendor-specific.*
- It works on the data link layer of the OSI model.
- It is provided by the device's vendor at the time of manufacturing and embedded in its NIC, which is ideally cannot be changed.
- The **ARP protocol** is used to associate a logical address with a physical or MAC address.

Reason to have both IP and MAC addresses.

As we already had the IP address to communicate a computer to the internet, why we need the MAC address. The answer to this question is that every mac address is assigned to the NIC of a hardware device that helps to identify a device over a network.

When we request a page to load on the internet, the request is responded and sent to our IP address.

Both MAC and IP addresses are operated on different layers of the internet protocol suite. The MAC address works on layer 2 and helps identify the devices within the same broadcast network (such as the router). On the other hand, the IP addresses are used on layer 3 and help identify the devices on different networks.

We have the IP address to identify the device through different networks, we still need a MAC address to find the devices on the same network.

TCP and UDP Ports

TCP and UDP Ports are used to connect two devices over the Internet or other networks. However, to give data packages an entrance to the PC or server at the other end of the connection, the "doors" have to be open. These openings into the system are called ports.

What Are TCP Ports?

TCP stands for Transmission Control Protocol, and it is a connection-oriented protocol. In networking, protocols are rules or standards that govern how data is transmitted between devices. TCP is called a connection-oriented protocol because it establishes a connection between the receiving and sending devices before transmitting any data.

TCP ports are ports that comply with the transmission control protocols. Some TCP ports include File Transfer Protocol ports (20 and 21) for file transfers, the SMTP port (25) and IMAP port (143) for emails, and the Secure Shell port (22).

How Do TCP Ports Work?

TCP ports create connections before data is shared. For instance, if you want to tell your friend about a new movie or game, you can make a phone call. You dial your friend's number, and if she gets the call and confirms that you are on the other end of the line, she picks up. Then you can begin to tell her about the game.

In TCP, the device sending the data connects to the device supposed to receive it. The way TCP ports establish reliable connections is called a three-way handshake.

As the name implies, a three-way handshake requires three different interactions, which come in the form of three messages: SYN→SYN-ACK→ACK.

The first is the SYN segment. The sending device sends out a SYN (synchronized sequence number) message to try to communicate with the receiving computer. It is trying to say, "Hello! Are you available to make a connection? "

If the receiving device is available to make a connection, it responds to the device sending the connection request with a SYN-ACK segment. The SYN-ACK segment acknowledges the connection request and sends out a synchronized sequence number in return. In plain terms, the device is saying, "Yes, I acknowledge your request, and I am willing to make a connection."

When this happens, the sending device sends an ACK segment to the receiving device, telling it that it has acknowledged its message. Then a connection is formed, and it begins to transfer data. When the data transmission has been confirmed and completed, the connection is ended.

What Are UDP Ports?

UDP stands for User Datagram Protocol. The User Datagram Protocol is connectionless, which means a host device can transmit data to its recipient without establishing a connection beforehand. UDP ports depend on the UDP/IP protocols. UDP ports include the DNS port (53), the Dynamic Host Configuration Protocol port (68), and the Kerberos port (88), which is used by gaming services.

How Do UDP Ports Work?

Unlike the TCP ports, UDP ports do not need to establish connections before transferring data. So, if you wanted to tell your friend about a new movie imitating the up port, you would have to shout out your conversation and hope that your friend is in the vicinity and can hear you. Pretty unreliable, right?

The responsibility of receiving the information you are trying to pass lies solely on your friend. Because you have not made a connection yet, your friend might not hear you properly and only hear bits and pieces or nothing at all.

In UDP ports, the host sends out data in packets (small segments) with no certain destination in mind. Then it hopes that the receiving device gets those packets, which is unreliable as it does not guarantee the data will be received seamlessly. As a result, packets do not get to the receiving end, and data is lost. This is known as packet loss.

The responsibility of receiving the information you are trying to pass lies solely on your friend. Because you have not made a connection yet, your friend might not hear you properly and only hear bits and pieces or nothing at all.

In UDP ports, the host sends out data in packets (small segments) with no certain destination in mind. Then it hopes that the receiving device gets those packets, which is unreliable as it does not guarantee the data will be received seamlessly. As a result, packets do not get to the receiving end, and data is lost. This is known as packet loss.

Because of its reliability, TCP ports are used for services where you need secure and complete data transmission like emails, pictures, websites, etc.

Time

Because UDP ports are connectionless protocols, they save a lot of time by not establishing connections before sending out data packets, which comes in handy for time-sensitive services and where data is received in real-time. UDP ports are used in video, voice, and game streaming.

TCP and UDP Ports, Explained

TCP and UDP ports are networking terms you do not hear regularly, but they are the bedrock of our internet. These two ports play a huge role in your day-to-day life, as without these ports, data transmission would be nigh impossible.

Application layer Protocols

The application layer of a network enables network entities to identify and interact with each other.

MQTT (Message Queueing Telemetry Transport)

MQTT is a lightweight communication protocol specifically designed for IoT and M2M applications. It's ideal for remote environments or applications with limited bandwidth. MQTT uses a connection oriented publish/subscribe architecture, where MQTT applications can either publish (transmit) or subscribe to (receive) topics, and an MQTT broker passes information from the publishing client to the subscribed client.

So for instance, an oil rig may have a predictive maintenance sensor that detects changes in vibrations and uses the MQTT protocol. The sensor "publishes" the vibration level to the broker, which the MQTT broker then passes to a software application that has subscribed to the topic "vibrations," which can then trigger an alert when the level is above a threshold.

HTTP (HyperText Transfer Protocol)

Hypertext Transfer Protocol is the most widely used protocol for navigating the Internet and to make data available over REST-APIs. The main advantage of using HTTP for IoT is that web application developers can use the same mechanism to send data to a Webserver - via an HTTP POST request.

The drawbacks are that HTTP uses a connectionless request-respond communication, meaning every message needs to include authentication information—which requires data and energy

consumption. Nevertheless HTTP might be ideal for use cases which have fewer data and battery constraints and where devices already need to call existing REST-APIs.

Like MQTT and MQTT over WebSocket, HTTP is one of the protocols that is supported by standard IoT cloud services such as AWS IoT and Azure IoT.

WebSocket

WebSocket is a bi-directional communication protocol designed to quickly send large quantities of data in web applications. A WebSocket establishes a connection between client and server and therefore after the initial connection establishment—every single message only has small overhead. Devices and servers can simultaneously transmit and receive data in real-time, making this protocol best-suited for IoT applications where low latency is critical, communication happens frequently, and data consumption is less important.

AMQP (Advanced Message Queuing Protocol)

AMQP is an open source protocol for Message-Oriented Middleware (MOM). It's designed to facilitate communications between systems, devices, and applications from multiple vendors—and it was not directly designed for IoT. Opposed to MQTT it supports more routing options than just publish / subscribe on topics—but this flexibility comes with complexity on application setup and additional protocol overhead. AMQP is also supported by Azure IoT for device communication.

CoAP (Constrained Application Protocol)

Constrained Application Protocol (CoAP) is designed for low-power, lossy networks, also known as "constrained" networks. CoAP is usually paired with User Datagram Protocol (UDP) which makes it highly efficient, making it appealing for IoT applications where battery conservation is important. For example, it's often used in smart meter communications. CoAP can also use TCP or SMS as transport mechanism.

LwM2M (Lightweight Machine-to-Machine)

The Lightweight Machine-to-Machine (LwM2M) protocol builds on CoAP for a highly efficient low-power communication, but LwM2M also specifies device management and provisioning functionality. Therefore, LwM2M, for example, has a standardized procedure to clarify which

security mechanism is used and how device firmware is updated. As LwM2M is build on top of CoAP it can be used with UDP, TCP, and SMS for data transport.

XMPP (Extensive Messaging and Presence Protocol)

Extensible Messaging and Presence Protocol is a communications protocol built on Extensible Markup Language (XML). XMPP was initially designed for instant messaging (IM)—which also explains that it comes with an overhead for the exchange of presence information and is not optimized for memory constrained devices. Nevertheless XMPP allows for the definition of the message format and therefore handles very well structured data, as well as establishes a device's identity and facilitates communications between different platforms. This open-source technology is highly accessible and is still being enhanced with new IoT-related developments.

DDS (Data Distribution Service)

Data Distribution Service protocol is a real-time, interoperable communication protocol designed for solutions that require significant coordination, reliable transmissions, and distributed processing between the devices itself. Instead of sending data to a central hub or broker, data can be directly been exchanged between peers making it more robust and efficient. DDS uses a publish/subscribe mechanism where devices subscribe to a topic and devices sending to the topic are then using multicast to distribute the information to the subscribers. DDS can use TCP and UDP as transmission protocol.

SMS / SMPP (Short Message Peer-to-Peer Protocol)

Short Message Service allows devices and applications to send and receive text messages over a cellular connection. Cellular IoT devices use Short Message Services (SMS) to send and receive data to the application or to another peer in the mobile communication network.

An application can communicate with devices programmatically by connecting to the Short Message Service Center (SMSC) of a service provider using the Short Message Peer-to-Peer Protocol (SMPP) or an SMS Rest-API. Telematics providers can use SMPP to provision and configure their devices remotely.

USSD (Unstructured Supplementary Service Data)

Also known as "Feature Codes" or "Quick Codes," USSD is a messaging protocol used in cellular networks based on the Global System for Mobile Communications (GSM). IoT businesses use USSD to retrieve text-based data (such as location, temperature, and other status updates) from IoT devices without relying on a data connection. Unfortunately, since many carriers are sunsetting their 2G and 3G networks, USSD will become obsolete in the near future.

Simple Sensor Interface (SSI) protocol

SSI enables IoT sensors to transmit data to or receive data from a terminal such as a computer or mobile device. As the name implies, the protocol is not very complex, so communicating via SSI uses very little power. However, there's been no update to the protocol since 2006, so it may already be obsolete.

GETTING STARTED WITH AN API

API stands for Application Programming Interface. In the context of APIs, the word Application refers to any software with a distinct function. Interface can be thought of as a contract of service between two applications. This contract defines how the two communicate with each other using requests and responses.

1. Select an API.

First, you'll want to find an API to incorporate into your business. You might already have your eye on an API, particularly if you're interested in one of the big wigs like the Facebook API. You might also want to search by cost — you may want to start with a free API before exploring paid APIs, for example.

Once you have an API selected, get your reading glasses on. It's time to look through the API documentation.

2. Get an API key.

As mentioned, an API key is used to identify yourself as a valid client, set access permissions, and record your interactions with the API.

Some APIs make their keys freely available, while others require clients to pay for one. Either way, you'll most likely need to sign up for the service. You'll then have a unique identifier assigned to you, which you will include in your calls.

Always key your key private, like you would a password. If your key leaks, a bad actor could make API requests on your behalf. You may be able to void your old key and get a new one if such a breach occurs.

3. Review the API documentation.

API Documentation is essentially an instruction manual about how to use an API. In addition to providing documentation, it usually includes examples and tutorials.

Refer to the documentation for how to get your key, how to send requests, and which resources you can fetch from its server.

It's hard to understate the importance of good API documentation. A company might offer a powerful API, but if developers can't quickly learn how to use it, it's not very valuable

4. Write a request to an endpoint.

Next up, you'll write your first request. The easiest method is to use an HTTP client to help structure and send your requests. You'll still need to understand the API's documentation, but you won't need much coding knowledge to be successful.

5. Connect your app.

Now that you understand how to make requests to your API of choice, you can sync your application with it. As a marketer, you don't need to worry about this stage of API integration. This is the job of a developer who will employ languages like Python, Java, JavaScript (and NodeJS), PHP, and more.

(OR)

GETTING STARTED WITH AN API

The most important part of a web service, with regards to an Internet of Things device, is the Application Programming Interface, or API.

• An API is a way of accessing a service that is targeted at machines rather than people.

•If you think about your experience of accessing an Internet service, you might follow a number of steps.

Ex: Flickr is an American image hosting and video hosting service,

Flickr enables users to post photos from nearly any camera phone or directly from a PC.

For example, to look at a friend's photo on Flickr, you might do the following:

1. Launch Chrome, Safari, or Internet Explorer.

2. Search for the Flickr website in Google and click on the link.

3.Type in your username and password and click "Login".

4. Look at the page and click on the "Contacts" link.

5. Click on a few more links to page through the list of contacts till you see the one you want.

6. Scroll down the page, looking for the photo you want, and then click on it.

•These actions are simple for a human, but they involve a lot of looking, thinking, typing, and clicking.

• A computer can't look and think in the same way.

•The tricky and lengthy process of following a sequence of actions and responding to each page is likely to fail the moment that Flickr slightly changes its user interface.

•For example, if Flickr rewords "Login" to "Sign in" , or "Contacts" to "Friends" , a human being would very likely not even notice, but a typical computer program would completely fail.

• Instead, a computer can very happily call defined commands such as login or get picture

WRITING A NEW API

When you know what data you have, what actions can be taken on it, and what data will be returned, the flows of your application become simple.

•This is a great opportunity to think about programming without worrying (at first) about the user interface or interactions.

•Although this might sound very different from writing a web application, it is actually an ideal way to start: by separating the business problem from the front end, you decouple the model (core data structure) from the view (HTML/JavaScript) and controller (widgets, form interaction, and so on).

•If you've programmed in one of the popular MVC frameworks (Ruby on Rails, Django, Catalyst, and so on), you already know the advantage of this approach.

•The best news is, if you start designing an API in this way, you can easily add a website afterwards.

WRITING A NEW API : Clockodillo

• Clockodillo is an Internet-connected task timer.

• The user can set a dial to a number of minutes, and the timer ticks down until completed.

•It also sends messages to an API server to let it know that a task has been started, completed, or cancelled.

•A number of API interactions deal precisely with those features of the physical device: o Start a new timer o Change the duration of an existing timer

o Mark a timer completed o Cancel a timer

o View and edit the timer's name/description

• And, naturally, the user may want to be able to see historical data: o Previous timers, in a list

- Their name/description
- Their total time and whether they were cancelled