TCP vs UDP, ICMP, and Raw Packet Crafting in C++

The TCP/IP Protocol Stack

Let's enter the real world! The TCP/IP model is the first of several communication stacks that we will study during this course. You use it every single day, just by *thinking* about your phone. Every piece of data transmitted across internet traverse the TCP/IP-stack in some shape or form[1].

The TCP/IP model is a simplified, practical version of the OSI model.

OSI Layer	TCP/IP Layer	Protocol Examples
7. Application	Application	HTTP, MQTT, CoAP, FTP, DNS
6. Presentation	↳ Application	TLS, SSL, encryption and encoding functions
5. Session	↳ Application	Session tokens, WebSockets
4. Transport	Transport	TCP, UDP
3. Network	Internet	IP, ICMP, IGMP
2. Data Link	Link	Ethernet, WiFi MAC, ARP
1. Physical	↳ Link	802.11, Bluetooth, RS-232, LoRa, ZigBee PHY

In this mapping:

- The **Application layer** in TCP/IP covers OSI layers 5–7.
- The **Link layer** in TCP/IP covers OSI layers 1–2.
- TCP/IP focuses on **interoperability and implementation**, not separation of concerns.

This chapter will focus on **TCP, UDP, and ICMP**, and how they operate at the **transport and internet layers**.

Embedded View: What Layers Really Exist?

On a device like an ESP32 or STM32 running a bare-metal application or RTOS:

- You typically **don't see the OSI layers** — just a socket API.
- Encryption (TLS) is often optional and handled by a separate library (e.g., mbedTLS).
- Session management is minimal or custom (e.g., MQTT keep-alives).
- The actual "stack" might look like:

```
[ App logic ]        --> your C++ program
[ MQTT / HTTP ]      --> application protocol
[ TLS / JSON ]       --> optional security/formatting
[ TCP or UDP ]       --> transport layer
[ IPv4 / IPv6 ]      --> network layer
[ WiFi / BLE / LoRa ]--> link + physical layers
```

Protocol Placement in IoT (by Layer)

Layer	Protocols / Functions
Application	MQTT, CoAP, HTTP, Modbus, LwM2M
Transport	UDP (CoAP), TCP (MQTT, HTTP)
Internet	IPv4, IPv6, 6LoWPAN, ICMP
Link	WiFi MAC, BLE, Ethernet, IEEE 802.15.4
Physical	RF transceivers (BLE PHY, LoRa, ZigBee)

Note on 6LoWPAN

- A **header compression layer** enabling IPv6 over low-bandwidth radio networks (802.15.4).
- Sits **between** the Internet and Link layers.
- Common in smart home and industrial IoT setups (used with **Thread**, **ZigBee IP**).

TCP vs UDP

TCP – Transmission Control Protocol

The TCP protocol is **connection-oriented**. That means that before any data is sent, we first have to establish a connection. We do that using the **3-way handshake**. For some reason there are a lot of weird handshaking going on in computer communications and I'm proud to announce that I'm not including any introvert computer nerd stereotypes in this chapter.

The TCP protocol is for this reason **reliable**. There are a lot of smart things going on. It can divide large amounts of data into smaller **packets** and send them across the Internet. Each packet is then assembled in the correct order, and they are checked for errors. If there are errors, the TCP protocol will not accept the package.

It is also **silly**. If a packet does disappear along the way for some reason, it will just sit there and wait for it to reappear. There are ways to work around this, but this is the reason the protocol is slow. There will be a lot of retransmissions and confusion.

Some protocols that work over TCP:

- HTTP(S) for web pages
- FTP for old school file transfers
- SSH for server administration and other really cool stuff
- SMTP for email

> Again, here there will be a sketch on the 3-way handshake. Hang in there, or send me one over teams. Please don't have ChatGPT generate it, I could have asked it to do that myself but I have too much artistic integrity.

UDP – User Datagram Protocol

So the UDP protocol is the Yolo protocol. The sender just fires off packets and then sort of shrugs. No connections are made, so we call it **connectionless**. The protocol will do no attempts at making sure the

packets arrive in order, I mean it doesn't even attempt to guarantee that the packets arrive at all.

This makes the protocol **fast** and **lightweight**. Anytime we care more about the packages arriving quickly than arriving at all, we turn to UDP.

For that reason IoT folks really like working with UDP. First of all, if the temperature data doesn't arrive on time, who cares? There will be a new update in like 5 seconds and if the temperature has changed wildly in the space of five seconds **and then back** you have more serious problems to attend to.

Second of all, no one's stopping you from building reliability on top of the UDP protocol. You can have algorithms that check if there hasn't been temperature data coming in for a few minutes, and then ask the sensor to retransmit. You can have algorithms that checks for errors. You can do whatever you do on top of the UDP protocol, picking and choosing from what you like about the TCP protocol.

Use cases for UDP:

- DNS (which also can use TCP, so cool!) is what lets you type `reddit.com` in your web browser instead of `151.101.129.140`.
- DHCP which gives you an IP address automatically, instead of you having to ask IT about it every time.
- VoIP is a protocol for phone calls.
- Video streaming

Why does it make sense for these protocols to use UDP?

Comparison Table

Feature	TCP	UDP
Connection	Yes (Handshake)	No
Reliability	Yes	No
Ordering	Guaranteed	Not guaranteed
Use Cases	Web, SSH, Email	Streaming, DNS, VoIP

ICMP – Internet Control Message Protocol

Purpose

We won't talk much about the ICMP protocol. It is used for diagnostics and error message, and regular users will not get in contact with it. Indeed, most advanced users will only ever use the type 8 and the type 0 parts of the protocol. They will not know this themselves, knowing only that they type:

```
ping 8.8.8.8
```

To see if they have an Internet connection.

Common ICMP Message Types

Type	Description	Example Use
0	Echo Reply	`ping` response
3	Destination Unreachable	Routing failure
8	Echo Request	`ping` request
11	Time Exceeded	`traceroute`

Tools that use ICMP

- `ping`
- `traceroute`

Traceroute is kind of cool. How does that work? Do try to use it and see what happens!

Byte Order and `htons`

What is Byte Order?

Different CPU architectures represent multibyte numbers in different orders:

- **Little-endian**: Least significant byte first (e.g. x86)
- **Big-endian**: Most significant byte first (used in networking)

What is `htons()`?

```
uint16_t htons(uint16_t x);
```

- Stands for **Host to Network Short**.
- Converts a 16-bit value from **host byte order** to **network byte order**.

Example:

```
uint16_t x = 0x1234;
uint16_t y = htons(x);
// On little-endian machine:
// x = 0x1234 → stored as 34 12
// htons(x) → converts to 12 34
```

Working with Raw Sockets in C++

What is a Raw Socket?

- A **socket that bypasses the normal TCP/UDP protocol processing**.
- Lets you manually construct packets (e.g., IP or ICMP).
- Requires **root privileges** (e.g., via `sudo` on Linux).

System Call Example

```
int sock = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
```

- `AF_INET`: IPv4
- `SOCK_RAW`: Raw socket
- `IPPROTO_ICMP`: Protocol field for ICMP

Address Conversion

```
inet_pton(AF_INET, "8.8.8.8", &dest.sin_addr);
```

- Converts human-readable IP to binary form
- `inet_pton` = "presentation to network"

ICMP Packet Construction (bonus)

We built an ICMP Echo Request packet manually using C++ and printed its contents in **hexadecimal**:

ICMP Header Fields (RFC 792)

Field	Size	Description
Type	1 B	Message type (8 = Echo Request)
Code	1 B	Subtype (0 for Echo)
Checksum	2 B	Integrity check (over entire packet)
Identifier	2 B	Matches requests and replies
Sequence	2 B	Order of request packets

Example Initialization in C++

```
icmp->icmp_type = ICMP_ECHO;
icmp->icmp_code = 0;
icmp->icmp_id = htons(0x1234);
icmp->icmp_seq = htons(1);
```

Checksum Calculation

```
unsigned short checksum(void* b, int len);
```

- Sums every 16-bit word
- Folds result to 16 bits
- Returns the one's complement

Dummy Payload

To fill the rest of the packet with sample data:

```
for (int i = sizeof(struct icmp); i < packet_size; ++i)
    packet[i] = i;
```

Printing the Packet in Hexadecimal

```
void print_hex(const unsigned char* data, int length);
```

- Groups output 16 bytes per row
- Useful for **debugging**, **learning**, and **Wireshark comparison**

Sending the Packet

```
sendto(sock, packet, packet_size, 0, (sockaddr*)&dest,
sizeof(dest));
```

- Sends the raw ICMP packet directly to the target
- No TCP or UDP layer involved

Observing in Wireshark

Use Wireshark with the following filters:

- **`icmp`** → shows ICMP packets
- **`icmp.type == 8`** → Echo Requests
- **`icmp.type == 0`** → Echo Replies

Compare the bytes captured to your printed hexadecimal output.

Summary of System Concepts

Concept	Summary
`htons()`	Convert 16-bit value to network byte order
`raw socket`	Allows full manual control over packet construction
`inet_pton()`	Converts text IP to binary format
`sendto()`	Sends data to a specific IP and port

`checksum()`	Ensures integrity of the packet header and payload
`ICMP`	Protocol for control messages, not transport

Suggested Practice for Students

1. **Capture and decode** the packet in Wireshark
2. Modify and recompile:
3. Change identifier and sequence number
4. Change payload
5. Implement ICMP Echo Reply handling (advanced)
6. Try sending packets to a local IP address and observe behavior
7. **Build Your Stack**: Implement a UDP-based telemetry client on ESP32 using lwIP or Arduino Ethernet.
8. **Draw the Stack**: Create a layered diagram showing how an MQTT message travels from a sensor to a cloud service over WiFi.
9. **Header Field Identification**: Compare IP and TCP headers and explain what fields belong to which layer.

Quiz Questions

1. What is the primary difference between TCP and UDP in terms of connection handling?
2. Which ICMP type is used for Echo Requests and which for Echo Replies?
3. Why is `htons()` necessary when assigning values to ICMP fields?
4. What is a raw socket and why does it require root privileges?
5. How is the checksum for an ICMP packet calculated?
6. In what scenarios would you prefer UDP over TCP?
7. What is the purpose of `inet_pton()` and what would happen if you omitted it?
8. How does `sendto()` differ from `send()` in the context of raw sockets?
9. What fields are required in a basic ICMP Echo Request packet?
10. How can you confirm that your raw ICMP packet was sent successfully using Wireshark?
11. Which layers in the OSI model are merged in the TCP/IP model?
12. What protocols typically run at the Application layer in IoT devices?

13. What are the key differences in purpose between the OSI model and the TCP/IP model?
14. How does 6LoWPAN support IoT devices in low-power networks?
15. Where would you classify TLS in the OSI model? What about in TCP/IP?
16. Why is the TCP/IP model often preferred in embedded IoT systems?
17. What layer would a WiFi MAC address be associated with?
18. Which tool can you use to observe all layers of a communication stack in practice?
19. How is reliability handled differently in TCP vs UDP, and what layer is responsible?
20. What protocol in the Internet layer provides diagnostic capabilities (e.g., ping)?

---

1. Well, not *literally* every piece of data. There are other protocols out there (like OSI, X.25, or even carrier pigeons), but let's not ruin the moment with technicalities. Your phone probably uses TCP/IP though, so we'll call it a win.