NETWORKING OSI MODEL

Networking 101: A Basic Overview of Packets and OSI

by Kevin Mai, Corbin Crutchley

March 11, 2020 2220 words

Part of our series: Networking 101

1. Part 1: A Basic Overview of Packets and OSI
2. Part 2: UDP & TCP

Networking is the foundation that all interactions on the internet are built upon. Every chat you send, every website you visit, *every interaction that is digitally shared with another person is built upon the same fundamentals*. These fundamentals layout and explain how computers are able to communicate with one another and how they interlink with one another in order to provide you with the experience you've come to know and love with the internet. *This article is the beginning of a series that will outline the core components that construct those very same fundamentals*. However, without a holistic view, it may be difficult to follow along with many of the more minute details. As such, we'll be covering what each of the core components at play is and how they fit into the grand scheme of things. The articles to follow will explain and expand upon this base of knowledge.

It's important to note that *"networking" is a broad, catch-all term that infers **any** communication between one-or-more computer devices*. This includes parts in your computer communicating between themselves. For example, do you need your computer keyboard input to be processed by your CPU to display data on-screen? That requires the networking to transfer the data to the CPU and from your CPU to your monitor, so on and so forth. As a result, this article will be a bit broad in order to cover not just cross-device hardware, but inter-device communication fundamentals as well. We'll dive deeper into cross-device communication in future articles in the series

> If you don't understand how CPUs work, that's okay, it's not required knowledge for this post. Just know that they take in binary data, process it, then send data out to other devices to use. They convert binary data into binary instructions for other devices to follow
>
> That said, you need the right binary data to be input into the CPU for it to process, just like our brains need the right input to find the answer of what to do. Because of this, communication with the CPU is integral

Architecture

There are a lot of ways that information can be connected and transferred. We use various types of architecture to connect them.

Computers speak in `1`s and `0`s, known as binary. These binary values come in incredibly long strings of combinations of one of the two symbols to *construct all of the data used in communication*.

> We covered how these two numbers can be combined to turn into other data in another post. For a better understanding of how binary represents data, check out that post.

This is true regardless of the architecture used to send data - it's all binary under-the-hood somewhere in the process. The architecture used to send data is simply a way of organizing the ones and zeros effectively to enable the types of communication required for a specific use-case.

Bus Architecture

For example, one of the ways that we can send and receive data is by, well, sending them. *The bus architecture*, often used in low-level hardware such as CPU inter-communication, *simply streams the ones and zeros directly*.

It doesn't wait for a full message to be sent or provide many guidelines for how to send the data, it just tells them to "come on over".

0:00 / 0:01

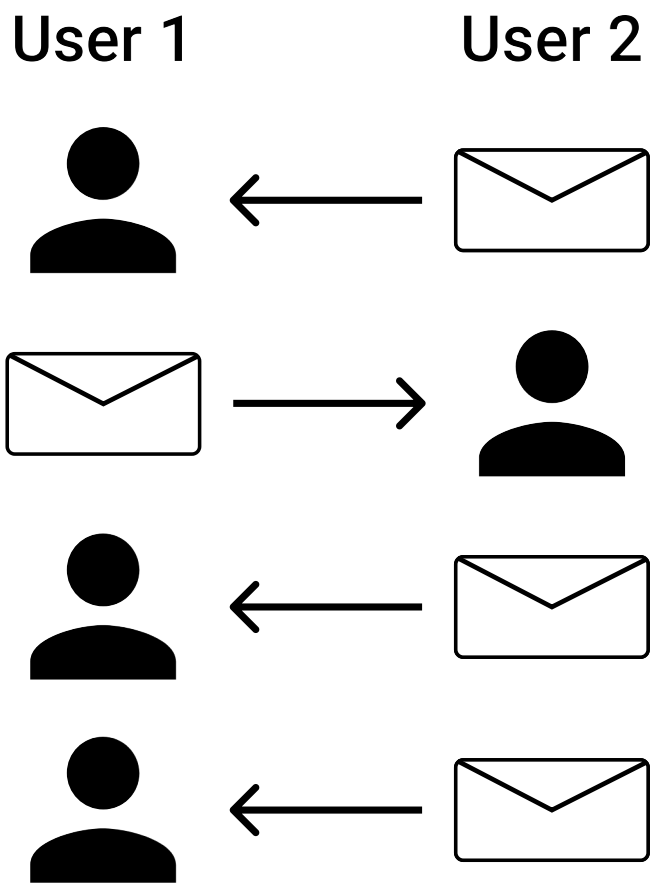
In this example, the bus icons are similar to binary data - either a one or a zero. They're able to *move as quickly as possible down a "lane" that is allocated for a specific stream of data to come through*. A collection of "lanes" is called a "bus" (which is where the name of the architecture comes from. I was just being silly by representing the binary data as buses in the video above). Your system, right now, is streaming through ***many*** thousands of these busses to communicate between your CPU and I/O devices (like your keyboard or speakers) and tons of other things. They're *typically divided to send specific data through specific lanes (or busses)*, but outside of that, there's little high-level organization or concepts to think through.

Furthermore, because error-handled bi-directional cancelable subscriptions (like the ones you make to servers to connect to the internet) are difficult using the bus architecture, *we typically don't use it for large-scale multi-device networks like the internet*.

Packet Architecture

The weaknesses of the bus architecture led to the creation of the packet architecture. The packet architecture requires a bit more of a higher-level understanding of how data is sent and received. To explain this concept, we'll use an analogy that fits really well.

Let's say you want to send a note to your friend that's hours away from you. You don't have the internet so you decide to send a letter. In a typical correspondence, you'd send off a letter, include a return address, and wait for a response back. That said, *there's nothing stopping someone from sending more than a single letter before receiving a response*. This chart is a good example of that:



Similarly, a packet is *sent from a single sender, received by a single recipient, addressed where to go, and contains a set of information*.

Metadata

Letters may not give you the same kind of continuous stream of consciousness as in-person communications, but they do provide something in return: structure.

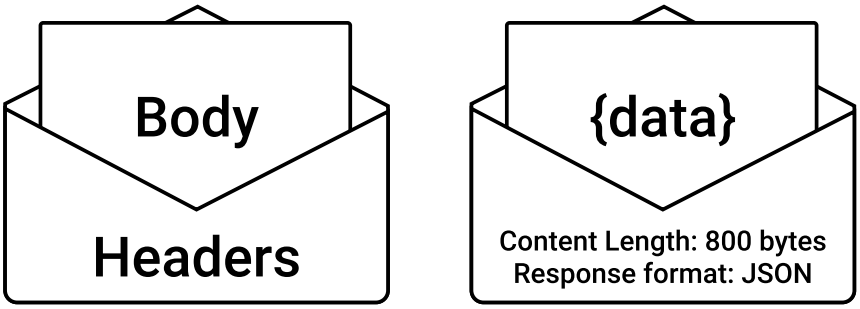
The way you might structure your thoughts when speaking is significantly different from how you might organize your thoughts on paper. For example, in this article, there is a clear beginning, end, and structured headings to each of the items in this article. Such verbose metadata (such as overall length) cannot be communicated via in-person talking. *The way you may structure data in a packet may also differ from how you might communicate data via a bus*.

That said, simply because there's a defined start and an end does not mean that you cannot *send large sequences of data through multiple packets and stitch them together*. Neither is true for the written word. This article does not contain the full set of information the series we hope to share, but rather provides a baseline and structure for how the rest of the information is to be consumed. So too can packets provide addendums to other packets, if you so wish.

Headers

Not only does the spoken-word lack the same form of structure that can be provided by the written word, but *you're also able to categorize and assign metadata to a letter* that you wouldn't be able to do with a conversation. You do this every time you send a letter to someone through the mail: You include their name, address, and return address on the envelope that's used to send the letter. The same is done with packets.

While the "body" of your packet would contain the data you want the other party to receive, the "header" of the packet might contain data **about** said data. Such metadata might include the size of the contained data or the format that data is in.



As a result, you might have a middleware packet handler that reads only the header of the packet in order to decide where to send the packet in question - much like the mail service you use will read the outside of the envelope to see where to send your letter

0:00 / 0:12

It Takes A Village To Send A Letter

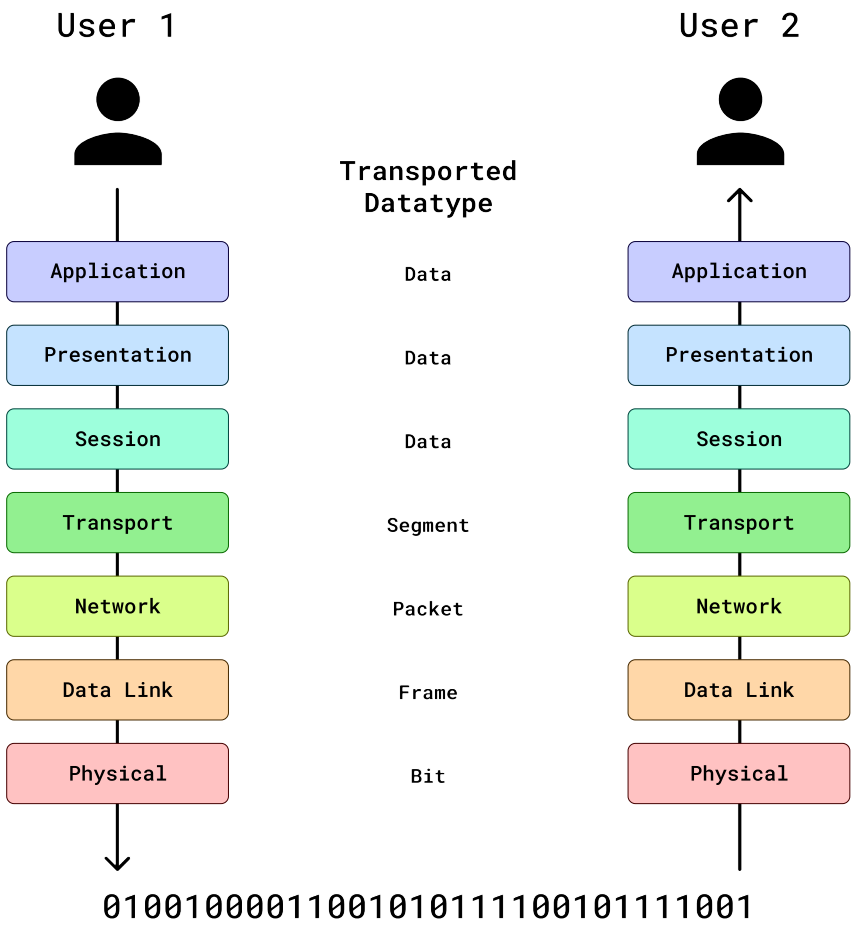
Understanding what a letter is likely the most important part of the communication aspect if you intend to write letters, but if someone asked you to deliver a letter it helps to have a broader understanding of how the letter gets sent. That's right: *there's a whole structure set in place to send the letters (packets) you want to be sent*. This structure is comprised of many levels, which we'll outline here.

> For each of these levels, there are many intricacies that we won't be touching on. This is for the sake of conciseness. As this article is only the start of the series, you can expect these intricacies to be explained with greater detail as these articles are released

This structure is comprised of seven levels for most networking applications. *These layers interact with one another as a form of stack-the-blocks method*. For example, describing layer 4 encapsulates the behavior of layers 3 and 2. As a result, lower-level applications of networking may have fewer than seven layers. That said, those seven levels are, in order:

- Application
- Presentation
- Session
- Transport
- Network
- Data Link
- Physical

As you communicate with others online, and as computers communicate within themselves using packets, they go down those layers, then back up them to be processed and interpreted



This breakdown of layers is referred to as the OSI model. This conceptual model allows us to think abstractly about the different components that make up our network communications. While we won't do a deep-dive into each layer here, we'll try to at least make them fit into our mailroom analogy.

Let's start from the bottom and make our way up. Remember that each of these layers builds on top of each other, allowing you to make more complex but efficient processes to send data on each step.

Physical

The physical layer is similar to the trucks, roads, and workers that are driving to send the data. Sure, you could send a letter just by handing letters one-by-one from driver to driver, but without some organization that's usually dispatched to higher levels, things can go wrong (as they often do in a game of telephone).

In the technical world, *this layer refers to the binary bits themselves* (which compose to makeup letters and the rest of structure to your data) *and the physical wiring* constructed to transfer those bits. As it is with the mail world, this layer alone *can* be used alone, but often needs delegation from higher layers to be more effective.

Data Link

Data link would be like UPS or FedEx offices: sending information between post office to post office. These offices don't have mail sorters yet (that's a layer up) but they do provide a means for drivers to arrive to exchange mail at a designated area. As a result, instead of having to meet the drivers in the road to receive my mail, I can simply go to a designated office to receive my mail.

Likewise, *the data link layer is the layer that transfers binary data between different locations*. This becomes especially helpful when *dealing with local networks that only exchange data between a single physical location*, where you might not need the added complexity large-level packet sorting might come into play.

Network

The network layer is similar to the mail sorters. Between being transferred from place to place, there may be instances where the mail is needed to be sorted and organized. This is *done with packets in the network layer to handle routing* and other related activities between clients

Transport

The transport layer delivers it from the post office to my apartment building. This means that not only does the package gets delivered from post-office building to post-office building, but it gets to-and-from its destination as intended.

Session

With newer packages delivered through services like UPS, you may want a tracking number for your package. This is similar to the session layer. With this layer, it includes a back-and-forth that can give you insight into the progress of the delivery or even include information like return-to-sender.

Presentation

But when a package gets received by you, it doesn't stop there, does it? You want to bring the package inside your home. For most packages, this is relatively trivial - you simply take it inside. However, for some specialized instances, this may require hiring movers to get a couch in your house. In this same way, HTTP and other protocols don't typically differentiate between the presentation layer and the application layer, but some networks do. When they do, they use the presentation layer to outline how the data is formed for sending and receiving

Application

You've just been delivered the fancy new blender you ordered for smoothies. After unwrapping the package, you plug it in and give it a whirl, making the most delicious lunch-time smoothie you've ever had. Congrats, you've just exemplified the application layer. In this layer, it encapsulates the layer your user (developer or end-user alike) will use, the application that communicates back-and-forth and the reason you wanted to send data in the first place.

Conclusion

We've done an initial overview of what layers you utilize when accessing a network. Although we've only done an initial glance at those layers, the next few steps will outline what those layers comprise of, and how data can transfer across a network. These steps will come in the order of various articles in the series in the future. To make sure you don't miss those next articles, be sure to subscribe to our newsletter down below!

You can also join us in our community Discord and chat with us there!