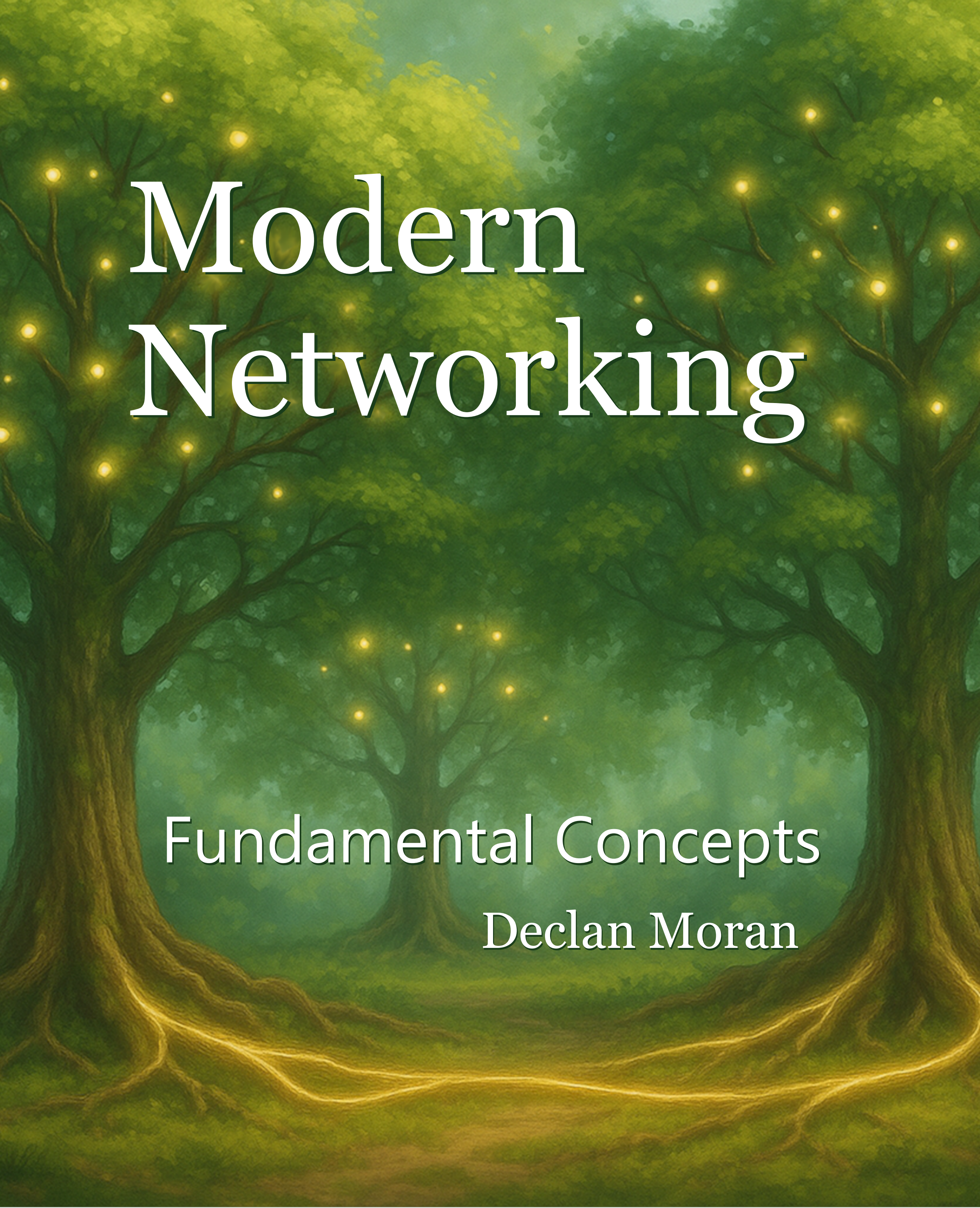

Modern Networking

Fundamental Concepts

Declan Moran

Modern Networking

Fundamental Concepts

Declan Moran

Table of Contents

Preface

When I was starting out in software, one of my senior colleagues sent me a network related snippet he suggested I integrate into my code base. I was having some difficulty understanding it, and asked if he could explain it to me. He said (a little impatiently), that it was part of *the basics* all software developers must know. A little taken aback, I replied that if he would be so kind as to recommend me a book where these basics are explained, I would be happy to read it. Initially, he didn't reply. But the next day he approached me and said he had been thinking — that there indeed wasn't any such good book he could recommend. It's now over 20 years later, and the networking landscape isn't any simpler. This book is my attempt to help you understand the basics of networking so you don't have to figure it out alone like I did.

Who this book is for

This book should provide a good starting point for any technically savvy reader who wishes to acquire a solid grounding in the fundamentals of networking. Those who have previous exposure but feel some of the dots don't yet connect (e.g., software developers, system administrators, and computer science students) should feel particularly at home. Experienced cloud engineers or network specialists will not likely find much new here, but those wishing to become such specialists may well find this book a very good first step in the right direction.

Whatever the background of the reader, I hope they will discover, that once they master the fundamental concepts, that a shroud of mystery lifts from networking, and what's left presents a very logical and satisfying mosaic. In a world where technologies come and go at a rapid pace, networking is an area with a bright future, where, with a solid grasp of the *fundamental concepts*, the limits are defined largely by one's creativity.

The Chicken and the Egg

It can sometimes be difficult to explain a concept clearly and succinctly without, in the process, at least mentioning a related concept that hasn't been explained (independently) yet, and vice versa. For example, a good explanation of what a chicken is, may require mentioning that they lay eggs. And a good explanation of an egg might require mentioning that they are often laid by chickens. But which to explain first—the chicken or the egg? No matter what the choice, the result is less than ideal. But this seems to be an unavoidable consequence, and one may need to read a little further before the current concept becomes fully clear. Rest assured, however, that this book shall strive not to hide any eggs, but to place them as close and clearly as possible to their respective chickens.

Why networking is difficult, and why its not.

> *“When I use a word, it means just what I choose it to mean—neither more nor less.”*
>
> — Humpty Dumpty in Lewis Carroll's, Through the Looking-Glass

Unfortunately, Humpty seems to have no small following in networking circles. A single term is often used in different contexts to mean different things, or different terms in separate places for a single concept. Often sources use terms inconsistently or don’t clearly define what they mean by them, forcing you to consult other sources, which in turn use the term you’re trying to clarify in yet a different way.

Another challenge is the sheer size of the subject. There is a lot of ground to cover in networking—hardware, software, layers, security, cloud—the list goes on, and it’s not getting any shorter. It’s very easy to get lost *down a rabbit hole* of details at various points, and lose sight of (and possibly motivation to grasp) the bigger picture.

In this book, I strive to use terms consistently, and pursue as best as possible (chicken-egg paradox notwithstanding), a top-down approach. Throughout the text, you’ll find sections designated as tips or cautions—these provide additional insights but can be safely skipped on your first pass without hindering your understanding of subsequent material. I also make ample use of footnotes to isolate edge cases, elaborations or precision not crucial for a basic understanding, and to better highlight the central concepts.

The book focuses on *modern* networking, and as such omits discussion of unusual configuration or legacy hardware. The hope is that once the reader grasps the core concepts, in a big picture setting, using the terms presented, it will put them in a much better position to navigate other sources or further pursue any more specialized directions.

Fundamental Concepts (Volume 1)

This book lays the essential groundwork for understanding modern computer networks. Whether you’re a technical professional or someone who simply wants to demystify networks, it will give you a clear and accessible introduction to the fundamental concepts. The fundamental concepts one must understand in order to understand the topic as a whole. And it strives to present them as *simply as possible—but no simpler*.

You'll explore everything from network basics to more advanced ideas like security, protocols, URLs, and the layered structure of networks. Key principles such as nodes, ports, abstraction, standardization, and network domains are covered, along with concepts spanning all network layers — from physical connections to higher-level services. It deliberately keeps code and configuration listings to a minimum, preferring instead more intuitive, easily understandable explanations, without sacrificing clarity in the process.

If you're planning on diving into the planned follow-up volume *(Modern Networking - Software and Techniques)* which focuses on more advanced, hands-on techniques, code samples, and troubleshooting recipes, this first volume will provide the conceptual background you need to confidently apply those practical skills, and what is (or must be) happening behind the scenes rather than just following instructions blindly. Even if you're not interested in writing code or configuring networks, this book will still give you valuable insights into the networks that underpin our digital world. By the end, you'll have a solid understanding of how networks function, empowering you to make sense of the technology around you.

Acknowledgments

This book is the fruit of the combined efforts of many people, without whom it would not exist in its present form. I would like to sincerely thank everyone involved, particularly the following:

Oliver Moran, whose eagle eye, deep technical competence and remarkable capacity to think from first principles shaped this work profoundly. His tireless attention to detail, cognitive flow and consistency, along with our many fruitful discussions, elevated the quality of this book in countless ways.

Kevin Sandermann and *Marcus Mechmann*, who devoted considerable time and effort to sharing their vast knowledge, despite the demands of leading teams of 100+ cloud and DevOps experts and architecting Azure and AWS infrastructure for world-renowned organizations. I had the great fortune of receiving their guidance, always delivered with patience and grace, and the book benefited enormously from their contributions.

Gerry Siegemund, for his unwavering support, congenial spirit and professional encouragement.

Florian Moran, whose impressive ability to think outside the box, ask deeply insightful questions, and recognize patterns others miss, brought fresh perspectives to every chapter.

John Devins, former senior editor at O'Reilly, who convinced me to begin writing this book. I'm deeply grateful for his support, encouragement, patience and gracious nature. Without him, this book would likely never have existed.

Oliver Feeley an inspirational English teacher, who motivated, enriched and could make even the mundane seem somehow captivating.

Scott Meyers who brought technical writing to a new level with clarity and elegance, writing with the simplicity that comes from true understanding.

Jonathan Griffiths, whose pertinent and insightful feedback greatly improved the content and clarity, and whose meticulous eye caught numerous typographical mistakes.

For substantial feedback, valuable insights and thorough error checking: *Raymond Goggins, Sergey Antonyukov, Aoibhe Fleming, Tino Kreutzberger, Max Darscht, Markus Ewers, Lennart Graewe, Karsten Ludwig Hauser, Björn Wellegehausen, Thomas Moran, Tim Tenhagen-Krabbe, Christoph Schlosser, Irina Bartmann, Evelyn Moran, Andreas Belke, Pierre Totzki, Niklas Mevenkamp, Stephan Linz, Maeve Moran, Kaltrina Nuredini, Sean Fleming, Jonathan Johnson.*

For various helpful comments and constructive contributions: *Irene Grusla, Carlos Eduardo Niño, Kaveh Faghihi, David Petroff, Reinhard Meinel, Craig Scott, Matthew Filipovich, Florens Dietz, Sara Hunter, Melissa Potter.*

If I have forgotten anyone, please let me know.

Conventions used in this book

The following typographical conventions are used in this book:

Sidebar Title

Here's a sidebar. Sidebars are great for setting aside a section of text that is related to the surrounding content but that doesn't necessarily fit into the main flow.

This is a tip or suggestion.

This is a caution or warning

Cross-References and Index Entries

Cross-references are highlighted like hyperlinks on the page. If you have an electronic version of this book, you can click on the highlighted text to jump to the *target* location. For example, clicking on *electromagnetic wave* will jump you to the section *Electromagnetic Waves and the Role of Media*. This helps mitigate the *chicken-and-egg* problem.

If you're reading a printed version, you can still follow these cross-references by looking up the highlighted term in the index. The index contains two types of entries: cross-reference terms (like *electromagnetic wave*) that point to their target sections, and traditional index entries (like section titles and key concepts). For example, both *electromagnetic wave* and *Electromagnetic Waves and the Role of Media* may appear in the index pointing to the same page number, allowing you to navigate by either the link text you encountered or by the section title itself.

Metaphor as a tool

Metaphor and analogy can be helpful tools for understanding new concepts. I felt this might be particularly so in the case of networking. Even the best tend to have limits where they break down, and are no longer valid or even counterproductive. Though I came up with some metaphors and analogies that help clarify isolated concepts in networking, try as I might, I couldn't manage to find existing ones that would serve the big picture understanding, without breaking down somewhere important. So I decided to create some new ones — short science fiction stories, *Aetheria*, and a sequel called *Metropolea* in chapter 3, which, I hope, might be easily relatable for the target audience. I hope the reader will indulge this unusual interlude in an otherwise factual book. My hope is that, at various places in the rest of the book, the reader may be able to draw parallels with concepts (or people) from the narrative, that help them see where the individual pieces fit into a bigger picture. Those who would prefer to skip it can, however, feel free to do so.

Book Website

The companion website for the book is *modern-networking.com*. Please keep an eye on it for new editions, bonus content and extra information, such as *Frequently Asked Questions*—and answers.

Aetheria

In a kubit of Lumia, the home planet of the Aetherians, Dalvin lay on what we would have called a bed (if there had been a bed). The Zertax crystal his uncle had sent him glowed subtly. It was one of the few material objects in his kubit, the majority of which was a motor sensory simulation created by the kubit's electromagnetic field — what we might have called a wireless virtual reality (if it had been virtual). It vibrated imperceptibly at frequencies that resonated with Dalvin's theta waves, helping him to relax and stimulating his creativity.

That the Aetherians had become a dominant civilization in that sector of the Andromeda galaxy was due, in no small part, to the invention of the Flux Transporter. Dalvin's great-great-grandmother, Radia, had designed the first functioning prototype—a device capable of creating a precise subatomic blueprint of an object. This blueprint could then be transmitted through a tachyon stream faster than the speed of light through the subspace medium to a remote receiver, where advanced nanotechnologies could assemble an object that was an exact replica of the original.

Dalvin gazed at the crystal and thought of his great-great-grandmother, caught up in the controversy surrounding the proposed use of the technology to transport blueprints of sentient beings, with optional (and costly) enhancements. Though she herself received little credit in her lifetime, she had shaped the lives of so many. His uncle, on one of his many voyages, had found the crystal on a moon of Targon and sent it by Flux to Dalvin.

Once her prototype of the Flux Transporter was made production-ready, the prime planets and major moons all hurried to erect high-performance flux interfaces, each of which broadcast its subatomic signals into the medium and received those broadcast by the others. However, as more and more interfaces were created, interference between the individual signals competing for space in the medium caused more and more transports to fail.

That's when the Stellar Switch was created—a magnificent feat of engineering even for the Aetherians. It was the size of a small protoplanet, with 100 times more portals than the number of inhabited worlds in all the Aetherian star systems (at the time). Each interface on a planet or moon was assigned its own dedicated portal on the switch, to which it sent its subatomic signal in a focused beam directly, instead of polluting the medium. And from which it directly received signals from others, the switch forwarding the signals reliably and instantaneously. In fact, it worked so well it was easy to forget it was there at all.

That was a century ago—an aeon at the current rate of Aetherian technological advancement, during which the Aetherians had created their first Dyson Sphere in less time than it took Khufu to build the Great Pyramid. Most of the sphere was flux transported on-site using a winding fleet of interface (WiFi) satellites that orbited the home star.

Neighbouring stellar systems (some of which possessed rare crystals) that were colonized (or assimilated) by the Aetherians had their own Flux networks, centered on their own stellar switches. But *not even the Aetherians were able to build a switch big and powerful enough to span multiple star systems. The individual stellar networks, each with their own switches, were never designed to facilitate the sending of an object managed by one switch to a destination interface in a distant part of the galaxy—possibly separated by a convoluted web of thousands or more switched networks, constantly changing and updating with the individual stellar systems themselves.*

Dalvin shifted his focus to the holographic display, his retina dancing commands to his personal Comanche 64 quantum computer, interacting with it almost telepathically as the crystal continued to vibrate. Sometimes he wasn't sure where his thoughts ended and those of the computer began. Were the holograms generated by the computer reacting to him, or the other way around? Did his thought cause information to flow in his brain, or did the flow cause the thoughts? Are we more than our thoughts? Do we perceive reality, or does it create a perception of us? Action and reaction dancing in a quantum blur as the crystal hummed in harmony. Oscillations in the invisible spectrum, simple photons carried like foam on the tide of an expanding and contracting universe, bits of existence, emergent sentience, collective reality.

Dalvin didn't know it yet, but 11 versions of himself would one day experience the Aetherian advance to a Type 3 civilization, to which one of them would make a decisive contribution. And upon their Flux Transportation would be engineered the galaxy-spanning Interstellar Protocol (IP).

For my mother who taught me to read and write, and my father who taught me to learn—I owe you both so much. For my children Merlin, Jolinus, Rosalin, Oliver, and Florian who fill me with pride and joy. For Irina, who brings love into the world.

1. Foundations—understanding and describing networks

This chapter provides the essential foundation for understanding what a computer network is, and what it is not. We'll begin by defining the basic building blocks of modern networks and explore the core concepts needed to navigate more complex network structures.

You'll learn about nodes (the devices that make up the network), ports (the entry and exit points for information), and the media through which data flows between them. From there, we will introduce fundamental ideas like layers of abstraction and standardization, which help to simplify and organize the complexities of networks.

The chapter will also dive into important concepts such as broadcast domains, collision domains, and network interface cards (NICs), and how networking differs from other kinds of communication technology. These ideas set the stage for understanding the interactions and relationships between devices on a network, helping you see how and why networks are structured the way they are.

By the end of this chapter, you'll have a solid grasp of the basic principles that form the backbone of all networks, giving you the conceptual framework to explore more advanced topics, and specific types of networks, in the chapters to come.

What is a Network?

In the most general sense, a network is two or more separate entities somehow connected, together with any infrastructure required to connect them. In the context of this book, I will be discussing *modern computer networks*—the entities are computerized and the connections between them are for communication.

Nodes and Ports

A network device is a (computerized) device equipped with the hardware and software it needs to take part in network communications. A computer network is composed of one or more (network) *devices* called (network) *nodes* connected by *media*, as illustrated in *Figure 1*.

Each of the devices can be almost anything (as long as they have the appropriate software and hardware): a high-end server, a desktop pc, a smartphone, a Raspberry Pi, or a tiny sensor measuring temperature. They are all *computerized devices* and, being integrated into a network through which they can communicate with one another, makes them nodes of that network. All network nodes are devices, but not all devices are network nodes. Just like all footballers are people, but not all people are footballers. Being capable of and motivated to play football, makes certain people footballers, and being integrated into a team makes those footballers players of that team.

Media are discussed more in the next chapter (remember the *chicken and the egg*?). For now, you can just consider them to be cables (joining the nodes). Each node has one or more *ports*, each of which acts like a *door* between the device and the rest of the network. A network node needs at least one port, since without one there is no way in or out and thus no possibility to *attach* to the network. Any port can be the *door* to at most one node, but a node may contain multiple ports. The rest of the network only *sees* (or cares about) the ports. It's a bit like a courier delivering a package who may not see if multiple doors on a large building are multiple entrances/exits to a single business office or each a separate entrance to a separate (independent) business office. They just care about the single door they need to use to deliver the package, and if they can identify (e.g., from a number written on the door) that it's the correct *address*. What is otherwise happening in the building or whether other doors also lead in or out of the same building, and if so what's behind them, need not concern the courier.

There are two basic types of network nodes: those that make use of (the services of) the network, and those that help provide (services to) the network. Consider again the analogy with a package delivery system, in the context of which one can distinguish two different types of buildings: those of the users of the system who are the original source and final (intended) destination of packages, and those of the logistics company who may need extra buildings to temporarily store and process packages while in transit. The users never address a package to one of the storage facilities of the logistics company, and aren't generally aware of or interested in which, if any, logistics company building a package enters or leaves on its journey to its final destination. The details may vary over time (if, for example, the logistics company needs to add or remove buildings as part of an expansion or temporarily to avoid roads under maintenance or make better use of new improved ones) or from one package to another (packages sent further may need to be stored or processed more times), of which the users of the system may remain blissfully unaware. The primary concern of the users is that their packages get delivered reliably and speedily.

There are no accepted standard terms to distinguish nodes that are users of the network from nodes that service the network. This is unfortunate, as having names for important concepts one regularly talks about is very useful (as even Humpty Dumpty would surely agree). So in this book, I will use the terms network *user nodes* and *service nodes* respectively. Similarly, there is no standard term for ports on service nodes to distinguish them from those (called *interfaces*) on user nodes, so I will refer to the former in this book as *service ports*.

One can distinguish two fundamental types of nodes on a network: *user nodes* — which use the services of the network, and *service nodes* — which help provide it.[1] Note that, as devices, there is no fundamental difference between user nodes and service nodes apart from their roles in the network. Consider people in a restaurant, where the staff represent service nodes, and the customers represent user nodes. Fundamentally, there is no inherent difference between them (each is, in principle, capable of either working or dining in a restaurant). However, in any particular restaurant, each person can only play one of these mutually exclusive roles.

Infrastructure

The network service nodes, together with any media (e.g. cables) that connect nodes, make up what is called the *infrastructure* of the network, as illustrated in *Figure 2*. You can think of it as the logistics company, together with any pathways, roads, and buildings they use before initial pickup and after final delivery.

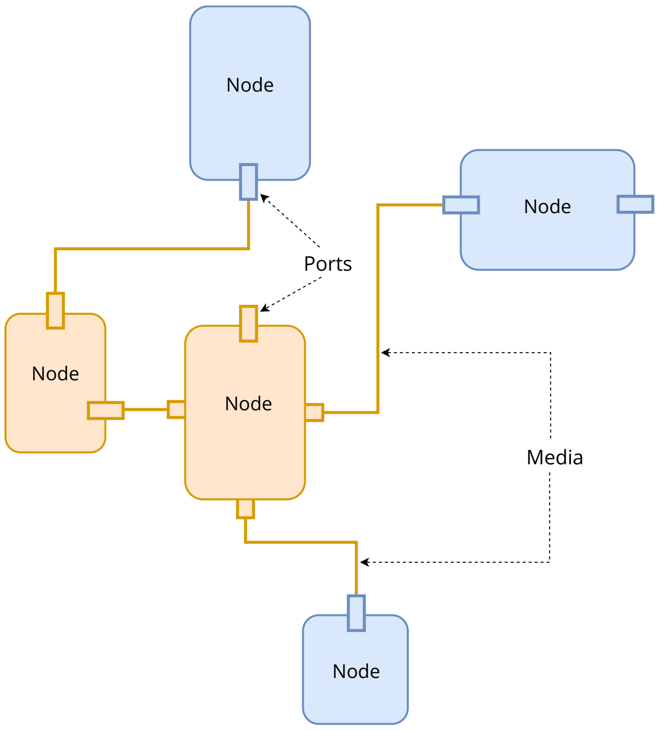

Figure 1. **Structural View**—*the components of the network. A network is composed of nodes and media (such as cables), which connect the nodes at their ports. There are two types of nodes—user nodes (shown in blue) which make use of (the services of) the network, and service nodes (shown in brown). Together, the service nodes and media form what's called the infrastructure of the network—see Figure 2.*

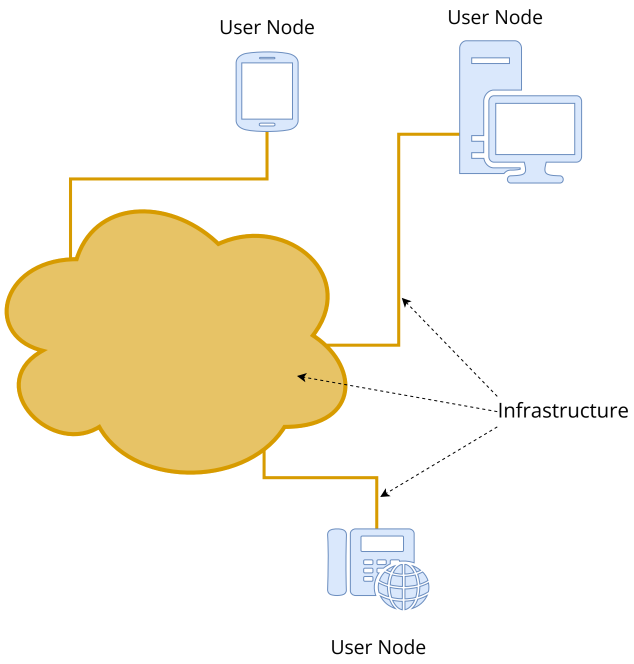

Figure 2. **Functional View** *of a network — The infrastructure (shown in brown) provides a service for user nodes (shown in blue), which enables the user nodes to exchange information with one another. The details of the infrastructure itself, and how it is (internally) composed, is neither readily apparent nor of interest to the users of the network. The user nodes only care if they can exchange information reliably with one another. Compare Figure 1*

Network

A computer network can be defined as one or more *ports* located on *(user)* nodes[2] that are connected by *infrastructure*, through which they communicate by exchanging information with each other. The ports act like doors, on their respective nodes. Each node requires at least one port to access the network, which are the original sources and final destinations of all packets of network information sent by one to another. The infrastructure is composed of *(service)* nodes and media such as cables. See also *Figure 2*.

Communication Technology

Networking (technology) can be seen as a subset of *communication* (technology) where the emphasis is on enabling effective communication between *many interfaces*, more *widely distributed*, in more *complicated topologies*. Other communication technologies, such as USB, FireWire, RS-232 (Serial), Bluetooth etc, are neither designed with these use cases in mind nor suitable for such communication, and as such will not be the subject of this book[3].

Network Interface Cards

Devices like PCs often have their ports located on a *Network Interface Card (NIC)*. In addition to the ports themselves, as can be seen in *Figure 3*, such cards also contain specialized hardware (chips) that are typical of (shared) resources used to pre-process information before sending, and post-process information after it arrives at the ports.

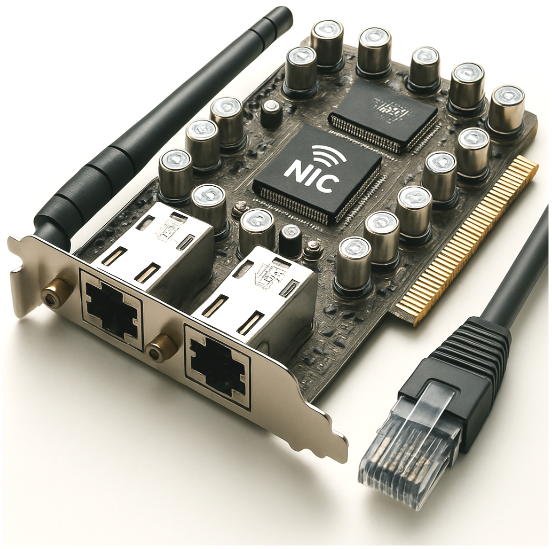

Figure 3. A **Network Interface Card (NIC)** *with two Ethernet ports (the slots) and one Wi-Fi port (the antenna). Also shown is an Ethernet cable that can be inserted into one of the Ethernet ports. On the end of the cable, you can see a semi-transparent connector, that helps it fit into the slot. Though important, for example, for technicians installing network equipment, as far as this book is concerned, you can consider such connectors to be part of the medium, as they play no significant role in the networking concepts discussed here.*

Though a single node can contain multiple ports, all of them need not necessarily all be on a single network. Some may be on one network, some on another (separate/independent) network, and some on no network at all.

Broadcast and Collision Domains

The **Broadcast Domain** is the set of (other) interfaces on the network an interface can successfully send information to. Ideally, this will be all (other) interfaces on the network.[4]

The **Collision Domain** is the set of (other) interfaces information reaches (as an unintended side effect) when an interface (intentionally) sends information to specific interfaces. The smaller the collision domain, the better.

Imagine a large room full of people. When a person wants as many people as possible to hear what they are saying, they speak loudly. The (maximum) number of people who actually hear what they are saying is the speaker's *broadcast domain*. Ideally, this would be everyone in the room. When a speaker wants just the person standing closest to them to hear, they speak quietly and directly to their partner. Those other than this partner who still hear (and who as a result may feel like their own private/direct conversations are being disturbed) are like the speaker's *collision domain*. The direct/private conversations of these people interfere (unintentionally) with each other. Ideally, this would be nobody.

Domains as discussed here may seem a little contrived or academic at first, but they play an important role which will become clearer in later chapters, so it's well worth taking note of them now.

❗ In many sources, the terms *Node* and *Host* are often confusingly used when a *port*[5] is actually meant.

❗ Don't confuse *port* as defined here with *port number* — often confusingly abbreviated to just *port*, which has a different (software related) meaning.

❗ The term *domain* is unfortunately used in different contexts to mean different things. Don't confuse the concepts of *broadcast* and *collision domain* as described here with others such *administrative domain*, *routing domain* and *DNS domain* — see *Domains*.

Abstraction and Standardization

Layers of Abstraction

Conceptually, one can divide a complicated topic into different *layers of abstraction*. Where answering questions or solving problems in higher layers, relies on the existence of answers to questions or solutions to problems in lower layers. The trick is defining layers with boundaries between them that are as clean as possible, so that a person thinking at one level of abstraction is seldom forced to digress to a different level of abstraction to complete their thoughts or solve their problems. Instead, they can focus on just knowing *what* solutions and answers *other layers provide* without having to get caught up in the details.

Just as a biologist may know that life on earth would hardly be possible without Carbon, by virtue of its being able to form so many different kinds of complex molecular bonds with so many different kinds of atoms. But what exactly a molecular bond is and why life couldn't have just as easily formed based on silicon instead, they can feel free to leave to chemists or molecular biologists who prefer spending their time and mental energy at that level of abstraction. In turn, they may profit from the results and answers of quantum physicists who focus on an even lower layer of abstraction, who themselves are not overly concerned about how all their results will be used by those at higher levels.

And if at some future stage a particle physicist were to discover a new elementary particle, or some new properties of an existing one, though it may radically affect their research and view of reality, it may have little or no (disruptive) effect on that of the researcher who has spent the last 3 years observing primates in the Ngogo jungle.

Such a layered approach also proves very helpful when trying to understand or implement computer networks. The lower layers are the most fundamental, each higher layer making use of the solutions to problems and questions answered in the layer below, in such a way that there is as little as possible *spill over* from any layer into the layer above. Boundaries are chosen that cleanly (as possible) separate the concerns of adjacent layers. Each lower layer provides to the layer above it, clear succinct answers, and easy to use solutions, which hide the complexity and detailed nature of the processes involved in finding these answers or solutions.[6]

In order to function correctly, every layer depends on the functionality of all layers *below* it. But higher layers tend to offer a broader range of functionalities to a wider audience. In this sense, for example layer 1 can be considered the most *fundamental* layer of all, since without it, no network communication at all is possible. (*Think*: operating system (lower layer) and the apps (higher layer) you as a smartphone user actually use. Or a small handful of elementary particle types giving rise to possibly infinite variety of sentient species).

Models — network layers

Deciding on a *model* for such a layered abstraction is tantamount to choosing what layers to use and what to call them. In computer networking, there are two common models: The *OSI Model* (which is the older of the two and divides the subject matter into 7 layers) and the *TCP/IP Model* (which divides the *same* subject matter into 4 layers).[7]

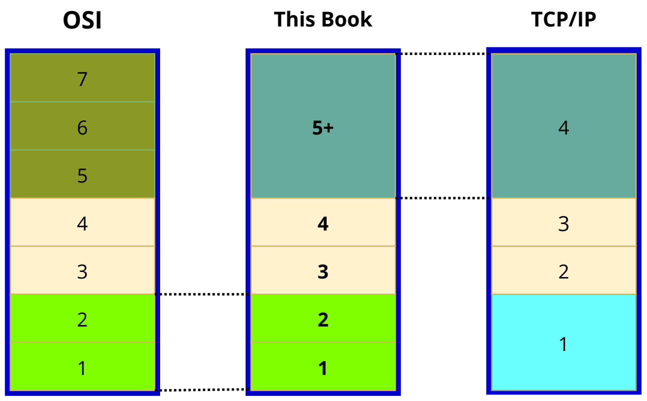

Figure 4. **Layer Models**. *The layers used in this book correspond to the lower two of the OSI model, and the top layer of the TCP/IP model (but referring to it as layer "5+" as a reminder that it corresponds to all of the OSI layers 5 and above) together with intermediate layers 2 and 3 that both the other models have in common.*

Layer-Specific Devices

It's not uncommon to see a node being referred to as a being an *layer n device*, or that it *operates at layer n*, or is *layer n enabled* (where *n* is any layer number in the *model* being used).

Switches and *Routers* will be introduced and explained later in this book. For now, however, note that you may often read that a *switch is a layer 2 device*, whereas a *router is a layer 3 device*. This means that a router has characteristics of, or exhibits behaviour that are the subject matter of, layer 3. Similarly, for a switch and layer 2. What's maybe not so obvious, but important to note, is that this (by definition of our layers of abstraction) means that a router is also both a layer 2 device and a layer 1 device. Similarly, a switch being a layer 2 device also makes it (by definition) a layer 1 device.

When we say a device *operates at* a specific layer, we generally mean this is the *highest* layer it can be considered operating at. Implied is however, that it also operates at all layers below this too.[8] So in general, if a node *is a layer n device* this means this is the *highest* layer it can be considered being in, or operating at, but by induction, that it is also a layer *n-1 device*, a layer *n-2 device*, … and a *layer 1 device*, for every layer number n in our model.[9]

Official Standards

Standards are formally established guidelines and specifications that ensure consistency, quality, and interoperability across different products and technologies. They provide a common framework that manufacturers and developers follow to create compatible and reliable solutions.

Consider a possible *standard* for Lego. Though maybe not so formally documented or complicated as that for networking technology, its purpose and value would be analogous. The standard might specify requirements for the studs (the raised circular *bumps* on the top of Lego pieces) and tubes (the hollow circular *holes* on the underside of Lego pieces that fit over the studs). It may mandate their size, positioning, and distribution. A product team designing a new Lego set must adhere to these specifications to ensure compatibility with existing sets.

Adhering to the specifications of the Lego standard involves extra effort and *compliance* on the part of the product team, but both users and manufacturers benefit. Users can freely combine pieces from different sets to create designs beyond what any single team envisioned. This compatibility makes Lego more appealing, leading to increased sales. Manufacturers benefit not only from more customers, but also from the ability to reuse specialized pieces created by other teams, reducing development costs and time to market. This creates a "win-win-win" cycle of value for users, manufacturers, and the ecosystem as a whole.

Standards in networking serve a similar purpose, but in a far more complex field. The layered approach to networking makes it possible to create specific standards for the part of the technology covered by specific layers (which by design are separated by clear borders). For instance, Ethernet and Wi-Fi are collections of standards covering Layer 1 and Layer 2. Any manufacturer wishing to sell standards-compliant equipment must meet the specifications, such as ensuring proper cable shielding, defined in these standards. This ensures quality, interoperability, and user trust. It’s also common to hear terms like Ethernet or Wi-Fi used to refer both to the technologies themselves and to the standards governing them.

Saying a node *operates at a given layer*, implicitly implies it operates at all lower layers also.

Don’t confuse the *TCP/IP model* discussed here with the so-called *TCP/IP protocols* used within the intermediate layers of *all models* discussed here.

Summary

- A computer network is composed of devices called *nodes* connected by *media*, through which they can pass information to one another
- The nodes are connected to the media, and thus the rest of the network, through their *ports* which act like doors through which the information passes in and out.
- Each node needs at least one port to join a network.
- A node may have multiple ports, in which case each port may be used to join a different network. Thus, a single node may simultaneously be part of multiple networks.
- On devices like a PC, ports are often recognizable as the antennas used for Wi-Fi, or the slots into which an Ethernet cable is inserted.
- The ports (along with any other possibly shared, hardware resources needed for pre- or post-processing the network information) are often located on a so-called Network Interface Card (NIC).
- The *infrastructure* of the network is composed of the media and devices whose role it is to make the network *function*. The network will not function (as well) without them. These devices are referred to as *service nodes* in this book.
- Devices like your smartphone, or pc which can optionally join an existing network, and whose role it is to use the network and benefit from its functionality, are called *user nodes* in this book.
- A port on a user node is also called an *interface*.
- The set of other interfaces an interface can intentionally send information to is called the *broadcast domain*.
- The set of other interfaces that information (unintentionally) reaches when an interface (intentionally) sends information to specific interfaces is called the *collision domain*.
- Ideally, the broadcast domain would be all (other) interfaces on the network, and the collision domain would be empty.

- Dividing larger complex subject matter into separate layers of abstraction facilitates understanding, and focusing attention (more) independently on one such part at a time.
- The lower layers typically encompass more fundamental considerations of the subject matter.
- Higher layers typically encompass more big-picture considerations.
- Higher layers benefit from clear answers and simpler solutions to complex questions, the details of which are *hidden* inside lower layers.
- The more independent the layers are, and the more clearly defined their boundaries, the better.
- A *model* for such layered abstractions is a choice of layers and a means of referring to them.
- The OSI and TCP/IP models are two common models for networks, and this book uses a

combination of both.

- Saying that a device is in or operates at a certain layer, generally means this is the *highest* layer it can be considered being a part of, or operating in. Such a device implicitly operates at all lower layers too.
- Official standards are formal guidelines and specifications that benefit all involved.
- Ethernet and Wi-Fi are two such standards.

Conclusion

In this chapter, you've established a solid foundation for understanding modern computer networks. We explored key concepts such as nodes, ports, and network infrastructure, clarifying how devices connect and communicate. You learned to distinguish between user nodes and service nodes, and gained insight into essential ideas like broadcast domains and collision domains, and standardization.

We also discussed how layers of abstraction can help manage network complexity by isolating different aspects of functionality, making networks easier to understand and troubleshoot. This book uses a custom model that blends elements of the OSI and TCP/IP models for a practical, comprehensive approach. As we explore each layer in the chapters that follow, you'll see how they work together to create functioning networks. Finally, we touched on the importance of official standards like Ethernet and Wi-Fi, which ensure compatibility and efficiency in real-world networks.

With these key principles in mind, you're ready to explore how data physically moves through networks — starting with the cables, signals, and local connections that form Layer 1 and Layer 2 in the next chapter. These are significant first steps toward one of the main goals of this book—building up an understanding of how the internet works—a network so large and complex it connects billions of devices worldwide.

[1] more precisely, one could see this division as being at the level of ports rather than at that of a node as a whole. However, it's very rare for a single device to be *both* a user *and* provider of a single network, so classification at the node level is acceptable in this context. Furthermore, as we shall later, the role a node plays in a network, depends on what layer we are talking about. *Routers* for example, are service nodes at layer 3, but user nodes at layer 2. Layers will be discussed more in *Layers of Abstraction* and layer 2 and 3 in detail in *Layer 2* and *Layer 3*

[2] we will see in the next chapter that such ports are called *Interfaces*

[3] Several of these other communications technologies are however very similar to Ethernet/Wi-Fi in their design

[4] Large broadcast domains increase vulnerability to Layer 2 attacks that can cause *DoS (Denial of Service)* - making resources unavailable to legitimate users). In *ARP spoofing*, fake address announcements reach all devices at once. *MAC flooding* forces switches to broadcast all traffic when their address tables overflow. In networks with multiple interconnected switches, *STP (Spanning Tree Protocol)* prevents loops. *STP manipulation* involves sending false messages to become the *root bridge* (the central switch others calculate paths through to prevent loops), potentially redirecting traffic or causing instability. These attacks exploit the shared nature of broadcast domains —one attacker can impact all connected devices, which is why modern networks favor smaller, segmented broadcast domains through *VLANs*.

[5] or an *interface*

[6] Imagine studying a complicated topic like the social dynamics in one of the most advanced species on our planet like primates, dolphins or elephants. One could think for example, about how the interactions in a group are motivated by the need to secure food and under what conditions a focus on individual success actually benefits each member in the long term more than a collective, cooperative approach (*think*: the first farmers in human history, who provided enough food for cities to eventually emerge, filled by many individuals freed from the need to directly forage or hunt for food. Instead, they are able to specialise in other ways to contribute to and shape their societies). This may in turn beg the question of what exactly is food. Or how and why certain kinds of food are more available in certain regions, or how it contributes to size difference or longevity of individuals, in turn playing a role in the social hierarchy of the respective groups. Regions with more food may lead to the prospering of larger groups with more complicated hierarchies, or individuals playing more significant roles for longer due to increased longevity. This in turn may lead one to think about ecology, biology, and even the biochemistry involved in favouring the growth of certain kinds of plants or digesting different kinds of food. This is a lot of ground to cover and not everyone studying social dynamics will be equally interested in all of these aspects, and all the associated scientific disciplines mentioned, at least in the same detail. But nonetheless, to be able to satisfactorily answer *any* question on the topic in detail requires a familiarity with a *diverse range* of complicated topics. Going from the bottom up instead, a physicist may argue that a full understanding of anything that exists (including food, primates and social structures) isn't possible without also understanding elementary particles. After all, elementary particles make up atoms, which make up molecules, which make up simple organisms, and food, which give rise to more complicated species and social dynamics. If elementary particles behaved fundamentally differently, this would give rise to different chemistry and different versions of what we call life. Most likely it would in fact lead to the existence of no life at all, since even tiny changes in the fundamental constants of nature would cause huge changes in the universe and make life as we know it unthinkable. However, if every social scientist first had to learn all about quantum physics before even being able to consider why chimpanzees tend to live in groups of less than 100, there would probably be very slow progress in answering such questions.

[7] The TCP model combines layers 1 and 2, and layers 5, 6 and 7 respectively of the OSI model into two single layers. The TCP model is generally regarded as the more practical of the two. It covers the same subject matter as the layers 5, 6 and 7 of the OSI model in its single layer 4. This is because it became apparent that the upper divisions originally chosen by the OSI model were not always as practical as intuitive as they might in modelling real world networking. Its merging of the two lower layers is mainly attributed to the boundary between them often being less than clear in practice. Each of the two models refers to the layers not only by number (as shown in *Figure 4*) but also by name. Unfortunately their naming differs significantly, and can sometimes seem less than fitting, which is why I shall in this book refer to them solely by number. In this book I shall effectively use the OSI model but condense the subject matter of layers 5-7 into a single layer as the TCP/IP model does - at the level of

detail in which this books treats layers 1 and 2, if feel distinguishing them adds clarity

[*8*] An ethologist may say that elephants they are studying are highly emotional and intelligent sentient beings that exhibit advanced social dynamic behaviour, whereas a rock in the desert does not. The elephants fall clearly into the layer of abstraction the zoologist is interested in, but the rock does not. A quantum physicist, on the other hand, may not differentiate between the two at all. Both elephants and rocks are made up of the same (type of) elementary particles, that obey the same fundamental laws of physics that the physicist is interested in, at this lower layer of abstraction. An elephant being a magnificent, socially advanced being does not mean it's not also composed of elementary particles that at a fundamental level behave the same as those in rock, and thus for the physicist both, as are many other objects, are collections of elementary particles. But the elephant is a collection of elementary particles that is also much more - a sentient social being. In an abstract layer model, we might say the elephant *operates in a higher layer* - that of sentient beings. It also operates, like the rock, in the lower layer of *collection of elementary particles*

[*9*] Technically, its individual ports of a device which operate at given layer, but since It's very uncommon (though not impossible) for the individual interfaces not to all operate at the same layer(s), one often speaks of the *layer of operation of the node* as a whole

2. Lower Layers — physical networks and LANs

In this chapter, we'll dive into the two most fundamental layers of networks: layer 1 and layer 2. You will learn that layer 1 is the only layer that *exists* between separate physical devices — all other layers operate within the devices themselves. When data needs to move from one device to another, layer 1 provides the actual physical path through electromagnetic signals in cables or air. Everything else — the processing, packaging, routing decisions — happens inside the devices at their respective layers. Layer 1 is the bridge between them.[1] You'll gain a clear understanding of the media that carry these signals between the nodes.

Layer 2 serves as the crucial interface that connects devices to the layer 1 bridge. With incoming signals, it receives bit streams from layer 1 and reconstructs the data into manageable *packets* (also known as *frames*). For outgoing signals, it decomposes the packets into bit streams for layer 1 to send them. Layer 2 also manages *addressing* — assigning unique identifiers to each device's network ports so that data can be directed to specific destinations and devices know which signals are meant for them.

We'll also look at the limitations of these lower layers and the LANs (Local Area Networks) they form, which will help you appreciate their crucial yet somewhat constrained role in larger networks like the internet.

Layer 1

All network communication involves the transmission of *electromagnetic waves* through a medium, as described in more detail in *Electromagnetic Waves and the Role of Media*. Layer 1 is concerned with the details of this transmission, as well as the conversion of the waves to and from a *stream of bits* at each end of the medium, as illustrated in *Figure 5*.

Media

The media used in modern networks are almost exclusively one of *Copper Cables*, *Fibre Optic cables* or *Air*[2]. See also *Lower Layer Landscape*. A medium *unconditionally* connects all ports attached to it, so that every packet sent by any port arrives at ever other port attached to said medium. The more ports attached, the more potential (collision) problems can occur, since it becomes more likely two different ports may try to send packets simultaneously, which then *collide*, resulting in communication difficulties. Such connections (of three or more ports) over a single medium can be at best *half-duplex*, i.e. the nodes involved try to observe a policy of *"transmit if quiet—back off on collision"*, thus endeavouring to recover from any collision problems. Examples are *Ethernet Bus* and *Wi-Fi*. The use of an Ethernet Bus (a medium) to connect more than 2 ports has been largely superseded by the use of a switch (an *intelligent* node that operates at layer 2 which will be discussed more in the next chapter) in modern networks.

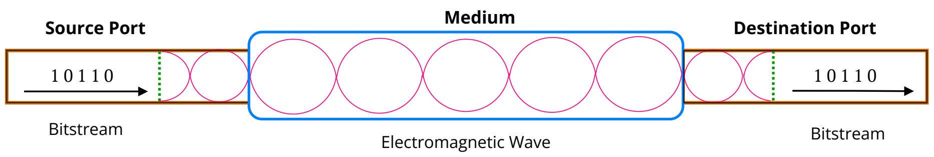

Figure 5. A ***Bitstream*** *coming out of a source port is encoded into (the modulation of) an* ***electromagnetic wave*** *which is transmitted to the other end of the medium, where the output bitstream is reconstructed (in the same order) as input to the destination port. The digital information the bits represent are encoded into analogue waves and vice versa*[3]*. Though not explicitly shown, the source and destination ports are each located on a node (as ports always are).*

Bits and Bitstreams

A *bit* is the smallest unit of information—with two possible values, which we often represent as 1 and 0 (*think*: light switch, which can be either *on* or *off*). Computers *naturally* use binary (two values) because their processing is built on transistors—tiny electronic switches that *either* are conducting electricity *or* not. Modern computer chips contain billions of these microscopic switches. All data that computers process and transmit—text, images, videos—is ultimately represented as long sequences of bits. See also *Decimal, Binary, and Hexadecimal*. By *stream* of bits is meant that the bits *pass* a certain point in a specific order and can only be processed one at a time in this order. *Think*: drops of water falling out of a tap. You can't *get* any drop until after those before it have fallen first. You can collect a cup full of water, but only after you have patiently waited for each drop to fall into the cup in the particular order in which they fell.

When using cables is feasible, an even better solution is to ensure that each cable connects exactly 2 ports. This creates a so-called *point-to-point* connection. Modern cables are generally *full-duplex* which means that 2 packets can travel in opposite directions simultaneously (like a road where cars going in each direction have their own separate *side* of the road), thus completely avoiding the collision problem, for the 2 ports connected. *Think*: you have a road with no intersections, with two sides (one for each direction of travel) and cars all travel at the same speed.[4]

Electromagnetic Waves and the Role of Media

Even though one often talks about seemingly different physical phenomena, such as *radio* waves, *infrared* radiation, *optical* signals in fibre cables, *voltages* in copper wires etc. being used to implement network communications, these can all be viewed as different aspects of a single fundamental physical process — the propagation of electromagnetic waves.

Visible light, radio waves and infra-red radiation are all forms of electromagnetic waves—they just have different frequencies, and our eyes happen to be able to *see* one but not the others. Voltages are just a consequence of the forces electric charges exert when they repel or attract each other, and the charged particles in a copper wire don't really move (far). They just oscillate back and forth, constraining the electromagnetic waves, which is what actually propagates down the wire. (This is equally true of electric power supply cables—the power company doesn't actually deliver any *new electrons* to your home along the electricity cables, but rather electromagnetic waves, which *contain* the energy).

The role of any media, is not so much to enable the waves to propagate (electromagnetic waves can, in fact, propagate most easily and rapidly through vacuum), but for the body of the medium to *not hinder* the propagation, and for the boundary (if any) of the medium to *keep the waves inside*. The cladding on an optic fibre keeps light (waves) inside through (total internal) reflection. And the free electrons oscillating back and forth on the surface of a copper cable play a similar role for the radio waves propagating along it.

A medium like a cable (e.g. used for Ethernet) can be likened to a water pipe. The interior of the pipe (which is ideally empty except maybe for some air) doesn't hinder the water from flowing though it, and the boundary (shell) of the pipe keeps the water inside. So, as much as possible of the water flowing into one end later flows out of the other end. When you turn on a tap connected to a garden hose, all the water that flows into the hose at the tap (assuming no leaks or defects) will reach the plant at the other end of the hose. Electrons bunch and spread in a wave-like pattern, akin to a compression wave in a spring where segments push and pull, creating a push-pull effect where the energy falls off with the square of distance between charges. This wavelike electromagnetic field, relies on the free electrons in the conductor to propagate it.

In the case of Wi-Fi, the medium (such as air) doesn't generally have a physical boundary to contain the waves, so they aren't confined. The waves emanate from their source and spread in all directions. This is more like using a sprinkler, which sprays water broadly across the garden. This approach is convenient because it waters multiple plants without needing to connect each one individually with a hose. However, it also means that only a fraction of the water reaches the plants you intend to water. A significant portion is wasted when it lands on rocks, paths, or even outside the garden, and sometimes on things like your shoes where it's unwanted. Wi-Fi offers the convenience of not requiring direct connections with cables, but it also has the disadvantage of wasting energy to generate waves that never reach useful destinations. This is what people typically call electromagnetic waves or radiation, moving freely without a physical guide.[5]

Nodes — analog-digital converters

Most layer 1 nodes transform incoming waves to bit streams, and vice versa, as illustrated in *Figure 5*. It's not always essential for their operation, however, that they do, i.e. they may operate in a purely analogue fashion working just with waves directly without any conversion to/from *bits*. What's important to note, however, is when a layer 1 service node does convert to/from bits as shown, that they do *not change the values of the bits* or in any way *make logical decisions* based on these values. The primary purpose of the transformation (if any) is to prevent signal degradation and enhance noise reduction, ensuring the integrity and reliability of the transmitted data. And this is often also the main purpose of the layer 1 service nodes.

Networks (Bitstreams)

A layer 1 network is composed of one or more user nodes, and zero or more service nodes, all joined by media. The service nodes and media together form the infrastructure of the layer 1 network. Such a layer 1 network is sometimes called a *physical network*, but, as with many terms in networking, it's unfortunately not used consistently. For this reason, I shall refer to it simply as a *layer 1 network* in this book. It forms a *single collision domain*, which is identical to its *single broadcast domain*.[6] This single collision domain, however, makes it unsuitable for the inclusion of more than a few nodes, since as previously mentioned, the more nodes in a single collision domain, the more likely collisions will occur. And more collisions means a less responsive and less reliable network. Furthermore, the fact that it only handles streams of bits rather than whole *packets* of data makes it somewhat cumbersome to use. After all, it's entire packets, and not just individual bits, that we actually want to send and receive. Which leads us to the topic of the next chapter, and up to a higher layer of abstraction.

Segment is sometimes used to describe one or more layer 1 nodes connected only by a single type of medium (but it's not uncommon, however, to observe its use in other contexts to mean something different — often some less clearly defined section of a network in the context of a higher layer).

Link is sometimes used to describe a segment where the medium is a single *cable* (*point-to-point*) and, more confusingly, in the context of layer 3, sometimes also as a synonym for a whole *Subnet*.

Layer 2

Layer 2 is concerned with *packets* (which, in the context of layer 2, are often also referred to as *frames*),[7] as well as their conversion to and from a *stream of bits*.

Interfaces

Ports on nodes which are not layer 2 infrastructure nodes are also called *interfaces*.[8] Only interfaces can serve as original sources or final destinations of layer 2 packages.

Hash and Checksum

A *hash* (or *checksum*) is a relatively small, effectively unique value that can be calculated for any particular body of data (including text). It's a bit like a *fingerprint* of the data in question. For each (finger of each) person, there is an (effectively) unique corresponding fingerprint. Similarly, for each body of data there is an (effectively) unique hash which can be derived from it, and can be used to *identify* the data. Changing any part of the data, no matter how small, will (almost always) result in a different hash value.[9]

Hashes are used in *Cryptography* to detect even deliberate tampering whereas more simple checksums (like *FCS* in the trailer of layer 2 packets) are designed to detect accidental corruption during transmission. Hashes are preferred when certainty of uniqueness is more important than speed and simplicity of calculation.

Packets (Frames)

A packet is the smallest independent *chunk* data gets handled as, before it's disassembled into a sequence of bits for layer 1 to transmit, or after it's assembled from a sequence of bits layer 1 has transmitted.

Assembly and Disassembly

Packet *assembly* is the construction of a packet from a sequence of bits (at a destination port) after transmission by layer 1. Packet *disassembly* is the deconstruction of a packet into a sequence of bits (at a source port) so layer 1 can transmit them. Here, the details (in what they refer to as the *MAC sublayer*) of the Wi-Fi and Ethernet standards differ significantly.

The process of assembly and disassembly may seem trivial, but consider an air traffic controller at a busy airport. Their (very non-trivial) job is to ensure that as many aeroplanes as possible can take off and land safely in limited airspace (and land), which means minimizing the separation between planes, but without them ever getting dangerously close. But aeroplanes (like weather and passengers) don't always behave as expected. A build up on the runway may require planes waiting to land to circle the airport temporarily, and of those waiting to depart to join a waiting queue somewhere. Similarly, network nodes work hard to ensure bits can *take off* and *land* as efficiently as possible, applying various techniques to adapt when network conditions aren't ideal, in order to keep performance high, while minimizing any danger of *bit collisions* and data loss. Additionally, every node has to agree on the exact location of the bits in a packet. Do they get filled with incoming bits from the top up, bottom down, left to right etc? Even if all the pixels of an image you send arrive at your friend's phone, you might not be very happy if every pixel doesn't end up there in the same place as it was on your phone.

Structure

A packet consists of a *payload* (sometimes also called *body*), together with a *header* and a *trailer*, as illustrated in *Figure 6*. The payload is the actual data the sender wants to send (*think*: *contents of a letter*). The sole purposes of the header and trailer are to store extra information used to help deliver the payload reliably (*think*: *address* and *stamp* on the *envelope*). The details of packet structure (in what is referred to as the *LLC sublayer*) are largely similar in both the Ethernet and Wi-Fi standards.

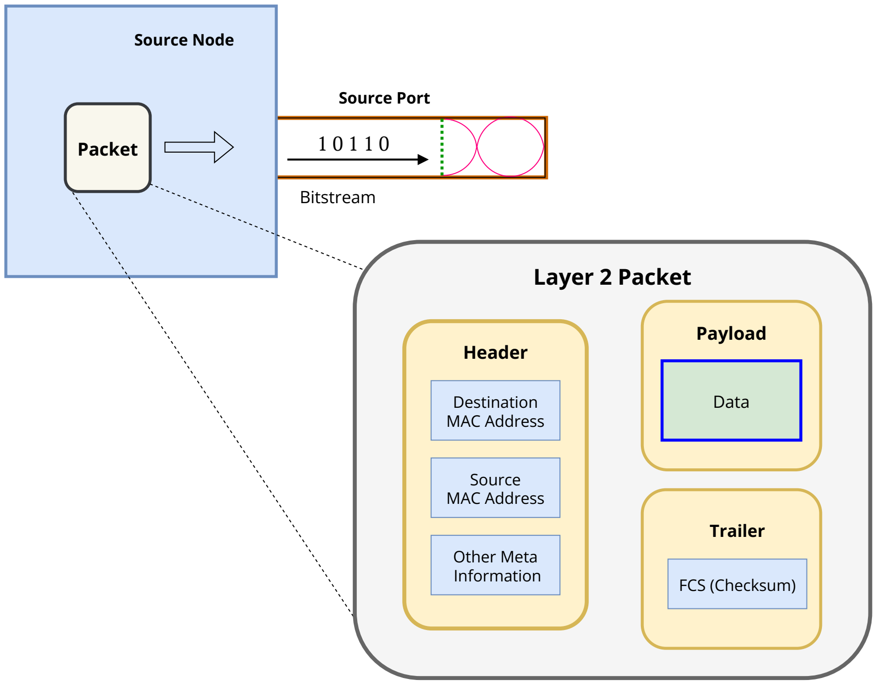

Figure 6. **Packet (Layer 2)** *and its structure showing the main fields*[10] *such as Destination Address, Source Address, and Data, in their respective groups (Header and Payload). Note that, though the image illustrates a package being disassembled at a source port, the assembly at a destination port is completely symmetrical.*

Header

- **Destination** Address. This identifies[11] the intended recipient interface(s).
- **Source** Address. This identifies uniquely the sending interface.
- **Other** Meta Information, used to facilitate reliable and efficient transmission[12].

Payload

This contains the actual information (content) the sender wishes to transmit. The payload is sometimes also referred to as the *body* of the packet[13]

Trailer

Contains the **Frame Check Sequence (FCS)**, a *checksum* which can be used by the receiver to check if the packet arrived correctly.[14]

MAC Address

The address used for an interface in layer 2 is known as a *MAC* (or *hardware*) *address*.[15] They are *burned* into the hardware (e.g. on the *NIC*) by the manufacturer and cannot subsequently be changed.[16]

The addresses in a layer 2 packet are always those of the interfaces originally sending and intended to ultimately receive the packet, respectively, and never those of ports on any service nodes the packet may pass through on the way. (Even though the manufacturer also burns such an address into the ports of all nodes, not knowing how they will later be used—i.e. as user or service nodes). Just like the addresses on a packet a courier is delivering is never that of any building they use to temporarily store or process the package on its journey, but those of its final destination and original sender.

Nodes — addressed packet processors

As mentioned in *Layer-Specific Devices*, layer 2 nodes (can) do everything layer 1 nodes (can) do. In particular, transforming bit streams into waves and vice versa at their ports, as illustrated in *Figure 5*. Unlike layer 1 nodes, however, they can (and often do) also actively *interpret* the incoming *bit patterns*, and make *logical decisions based on these values_about how to behave. A layer 1 service node will, for example, always resend the same bit pattern on its output port, that it has received on its input port. A layer 2 service node, however, may (depending on how it's configured) make a logical decision not to _forward* certain incoming bit patterns through its output port. It may just *drop* them instead.[17]

Only being able to access the bits of data one at a time in whatever order they happen to be arriving does not lend itself ideally to further processing. Imagine someone tells you over the phone, one point at a time, what the colour is of every pixel in an image you wish to analyse. You probably wouldn't find it conducive to do this analysis while they're still speaking. First you might construct the complete image, so you could *see* all pixels at once, or be able to shift your focus at will to various parts of the image in any order you wish in order to analyse it. Similarly, layer 2 nodes don't try to make any of their logical decisions based on the values of the bits in the stream while they are still incoming. Instead, they first construct whole packets from them, which they can subsequently analyse and further process much more effectively.

Switches

The most important type of layer 2 service node is a switch. Its main function is to perform the logical processing described above based on the address fields of the incoming packets. Each interface that should be added to a given layer 2 network is typically attached directly to a single port on the switch that acts like a *central switchboard* for that network. See *Figure 7*. Every time a packet arrives (by which I mean an em wave arrives, which gets transformed into a bitstream which gets assembled into a layer 2 packet), the switch analyses the destination address of the packet. It then forwards the package on its other port(s) (by which I mean it disassembles the layer 2 packet into a bitstream which gets output as an em wave) to which it *knows* the destination interface(s) is attached. This way, no other attached interfaces except the intended destinations get packets not intended for them.

This effectively maximizes the broadcast domain (any interface can send a packet to any other interface), while reducing the size of the collision domain (no node will receive any packets not intended for it, but it may still receive too many simultaneously, which collide with one another).

How does the switch *know* which interface is attached at which of its ports? Nodes can effectively join and leave a layer 2 network without *warning*, and with them any interfaces they attach to the switch. The answer is that the switch *learns* which interfaces are attached by *observing* not only the destination addresses of packets it receives, but also the *source address*. Every time a packet arrives at a given switch port, it updates its internal *list* of which interface is attached to which port. This internal list is called the switch's CAM Table. See *CAM Tables* for further details.

Sophisticated switches often use (intelligent) *buffering* to avoid multiple (simultaneous) packets arriving at a single destination interface, which might lead to congestion/collisions. The switch is careful not to forward any more than one packet at a time to a single interface (*think*: giving one car a head start on a road before letting another set off in the same direction from the same starting point). This, combined with modern full-duplex cables (see *Media*) for the connections between the interfaces and the switch, can completely eliminate the collision domain. No interface will receive any more than one packet at a time, and every such packet will have been intended for this interface. The full-duplex cables ensure that every interface can also send and receive simultaneously.

Of course, when exiting interfaces are removed or new ones added, or a switch is replaced, the performance of the network is less than optimal, as the switch gradually learns what interfaces are attached where. However, even in the (temporary) worst-case scenario where the CAM table is still completely empty, the network performance is still significantly better than it would be without the switch, thanks to features like buffering.

Imagine, by way of analogy, a switch to be like an efficient *airport shuttle bus system*. It gets passengers from their respective gates to the parked aircraft waiting outside. The passengers don't have to concern themselves about where their aeroplane is parked or how to get there. Without it, the passengers would have to walk out the gate and *somehow* find their way to their respective aircraft, possibly wandering around, getting lost and colliding with one another or ground crew vehicles in the process. And the bigger the airport (more gates, more waiting aircraft, more collisions, and longer distances to walk), the more necessary the shuttle bus system becomes.

A network *switch* is more like a regular *switchboard* than a (single) *regular* switch (like a light switch), since it internally contains multiple regular switches, each of which can be individually turned on or off, depending on what should be connected or disconnected.

In the older days, a telephone switchboard was managed by an *operator*. When a call came in, a light would blink next to the caller's line. The operator would then ask, "Number, please?" (analogous to the switch learning the source MAC address from the incoming frame and associating it with the port it arrived on). Once the caller provided the number of the person they wanted to reach (the destination MAC address), the operator would manually plug a cable from the caller's line into the recipient's line, establishing a direct, dedicated connection between those two specific parties.

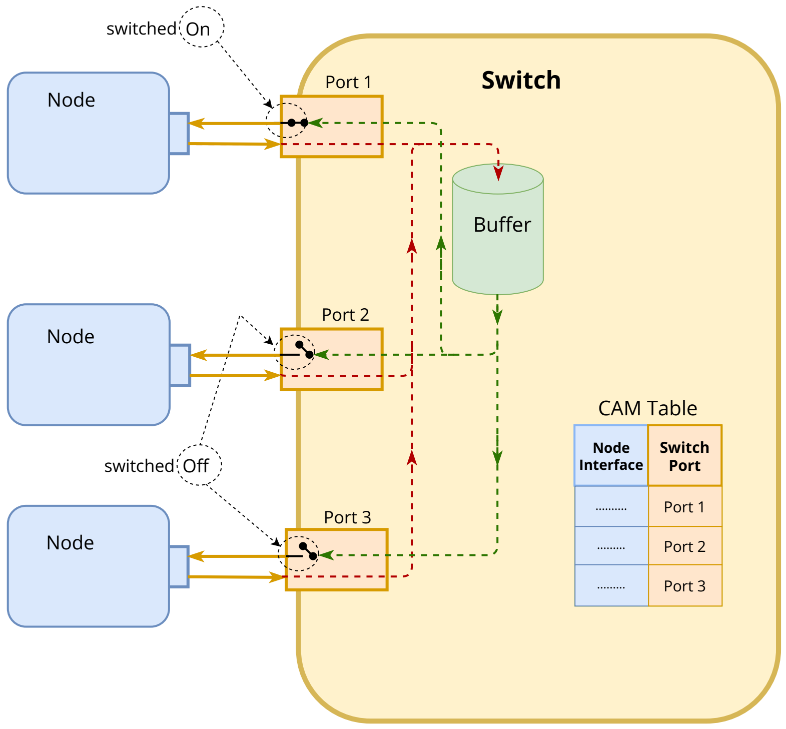

Figure 7. **A Switch** *(which contains multiple traditional "on-off switches") is a critical layer 2 service node which forms the core of most modern layer 2 networks (LANs). It can efficiently connect a large number of interfaces (for simplicity, just 3 are shown here). If multiple nodes try to send simultaneously, it queues the packets in a buffer, and forwards them sequentially. It also ensures that only intended recipients receive any packet by switching the respective outputs on/off depending on the destination address in the packet.*

Network Topology

Many networking introductions start with "*network topologies*" such as *ring*, *star*, and *mesh* as foundational concepts. Emphasizing these terms made more sense when network ports could be both *addressable* (i.e. interfaces) themselves and could serve as *pass-through* points for other ports. They are less relevant in modern networks where there is a clear separation between infrastructure and user nodes, as illustrated in *Figure 2*. Topology is still relevant for *infrastructure design*—how the infrastructure nodes are connected strongly influences network properties (e.g. performance and reliability), but this is specifically the topology of the infrastructure of the network—not that of the network as a whole. The simple ring/star/mesh labels are not very helpful for big-picture understanding of modern networks, where pure-infrastructure devices like *switches* connect interfaces that are purely users of the network.

CAM Tables

As mentioned above, a switch *remembers* which interface (of a user node) is attached at which of the switch's ports, by *noting* the information in its so-called *CAM (Content Addressable Memory)* table. The CAM table of a switch maps the address of each (connected) interface (address) to a port on the switch. When an incoming packet arrives at a port of the switch, the switch looks up the destination address of the packet in its CAM table. This tells it where the destination interface(s) of is connected to the switch. It then forwards the packet out on the corresponding port. (Switches may also use additional columns in their CAM table to note other information such as which *VLAN*, if any, the interface belongs to, and when the entry was last updated). The CAM table of a switch is typically empty to begin with (when the network is first set up), and it builds up the CAM table incrementally over time.

Every time an incoming packet arrives, the switch uses the **source** address of the packet to **update** its CAM table. If it already has an entry mapping the source interface (address) to the port it received the packet on, it updates the entry if necessary (which may happen if interfaces get reconnected to different ports). If not, it adds a corresponding entry to the CAM table.

When a switch receives a packet for whose **destination** interface (address) it does not have an entry in its (CAM) table for, it floods the packet - i.e. resend the packet on every port to which an interface is connected (within the respective broadcast domain) except the one on which the packet arrived.

So in summary, for each packet arriving at a particular switch port, the switch makes a *note* (in its CAM table) in order to *remember* that the interface with this *source address* is attached to this port. And to figure out which of the switch's ports it should forward the packet in question out of, so that it reaches the (user node interface corresponding to the) *destination address* in the same packet, it looks for a *note* it previously made (when said port received an earlier packet from the interface that is now the destination of the current packet).

Networks (LANs)

The infrastructure of two or more layer 1 networks can be joined by layer 2 service nodes to form the infrastructure of a combined layer 2 network, also known as a *LAN (Local Area Network)*. As mentioned in *Switches*, an optimal (and typical) configuration is when all the layer 1 infrastructures are single full-duplex point-to-point connections between the interfaces and ports on a switch. A single switch per LAN is the most common and simple to maintain configuration, but it's not impossible to have multiple switches, with one being the primary switch *in charge* of the others, to which it delegates some of the work.

To say that a switch is an important type of layer 2 service node is a bit of an understatement.

They are more like the *rock stars* of layer 2, and sophisticated (and correspondingly expensive) switches can connect thousands of interfaces in a highly optimized fashion, with advanced features like the buffering mentioned above. Many organizations who value performance and security invest significant amounts of money in correspondingly sophisticated switches.

A LAN, like a layer 1 network, forms a *single broadcast domain*, but, unlike a layer 2 network, it forms multiple small collision domains, or more often than not, *no collision domains* at all. This makes it able to accommodate many more interfaces much more reliably than a layer 1 network, and the switch with its conditional forwarding plays a key role here.

A LAN is often *defined* as a set of nodes managed by a single organization and/or restricted to a limited geographical area. While these are typical characteristics (as the name suggests) of existing LANs, they are not defining characteristics. Any more than say, *likes to swim* and *lives in the sea* are defining characteristics of a fish. It's very possible, for example, to create a LAN that is neither local nor managed by a single organization.[18] A defining characteristic of a LAN is the (exclusive) use of MAC addresses — every source interface must know the (permanent) *MAC address* of the intended destination address, before it can send that interface a packet. We will see in later chapters how this distinguishes a LAN from networks of *Layer 3* and above. The application of *conditional logic based on these MAC addresses* (leading to a reduction or elimination of collision domains, by layer service 2 nodes like a switch), distinguishes a LAN from layer 1 networks.

LAN

A layer 2 network. It connects ports on nodes which can communicate at layer 2, by exchanging packets of data using layer 2 (*MAC*) addresses. Most LANs are either:

- *Point-to-Point* : Two nodes connected directly by a medium (such as an Ethernet cable). See also *Link*.
- *Switch based*: multiple ports connected (by media) through a switch.

Despite LAN being an abbreviation for *Local Area Network* (and some definitions to the contrary), a LAN need not be restricted to a local area.

Destination Address Types

A destination address is always one of three types:

Unicast

The address is that of a specific existing interface[19]. When an interface receives a packet with its address in the destination of the header, it knows that the packet is meant specifically for it (alone). When a switch receives such a packet on one of its ports, it will forward it on the port connected to the recipient interface.

Multicast

The address is within a reserved range, representing a group of interfaces configured to know they belong to that group. When an interface receives a packet with a value in the destination of the header, that corresponds to a group the receiving interface belongs to, it knows that the packet is meant for it (as well as all other interfaces in the group). When a switch[20] receives such a packet on one of its ports, it will forward it out of each of its ports where an interface in said group is connected (excluding the one it received it on).

Broadcast

An address representing all connected interfaces (usually all **1s**). When an interface receives a packet with this value in the destination of the header, it knows that the packet is meant for it and all other interfaces in the network. When a switch receives such a packet on one of its ports, it will forward it out of each of its ports where an interface is connected (excluding the one it received it on).

Lower Layer Landscape

Three technologies dominate the landscape at layer 2: *Ethernet*, *Wi-Fi* & *Cellular*. Whereas at layer 1 *Fibre Optics*, *Wi-Fi* & *Cellular* are becoming increasingly popular. See *Figure 8*. Note that Ethernet, Wi-Fi, and Cellular [21] are *Official Standards* covering all aspects of *both* layer 1 *and* 2.

Some of these technologies lend themselves to pick-and-mix from different layers, e.g. layer 2 Ethernet (*think*: packets) can be used on top of layer 1 Ethernet or layer 1 fibre optics (*think*: type of cables). Whereas some (e.g. layer 2 Wi-Fi and cellular) can be combined only with their dedicated layer 1.[22]

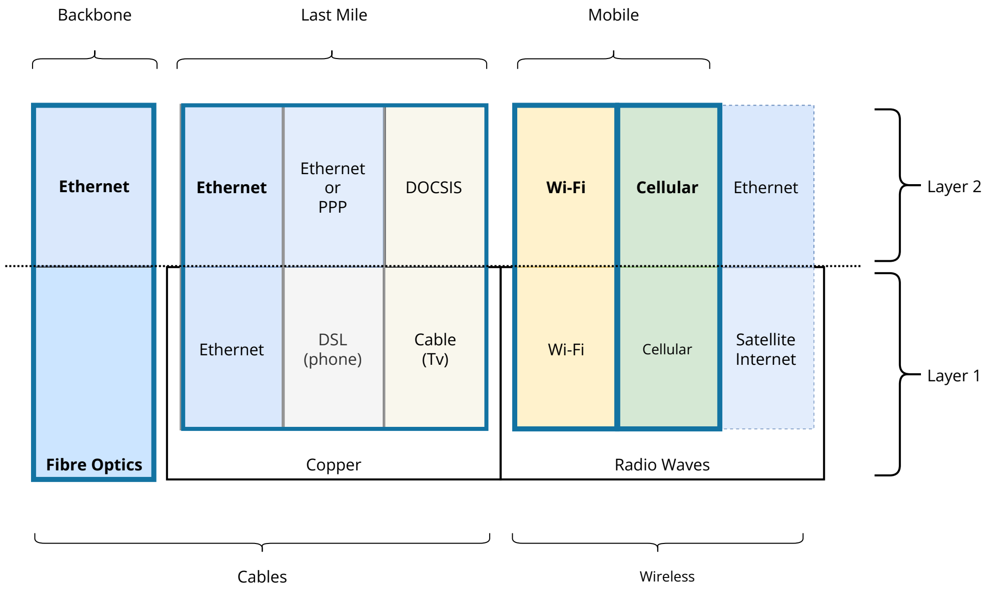

Figure 8. Layer 1 signals travel either through cables or wirelessly. The internet backbone (much of which is under the ocean) uses fibre optics, while the "last mile" to homes and businesses still commonly uses copper cables (though fibre is increasingly replacing them), and to mobile devices usually wireless.

The history of networking reveals an interesting reversal. Early internet access *hijacked* existing infrastructure never designed for data. Dial-up modems (1990s) repurposed telephone lines — unshielded copper wires designed for 4 kHz voice — using mostly analogue circuitry internally to convert bits into audio tones that could travel over these voice lines at 56 kbps. *ISDN (Integrated Services Digital Network)* improved on this with digital signal processors and dedicated channels for voice and data, eliminating the *can't use phone and internet simultaneously* problem. *DSL (Digital Subscriber Line)* pushed further by using digital processing to split the telephone line into multiple frequency channels — voice on low frequencies, multiple data channels on high frequencies — reaching up to 1 Gbps+ over the same thin, unshielded copper originally installed for analogue phones.[23] Similarly, cable internet using *DOCSIS (Data Over Cable Service Interface Specification)* hijacked unused frequency bands in TV coaxial cables, carving out channels for bidirectional data alongside television broadcasts.

However, purpose-built networking technologies (e.g. Ethernet, Wi-Fi, Fibre Optics) evolved so rapidly that the trend completely reversed. Instead of forcing data through infrastructure built for voice and television, we now eliminate that legacy infrastructure entirely and run everything — including phone calls and TV — as IP packets over data networks. Modern *phone service* is just *VoIP (Voice over IP)*, and television is streaming video over the same fibre that carries email and web traffic. The global telephone system, once entirely separate with circuit-switched connections,[24] has been absorbed into the packet-switched internet. This reversal is driven by physics and economics: telephone-grade copper (unshielded, thin gauge) hits hard limits on speed and distance, while modern Ethernet cables (shielded, twisted pairs) and especially fibre optic cables can carry gigabits or terabits of data reliably. Maintaining separate networks for voice, TV, and data no longer makes sense when a single fibre can handle everything.

Note that a side effect of reversing the trend (from *hijacking* phone and TV lines for internet access to running voice and TV over purpose-built data networks) is the corresponding shift from circuit-switched to packet-switched systems for all communications, including legacy voice and broadcast services.

A single LAN can, in principle, use a combination[25] of layer 1 and 2 standards (as long as the switch managing the LAN supports the technologies in question). This is how you can connect both your mobile phone (via Wi-Fi) and your PC (via Ethernet cable or Wi-Fi) to the internet through your *home router*.

In the early stages of the internet, existing infrastructure like phone lines was *hijacked* to transport its data. Now the internet has a dedicated infrastructure of its own, and the opposite is happening—phone calls, TV etc. are increasingly using the infrastructure created for the internet.

VLANs

It's not uncommon for an organization to use a single switch (or combination thereof) to create multiple LANs. Each of these smaller LANs is then called a *VLAN (Virtual LAN)*. It's important to note, however, that each such VLAN is no different to a LAN created with its own separate switch to start with, from the point of view of the interfaces on the VLAN[26]. A VLAN is itself a (new) LAN and the original LAN then no longer *exists* in the sense that each VLAN has its own separate broadcast domain independent of the others. The advantage of dividing up a single (original) LAN like this, as opposed to creating them with separate switches to start with, is improved flexibility and performance.[27]

Ethernet (IEEE 802.3) and *Wi-Fi* (IEEE 802.11) are official *standards* developed by *IEEE* (*Institute of Electrical and Electronics Engineers*), while Cellular standards such as 3G, 4G/LTE, 5G (and 6G) are developed by *3GPP (3rd Generation Partnership Project)*. All three govern all aspects of both layer 1 and layer 2 (see *Official Standards*). Furthermore, the standards for *Layer 3* and above are developed by *IETF* (*Internet Engineering Task Force*). While distinct in their primary focus, the IEEE, IETF and 3GPP work closely together.

> Twisted-pair copper cables, such as *Cat6* or *Cat7*, are also simply known as *Ethernet cables* — since they were originally the only kind supported by layer 2 Ethernet. However, layer 2 Ethernet was later generalized to support other cable types, including fibre optic, as illustrated in *Figure 8*. So, it's not always clear from the context whether *Ethernet cable* is referring specifically to the (traditional) copper cables or Ethernet cables in general.

Limitations

Several shortcomings become apparent when considering scaling a layer 2 network to include more interfaces distributed more widely (let alone across the whole planet).

MAC addresses are inflexible

The MAC addresses used at layer 2 are permanent — the address of a given interface can never be changed. They burned into the hardware (e.g. NIC) during assembly, before it is even sold (let alone its usage determined). When you replace an interface (even using a NIC that is the same model from the same manufacturer), the address changes, requiring all other nodes on the network to relearn it

It's not possible to group interfaces in any sensible way by address. The addresses are effectively *random numbers*. For example, there's no inherent way to identify addresses by **role** (e.g. *the printers*) or **region** (e.g. *in our office*). Any address could belong to any device or location.

No Address Discovery

There's no built-in way for an interface to ask *who's there?* (i.e., discover the addresses of other listening interfaces). The nodes have to *somehow* figure out what other interfaces are connected and with which addresses, and/or be manually configured (and updated) by *someone* with the information.

Limits of a Single Switch

A single LAN can grow using a single switch (or multiple switches, with one being the primary switch *in charge* of the others), and such a switch can handle many (1000+) interfaces well, but every switch reaches its limit at some stage. Even with multiple switches servicing a single LAN, coordination overhead grows with the number of interfaces.

Limits of Multiple Combined Switches

Instead of growing a single big LAN with a single (primary) switch, one could try letting separate (smaller) LANs communicate by enabling switches to send packets to each other.

One could envisage a switch theoretically forwarding a packet (arriving from an external switch) to the destination interface, if the interface in question is in the LAN managed by the recipient switch. And if it's not, forwarding the packet instead further along, one switch at a time, until it ultimately arrives at the intended destination LAN, and the switch that manages the destination interface.

This might seem feasible when the number of switches involved, and the number of paths connecting them, is small. But imagine a global collection of switches, all connected directly or indirectly in a maze of complicated paths. A big challenge would then be determining the *best path* for a packet to take across multiple switches. And how would the switches communicate these paths to each other? See *Figure 9* below.

Since MAC addresses are *random*, a switch would need to store the chosen path (or at least the next *hop* along that path) for every possible destination, which scales very poorly (linearly with the number of connected interfaces).

Packets would need to include a further address (in addition to the ultimate destination)— that of the next switch on the path to this destination. This is not something switches, and layer 2 in general, were designed to accommodate, and it's not clear how to *extend* them to do so.

Security

Layer 2 traffic is generally insecure. There is no default encryption of Layer 2 packets or way for interfaces to verify (*authenticate*) each other. Switches just *learn* which interfaces are connected to what port, by *observing the source* addresses of packets they forward to the intended recipient. Any interface can *pretend* to have any address (so-called *MAC spoofing*) by simply writing a different source address into the header of outgoing packets.

An interface could be replaced by a malicious one at any time (without the switch or other interfaces being aware), potentially *intercepting packets* intended for the original device.[28] Chapter *5. Security*, will go into more details.

These vulnerabilities highlight why using public Wi-Fi is fundamentally insecure. On any network you join, all devices connected can communicate at Layer 2 with your device, exposing it to potential attacks such as *MAC spoofing* or *packet interception*. Using a *VPN*, for example, will *encrypt* traffic at higher layers but doesn't directly address layer 2 vulnerabilities.

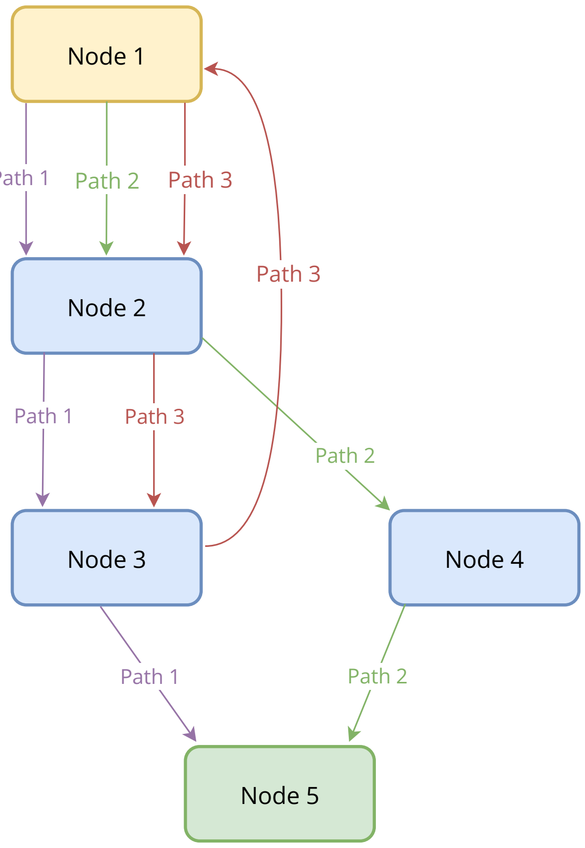

Figure 9. **Best Path Problem**. *A web of nodes (labelled Node 1-5), and some possible paths connecting even this small number of nodes. Note that some paths (e.g. Path 3) from the initial source (Node 1) may never lead to the final intended destination (Node 5), but instead to a (possibly endless) loop. Somehow, for each packet from a given initial source to a final destination node, the best of many possible paths needs to be determined, and each node on this path needs to know to which of the nodes connected to it, it should forward the packet in question to next.*[29]

Summary

- Layer 1 is concerned with the transmission of electromagnetic waves, as well as their conversion to and from a stream of bits.
- Layer 1 service nodes do not change the values of the bits or in any way make logical decisions based on these values.
- Layer 1 networks form a single collision domain, which is identical to its single broadcast domain.
- This single collision domain places severe restrictions on the size of Layer 1 networks, which is typically limited to a few interfaces.

- Layer 2 is concerned with *packets* (which in the context of layer 2 are often also referred to as *frames*) as well as their conversion to and from a stream of bits.
- A packet is the smallest independent chunk data gets handled as, before it's disassembled into a sequence of bits for layer 1 to transmit, or after it's assembled from a sequence of bits layer 1 has transmitted.
- Packet is structured as a payload, together with a header and a trailer.
- The payload is the actual data the sender wants to send.
- The header and trailer contain fields containing extra information used to help deliver the payload reliably.
- The source address identifies which interface sent the packet. The destination address identifies which interface(s) should receive the packet.
- Unlike nodes which only operate at layer 1, layer 2 service nodes can (and often do) actively interpret the incoming bit patterns, and make logical decisions about how to behave, based on these values.
- A switch is the most important layer 2 service node.
- It can efficiently connect a very large number of interfaces into a reliable layer 2 network, frequently called a LAN.
- Ethernet, Wi-Fi and Cellular are technologies commonly used at both layer 1 and 2. Fibre Optics are commonly used (only) at layer 1. Each are governed by official standards to ensure reliable quality and interoperability for everyone involved.

- Despite the huge improvements over layer 1, layer 2 networks still have their limits.
- A layer 2 network can contain thousands of interfaces, but this falls far short of what's needed to connect something as large as the whole internet.
- One of the challenges faced when trying to combine multiple layer 2 networks, is finding the best path through them for any given packet.

Conclusion

You now understand how the lower layers form the foundation of network communication. Layer 1 handles the transmission of bits as electromagnetic signals across copper, fibre, or wireless media, while layer 2 packages these bits into frames and manages delivery between directly connected devices using MAC addresses.

While these layers excel at local communication within small to medium-sized LANs, they face critical limitations. The inflexible hardware-bound addressing, limited error recovery, and minimal security make it impossible to scale LANs to anything approaching internet scale.

Chapter 3 introduces layers 3 and 4, which overcome these constraints. Layer 3 enables us to join multiple LANs into interconnected networks that can scale globally — forming the basis for the internet itself. You'll discover how IP addresses provide the flexible, hierarchical addressing scheme that makes this possible, allowing packets to be routed across countless intermediate networks to reach any destination worldwide.

Layer 4 adds interface sharing through port numbers. While layer 3 delivers packets between network interfaces (like delivering mail between buildings), layer 4 lets multiple applications share the same interface—adding apartment numbers to our postal analogy. This enables a single device to run hundreds of network services simultaneously. You'll also explore how TCP, one specific layer 4 protocol, adds reliability and connection management on top of this basic multiplexing function, while UDP provides a simpler alternative when these features aren't needed.

[1] While *virtual networks and software-defined systems* can operate without traditional layer 1 connections, understanding layer 1 remains fundamental to grasping how most real-world networks function.

[2] more accurately, any space, including one filled by a vacuum, can act as a medium for electromagnetic waves - see also *Electromagnetic Waves and the Role of Media*

[3] readers with a background in physics may feel tempted to appeal to the principle of *wave-particle duality* [en.wikipedia.org/wiki/Wave%E2%80%93particle_duality] here, the photon being the particle counterpart to the wave, but further elaboration would probably be better left to a book on quantum physics. Note also that the illustration of the wave shown (like a standing wave on a guitar string) is not very accurate. The actual wave is much more complicated and is generally composed of a base wave (the 'carrier') on top of which multiple other waves of different wavelengths are superimposed. The exact wavelengths and amplitudes involved and the way they are superimposed are what encodes the information carried by the complete wave.

[4] The physical media (part of Layer 1) represents a complex engineering domain where multiple challenges must be solved simultaneously — much like road construction. Roads must be smooth, weather-resistant, and support bridges. Similarly, network media must handle electrical interference, signal degradation over distance, physical durability, and bandwidth requirements. Not all media solutions are created equal: Cat5e cable (*think*: a local road) supports 1 Gbps up to 100 meters, while Cat6 (*think*: a highway) handles 10 Gbps but only up to 55 meters at that speed. Cat6a extends this to 100 meters but requires thicker, less flexible cables. Fiber optic cables (*think*: high-speed rail lines) can carry signals for kilometers without degradation but require specialized, expensive termination equipment. Each medium represents different trade-offs between cost, performance, distance limitations, and installation complexity — choosing the right one requires understanding both current needs and future growth, just as city planners must decide whether to build a two-lane road or a six-lane highway.

[5] imagine an electron oscillating to and from the end of a conductor in the direction of the cable. As it gets closer to the end of the wire, the repulsive Coulomb force on the free electrons at the end of the wire increases, nudging them down the wire a little. In turn, this increases the repulsive force on the electrons a little further down, and so on. When the first electron moves away from the start of the wire the opposite happens — this is very much like pushing and pulling the end of a spring causing compression waves to propagate down it. And just as no part of the spring moves very far, none of the free electrons in the wire move very far — they just oscillate back and forth harmonically like the first electron. The wavelike electromagnetic propagation *in* the conductor is not normally called an electromagnetic wave, but it is an electromagnetic field propagating as a wave. The magnitude of the field inside the conductor decreases fast — with the square of distance from the source — which is why neighbouring free electrons are needed to sustain it. This very same oscillation of the electron also causes an *independent* electromagnetic wave to be emitted perpendicular to the length of the conductor. Unlike the wave *in* the conductor, it needs no medium with free electrons to propagate it — no neighbouring electrons to feel repelled, and pass on this repulsive force. This wave, outside the conductor and moving axially away from it, is what’s normally called an electromagnetic wave (or far-field radiation) and its magnitude decreases linearly in proportion to its distance from the source — here the oscillating electron. The total electromagnetic field generated by a moving charge is called a Liénard-Wiechert Field. Interested readers familiar with Python may wish to have a look at the wonderful simulations by Matthew Filipovich as part of his *pycharge* [github.com/MatthewFilipovich/pycharge] project available at github.com/MatthewFilipovich/pycharge.

[6] *Broadcast and Collision Domains* defines the collision/broadcast domain of an individual port. The collision/broadcast domain of the whole network can then be understood to be the union of the set of collision/broadcast domains of the individual ports in the network.

[7] This book shall tend to prefer the general term "*packet*" unless using different context-specific terms for a single concept adds clarity. Another common general term for packet is "*Protocol Data Unit (PDU)*". The reader is advised that when other sources refer to "frames" they are simply referring to packets (or PDUs) of layer 2

[8] *Switches* are layer 2 infrastructure nodes, so their ports are *not* called interfaces. *Routers* (though layer 3 infrastructure) are not layer 2 infrastructure nodes — their ports *are* referred to as interfaces. Layers 1 and 2 will be covered in *2. Lower Layers — physical networks and LANs*. Only interfaces can serve as original sources or final destinations of layer 2 packages

[9] Strictly speaking, a hash is more complicated to calculate than a checksum, and produces a longer value. As a result, it's much more certain to be unique (*think*: much higher resolution fingerprint) - the *effective* and *almost always* qualifiers above are much more likely to be superfluous.

[10] Note that some fields are omitted, which I feel are details that distract more from a fundamental understanding of the concepts presented here, than add to it. And Wi-Fi defines/uses more such fields than Ethernet, in particular because in the case of Wi-Fi, the media are much less reliable, and thus need more checking. Ethernet cables, for example, are well shielded and much less susceptible to interference or loss than a Wi-Fi signal. Most of these additional fields are part of what's shown as "Other Meta Information"

[11] not necessarily uniquely - see *Destination Address Types*

[12] What fields are available depends on what layer 2 standard (e.g. *Ethernet*, *Wi-Fi* or *Fibre Optics*) is used

[13] may have padding added to meet minimum packet size requirements

[14] the sender calculates a checksum based on the contents of (the rest of) the packet it's sending, and appends this value to the packet. The receiver performs the same calculation based on the contents of (the rest of) the packet it receives. If the value calculated by the receiver is equal to the value calculated and appended by the sender, it is presumed that the packet contents are equal, since any change (no matter how slight) in packet content would lead to a different checksum value. If not, the frame is discarded by the receiver as invalid. Higher layers e.g. *TCP* in *Layer 4*—may then request the frame to be resent

[15] A *MAC (media access control)* address is usually represented as six groups of two hexadecimal digits, separated by colons (:) or hyphens (-). For example: `00:1A:2B:3C:4D:5E` or `00-1A-2B-3C-4D-5E`

[16] Each manufacturer is typically allocated a range of addresses by the *IEEE (Institute of Electrical and Electronics Engineers)*, the first 24 (of 48) bits of which identify the manufacturer itself. In *Virtual Devices and Interfaces* we'll discuss virtual/software interfaces. While there's no physical *burning* involved for these, they still have MAC addresses, and changing them is conceptually equivalent to replacing the interface. Note that while software can be used to make an interface *appear* to use a different MAC address (called *MAC spoofing*), this doesn't actually change it.

[17] For layer 2 service nodes, *dropping* and *forwarding* are the only two options, since service nodes are never the intended final destination for layer 2 packets. However, if this were not a service node but rather an user node, it could instead *consume* the packet (if the interface was its intended final destination) — meaning it would process the packet's contents for its own use. To the outside world, consuming looks the same as dropping: the packet isn't forwarded anywhere else.

[18] See, for example, *Overlay Networks*

[19] a specific value (usually all **0s**) is reserved to mean *invalid* or *address unknown*. It is, however, rarely needed

[20] technically, a switch must be (and most modern switches are) capable of so-called *IGMP snooping* to handle multicast correctly. Other less capable switches treat multicast packets as broadcast

[21] The cellular standard is specified by an organization called *3GPP (3rd Generation Partnership Project)*. Like Wi-Fi, it has progressed through various generations, e.g *3G* → *4G (LTE)*→ *5G* → the emerging *6G*. And like Wi-Fi it covers all aspects of layer 1 and 2. 3GPP subdivides layer 2 into three separate sublayers (called *MAC*, *RLC*, *PDCP*), and defines protocols for each.

[22] 1). Ethernet is defined by IEEE 802.3; it specifies both the physical wiring/signals (Layer 1) and link sharing/MAC addressing (Layer 2). Common types: 10/100/1G/2.5G/5G/10G/800G/1.6TBASE-T (twisted-pair copper cables, e.g., Cat5e, Cat6, Cat7) and 1000BASE-LX/10GBASE-LR (fibre); VLANs via IEEE 802.1Q. 2). Wi-Fi is defined by IEEE 802.11. Versions include: 802.11a, 802.11b, 802.11g, 802.11n (Wi-Fi 4), 802.11ac (Wi-Fi 5), 802.11ax (Wi-Fi 6/6E), and 802.11be (Wi-Fi 7), operating in 2.4, 5 and 6 GHz frequency bands. Lower frequency bands have better range, but less bandwidth (more congested). Wi-Fi security is defined by WPA2/WPA3, which is independent of the Wi-Fi version used—you can use WPA3 on older Wi-Fi standards if the hardware supports it, and newer Wi-Fi versions maintain backward compatibility with older security protocols for legacy device support. 5). Between cell towers and beyond (called the *backhaul*), cellular networks switch to Ethernet at layer 2 6). DSL uses copper phone

lines. Speeds up to 1Gbps+ with ADSL/VDSL2/G.fast. Layer 2 runs PPPoE, PPPoA, ATM, or plain Ethernet. 7). Cable internet uses coaxial cables with DOCSIS 3.1/4.0 at layer 2. Your modem outputs Ethernet. 8). Fibre to the home (FTTH) uses GPON/XGS-PON/50G-PON standards. The ONT box in your home converts fibre signals to Ethernet. 9). Satellite internet uses radio waves in Ku-band (12-18 GHz) or Ka-band (26-40 GHz). Layer 2 runs DVB-S2/S2X/RCS2 or proprietary protocols like Starlink. Your satellite modem outputs Ethernet. 10). *Power Line Communication (PLC)* allows data transmission over electric power lines. It's primarily used in home automation systems (like smart meters) and as an alternative to Wi-Fi for extending network coverage within buildings. While convenient for certain use cases, its adoption remains limited compared to dedicated networking technologies.

[*23*] Note that the marketing emphasis the "digitalization" of internet access—it was only the circuitry in the (modem) devices themselves that made a jump from analogue to digital with ISDN and DSL—before and after layer 1 remained purely analogue (think: electromagnetic waves), and layer 2 purely digital (think: packets of bits)

[*24*] Circuit-switching creates a dedicated, fixed-bandwidth path between endpoints (like a traditional phone call, where the line is reserved even during silence, wasting resources). *Think*: you get a road to yourself. Packet-switching breaks data into small packets sent independently over shared paths, reassembled at the destination — more efficient, flexible, and scalable for mixed media, but can introduce minor delays or jitter if not managed well. *Think*: you get a car to yourself on a road you share with other cars. You have no control over traffic conditions. This reversal enables unified IP-based networks handling everything from calls (VoIP) to streaming TV.

[*25*] A typical *home router* used to connect your home to the internet through your Internet Service Provider(ISP) actually a combined *switch* and *router* in a single device. Each are not nearly as high-powered as those designed to handle heavy enterprise loads, but are more than capable of handling the traffic for a small to medium-sized home. It generally supports a combination of Ethernet, Wi-Fi and whatever technology is used to connect your home to the ISP — often DSL or, increasingly common nowadays, Fibre Optic

[*26*] Strictly speaking this is only true at layer 2. As we will see in the next chapter, so-called *multilayer switches* can also operate at layer 3 and route layer 3 packets between VLANs—but only between those VLANs it manages. *VLAN trunking* allows a single physical connection to carry packets for multiple VLANs. It's typically necessary since packets from different VLANs often need to share the same cables. *VLAN tagging* adds an identifier to each frame so the receiving device knows which VLAN it belongs to and can deliver it correctly. Without tagging, there would be no way to distinguish which VLAN each frame came from once they're combined on the trunk link. In this book, wherever you see "*(V)LAN*" read "*LAN or VLAN*"—used in order to emphasise that what's being discussed applies equally well to both.

[*27*] Flexibility: in the sense that it's much easier to change VLANs than LANs—the former requires reconfiguring a single switch using software, the latter requires adding/removing whole switches as well as reconnecting individual interfaces, including any cables. Performance: in the sense that an organization can invest in a single high-performance switch which can apply its processing power intelligently to whatever VLANs currently have the most traffic, rather than multiple less capable separate switches, some of which may be idle while others are maxed out with current traffic.

[*28*] *WPA3*, the latest wireless security technology, significantly improves security on Wi-Fi networks, with encryption and stronger authentication. It's, however, still quite recent and adoption remains limited. WPA3-Enhanced Open, used in some public Wi-Fi, encrypts traffic without authentication, while WPA3-Enterprise, common in corporate settings, offers robust authentication but is rarely used in public networks due to complex setup and compatibility needs.

[*29*] as we will see in later chapters, the solution to this problem, as to several others, involves a new layer (layer 3) and new class of device called a *router* that plays as important a role there, as the switch does in modern layer 2 networks

3. Middle Layers — encapsulation, logical addressing, routing and conversations

This chapter explores layers 3 and 4 — the critical middle layers that transform isolated LANs into the global internet, enabling packets to flow using logical addressing and routing protocols. These layers solve the fundamental limitations of hardware-bound communication constraining individual LANs.

Layer 3 introduces IP addresses and packet encapsulation. You'll discover the central role played by routers, and how they enable a packet of data to travel from a source interface on one LAN to a destination interface on another. This doesn't just connect two LANs — it can span the entire world, forming the fundamental basis of the internet, enabling packets to traverse dozens of routers, each making independent forwarding decisions based on IP addresses. Layer 4 enables port-based interface sharing, with a choice of protocols such as TCP that ensures reliable in-order delivery, or UDP providing a lightweight alternative.

We'll examine the structure of the internet from two perspectives. Firstly, using a simplified tree-like model that reveals the core concepts of hierarchical routing, address ranges, and packet forwarding. Then, how the messy reality of different systems and economic relationships cause deviations from this model — and how model and reality combine to shape actual internet traffic flow. This dual approach ensures you grasp both the elegant theory and practical complexities of the internet.

We begin with a brief fictional narrative (*Metropolea*) that provides an intuitive metaphor for the technical concepts that follow.

Metropolea

The people of Metropolea were proud of their transport system. Each capsule could comfortably hold about 6 individuals, gliding almost friction-free through invisible electromagnetic tunnels from one station to the next. But impressive as the technology behind the capsules themselves and the details of their locomotion was, what invariably impressed visitors from other parts of Aetheria the most, was the vast and intricate web of routes between the countless stations. Stations which could appear overnight, and disappear even faster.

Aetheria discovered early in its development that solid objects are an illusion. The experience of touching another object is a result of interaction between the electromagnetic field of one object (even your body) and that of the other object. An interaction at a distance. All objects, no matter how solid they seem, are, in fact, primarily composed of empty space, and any touching that might be considered to take place (an overlap of these spaces) is prevented by the mutual repulsion of the electron clouds shrouding the objects. A repulsion, the strength of which increases exponentially as the objects in question near each other. Their mastery of electromagnetism, one of the four fundamental forces of nature, was a key to their technological advancement, and why so much of it is holography-based — what others might call virtual reality. The electromagnetic guides for the capsules, and the ability to make not only stations, but also whole local neighbourhoods serviced by them, appear and disappear overnight, bore testament to this.

Between (almost) every pair of cities was a single, direct, high-speed multi-capsule highway. These inter-city connections formed a mesh that could only be described as a translucent blur, along which the capsules sped in the blink of an eye. The inter-city station of each city was a correspondingly busy and highly secured (virtual) hub. In the rare cases where such a link broke (it generally did so only due to natural disasters, like the solar flares on Alpha-Proxima), it was no longer possible to travel directly from one city to the other. The inter-city mesh was, however, so complete that an alternative indirect route, through one or more intermediary cities, could still be found and used, while the direct link was repaired.

Within each city in the land of Metropolea, the stations were (originally) planned like a tree, the trunk of which sprang from the inter-city station. At each bifurcation of the tree was a new station, and each new station name was logically derived from its parent station. A city was typically divided into 4 zones (north, south, east and west), each of which was directly reachable by capsule from the inter-city hub. Each zone was subdivided into districts, numbered or named uniquely, each of which was further subdivided as necessary. Most (but not all) homes were in the suburbs (close to the stations forming the leafs of the tree). This was a very logical and easy to navigate system. Take, for example, a traveller in (or close to), station luna-east-12 (i.e. district 12 of the east zone of the city of luna), who wished to travel to luna-west-5 (i.e. district 5 of the west zone of the same city). See Figure 10. They first needed to get a capsule to luna-east station, from there another to luna hub (the inter-city station of the city of luna, and ultimate parent station of all other stations in luna), from there to luna-west station, and finally from there to luna-west-5.

The beauty of this system was its simplicity, and it worked impeccably—most of the time. But some routes became much more popular than others, putting a strain on this system. And since many travellers lived (in suburbs) near leaf stations, they typically had to travel all the way up and down the respective branches, to visit family or friends in a different suburb.

Shortcuts began to emerge, often run (or at least sponsored) by private real-estate companies, with a strong vested interest in the areas connected. These shortcut lines offered direct connections between a pair of stations that were not parent-child in the tree structure. They also, however, made the originally clear hierarchical tree structure more mesh-like. One could no longer discern what stations to travel through, in which order, based solely on their names.

One result of this was the transition of the role of station master from almost superfluous to crucial. Understanding and using the prefixes (city, zone, district etc) was still a good first guess, but now, to be able to calculate the best path for capsules to their intended destination stations, they needed to include the ever-changing shortcut lines. But there was no centrally managed, up-to-date list of these shortcuts, that they could query. The only ones who directly observed a shortcut connection joining or leaving the capsule network were the station masters in the stations the new line provided a shortcut between. So the station masters invented a new router protocol to pass on this information to station masters in other stations directly connected to them, and they in turn to those a step further away. So, up-to-date news of the locations and capacities (not all stations could handle the same amount of traffic) spread like a wave through the transport network. The elegance of this router protocol wave was second only to that which the station masters used to incorporate it into split second decisions, to determine which was the best track on which to send each waiting capsule on the next step of its (quickest possible) journey to its final destination station. All the while, the average passenger in the capsule remained oblivious to the complexity of the network it was navigating or the fact that they were even passing through stations along the way.

The invisible network that carried Metropoleans through their daily lives had evolved from a simple, predictable tree into something far more organic — a system that adapted moment by moment to the needs of its users. Like the neural pathways of a brain or the hidden root systems of an ancient forest, the transport network's true complexity remained hidden from those who relied upon it most. As travellers stepped into their capsules each morning, inputting their destinations with casual ease, they remained blissfully unaware of the symphony of split-second decisions. The constant flow of routing information, and the elegant dance of electromagnetic forces, would carry them safely to their destination. Perhaps this was the greatest achievement of Metropolea's transport system: not that it worked so well, but that its users never needed to understand how.

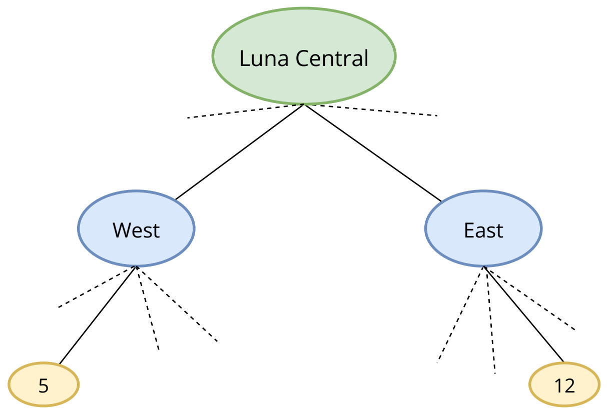

Figure 10. A hypothetical hierarchical connection between station luna-east-12 (i.e. district 12 of the east zone of the city of luna), and luna-west-5 (i.e. district 5 of the west zone of the same city) in Metropolea, before shortcuts became prevalent.

Layer 3

We saw in *Layer 2 Limitations* various limits one encounters when trying to scale a LAN to include more interfaces. Layer 2 networks (LANs) can't *simply be extended* to overcome these limitations, without introducing new problems. So instead, a new layer — Layer 3 — was built on top of Layer 2, allowing multiple LANs to be connected together.[1] A key technique enabling this is *Packet Encapsulation* — a clever trick used throughout networking, at various layers, as we'll repeatedly see in the coming chapters.

Logical Addressing

A single switch can handle thousands of interfaces, but this is still far too few to connect something nearly as large as the internet. The best hope is to *divide and conquer* by somehow connecting multiple switches together, each of which manages a subset of the interfaces. Each connected switch can *hide* the details of what port each interface it manages is connected to, saving all the other switches from the burden of having to do so. But the other switches still have to keep a record of all the interfaces managed by every switch—remember the (hardware) addresses are effectively *random numbers*—it would be great if there were a way of *summarizing* concisely which interfaces are managed by any given switch. For example, instead of having to maintain a list of 1000 (or more) entries, each of which is the (*effectively random*) numerical address of an interface managed by a given (other) switch, it would be great if the addresses in question *happened* to be 1000 *consecutive* numbers. Other switches would then only have to *maintain* a list of 2 numbers instead of 1000—the first address, and the last address[2]. See *IP Address Ranges* below.

To achieve just this, each interface is assigned an additional *logical address* in addition to the MAC address the manufacturer burned into it. The switch[3] in any given LAN *knows* both the logical and the MAC address of each interface in the LAN. But the *world outside* (the LAN managed by the switch) need only concern itself with the (convenient) logical addresses of the interfaces within. Whenever a packet enters or leaves the LAN, the switch *translates* the logical addresses to the MAC addresses and vice versa to make this work. Not only does this make it possible to **scale** multiple connected LANs to a network of many more (total) interfaces than would otherwise be possible, it also makes the resulting network much more **flexible** (interfaces may be grouped by role, region etc.). If, for some reason, a node, or interface needs to be replaced for some reason (*think*: replace a printer by a new one), the new interface(s) can be assigned the same logical addresses as before—so outside the LAN where all nodes were using only the logical addresses anyway, no change is *noticed*. One is reminded here of a famous quote, often referred to as the *Fundamental Theorem of Software Engineering*.

Consider by analogy cars, each of which get a globally unique *Vehicle Identification Number (VIN)* burned into it by the manufacturer. Each VIN is a 17-character code like *1HGBH41JXMN109186* that permanently identifies that specific vehicle worldwide. The VIN encodes details about the manufacturer, plant location, model year, and a sequential production number, but from the perspective of a car owner or local authorities, VINs are essentially random numbers. You cannot tell from a VIN alone whether a car is registered in Dublin or Berlin, who owns it, or what it's being used for. The VIN stays with the car forever—if the vehicle is sold, moves across the country, or changes hands multiple times, the VIN remains unchanged. While VINs are essential for manufacturers to track recalls and for investigators to identify stolen vehicles, they provide no organizational convenience for day-to-day vehicle management.

Licence plates provide a logical addressing system layered on top of VINs. Unlike the permanent VIN, licence plates are assigned by local authorities and can be grouped geographically and organizationally. A plate like *IRL 252-D-12345* immediately tells you this vehicle is registered in Dublin, while *D B-AB 1234* indicates registration in Berlin. This geographic grouping provides enormous convenience—when authorities need to search for vehicles in a specific state or city, they can immediately identify candidates by their licence plate format, rather than having to cross-reference millions of random VINs against registration databases. Crucially, licence plates are dynamic and transferable. When you sell your car, the new owner typically gets a new licence plate that reflects their location and registration preferences. The same physical vehicle (same VIN) might have a Texas plate when owned by someone in Dallas, then get a Colorado plate when sold to someone in Denver. This flexibility means the logical addressing system can adapt to how and where the vehicle is actually being used, regardless of where or when it was manufactured.

Network interfaces use this same two-tier addressing approach. Each interface gets a permanent MAC address burned in by the manufacturer (analogous to VIN)—a seemingly random number that stays with the hardware forever. But each interface also gets assigned a logical IP address (analogous to a licence plate) that can be grouped by network location, department, or function. Just as licence plates let authorities easily identify *all California vehicles* or *all vehicles in this city*, IP addresses let network administrators easily identify *all devices in the accounting department* or *all devices in the Paris office*. And just like licence plates, IP addresses are dynamic—when you replace a broken printer, the new device gets a different MAC address but can be assigned the same IP address, so everyone who was printing to that logical address continues to work seamlessly.

The word *logical* in *logical addressing* can be a bit misleading, as it perhaps conveys mathematical connotations, which are not intended in this context. When you read *"logical"* here, think instead *"convenient and flexible (from the perspective of how they are used)"*. [4]

Fundamental Theorem of Software Engineering

We can solve any problem by introducing an extra level of indirection.[5]

— Andrew Koenig

Packet Encapsulation

Packet encapsulation is the insertion of a new type of packet (used by a higher layer) into the *payload within* an existing (lower-layer) packet. See *Figure 11*.

Imagine a large company with branches in different cities, each with numerous internal departments. An employee at one branch wants to send a packet to an employee in a specific department at another branch, using an external courier. Two options exist. First, the sender could write all address details — including the company internal department — on the packet, hoping the courier ignores or isn't confused by the unfamiliar company-specific info. This mirrors trying to extend Layer 2.

The second option — packet encapsulation — is smarter: The employee places the packet with internal details inside an outer packet. The outer packet is addressed only with the info the courier expects. The courier, unaware of the inner packet's special contents, delivers it to the destination branch without issue. There, a recipient familiar with the company internal *protocol* opens the outer packet and processes the inner one according to its instructions.

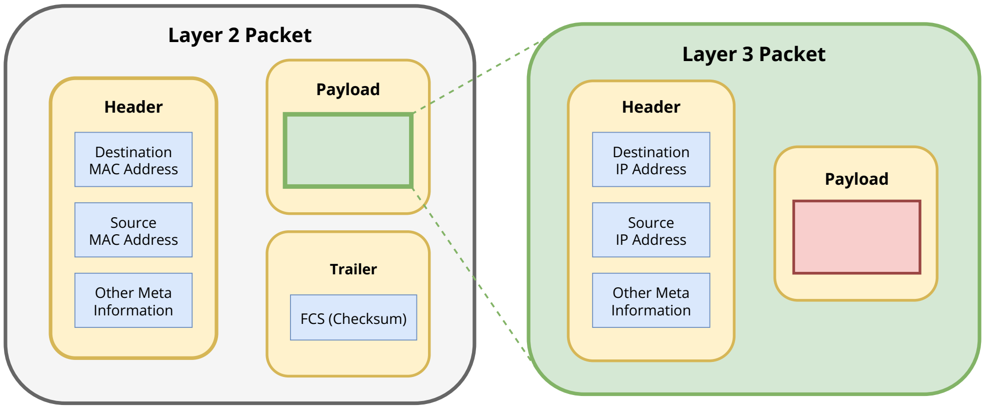

Figure 11. **Layer 3 packet encapsulated** *as the payload of a layer 2 packet. The layer 3 packet contains the logical (IP) address, whereas the layer 2 packet contains the hardware (MAC) address of the destination and source interfaces, respectively. The payload of the layer 3 packet contains the actual data the source is sending to the destination.*[6]

Protocols

A protocol is a set of rules that defines how different parties should behave when they need to accomplish something together. Protocols exist everywhere in human society — they're the agreed-upon ways of doing things that ensure everyone knows what to expect and how to participate. At its core, every protocol specifies:

- Who the participants are and their roles
- What information needs to be exchanged
- What format that information exchanged should take
- What order things should happen in
- How to behave in different situations

Take, for example, when an ambassador is newly appointed, and needs to formally establish their role in a foreign country. The ambassador must request an audience with the local Head of State, dress in specific formal attire, follow a pre-determined procession route, present their credentials (formal letters from their own Head of State) in a specific manner, and exchange greetings using predefined phrases (*Your Excellency*, *Mr. President*). There are strict rules about who speaks, for how long, and what they should say.

In the world of digital communications, protocols serve the same fundamental purpose — they're the agreed-upon rules that allow different systems to work together effectively. A network protocol specifically defines how information should be exchanged between systems, including:

- *Structure*: The exact layout of *packages* exchanged, including which fields (pieces of information) can be present and where each field is positioned
- *Semantics*: What each field means and how it should be interpreted
- *Timing*: When messages should be sent and how long to wait for responses
- *Error handling*: What to do when things go wrong

At layers 1 and 2 one speaks of *standards* (which *both* define protocols *and* specify hardware). At Layers 3 and above there is no (new) hardware to specify, so one speaks there only of *protocols*.

Layer 3 packets and protocols

Layer 3 packets are often called *IP packets*, and the most fundamental layer 3 protocol is called the *Internet Protocol (IP)*, the most commonly used versions of which are 4 and 6. The main function of the IP protocol is to define a standard structure and semantics of the IP packets — the most important of which are the *IP Addresses*. There are other layer 3 protocols, which we will meet in the course of the chapter, each of which has a specific purpose, but every other (modern) layer 3 protocol relies directly or indirectly on the IP protocol.[7] *Think:* IP is like the postmen who actually deliver packets, while other layer 3 protocols are used to keep that delivery system running (fixing delivery trucks, updating maps, reporting incorrect addresses).

Protocol Layering

Packet Encapsulation facilitates *protocol layering*—a new protocol is layered *on top* of an existing one (recall *Layers of Abstraction*), which involves embedding a new *packet* type *inside* (as the payload of) an existing one, as illustrated in *Figure 12*. The lower layer protocol doesn't need to understand what's inside its payload — it just treats the inner packet as data to be transported. *Think*: Russian dolls, where the *inner* doll shells correspond to *higher* (more abstract) protocol layers. And true to the Russian doll analogy, as we will see in later chapters, the inner (higher-layer) packets can themselves in turn contain (even higher-layer) packets, and so on, recursively.

The header of the new layer 3 packet is where the new logical addresses (namely that of the sender and intended final destination of the packets), as well as any other meta information the new layer 3 protocol needs, is stored. Each layer handles its own specific job (like addressing, routing, or error detection) while carrying the complete messages from the layer above it. It's like putting a diplomatic pouch (one protocol) inside a shipping container (another protocol) — the shipping company doesn't need to know about diplomatic procedures to transport the pouch correctly.

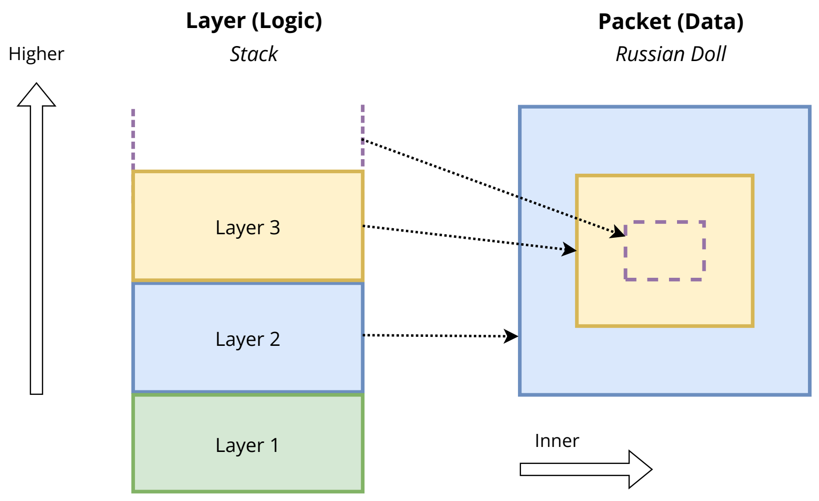

Figure 12. **Protocol Layering** *and* **Packet Encapsulation**. *The extra information in inner packets is "hidden" from lower layers that might be "confused" by it. Higher layers correspond to processing logic that understand and expect this extra information, and know explicitly where, i.e. in what doll shell, to look for it. Layer 1 is an exception in that it does not work with packets of data, but rather streams of bits (and electromagnetic waves).*

The order of packet creation at the source and processing of the logic associated with each layer happens in top-down order, as illustrated in *Figure 13*. *Think*: When you start planning your next holiday, you first start off with high level considerations like the time of year, and whether you would like to be near a beach or mountains to climb. Only after this has been clarified, do you look into more low level details like flights on suitable dates, or specific train connections to the airport. This higher layer first aligns with packet encapsulation order shown in *Figure 12*—you need to create an inner packet before you can encapsulate it in an outer one. On the receiving end, the order is reversed. *Think*: Your plane first has to touch down, and you have to figure out the details of train connection from the airport to your hotel, before you can start to enjoy a swim at that beach or a hike up that mountain.

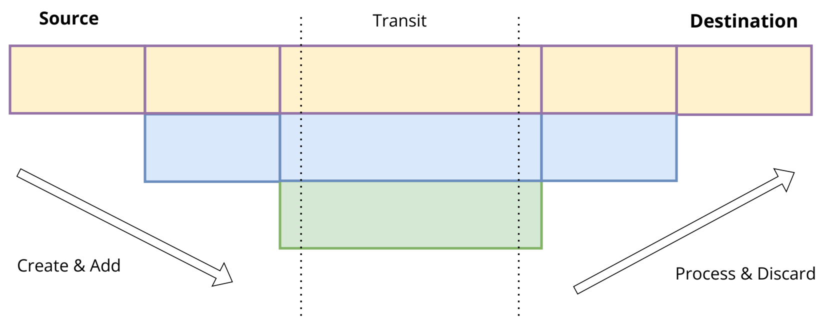

Figure 13. **Layer Lifecycles**. *At the source, the packet for each layer is created, and the layer logic applied, in top-down order. At the destination, the packet for each layer is read, and the layer logic applied, in the reverse order.*

IP

IP (Internet Protocol) is the most important layer 3 protocol. It has evolved over the years into successive standard *versions*.[8] Version 4 of the IP protocol, abbreviated to *IPv4*, has been the predominant version in use since 1981. It is, however, currently being transitioned to version 6, initially released in 1998, and abbreviated to IPv6, which is designed to address IPv4's limitations, especially the much more limited number of IP addresses it allows. However, the transition is happening relatively slowly, and will presumably take several years to complete, as, in order for a packet to be transmitted successfully form its source to final destination using v6, *all* (of the possibly very numerous) nodes the packet passes through, must fully support v6. There are still quite a significant number of legacy nodes that relay internet traffic, that support only v4, and any one of them receiving a packet using IPv6 causes the transmission as a whole to fail—the packet will not reach its final destination. Until all such legacy nodes have been replaced or updated to support v6, all other nodes continue to support both v4 *and* v6 simultaneously. A source node that receives an error message saying that transmission of an IPv6 packet failed due to lack of IPv6 supported along the way, has no choice but to give up or send with v4 instead. As a result, it’s not uncommon for many nodes to just use IPv4 from the start. And herein lies both the strength and weakness of the transition - its strength is its flexibility in allowing nodes to continue to (successfully) use IPv4, but the weakness is the lack of sufficient *incentive* for them to update to IPv6, a version of the protocol standardized almost half a century ago, so all nodes can benefit from its improvements.

Rather than dryly list all the differences between v4 and v6 here (and risk corresponding chicken-egg complications), each difference shall be presented as it becomes relevant during the course of the rest of the chapter. See also the chapter *summary* for a more concise presentation.

Decimal, Binary, and Hexadecimal

The system we usually use for numbers is called the decimal system. It uses the digits 0-9, and we say its *base* is 10. A number is an abstract concept, linguistically called a *quantifier*—there is no such thing as a number *on its own*. There is for example no *2* in itself, but only *2 of* something else, e.g. *2 apples*. A number expressed in decimal format is what many of us intuitively consider to *be* a number. However, there is nothing more special or fundamental about the decimal number system than many others. In fact, if humans had evolved with say 8 or 12 fingers,[9] then we would almost certainly be using a number system based on 8 or 12, rather than 10. (The word *digit* comes from the Latin *digitus* which means *finger*).

The *simplest* and most *natural* numbers system for a (digital) computer is binary, which uses just 2 digits: 0 and 1, as mentioned already in *Bits and Bitstreams*. And any number (we express as a series of decimal digits) can also be expressed as a (longer) sequence of binary digits. For example: The number expressed in decimal as 192, can also be expressed in binary as 11000000. It's the same number, just a different way of writing it. Binary is more natural for expressing numbers used by digital computers, since a digital computer works internally with 2 values (which map directly to two hardware *states* in a transistor, often called *on/off* or *high /low*, and the corresponding *bits* in software).

When discussing number systems, it's more accurate to use the term *symbol* than *digit*, since in general any symbol from the set may be used (not only the digits 0-9) in the sequence representing a number. Decimal expresses numbers as a sequence of the *set of symbols* 0-9, and binary from the set of symbols 0,1. Using decimal to express a number is more compact, in that it generally requires a shorter sequence of symbols (from a larger set) than does binary. A number system which strikes a good balance between compactness (it's 4 times more compact than binary, and about 1.2[10] times more compact than decimal) and which has a natural mapping to the internal processing of numbers in digital computers is *hexadecimal*. The *hexadecimal* system uses the symbol set 0-9 together with A-F, i.e. it adds 6 more symbols to those used by decimal. It maps easily (low cognitive load) to binary because each possible sequence of 4 binary symbols corresponds to exactly one hexadecimal symbol. For these reasons, it is often used when dealing with computers and in particular long IP addresses, such as those used in IPv6.[11]

IP Addresses

IP addresses are really just numbers from 0 to some maximum value. How large this maximum value is, is one of the most significant differences between IPv4 and IPv6. In IPv6, this maximum value is much larger, meaning the total number of addresses available is much larger than in IPv4. Each unique value of the number between 0 and the maximum value, corresponds to a unique address—think a street with numbers for the houses[12]—each number can be given to only one house, and if the maximum house number possible is larger, then you can have more (*uniquely numbered*) houses on a single street.

IPv4

An IPv4 address is a **32-bit** number, which means it can represent 2^{32} (about 4.3 billion) unique addresses. Rather than writing this as one (possibly enormous) decimal number, it's usual to divide it into four equally sized, 8-bit sections (octets) and represent each in decimal. For example:

```
192.168.1.1
```

This so-called *dotted decimal* notation is simply a human-friendly[13] way to express the underlying 32-bit number: `11000000.10101000.00000001.00000001`. Which is equivalent to the single very large decimal value: `3,232,235,777`.

IPv6

An IPv6 address is a **128-bit** number, allowing for 2^{128} (an almost inconceivably large number) unique addresses. To represent these efficiently, we use hexadecimal notation (base 16), and group bits into *eight* 16-bit sections separated by colons. For example:

```
2001:0db8:0000:0000:0000:8a2e:0370:7334
```

There are two shorthand *tricks* that can be used when writing such numbers (without causing ambiguity)

- Leading zeros within a group can be removed. e.g. `0db8` can be replaced with `db8`
- One (and only one) sequence of *consecutive* `0000` *groups* can be replaced with `::`
  The double colon can appear *at most once* in an address. If used twice, the address would be ambiguous, as we wouldn't know how many zeros each *::* represents. Effectively, wherever you see *::* you can mentally replace it with as many groups of *0000* as are necessary to fill out the eight 16-bit sections.

  So, the address `2001:db8:0000:0000:0000:8a2e:0370:7334` can be rewritten in the shorter (but equally unambiguous) form `2001:db8::8a2e:370:7334`

What happened to IPv5

You may be wondering why v6 is the successor to v4 instead of v5. There was, in fact, a version 5 of the IP protocol, which was designed to support real-time traffic, like voice and video. It, however, was not fully standardized for general internet use, before it became apparent that the shortage of IPv4 addresses was a more pressing problem than streaming media efficiently. Work on standardizing v5 was effectively abandoned (streaming support is now the purview of higher layers, which later became apparent are better suited to handling it—we'll see more about this in later chapters).

It's common to specify a particular version of the fundamental protocol of layer 3 by adding a suffix (e.g. v4 or v6) to the base name (IP) as in IPv4 and IPv6. Some of the non-fundamental layer 3 protocols follow this pattern, e.g. ICMPv4 and ICMPv6. Whereas others have entirely different names depending on whether they are designed to work with IPv4 (e.g. ARP) or IPv6 (e.g. NDP).

Subnets

A subnet is a LAN where the interfaces involved are layer 3 (enabled). By this is meant that they understand and participate in communication using the protocols of layer 3 (in particular the *IP protocol*). Recall from *Layer-Specific Devices* that this implicitly means that they are both layer 2 and layer 3 enabled. When a node wants to send an IP packet to a different interface on the same subnet, it encapsulates the layer 3 packet in the payload of a layer 2 packet, as described in *Packet Encapsulation*. The header of the layer 3 packet includes the (logical) IP address of both the sending and receiving interfaces. The layer 2 packet (also known as a *frame*) gets sent like all other layer 2 packets, and arrives as such at the intended destination interface, corresponding to the destination address in the layer 2 header. The recipient (which, like the sender, must be layer 3 enabled—in particular to be able to *understand* the IP protocol) extracts the layer 3 packet from the payload of the layer 2 packet, a process also called *decapsulation*, and processes the rest of the IP packet accordingly.

IP Address Ranges

In theory, a subnet could use any unique IP addresses for its interfaces, and these interfaces could communicate with one another at layer 3 using them. However, in order to be able to communicate with interfaces beyond (which is one of the main motivations for using layer 3 in the first place), a subnet must assign IP addresses to its interfaces from a valid *range*. In general (see *Subnet — reserved addresses*), the first and last possible addresses in the range are reserved, leaving two fewer usable addresses which can be assigned to actual interfaces.

Additionally, as mentioned in *Logical Addressing*, one of the big motivations for using logical addresses is that related[14] interfaces can get assigned (consecutive) addresses from a single *address range*. This has the big advantage that referring to all such related interfaces requires (remembering and) referencing only the range of the addresses (e.g. start and end address), rather than every individual address involved (which would be necessary if the addresses involved were not consecutive).

Imagine by analogy[15] a large storage warehouse with *10 floors*, numbered 0-9. Each floor in turn has space for *1000 units*—individual containment units, each of which can potentially be rented out to a different customer to store their personal property. One could encode the *location* of any individual containment unit using a combination of *1 (0-9) + 3 (0-999) = 4 decimal digits*. However, on each floor, units ending in 000 and 999 are reserved for administrative purposes (*think*: an office for the floor manager and a supplies room)[16], leaving units 001-998 (998 units) available for customer rental on each floor[17]. For example, *7456* might identify the unit 456 on floor 7. Similarly, *7000/1* could identify the range of all units on floor 7. And *7456/1* would identify the same specific unit (456 on floor 7), but additionally signify that the floor has *space* for at total of 998 customer units).

CIDR

CIDR (Classless Inter-Domain Routing) notation is a way of specifying an IP address together with a suffix, that conveys information about an associated address range, as illustrated in *Figure 14*. It's the standard way to specify the address range that defines a subnet. The suffix specifies the *prefix length*,[18] which defines how many left-most bits in the (binary notation of the) IP address are fixed for all addresses in the range (and thus defines all possible values in the range). These left-most bits are often called the *network prefix*. The bits to the right of the network prefix are often called the *host bits*. If the host bits are all 0, then the address is taken to identify the range as a whole (and cannot be assigned to any individual interface). If the host bits are not all 0[19], then the address can be assigned to an individual interface, and the suffix just adds the additional (context) information as to what range this IP address is a *member of*.

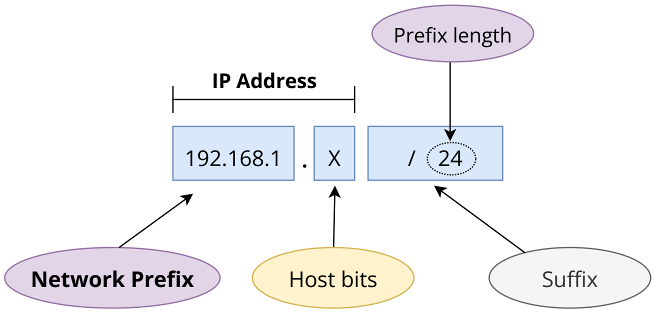

Figure 14. **CIDR** *example specifying an IP address and a suffix. The number in the suffix is called the "prefix length" and can be used to separate the IP address into two parts—that on the left is called the "network prefix" (which defines an address range, and contains as many bits as the value in the suffix) and that on the right is called the "host bits" (or "host identifier"), which defines a specific address in this range. The larger the prefix length, the more digits in the network prefix, and correspondingly fewer host bits (and vice versa, since the sum of both is constant). If the address in question defines a subnet, then the network prefix is also known as a "subnet prefix".*

For example, *192.168.1.0/24* means the range of IPv4 addresses: *192.168.1.0*, *192.168.1.1*, ... *192.168.1.255*. The first 24 bits (192.168.1) identify a range of IP addresses, and the remaining 8 bits, are in principle[20], available for specifying a particular IP address within that range. This range contains $2^{(32-24)} = 2^8 = 256$ IP addresses, and thus can, in principle, contain 256 interfaces. *192.168.1.1/24*, on the other hand (where the host bits are 00000001), specifies the single IPv4 address *192.168.1.1* as a member of the range *192.168.1.[0-255]* (or more accurately, *1-254*, after accounting for reserved addresses).

When using CIDR notation to specify IP address ranges, an important constraint emerges: the ranges cannot be completely arbitrary but must have a start and end which can be expressed as powers of 2—one speaks of the ranges being *aligned on boundaries that are powers of 2*. This is the trade-off for using a single number (the prefix length) to specify a range, rather than explicitly providing both start and end addresses. For example, if you want to create a subnet with 256 addresses (a /24 network), it must start at an address where the last 8 bits are all zeros—such as 192.168.1.0, 192.168.2.0, or 192.168.3.0, but not 192.168.1.50 or 192.168.1.128. This boundary alignment is a natural consequence of using a single number to represent the range, but it also enables routing tables to efficiently represent and process subnet information using simple bit-masking operations[21]. See *Figure 15* and *Figure 20*.

Index