Exploring Communication Protocols

> *"A protocol specification defines the rules and behavior participating in the transfer of data." - Patterns in Network Architecture by John Day*

Problem Statement

We have a pair of nodes that need to share data across a network. These nodes may be on the same link, but it's also possible they're connected through a series of intermediate devices (such as routers). More than just sharing data, the nodes need assurance that the data will be communicated in a way that meets the expectations of our application, which can be quite precise in the case of an application like voice or video. Likewise, the nodes need to be able to communicate with each other while also sharing resources with others on the network and permitting multiple applications on the same device. This level of coordination will be impossible without the ability for nodes to share some information about state along with the raw data.

Overview

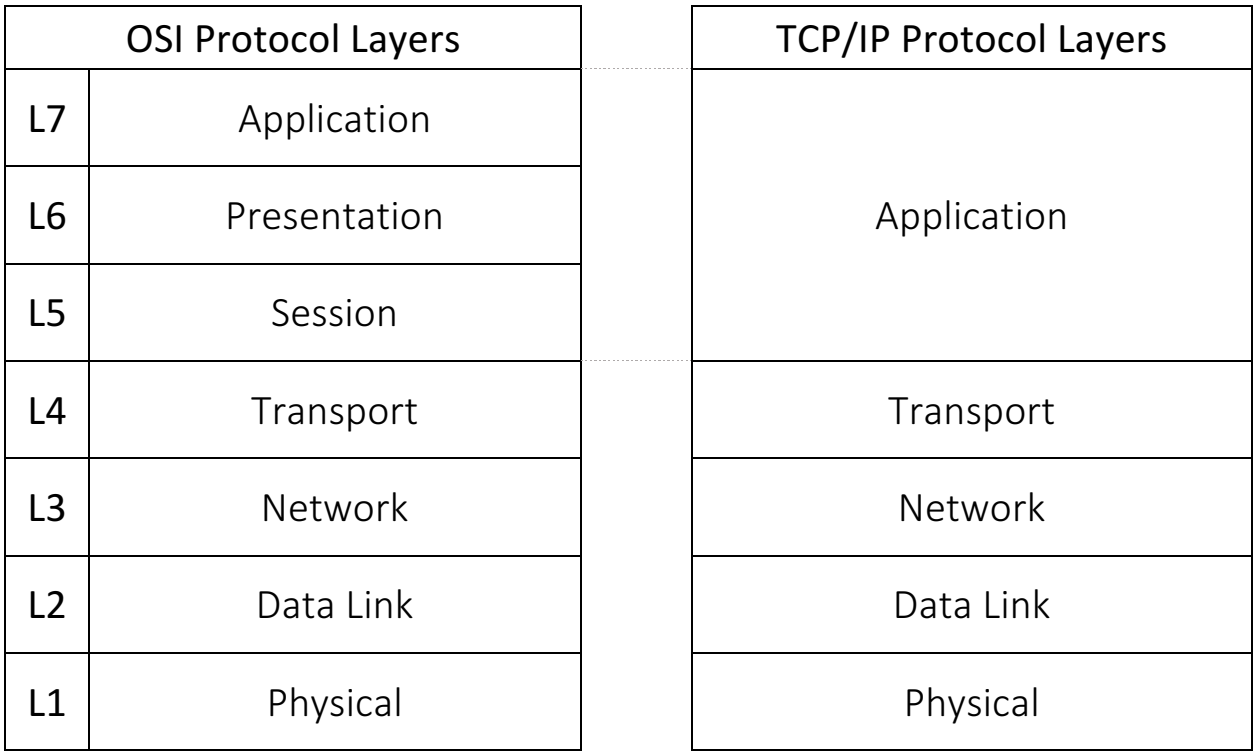
To accomplish these objectives, our communication systems make heavy use of protocols that enable our devices to coordinate these details in a systematic manner. The protocols we are discussing follow a common pattern of organization and information flow, i.e., communication proceeds by passing data and state through largely distinctive layers that bridge the gap between an application and the "wire."

The set of layers used for an application on a particular device is commonly referred to as a *protocol stack*. In addition to keeping protocol implementation more manageable by separating concerns, this model enables us to swap out pieces of the stack to support different use cases so that the two sides of a connection can live on different types of networks or devices (e.g., a mobile app on a wireless network talking to a Linux server on a virtual machine in the cloud).

As we look at protocols, we're going to consider two models that can be used to understand protocols in modern communication systems. Originating in the early days of the internet, the *OSI Reference Model* and the *Internet Protocol Model (also known as the TCP/IP Model)* have woven themselves quite deeply into the language of our profession.

Arguably the most important, the TCP/IP model aligns most closely with the communication infrastructure of general purpose computing devices and networks. It uses five layers to bridge the gap between the physical network and the application. The OSI reference model provides some additional granularity, further dividing the Application layer for a total of seven layers. These models are depicted below:

Copyright 2017 - Clinton J. Campbell



We'll spend plenty of time digging into these two models throughout the quarter, but we should first review the broad composition of these protocols.

Information Flow

When we're looking at the end-to-end connection between devices, communication is encapsulated on a layer-by-layer basis. By this, I mean that the application layer of your phone (running a web browser) is addressing its messages to the application layer of the remote web server. You'll see that the application does have some coupling to the layers immediately below it, but it is more or less completely decoupled from the Data Link and Physical Layer at the bottom of the stack. Likewise, as we move down the stack, we find that each layer finds a peer at the same level on the other end of the connection.

There are a few distinctions to be aware of at this point. While connections in the Network, Transport, and Application layers are end-to-end between devices, connections at the Physical layer and Data Link layer are handled hop-by-hop. As such, if we compare the data sent on the wire to the data received, we'll find that numerous changes have occurred in the *lower* layers while the *upper* layers are mostly identical on both ends. Some changes might have occurred on the network layer to support smaller transmit units encountered along the way, but these changes can be reversed in a straightforward manner to continue processing the message that was originally sent.

Copyright 2017 - Clinton J. Campbell

Give some thought to how this process relates to analog communication processes, whether that be the notes you might have passed in grade school or "snail mail" that you send today.

Different Types of Relationships

Several terms will be used to describe peering between devices and processes that participate in communication. A few terms you're most likely to encounter are *association*, *flow*, and *connection*. Though they're sometimes used interchangeably without much thought, these terms highlight differences that exist between different types of applications. They tell us how much state our peers share and how the application responds to events impacting the stream of bits flowing from one end to the other.

An association represents the minimal coupling between the endpoints (and higher/lower protocols in the stack). In contrast, a connection indicates a much higher level of coupling in the state of the endpoints. Whereas an association may refer to a one-off message sent between two points, a connection is used to describe an ongoing relationship and a set of expectations (e.g., error correction services) that apply to conversation. A flow falls somewhere in between. An application has established something more than a casual association with a peer, but it makes fewer attempts to control the state of the conversation.

Moving Data through the Stack

While peering is taking place in a "horizontal" manner within layers, vertical relationships also exist between adjacent layers. Beginning with the application, each layer on our stack consumes services offered by the layers below. The Application layer relies on the Transport layer to establish a process-to-process link on the remote endpoint, possibly managing connection characteristics like error correction and flow control. Likewise, these two layers look to the Network layer to manage the logistics of moving the packet from one host to the other. In our conceptual model, data flows down the stack on the sender and back up at the recipient.

We can expand this idea somewhat and say that each protocol is implemented with interfaces to facilitate:

- Interactions with the upper interface
- Interactions from a remote peer at the same layer
- Interactions with the local system
- Interactions with the lower interface

Distinguishing Application Protocols

It should be somewhat clear just by looking at our reference models that protocols can be divided between those that support specific applications and those that provide more generic data

Copyright 2017 - Clinton J. Campbell

transfer services. A formal way of thinking about this distinction is to consider what *state* a particular protocol is concerned with modifying. Application protocols are concerned with system state outside the boundaries of the protocol stack, whether that is manifested through changes to a user interface, a database, or a file system. In contrast, data transfer protocols work with the state of the communication stack and processes. Perhaps we will keep logs related to these protocols to support external requirements, but data transfer protocols are self-contained within the stack. They do not reach further into the system than the application asks.

At this point, we've defined the overall patterns of data and information flow. This perspective is just one way to look at the broader concept. We'll discover some additional perspectives as we continue, while also getting more familiar with the details of the OSI and TCP/IP reference models.

Breaking down the Layers

By this point, you should understand that applications communicate by passing information up and down a protocol stack in order maintain a logical association with a process on the other end of the wire. We're going to dig deeper into the components of a modern protocol stack using OSI and TCP/IP as our guiding models. Through this discussion, we'll also become more familiar with the "primitives" we use to construct or describe a given protocol.

Protocol Data Units

The first primitive that you should be aware of at this point is the protocol data unit (PDU), which is the generic name we give to the data structure that is native to a protocol. You probably won't use this term much in practice, but you should know that it defines the relationship between the names that are more commonly assigned at each layer.

In terms of structure, the overall shape of a PDU is mostly independent of the layer or name we give to a type of PDU. Each protocol accepts user data from a higher layer and wraps it with a structured header and (in some cases) a trailer.

```
PDU ::=  <Header> <User Data> [Trailer]
```

Protocol Headers

Headers carry information about system state that is relevant to peers on the other end of the association. If our message is destined for a certain port or address, that information must go into a header at the appropriate layer. If a protocol requires endpoints to negotiate parameters pertaining to a connection, e.g., encryption keys or algorithms, that data is encoded into headers.

Almost universally, headers will provide a cue to the other end about how much data the PDU contains. This cue will typically take the form of a length field or predefined delimiter. Length is an important, and easily overlooked, concept in data communication. Decisions about PDU

Copyright 2017 - Clinton J. Campbell

length balance tradeoffs between various engineering concerns. E.g., processing/bandwidth efficiency, error cost, buffering constraints, fairness, etc.

User Data

While the term user data is relatively self-explanatory, we should note that the concept of layering simplifies our protocols in that they don't need to understand the contents of the user data they encapsulate beyond some basic characteristics, e.g., length of user data.

Given a chunk of data from an adjacent layer, we add or remove our own headers and footers and pass the result up or down the chain. As a message moves down the stack on a sending node, the concept of user data grows to encompass the actual data originating from the application in addition to the new headers and footers that were added on the way through the stack.

Policy vs Mechanism

While many of these fields support the basic mechanism of the protocol, others are present to support configurable parameters associated with policies. In this context, policies shape the behavior of the protocol beyond what is determined by the base mechanism. Policy provides a great amount of flexibility and enables us to rely on smaller set of protocols than we might if every parameter was pre-determined. Given a policy, we can modify a protocol on the fly in order to adapt to the current network conditions.

Protocol Trailers

Nearly every protocol specifies a header, but trailers are rarely used except in the layers closest to the physical medium since they append information that wasn't available when the device started sending the *frame*. Error detection and synchronization fields are the most common items included in a trailer. You'll see that 802.3 uses its trailer to append a cyclic redundancy code (CRC) on each frame that the receiving NIC can use to detect an error that occurred in transmission. By placing this value in the trailer, rather than header, the sending NIC can eliminate buffering cost and stream the data onto the wire as it processes what arrives from above.

Exercise

Compare the headers and trailers of a data PDU to the information that goes onto a snail mail envelope, i.e., recipient name and address, return address, and postage. Note the standards that we have developed to ensure proper delivery of mail, and take some time to explore the concepts more deeply.

- What can you infer about the way the lower layers of the *Snail Mail Protocol Suite™* interact with your item in transit?
- What are some distinct protocols and services you can identify based on your analysis?

Copyright 2017 - Clinton J. Campbell

- The contents of the envelope likely conform to an alternate set of standards or conventions. How would you map these concepts to the formal ideas we've discussed so far?

Copyright 2017 - Clinton J. Campbell