Chapter 1 The Internet Protocol Stack and the Network Workbench

The Language of Networking

What is a network? Few people in our technological society can avoid hearing this term several times every week, but most of us would be hard pressed to define the word *network*. Definitions abound, often differing slightly from author to author. We use the following descriptions of network technologies in this book (see the Glossary for more definitions).

Communication: The process of passing information from a *sender* to a *receiver*. This process requires a *channel* or *medium* between the two and a way of representing information that is shared between the two. Figure 1.1 shows the basic communications process. In all real communications, there are occasional lapses in the quality of information received so that not all of the information sent by A reaches B. We refer to this garble added to the information as *noise*. The sort of communication that interests us in this book is *telecommunication* — that is, communication achieved by electronic means at a distance, usually under circumstances in which old-fashioned communication by human voice alone is impossible.

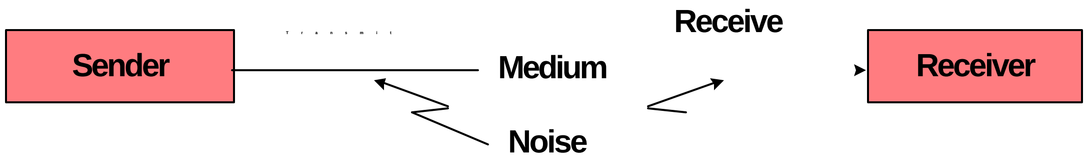

Figure 1.1 Basic communication process.

Data communication: In the case of computer communication, the information has a clear, definable value. In this book, we assume that A and B are computers, and that they can communicate in both directions at the same time (A to B **and** B to A). When this happens, the communication is called *full-duplex* communication. If communication can take place in only one direction at a time (A to B **or** B to A), that is *half-duplex* communication. Computers work with *binary digits (bits),* so the rate at which the information is passed is measured in *bits per second*. In computer communication, the noise takes the form of *errors,* meaning that the received information does not match what was sent. As we see later, the "data" being communicated between computers today may represent the traditional sort of data (numbers or text), or it may represent computer-encoded audio, video, or graphic images (known as *multimedia* communication). You will learn more about data communication in the chapters on Data Link Control (DLC).

Units of Data Communication

Link data rates may be measured in:
- kilobits per second (kbps): thousands of bits per second
- megabits per second (Mbps): millions of bits per second
- gigabits per second (Gbps): billions of bits per second

When using these values in C, we will use *exponential notation*, for example one megabit per second, which also can be expressed as $1.0\text{x}10^6$ bps in scientific notation, is 1.0E6 bps in exponential notation.

Sometimes the data rate is given in *bytes* per second (Bps) where one byte = 8 bits, thus 1 kbps = 125 Bps.

Network: A collection of processing elements that have the ability to communicate with each other. The elements of the network are generally called *nodes*. Nodes represent fixed points to which communication *links* connect. Communication in the network can happen either directly, through a link between two communicating elements, or indirectly, in which case the two participants pass the information through one or more other intermediary elements (which of course requires a path of links and nodes between the two). Figure 1.2 is a diagram of a simple network, in which the nodes are shown as circles and the links as lines. We can see that some pairs of nodes are able to communicate directly (for example, nodes 1 and 4). Other pairs—for example , nodes 1 and 7—must pass information through other nodes. The ability to share resources for communication is one reason networks have become used so widely. Many networks, such as the one in Figure 1.2, also provide more than one path between any pair of nodes for improved reliability.

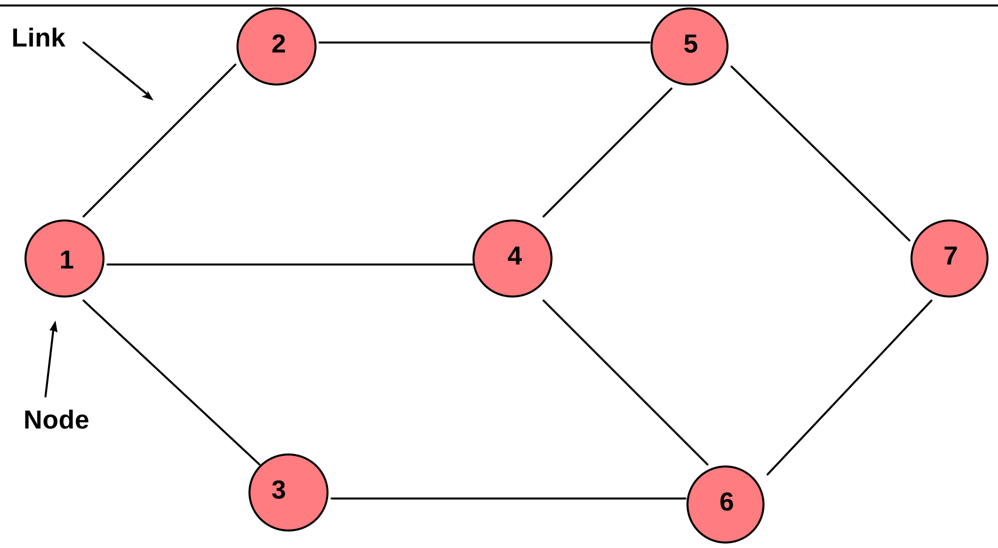

Figure 1.2 Seven-node network.

Local area network (LAN): A network that spans a small distance, typically a single building or a few buildings on a campus. Most LANs offer high data rates (from 2 megabits per second upward) but do not provide multiple (or *redundant*) paths between nodes. LANs are relatively inexpensive and generally are owned by the organizations that use them. Figure 1.3 shows a simple LAN in which several computers communicate with each other over a shared wire, known as a *bus*, which uses a simple mechanism called *broadcast*, whereby each transmission is received by all stations. You will learn more about LANs in chapters 6 and 7.

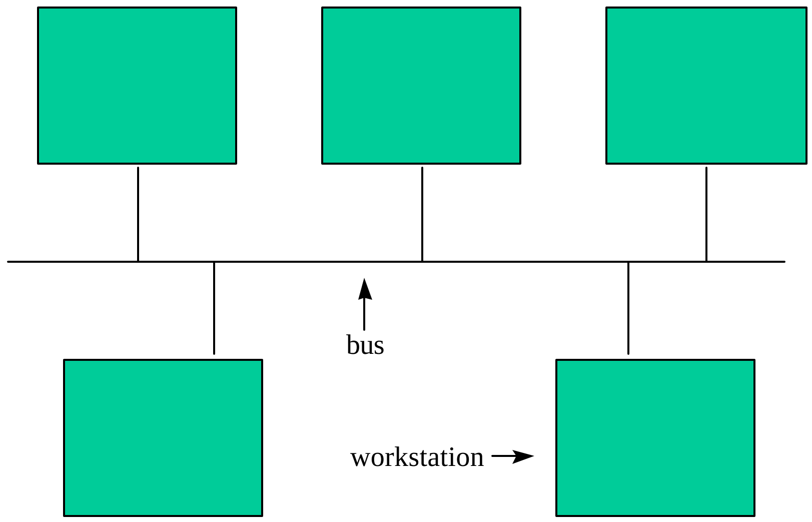

Figure 1.3 Bus LAN with five nodes.

Wide area network (WAN): A network that spans a larger geographic distance, ranging from a few separated buildings to worldwide locations. WANs normally are assembled from *leased links*, provided by a commercial telecommunication *carrier*: a company that is in business to move information electronically between distant points. The computers at which the links terminate are known as *routers* because they typically have the ability to route the information toward the destinationover more than one link, using multiple paths to other router nodes. A typical WAN might be organized as in Figure 1.2. The routers might be operated by the carrier, by the using organization, or by a third organization, known as a *value-added network provider,* that provides more powerful network services usingthe carrier's communication circuits. Today the most common type of value-added provider is an *Internet service provider (ISP)*.

The network illustrated in Figure 1.2 is known as a *switched network* because the routers pick specific paths among the links and nodes for information flows. That is, they switch the information into different parts of the network rather than broadcasting it to all parts of the network.

internet: This term means a network of networks. Figure 1.4 shows how several WANs can be interconnected so that any two computers connected to any of them can communicate. This process involves installing *gateways*—computers that interconnect the participating networks. Internetting provides improved reliability: when the direct path between two networks is out, they may be able to communicate via a third network. Spelled with a lower-case i, an internet is any such network of networks. Spelled with an upper-case *I* (as in Internet), the word is the name for the worldwide network of networks that interconnect using the Internet Protocol (IP). You will learn more about IP in chapters 8 and 9.

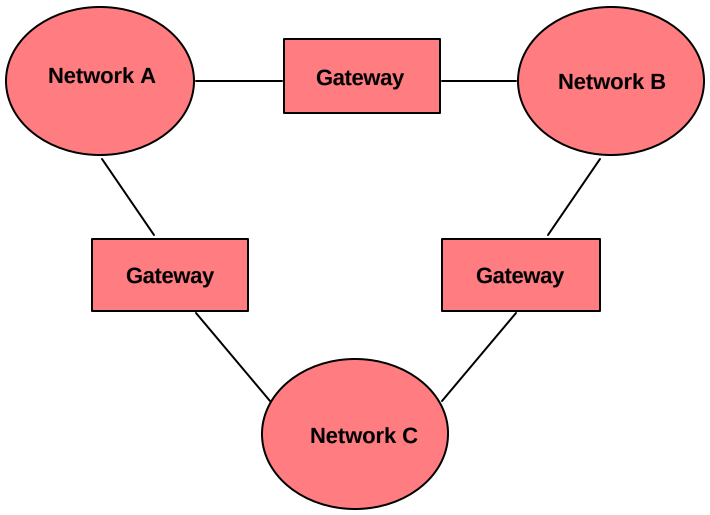

Figure 1.4 Network of networks.

Intranet: Note that a LAN can participate in an internet. At one time, most WANs interconnected large computers called *hosts*. Today it is common for smaller computers (PCs and desktop workstations, still called hosts) to be connected to a LAN, with the LAN in turn connected to a WAN router. The WAN may support a private corporate network of LANs, known as an *intranet*, or it may be part of the Internet.

The Protocol Stack

The Internet consists of many millions of computers on tens of thousands of networks. It is arguably the most complex system ever assembled by mankind. How can such a complex system function reliably, particularly when it grows several times larger every year? The answer is that the Internet is assembled from components that have been built by many manufacturers to a common set of standards. The most fundamental of these standards, the ones we consider in this book, relate to a basic set of functions that has been defined collectively by the networking industry. At the core of these functions isa set of rules for exchanging information. These rules are known as *protocols*.

Why are they called "protocols"?

Diplomats learned a long time ago that, where different cultures come together, you need rules for accurate transfer of information. For example, in some cultures shaking your head up and down means "yes", in other cultures it means "no." If you don't ensure accurate communication, you will soon have a war on your hands! The rules diplomats develop for communicating are called protocols. Because we have a similar function in networks, we use the same name for our rules.

Networking technology is highly *modular*: Its systems are divided into "chunks" of well-defined functions. We organize our study of the Internet protocols around the model that has been defined by their developers, the *Internet Engineering Task Force (IETF)*. We present this in the form of a five-layer model, shown in Figure 1.5, which separates the various functions of network communication into *layers*. Each layer has a different purpose.

Layering promotes software modularity and reuse because it facilitates creation of products that can be combined on a "mix and match" basis to provide a system solution to any networking problem. Because of the way they are layered on top of each other, the arrangement of protocols shown in Figure 1.5 is called a *stack*. One well-known stack with seven layers is called the *Open Systems Interconnect (OSI) Reference Model*. In this book, we use a simpler five-layer stack that is associated with the Internet. This stack is sometimes shown with only four layers, with the DLC and Physical layers combined into a single *host-to-network layer*. This stack is illustrated in Figure 1.5.\\

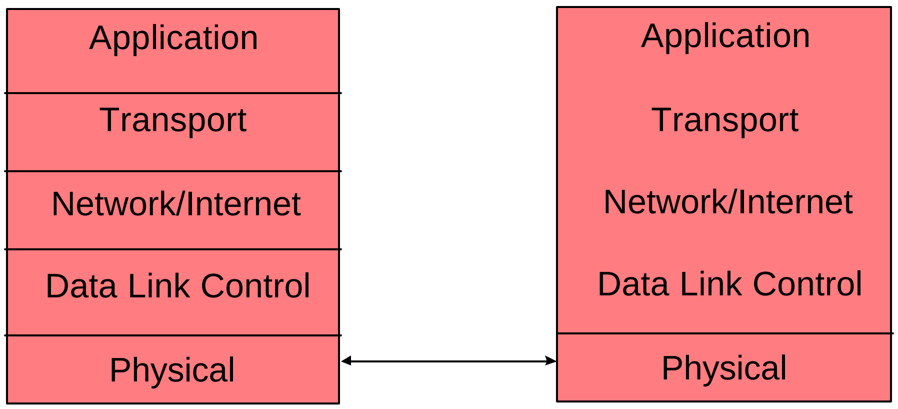

Figure 1.5 Five-layer protocol stack.

The functions of the five layers are:

Application Layer: Responsible for whatever the user wants the computer to do, such as interacting with a remote computer, transferring files, or displaying graphics obtained over the World Wide Web (which we refer to in this book simply as the Web). This interaction is achieved by sending *messages*.

Transport Layer: Responsible for packaging data for host-to-host delivery. For long streams of data, this requires dividing the information into *segments*. This layer also provides a means to identify how messages are to be used in the receiving host in that a set of messages is associated with a particular application. In many cases, this layer also keeps track of information flow between sender and receiver (which can be anywhere in the network) so that no information is lost or presented to the receiving application layer out of order.

Network Layer: Responsible for putting the segments into "electronic envelopes" called *packets* and providing the organization necessary to get the packets from sender to receiver. This process involves providing a consistent means of *addressing* (locating) the sender and receiver as well as a workable means of *routing* the packets through the communications links, routers, and gateways of the Internet. With good routing, the packets will flow efficiently and will be moved to another path quickly if problems arise.

Data Link Control (DLC) Layer: Responsible for controlling operation of a single data communication link to move information efficiently from end to end, even though the link may be experiencing transmission errors. An important function of this layer is Medium Access Control (MAC), which allows multiple computers to share a single information channel, as shown in Figure 1.3. Other DLC functions include putting delimiters around the packet to make a *frame*, detecting (and possibly correcting) transmission errors, and controlling the rate at which the sender transmits so the receiver is not overwhelmed with data.

Physical Layer: Responsible for passing information between two physical locations. In its simplest form, the physical layer is a wire. In most cases, it is considerably more complex, being derived from a larger telecommunications system that supports a variety of uses (mostly commercial telephone service).

Data units: Notice in the list shown in Figure 1.6, each layer has its own name for the unit of data it transmits. The formal term for these is *protocol data units*.

Layer	Common Data Unit
application	message
transport	segment
network	packet
data link control	frame
physical	bit

Figure 1.6 Protocol stack layers with common data units.

The layers can be identified by number, starting from the bottom. In particular, it is common to refer to DLC as "layer two" and the network layer as "layer three."

From Figure 1.5, we can see that no less than four independent software modules (sometimes more) are required for a computer to communicate over the Internet. Of these, only the application layer normally is under the user's control. The transport, network, and DLC layers almost always are built into the computer's operating system. Although not separately visible to the user, each layer has a separate function and affects the performance seen by the user in a different way. These layers are designed and specified separately, but each conforms to the general notion of a "layer" shown in Figure 1.7. Each protocol has an *interface,* or "connection point," with the protocols immediately above and below it in the stack. The function it offers to the next higher protocol layer is called a *service*. Its communication with the protocol at the same level in the stack at the other end is called a *peer connection* and is possible only if the peers use the same protocol.

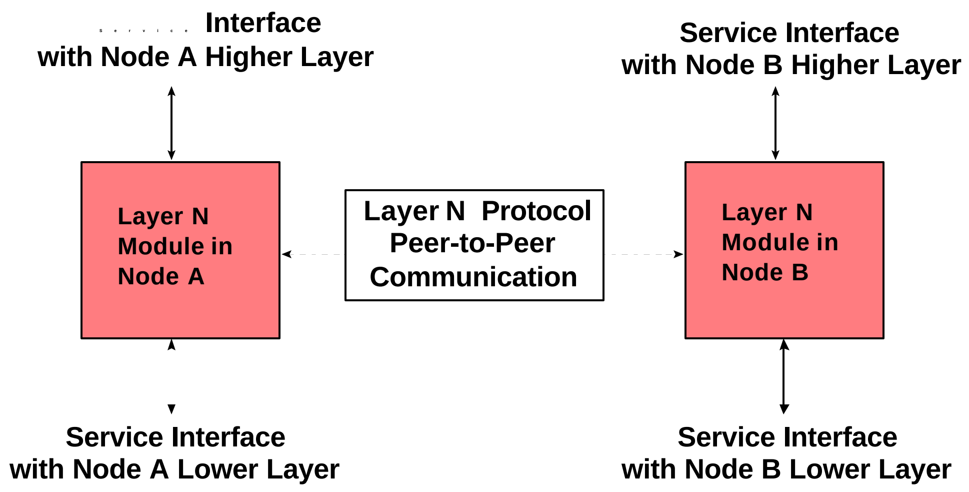

Figure 1.7 Generalized protocol layer.

Network Workbench Basics

Now we are ready to take a look at an actual protocol stack. The unique aspect of this book is the Network Workbench (NW for short). NW is *simulation* software. That means it uses a computer to model a real-world process or system in such a way that its important aspects are *abstracted*. In other words, a good simulation captures the important properties of whatever is being studied and models its behavior with regard to those properties to a good degree of accuracy. The question of what constitutes "good" is a critical one for the simulation developer. Too much accuracy slows down the simulation and may obscure the very properties being studied; too little accuracy can make the simulation misleading. Network simulators are available in a very wide range of levels of abstraction and accuracy. NW was designed to have accuracy sufficient for understanding the basic workings of protocols, without overwhelming its user with unnecessary details. NW does not have the power to depict network operations with timing accuracy less than a single bit, but it does represent the operation of each protocol in detail.

The concept behind NW is that the operation of each protocol is implemented in software that is nearly identical to that of a real network but without many of the features and details of normal network software. The intention is that those who want to understand how the protocols work can program the protocol modules of NW. In arriving at a working program, they also arrive at a good understanding of the way the protocol works. The NW protocols have been abstracted with care to expose the fundamental works of the protocols used in the Internet.

To achieve good accuracy and efficient operation, NW uses a method known as *Discrete Event Simulation (DES)*, which is useful for studying the behavior of systems over time. DES is useful with interconnected systems, such as networks, where the complexity of the system and its components makes it difficult to predict what it will do, given a particular set of inputs. The concept behind DES is to define the smallest increment of time that is required to represent the simulated process accurately. All actions in the simulation are then described in terms of this unit (in NW it is called a simulation time "tick). It is important that this increment be small enough that using it as a basic unit does not mask fine details of the behavior of the system but still large enough that the count of ticks for the total time simulated can be represented easily within a long integer. The default tickin NW is 100 nanoseconds (10^{-7} seconds). Using this tick size, a time duration of 200 microseconds would have the following value:

```
200E-6/1E-7 = 2000 ticks
```

Once the basic time increment has been selected, a discrete event simulation is programmed to keep track of all actions by system components. Typically, the result of an action by one component triggers an action by another component. For example, a message sent by the application layer in the protocol stack triggers action in the transport layer, which in turn triggers action in the network layer, etc. The length of time each event requires can be predicted with good accuracy. For example, if we are sending a 1000-bit packet on a 100 kbps link, we know it will take 10 milliseconds (.01 second) for the packets to be transmitted, so we set up the DES to produce an event indicating transmission is complete in 100,000 ticks = .01 second. At the heart of NW lies DES software that keeps track of all the events "waiting to happen." The code for this is in module `des.cpp`.

About Typography

Computer file names, program algorithms and C/C++ program code are identified in this book by a non-proportional typeface, for example the NW header file is `code/nw.h`. Important concepts are identified by **bold face type**, and words with Glossary entries are identified by *italic font.*

Built around the DES function, NW has a five-layer protocol stack like the one in Figure 1.5. Only one accommodation is needed to make the simulation work: Instead of directly invoking the next layer down, each layer passes the function call through the DES. In addition to providing for appropriate amounts of delay in the simulation, this process also provides a place for NW to print a trace as the simulation proceeds and collect statistics about the operation of all aspects of the network.

NW provides five basic services for network simulation. In the next few chapters, you will learn how to take advantage of these:

- DES, as discussed above
- All software required to make the five-layer stack work
- A set of input files representing networks and electronic mail (also called *email*) data, with functions to show the values being used at the beginning of the simulation
- A user-selected trace of actions happening in the simulated network as time proceeds
- A summary of network performance statistics at the end of the simulation

NW Files

Whether you download NW over the Internet or use the copy on the CD that accompanies this book, you will receive the same collection of files. If you download, you can select a version for your particular computer and compiler, whereas the CD contains several versions. In either case, you will find the files are organized into directories or folders, where `xxx` stands for a supported system—for example, `sun` for Sun Microsystems and `bcb` for Borland C++ Builder. For each version, the top-level `nw` directory contains the following directories plus some general descriptive files such as `nwdesc.txt`, a general description of NW:

`Code`: Contains C++ code for all available modules and stubs for student solutions
`Data`: Contains the `email` files and network descriptions used by NW
`object.xxx`: Contains `xxx`-compiled versions of everything in `code` plus NW solutions

`wkb.xxx`: Contains any special software plus instructions for system `xxx` (you should look at `xxx-instr.txt` in particular)

The NW Header File nw.h

It is customary to organize large C and C++ programs into modules and to create a common *header file* that defines the data structures and C++ classes used across the collection of modules. The Workbench has been developed in this way, with all key definitions in file `nw.h`. A copy of the header file for NW version 4.0 is included in this book as Appendix C. The file is organized as follows:

- Narrative description of the Network Workbench (NW). Brags, boasts, disclaimers, warnings, and advice.
- Compilation constant definitions. Names associated with constants to be used when the compiler generates NW. For example, `#define MAX_MSG_SIZE 101` defines the size of the largest message NW will support, which in turn is used in the data definitions that follow.
- Global NW type definitions. Data types that are used in the various NW classes. For example, `message` defines a structure with six header bytes (`size`, source_`net`, etc.) and a text field of size `MAX_MSG_SIZE`. Types are not actual program variables; rather, they describe generic data structures that are used by the classes that follow.
- NW Class definitions. This is where C++ becomes important. A C++ *class* defines a collection of data elements associated with an instance of the class and the functions that manipulate those data elements. An important property of classes is *inheritance,* which means one class can pick up all the properties of another *base class* and add its own properties. The C++ functions in the Workbench create and use instances of these classes, which are called *objects*. To complete the projects in this book, you will not need to write any classes of your own. You can use the classes in `nw.h` to complete all of the projects. In effect, this means that you can do the projects if you know the C language, because the C++ part is all provided for you.

Finding Things in `nw.h`

The header file `nw.h` is sizable-- over 60,000 characters. Finding the particular definition you are looking for can take a long time if you must read clear through the file. Fortunately, modern programming environments include an editor with search capability, making it easy to find what you are seeking. For example, you might want to find the definition of `packet` used in NW. To do this you would load `nw.h` into the text editor program on your computer and invoke the "search" function. Because `packet` is used in the narrative and constant sections of the header file, you would need to search forward through about a dozen instances of `packet`, but after a few clicks you would come upon what you need: the data type `packet` with eight bytes of header and `segment` payload.

Hands-On Activities

Every chapter in this book contains one "homework" problem, which may require use of the computer but does not entail a significant programming effort. Each chapter also contains one project, which in most cases will require a few hours to complete, depending on the skill of the reader. The problem and programming assignment for this chapter follow.

Problem: Identifying Stack Modules in NW

The “master index” to the Network Workbench is in file `code/nw.h`. This is a C/C++ “header” file that defines all of the constants, data types, and C++ classes (functions and associated data structures). Your assignment is to become familiar with `nw.h` by reading it. In the process of reading, you are to produce a one-line description of each class and write out the names of the major software modules representing the five layers in the NW stack.

Project: Loading and Running NW

1. Run the appropriate NW setup program to create a working directory of files for the Network Workbench. If you intend to load NW onto your own computer, you will have to do that first. See Appendix B for instructionsYou can find the setup process and the directions for running it in `nw*/wkbxxx`, where `*` is the current version of NW, and `xxx` is the system you are using (for example, nw40/wkb.sun). The directions will be in `xxx-instr.txt`Program `setupn` will load program and data files appropriate to your computer and compiler. Here, `n` identifies the size of the network to be used. To start, you should run `setup7`, which will load a seven-node network.

~~3.~~2. After completing the setup, run the built-in NW solution to project DLC1. To do this, go to your working directory and edit program `dlc1.cpp`, which stands for “NW project DLC1, C++ code.” The edit you should make is to insert two slashes (`//`) at the beginning of the line that contains:

```
#include “stuff.cpp”
```

This makes the line into a C/C++ comment, so it is ignored by the compiler and is called “commenting out” the line. Now compile and run `dlc1.cpp`. The output will pass by quickly on your screen, but it will also be written into file `working/diskout.txt`.

3. Now load `diskout.txt`. in a text editor program to analyze it. You should be able to identify:

- The various NW software modules loaded
- The simulation parameters to be used
- The nodes and links in the network
- The operation of this simulation, in which the application sends only one message
- The flow of data between layers, down the stack at the sending node from application to DLC layer, and back up the stack the receive node
- The statistics at the end of the simulation, which in the case of DLC1 show that only one frame was transmitted and one frame was received