Stanislav Shalunov

How can an internet work
and how does the Internet work

MCCME
Moscow, 1998

Contents

1 Introduction

We all use the net and the basic services it provides (E-mail, web, FTP) without thinking much about how all this stuff really works. For many people this attitude is completely normal. After all, who cares what happens when you pick up a phone and dial a foreign number—the only thing that's important is that you can talk!

Some others, however, need to understand how this thing works. Their attitude pays off when it comes to the net. Because there are so many diagnostic messages you can encounter, because there are so many different options in various programs, and because it's one of the most wonderful things in modern technology. Do you know how your E-mail gets to Australia? Try to ask someone who is *technically competent* questions about it. Most probably, in response you'll get either tech-speak you don't understand or a "why do you want to know it?"

Cookbook recipes really don't work with the Internet. You have to understand the basics or you'll remain a layman for the rest of your life.

The purpose of this book is to give an understanding of how the Internet works. We won't discuss formats of packets and such exact technical details. We'll be just gaining an understanding.

Our method will be to take one simple action and follow exactly what events it generates in the net. We have chosen an E-mail delivery as such a model event, because E-mail is still the basic service the Internet provides, and because E-mail is such an important service.

1.1 How to read this book

This book can and should be read from the beginning to the end. Some stuff can be skipped in the first reading. It will be identified as such.

Here's a brief overview of contents.

In section 2, p. 9 we discuss *Ethernet,* something that is commonly used for Local Area Networks (like an office network). Section 3, p. 12 is dedicated to initial description of the *Internet Protocol* (IP); it defines an internet. *Internet Control Messages Protocol* (section 4, p. 25) is used to control and diagnose IP networks. *User Datagram Protocol* (section 5, p. 28) is a protocol that can actually be used (and is used in real life)

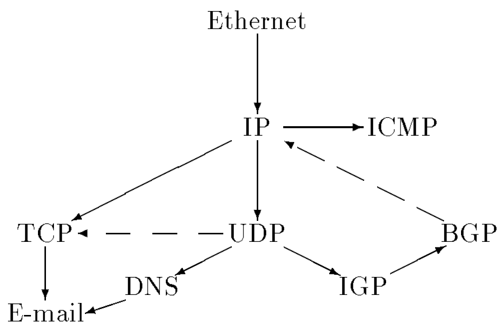

Figure 1: Graph of dependencies of sections.

to transfer useful data. Section 6, p. 30 discusses how can we instruct hosts on a small network to deliver packets to the right places. TCP is one of the most well-known networking acronyms; we'll tell you what it really means in section 7, p. 38. DNS is another well-known acronym; it stands for *Domain Name Service;* we speak about it in section 8, p. 44. Section 9, p. 49 can finally explain how E-mail works, but the real problem of getting packets to their destinations in a very large network is discussed only in section 10, p. 57.

The graph of dependencies of sections is on fig. 1. We supply an index, which can be used to look up Internet-related terms.

1.2 Why and where to get the RFCs

We give qualitative descriptions of various Internet mechanisms and protocols. The definitive description, the standards, are the Internet *Requests For Comments,* or RFCs. They all have numbers, and are commonly referred to by them, e.g., RFC 822 [?] describes the format of E-mail headers. We provide citations for RFCs and an extensive cross-referenced RFC bibliography. You can read them, too. Many RFCs are available online. They can be obtained via anonymous FTP from `ftp.internic.net` from `/rfc` directory under the name `rfc`*XXX*. The list of RFCs is available as `rfc-index.txt`. Many sites on the Internet mirror the RFCs, so they are widely available. HTML version of the index file with links to full text documents is available at *ftp://ftp.math.utah.edu/pub/tex/bib/rfc.html*.

Readers are encouraged to study selected subjects they are interested in by reading the RFCs.

1.3 Acknowledgments

The following people helped me by reading drafts, making useful suggestions and pointing out errors: Marina Feigelman, Dima Kaledin, Serge Lvovski, Alexander Shen. I am especially grateful to Toby Axelrod who has worked hard on improving my English. The cover picture is drawn by Marina Feigelman.

Misha Panov has helped me with figures. Vadim Radionov provided very valuable help with LaTeX.

This book is by and large meant as lecture notes for my short course in IUM; without all the questions from my students I wouldn't be able to write it.

2 Thin Ethernet as an example of the physical layer

Before any networking can take place we have to have some physical way to transfer data. Naturally, we won't consider all the ways that are actually used. We'll consider only one in moderate detail: namely, the so-called *thin Ethernet.*[1]

This choice was made because Ethernet is not too hard to describe; because it is used in many places, so you are likely to encounter it; and because we had to choose something. We'll be describing *thin* Ethernet. It is commonly referred to as 10Base2[2], and uses thin coaxial cable. There are other species of this beast, the most common being 10BaseT, which uses phone-looking wiring (twisted pair) and phone-looking 8-pin connectors (RJ-45); all Ethernet modifications are similar to one another. Thin Ethernet is simpler than, say, 10BaseT, because it does not use *hubs:* devices talk directly to each other. Later on, we'll be just using the term *Ethernet,* always meaning 10Base2 (thin) Ethernet.

In Ethernet, all devices are connected to one coaxial cable. Each device can send and receive electrical high-frequency impulses that bear signal. Both ends of the cable are equipped with *terminators:* 50 Ohm resistors; terminators are used to suppress reflection of impulses (reflection would interfere with signal being actually sent). Devices are connected to the cable through resistors, too. Devices connected to a typical Ethernet are Network Interface Controllers, or NICs; they are also called Ethernet cards. Naturally, NICs are not all alone, but installed into something like a computer, or a printer, or whatever (the nature of the host is not essential at this point).

Every device can send and receive *packets.* Packets are chunks of data Ethernet can pass. A typical length of an Ethernet packet is between 100 bytes and 1.5 K.

[1] In reality, the media that keeps the Internet running and is used by major backbones is *optical fiber;* we won't consider it because we want to deal with something everyone has seen and is familiar with.

[2] In this magical incantation "10" stands for 10Mbps (there's also a faster version of Ethernet, 100Mbps), and "2" describes the media: coaxial cable that only has two wires.

2.1 Media Access Control (MAC) addresses

Ethernet cards are assigned *Media Access Control (MAC) addresses* (or hardware addresses) at factories. A MAC address is 6 bytes long. No two Ethernet cards have the same hardware address. This is achieved by assigning three-byte prefixes to manufacturers.[3]

MAC addresses are used to communicate within an Ethernet. Every Ethernet packet starts with a fixed-length header.[4] This header is called the *link-level* header. It includes sender's MAC address and recipient's MAC address. All the NICs see all the packets in an Ethernet. However, any given NIC passes to the computer only the packets that are addressed to its MAC address.[5]

2.2 Collisions

The problem we encounter is that two (or more) NICs may want to start talking simultaneously. This situation is detected by NICs, and is called a *collision.* It is strongly suggested that you try to resolve the problem of collisions yourself before reading any further, and then compare your solution to that commonly used.

The first approach would be to make both NICs wait for some fixed time (say, 1 millisecond) and then retransmit the packet. However, we gain nothing: collision will repeat.

A more elaborate idea would be to assign to each NIC on the Ethernet its own delay-after-collision time. This would solve the problem, but very inefficiently: the administration overhead is too heavy, and the delays may be unnecessarily long.

Another idea would be to make an NIC reserve a right to speak and then send the packet. This isn't a solution at all: just *how* do we reserve this

[3]Some careless pirate vendors can steal someone else's prefix and use it for their own NICs thus creating the potential of two NICs with the same MAC address. However, I have never heard anyone complaining about having two NICs with identical hardware addresses.

[4]We won't describe the exact formats of any packets, headers, etc. Every time we need to mention a header we'll only mention fields we need at the point. It does not mean there are no other fields.

[5]Unless the interface is in *promiscuous mode,* which allows to grab all the packets, e.g., for auditing. See also section 2.3, p. 11.

right? The only way is to send some packet. But what do we do if we get a collision sending these reservation packets?

The solution that is used does not prevent collisions. In fact, collisions are normal events. Many NICs have an LED marked COL, which will blink red when a collision is detected. NICs that detect collision just wait for a *random* time and then retransmit the packet.[6]

2.3 Ethernet broadcasting

There is one MAC address reserved for special purposes, which consists of all ones (or `ff:ff:ff:ff:ff:ff` in the traditional hexadecimal notation for MAC addresses). Any packet sent to this address will be passed to computers by all NICs. Packets addressed to this MAC address are called *broadcast* packets, and the act of sending such packets is called *broadcasting.*

[6] Another question is, "Where do we get the random numbers?". If we know what a pseudo-random number generator is, we'll immediately answer: make the time pseudo-random. That's what most manufacturers do. And the MAC address is used as the seed. If you don't know what a pseudo-random number generator is, here's an example: take any 16-bit integer as the seed; to get the next pseudo-random number, square the previous one and take the middle 16 bits of the resulting 32-bit number.

3 Internet Protocol (IP)

We have addresses (MAC addresses) and we can start networking. A set of computers and other devices connected with an Ethernet is usually referred to as a *Local Area Network,* or a LAN.

A LAN can only have a limited number of hosts (in practice, no more than a hundred or so). If we put too many hosts on the same LAN we'll get too many collisions and the networking will become extremely slow. There are also concerns with the physical length of the coaxial cable.

Our goal is to build a very large *Wide Area Network* (WAN).

So, it looks natural to connect several LANs together. We'll start with two LANs. We can install into the same host *two* NICs, and make the host participate in both LANs. We can also instruct this host to pass packets from one LAN to another, thus acting as a *gateway.* The entry points for packets into hosts are called *interfaces,* so in this case the gateway will have two interfaces. Generally, a host that connects two or more networks together by forwarding packets from one into another is called a gateway. We shall sometimes use the term *router,* which has essentially the same meaning.[7]

Hosts with a single interface might send all the packets to the gateway (every host will know the MAC address of the interface of the gateway that belongs to its LAN). Gateway will know what to do next.

The disadvantages of this approach are obvious:

- When two hosts on the same LAN want to talk they have to do it through the gateway. Thus, the traffic is doubled.
- The gateway has to remember all the MAC addresses of all the hosts on the network.

This second disadvantage is very important. We must think about its implications. When our network grows and includes possibly hundreds of thousands of LANs, all the gateways will have to remember some information about *each host* on the network. More than that, when

[7]The difference is that a router can do more than a gateway. All gateways are routers. However, a router can, e.g., somehow embed IP packets into other IP packets; a gateway doesn't do this: the only thing a gateway can do is forwarding packets.

new hosts are added, the information about them has to reach all the gateways.

Now it became apparent that MAC addresses are not suitable for building a large network. We have to use some other addressing scheme. The disadvantage of MAC addresses is that they are unique, but random. There's nothing similar in MAC addresses of hosts that are near each other.

But, we have no other addresses. So, they have to be made up. We'll build a different protocol on top of Ethernet, and make it scalable.

The *Internet Protocol,* or IP [?] uses 32-bit (4-byte) addressing; the addresses are assigned to make *routing* (getting packets to their destinations) easier. These addresses are traditionally written in the form of a dotted quad, like 10.253.12.57 or 127.0.0.1; each number in the quad is the decimal value of the corresponding byte in the IP address.

3.1 IP Local Area Network

First thing we have to do is to explain how the Internet Protocol will work in a single isolated LAN. So, there is only one LAN, and we want to use IP in it.

3.1.1 IP numbers: new addressing scheme

In an IP network (also called an *internet*) all interfaces are numbered with 32-bit integers. These numbers are called *IP numbers,* or IP addresses. It should be stressed that it is interfaces that have IP numbers. Not hosts. Not users. Not programs. A typical host, however, (say, a workstation, or a PC, or a printer) only has one interface and therefore one IP number.[8] So, in an internet, hosts that have only one interface are often identified with their IP numbers.

Since everything is on the same LAN we don't have to worry yet how we should assign the IP numbers. This can be done arbitrarily.

[8] As usual, there are exceptions to this rule. For example, one physical interface can have more than one IP number; this is done by creating *virtual interfaces* in the operating system. And *vice versa,* in a more esoteric setup the same IP number can be served by many computers if a certain *load balancing system* is used. We won't consider these pathologies. One interface—one IP number.

3.1.2 IP header; IP over Ethernet

Every IP packet has a fixed-length header. Header starts the packet. It includes, in particular, sender's IP number and recipient's IP number.

An IP packet can be embedded into an Ethernet packet. Since the Ethernet packet has its own header, the IP packet will be the body of the Ethernet packet.

Later on, we shall often encounter such embedding: UDP header will start the body of an IP packet, etc.

3.1.3 Address Resolution Protocol (ARP)

Once we have assigned IP numbers we have to know how they correspond to MAC addresses in order to actually communicate. We have to embed the IP packets into Ethernet packets. The process of figuring out MAC address by an IP address (within a LAN) is called *address resolution.*

It is a challenging problem to do address resolution robustly and effectively. You are strongly encouraged to think about it before you read the solution.

The ideas that come to mind first are:

- Have a table of correspondence of IP numbers to MAC addresses on each host.
- Have such a table on one "smart" host and teach all the other hosts its MAC address.

The first solution is an administrative headache. Adding new hosts is still a hassle. The second solution doubles traffic within the LAN.

What is actually used is *Address Resolution Protocol,* or ARP [?]. Initially, hosts don't know any MAC addresses. A dynamic table of correspondence of IP numbers to MAC addresses is maintained. Each time a host needs to send a packet to a given IP number, this number is looked up in the table. If the table has the MAC address, it is used. If the table does not have the MAC address, ARP is used to find it out.

ARP employs the following procedure to find a given IP number's MAC address: a special broadcast Ethernet packet (see section 2.3, p. 11) is sent, *ARP who-has* request. This packet is not an IP packet. Notice that ARP packets are the only non-IP Ethernet packets that we shall ever encounter. The link-level header contains our MAC address as the sender's address and the broadcast address as the recipient's address. The body of packet identifies it as an ARP who-has request, and says which IP number we are interested in. The owner of this IP number will see the packet and respond to us with an *ARP is-at* packet saying what is its MAC address.

Once we know the MAC address, it is entered into the dynamic table.[9] After some time (variable, depending on the operating system) this entry will be deleted.

Some operating systems optimize lookups of dead hosts by entering an invalid MAC address consisting of all zeros for IP numbers that do not respond to ARP who-has requests. This way when a packet has to be sent to a dead host the system can generate an error promptly rather than sending ARP requests again and waiting for a timeout.

3.2 Internetworking in several IP LANs

Once we understand how IP works in one isolated LAN we can use it to join several LANs together. Remember, that was the purpose of introducing the new scheme of addressing.

3.2.1 Two LANs and a gateway

For starters, we'll connect two LANs together with a gateway. We have introduced IP numbers because MAC addresses were not regular enough. So, this time we'll use the fact that we can assign IP numbers any way we want to make it easy to decide which LAN an IP number belongs to just by looking at the number.

One way of doing this is to make the first three bytes of IP numbers in one LAN, say, 10.0.0, and in the other LAN, say, 10.0.1[10]. Gateway will have two NICs and will be present in both LANs. Each interface of the gateway will have its own IP number. For example, it might be

[9] In Unix, one can view the ARP table with `arp -an` command.

[10] Last byte is arbitrary, so we have up to 256 hosts on each of the LANs. In fact, two values—all ones and all zeros—are reserved, so we can have up to 254 hosts.

10.0.0.1 and 10.0.1.1. Each host will know its own IP number, and the IP number of the interface of the gateway in its own network.

When a host (that is not the gateway) needs to send an IP packet, it looks at the destination address. If the first three bytes coincide with that of host's own IP number, destination is on the local LAN; in this case packet is sent directly. If the first three bytes aren't the same as that of host's own IP number, destination is not on the local LAN; packet is sent to the gateway.

Gateway acts as follows: if the first three bytes are 10.0.0, send the packet to the 10.0.0 LAN through 10.0.0.1 interface; similarly, for 10.0.1.

Voilà, all the hosts can communicate with each other. We should mention that when the gateway is down hosts on the same LAN still can communicate (but naturally, not hosts on different LANs).

3.2.2 IP address + netmask = network

The way we used to distinguish which LAN an IP number belongs to is used further [?]. Traditionally,[11] an Internet *network* (the term *subnet* is often used in a very similar fashion) is characterized by a fixed prefix. Say, all addresses that start with 10 form an (Internet) subnet. Similarly, for addresses starting with 192.168.[12]

It is natural to use such blocks of IP numbers (subnets) for such purposes as giving IP space to an organization, a department or a lab.

In the first reading you go to section 3.2.3, p. 17 at this point.

So we have the following meanings of the word *network:*

- A set of computers that can send packets to one another. (Such a network can include many LANs and gateways.)
- A Local Area Network, or anything that is a unit of network in the previous meaning.
- A set of IP numbers as defined in this section.

[11] Later, particularly in section 10, p. 57, we shall provide more information on *classless routing,* which changes this scheme.

[12] These examples mention network numbers reserved for *private networks* [?] and they cannot be actual IP numbers you can see for, e.g., web servers.

It is important to not confuse these meanings. Unfortunately, the use of the term *network* can often lead to confusion. We shall try to be as clear as possible as to what we mean.

To be more precise about this last meaning: Every IP network has an IP number and a netmask. Both these numbers are 32-bit integers. Netmask has a special property: it consists of some number of ones followed by zeros (i.e., 255.0.0.0 or 255.255.255.0 are valid netmasks while 17.18.19.20 is not). An IP number is said to belong to a given network if and only if the result of logical bitwise AND of the number with netmask is the IP number of the network.

For example: an IP network with IP number 12.13.0.0 and netmask 255.255.0.0 includes all the IP numbers that have first two bytes 12.13. This network is traditionally referred to as 12.13.0.0/255.255.0.0.

3.2.3 Class A, B and C networks; traditional routing

A set of IP numbers that have some fixed value of the first byte is called a *class A network.* A set of IP numbers that have fixed values of the first two or three bytes is called a *class B* and *class C* networks, respectively.[13]

There are different ways to do routing on a host. Generally, there are three common ways: traditional Unix, modern Unix, and Windows routing. We shall consider the first method.

Routing considered in this section is called *static* (as opposed to "dynamic" routing controlled by a routing protocol, like the ones described in sections 6, p. 30 and 10, p. 57). Static routing is set up manually by an operator or network manager.

Routing table Each host has a *routing table.*[14] This table maps networks[15] to gateways. For example, it can say that class B 10.6 network should be routed to 10.0.0.3. This means that if we need to send a packet to an IP number starting with 10.6 it should be sent to 10.0.0.3

[13] Some information on how class A networks are used in the modern Internet is provided in [?].

[14] In Unix one can view the routing table using the `netstat -nr` command.

[15] It is also possible to have entries for individual hosts. We shall never use host routes.

(which presumably knows what to do with the packet further). Notice, that we have to know how to send packets to 10.0.0.3 directly; otherwise, this route won't work. So, we must be on the same LAN as 10.0.0.3. Also notice that having the same network listed twice or more with different gateways is OK. Each route (by route we mean here an entry in the routing table) also has an attribute called *metric* associated with it; metric is a small integer. We do not explain this entry until section 6, p. 30; everywhere before this section metric is 1 for all routes and is not used in routing decisions.

The routing table may have one special entry, the *default route.* This route is used to handle packets that don't match any specific routes.

Routing algorithm The routing algorithm works as follows. Suppose we need to send an IP packet. The destination is matched against all the non-default entries in the routing table. If any matches are found, the most specific match will be used; that is the longer match: class C matches are preferred to class A matches. If several equally specific routes are found, the one that will be used is chosen *randomly.*[16] If no matches are found, the default route is used.[17]

Usage of this routing scheme Leaving apart the question of routing complexity and managing this mechanism allows connecting LANs together. For a small number of LANs, or for simple network topology (network topology is the way LANs are connected together) this way of routing is even efficient.

It should be pointed out that the only information used when deciding where to send an IP packet is the destination IP number. All the routing on the Internet (including modern most advanced schemes) is still this way.[18]

[16] Some implementations of IP allow specifying *priorities* to routes. These priorities can be used to change the probabilities of using alternate routes. In section 3.2.6, p. 20 we'll see why this might be useful. We choose a model without routing priorities.

[17] If we would allow the whole Internet in the routing table the picture would become even more consistent. The default route is simply the route for the whole Internet. If more specific routes are found, they are used, following the general rule: always choose most specific route. Considering the default route separately is just a tradition.

[18] It should be noted that this does not contradict the fact that packets from your organization and from another organization having the same Internet Service Provider

3.2.4 IP broadcasting

In section 2.3, p. 11 we have introduced Ethernet broadcasting. In the Internet Protocol, there's a similar notion [?, ?, ?].

Namely, on each LAN there's a reserved address: e.g., for class C 10.17.23 network it shall be 10.17.23.255.[19] Packets sent to this address are broadcast over Ethernet (using Ethernet broadcast address as destination MAC address). Operating systems shall serve these packets as if they were for this host's IP number.

IP broadcasting is used mostly by routing (see, e.g., section 6.1, p. 30) and booting protocols (booting protocols allow to have hosts without disks on the network).

It should be noted that the address 255.255.255.255 is somewhat special. In theory, it is the broadcast address for the *whole Internet.* In reality, no gateways shall forward packets for this address. So, this address is a good way to reach all the hosts on a LAN. It is mostly used by booting protocols.[20]

IP broadcasting can work not only over Ethernet but over other network media as well [?].

3.2.5 Two LANs and a channel

We are always talking about LANs and gateways. However, it is obvious that in real life we'll need *circuits,* long channels connecting remote LANs. We won't discuss how such point-to-point links work. One way to transmit data is via regular modems (of course, Internet backbones use different media!). In a way somewhat similar to that used for Ethernet, IP packets can be passed over such modem links (one of the possible protocols used for this purpose is the *Point-to-Point Protocol,* or PPP [?]).

We shall not describe how point-to-point links work. What's important is that IP packets can be passed through them and they are in many

(ISP) and destined for the same IP address can take significantly different routes. This is easy to understand if you keep in mind that packets hit different routers at your ISP from the beginning.

[19] More precisely, the broadcast address for an IP network is obtained by logical bitwise OR of the IP number of the network with the negation of the netmask.

[20] But you should try the command `ping 255.255.255.255` in Unix.

respects similar to a LAN. However, they are much easier to work with than a LAN: it's always clear where to send data.

In fact, we'll see that they are not essential for our purposes. Indeed, consider two remote LANs that need to be connected. In each LAN there will be a host that is connected to the modem going to the remote location. It will act as a gateway.

From a given LAN's perspective, nothing will change: packets that earlier had to go to the gateway still go there. No-one noticed anything.

The gateways are configured slightly differently, but the difference isn't too great. Each gateway acts as follows: packets destined for local Ethernet are sent there; packets destined for the opposing end of the point-to-point link are sent there; and packets destined for the foreign LAN are sent to the opposing end of the point-to-point link.

Having mentioned point-to-point links and described how they will work for two LANs we understand that they don't change the picture significantly. We'll now forget about these links for a while, and only talk about gateways.

Another (and better) way to get rid of special consideration of channels is to consider them as LANs. Simple LANs they are, but still technically LANs.

Later on, we will be always following the second approach, treating point-to-point links just the same way we treat LANs (point-to-point links are LANs that happened to have only two hosts).

3.2.6 Three LANs; transit LAN

Similarly to the way two LANs were connected in section 3.2.1, p. 15 we can connect three LANs together. Let us assign numbers to LANs: 1, 2 and 3, so that we can refer to them easily.

For three LANs, there can be two different network topologies: linear (LAN 1 connected to LAN 2, which is connected to LAN 3) and triangle (each LAN connected to each).

We shall consider these topologies separately.

Linear topology For linear topology, there will be two gateways. The situation is essentially the same as in section 3.2.5, p. 19, but we'll describe the exact routing anyway.

All the hosts on LAN i will have IP numbers in 10.0.i class C network. Gateway between LAN 1 and LAN 2 will have IP numbers 10.0.1.2 and 10.0.2.1. Gateway between LAN 2 and LAN 3 will have IP numbers 10.0.2.3 and 10.0.3.2.

Routing table of a (non-gateway) host on LAN 1 (e.g., 10.0.1.17) will be as follows: send 10.0.1 directly; default route goes to 10.0.1.2. Similarly, for LAN 3. Routing table of a (non-gateway) host on LAN 2 will be as follows: send 10.0.2 directly; send 10.0.1 to 10.0.2.1; send 10.0.3 to 10.0.2.3.

Routing table of the gateway between LAN 1 and LAN 2: send 10.0.1 directly; send 10.0.2 directly; send 10.0.3 to 10.0.2.3 (through 10.0.2.1 interface). Similarly, for the other gateway.

With these routing tables, all the hosts can communicate with each other. LAN 2 carries traffic between LAN 1 and LAN 3, thus acting as a *transit LAN.* (A transit LAN is a LAN that carries traffic that is neither originated at or destined for the LAN.)

Notice that when the gateway between LAN 1 and LAN 2 fails (goes down for whatever reason) LAN 2 and LAN 3 can still communicate normally.

In this network topology there's no other way to do routing, so the routing we have is ideal.

Triangle topology For triangle topology, there will be three gateways, one for each pair of LANs. All the hosts on LAN i shall have IP numbers in 10.0.i class C network. Addresses with last byte being 1, 2 or 3 are reserved for gateways. The rest is assigned to hosts arbitrarily. The gateway between LANs i and j will have last byte of the IP number of the interface connected to LAN i equal to j and *vice versa.* For example, gateway joining LANs 1 and 3 shall have IP numbers 10.0.1.3 and 10.0.3.1. IP addresses 10.0.i.i for $i = 1, 2, 3$ are not used in any way. Routing table of a (non-gateway) host of, e.g., LAN 2 will look as follows: send 10.0.2 into the locally attached LAN; send 10.0.1 to 10.0.2.1; send 10.0.3 to 10.0.2.3 (no default route). Routing table for (non-gateway) hosts on other networks will be similar.

Routing on gateways is not much harder to do; an interesting aspect of this situation is the fact that it is natural to have two different routes for the same network. Gateway connecting LANs 2 and 3 will have the following routing table: send 10.0.2 directly (through interface 10.0.2.3); send 10.0.3 directly (through interface 10.0.3.2); send 10.0.1 to 10.0.2.1 (through interface 10.0.2.3); send 10.0.1 to 10.0.3.1 (through interface 10.0.3.2). Routing on the other gateways is similar.

Let us study fault tolerance of our small network. Our network has certain *redundancy;* packets can (potentially) get to their destinations via more than one route. A mathematician would say that the graph formed by LANs is not a tree. Theoretically, our network might be able to continue to function even if one gateway goes down. However, with the routing we have described, this won't happen.

Indeed, suppose one of the gateways (e.g., gateway between LAN 1 and LAN 2) fails. In this case LAN 1 can communicate with LAN 3, and LAN 2 can communicate with LAN 3, but hosts on LAN 1 and LAN 2 cannot exchange packets. It would be nice if we could teach hosts to pass packets via an alternate route if the primary route is down; however, the mechanism of routing by routing tables doesn't allow such things.

What this mechanism does allow is sending packets for each LAN via a random path not only on the gateways, but also on the hosts (so, host 10.0.1.17 would have the following routing table: send 10.0.1 directly; send 10.0 to 10.0.1.2; send 10.0 to 10.0.1.3). At first glance this approach might look promising. However, it is wrong. Look:

- When everything is OK, packets for a foreign LAN are transmitted three times rather than two in 1/2 cases, thus increasing traffic.
- When the gateway between LANs 1 and 2 fails, the probability of a packet getting from LAN 1 to LAN 2 is only 1/4; the probability of sending a packet and getting a response is only 1/16; it's difficult to talk when delivering packets is an exception rather than the rule! More than that, the probability for a packet to get from LAN 2 to LAN 3 (which are connected with a perfectly operational gateway) is 1/2, while the probability for a packet going in the other direction to reach its destination is 3/4, so we even have asymmetry here.

- When two gateways fail (with, e.g., the gateway between LAN 2 and LAN 3 functioning) LAN 1 becomes isolated, and probability of a packet to get from LAN 2 to LAN 3 or *vice versa* is 1/2 (while nothing is damaged between these LANs).

So we have to have a way to know what routes are working. We'll provide one such way in section 6.1, p. 30.

3.2.7 Star topology: centralized routing

Star topology is a topology where there's one central network object (we consider LANs as such at this point) and all the other network objects are connected to this central one.

Hosts on all the "client" LANs normally send local traffic locally and have a default route pointing to their connection to the central LAN. All the "smart" routing is on the central LAN. There are two ways to do it there: Each of the gateways knows where to send all the traffic, or it sends traffic to one central router on that LAN, the only host that knows all the networks. In reality, only the first of these two methods is used.

When a gateway crashes, the topology of the rest of network doesn't change, while its LAN becomes isolated. We do nothing in case of a crash, and there's no need to do anything.

3.2.8 Tree topology: hierarchical routing

Another approach would be to organize network objects in a hierarchic fashion. There would be a "main network commutator," which would distribute packets amongst regional networks that will pass packets down to large organizations and service providers who are responsible for distributing traffic amongst LANs.

This way of organizing the network was considered natural when there were no extremely large networks. Later on, problems were identified that prevent using tree topology with simple static routing even for single organizations.

This approach works fine for small networks, but it already doesn't scale to, e.g., a large campus network.

Problems with tree-like organization are:

- When the "main center" is down or unreachable the entire network becomes inoperational.
- There's no good way to introduce redundancy.
- Switching of a network object to a different upstream provider usually requires renumbering all its networks and all the networks of downstream neighbors.

In real life, network topologies are planned to allow certain redundancy.

3.2.9 Redundant vertices

What we could do to improve the scheme with tree-like topology is the following: Add more top-level packet distribution centers and send packets that need to go upstream from a second-level routing center to a random top-level center.

This is a dead-end approach: it doesn't allow us to use backup links when we want them. It doesn't allow us even to avoid sending traffic into links that are dead.

3.2.10 How do we know if a channel is down?

All the tricks that we would try to introduce to improve network scalability and robustness are in vain until we can do smart routing decisions. We could not even use redundancy in our model three LANs network (section 3.2.6, p. 20).

We have to have a way to know if some particular gateway or link is down. Our current protocols provide no way of learning any such information. It is natural to use IP to distribute such information. It might look slightly strange: IP packets will control decisions about where other IP packets need to be sent. However, this is right: if we use something under IP we have to develop it for *each* media that shall ever be used. Why reinvent the wheel?

To distribute routing information over IP we'll have to have some protocols over IP first.

4 Internet Control Messages Protocol (ICMP)

The simplest protocol over IP is the *Internet Control Messages Protocol* (ICMP) [?]. This protocol is used for testing networks, sending simple notifications, etc. Most uses of ICMP are to indicate some sort of failure in the network.[21]

4.1 ICMP header; there are no "pure" IP packets

ICMP packets are sent inside IP packets. Each ICMP packet has a fixed length header. This header identifies the packet as an ICMP packet and contains other information such as the *type* of the packet (different packets mean different things). ICMP packet also has a body, which must be empty for most types of ICMP packets.

It should be pointed out that IP *per se* is not used for sending user data. IP is a protocol that is always used to carry something else. The first thing an operating system does with an IP packet it has to process (i.e., IP packet for this host's IP number or IP broadcast packet) is determining what protocol it carries. Different operating systems support different sets of protocols that are sent over IP. The following three are a must: TCP (see section 7, p. 38), UDP (see section 5, p. 28) and ICMP (described in this section).

Every host that talks TCP must also support ICMP. Especially important is the ICMP Fragmentation Needed message (see section 4.5, p. 26).

4.2 ICMP Echo Request and Echo Reply

The first ICMP packets people usually learn are ICMP *Echo Request* and *Echo Reply*.[22] It's really very simple: a host that receives an ICMP Echo Request must send an ICMP Echo Reply to the originator. A body is normally present in Echo Request. Echo Reply must have an identical body.

Using ICMP Echo Request/Reply one can test network connectivity.

[21] A rather old survey of hosts that talk ICMP is available as [?].
[22] These packets are used by the Unix `ping` command.

4.3 ICMP Redirect

ICMP Redirect packet is for very simple routing control. It is generated under circumstances when a host receives an IP packet from another host and the packet has to be sent to a gateway that is known to be directly reachable from the sender. A notification is sent to the sender: "I have to send packets for that host to this gateway; you could do it directly!"

The receiver is expected to comply. It should be noted that this type of ICMP packet is often ignored by default by many operating systems for security reasons.

4.4 Back to IP: time to live field; ICMP Time Exceeded

Routing can have loops. Loops are one of the most serious problems with routing. While everyone tries to prevent them they can still occur.

IP has a mechanism for discarding packets that seem to be looping. Namely, IP header has not only sender's and recipient's IP numbers, but also a field called *time to live* (TTL). It is one byte.

In the modern Internet every gateway that passes an IP packet decrements its TTL field by exactly one.[23]

A packet that has TTL zero is dropped.

A typical initial TTL of packets leaving a Unix system is something like 30.[24] (The maximum possible value is 255.[25])

4.5 Back to IP: fragmentation; ICMP Fragmentation Needed

While learning IP header fields we might also mention the *Don't Fragment* bit.

[23] The standard allows decrementing it by larger numbers but that is not used.

[24] And this is enough to reach any host on the Internet from any other host.

[25] The Unix `ping` command sets maximum possible TTL (255) for ICMP Echo Requests it sends.

But first we should consider the problem of packets that are too big. Imagine a gateway receives a packet from a network that can pass very large packets; now this packet has to be forwarded through another network that doesn't support such large packets. What should happen?

Using IP the sender has a way to configure this. There's a bit (Don't Fragment) in IP header; if this bit is set (has value one) the packet will not be delivered and an ICMP *Fragmentation Needed* notification will be sent to the originator of the packet.

If the Don't Fragment bit is not set, more interesting things will happen. Packet will be *fragmented,* split into several smaller packets. Each of these smaller packets will be a *fragmented IP packet;* from the point of view of routers fragmented IP packets are just the same stuff as "regular" packets. The target system will reassemble the packet and serve it further. Each fragment is identified as such in its IP header and bears its fragment number as well as the total number of fragments. This allows the target system to reassemble packets without ambiguity. If one or more fragments are lost in transmission the whole IP packet is discarded.

Most modern systems set the Don't Fragment bit on all outgoing packets and start with packets that are large probing the path. If ICMP Fragmentation Needed messages come a different (smaller) length of packets is tried. This behavior is called *MTU Path Discovery.*[26]

Assume that two hosts want to talk, and one of them supports MTU Path Discovery, while some router in between doesn't send Fragmentation Needed messages (or they are filtered out). Communication may become impossible. This has really happened recently to `www.internic.net` due to a misconfigured firewall.

[26] Where "MTU" stands for "Maximum Transmission Unit."

5 User Datagram Protocol (UDP)

Until now all the communications we were describing were between hosts. In reality, one host can have many system programs running all the time and many users logged on at the same time. How are they going to split incoming traffic?

As we have said in section 4.1, p. 25, IP packets always bear some other protocol inside. One of such protocols is the *User Datagram Protocol* (UDP). It is the simplest protocol that allows to communicate between programs rather than hosts.

5.1 UDP port numbers; UDP header

For this protocol we introduce a virtual resource in every participating host: UDP port numbers. UDP port numbers are 16-bit integers used to further characterize the recipient of the packet: while IP number says what host the packet should be sent to, UDP port numbers say *which program running on that host* should get the packet.

UDP packets are embedded into IP packets. The body of IP packet in this case starts with UDP header. This header specifies sender's and recipient's UDP port numbers.

A program can allocate a UDP port. After a UDP port is successfully allocated UDP packets with the allocated port as sender's UDP port can be sent out; incoming packets that specify this port as recipient's port are passed to the program. The program that allocated some UDP port is said to *listen* on that port.

Using UDP, two programs on different hosts can communicate.

5.2 UDP uses; well-known port numbers

This section describes uses of UDP and gives examples. You can skip it in the first reading. You should also just ignore things you do not know in this section.

Some UDP port numbers are *well-known*.[27] This means that a host that has a program listening on some magic UDP port number is expected to have a program there that does something specific.

UDP is usually used for such services as NFS (port 2049), DNS (port 53), RPC (port 111), talk (ports 517, 518), TFTP (port 69), syslogd (port 514), Kerberos authentication (ports 749, 750, 751), etc.

Services such as Telnet, SSH, rlogin, SMTP, HTTP, and FTP do *not* use UDP. They use Transmission Control Protocol (TCP) described in section 7, p. 38.

5.3 Back to ICMP: ICMP Port Unreachable

If a UDP packet arrives for a UDP port that has no listening program associated with it a special type of ICMP notification is sent to the originator: *ICMP Port Unreachable* message.

Upon receipt of such a message the originating system should return an error condition to the program that sent data.

In the real world, many systems do not send ICMP Port Unreachable messages. Since UDP provides no internal way to confirm receipt of data it may be difficult to figure out whether there's something listening to a given UDP port on a given remote host. (The situation is worsened by the fact that many programs will expect to get only certain data and will silently discard any data not in their format.)

It should be noted that UDP does not guarantee that packets will really be delivered. Packets can be lost should a network failure occur. Packets do not have to arrive in the same order they were sent.

[27] In Unix, you can find a list of well-known ports (for both UDP and TCP) in the file `/etc/services`.

6 Interior Gateway Protocols

Once we have protocols such as UDP and ICMP we can try to build a *routing protocol* on top of these. A routing protocol would enable hosts to exchange routing information and adjust their routing tables accordingly.

For different purposes one needs different routing protocols. In this section we describe *Interior Gateway Protocols* (IGPs). IGPs are in use in medium-sized private networks.

6.1 Routing Information Protocol (RIP)

An excellent description of the *Routing Information Protocol* (RIP) can be found in now obsoleted but still very useful RFC1058 [?]; our description of general aspects of RIP is largely based on this document.

The current standard of RIP is RIP version 2 [?, ?, ?, ?].

An internet is organized into a number of networks connected by gateways. The networks may be either point-to-point links or more complex networks such as Ethernet. Hosts[28] have to send IP packets. *Routing* is the method by which the host decides where to send the packet. It may be able to send the packet directly to the destination (if that destination is on one of the networks that are directly connected to the host). Or, if the destination is not directly reachable, the host sends the packet to a gateway that is supposed to be nearer to the destination.

The only information that is used when making the decision which gateway to send a packet to is the IP number of the destination. Also, we cannot specify the path for the packet. Once we have sent it, other gateways will start making routing decisions of their own.

The goal of a routing protocol is to supply the information that is needed for routing decisions.

RIP has a number of limitations:

- RIP is limited to networks whose longest path involves 15 hops or less.

[28] Gateways are hosts, too.

- RIP depends upon "counting to infinity" to resolve certain unusual situations (see section 6.1.3, p. 33.)
- RIP uses fixed "metrics" to compare alternative routes. (You cannot use it for load-balancing several channels, and it cannot use any real-time information such as delays.)

However, in many cases this protocol is useful. For example, it is sufficient for most campus networks and for most ISPs of moderate size.

While describing RIP we shall refine our description of routing tables. Each entry in the routing table will get additional attributes (several flags and timers); we shall also use *metric* we have mentioned earlier.

6.1.1 Distance vector algorithms

RIP is one of a class of algorithms known as "distance vector algorithms." The earliest description of this class of algorithms is believed to be in Ford and Fulkerson [2]. Because of this, they are sometimes referred to as Ford-Fulkerson algorithms. The term Bellman-Ford is also used. Distance vector algorithms were used in ARPANET as early as 1969. BSD `routed` daemon implements a version of RIP.

A routing algorithm is used to find the right gateways between networks. Distance vector algorithms are based on the exchange of only a small amount of information. Each entity (gateway or host) that participates in the routing protocol is assumed to keep information about all of the destinations within the system. Generally, information about all entities connected to one network is summarized by a single entry, which describes the route to all destinations on that network. This summarization is possible because as far as IP is concerned, routing within a network is invisible. Each entry in this routing database includes the next gateway to which datagrams destined for the entity should be sent. In addition, it includes a "metric" measuring the total distance to the entity. Distance is a somewhat generalized concept, which may cover the time delay in getting messages to the entity, the dollar cost of sending messages to it, etc. Distance vector algorithms get their name from the fact that it is possible to compute optimal routes when the only information exchanged is the list of these distances. Furthermore, information is only exchanged among entities that are adjacent, that is, entities that share a common network.

Let us forget for a minute (or whatever time you need to get to section 6.1.8, p. 36) about all the hosts that are not gateways. (This isn't such a big deal: if gateways can do good routing and hosts just send all their traffic to any gateway they know this would increase hop count by no more than one. However, if the gateway a host has chosen to send all "foreign" traffic to goes down, the host looses connectivity even if there are other available gateways. We shall get back to this issue later.)

If we represent the whole IP network as a *graph* where some gateways are connected to other gateways by direct links (going over any media), and all links are assigned a *cost* the problem of routing can be formalized nicely. Ideal routing is routing where every packet would travel via a path with minimal total cost.

A very natural approach to routing in this model is the following:

- Let each gateway know IP numbers of all its interfaces as well as the number of links it has together with their cost.
- Let each gateway maintain a table of destinations. For each listed destination also include the "next-hop" gateway and estimated metric. (Looks like a routing table, huh? We'll just call this thing routing table in the future.)
- Initially, each gateway has one entry in its routing table: an entry saying that it can send a packet to itself for zero cost.
- Periodically, *updates* are sent into all the links to which this gateway is connected. Updates just contain the routing table of the gateway. (In real life updates are sent every 30 seconds.)
- When an update is received, each entry in it processed as follows:
  - Add the cost of the link the update came from to the listed metric.
  - If the destination is listed, and our metric is more that that obtained on previous step, replace our entry. Use the gateway we obtained the update from as the next hop for this destination.
  - If destination is not listed in our routing table add it there. Use metric from update (after we have adjusted it by link cost) and the gateway the update came from as the next-hop gateway.

– If we have a better way to send packets, ignore this update entry.

One can mathematically prove that if no gateway stops sending update messages after some time routing will be ideal. (This proof can be found in [1]; if you are a mathematician, do this simple exercise on graphs and induction yourself.)

6.1.2 Dealing with changing network topology

The described algorithm is all nice and cozy, but it only works for network topology that doesn't change. It even serves *added* links just fine. However, as for each destination the metric can only decrease removed links are fatal for our routing algorithm.

How do we overcome this difficulty? We should learn to get rid of old routes that may be bad.

Let us associate a timer with each entry in the routing table. If we do not get any confirmation for a route within, say, 180 seconds the route is deleted.[29]

Another measure to take is to modify our algorithm: if our gateway for a given destination sends an update, replace our metric with the listed metric (plus cost of link) no matter what. Really, how are we supposed to send stuff to destination through a given gateway cheaper than that?

This way we can hope to get our routing right after a link or a gateway goes down.

6.1.3 Counting to infinity

With expiring routes we still have problems. Indeed, consider the network on fig. 2 I. Consider only the routing for destination T. A and B have gateway C with metric 3, C has gateway D with metric 2, D sends directly to T.

Assume that the link between C and D goes out of service. Let's look at what happens. Eventually, C notices that packets cannot be sent to D

[29] Theoretically, a value of 35 seconds would be sufficient, but we don't want to spread panic because of a single lost packet.

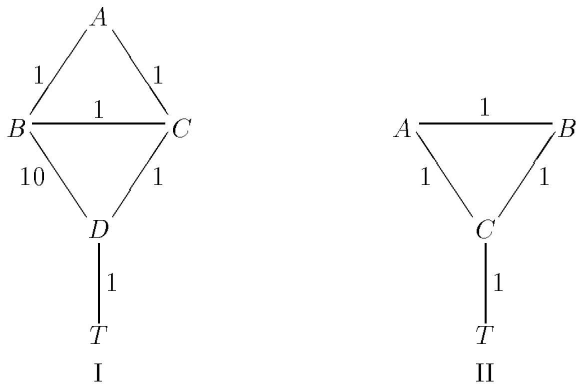

Figure 2: Counting to infinity. Letters here stand for gateways, lines are networks.

(the route expires). When it receives an update from, e.g., B it starts thinking that it can send packets for T to B and the metric is 4 (because B announces metric 3). After C sends its update, B replaces its metric with 5. And so on. Routing is broken until metric on B reaches 12 and it starts sending packets for T to D.

The worst case is when some destinations become unreachable. This is never detected. Look at figure 2 II and imagine that the link between C and T breaks. There will be a routing loop between A and B forever (with metric being incremented by 1 every 30 seconds or so). This phenomenon is called *counting to infinity.*

Counting to infinity can occur in a simpler network as well (consisting just of three gateways), however our examples are more interesting. Look at what happens in these network topologies each time we learn new means of eliminating or stopping counting to infinity.

Our goal is to prevent counting to infinity by all means. The first idea that comes to mind is to make infinity *finite.* Just pick some number, and if metric is this number consider the destination unreachable. This number must be large enough to allow the longest path in the network but as small as possible. The smaller it is, the sooner the counting to infinity will stop. In most real implementations, all links are assigned a cost of 1. The number 16 was chosen as the value of infinity because RIP is not supposed to handle networks that have longer paths.

6.1.4 Split horizon

Why did we have these loops in the previous section? Two gateways were engaged in a pattern of mutual deception. Each was thinking that packets are best routed to the other one.

This can be easily prevented. Let's modify our routing algorithm again. This time we shall be more careful about information we send out in updates. If we think that the best path to some destination goes through a given neighbor we have no reason to announce to this neighbor that we can deliver packets to the destination. Let the neighbor not count on us!

This practice is called *split horizon.* If split horizon is in use, loops consisting of two gateways cannot exist. Longer routes are still there.

6.1.5 Split horizon with poisoned reverse

A practice even more efficient than split horizon is *split horizon with poisoned reverse.* In this modification, we not only are silent about some routes. Instead, we do send routes that would be filtered out by split horizon, but specifying a metric of *infinity.*

This method is better at eliminating and preventing loops than split horizon but it consumes more bandwidth (longer tables need to be sent).

6.1.6 Triggered updates

OK, if we cannot completely get rid of counting to infinity, let's at least speed it up so that the network comes to good routing faster.

By triggered updates we mean updates that are sent without waiting for the time to broadcast an update every 30 seconds. Such updates are sent for destinations that have just increased their metrics. These updates are sent immediately and only include these specific destinations, rather than full routing table.

6.1.7 Back from graph to network

We have replaced every LAN with a set of links between gateways to come to a graph because it was easier to analyze. Now we can replace the graph back by the original network consisting of LANs.

Assume we do not use split horizon in any form. Then every gateway sends *identical* updates to all its neighbors. If several neighbors are connected to the same LAN they can get just one copy. So, RIP updates can be broadcast in this case.

Now, consider split horizon. It doesn't change anything: If we don't want a gateway to send us traffic for some destination (because we want to send it to the gateway) other gateways on the same network as the one we share with that gateway should send their traffic for that destination to the same host, *and not to us.* So, split horizon updates can be broadcast, too.

Therefore, instead of sending RIP updates to individual hosts the updates can be broadcast.

6.1.8 Non-gateway hosts

We have excluded hosts that are not gateways from our considerations. Now it is time to get back to them.

Let's make an observation: It is never right to use a host that only has one interface as an intermediate point. No-one should list hosts that are not gateways as next-hop gateways for any destination.

So, there's no point in sending RIP updates for non-gateway hosts. However it makes sense for them to *listen* to RIP announcements to choose gateways better.

6.1.9 RIP technicalities

RIP normally uses IP broadcasting to send updates. RIP updates are UDP packets, and they are usually sent from and to port 520.

RIP uses and modifies the routing table of a host. The routes described in section 3.2.3, p. 17 are *static* routes; they are marked with a special flag in the routing table and aren't used or modified by RIP. They also never expire.

Routes created by RIP have a timer associated with them. When they are created or confirmed, this timer is set to 180 seconds. When the timer expires, the route is marked as being down. Routes that are down won't be used in routing decisions, and have metric 16. They are sent in update messages, however. A route marked down walks 120 seconds dead and then gets deleted from the table.

6.1.10 Preventing bursts of RIP traffic

When several gateways share the same LAN their updates tend to synchronize. This causes unnecessary bursts of activity on the LAN. So, implementations of RIP are required to either wait a small random time after 30 seconds, or to use some clock that doesn't depend on system load.

Triggered updates tend to create fast-moving waves of traffic. To slow down this tsunami a very simple technique is used: one must wait a small random time before sending a triggered update.

6.2 Other IGPs

A much more elaborate IGP than RIP is the *Open Shortest Path First* (OSPF) protocol. OSPF is indeed a very complex protocol. The document RFC 2178 [?] that describes it is 500 K long. We won't provide any description of OSPF. It should be noted that this protocol is a *link-state* protocol; each router keeps track of network topology rather then just metrics. OSPF handles backup channels, on-request links, distributing traffic among several paths, scalability by considering a part of the network as a single destination, etc.

OSPF is suitable for reasonably large ISPs. ISPs that are really huge usually run some *Exterior Routing Protocol,* such as BGP (section 10, p. 57) internally.

Another IGP in common use is *Intermediate System to Intermediate System Routing Protocol* (IS-IS) [?, ?]. It is a very complex routing protocol as well.

7 Transmission Control Protocol (TCP)

Now that we've described some routing algorithms we can imagine how IP packets could get where they belong. We'll describe how routing works in the real Internet later. Now we shall try to get closer to the explanation of E-mail.

Until now, we were talking about packets. IP (or UDP that simply adds port numbers to IP) doesn't guarantee that all the packets we send will get to their destinations. It also doesn't guarantee the right order.

Packets can very well be lost should a network failure occur. More than that: packet loss can be normal in some situations. Imagine a router that connects two networks. One of the networks has capacity to transmit, e.g., 1000 packets of size 1 K per second, while the other one can only transmit, e.g., 10 such packets. If there's a continuous stream of packets coming in from the faster network and all have to be sent to the slower one, what should happen? The router can queue some packets. However, the stream is continuous and the amount of memory is finite. The only thing that is reasonable is to *drop* packets and just forget about them.

Now, let's assume one program needs to send some data to another program. For example, a 100 K file. The only way we have to send it now is UDP. It is impossible to send the whole file in one packet: The theoretical maximum size of an IP packet is 65535 bytes, and besides, packets larger than 1500 bytes will have to be fragmented (see section 4.5, p. 26) which is bad (if one fragment gets lost, the whole packet is discarded). So, the file has to be split in chunks that will be sent as small packets. You are strongly encouraged to think about how this can be done *reliably* before you read any further. We want to make sure the whole file is transmitted without damage.

One of the ways to do this is to do the following:

- Split the file into 1 K chunks to be sent in one packet.
- Send them all, one chunk in one UDP packet; add the following information to each packet: which chunk in the sequence this is and the total number of chunks.[30]

[30] It would also be wise to add a *checksum* to each packet. This way the receiving side can know which packets were damaged.

- After sending each chunk, wait for confirmation for several seconds; the receiving side must send a confirmation for every packet it gets. The confirmation must mention which chunk has been received. If the confirmation doesn't come, resend the packet and wait for confirmation again.

- If confirmation doesn't come after resending the packet a given number of times, give up and generate an error condition.

Sending a file is a very common operation. That's what we want all the networking for after all. Using UDP every program would have to have this or similar algorithm coded in. Every program would have its own bugs in its implementation of this algorithm. Life would become a nightmare.

The *Transmission Control Protocol* (TCP)[31] [?] is a generic solution to this problem and all similar problems. TCP introduces a new abstraction: a stream of data. A program that uses TCP doesn't have to worry about all the confirmations and retransmissions; they will be done by the operating system.

TCP is a protocol of the same level as UDP. That is, TCP packets get encapsulated in IP packets and the TCP header starts the body of IP packet.

The Transmission Control Protocol has been extended for various specific purposes [?, ?, ?, ?, ?].

7.1 TCP ports and connections

TCP introduces its own port numbers, similarly to UDP. Ports are 16-bit integers, a virtual resource similar to UDP ports. Programs can allocate ports, use them and release them. TCP ports are numbered in the same way as UDP ports, but they are independent. One program can allocate UDP port 9876, and another program can allocate TCP port 9876; they won't interfere in any way.[32]

---

[31] You could often see the acronym TCP/IP. This means the same as just "TCP." The form "TCP/IP" emphasizes the fact that TCP works *over* IP.

[32] See also section 7.7, p. 43.

TCP introduces a new concept: a *connection.* A TCP connection is characterized by four parameters: source IP number, source port, destination IP number, destination port. A TCP connection can be in many states. Vast majority of all possible connections are in the CLOSED state. A connection that is being used to send data is in ESTABLISHED state. The rest of states are intermediate states for opening and closing connections.[33]

A TCP connection is like a pipe: you can put bytes into any end, and they will arrive in order and intact to the other end. It allows two-way reliable communications.

The operating system associates a *TCP Control Block* (TCB) with each connection not in CLOSED state. A TCB is a fixed-size memory area that records the four characteristic parameters, the state of connection and some other data.

A program can do the following with TCP: it can *bind* to a TCP port (this port becomes reserved for use by the program), it can then either start *listening* on the port, or it can *connect* (establish a connection somewhere). An established connection can be *closed.*

While the connection is open (established) any side can send data to the other side. The library or system call interface to TCP always guarantees that when the function that sends data has returned with success all the data has been indeed fully and intact transmitted to the other side (and the other side has acknowledged this fact). It is also guaranteed that if the connection has been closed normally all the data that we sent while it was open has reached the destination, and all the data that our peer has sent has been delivered to us.

With a TCP connection, one side is passive when starting it (it is called the listening side, or server) while the other one is active (it is called the originator, or client).

7.2 TCP sequence numbers

Every TCP connection in ESTABLISHED state has two sequence numbers associated with it. One sequence number for the outgoing stream of

[33]In Unix, the list of all TCP connections that are not in LISTEN state and not in CLOSED state can be viewed using the `netstat -n` command.

data, and the other one for the incoming stream. Every time some data arrives, the incoming sequence number gets incremented by the number of bytes. Every time we send some data and receive confirmation that it has arrived, outgoing sequence number is incremented.

These numbers serve the purpose of the numbers we have associated with chunks of our model 100 K file.

Sequence numbers are 32-bit integers.

7.3 Opening a TCP connection

When two programs on different hosts (or on the same host, this doesn't really matter) have both reserved their ports, and the passive side has started listening, the connection is ready to be opened.

Opening a TCP connection involves exchanging three packets. It is called three-way handshake. These packets are SYN, SYN+ACK and ACK. They are called this way, because TCP header has SYN (for *synchronize*) and ACK (for *acknowledge*) bits. The first packet has SYN bit set (equal to one); the second packet has both these bits set; the third packet has only the ACK bit set.

This exchange is used to establish initial sequence numbers of the connection. After the three-way handshake is complete, connection is ESTABLISHED.

The first packet is sent by the originator and bears originator's outgoing sequence number, which is chosen randomly. The second packet is sent by the listening side and contains the same sequence number repeated (thus the receipt of SYN packet is confirmed) and its own outgoing sequence number. Third packet is sent by the originator to confirm the receipt of the second packet. At this point both sides know both sequence numbers (and they are recorded into TCBs).

7.4 Sending data through a TCP connection

Once the connection is ESTABLISHED both sides can send data. It is no longer important who originated the connection, the situation is now completely symmetrical.

When one of the sides needs to send some data, it sends a TCP packet that has it in the body. The other side replies with an ACK packet, confirming the receipt of data and the sequence number is adjusted accordingly on both sides.

If one of these packets gets lost, confirmation doesn't arrive and retransmission will occur.

7.5 Closing a TCP connection

We cannot just abandon a TCP connection. We have to make sure that all the data we have sent has reached its destination, and that the other side has transmitted all its data.

There is a bit in the TCP header, FIN, which is used when a connection is about to close. Closing a connection is somewhat similar to opening it. The side that decided to close the connection sends the other side a FIN packet. The other side must confirm it and say that it has nothing more to send by sending a FIN of its own. This FIN must be acknowledged as well.

Normal close of a TCP connection involves a handshake similar to that used to open the connection. The key point of the exchange is using the FIN bit in the TCP header. There is also another bit, RST (for *reset*) that serves a different purpose. Normally, a TCP packet with this flag set is sent in response to any packet that is apparently not meant for this connection. This includes a *desynchronized* packet (a packet that has wrong off the wall sequence numbers) or a SYN for a port that doesn't have any listening process associated with it.[34]

7.6 TCP details we do not describe

We haven't told you everything about TCP. To describe exactly what should be done when we'd have to present the finite state machine that moves between connection states upon receipt of various packets. Also, TCP includes means of flow control. We don't say anything about it. We have not told you much about the sequence numbers: What happens

[34] RST is the packet that causes you to see the message "`Connection refused.`" while using `telnet`.

when a sequence number becomes more than 2^{32}? (Nothing bad.) Why do we need to generate them randomly? (For security and reliability reasons.)

RFC 793 [?] describes all these details. The reader should now be prepared for it. What is important is that TCP allows to transmit streams of data. Should a network failure occur, it will be corrected, if possible; if such correction does not seem to be feasible (e.g., we don't get *any* responses to our packets) the sending program will be notified.

7.7 TCP well-known port numbers and TCP uses

TCP is widely used by various services. Most common services use TCP because that's what is natural for them. Services that usually use UDP can sometimes use TCP as well (e.g., RPC services including NFS can use TCP; DNS can use TCP, too).

Some ports are *well-known.* This means that a specific service is expected to listen to these ports. A list of such services with their port numbers: Telnet (port 23), FTP (port 21 for control connection and port 20 for data), Secure Shell, or SSH (port 22), SMTP (port 25), finger (port 79), HTTP (port 80), POP-2 (port 109), POP-3 (port 110), Ident (port 113), IMAP-2 (port 143), Secure HTTP, or SSL (port 443), NNTP (port 119).

X11 uses ports that are known as well. Most commonly used is port 6000, but there's no standard that codifies this.

While TCP and UDP ports are not related technically, it is presently the Internet Assigned Numbers Authority (IANA) policy to assign a single number for both UDP and TCP ports to a given service even if service uses only TCP or only UDP.

The official list of assigned port numbers can be found in [?].

8 Domain Name System (DNS)

The *Domain Name System* (DNS) [?, ?, ?, ?] is a database distributed over the whole Internet.

If we look at it as a black box, it allows to convert strings of characters (names) into IP addresses as well as carries other information. It should be pointed out that DNS is not limited to certain types of stored information [?, ?, ?, ?, ?, ?]. It is a very generic distributed database.

DNS clients are normally programs running on hosts connected to the Internet. Every host has a list of *DNS servers* that will serve queries from this host.[35] These hosts are known by IP numbers.

DNS normally operates over UDP using port 53 (but TCP can also be used). A client sends a query to the server, and the server does something to resolve it and sends back an answer. There are queries of several types.[36] We shall deal with queries of three types: A, NS and MX. A query of type A supplies a string of characters and gets in response an IP number; this is the sort of query that is used when the user specifies symbolic names in context like host name for HTTP. A query of type NS is an internal to DNS query; users should never have any reason to send these queries; its input is a string of characters and the output is string of characters as well; the output is the name of the name server responsible for domain specified in input. A query of type MX will be described in section 8.2, p. 47; it is used to find out *mail exchangers* for a given domain.

One query can get several answers. That is, if we ask for an IP number of some host (sending a type A DNS query) we can get several IP numbers in response. This is normal. It simply means that the host has several IP numbers. If we are going to connect to the host we must choose one IP number randomly. In the same fashion, a domain can have several name servers (it is even a requirement [?] to always have more than one name server just in case one of them crashes or becomes unreachable).

[35] In Unix, these name servers are given by the `nameserver` directives in the file `/etc/resolv.conf`.

[36] Most commonly supported DNS record types are A (a host address), NS (an authoritative name server), MX (mail exchanger), CNAME (the canonical name for an alias), SOA (start of a zone of authority), NULL (no record or data), RP (responsible person), PTR (a domain name pointer) and HINFO (host information: CPU type, OS). There are other types of DNS records, such as TXT, WKS, SRV, etc.

These several answers are normally packed into the same UDP packet.

Also, a DNS response can have additional information: answers to questions we didn't ask but are likely to ask on the next step. We are free to use this information, or ignore it completely.

8.1 DNS servers operation

While it is relatively clear how the DNS clients operate, it is also interesting to look into the black box of internal DNS operation.

First, the character "`.`" (a period) is special to the DNS. Every domain name *in the DNS* ends with a period. It should be pointed out that this is internal representation of things in DNS; you don't have to ever type the dot in the end of host names.[37]

A *domain* is the set of all character strings[38] that end with a dot followed by some fixed string followed by a dot. This fixed string is called the name of the domain. DNS is case-insensitive (or should I say "*should be*"? Some broken rarely found software *is* case-sensitive).

Each domain has one or more servers responsible for it.[39] The process of query resolution reduces to finding these authoritative servers and asking them questions. They have the information.

The easiest way to understand all this is by an example. Suppose we need to serve a query of type A. To be specific, let us consider what happens when we get a query of type A regarding host "`www.porcupine.org.`" (the dot will be added by the client DNS querying software). First, we need to figure out who is responsible for the *root domain,* the domain "`.`". This information is available to our software statically. We know the names and IP numbers of hosts that are responsible for this domain. The current list of these servers follows:

```
a.root-servers.net.            198.41.0.4
```

[37] Since in this section we deal with DNS, the terminating dot is usually spelled out. This makes host names look ugly in many contexts. We use double quotes to remedy this problem. Looking at the quotes you can know where a period terminates a sentence and where it is just a part of host name.

[38] There are certain limitations on which characters are allowed in the DNS. In particular, the underscore character (_) is forbidden in the DNS completely.

[39] In reality, there are always at least two servers (but the DNS could operate with just one server as well).

```
b.root-servers.net.            128.9.0.107
c.root-servers.net.            192.33.4.12
d.root-servers.net.            128.8.10.90
e.root-servers.net.            192.203.230.10
f.root-servers.net.            192.5.5.241
g.root-servers.net.            192.112.36.4
h.root-servers.net.            128.63.2.53
i.root-servers.net.            192.36.148.17
j.root-servers.net.            198.41.0.10
k.root-servers.net.            193.0.14.129
l.root-servers.net.            198.32.64.12
m.root-servers.net.            202.12.27.33
```

Then, we choose one of these name servers randomly, e.g., it might happen to be "`g.root-servers.net.`". Next we need to figure out its IP number. But we know it already. It's 192.112.36.4. Then we send to this server a query of type NS for "`org.`" asking it in effect "Who's responsible for '`org.`'?". I have just done that. In response I got the following list:

```
a.root-servers.net. b.root-servers.net. c.root-servers.net.
d.root-servers.net. e.root-servers.net. f.gtld-servers.net.
f.root-servers.net. g.root-servers.net. h.root-servers.net.
i.root-servers.net. j.gtld-servers.net. k.gtld-servers.net.
```

Additionally, for each of the servers in this list I got its IP number. On the next step, we choose any server on this list randomly. It might be, e.g., "`e.root-servers.net.`" with IP number 192.203.230.10. We send it query of type NS for "`porcupine.org.`" domain. In response I got three name servers: "`ns1.cloud9.net.`", "`ns2.cloud9.net.`" and "`spike.porcupine.org.`"; additionally, I got the IP numbers of these three servers: 168.100.1.2, 168.100.1.3 and 168.100.189.2, respectively. Now we choose one of these servers randomly, e.g., "`ns2.cloud9.net.`" and send it type A request for "`www.porcupine.org.`". In response I have just gotten IP number 168.100.189.2 as well as some additional information.

Exercise: get access to a Unix shell. Read the manual page for the `host` utility (type "`man host`"). Repeat the process of figuring out the IP number by name, step by step as we did, for the host `prep.ai.mit.edu`. Hint: `host` will print all the information with "`-v`" option. Without this option, additional information will be hidden, but the results will be in human-readable form rather than in zone description format.

After doing this exercise you should understand pretty well what happens when a DNS query gets resolved.

Terminological convention: The domains COM, ORG, NET, MIL, EDU, RU, FR, IT, UK, etc. (those that do not have a dot in their names—except the implied dot in the end) are called the *Top-Level Domains* (TLDs). Domains like EXAMPLE.COM and MCCME.RU are second-level domains, and so on.

A more precise description of how the name server works would involve a recursive algorithm (because in general we would have to resolve the names of the authoritative name servers; in practice we would most often get their names as additional information). We won't provide this exact description. Interested technically inclined reader can now easily understand the RFCs while a non-technical person would be overwhelmed by details if we try to include them all here.

8.2 MX DNS records

DNS records of type MX are a very simple thing. While both A and NS are used internally by DNS (type A is also useful externally), other types such as MX are completely for the rest of the world.

A query of type MX takes a name as input and produces a list of names each accompanied with a number as output. This data is used when sending mail out (see section 9.3, p. 51). The names are host names of hosts that will accept mail for this domain. The numbers are *priorities* of these mail exchangers: the smaller the better.

8.3 DNS authoritative data

In the situation we have described earlier *other* servers know all the answers and we just need to get the information. However, we can be the providers of information, too. Every server can be *authoritative* for certain domains; that is, hold their data.

If we are authoritative for domain the "`example.com.`" we answer any questions regarding host names "*foo*`.example.com.`" immediately based on information supplied by an operator. No further requests to any other servers are made. Naturally, an operator or network manager must supply information to each name server as to which domains this server is authoritative for. All these domains are described in a special format in configuration (zone description) files.

8.4 DNS data caching

If the DNS would work exactly the way we described, a typical name server would be sending continuous stream of questions like "Who's responsible for '`com.`'?" to the servers responsible for the root domain, etc. However, this doesn't happen because of *DNS data caching:* a name server, if it is smart enough, can remember the answer it got earlier for different purposes for some time just in case the information might be useful.

In fact, a typical DNS server has certain amount of memory dedicated to remembering answers the server doesn't need, strictly speaking. They are remembered just in case they'll be needed later.

This allows to speed up DNS lookups tremendously. Typically, a name server already has the names and IP numbers of all the authoritative name servers for, e.g., a second-level domain it has to deal with often enough.

9 How E-mail works

Now we can finally get to the explanation of E-mail. E-mail is a complex thing. It uses many transports, and many address formats. We shall only describe Internet mail with domain-style addresses. Unix-to-Unix Copy Protocol (UUCP) [?], X.400 [?, ?, ?] and BITNET gatewaying are not covered. We also do not explain Multipurpose Internet Mail Extensions (MIME) [?, ?, ?, ?, ?]. We only describe the traditional Internet mail: the transmission of text[40] messages addressed in the *user@domain* style.

9.1 E-mail format: headers and body

An E-mail message is divided into headers and body. The separation is done very simply: headers start the message[41] and are separated from the body by a blank line.[42] The body of the message is its text; naturally, there are no specifications as to what should be written there. Headers are documented in [?].

Addresses formats according to [?] can get really complex. We shall only consider the simplest case: addresses in format *user@domain*, where *user* is the *local part* of the address (like "`john`" or "`Joe.User`"; local part is interpreted by the recipient machine and can be mostly anything) and *domain* is a domain name in the DNS sense.[43]

Each header (or *header field*) consists of its name followed by a colon followed by text. If text doesn't fit on one line of reasonable length it is put on continuation lines. A continuation line starts with whitespace (space or tab character(s)) and is logically part of the previous line.

Most commonly, a header includes at least fields `From` and `To` (where the message comes from and where it was sent). A `Subject` line is very common, too. The `From` line contains a single email address, `To` line can

[40] Original specifications do not allow characters with high bit set; only US-ASCII is allowed. However, most software can handle characters with high bit set (such as Cyrillic or European characters) just fine.

[41] In traditional Berkeley Unix mailbox format, each message is started with a "`From `" line. This line is *not* a part of the headers. See section 9.2, p. 50 for more information.

[42] This blank line cannot contain even spaces.

[43] E-mail addresses should *not* be terminated with a dot: "`john@example.com`" is good, "`john@example.com.`" is not.

contain multiple addresses, separated by commas. Additional recipients can be listed in `CC` (which stands for *carbon copies*) line; there's no major difference between `To` and `CC`. More recipient can be added to the `BCC` line (which stands for *blind carbon copies*); this line is treated in the same way as `CC` with the only difference being that `BCC` will be removed in transmission (and recipients won't see the full list of people message was sent to).

9.2 The headers and the envelope

Headers are generated by the *Mail User Agent* (MUA) of the person who sends an E-mail. A MUA is a program that is used to read and compose messages, such as `mail`, Pine, Elm, Mutt or Eudora. The program that delivers the message is called a *Mail Transfer Agent* (MTA). Examples of MTAs are Sendmail, Postfix, Qmail, Smail, Microsoft Exchange, etc. When the message is first submitted to an MTA by the MUA, the headers get parsed to find out who is the sender[44] and who are the recipients of the message. Later on, headers are not used to get this information. In fact, it is not necessary to even read headers later.[45]

These sender and recipients are called *envelope* sender and *envelope* recipients. For example, if the message contained a `BCC` line, it was parsed by the MTA, the recipients were added to envelope, and the line itself was deleted. MTAs that will get to see this message further won't be able to get recipients from `BCC` anymore.

Another situation where this distinction between header and envelope is critical is *mail forwarding.* By mail forwarding we mean a situation where a user has asked the mail system to forward all his mail to another address. Naturally, changing the header would be wrong in this case: looking at the header user can learn which address this message was

[44]The sender is sometimes determined by different means (e.g., in Unix we know both our host name and the user name of the invoker). Recipients are always initially determined by looking at the headers.

[45]But many MTAs will do that and will even "fix" and "improve" headers in many ways. This procedure is called *headers rewriting.* Many people believe that it is wrong to rewrite headers. However, at many sites headers are generated incorrectly and need to be fixed. We will pretend that the world is closer to the ideal than it really is: we'll forget about the fact headers can be rewritten. In our world an MTA that gets a message from another MTA (rather than from a MUA) doesn't even read the headers.

sent to. So, envelope recipient is replaced by the forward address.

Should the message be found undeliverable for any reason (user doesn't exist, host is not in the DNS, etc.), a notification is sent to the *envelope sender*[46] of the message.

You can view envelope addresses versus header addresses in E-mail much like you would view envelope addresses versus addresses on top of letters in the good old *snail mail.*

9.3 Identifying host mail should be transferred to

Now let's look into how an MTA works once the message has been parsed and envelope has been built. The message has to be delivered to several recipients.[47] It is obviously sufficient to be able to deliver the message to one recipient. In the next section we'll see that we can transfer message for more than one recipient, but it is up to the sending MTA to decide whether it will attempt any optimizations with grouping recipients (for example, if we have two recipients with the same domain part it might be natural to transmit the body of the message just once).[48]

We'll be transferring the message through a TCP connection. So, we need to know the IP number of the destination host. Some hosts can be configured to transfer all mail to one given *smart host* that will do the delivery. These are of no great interest. We want to know what the smart host will do.

The smart host will perform certain lookups in the DNS to determine where to send the message. The algorithm is as follows:

- Do an MX type query for the domain part. It may produce a list of *mail exchangers*—hosts that are expected to accept this message. Each mail exchanger is accompanied with a number, which specifies *priority.* One can view priority as the cost of sending the message through this mail exchanger.

[46] In Berkeley Unix mailbox format, the "`From `" line before the header lists the envelope sender.

[47] A rule of thumb is, by the way, that an average E-mail message is 2.5 K long and has 3.5 recipients. Of course, this can be different for different sites.

[48] Some MTAs, notably Qmail, never optimize this aspect and always deliver one copy of a message to one recipient. Other MTAs group recipients by the domain part of addresses. Other such as Sendmail use more sophisticated approaches.

If this first DNS query fails (there is no MX type record for the domain name) we treat the domain part of the address as the only mail exchanger host (with priority zero, which doesn't matter since there's only one mail exchanger).

- Sort the mail exchangers by priorities.
- For each mail exchanger in the list, starting with smaller values of priority, we do the following: send DNS query of type A for this host; if it succeeds, try to send the message to the IP number we obtained as described in the next section.

9.4 Simple Mail Transfer Protocol

Once we have the IP number of the host we want to transfer mail to, SMTP [?] is used to transfer mail. SMTP uses TCP connection to port 25 of the remote host. The sender issues certain commands, and the receiver sends numeric responses to these commands (numeric responses are accompanied with explanatory text meant mostly for humans). Each command must get such a response. Commands are terminated with an end-of-line. End-of-line in SMTP is always represented as Carriage Return character followed by Linefeed character.

There are the following SMTP commands:

HELO Specify sender's host name. Format: "`HELO` *hostname*". Should be the first command sender issues.

MAIL Specify sender. Format: "`MAIL From:<`*user@domain*`>`". Normally, follows HELO directly, but can also be used after DATA to send another message.

RCPT Specify recipient. Format: "`RCPT To:<`*user@domain*`>`". Must be after MAIL. May be issued multiple times.

DATA Start transmission of the message. Format: "`DATA`". Is followed by the message, terminated with a line that contains a single dot.

QUIT Terminate connection. Format: "`QUIT`". The receiver sends a response and closes the TCP connection.

RSET Reset state. Format: "`RSET`". Tells the receiver to forget everything that was told earlier (cancels effects of MAIL and RCPT commands).

VRFY Verify whether user exists. Format: "`VRFY` *user@domain*" or "`VRFY` *user*". Isn't used in normal mail transmission. The receiver is expected to say whether mail to the specified address would be delivered.

EXPN Same as VRFY, but the receiver is expected to *expand* the address. For example, if mail has been forwarded, it might say, where. This command is optional.

HELP Get some help from the receiver. Obviously, this is meant for humans manually talking to port 25. This command is optional.

In the responses, the most interesting part is the first character. It says whether the command was successful. Its meanings are:

2 OK, command successful.

3 Intermediate response, "go on." Is usually used after DATA (meaning "go on, send the message").

4 Temporary failure. This can indicate a variety of conditions: disk full, recipient is over mail quota, etc. The sender is expected to try to deliver the same message later.

5 Permanent failure. Tells the sender that this message won't ever be accepted. A modification to the recipient's address or the message body is required. A common reason for this response is the fact that recipient user is unknown.

The next two digits of the response are used to further characterize the response so that, e.g., a more informative bounce message can be generated.

9.5 E-mail delivery example

Let us consider an example of an E-mail delivery. Assume we are on the host `mccme.ru` and we want to send an E-mail message to the E-mail address *shalunov@landau.ac.ru*[49] in the same way an MTA would do it.

First, we look at the domain part of the E-mail address; in this address it is "`landau.ac.ru`". We send a DNS query of type MX for "`landau.ac.ru`".[50] In response we get three mail exchangers as follows: `netserv2.free.net` with priority 60, `cpd.landau.ac.ru` with priority 20 and `netserv1.free.net` with priority 50. The smallest value of priority is for `cpd.landau.ac.ru`, so we try it first. A DNS query of type A shows that the IP number is 193.233.9.7.

We open a TCP connection to 193.233.9.7 port 25.[51] Then we have the following conversation:

Receiver:	220 cpd.landau.ac.ru ESMTP Sendmail
Sender:	HELO mccme.ru
Receiver:	250 cpd.landau.ac.ru Hello mccme.ru, pleased to meet you
Sender:	MAIL From:<shalunov@mccme.ru>
Receiver:	250 <shalunov@mccme.ru>... Sender ok
Sender:	RCPT To:<shalunov@landau.ac.ru>
Receiver:	250 <shalunov@landau.ac.ru>... Recipient ok
Sender:	DATA
Receiver:	354 Enter mail, end with "." on a line by itself
Sender:	From: shalunov@mccme.ru
Sender:	To: shalunov@landau.ac.ru
Sender:	Subject: test
Sender:	
Sender:	This is a test message.
Sender:	.
Receiver:	250 PAA07529 Message accepted for delivery
Sender:	QUIT
Receiver:	221 cpd.landau.ac.ru closing connection

In a few seconds, I had this test message in my mailbox. You should use your friendly nearby SMTP server to send a test message to yourself to become more fluent in SMTP. (Choose a server that doesn't run Qmail or you'll have problems entering correct network end-of-line sequences; Sendmail, Smail and Postfix will accept a plain Carriage Return for end-of-line.)

---

[49] First I wanted to use `math.wisc.edu`, but found out it is misconfigured: the mail exchanger is a CNAME, which is forbidden by the standards.

[50] I used the command `host -t mx landau.ac.ru`.

[51] One can do this in Unix by typing "`telnet 193.233.9.7 25`". I use `nc` instead of `telnet`.

9.6 Continuation lines in SMTP responses

Sometimes, an SMTP receiver might want to send a long textual part of response (e.g., to explain why some particular message was rejected). In this case one line of text might not be enough. Sending responses longer than 80 characters is considered a bad style.

For these purposes SMTP responses *continuation lines* are used. The three-digit number is normally followed by whitespace. If it is followed by a hyphen ("-") the line is assumed to be continued. There are more lines coming. They must start with the same three-digit number. The last line of a response must have whitespace rather than hyphen after the digital part.

So, we might see something like this as an SMTP greeting banner:

Receiver: `220-mail.example.com ESMTP Super-Duper server.`
Receiver: `220-Example, Inc. forbids sending UCE through its servers.`
Receiver: `220 We delete all mail with porno contents.`

Of course, it is not a good idea to make the banner too long (because it gets sent to every site that connects to us, every time they send a message).

9.7 Queuing mail

When a permanent failure occurs during the delivery of a message, a *bounce* is generated and sent back to the envelope sender. A permanent failure is a response from the SMTP-receiver starting with a "5" or a permanent DNS failure. (DNS can distinguish permanent and temporary failures, too. If a name server is simply unreachable or doesn't answer, a temporary failure condition is returned. However, if a name server does reply and it says that there's no such host or domain, it is a permanent failure. MTAs use this distinction between permanent and temporary DNS failures.)

However, if a temporary failure occurs (such as: temporary DNS failure, inability to connect to mail exchangers, a response starting with a "4" from the SMTP-receiver) the message is kept for future deliveries and is tried later again and again until it is delivered or a permanent failure occurs.

This practice is called *queuing mail.*

If the message is simply sitting in the queue for too long (many implementations use the value of 5 days here) it is treated as a permanent failure, too.

Some implementations of SMTP will send a warning message to the envelope sender after the message is sitting a specified amount of time in the queue (most commonly, 4 hours).

9.8 Extended SMTP

In our ideal world, a sending MTA doesn't pay any attention to anything but the first digit of server replies. In the real world, there are exceptions to this rule.

SMTP is a restrictive protocol. For example, it does not allow to specify message size in advance. Thus, sites that wish to reject messages larger than a given limit have to waste bandwidth receiving them anyway, and only then give a permanent failure response after the sender sends the dot that terminates the message. Also, strict SMTP doesn't allow any data with high bit set anywhere in the message or in the commands.

To work around these restrictions, the *Extended Simple Mail Transfer Protocol* (ESMTP) [?, ?, ?, ?, ?] was designed. It has more features and is backwards compatible with SMTP.

We won't describe this protocol because it is not essential for E-mail delivery. SMTP works fine, ESMTP just adds features. However, it's interesting to know how an ESMTP-capable client knows it talks to a ESMTP-capable server. First, in the initial "220" greeting banner, the second word is taken into account. If this is the word "ESMTP" the server is tried as an ESMTP server by sending it a modified HELO command. Namely, ESMTP introduces new command: EHLO. An ESMTP conversation is started with an EHLO; if it is sent and succeeds both sender and receiver know their peer is ESMTP-capable. If it fails the sender will just back out to SMTP and send good old HELO.

It should be safe to never bother with ESMTP, and just use SMTP if the advanced features of ESMTP are not required.

10 Border Gateway Protocol (BGP)

The Internet is big. A protocol like RIP or even any other IGP simply doesn't scale to the whole Internet. Besides, these protocols assume that the whole network is under the same administrative control, which is not the case with the Internet.

For the purposes of routing, the Internet is divided into *Autonomous Systems* (ASs). The classic definition of an Autonomous System is a set of routers under a single technical administration, using an interior gateway protocol to route packets within the AS, and using an exterior gateway protocol to route packets to other ASs. Since this classic definition was developed, it has become common for a single AS to use several interior gateway protocols within an AS. Our use of the term Autonomous System stresses the fact that, even when multiple IGPs are used, the administration of an AS appears to other ASs to have a single coherent interior routing plan and presents a consistent picture of what networks are reachable through it.

ASs are identified by unique 16-bit numbers: *Autonomous System Numbers* (ASNs). These numbers are used for routing purposes but there's no way a user can learn by the means of IP to which AS a given IP number "belongs" (is routed to). More than that, it might be impossible even if all routers would want to provide such information.

Most ISPs, universities, large companies, etc., have their own AS number. As mentioned earlier, these numbers cannot be learned by means of the Internet Protocol.[52] The relation of ASNs to IP numbers is complex; it changes as time passes and the view of this mapping can be different in different parts of the Internet. We'll see how that can happen later.

The whole Internet can be viewed as a graph formed by ASs. Some ASs are connected one to another; they are called adjacent. Routers that are connected to router(s) in other AS(s) are called *border gateways. Border Gateway Protocol* (BGP) [?, ?] is used by border gateways in adjacent ASs to exchange routing information. It can also be used by routers belonging to the same AS to exchange routing information concerning the rest of the Internet or even as an IGP.

A BGP-speaking router knows its own ASN as well as information about

[52] But a person who knows how to use the `whois` command can learn ASN of a given organization.

all its links (to which AS they go). The fundamental rule of BGP is that a BGP-speaker announces only those routes it uses itself.

Every AS must present a consistent view of routing to the outside. From the point of view of BGP, AS is just a single point.

Terminological convention: An AS with only one connection is called a *stub* AS. An AS that has more than one connection and that passes packets that are not destined for this AS and not originated at this AS is called a *transit* AS. An AS that has more than one connection but that doesn't pass any packets not of its own is called a *multi-homed* AS. In particular, an AS with two connections that doesn't pass packets for other ASs is called *dual-homed.*

10.1 Classless routing

It is very natural to give an organization a class A, B or C network. However, as the Internet grew problems were identified with this approach:

- Exhaustion of the class B network address space. The cause of this problem is the lack of a network class of a size which is appropriate for a mid-sized organization; class C, with a maximum of 254 host addresses, is too small, while class B, which allows up to 65534 addresses, is too large to be densely populated.
- Growth of routing tables in Internet routers beyond the ability of software (and people) to effectively manage.
- Eventual exhaustion of the 32-bit IP address space.

While there's nothing much one can do with the third problem (except ensuring that the assigned blocks of IP numbers are, in fact, being used) the first and the second problems are consequences of a bad design of routing. To get classes of networks we fix a prefix of IP numbers while the rest varies arbitrarily. The prefix has length 8, 16 or 24 bits. But why do we have to choose these lengths only? Why not consider a set of IP numbers that has the first 17 bits fixed as a network?

This way we come to *classless routing,* or Classless Inter-Domain Routing (CIDR) [?, ?]. Network is defined as the set of IP numbers that have some fixed prefix (this prefix can have any number of bits from 0 to 32; 0 means the whole Internet, 32 means some specific host). Every network has an IP number and a netmask (both are 32-bit integers). Netmask always has a special property: It starts with a block of bits equal to 1 followed by a block of bits equal to 0 (so, 255.255.192.0 and 255.0.0.0 are valid netmasks while 255.15.0.0 is not). An IP number is said to belong to the network if and only if the logical bitwise AND of the IP number with the netmask is the network's IP number.

There are two notations for CIDR networks: 12.13.0.0/255.255.224.0 is equivalent to 12.13.0.0/19; in the latter form the number after the slash specifies how many bits of netmask are equal to 1.

This approach has two advantages over the scheme with networks of classes A, B and C:

- It allows one to use a network more suitable for a given organization than a class B or C;
- It allows one to *aggregate* routing information better: If we know that both 10.20.128.0/18 and 10.20.192.0/18 are routed in the same direction, we can just remember that 10.20.128.0/17 is to be sent there.

BGP version 4 (which is used by the modern Internet) is a classless routing protocol.

10.2 Transport protocol connections

Two routers that want to talk BGP establish a *transport protocol connection.* It is a TCP connection from port 179 to port 179. Notice that since the routers are directly connected there's no need for any routing protocol to send data through the transport protocol connection.

Links between routers belonging to the same AS are called *external* while links between routers in different ASs are called *internal.*

The usage of TCP allows us to exchange information reliably without having to care about confirmations and retransmissions.

10.3 Routing Information Base

The *Routing Information Base* (RIB) within a BGP speaker consists of three distinct parts:

Adj-RIBs-In The Adj-RIBs-In part stores routing information that has been learned from BGP neighbors.

Loc-RIB The Loc-RIB contains the local routing information that the BGP speaker has selected by applying its local policies to the routing information contained in its Adj-RIBs-In.

Adj-RIBs-Out The Adj-RIBs-Out stores the information that the local BGP speaker has selected for advertisement to its peers.

The Adj-RIBs-In contain unprocessed routing information that has been advertised to the local BGP speaker by its peers; the Loc-RIB contains the routes that have been selected by the local BGP speaker's decision process; and the Adj-RIBs-Out organize the routes for advertisement to specific peers.

10.4 Messages exchanged by BGP peers

Routers that want to speak BGP establish a transport protocol connection when they are turned on. (If one of them can't establish the connection it will retry periodically.)

BGP protocol messages are exchanged through the transport protocol connection. The message that bears routing information is UPDATE message. Update message specifies a CIDR IP network and a *path* consisting of ASNs. This path specifies how this router will send packets to the specified network. This way routers are aware of network topology rather than just distances to various locations.

The router that receives an UPDATE message prepends the ASN of its peer to the path if the link is external and then adds this information to its Adj-RIBs-In. The UPDATE message will also be propagated to all the other peers (each BGP peer in every router has a queue of not-yet-transmitted BGP UPDATEs for it in Adj-RIBs-Out).

If a BGP peer is detected to be unreachable (this can happen in two ways: transport protocol connection may break and the peer might not respond to a special KEEPALIVE message) all the entries in the Loc-RIB learned from this router are deleted, and all the other peers are notified that these routes are no longer valid.

An UPDATE message can withdraw routes as well as advertise them. Once a route becomes invalid (e.g., the transport protocol connection to the peer that has advertised it breaks) *we* stop using it. But, we mustn't advertise any routes we don't use. So we send UPDATE messages to all the other peers saying that the route became invalid.

10.5 Route aggregation and filtering

Before the route that has arrived in an UPDATE message is used it is subject to two procedures: filtering and aggregation.

By filtering we mean the process of figuring out whether we will accept this route from this peer. For example, if we know that peer is passing packets to some given networks we might not want to accept routes for any other networks from this peer. Loop filtering is a must: loops in the AS path must be excluded.

If several routes can be joined to form a route for a larger IP network, routers must do this and propagate further this new route.

10.6 Routing decisions

When an IP packet arrives a BGP-capable router looks at the routing database and chooses the next-hop AS for this packet based on information found there. Normally, the routing database contains 30–40 thousand entries, and many of them will match. The most specific matches are used.

The router chooses one of them based on the value of a function on the path. This function is configurable (and in the simplest case it may be just the length of the path).

Once the next-hop AS is known the packet is passed to this AS. If the router shares a network with a router in that AS the packet is sent directly, otherwise a router must know from sources other than BGP how to send packets to that AS.

References

[1] D. P. Bertsekas and R. G. Gallaher. *Data Networks.* Prentice-Hall, Englewood Cliffs, N.J., 1987.

[2] L . R. Ford Jr. and D. R. Fulkerson. *Flows in Networks.* Princeton University Press, Princeton, N.J., 1962.

Index