Network Systems (202001026/202001151)
Reader

Ralph Holz, Pieter-Tjerk de Boer
University of Twente

January 2024

Preface

This reader contains material for the module Network Systems of two bachelor programs at the University of Twente: it is both module 3 in the Computer Science program (with course code 202001026) and module 7b in the Electrical Engineering program (with course code 202001151). The module is taught using version 6.2 of the on-line edition of the textbook "Computer Networks, a systems approach" by Larry Peterson and Bruce Davie. Next to this textbook, sections from other books and a few journal articles are used in the module, which can be found in this reader.

For the 2022 edition, several sections have been added to the reader, for two reasons. One is the long overdue addition of more material on security; this topic was expanded after dropping the localization and timing topics a few years ago, but corresponding written material was missing. The other reason is that there are a few topics where we found the Peterson&Davie book not clear or complete enough, so replacement material is included in the reader. Note that these changes make the page and section numbers in this reader incompatible with previous editions.

Pieter-Tjerk de Boer
Ralph Holz

First edition in 2014

Reprint 2016: unchanged

Reprint 2017: unchanged except for course codes

Reprint 2018: minor correction in the table of contents, and a small update on page 31

Edition 2019: Due to changes in the module, the former chapters 10 and 11 (on synchronization and localization) have been removed. Furthermore, the reader will now be distributed electronically rather than in paper form. The material is still covered by copyright though; other distribution than via the Canvas site of this module is not allowed.

Edition 2020: Minor changes in Chapter 5, mostly to match changes in the P&D book.

Reprint 2021: unchanged except for course codes

Edition 2022: Material added on a number of topics, particularly security

Edition 2023: Added short section about e-mail servers, and replaced the AODV material by a part on BGP

Edition 2024: Very minor cosmetic changes

Contents

Sources:

1. (written for this reader by R. Holz)
   .
2. Kurose & Ross: Computer Networking – A Top-Down Approach
   7th edition, Pearson, 2017, ISBN 9781292153605/9781292153599, pp. 704 – 710.
3. Kurose & Ross: Computer Networking – A Top-Down Approach
   7th edition, Pearson, 2017, ISBN 9781292153605/9781292153599, pp. 168 – 174.
4. Gleick: The Information
   Pantheon, 2011, ISBN 978-0-375-42372-7, selected from pp. 221 – 224 and 228 – 231.
5. (written for this reader by P.T. de Boer)
   .
6. Comer: Computer Networks And Internets
   5th edition, Pearson, 2009, ISBN 0-1-504583-5, pp. 113 – 121.
7. Schiller: Mobile Communications
   2nd edition, Adison-Wesley, 2003, ISBN 0-321-12381-6, pp. 35 – 41.
8. Kurose & Ross: Computer Networking – A Top-Down Approach
   6th edition, Pearson, 2013, ISBN 978-0-273-76896-8, pp. 471 – 487.
9. Kurose & Ross: Computer Networking – A Top-Down Approach
   7th edition, Pearson, 2017, ISBN 9781292153605/9781292153599, pp. 370 – 373.
10. Kurose & Ross: Computer Networking – A Top-Down Approach
    5th edition, Pearson, 2010, ISBN 978-0-13-136548-7, pp. 385 – 388.
11. RFC3194
    `http://www.rfc-editor.org/rfc/rfc3194.txt`.
12. Bertsekas & Gallager: Data Networks
    2nd edition, Prentice-Hall, ISBN 0-13-200916-1, 1992, pp. 396 – 398.
13. Bonaventura et al.: Computer Networking : Principles, Protocols and Practice, 2nd edition
    `https://www.computer-networking.info/2nd/html/index.html`.
14. Kurose & Ross: Computer Networking – A Top-Down Approach
    7th edition, Pearson, 2017, ISBN 9781292153605/9781292153599, pp. 83 – 87.
15. Kurose & Ross: Computer Networking – A Top-Down Approach
    7th edition, Pearson, 2017, ISBN 9781292153605/9781292153599, pp. 659 – 673.
16. Kurose & Ross: Computer Networking – A Top-Down Approach
    7th edition, Pearson, 2017, ISBN 9781292153605/9781292153599, pp. 679 – 690.

1 A clarification about mail servers

Email terminology is actually quite complex. Worse, definitions may overlap, and implementations may cover several concepts at once, or one in its entirety and another only partially. This is the result of organic growth of the Internet. Let's have a look at the details.

The client side

The central concept on the client side is that of a Mail User Agent. This is software to read and write email, but also to 'get' the ready email from the user and hand it off for delivery by further agents: the MTAs and MSAs, which run on the server, see below. What we call 'mail clients' implements at least the concept of the MUA, but in general comes with more functionality. For example, it also lets users access their incoming email, which is held on server side for them.

The server side

The server side is where things get really complex, and the original slides were an attempt to condense this. Unfortunately, it led to the question posed at the beginning of this page, and hence we need to have a look in more detail. The following is the most common way how mail is sent and received conceptually.

First of all, how do outgoing emails move from MUA to their destination? The first step is the so-called submission from MUA to a Mail Submission Agent (MSA). This agent then hands over to a Mail Transfer Agent (MTA), which is responsible to move the email to the destination server, where the recipient can access it. The destination server needs to distinguish between many different users it serves, and hence a Mail Delivery Agent (MDA) stores the email in a location that is separate for each user (basically, a mailbox). These concepts can be implemented in separate software, but it is quite common that they are all merged in a so-called 'mail server' software. That is why a mail server contains an MTA, but it usually has more functionality.

Still with me? We're almost done, I promise. There is one last piece missing: how can recipients access or download their incoming email? That is again the job of the MUA, which has to communicate with the mail server software to this end.

Side note: practice and some more confusing overlaps

To make things more confusing, many implementations implement only part of the concept and leave the rest to some other piece of software. For example, the mail client *mutt* considers itself an MUA (its homepage says so). However, it does not communicate with the MSA itself, it relies on other software for that — which it also calls MSA, but the 'S' there is for 'sending', and hence the term is very distinct from the MSA as the RFCs (and we) define it, namely the software that accepts email that the MUA wants to have delivered. Confusing? Very. On the other hand, the MUA *Outlook* incorporates everything we have described above as functions of an MUA.

On top of all that — I described above the most common way how everything is done on a conceptual level, and I told you that MUAs normally reside on some personal hardware (end user), whereas MSAs and MTAs normally run on a professionally run server. However, this is not set in stone. There is no conceptual or practical obstacle to installing an MUA and an MTA on your laptop, and sending mail straight from your laptop to a recipient's MTA. It's just very rarely done because there is no good reason for it. For example, when you sign up with an ISP (or get a user account at UT), they will often tell you 'which mail server you can use'. This means they are telling you that you can use the MTA they have already set up for all their users. Why would you then install your own? Interestingly, one factor in identifying spam today is which IP sent a mail — and a mail sent by an MTA in some home network is relatively suspicious. Why? Because that is what malware likes to do — set up an MTA on your Windows (or whatever) at home that you have no knowledge of, and then abuse your computer to send ... spam.

We suggest two things: 1) stick with the standards, the RFCs, when you use terminology. In practice, you will find that you often have to clarify what you mean by an acronym. 2) Clearly distinguish between the concept

and the implementations. 'Mail client' is mostly understood to be something like *Outlook*. Mail server is most commonly understood to be a combination of MTA, MSA, and MDA, plus functions to allow access to email for the recipients.

Protocols and accessing email

So, which protocols are used between the different agents? It turns out this is more straight-forward.

The Simple Mail Transfer Protocol (SMTP) is used for communication between MUA and MSA, and between MTA and MTA. The communication between MTA and MDA can be implemented in custom ways (there is no standard protocol, nor is there strictly a need for one if everything is implemented by the same software).

The protocols POP3 and IMAP4 can both be used by recipients (using MUAs again) to access their incoming email (and, actually, also often copies of sent emails and archived emails). POP3 is fairly simple: emails are downloaded to the MUA. Copies can remain on the server, but that's pretty much it. IMAP4 is much more complex: emails do not need to be downloaded, and the MUA interacts with the mail server software to access mail on demand. Often, the so-called mail headers are communicated between mail server and mail client, and the user can then click on a mail to have the actual content fetched. The mail is not necessarily stored locally, but it is often cached.

Wait a second! What is Gmail? And Microsoft Live? And Outlook 365? And ProtonMail? I think I need to sit down.

Glad you asked. Today, most users think of 'webmail' when they talk about email. That is another organic growth: mail clients have always been a bit irksome in terms of configuration. Webmail solved this: it's basically an MUA running in your browser, implemented by the server sending the browser HTML and JavaScript to execute. The MUA interacts with software on the server, and in the background the software implements MSA, MTA, MDA, and supports access to incoming email. How that is done exactly does not matter — the webmail providers keep the details to themselves. They can even swap software in the background without the user noticing it. Webmail became popular because it is so simple to use.

Oh my, how much of this do I need to know? I only care about the exam, and you told me more than I ever wanted to know!

Well, you asked... But for the exam, we want you to understand and be able to explain:

- What an MUA is
- What an MTA is
- What mail clients and servers do
- What webmail is

9.1 Multimedia Networking Applications

We define a multimedia network application as any network application that employs audio or video. In this section, we provide a taxonomy of multimedia applications. We'll see that each class of applications in the taxonomy has its own unique set of service requirements and design issues. But before diving into an in-depth discussion of Internet multimedia applications, it is useful to consider the intrinsic characteristics of the audio and video media themselves.

9.1.1 Properties of Video

Perhaps the most salient characteristic of video is its **high bit rate**. Video distributed over the Internet typically ranges from 100 kbps for low-quality video conferencing to over 3 Mbps for streaming high-definition movies. To get a sense of how video bandwidth demands compare with those of other Internet applications, let's briefly consider three different users, each using a different Internet application. Our first user, Frank, is going quickly through photos posted on his friends' Facebook pages. Let's assume that Frank is looking at a new photo every 10 seconds, and that photos are on average 200 Kbytes in size. (As usual, throughout this discussion we make the simplifying assumption that 1 Kbyte = 8,000 bits.) Our second user, Martha, is streaming music from the Internet ("the cloud") to her smartphone. Let's assume Martha is using a service such as Spotify to listen to many MP3 songs, one after the other, each encoded at a rate of 128 kbps. Our third user, Victor, is watching a video that has been encoded at 2 Mbps. Finally, let's suppose that the session length for all three users is 4,000 seconds (approximately 67 minutes). Table 9.1 compares the bit rates and the total bytes transferred for these three users. We see that video streaming consumes by far the most bandwidth, having a bit rate of more than ten times greater than that of the Facebook and music-streaming applications. Therefore, when design-

	Bit rate	Bytes transferred in 67 min
Facebook Frank	160 kbps	80 Mbytes
Martha Music	128 kbps	64 Mbytes
Victor Video	2 Mbps	1 Gbyte

Table 9.1 ♦ Comparison of bit-rate requirements of three Internet applications

ing networked video applications, the first thing we must keep in mind is the high bit-rate requirements of video. Given the popularity of video and its high bit rate, it is perhaps not surprising that Cisco predicts [Cisco 2015] that streaming and stored video will be approximately 80 percent of global consumer Internet traffic by 2019.

Another important characteristic of video is that it can be compressed, thereby trading off video quality with bit rate. A video is a sequence of images, typically being displayed at a constant rate, for example, at 24 or 30 images per second. An uncompressed, digitally encoded image consists of an array of pixels, with each pixel encoded into a number of bits to represent luminance and color. There are two types of redundancy in video, both of which can be exploited by **video compression**. *Spatial redundancy* is the redundancy within a given image. Intuitively, an image that consists of mostly white space has a high degree of redundancy and can be efficiently compressed without significantly sacrificing image quality. *Temporal redundancy* reflects repetition from image to subsequent image. If, for example, an image and the subsequent image are exactly the same, there is no reason to re-encode the subsequent image; it is instead more efficient simply to indicate during encoding that the subsequent image is exactly the same. Today's off-the-shelf compression algorithms can compress a video to essentially any bit rate desired. Of course, the higher the bit rate, the better the image quality and the better the overall user viewing experience.

We can also use compression to create **multiple versions** of the same video, each at a different quality level. For example, we can use compression to create, say, three versions of the same video, at rates of 300 kbps, 1 Mbps, and 3 Mbps. Users can then decide which version they want to watch as a function of their current available bandwidth. Users with high-speed Internet connections might choose the 3 Mbps version; users watching the video over 3G with a smartphone might choose the 300 kbps version. Similarly, the video in a video conference application can be compressed "on-the-fly" to provide the best video quality given the available end-to-end bandwidth between conversing users.

9.1.2 Properties of Audio

Digital audio (including digitized speech and music) has significantly lower bandwidth requirements than video. Digital audio, however, has its own unique properties that must be considered when designing multimedia network applications.

To understand these properties, let's first consider how analog audio (which humans and musical instruments generate) is converted to a digital signal:

- The analog audio signal is sampled at some fixed rate, for example, at 8,000 samples per second. The value of each sample will be some real number.
- Each of the samples is then rounded to one of a finite number of values. This operation is referred to as **quantization**. The number of such finite values—called quantization values—is typically a power of two, for example, 256 quantization values.
- Each of the quantization values is represented by a fixed number of bits. For example, if there are 256 quantization values, then each value—and hence each audio sample—is represented by one byte. The bit representations of all the samples are then concatenated together to form the digital representation of the signal. As an example, if an analog audio signal is sampled at 8,000 samples per second and each sample is quantized and represented by 8 bits, then the resulting digital signal will have a rate of 64,000 bits per second. For playback through audio speakers, the digital signal can then be converted back—that is, decoded—to an analog signal. However, the decoded analog signal is only an approximation of the original signal, and the sound quality may be noticeably degraded (for example, high-frequency sounds may be missing in the decoded signal). By increasing the sampling rate and the number of quantization values, the decoded signal can better approximate the original analog signal. Thus (as with video), there is a trade-off between the quality of the decoded signal and the bit-rate and storage requirements of the digital signal.

The basic encoding technique that we just described is called **pulse code modulation (PCM)**. Speech encoding often uses PCM, with a sampling rate of 8,000 samples per second and 8 bits per sample, resulting in a rate of 64 kbps. The audio compact disk (CD) also uses PCM, with a sampling rate of 44,100 samples per second with 16 bits per sample; this gives a rate of 705.6 kbps for mono and 1.411 Mbps for stereo.

PCM-encoded speech and music, however, are rarely used in the Internet. Instead, as with video, compression techniques are used to reduce the bit rates of the stream. Human speech can be compressed to less than 10 kbps and still be intelligible. A popular compression technique for near CD-quality stereo music is **MPEG 1 layer 3**, more commonly known as **MP3**. MP3 encoders can compress to many different rates; 128 kbps is the most common encoding rate and produces very little sound degradation. A related standard is **Advanced Audio Coding (AAC)**, which has been popularized by Apple. As with video, multiple versions of a prerecorded audio stream can be created, each at a different bit rate.

Although audio bit rates are generally much less than those of video, users are generally much more sensitive to audio glitches than video glitches. Consider, for example, a video conference taking place over the Internet. If, from time to time, the video signal is lost for a few seconds, the video conference can likely proceed

without too much user frustration. If, however, the audio signal is frequently lost, the users may have to terminate the session.

9.1.3 Types of Multimedia Network Applications

The Internet supports a large variety of useful and entertaining multimedia applications. In this subsection, we classify multimedia applications into three broad categories: *(i) streaming stored audio/video*, *(ii) conversational voice/video-over-IP*, and *(iii) streaming live audio/video*. As we will soon see, each of these application categories has its own set of service requirements and design issues.

Streaming Stored Audio and Video

To keep the discussion concrete, we focus here on streaming stored video, which typically combines video and audio components. Streaming stored audio (such as Spotify's streaming music service) is very similar to streaming stored video, although the bit rates are typically much lower.

In this class of applications, the underlying medium is prerecorded video, such as a movie, a television show, a prerecorded sporting event, or a prerecorded user-generated video (such as those commonly seen on YouTube). These prerecorded videos are placed on servers, and users send requests to the servers to view the videos *on demand*. Many Internet companies today provide streaming video, including YouTube (Google), Netflix, Amazon, and Hulu. Streaming stored video has three key distinguishing features.

- *Streaming.* In a streaming stored video application, the client typically begins video playout within a few seconds after it begins receiving the video from the server. This means that the client will be playing out from one location in the video while at the same time receiving later parts of the video from the server. This technique, known as **streaming**, avoids having to download the entire video file (and incurring a potentially long delay) before playout begins.
- *Interactivity.* Because the media is prerecorded, the user may pause, reposition forward, reposition backward, fast-forward, and so on through the video content. The time from when the user makes such a request until the action manifests itself at the client should be less than a few seconds for acceptable responsiveness.
- *Continuous playout.* Once playout of the video begins, it should proceed according to the original timing of the recording. Therefore, data must be received from the server in time for its playout at the client; otherwise, users experience video frame freezing (when the client waits for the delayed frames) or frame skipping (when the client skips over delayed frames).

By far, the most important performance measure for streaming video is average throughput. In order to provide continuous playout, the network must provide an

average throughput to the streaming application that is at least as large the bit rate of the video itself. As we will see in Section 9.2, by using buffering and prefetching, it is possible to provide continuous playout even when the throughput fluctuates, as long as the average throughput (averaged over 5–10 seconds) remains above the video rate [Wang 2008].

For many streaming video applications, prerecorded video is stored on, and streamed from, a CDN rather than from a single data center. There are also many P2P video streaming applications for which the video is stored on users' hosts (peers), with different chunks of video arriving from different peers that may spread around the globe. Given the prominence of Internet video streaming, we will explore video streaming in some depth in Section 9.2, paying particular attention to client buffering, prefetching, adapting quality to bandwidth availability, and CDN distribution.

Conversational Voice- and Video-over-IP

Real-time conversational voice over the Internet is often referred to as **Internet telephony**, since, from the user's perspective, it is similar to the traditional circuit-switched telephone service. It is also commonly called **Voice-over-IP (VoIP)**. Conversational video is similar, except that it includes the video of the participants as well as their voices. Most of today's voice and video conversational systems allow users to create conferences with three or more participants. Conversational voice and video are widely used in the Internet today, with the Internet companies Skype, QQ, and Google Talk boasting hundreds of millions of daily users.

In our discussion of application service requirements in Chapter 2 (Figure 2.4), we identified a number of axes along which application requirements can be classified. Two of these axes—timing considerations and tolerance of data loss—are particularly important for conversational voice and video applications. Timing considerations are important because audio and video conversational applications are highly **delay-sensitive**. For a conversation with two or more interacting speakers, the delay from when a user speaks or moves until the action is manifested at the other end should be less than a few hundred milliseconds. For voice, delays smaller than 150 milliseconds are not perceived by a human listener, delays between 150 and 400 milliseconds can be acceptable, and delays exceeding 400 milliseconds can result in frustrating, if not completely unintelligible, voice conversations.

On the other hand, conversational multimedia applications are **loss-tolerant**—occasional loss only causes occasional glitches in audio/video playback, and these losses can often be partially or fully concealed. These delay-sensitive but loss-tolerant characteristics are clearly different from those of elastic data applications such as Web browsing, e-mail, social networks, and remote login. For elastic applications, long delays are annoying but not particularly harmful; the completeness and integrity of the transferred data, however, are of paramount importance. We will explore conversational voice and video in more depth in Section 9.3, paying particular attention

to how adaptive playout, forward error correction, and error concealment can mitigate against network-induced packet loss and delay.

Streaming Live Audio and Video

This third class of applications is similar to traditional broadcast radio and television, except that transmission takes place over the Internet. These applications allow a user to receive a *live* radio or television transmission—such as a live sporting event or an ongoing news event—transmitted from any corner of the world. Today, thousands of radio and television stations around the world are broadcasting content over the Internet.

Live, broadcast-like applications often have many users who receive the same audio/video program at the same time. In the Internet today, this is typically done with CDNs (Section 2.6). As with streaming stored multimedia, the network must provide each live multimedia flow with an average throughput that is larger than the video consumption rate. Because the event is live, delay can also be an issue, although the timing constraints are much less stringent than those for conversational voice. Delays of up to ten seconds or so from when the user chooses to view a live transmission to when playout begins can be tolerated. We will not cover streaming live media in this book because many of the techniques used for streaming live media—initial buffering delay, adaptive bandwidth use, and CDN distribution—are similar to those for streaming stored media.

9.2 Streaming Stored Video

For streaming video applications, prerecorded videos are placed on servers, and users send requests to these servers to view the videos on demand. The user may watch the video from beginning to end without interruption, may stop watching the video well before it ends, or interact with the video by pausing or repositioning to a future or past scene. Streaming video systems can be classified into three categories: **UDP streaming**, **HTTP streaming**, and **adaptive HTTP streaming** (see Section 2.6). Although all three types of systems are used in practice, the majority of today's systems employ HTTP streaming and adaptive HTTP streaming.

A common characteristic of all three forms of video streaming is the extensive use of client-side application buffering to mitigate the effects of varying end-to-end delays and varying amounts of available bandwidth between server and client. For streaming video (both stored and live), users generally can tolerate a small several-second initial delay between when the client requests a video and when video playout begins at the client. Consequently, when the video starts to arrive at the client, the client need not immediately begin playout, but can instead build up a reserve of video in an application buffer. Once the client has built up a reserve of several seconds of

buffered-but-not-yet-played video, the client can then begin video playout. There are two important advantages provided by such **client buffering**. First, client-side buffering can absorb variations in server-to-client delay. If a particular piece of video data is delayed, as long as it arrives before the reserve of received-but-not-yet-played video is exhausted, this long delay will not be noticed. Second, if the server-to-client bandwidth briefly drops below the video consumption rate, a user can continue to enjoy continuous playback, again as long as the client application buffer does not become completely drained.

Figure 9.1 illustrates client-side buffering. In this simple example, suppose that video is encoded at a fixed bit rate, and thus each video block contains video frames that are to be played out over the same fixed amount of time, Δ. The server transmits the first video block at t_0, the second block at $t_0 + \Delta$, the third block at $t_0 + 2\Delta$, and so on. Once the client begins playout, each block should be played out Δ time units after the previous block in order to reproduce the timing of the original recorded video. Because of the variable end-to-end network delays, different video blocks experience different delays. The first video block arrives at the client at t_1 and the second block arrives at t_2. The network delay for the ith block is the horizontal distance between the time the block was transmitted by the server and the time it is received at the client; note that the network delay varies from one video block to another. In this example, if the client were to begin playout as soon as the first block arrived at t_1, then the second block would not have arrived in time to be played out at out at $t_1 + \Delta$. In this case, video playout would either have to stall (waiting for block 2 to arrive) or block 2 could be skipped—both resulting in undesirable playout impairments. Instead, if the client were to delay the start of playout until t_3, when blocks 1 through 6 have all arrived, periodic playout can proceed with *all* blocks having been received before their playout time.

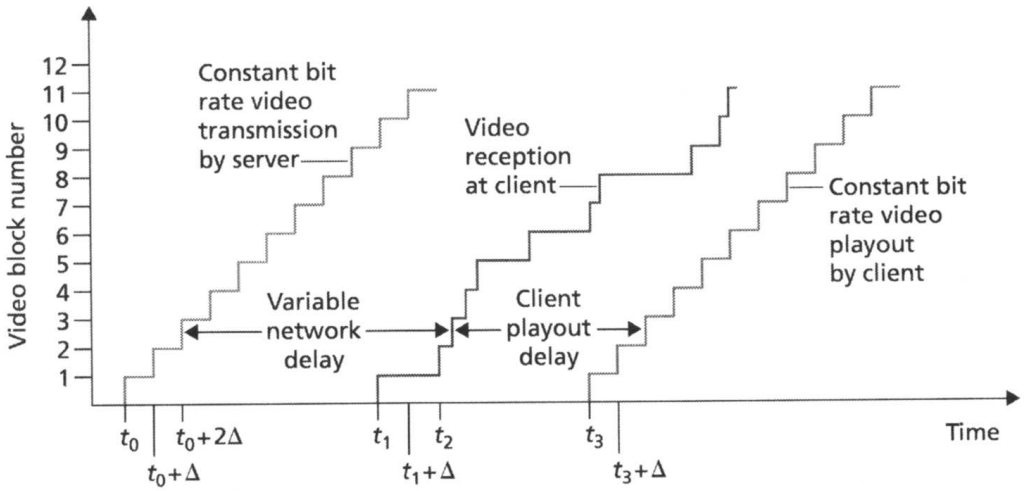


Figure 9.1 • Client playout delay in video streaming

2.5 Peer-to-Peer File Distribution

The applications described in this chapter thus far—including the Web, e-mail, and DNS—all employ client-server architectures with significant reliance on always-on infrastructure servers. Recall from Section 2.1.1 that with a P2P architecture, there is minimal (or no) reliance on always-on infrastructure servers. Instead, pairs of intermittently connected hosts, called peers, communicate directly with each other. The peers are not owned by a service provider, but are instead desktops and laptops controlled by users.

In this section we consider a very natural P2P application, namely, distributing a large file from a single server to a large number of hosts (called peers). The file might be a new version of the Linux operating system, a software patch for an existing operating system or application, an MP3 music file, or an MPEG video file. In client-server file distribution, the server must send a copy of the file to each of the peers—placing an enormous burden on the server and consuming a large amount of server bandwidth. In P2P file distribution, each peer can redistribute any portion of the

file it has received to any other peers, thereby assisting the server in the distribution process. As of 2016, the most popular P2P file distribution protocol is BitTorrent. Originally developed by Bram Cohen, there are now many different independent BitTorrent clients conforming to the BitTorrent protocol, just as there are a number of Web browser clients that conform to the HTTP protocol. In this subsection, we first examine the self-scalability of P2P architectures in the context of file distribution. We then describe BitTorrent in some detail, highlighting its most important characteristics and features.

Scalability of P2P Architectures

To compare client-server architectures with peer-to-peer architectures, and illustrate the inherent self-scalability of P2P, we now consider a simple quantitative model for distributing a file to a fixed set of peers for both architecture types. As shown in Figure 2.22, the server and the peers are connected to the Internet with access links. Denote the upload rate of the server's access link by u_s, the upload rate of the ith peer's access link by u_i, and the download rate of the ith peer's access link by d_i. Also denote the size of the file to be distributed (in bits) by F and the number of peers that want to obtain a copy of the file by N. The **distribution time** is the time it takes to get

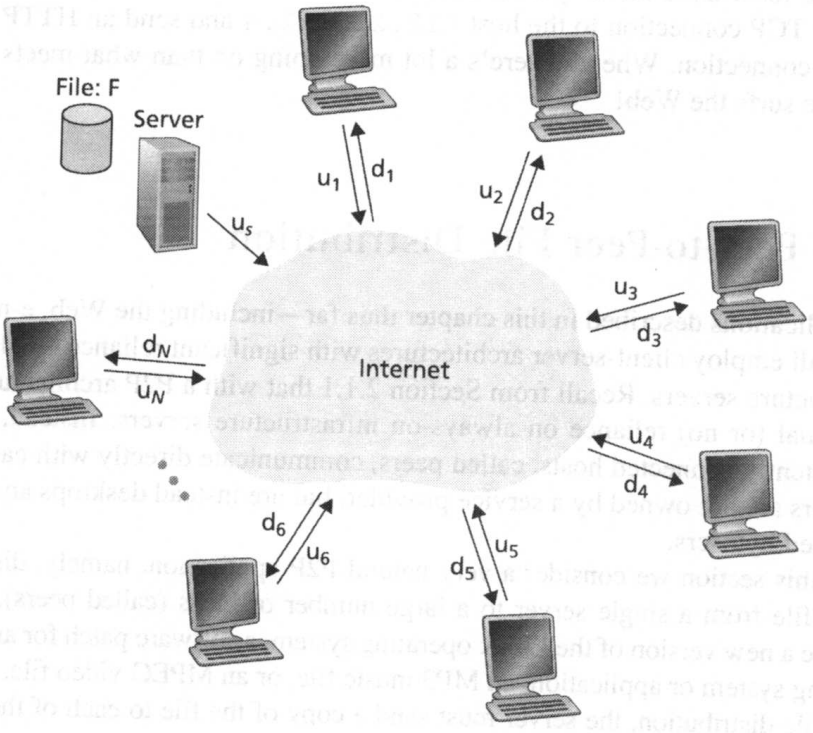


Figure 2.22 ♦ An illustrative file distribution problem

a copy of the file to all N peers. In our analysis of the distribution time below, for both client-server and P2P architectures, we make the simplifying (and generally accurate [Akella 2003]) assumption that the Internet core has abundant bandwidth, implying that all of the bottlenecks are in access networks. We also suppose that the server and clients are not participating in any other network applications, so that all of their upload and download access bandwidth can be fully devoted to distributing this file.

Let's first determine the distribution time for the client-server architecture, which we denote by D_{cs}. In the client-server architecture, none of the peers aids in distributing the file. We make the following observations:

- The server must transmit one copy of the file to each of the N peers. Thus the server must transmit NF bits. Since the server's upload rate is u_s, the time to distribute the file must be at least NF/u_s.
- Let $d_{\min}$ denote the download rate of the peer with the lowest download rate, that is, $d_{\min} = \min\{d_1, d_p, \ldots, d_N\}$. The peer with the lowest download rate cannot obtain all F bits of the file in less than $F/d_{\min}$ seconds. Thus the minimum distribution time is at least $F/d_{\min}$.

Putting these two observations together, we obtain

$$D_{cs} \geq \max\left\{\frac{NF}{u_s}, \frac{F}{d_{min}}\right\}.$$

This provides a lower bound on the minimum distribution time for the client-server architecture. In the homework problems you will be asked to show that the server can schedule its transmissions so that the lower bound is actually achieved. So let's take this lower bound provided above as the actual distribution time, that is,

$$D_{cs} = \max\left\{\frac{NF}{u_s}, \frac{F}{d_{min}}\right\} \tag{2.1}$$

We see from Equation 2.1 that for N large enough, the client-server distribution time is given by NF/u_s. Thus, the distribution time increases linearly with the number of peers N. So, for example, if the number of peers from one week to the next increases a thousand-fold from a thousand to a million, the time required to distribute the file to all peers increases by 1,000.

Let's now go through a similar analysis for the P2P architecture, where each peer can assist the server in distributing the file. In particular, when a peer receives some file data, it can use its own upload capacity to redistribute the data to other peers. Calculating the distribution time for the P2P architecture is somewhat more complicated than for the client-server architecture, since the distribution time depends on how each peer distributes portions of the file to the other peers. Nevertheless, a simple

expression for the minimal distribution time can be obtained [Kumar 2006]. To this end, we first make the following observations:

- At the beginning of the distribution, only the server has the file. To get this file into the community of peers, the server must send each bit of the file at least once into its access link. Thus, the minimum distribution time is at least F/u_s. (Unlike the client-server scheme, a bit sent once by the server may not have to be sent by the server again, as the peers may redistribute the bit among themselves.)
- As with the client-server architecture, the peer with the lowest download rate cannot obtain all F bits of the file in less than $F/d_{\min}$ seconds. Thus the minimum distribution time is at least $F/d_{\min}$.
- Finally, observe that the total upload capacity of the system as a whole is equal to the upload rate of the server plus the upload rates of each of the individual peers, that is, $u_{\text{total}} = u_s + u_1 + \cdots + u_N$. The system must deliver (upload) F bits to each of the N peers, thus delivering a total of NF bits. This cannot be done at a rate faster than u_{total}. Thus, the minimum distribution time is also at least $NF/(u_s + u_1 + \cdots + u_N)$.

Putting these three observations together, we obtain the minimum distribution time for P2P, denoted by D_{P2P}.

$$D_{\text{P2P}} \geq \max\left\{\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i}\right\} \tag{2.2}$$

Equation 2.2 provides a lower bound for the minimum distribution time for the P2P architecture. It turns out that if we imagine that each peer can redistribute a bit as soon as it receives the bit, then there is a redistribution scheme that actually achieves this lower bound [Kumar 2006]. (We will prove a special case of this result in the homework.) In reality, where chunks of the file are redistributed rather than individual bits, Equation 2.2 serves as a good approximation of the actual minimum distribution time. Thus, let's take the lower bound provided by Equation 2.2 as the actual minimum distribution time, that is,

$$D_{\text{P2P}} = \max\left\{\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i}\right\} \tag{2.3}$$

Figure 2.23 compares the minimum distribution time for the client-server and P2P architectures assuming that all peers have the same upload rate u. In Figure 2.23, we have set $F/u = 1$ hour, $u_s = 10u$, and $d_{\min} \geq u_s$. Thus, a peer can transmit the entire file in one hour, the server transmission rate is 10 times the peer upload rate,

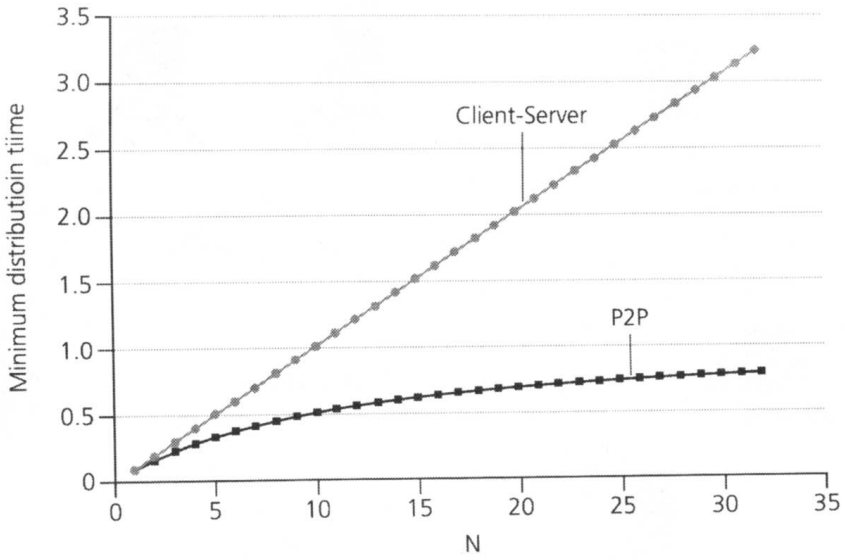


Figure 2.23 ♦ Distribution time for P2P and client-server architectures

and (for simplicity) the peer download rates are set large enough so as not to have an effect. We see from Figure 2.23 that for the client-server architecture, the distribution time increases linearly and without bound as the number of peers increases. However, for the P2P architecture, the minimal distribution time is not only always less than the distribution time of the client-server architecture; it is also less than one hour for *any* number of peers *N*. Thus, applications with the P2P architecture can be self-scaling. This scalability is a direct consequence of peers being redistributors as well as consumers of bits.

BitTorrent

BitTorrent is a popular P2P protocol for file distribution [Chao 2011]. In BitTorrent lingo, the collection of all peers participating in the distribution of a particular file is called a *torrent*. Peers in a torrent download equal-size *chunks* of the file from one another, with a typical chunk size of 256 kbytes. When a peer first joins a torrent, it has no chunks. Over time it accumulates more and more chunks. While it downloads chunks it also uploads chunks to other peers. Once a peer has acquired the entire file, it may (selfishly) leave the torrent, or (altruistically) remain in the torrent and continue to upload chunks to other peers. Also, any peer may leave the torrent at any time with only a subset of chunks, and later rejoin the torrent.

Let's now take a closer look at how BitTorrent operates. Since BitTorrent is a rather complicated protocol and system, we'll only describe its most important mechanisms, sweeping some of the details under the rug; this will allow us to see the forest through the trees. Each torrent has an infrastructure node called a *tracker*.

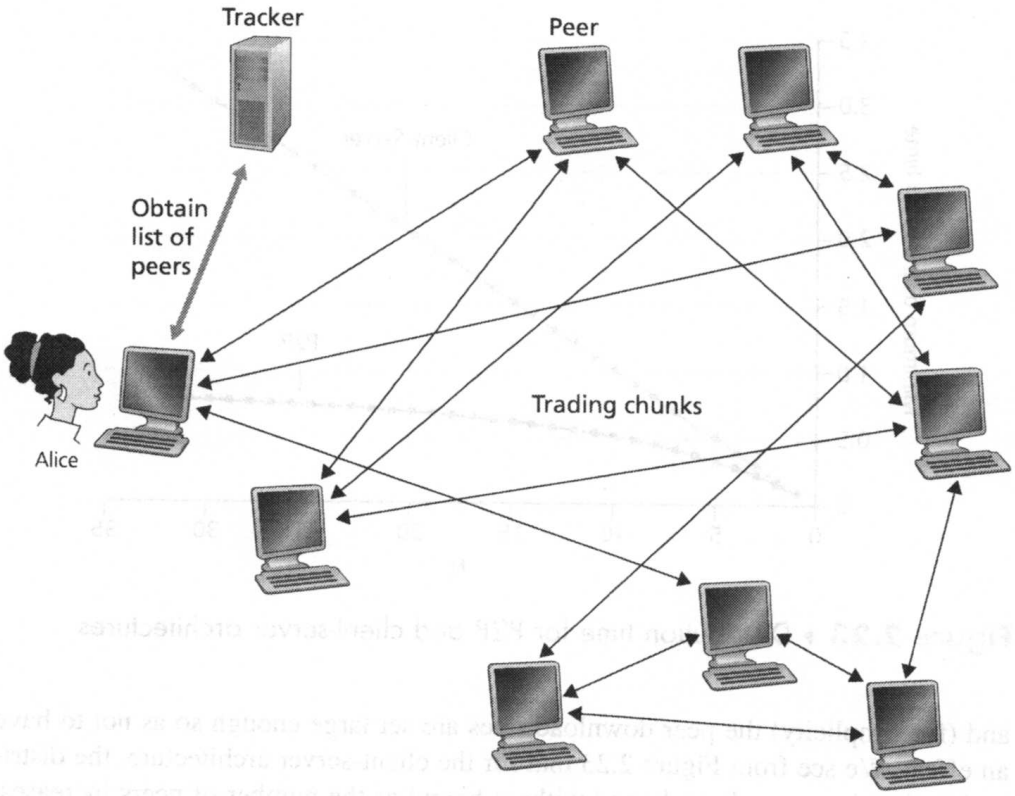


Figure 2.24 ♦ File distribution with BitTorrent

When a peer joins a torrent, it registers itself with the tracker and periodically informs the tracker that it is still in the torrent. In this manner, the tracker keeps track of the peers that are participating in the torrent. A given torrent may have fewer than ten or more than a thousand peers participating at any instant of time.

As shown in Figure 2.24, when a new peer, Alice, joins the torrent, the tracker randomly selects a subset of peers (for concreteness, say 50) from the set of participating peers, and sends the IP addresses of these 50 peers to Alice. Possessing this list of peers, Alice attempts to establish concurrent TCP connections with all the peers on this list. Let's call all the peers with which Alice succeeds in establishing a TCP connection "neighboring peers." (In Figure 2.24, Alice is shown to have only three neighboring peers. Normally, she would have many more.) As time evolves, some of these peers may leave and other peers (outside the initial 50) may attempt to establish TCP connections with Alice. So a peer's neighboring peers will fluctuate over time.

At any given time, each peer will have a subset of chunks from the file, with different peers having different subsets. Periodically, Alice will ask each of her neighboring peers (over the TCP connections) for the list of the chunks they have. If Alice has L different neighbors, she will obtain L lists of chunks. With this knowledge,

Alice will issue requests (again over the TCP connections) for chunks she currently does not have.

So at any given instant of time, Alice will have a subset of chunks and will know which chunks her neighbors have. With this information, Alice will have two important decisions to make. First, which chunks should she request first from her neighbors? And second, to which of her neighbors should she send requested chunks? In deciding which chunks to request, Alice uses a technique called **rarest first**. The idea is to determine, from among the chunks she does not have, the chunks that are the rarest among her neighbors (that is, the chunks that have the fewest repeated copies among her neighbors) and then request those rarest chunks first. In this manner, the rarest chunks get more quickly redistributed, aiming to (roughly) equalize the numbers of copies of each chunk in the torrent.

To determine which requests she responds to, BitTorrent uses a clever trading algorithm. The basic idea is that Alice gives priority to the neighbors that are currently supplying her data *at the highest rate*. Specifically, for each of her neighbors, Alice continually measures the rate at which she receives bits and determines the four peers that are feeding her bits at the highest rate. She then reciprocates by sending chunks to these same four peers. Every 10 seconds, she recalculates the rates and possibly modifies the set of four peers. In BitTorrent lingo, these four peers are said to be **unchoked**. Importantly, every 30 seconds, she also picks one additional neighbor at random and sends it chunks. Let's call the randomly chosen peer Bob. In BitTorrent lingo, Bob is said to be **optimistically unchoked**. Because Alice is sending data to Bob, she may become one of Bob's top four uploaders, in which case Bob would start to send data to Alice. If the rate at which Bob sends data to Alice is high enough, Bob could then, in turn, become one of Alice's top four uploaders. In other words, every 30 seconds, Alice will randomly choose a new trading partner and initiate trading with that partner. If the two peers are satisfied with the trading, they will put each other in their top four lists and continue trading with each other until one of the peers finds a better partner. The effect is that peers capable of uploading at compatible rates tend to find each other. The random neighbor selection also allows new peers to get chunks, so that they can have something to trade. All other neighboring peers besides these five peers (four "top" peers and one probing peer) are "choked," that is, they do not receive any chunks from Alice. BitTorrent has a number of interesting mechanisms that are not discussed here, including pieces (mini-chunks), pipelining, random first selection, endgame mode, and anti-snubbing [Cohen 2003].

The incentive mechanism for trading just described is often referred to as tit-for-tat [Cohen 2003]. It has been shown that this incentive scheme can be circumvented [Liogkas 2006; Locher 2006; Piatek 2007]. Nevertheless, the BitTorrent ecosystem is wildly successful, with millions of simultaneous peers actively sharing files in hundreds of thousands of torrents. If BitTorrent had been designed without tit-for-tat (or a variant), but otherwise exactly the same, BitTorrent would likely not even exist now, as the majority of the users would have been freeriders [Saroiu 2002].

Where a layman might have said that the fundamental problem of communication is to make oneself understood—to convey meaning—Shannon set the stage differently:

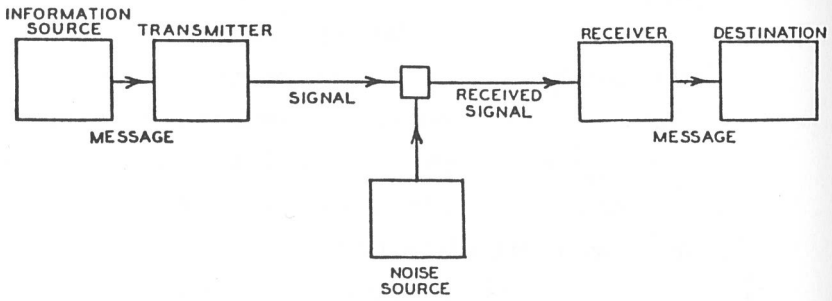
> The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point.

"Point" was a carefully chosen word: the origin and destination of a message could be separated in space or in time; information storage, as in a phonograph record, counts as a communication. Meanwhile, the message is not created; it is selected. It is a choice. It might be a card dealt from a deck, or three decimal digits chosen from the thousand possibilities, or a combination of words from a fixed code book. He could hardly overlook meaning altogether, so he dressed it with a scientist's definition and then showed it the door:

> Frequently the messages have *meaning;* that is they refer to or are correlated according to some system with certain physical or conceptual entities. These semantic aspects of communication are irrelevant to the engineering problem.

Nonetheless, as Weaver took pains to explain, this was not a narrow view of communication. On the contrary, it was all-encompassing: "not only written and oral speech, but also music, the pictorial arts, the theatre, the ballet, and in fact all human behavior." Nonhuman as well: why should machines not have messages to send?

Shannon's model for communication fit a simple diagram—essentially the same diagram, by no coincidence, as in his secret cryptography paper.



A communication system must contain the following elements:

- The information source is the person or machine generating the message, which may be simply a sequence of characters, as in a telegraph or teletype, or may be expressed mathematically as functions—$f(x, y, t)$—of time and other variables. In a complex example like color television, the components are three functions in a three-dimensional continuum, Shannon noted.
- The transmitter "operates on the message in some way"—that is, *encodes* the message—to produce a suitable signal. A telephone converts sound pressure into analog electric current. A telegraph encodes characters in dots, dashes, and spaces. More complex messages may be sampled, compressed, quantized, and interleaved.
- The channel: "merely the medium used to transmit the signal."
- The receiver inverts the operation of the transmitter. It decodes the message, or reconstructs it from the signal.
- The destination "is the person (or thing)" at the other end.

In the case of ordinary speech, these elements are the speaker's brain, the speaker's vocal cords, the air, the listener's ear, and the listener's brain.

As prominent as the other elements in Shannon's diagram—because for an engineer it is inescapable—is a box labeled "Noise Source." This covers everything that corrupts the signal, predictably or unpredictably: unwanted additions, plain errors, random disturbances, static, "atmospherics," interference, and distortion. An unruly family under any circumstances, and Shannon had two different types of systems to deal with, continuous and discrete. In a discrete system, message and signal take the form of individual detached symbols, such as characters or digits or dots and dashes. Telegraphy notwithstanding, continuous systems of waves and functions were the ones facing electrical engineers every day. Every engineer, when asked to push more information through a channel, knew what to do: boost the power. Over long distances, however, this approach was failing, because amplifying a signal again and again leads to a crippling buildup of noise.

Shannon sidestepped this problem by treating the signal as a string of discrete symbols. Now, instead of boosting the power, a sender can overcome noise by using extra symbols for error correction—just as an African drummer makes himself understood across long distances, not by banging the drums harder, but by expanding the verbosity of his discourse. Shannon considered the discrete case to be more fundamental in a mathematical sense as well. And he was considering another point: that treating messages as discrete had application not just for traditional communication but for a new and rather esoteric subfield, the theory of computing machines.

The point was to represent a message as the outcome of a process that generated events with discrete probabilities. Then what could be said about the amount of information, or the rate at which information is generated? For each event, the possible choices each have a known probability (represented as p_1, p_2, p_3, and so on). Shannon wanted to define the measure of information (represented as H) as the measure of uncertainty: "of how much 'choice' is involved in the selection of the event or of how uncertain we are of the outcome." The probabilities might be the same or different, but generally more choices meant more uncertainty—more information. Choices might be broken down into successive choices, with their own probabilities, and the probabilities had to be additive; for example, the probability of a particular digram should be a weighted sum of the probabilities of the individual symbols. When those probabilities were equal, the amount of information conveyed by each symbol was simply the logarithm of the number of possible symbols—Nyquist and Hartley's formula:

$$H = n \log s$$

For the more realistic case, Shannon reached an elegant solution to the problem of how to measure information as a function of probabilities—an equation that summed the probabilities with a logarithmic weighting (base 2 was most convenient). It is the average logarithm of the improbability of the message; in effect, a measure of unexpectedness:

$$H = -\sum p_i \log_2 p_i$$

where p_i is the probability of each message. He declared that we would be seeing this again and again: that quantities of this form "play a central role in information theory as measures of information, choice, and

uncertainty." Indeed, H is ubiquitous, conventionally called the entropy of a message, or the Shannon entropy, or, simply, the information.

A new unit of measure was needed. Shannon said: "The resulting units may be called binary digits, or more briefly, *bits*." As the smallest possible quantity of information, a bit represents the amount of uncertainty that exists in the flipping of a coin. The coin toss makes a choice between two possibilities of equal likelihood: in this case p_1 and p_2 each equal ½; the base 2 logarithm of ½ is −1; so H = 1 bit. A single character chosen randomly from an alphabet of 32 conveys more information: 5 bits, to be exact, because there are 32 possible messages and the logarithm of 32 is 5. A string of 1,000 such characters carries 5,000 bits—not just by simple multiplication, but because the amount of information represents the amount of uncertainty: the number of possible choices. With 1,000 characters in a 32-character alphabet, there are 32^{1000} possible messages, and the logarithm of that number is 5,000.

This is where the statistical structure of natural languages reenters the picture. If the thousand-character message is known to be English text, the number of possible messages is smaller—*much* smaller. Looking at correlations extending over eight letters, Shannon estimated that English has a built-in redundancy of about 50 percent: that each new character of a message conveys not 5 bits but only about 2.3. Considering longer-range statistical effects, at the level of sentences and paragraphs, he raised that estimate to 75 percent—warning, however, that such estimates become "more erratic and uncertain, and they depend more critically on the type of text involved." One way to measure redundancy was crudely empirical: carry out a psychology test with a human subject. This method "exploits the fact that anyone speaking a language possesses, implicitly, an enormous knowledge of the statistics of the language."

> Familiarity with the words, idioms, clichés and grammar enables him to fill in missing or incorrect letters in proof-reading, or to complete an unfinished phrase in conversation.

He might have said "her," because in point of fact his test subject was his wife, Betty. He pulled a book from the shelf (it was a Raymond Chandler detective novel, *Pickup on Noon Street*), put his finger on a short passage at random, and asked Betty to start guessing the letter, then the next letter, then the next. The more text she saw, of course, the better her chances of guessing right. After "A SMALL OBLONG READING LAMP ON THE" she got the next letter wrong. But once she knew it was D, she had no trouble guessing the next three letters. Shannon observed, "The errors, as would be expected, occur most frequently at the beginning of words and syllables where the line of thought has more possibility of branching out."

Quantifying predictability and redundancy in this way is a backward way of measuring information content. If a letter can be guessed from what comes before, it is redundant; to the extent that it is redundant, it provides no new information. If English is 75 percent redundant, then a thousand-letter message in English carries only 25 percent as much information as one thousand letters chosen at random. Paradoxical though it sounded, random messages carry *more* information. The implication was that natural-language text could be encoded more efficiently for transmission or storage.

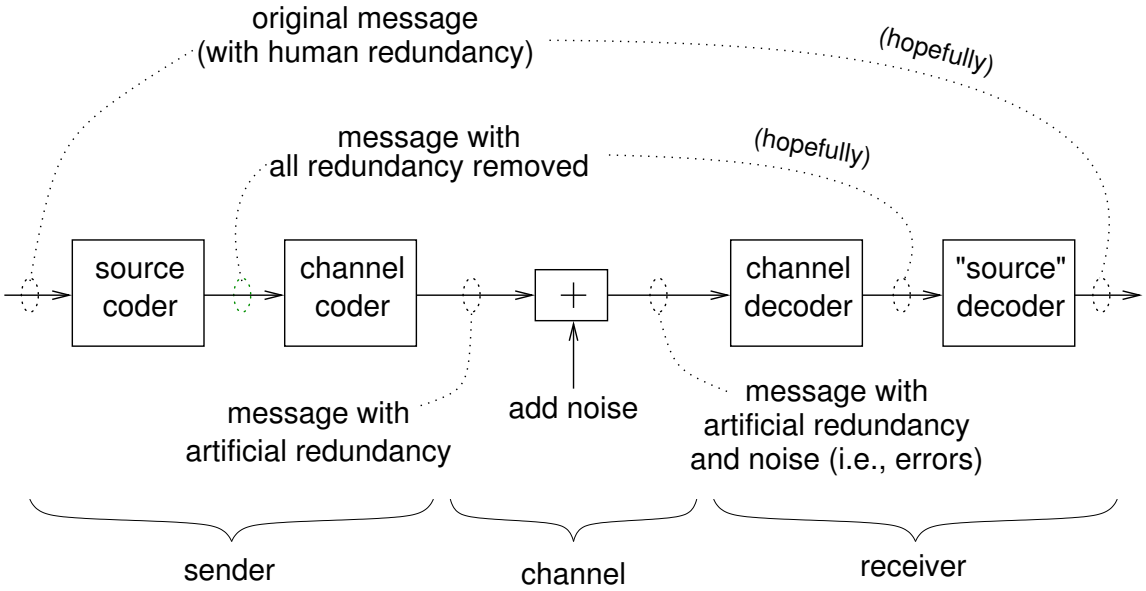
Shannon demonstrated one way to do this, an algorithm that exploits differing probabilities of different symbols. And he delivered a stunning package of fundamental results. One was a formula for channel capacity, the absolute speed limit of any communication channel (now known simply as the Shannon limit). Another was the discovery that, within that limit, it must always be possible to devise schemes of error correction that will overcome any level of noise. The sender may have to devote more and more bits to correcting errors, making transmission slower and slower, but the message will ultimately get through. Shannon did not show how to design such schemes; he only proved that it was possible, thereby inspiring a future branch of computer science. "To make the chance of error as small as you wish? Nobody had thought of that," his colleague Robert Fano recalled years later. "How he got that insight, how he came to believe such a thing, I don't know. But almost all

modern communication theory is based on that work." Whether removing redundancy to increase efficiency or adding redundancy to enable error correction, the encoding depends on knowledge of the language's statistical structure to do the encoding. Information cannot be separated from probabilities. A bit, fundamentally, is always a coin toss.

5 Shannon's information and communication theory

The previous section has introduced Shannon's results mostly in qualitative terms. In this section, we state a few of Shannon's key results in a more quantitative way, and discuss what they mean in practice.[1]

5.1 A general view of a communication system



The above diagram shows a general but idealized view of a communication system. Messages from a (human) user enter at the left, and are delivered at the right, after going through sender, channel, and receiver.

First, *source coding* is applied, which removes all redundancy. According to Shannon's measure of information, there is a well-defined lower bound on how many bits the messages can be compressed to. See Section 5.2.

Next, *channel coding* is applied. This adds carefully chosen redundancy to the messages, which makes it possible for the receiving side to detect and correct many or most of the errors that may have be introduced by the channel. This is discussed further in Section 5.5.

The result of channel coding is transported over the channel, which may add some "noise", e.g., bit errors. According to Shannon's results, every channel has some well-defined capacity: an amount of information that can be transported over it per unit of time, essentially without errors, if a good enough channel coding is employed. See Section 5.3.

At the receiving side, two steps of decoding are needed, corresponding to the two steps of encoding used at the sending side. The first one, channel decoding, corrects any errors introduced by the channel noise. The second uncompresses the data to get the original messages back.

5.2 Message entropy and the source-coding theorem

This is Shannon's source-coding theorem:

> Every (memoryless) source of (random) messages can be characterized by its "entropy" or "information per message", denoted by H and expressed in bits. The messages can be encoded such, that on average not more than H bits are needed per message.

[1] If you want to learn more about this topic, reading "The mathematical theory of communication" by Claude E. Shannon and Warren Weaver (originally from 1949, but reprinted many times and also available on-line) is recommended.

Example: Consider a source of weather reports, which can either be "snow", "rain", "clouds" or "sun", with probabilities of 12.5 %, 25%, 50% and 12.5%, respectively. There are four different messages, so we could just use two bits to encode them, as in the "trivial code" in the following table:

message	probability	trivial code	better code
snow	12.5 %	00	010
rain	25 %	01	00
clouds	50 %	10	1
sun	12.5 %	11	011

The table also shows a better code, which has a longer bit pattern for the less likely messages. This code is better because it needs, on average, fewer bits per message, namely:

$$0.125 \cdot 3 + 0.25 \cdot 2 + 0.5 \cdot 1 + 0.125 \cdot 3 = 1.75 \text{ bits per message.}$$

The entropy of this source, using the formula from page 24 of this reader, is:

$$H = 0.125 \log_2 \left(\frac{1}{0.125}\right) + 0.25 \log_2 \left(\frac{1}{0.25}\right) + 0.50 \log_2 \left(\frac{1}{0.50}\right) + 0.125 \log_2 \left(\frac{1}{0.125}\right) = 1.75 \text{ bits.}$$

So according to the theorem, the "better" code is not just better, it is in fact an optimal one! You cannot do better than this. Note though that in reality, it is not always so easy to find an optimal code.

Source coding is also known as *data compression*, as done e.g. by software tools such as ZIP. Also audio and image compression, as used in dataformats such as `.mp3` and `.jpg`, is a form of source coding, although these forms are often *lossy*: after decompressing, one gets a file which is not exactly bit-by-bit identical to the original, but the differences are so small as to be acceptable (or even unnoticeable) to the human user.

5.3 Channel capacity and the channel coding theorem

Another important concept in Shannon's theory is the "channel": a thing that transports signals from one place to another, and possibly distorts these signals, e.g., by adding noise.

A channel is an abstract notion, of which the practical realization can have many forms, such as electrical voltages travelling along a copper cable, sound vibrations travelling through the air, but also a piece of paper with text written on it. In each of these examples, one can imagine distortions to occur: noise from nearby electrical equipment on the copper cable, wind noise disturbing the sound, or dirt accumulating on the paper. All of these make the received signal different from the transmitted signal.

One way to make such a channel more reliable, is to add redundancy to the signal being sent, for example by writing the message in bigger letters on the paper, or writing it three times. This makes it more likely that the message can still be recognized at the receiving side, but of course means that fewer messages can be sent per unit of time. Clearly, there is a trade-off: more redundancy means less probability of error, but also means less data per second.

Shannon's crucial result (see also the lower half of page 26 of this reader) is quantifying this trade-off, and showing that in order to get the probability of error down to zero, the amount of redundancy needed does not become infinitely large. More precisely, he concluded:

> Every channel has some capacity C. Given an information source generating R messages per second, with an entropy of H bits per message, then if $R \cdot H < C$, there exists an encoding by which these messages can be transmitted with an arbitrarily low probability of error; conversely, if $R \cdot H > C$, such an encoding does not exist.

In other words, C is a fundamental limit on how fast information can be squeezed through the channel. If you need to send less than C bits per second, you can make the probability of error as low as you want; if your needs are more than C, then this is impossible.

5.4 The binary symmetric channel

To make this more concrete, consider the so-called binary symmetric channel. This is a channel which transports bits, i.e., 0s and 1s, but each bit has a probability p of being received incorrectly (0 becomes 1, 1 becomes 0). This probability of error is the same for 0 and 1, hence the word "symmetric" in the name. Furthermore, the errors of consecutive bits sent through this channel are independent (e.g., if the 5th bit was received with error, this does not influence the probability of the 6th bit also being received incorrectly.)

Shannon showed that the capacity of such a channel is given by:

$$C = R \cdot \left[1 - p \log_2\left(\frac{1}{p}\right) - (1-p)\log_2\left(\frac{1}{1-p}\right)\right]$$

where R is the number of bits transported by the channel.

Example: Suppose we have a binary symmetric channel which transports 1000 bits per second, and which has a bit error probability of 10 %. Then the capacity C = 531.0 bits/s. Now suppose we want to actually send 200 bits of information per second over this channel, and cannot tolerate an error probability of more than 0.01 %. The theorem says that it is possible to devise a way of encoding these 200 bits per second into 1000 bits/s, and send them through the channel, where on average 10 % of them will be flipped, and still at the receiving side recover the original 200 bits per seconde with not more than 0.01 % of error. And this also possible for not just 200, but also 400 bits per second, or 530 bits per second, and also for tolerable error rates of just 0.001 %, or 0.0000001 %; however, for these higher bit rates and lower error rates, the encoding and decoding will become more complicated Furthermore, it is *not* possible for 532 bits/s (or more), or to guarantee that the bit error rate becomes exactly 0 (i.e., never any bit error).

But what should that encoding look like? Unfortunately, Shannon did not tell us that; he only proved that it is *possible*... It took scientists over 40 years to actually come up with encoding schemes that are really close to Shannon's limit.

5.5 Error correcting codes

An "Error Correcting Code" is a way to encode a message such that even if one or more bits are received incorrectly, the message can still be recovered correctly. Using such a code allows one to reduce the bit-error-rate; in fact, suitable coding can reduce the bit error rate to an arbitrarily low value (i.e., correct almost all bit errors), as long as the data rate of the actual messages is lower than Shannon's channel capacity (see previous section).

Encoding a message such that bit errors can be corrected necessarily involves sending more bits than just the bits of the message itself. So for a k-bit message, one sends say n bits, with $n > k$. Thus, the code is specified by a table or algorithm that gives for every k-bit message, the set of n bits that need to be trasmitted. Since there are 2^k possible messages, which is less than the number of possible patterns of n bits, namely 2^n, many n-bit patterns do not correspond to any message. If such a pattern is received, it can be concluded that there must be at least one bit error: *error detection*. In order to *correct* such a bit error, the receiver then searches for a valid n-bit pattern that differs from the received n-bit pattern in the smallest number of bits. This relies on the assumption that bit errors are rare, so the valid pattern which matches with the smallest number of bit differences is the most likely to be the correct one.

Let us consider a few simple (but not so powerful) error correcting codes.

Repetition code

A very trivial code is the repetition code: simply send each bit several times, e.g., 3 times. If a bit is sent at least 3 times, the receiver can correct a single bit error by doing a majority decision on the 3 received copies of the bit. However, this is rather inefficient: for sending k data bits, it sends a total of $n = 3k$ bits on the channel.

Example: In the case of our binary symmetric channel with 1000 bits/s and error probability 10%, using a 3-fold repetition code would allow us to transport 1000/3=333.3 bits/s, with the error probability reduced to 2.8 %.

Parity matrix

The simplest form of error detection is provided by adding a single bit to the data. This single bit is called a parity bit, and it is chosen such that the total number of ones becomes even (or odd, that's up to the system designer to decide). The receiver can then count the number of received ones, and flag the data as erroneous if that number is odd.

This principle can be extended into an error *correction* code as follows. Take the bits to be transmitted, and arrange them into a square. At the end of each row, put a parity bit making the number of 1s in that row even, and similarly for each column. An example of 16 databits arranged in a square (on white background), giving 9 parity bits (grey background), is as follows:

0	1	1	0	0
0	1	1	1	1
0	0	0	1	1
1	0	1	0	0
1	0	1	0	0

Assume that a single bit in the matrix is received incorrectly. Upon checking, the receiver will then find that in exactly one column and one row the parity checks fails. Assuming that a single bit error is the most likely cause, it can conclude that the bit that is at the crossing of that row and column must be corrected.

Thus, we see that the parity matrix can correct single bit errors. One easily verifies that for sending k bits of data (where k must be a square, 16 in the above example), it sends $n = (\sqrt{k}+1)^2 = k + 2\sqrt{k} + 1$ bits (25 in the above example) on the channel. This is already much more efficient than the repetition code, especially for large k.

Note though that for $k > 1$, the repetition code can also correct *some* cases where more than one bit is received incorrectly, namely when these bit errors happen to hit different data bits. The parity matrix applied to the entire block cannot do this.

Example: In the case of our binary symmetric channel with 1000 bits/s and error probability 10%, using a 3 by 3 parity matrix allows us to transmit $4/9 \cdot 1000 = 444.4$ user data bits per second, at a bit error probability of about 5 %.

Better codes

In practice, much more powerful codes are used, going by names such as "Hamming[2]" "Reed-Solomon", "Turbo" and "Low-Density Parity Check" codes. They work on large blocks of data and are able to correct many bit errors in such blocks. Understanding most of them requires advanced mathematics (in particular algebra) and is therefore outside the scope of this course.

Turbo codes are a relatively recent[3] invention (1993); the idea behind them is to encode the same message using two different codes, and having the decoders for both codes exchange information, such that they gradually converge even in the presence of relatively many bit errors. They were the first codes to get very close to the Shannon limit, much closer than the best codes previously known. Since then, other codes coming close to the Shannon limit have been developed, in particular the so-called Low-Density Parity Check codes.

Note that there is a disadvantage to using error-correcting codes: much processing is needed, mostly on the receiving side. Therefore, error correction can only be used in situations where the data rate is not too high. Of

[2]Hamming codes will be discussed in the math case study in week 4 of the module.

[3]Recent is relative here, as 1993 is already more than a quarter century ago. However, Shannon's work dates back to the late 1940s, and much work on algebraic coding was done in the 1950s and 1960s. The development of Turbo codes thus came as a surprise in an otherwise well-established field.

course, what is "too high" evolves as technology advances. Nowadays, systems up to many megabits/second can use error correction, e.g. ADSL lines, GSM, UMTS. Really high-speed lines, like optical backbones of many gigabits/second, typically do not have error correction, but are run at such a high signal-to-noise ratio that the bit error rate is acceptably low even without coding.

5.6 Analog channels

For analog channels (e.g., a cable transporting an electrical voltage), the capacity is (in simple cases) given by the following formula:

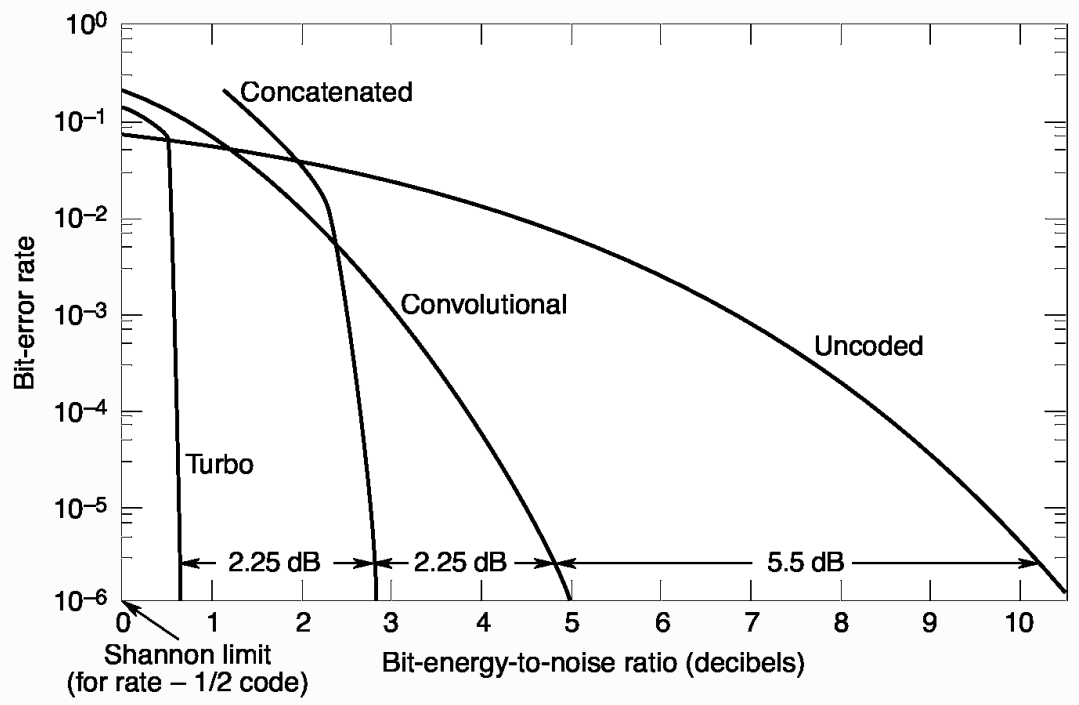
$$C = B \log_2 \left(1 + \frac{S}{N}\right)$$

where B is the *bandwidth* of the channel (roughly, a measure for how quick the signal is allowed to vary), S a measure for the strength of the signal, and N a measure for the noise level on the channel. (These are terms from electrical engineering; their precise meaning is beyond the scope of this course.)

One sees that if the signal can change more frequently, the capacity is higher, which intuitively makes sense.

One also sees that when the signal is stronger compared to the noise, the capacity is higher. That also intuitively makes sense: if the noise is smaller, then more different signal levels can be reliably distinguished.

5.7 Coding and the Shannon limit



The above figure[4] shows the bit-error rate (BER, i.e. the fraction of bits that are received incorrectly due to noise) as a function of the received signal-to-noise ratio[5], for several coding schemes.

Clearly, each of the curves has a negative slope; this is to be expected, since with a higher signal-to-noise ratio, one expects fewer bit errors. One also sees that coding gives a gain. For example, at a bit error rate of $2 \cdot 10^{-6}$, the use of the "convolutional" code reduces the needed signal strength by 5.5 dB compared to not using any error correction. One also sees that the Turbo code comes pretty close to the Shannon limit.

[4] from: http://www.aero.org/publications/crosslink/pdfs/V3N1.pdf

[5] more precisely, for electrical engineers: the amount of energy spent per data bit, divided by the noise power spectral density.

On the other hand, at very low signal to noise ratios, coding may actually increase the bit error rate: in such a case, there are so many bit errors, that the decoder makes wrong "correction" decisions. As a consequence, the curves for the more advanced coding schemes have a rather sharp cliff-like shape: if the signal is strong enough, there are almost no bit errors, but it it is only a llittle too weak, there are very many bit errors.

It should be noted that this figure is only an illustration, for some specific choices of the modulation method and the codes used. With other modulation types and other codes, the curves would be different. However, the general picture is always the same: with coding, a weaker signal is sufficient for the same small bit error rate.

Note that it is not by itself obvious that the use of an error correcting code indeed gives a gain. This is because every error correcting code introduces extra bits (e.g., parity bits) which also need to be transmitted. Thus, the available transmit power is divided over more bits, which means less energy per bit, and thus a larger probability for each of them to be received incorrectly. So at first, adding the redundant bits *increases* the bit error rate. However, a good error correcting code *more than makes up* for this: it can correct so many bit errors, that the final bit error rate is less than when only the data bits were sent with no coding.

7

Transmission Media

7.1 Introduction

Chapter 5 provides an overview of data communications. The previous chapter considers the topic of information sources. The chapter examines analog and digital information, and explains encodings.

This chapter continues the discussion of data communications by considering transmission media, including wired, wireless, and optical media. The chapter gives a taxonomy of media types, introduces basic concepts of electromagnetic propagation, and explains how shielding can reduce or prevent interference and noise. Finally, the chapter explains the concept of capacity. Successive chapters continue the discussion of data communications.

7.2 Guided And Unguided Transmission

How should transmission media be divided into classes. There are two broad approaches:

- By type of path: communication can follow an exact path such as a wire, or can have no specific path, such as a radio transmission.
- By form of energy: electrical energy is used on wires, radio transmission is used for wireless, and light is used for optical fiber.

We use the terms *guided* and *unguided* transmission to distinguish between physical media such as copper wiring or optical fibers that provide a specific path and a radio transmission that travels in all directions through free space. Informally, engineers use the terms *wired* and *wireless*. Note that the informality can be somewhat confusing because one is likely to hear the term *wired* even when the physical medium is an optical fiber.

7.3 A Taxonomy By Forms Of Energy

Figure 7.1 illustrates how physical media can be classified according to the form of energy used to transmit data. Successive sections describe each of the media types.

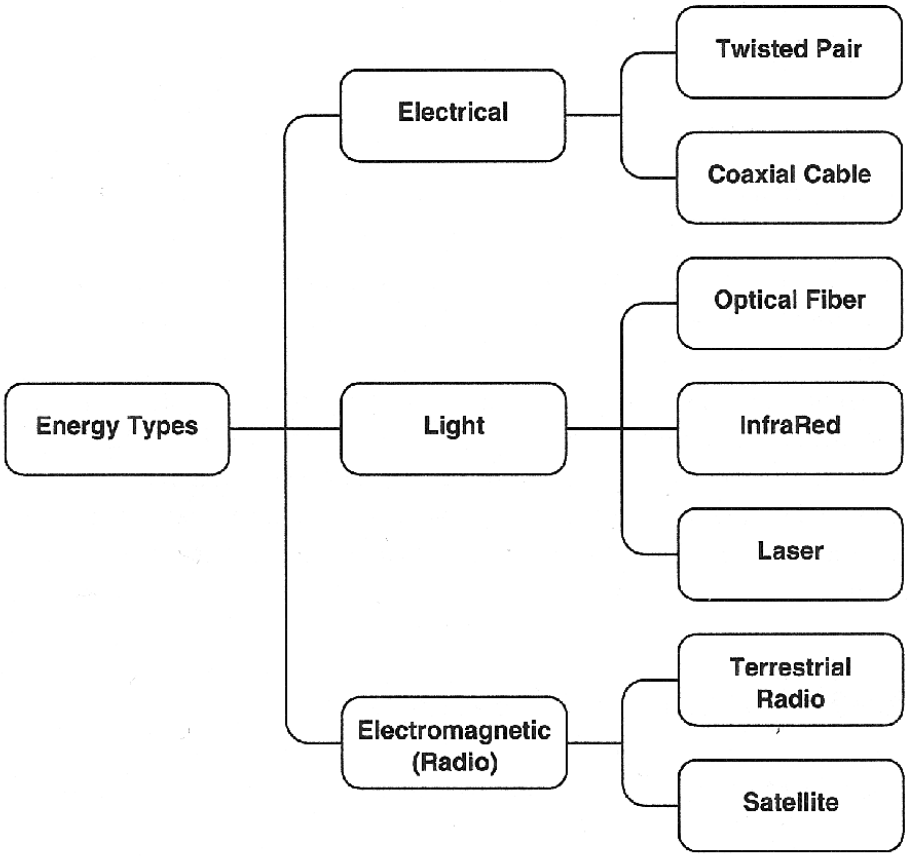


Figure 7.1 A taxonomy of media types according to the form of energy used.

Like most taxonomies, the categories are not perfect, and exceptions exist. For example, a space station in orbit around the earth might employ non-terrestrial communication that does not involve a satellite. Nevertheless, our taxonomy covers most communications.

7.4 Background Radiation And Electrical Noise

Recall from basic physics that electrical current flows along a complete circuit. Thus, all transmissions of electrical energy need two wires to form a circuit — a wire to the receiver and a wire back to the sender. The simplest form of wiring consists of a cable that contains two copper wires. Each wire is wrapped in a plastic coating, which insulates the wires electrically. The outer coating on the cable holds related wires together to make it easier for humans who connect equipment.

Computer networks use an alternative form of wiring. To understand why, one must know three facts.

1. Random electromagnetic radiation, called *noise*, permeates the environment. In fact, communication systems generate minor amounts of electrical noise as a side-effect of normal operation.
2. When it hits metal, electromagnetic radiation induces a small signal, which means that random noise can interfere with signals used for communication.
3. Because it absorbs radiation, metal acts as a *shield*. Thus, placing enough metal between a source of noise and a communication medium can prevent noise from interfering with communication.

The first two facts outline a fundamental problem inherent in communication media that use electrical or radio energy. The problem is especially severe near a source that emits random radiation. For example, florescent light bulbs and electric motors both emit radiation, especially powerful motors such as those used to operate elevators, air conditioners, and refrigerators. Surprisingly, smaller devices such as paper shredders or electric power tools can also emit enough radiation to interfere with communication. The point is:

The random electromagnetic radiation generated by devices such as electric motors can interfere with communication that uses radio transmission or electrical energy sent over wires.

7.5 Twisted Pair Copper Wiring

The third fact in the previous section explains the wiring used with communication systems. There are three forms of wiring that help reduce interference from electrical noise.

- Unshielded Twisted Pair (UTP)
- Coaxial Cable
- Shielded Twisted Pair (STP)

The first form, which known as *twisted pair* wiring or *unshielded twisted pair* wiring†, is used extensively in communications. As the name implies, twisted pair wiring consists of two wires that are twisted together. Of course, each wire has a plastic coating that insulates the two wires and prevents electrical current from flowing between them.

Surprisingly, twisting two wires makes them less susceptible to electrical noise than leaving them parallel. Figure 7.2 illustrates why.

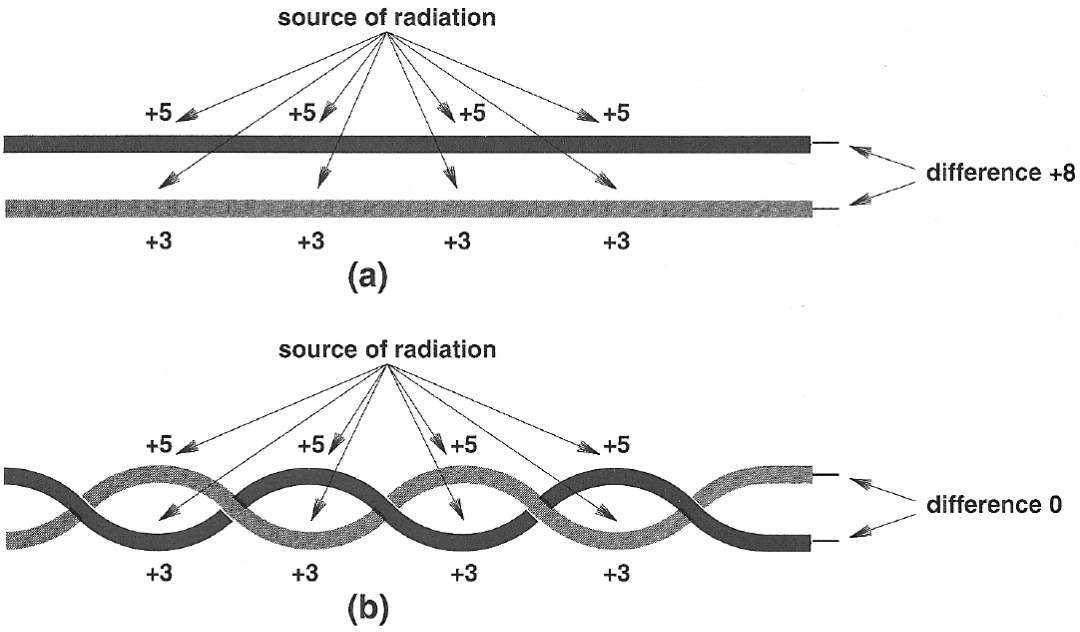


Figure 7.2 Unwanted electromagnetic radiation affecting (a) two parallel wires, and (b) twisted pair wiring.

As the figure shows, when two wires are in parallel, there is a high probability that one of them is closer to the source of electromagnetic radiation than the other. In fact, one wire tends to act as a shield that absorbs some of the electromagnetic radiation. Thus, because it is hidden behind the first wire, the second wire receives less energy. In the figure, a total of 32 units of radiation strikes each of the two cases. In Figure 7.2a, the top wire absorbs 20 units, and the bottom wire absorbs 12, producing a difference of 8. In Figure 7.2b, each of the two wires is on top one-half of the time, which means each wire absorbs the same amount of radiation.

Why does equal absorption matter? The answer is that if interference induces exactly the same amount of electrical energy in each wire, no extra current will flow. Thus, the original signal will not be disturbed. The point is:

†A later section explains the term *shielded*.

To reduce the interference caused by random electromagnetic radiation, communication systems use twisted pair wiring rather than parallel wires.

7.6 Shielding: Coaxial Cable And Shielded Twisted Pair

Although it is immune to most background radiation, twisted pair wiring does not solve all problems. Twisted pair tends to have problems with:

- Especially strong electrical noise
- Close physical proximity to the source of noise
- High frequencies used for communication

If the intensity is high (e.g., in a factory that uses electric arc welding equipment) or communication cables run close to the source of electrical noise, even twisted pair may not be sufficient. Thus, if a twisted pair runs above the ceiling in an office building on top of a florescent light fixture, interference may result. Furthermore, it is difficult to build equipment that can distinguish between valid high frequency signals and noise, which means that even a small amount of noise can cause interference when high frequencies are used.

To handle situations where twisted pair is insufficient, forms of wiring are available that have extra metal shielding. The most familiar form is the wiring used for cable television. Known as *coaxial cable* (*coax*), the wiring has a thick metal shield formed from braided wires that completely surround a center wire that carries the signal. Figure 7.3 illustrates the concept.

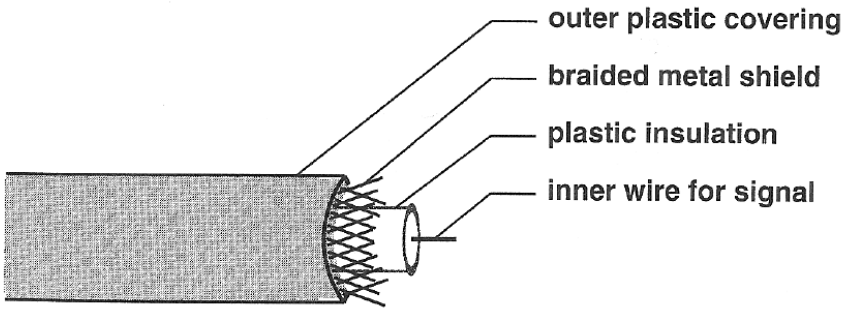


Figure 7.3 Illustration of coaxial cable with a shield surrounding the signal wire.

The shield in a coaxial cable forms a flexible cylinder around the inner wire that provides a barrier to electromagnetic radiation from any direction. The barrier also

prevents signals on the inner wire from radiating electromagnetic energy that could affect other wires. Consequently, a coaxial cable can be placed adjacent to sources of electrical noise and other cables, and can be used for high frequencies. The point is:

> *The heavy shielding and symmetry makes coaxial cable immune to noise, capable of carrying high frequencies, and prevents signals on the cable from emitting noise to surrounding cables.*

Using braided wire instead of a solid metal shield keeps coaxial cable flexible, but the heavy shield does make coaxial cable less flexible than twisted pair wiring. Variations of shielding have been invented that provide a compromise: the cable is more flexible, but has slightly less immunity to electrical noise. One popular variation is known as *shielded twisted pair* (*STP*). The cable has a thinner, more flexible metal shield surrounding one or more twisted pairs of wires. In most versions of STP cable, the shield consists of metal foil, similar to the aluminum foil used in a kitchen. STP cable has the advantages of being more flexible than a coaxial cable and less susceptible to electrical interference than *unshielded twisted pair* (*UTP*).

7.7 Categories Of Twisted Pair Cable

The telephone companies originally specified standards for twisted pair wiring used in the telephone network. More recently, three standards organizations worked together to create standards for twisted pair cables used in computer networks. The *American National Standards Institute* (*ANSI*), the *Telecommunications Industry Association* (*TIA*), and the *Electronic Industries Alliance* (*EIA*) created a list of wiring categories, with strict specifications for each. Figure 7.4 summarizes the main categories.

Category	Description	Data Rate (in Mbps)
CAT 1	Unshielded twisted pair used for telephones	< 0.1
CAT 2	Unshielded twisted pair used for T1 data	2
CAT 3	Improved CAT2 used for computer networks	10
CAT 4	Improved CAT3 used for Token Ring networks	20
CAT 5	Unshielded twisted pair used for networks	100
CAT 5E	Extended CAT5 for more noise immunity	125
CAT 6	Unshielded twisted pair tested for 200 Mbps	200
CAT 7	Shielded twisted pair with a foil shield around the entire cable plus a shield around each twisted pair	600

Figure 7.4 Twisted pair wiring categories and a description of each.

7.8 Media Using Light Energy And Optical Fibers

According to the taxonomy in Figure 7.1, three forms of media use light energy to carry information:

- Optical fibers
- InfraRed transmission
- Point-to-point lasers

The most important type of media that uses light is an *optical fiber*. Each fiber consists of a thin strand of glass or transparent plastic encased in a plastic cover. A typical optical fiber is used for communication in a single direction — one end of the fiber connects to a laser or LED used to transmit light, and the other end of the fiber connects to a photosensitive device used to detect incoming light. To provide two-way communication, two fibers are used, one to carry information in each direction. Thus, optical fibers are usually collected into a cable by wrapping a plastic cover around them; a cable has at least two fibers, and a cable used between large sites with multiple network devices may contain many fibers.

Although it cannot be bent at a right angle, an optical fiber is flexible enough to form into a circle with diameter less than two inches without breaking. The question arises, why does light travel around a bend in the fiber? The answer comes from physics: when light encounters the boundary between two substances, its behavior depends on the density of the two substances and the angle at which the light strikes the boundary. For a given pair of substances, there exists a *critical angle*, θ, measured with respect to a line that is perpendicular to the boundary. If the angle of incidence is exactly equal to the critical angle, light travels along the boundary. When the angle is less than θ degrees, light crosses the boundary and is *refracted*, and when the angle is greater than θ degrees, light is reflected as if the boundary were a mirror. Figure 7.5 illustrates the concept.

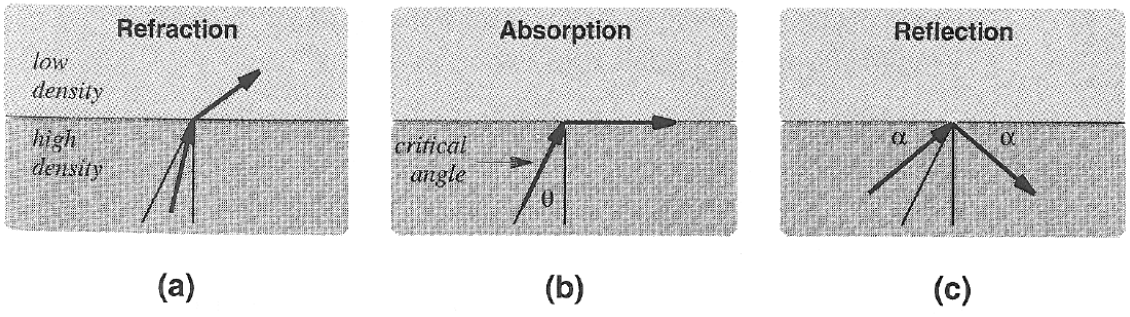


Figure 7.5 Behavior of light at a density boundary when the angle of incidence is (a) less than the critical angle θ, (b) equal to the critical angle, and (c) greater than the critical angle.

Figure 7.5c explains why light stays inside an optical fiber — a substance called *cladding* is bonded to the fiber to form a boundary. As it travels along, light is reflected off the boundary.

Unfortunately, reflection in an optical fiber is not perfect. Reflection absorbs a small amount of energy. Furthermore, if a photon takes a zig-zag path that reflects from the walls of the fiber many times, the photon will travel a slightly longer distance than a photon that takes a straight path. The result is that a pulse of light sent at one end of a fiber emerges with less energy and is *dispersed* (i.e., stretched) over time, as Figure 7.6 illustrates.

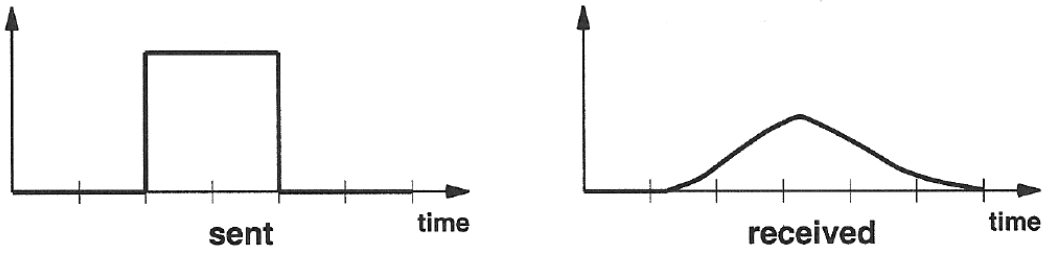


Figure 7.6 A light pulse as sent and received over an optical fiber.

7.9 Types Of Fiber And Light Transmission

Although it is not a problem for optical fibers used to connect a computer to a nearby device, dispersion becomes a serious problem for long optical fibers, such as those used between two cites or under the Atlantic Ocean. Consequently, three forms of optical fibers have been invented that provide a choice between performance and cost:

- *Multimode, Step Index* fiber is the least expensive, and is used when performance is unimportant. The boundary between the fiber and the cladding is abrupt which causes light to reflect frequently. Therefore, dispersion is high.

- *Multimode, Graded Index* fiber is slightly more expensive than the step index fiber. However, it has the advantage of making the density of the fiber ~~de~~crease near the edge, which reduces reflection and lowers dispersion. ↑ gradually

- *Single Mode* fiber is the most expensive, and provides the least dispersion. The fiber has a smaller diameter and other properties that help reduce reflection. Single mode is used for long distances and higher bit rates.

Single mode fiber and the equipment used at each end are designed to focus light. As a result, a pulse of light can travel thousands of kilometers without becoming dispersed. Minimal dispersion helps increase the rate at which bits can be sent because a pulse corresponding to one bit does not disperse into the pulse that corresponds to a successive bit.

How is light sent and received on a fiber? The key is that the devices used for transmission must match the fiber. The available mechanisms include:

- Transmission: Light Emitting Diode (LED) or Injection Laser Diode (ILD)
- Reception: photo-sensitive cell or photodiode

In general, LEDs and photo-sensitive cells are used for short distances and slower bit rates common with multimode fiber; single mode fiber, used over long distance with high bit rates, generally requires ILDs and photodiodes.

7.10 Optical Fiber Compared To Copper Wiring

Optical fiber has several properties that make it more desirable than copper wiring. Optical fiber is immune to electrical noise, has higher bandwidth, and light traveling across a fiber does not attenuate as much as electrical signals traveling across copper. However, copper wiring is less expensive. Furthermore, because the ends of an optical fiber must be polished before they can be used, installation of copper wiring does not require as much special equipment or expertise as optical fiber. Finally, because they are stronger, copper wires are less likely to break if accidentally pulled or bent. Figure 7.7 summarizes the advantages of each media type.

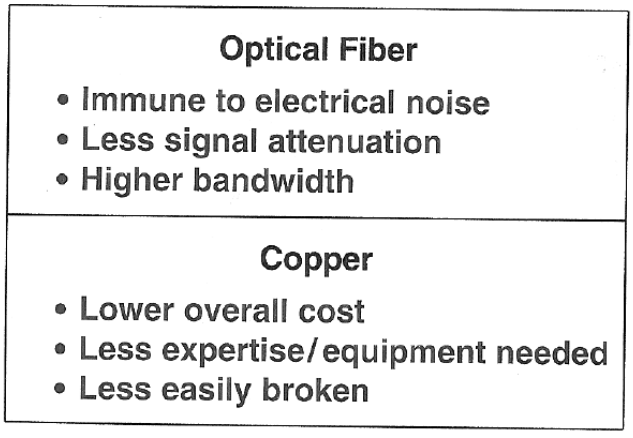


Figure 7.7 Advantages of optical fiber and copper wiring.

2.4 Signal propagation

Like wired networks, wireless communication networks also have senders and receivers of signals. However, in connection with signal propagation, these two networks exhibit considerable differences. In wireless networks, the signal has no wire to determine the direction of propagation, whereas signals in wired networks only travel along the wire (which can be twisted pair copper wires, a coax cable, but also a fiber etc.). As long as the wire is not interrupted or damaged, it typically exhibits the same characteristics at each point. One can precisely determine the behavior of a signal travelling along this wire, e.g., received power depending on the length. For wireless transmission, this predictable behavior is only valid in a vacuum, i.e., without matter between the sender and the receiver. The situation would be as follows (Figure 2.11):

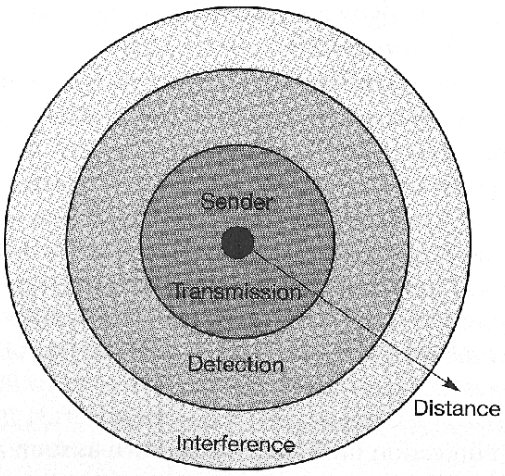


Figure 2.11 Ranges for transmission, detection, and interference of signals

- **Transmission range**: Within a certain radius of the sender transmission is possible, i.e., a receiver receives the signals with an error rate low enough to be able to communicate and can also act as sender.
- **Detection range**: Within a second radius, detection of the transmission is possible, i.e., the transmitted power is large enough to differ from background noise. However, the error rate is too high to establish communication.
- **Interference range**: Within a third even larger radius, the sender may interfere with other transmission by adding to the background noise. A receiver will not be able to detect the signals, but the signals may disturb other signals.

This simple and ideal scheme led to the notion of **cells** around a transmitter (as briefly discussed in section 2.8). However, real life does not happen in a vacuum, radio transmission has to contend with our atmosphere, mountains, buildings, moving senders and receivers etc. In reality, the three circles referred to above will be bizarrely-shaped polygons with their shape being time and frequency dependent. The following paragraphs discuss some problems arising in this context, thereby showing the differences between wireless and wired transmission.

2.4.1 Path loss of radio signals

In free space radio signals propagate as light does (independently of their frequency), i.e., they follow a straight line (besides gravitational effects). If such a straight line exists between a sender and a receiver it is called **line-of-sight (LOS)**. Even if no matter exists between the sender and the receiver (i.e., if there is a vacuum), the signal still experiences the **free space loss**. The received power P_r is proportional to $1/d^2$ with d being the distance between sender and receiver (**inverse square law**). The reason for this phenomenon is quite simple. Think of the sender being a point in space. The sender now emits a signal with certain energy. This signal travels away from the sender at the speed of light as a wave with a spherical shape. If there is no obstacle, the sphere continuously grows with the sending energy equally distributed over the sphere's surface. This surface area s grows with the increasing distance d from the center according to the equation $s = 4\pi d^2$.

Even without any matter between sender and receiver, additional parameters are important. The received power also depends on the wavelength and the gain of receiver and transmitter antennas. As soon as there is any matter between sender and receiver, the situation becomes more complex. Most radio transmission takes place through the atmosphere – signals travel through air, rain, snow, fog, dust particles, smog etc. While the **path loss** or **attenuation** does not cause too much trouble for short distances, e.g., for LANs (see chapter 7), the atmosphere heavily influences transmission over long distances, e.g., satellite transmission (see chapter 5). Even mobile phone systems are influenced by weather conditions such as heavy rain. Rain can absorb much of the radiated energy of the antenna (this effect is used in a microwave oven to cook), so communication links may break down as soon as the rain sets in.

Depending on the frequency, radio waves can also penetrate objects. Generally the lower the frequency, the better the penetration. Long waves can be transmitted through the oceans to a submarine while high frequencies can be blocked by a tree. The higher the frequency, the more the behavior of the radio waves resemble that of light – a phenomenon which is clear if one considers the spectrum shown in Figure 2.1.

Radio waves can exhibit three fundamental propagation behaviors depending on their frequency:

- **Ground wave** (<2 MHz): Waves with low frequencies follow the earth's surface and can propagate long distances. These waves are used for, e.g., submarine communication or AM radio.
- **Sky wave** (2–30 MHz): Many international broadcasts and amateur radio use these short waves that are reflected[2] at the ionosphere. This way the waves can bounce back and forth between the ionosphere and the earth's surface, travelling around the world.
- **Line-of-sight** (>30 MHz): Mobile phone systems, satellite systems, cordless telephones etc. use even higher frequencies. The emitted waves follow a (more or less) straight line of sight. This enables direct communication with satellites (no reflection at the ionosphere) or microwave links on the ground. However, an additional consideration for ground-based communication is that the waves are bent by the atmosphere due to refraction (see next section).

Almost all communication systems presented in this book work with frequencies above 100 MHz so, we are almost exclusively concerned with LOS communication. But why do mobile phones work even without an LOS?

2.4.2 Additional signal propagation effects

As discussed in the previous section, signal propagation in free space almost follows a straight line, like light. But in real life, we rarely have a line-of-sight between the sender and receiver of radio signals. Mobile phones are typically used in big cities with skyscrapers, on mountains, inside buildings, while driving through an alley etc. Hare several effects occur in addition to the attenuation caused by the distance between sender and receiver, which are again very much frequency dependent.

An extreme form of attenuation is **blocking** or **shadowing** of radio signals due to large obstacles (see Figure 2.12, left side). The higher the frequency of a signal, the more it behaves like light. Even small obstacles like a simple wall, a truck on the street, or trees in an alley may block the signal. Another effect is the **reflection** of signals as shown in the middle of Figure 2.12. If an object is large compared to the wavelength of the signal, e.g., huge buildings, mountains,

2 Compared to, e.g., the surface of a building, the ionosphere is not really a hard reflecting surface. In the case of sky waves the 'reflection' is caused by refraction.

or the surface of the earth, the signal is reflected. The reflected signal is not as strong as the original, as objects can absorb some of the signal's power. Reflection helps transmitting signals as soon as no LOS exists. This is the standard case for radio transmission in cities or mountain areas. Signals transmitted from a sender may bounce off the walls of buildings several times before they reach the receiver. The more often the signal is reflected, the weaker it becomes. Finally, the right side of Figure 2.12 shows the effect of **refraction**. This effect occurs because the velocity of the electromagnetic waves depends on the density of the medium through which it travels. Only in vacuum does it equal c. As the figure shows, waves that travel into a denser medium are bent towards the medium. This is the reason for LOS radio waves being bent towards the earth: the density of the atmosphere is higher closer to the ground.

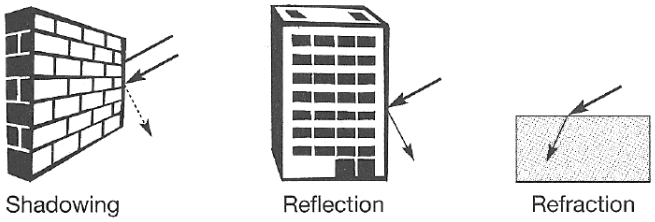


Figure 2.12 Blocking (shadowing), reflection and refraction of waves

While shadowing and reflection are caused by objects much larger than the wavelength of the signals (and demonstrate the typical 'particle' behavior of radio signals), the following two effects exhibit the 'wave' character of radio signals. If the size of an obstacle is in the order of the wavelength or less, then waves can be **scattered** (see Figure 2.13, left side). An incoming signal is scattered into several weaker outgoing signals. In school experiments, this is typically demonstrated with laser light and a very small opening or obstacle, but here we have to take into consideration that the typical wavelength of radio transmission for, e.g., GSM or AMPS is in the order of some 10 cm. Thus, many objects in the environment can cause these scattering effects. Another effect is **diffraction** of waves. As shown on the right side of Figure 2.13, this effect is very similar to scattering. Radio waves will be deflected at an edge and propagate in different directions. The result of scattering and diffraction are patterns with varying signal strengths depending on the location of the receiver.

Effects like attenuation, scattering, diffraction, and refraction all happen simultaneously and are frequency and time dependent. It is very difficult to predict the precise strength of signals at a certain point in space. How do mobile phone operators plan the coverage of their antennas, the location of the antennas, the direction of the beams etc.? Two or three dimensional maps are used with a resolution down to several meters. With the help of, e.g., ray tracing or radiosity techniques similar to rendering 3D graphics, the signal quality can roughly be calculated in advance. Additionally, operators perform a lot of measurements during and after installation of antennas to fill gaps in the coverage.

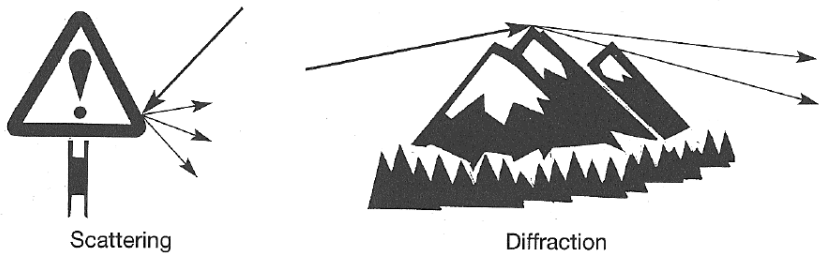


Figure 2.13
Scattering and diffraction of waves

2.4.3 Multi-path propagation

Together with the direct transmission from a sender to a receiver, the propagation effects mentioned in the previous section lead to one of the most severe radio channel impairments, called **multi-path propagation**. Figure 2.14 shows a sender on the left and one possible receiver on the right. Radio waves emitted by the sender can either travel along a straight line, or they may be reflected at a large building, or scattered at smaller obstacles. This simplified figure only shows three possible paths for the signal. In reality, many more paths are possible. Due to the finite speed of light, signals travelling along different paths with different lengths arrive at the receiver at different times. This effect (caused by multi-path propagation) is called **delay spread**: the original signal is spread due to different delays of parts of the signal. This delay spread is a typical effect of radio transmission, because no wire guides the waves along a single path as in the case of wired networks (however, a similar effect, dispersion, is known for high bit-rate optical transmission over multi-mode fiber, see Halsall, 1996, or Stallings, 1997). Notice that this effect has nothing to do with possible movements of the sender or receiver. Typical values for delay spread are approximately 3 μs in cities, up to 12 μs can be observed. GSM, for example, can tolerate up to 16 μs of delay spread, i.e., almost a 5 km path difference.

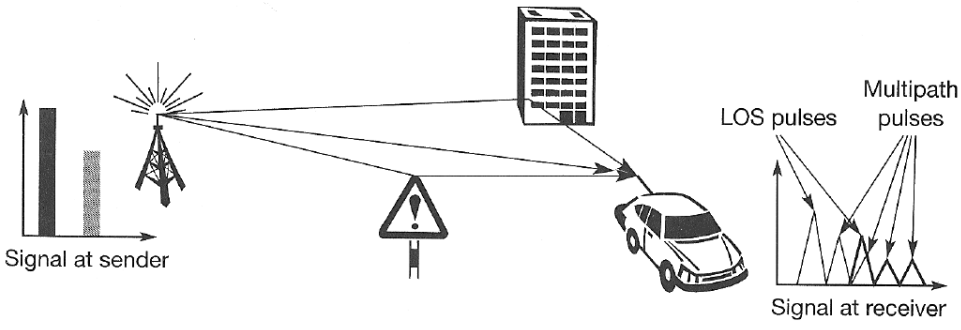


Figure 2.14
Multi-path propagation and intersymbol interference

What are the **effects** of this delay spread on the signals representing the data? The first effect is that a short impulse will be smeared out into a broader impulse, or rather into several weaker impulses. In Figure 2.14 only three possible paths are shown and, thus, the impulse at the sender will result in three smaller impulses at the receiver. For a real situation with hundreds of different paths, this implies that a single impulse will result in many weaker impulses at the receiver. Each path has a different attenuation and, the received pulses have different power. Some of the received pulses will be too weak even to be detected (i.e., they will appear as noise).

Now consider the second impulse shown in Figure 2.14. On the sender side, both impulses are separated. At the receiver, both impulses interfere, i.e., they overlap in time. Now consider that each impulse should represent a symbol, and that one or several symbols could represent a bit. The energy intended for one symbol now spills over to the adjacent symbol, an effect which is called **intersymbol interference (ISI)**. The higher the symbol rate to be transmitted, the worse the effects of ISI will be, as the original symbols are moved closer and closer to each other. ISI limits the bandwidth of a radio channel with multi-path propagation (which is the standard case). Due to this interference, the signals of different symbols can cancel each other out leading to misinterpretations at the receiver and causing transmission errors.

In this case, knowing the channel characteristics can be a great help. If the receiver knows the delays of the different paths (or at least the main paths the signal takes), it can compensate for the distortion caused by the channel. The sender may first transmit a **training sequence** known by the receiver. The receiver then compares the received signal to the original training sequence and programs an **equalizer** that compensates for the distortion (Wesel, 1998), (Pahlavan, 2002), (Stallings, 2002).

While ISI and delay spread already occur in the case of fixed radio transmitters and receivers, the situation is even worse if receivers, or senders, or both, move. Then the channel characteristics change over time, and the paths a signal can travel along vary. This effect is well known (and audible) with analog radios while driving. The power of the received signal changes considerably over time. These quick changes in the received power are also called **short-term fading**. Depending on the different paths the signals take, these signals may have a different phase and cancel each other as shown in Figure 2.15. The receiver now has to try to constantly adapt to the varying channel characteristics, e.g., by changing the parameters of the equalizer. However, if these changes are too fast, such as driving on a highway through a city, the receiver cannot adapt fast enough and the error rate of transmission increases dramatically.

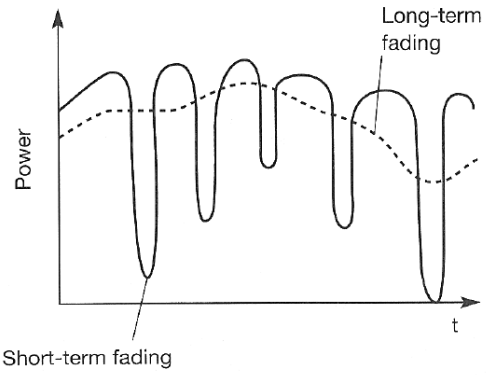


Figure 2.15 Short-term and long-term fading

An additional effect shown in Figure 2.15 is the **long-term fading** of the received signal. This long-term fading, shown here as the average power over time, is caused by, for example, varying distance to the sender or more remote obstacles. Typically, senders can compensate for long-term fading by increasing/decreasing sending power so that the received signal always stays within certain limits.

There are many more effects influencing radio transmission which will not be discussed in detail – for example, the **Doppler shift** caused by a moving sender or receiver. While this effect is audible for acoustic waves already at low speed, it is also a topic for radio transmission from or to fast moving transceivers. One example of such a transceiver could be a satellite (see chapter 5) – there Doppler shift causes random frequency shifts. The interested reader is referred to Anderson (1995), (Pahlavan, 2002), and (Stallings, 2002) for more information about the characteristics of wireless communication channels. For the present it will suffice to know that multi-path propagation limits the maximum bandwidth due to ISI and that moving transceivers cause additional problems due to varying channel characteristics.

5.3 Multiple Access Links and Protocols

In the introduction to this chapter, we noted that there are two types of network links: point-to-point links and broadcast links. A **point-to-point link** consists of a single sender at one end of the link and a single receiver at the other end of the link. Many link-layer protocols have been designed for point-to-point links; the point-to-point protocol (PPP) and high-level data link control (HDLC) are two such protocols that we'll cover later in this chapter. The second type of link, a **broadcast link**, can have multiple sending and receiving nodes all connected to the same, single, shared broadcast channel. The term *broadcast* is used here because when any one node transmits a frame, the channel broadcasts the frame and each of the other nodes receives a copy. Ethernet and wireless LANs are examples of broadcast link-layer technologies. In this section we'll take a step back from specific link-layer protocols and first examine a problem of central importance to the link layer: how to coordinate the access of multiple sending and receiving nodes to a shared broadcast channel—the **multiple access problem**. Broadcast channels are often used in LANs, networks that are geographically concentrated in a single building (or on a corporate or university campus). Thus, we'll also look at how multiple access channels are used in LANs at the end of this section.

We are all familiar with the notion of broadcasting—television has been using it since its invention. But traditional television is a one-way broadcast (that is, one fixed node transmitting to many receiving nodes), while nodes on a computer network broadcast channel can both send and receive. Perhaps a more apt human analogy for a broadcast channel is a cocktail party, where many people gather in a large room (the air providing the broadcast medium) to talk and listen. A second good analogy is something many readers will be familiar with—a classroom—where teacher(s) and student(s) similarly share the same, single, broadcast medium. A central problem in both scenarios is that of determining who gets to talk (that is, transmit into the channel), and when. As humans, we've evolved an elaborate set of protocols for sharing the broadcast channel:

"Give everyone a chance to speak."

"Don't speak until you are spoken to."

"Don't monopolize the conversation."

"Raise your hand if you have a question."

"Don't interrupt when someone is speaking."

"Don't fall asleep when someone is talking."

Computer networks similarly have protocols—so-called **multiple access protocols**—by which nodes regulate their transmission into the shared broadcast channel. As shown in Figure 5.8, multiple access protocols are needed in a wide variety of network settings, including both wired and wireless access networks, and satellite networks. Although technically each node accesses the broadcast channel through its adapter, in this section we will refer to the *node* as the sending and receiving device. In practice, hundreds or even thousands of nodes can directly communicate over a broadcast channel.

Because all nodes are capable of transmitting frames, more than two nodes can transmit frames at the same time. When this happens, all of the nodes receive multiple frames at the same time; that is, the transmitted frames **collide** at all of the receivers. Typically, when there is a collision, none of the receiving nodes can make any sense of any of the frames that were transmitted; in a sense, the signals of the colliding frames become inextricably tangled together. Thus, all the frames involved in the collision are lost, and the broadcast channel is wasted during the collision interval. Clearly, if many nodes want to transmit frames frequently, many transmissions will result in collisions, and much of the bandwidth of the broadcast channel will be wasted.

In order to ensure that the broadcast channel performs useful work when multiple nodes are active, it is necessary to somehow coordinate the transmissions of the active nodes. This coordination job is the responsibility of the multiple access protocol. Over the past 40 years, thousands of papers and hundreds of PhD dissertations have been written on multiple access protocols; a comprehensive survey of the first 20 years of

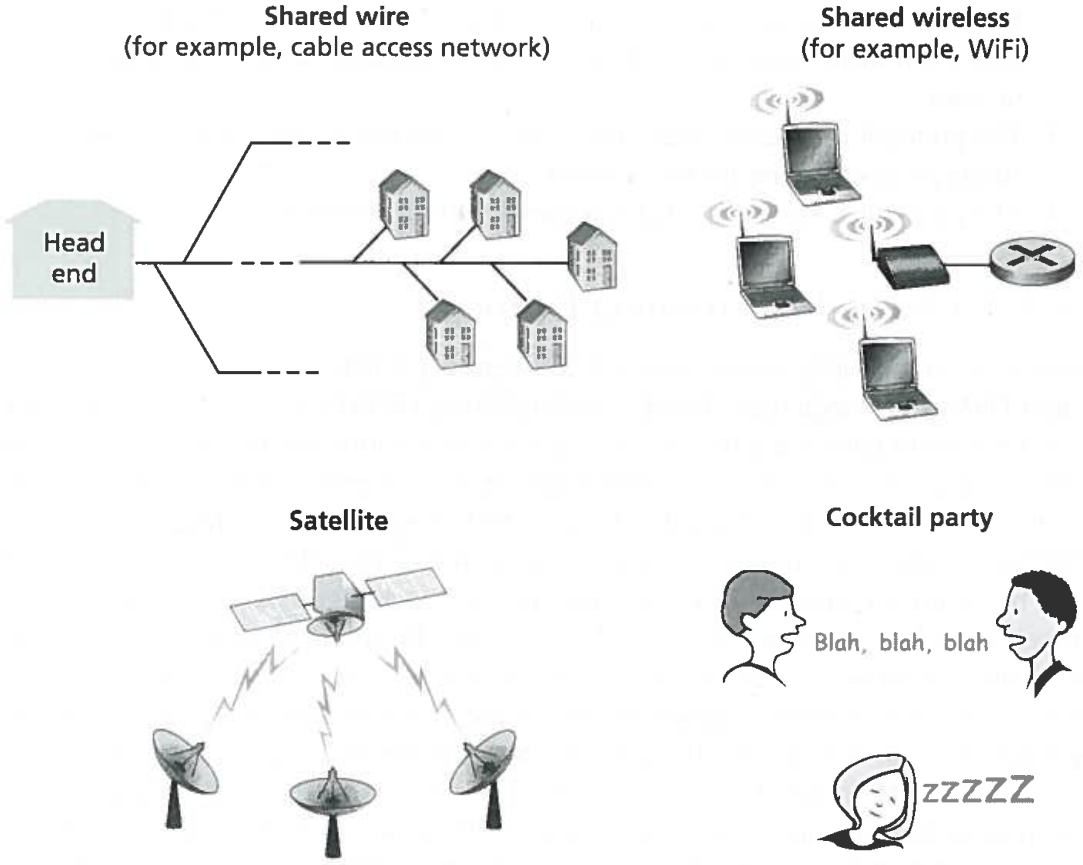


Figure 5.8 ♦ Various multiple access channels

this body of work is [Rom 1990]. Furthermore, active research in multiple access protocols continues due to the continued emergence of new types of links, particularly new wireless links.

Over the years, dozens of multiple access protocols have been implemented in a variety of link-layer technologies. Nevertheless, we can classify just about any multiple access protocol as belonging to one of three categories: **channel partitioning protocols**, **random access protocols**, and **taking-turns protocols**. We'll cover these categories of multiple access protocols in the following three subsections.

Let's conclude this overview by noting that, ideally, a multiple access protocol for a broadcast channel of rate R bits per second should have the following desirable characteristics:

1. When only one node has data to send, that node has a throughput of R bps.
2. When M nodes have data to send, each of these nodes has a throughput of R/M bps. This need not necessarily imply that each of the M nodes always

has an instantaneous rate of R/M, but rather that each node should have an average transmission rate of R/M over some suitably defined interval of time.

3. The protocol is decentralized; that is, there is no master node that represents a single point of failure for the network.
4. The protocol is simple, so that it is inexpensive to implement.

5.3.1 Channel Partitioning Protocols

Recall from our early discussion back in Section 1.3 that time-division multiplexing (TDM) and frequency-division multiplexing (FDM) are two techniques that can be used to partition a broadcast channel's bandwidth among all nodes sharing that channel. As an example, suppose the channel supports N nodes and that the transmission rate of the channel is R bps. TDM divides time into **time frames** and further divides each time frame into N **time slots**. (The TDM time frame should not be confused with the link-layer unit of data exchanged between sending and receiving adapters, which is also called a frame. In order to reduce confusion, in this subsection we'll refer to the link-layer unit of data exchanged as a packet.) Each time slot is then assigned to one of the N nodes. Whenever a node has a packet to send, it transmits the packet's bits during its assigned time slot in the revolving TDM frame. Typically, slot sizes are chosen so that a single packet can be transmitted during a slot time. Figure 5.9 shows a simple four-node TDM example. Returning to our cocktail party analogy, a TDM-regulated cocktail party would allow one partygoer to speak for a fixed period of time, then allow another partygoer to speak for the same amount of time, and so on. Once everyone had had a chance to talk, the pattern would repeat.

TDM is appealing because it eliminates collisions and is perfectly fair: Each node gets a dedicated transmission rate of R/N bps during each frame time. However, it has two major drawbacks. First, a node is limited to an average rate of R/N bps even when it is the only node with packets to send. A second drawback is that a node must always wait for its turn in the transmission sequence—again, even when it is the only node with a frame to send. Imagine the partygoer who is the only one with anything to say (and imagine that this is the even rarer circumstance where everyone wants to hear what that one person has to say). Clearly, TDM would be a poor choice for a multiple access protocol for this particular party.

While TDM shares the broadcast channel in time, FDM divides the R bps channel into different frequencies (each with a bandwidth of R/N) and assigns each frequency to one of the N nodes. FDM thus creates N smaller channels of R/N bps out of the single, larger R bps channel. FDM shares both the advantages and drawbacks of TDM. It avoids collisions and divides the bandwidth fairly among the N nodes. However, FDM also shares a principal disadvantage with TDM—a node is limited to a bandwidth of R/N, even when it is the only node with packets to send.

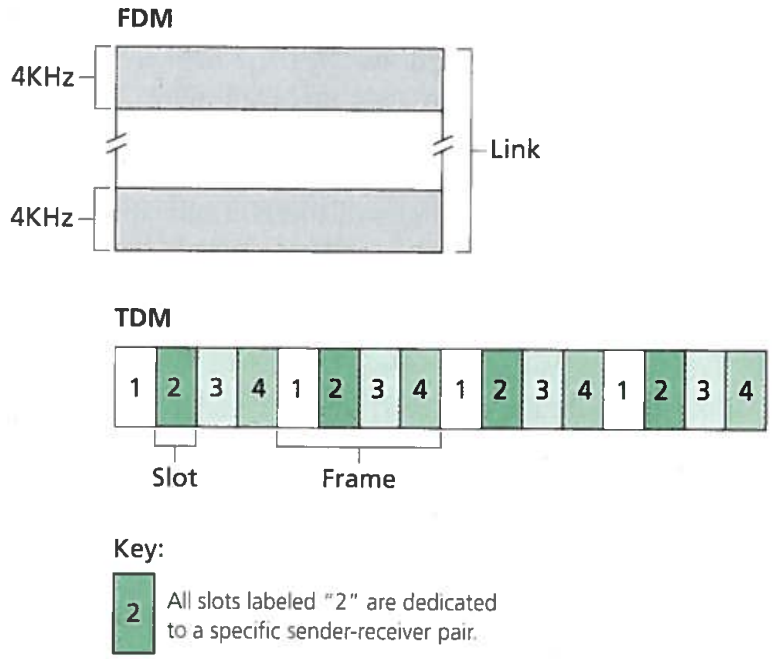


Figure 5.9 ♦ A four-node TDM and FDM example

A third channel partitioning protocol is **code division multiple access (CDMA)**. While TDM and FDM assign time slots and frequencies, respectively, to the nodes, CDMA assigns a different *code* to each node. Each node then uses its unique code to encode the data bits it sends. If the codes are chosen carefully, CDMA networks have the wonderful property that different nodes can transmit *simultaneously* and yet have their respective receivers correctly receive a sender's encoded data bits (assuming the receiver knows the sender's code) in spite of interfering transmissions by other nodes. CDMA has been used in military systems for some time (due to its anti-jamming properties) and now has widespread civilian use, particularly in cellular telephony. Because CDMA's use is so tightly tied to wireless channels, we'll save our discussion of the technical details of CDMA until Chapter 6. For now, it will suffice to know that CDMA codes, like time slots in TDM and frequencies in FDM, can be allocated to the multiple access channel users.

5.3.2 Random Access Protocols

The second broad class of multiple access protocols are random access protocols. In a random access protocol, a transmitting node always transmits at the full rate of the channel, namely, R bps. When there is a collision, each node involved in the collision repeatedly retransmits its frame (that is, packet) until its frame gets

through without a collision. But when a node experiences a collision, it doesn't necessarily retransmit the frame right away. *Instead it waits a random delay before retransmitting the frame*. Each node involved in a collision chooses independent random delays. Because the random delays are independently chosen, it is possible that one of the nodes will pick a delay that is sufficiently less than the delays of the other colliding nodes and will therefore be able to sneak its frame into the channel without a collision.

There are dozens if not hundreds of random access protocols described in the literature [Rom 1990; Bertsekas 1991]. In this section we'll describe a few of the most commonly used random access protocols—the ALOHA protocols [Abramson 1970; Abramson 1985; Abramson 2009] and the carrier sense multiple access (CSMA) protocols [Kleinrock 1975b]. Ethernet [Metcalfe 1976] is a popular and widely deployed CSMA protocol.

Slotted ALOHA

Let's begin our study of random access protocols with one of the simplest random access protocols, the slotted ALOHA protocol. In our description of slotted ALOHA, we assume the following:

- All frames consist of exactly L bits.
- Time is divided into slots of size L/R seconds (that is, a slot equals the time to transmit one frame).
- Nodes start to transmit frames only at the beginnings of slots.
- The nodes are synchronized so that each node knows when the slots begin.
- If two or more frames collide in a slot, then all the nodes detect the collision event before the slot ends.

Let p be a probability, that is, a number between 0 and 1. The operation of slotted ALOHA in each node is simple:

- When the node has a fresh frame to send, it waits until the beginning of the next slot and transmits the entire frame in the slot.
- If there isn't a collision, the node has successfully transmitted its frame and thus need not consider retransmitting the frame. (The node can prepare a new frame for transmission, if it has one.)
- If there is a collision, the node detects the collision before the end of the slot. The node retransmits its frame in each subsequent slot with probability p until the frame is transmitted without a collision.

By retransmitting with probability p, we mean that the node effectively tosses a biased coin; the event heads corresponds to "retransmit," which occurs with

probability p. The event tails corresponds to "skip the slot and toss the coin again in the next slot"; this occurs with probability $(1 - p)$. All nodes involved in the collision toss their coins independently.

Slotted ALOHA would appear to have many advantages. Unlike channel partitioning, slotted ALOHA allows a node to transmit continuously at the full rate, R, when that node is the only active node. (A node is said to be active if it has frames to send.) Slotted ALOHA is also highly decentralized, because each node detects collisions and independently decides when to retransmit. (Slotted ALOHA does, however, require the slots to be synchronized in the nodes; shortly we'll discuss an unslotted version of the ALOHA protocol, as well as CSMA protocols, none of which require such synchronization.) Slotted ALOHA is also an extremely simple protocol.

Slotted ALOHA works well when there is only one active node, but how efficient is it when there are multiple active nodes? There are two possible efficiency concerns here. First, as shown in Figure 5.10, when there are multiple active nodes, a certain fraction of the slots will have collisions and will therefore be "wasted." The second concern is that another fraction of the slots will be *empty* because all active nodes refrain from transmitting as a result of the probabilistic transmission policy. The only "unwasted" slots will be those in which exactly one node transmits. A slot in which exactly one node transmits is said to be a **successful slot**. The **efficiency** of a slotted multiple access protocol is defined to be the long-run fraction of successful slots in the case when there are a large number of active nodes, each always having a large number of frames to send.

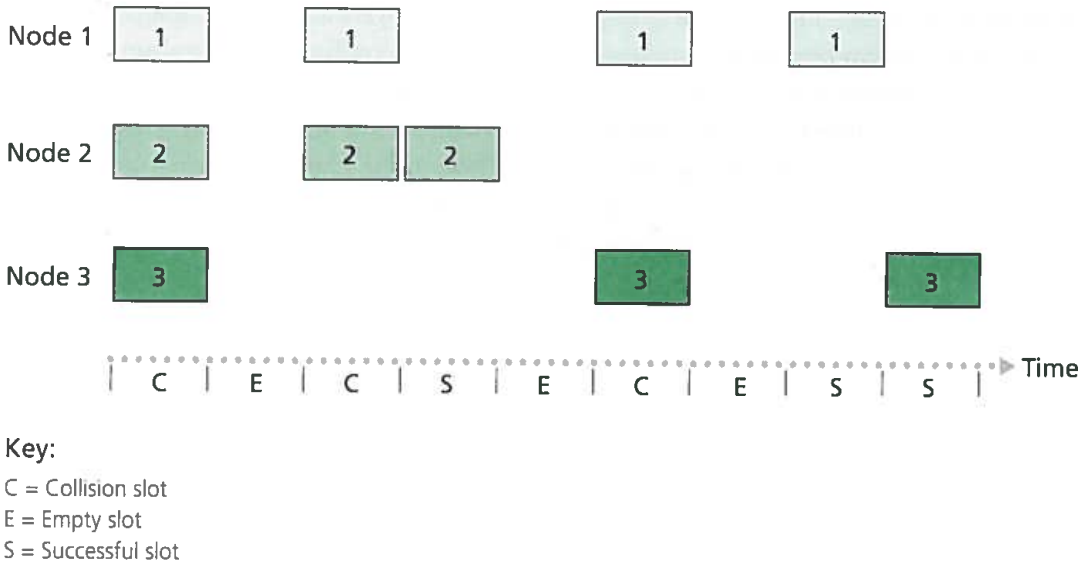


Figure 5.10 ♦ Nodes 1, 2, and 3 collide in the first slot. Node 2 finally succeeds in the fourth slot, node 1 in the eighth slot, and node 3 in the ninth slot

Note that if no form of access control were used, and each node were to immediately retransmit after each collision, the efficiency would be zero. Slotted ALOHA clearly increases the efficiency beyond zero, but by how much?

We now proceed to outline the derivation of the maximum efficiency of slotted ALOHA. To keep this derivation simple, let's modify the protocol a little and assume that each node attempts to transmit a frame in each slot with probability p. (That is, we assume that each node always has a frame to send and that the node transmits with probability p for a fresh frame as well as for a frame that has already suffered a collision.) Suppose there are N nodes. Then the probability that a given slot is a successful slot is the probability that one of the nodes transmits and that the remaining $N - 1$ nodes do not transmit. The probability that a given node transmits is p; the probability that the remaining nodes do not transmit is $(1 - p)^{N-1}$. Therefore the probability a given node has a success is $p(1 - p)^{N-1}$. Because there are N nodes, the probability that any one of the N nodes has a success is $Np(1 - p)^{N-1}$.

Thus, when there are N active nodes, the efficiency of slotted ALOHA is $Np(1 - p)^{N-1}$. To obtain the *maximum* efficiency for N active nodes, we have to find the p^* that maximizes this expression. (See the homework problems for a general outline of this derivation.) And to obtain the maximum efficiency for a large number of active nodes, we take the limit of $Np^*(1 - p^*)^{N-1}$ as N approaches infinity. (Again, see the homework problems.) After performing these calculations, we'll find that the maximum efficiency of the protocol is given by $1/e = 0.37$. That is, when a large number of nodes have many frames to transmit, then (at best) only 37 percent of the slots do useful work. Thus the effective transmission rate of the channel is not R bps but only 0.37 R bps! A similar analysis also shows that 37 percent of the slots go empty and 26 percent of slots have collisions. Imagine the poor network administrator who has purchased a 100-Mbps slotted ALOHA system, expecting to be able to use the network to transmit data among a large number of users at an aggregate rate of, say, 80 Mbps! Although the channel is capable of transmitting a given frame at the full channel rate of 100 Mbps, in the long run, the successful throughput of this channel will be less than 37 Mbps.

Aloha

The slotted ALOHA protocol required that all nodes synchronize their transmissions to start at the beginning of a slot. The first ALOHA protocol [Abramson 1970] was actually an unslotted, fully decentralized protocol. In pure ALOHA, when a frame first arrives (that is, a network-layer datagram is passed down from the network layer at the sending node), the node immediately transmits the frame in its entirety into the broadcast channel. If a transmitted frame experiences a collision with one or more other transmissions, the node will then immediately (after completely transmitting its collided frame) retransmit the frame with probability p. Otherwise, the node waits for a frame transmission time. After this wait, it then

transmits the frame with probability p, or waits (remaining idle) for another frame time with probability $1 - p$.

To determine the maximum efficiency of pure ALOHA, we focus on an individual node. We'll make the same assumptions as in our slotted ALOHA analysis and take the frame transmission time to be the unit of time. At any given time, the probability that a node is transmitting a frame is p. Suppose this frame begins transmission at time t_0. As shown in Figure 5.11, in order for this frame to be successfully transmitted, no other nodes can begin their transmission in the interval of time $[t_0 - 1, t_0]$. Such a transmission would overlap with the beginning of the transmission of node i's frame. The probability that all other nodes do not begin a transmission in this interval is $(1 - p)^{N-1}$. Similarly, no other node can begin a transmission while node i is transmitting, as such a transmission would overlap with the latter part of node i's transmission. The probability that all other nodes do not begin a transmission in this interval is also $(1 - p)^{N-1}$. Thus, the probability that a given node has a successful transmission is $p(1 - p)^{2(N-1)}$. By taking limits as in the slotted ALOHA case, we find that the maximum efficiency of the pure ALOHA protocol is only $1/(2e)$—exactly half that of slotted ALOHA. This then is the price to be paid for a fully decentralized ALOHA protocol.

Carrier Sense Multiple Access (CSMA)

In both slotted and pure ALOHA, a node's decision to transmit is made independently of the activity of the other nodes attached to the broadcast channel. In particular, a node neither pays attention to whether another node happens to be transmitting when it begins to transmit, nor stops transmitting if another node begins to interfere with its transmission. In our cocktail party analogy, ALOHA protocols are quite like

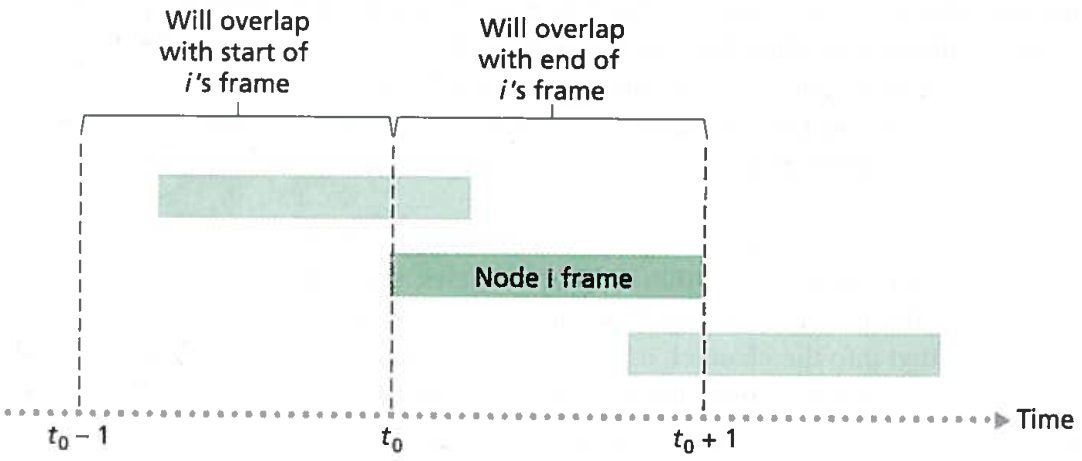


Figure 5.11 ♦ Interfering transmissions in pure ALOHA

CASE HISTORY

NORM ABRAMSON AND ALOHANET

Norm Abramson, a PhD engineer, had a passion for surfing and an interest in packet switching. This combination of interests brought him to the University of Hawaii in 1969. Hawaii consists of many mountainous islands, making it difficult to install and operate land-based networks. When not surfing, Abramson thought about how to design a network that does packet switching over radio. The network he designed had one central host and several secondary nodes scattered over the Hawaiian Islands. The network had two channels, each using a different frequency band. The downlink channel broadcasted packets from the central host to the secondary hosts; and the upstream channel sent packets from the secondary hosts to the central host. In addition to sending informational packets, the central host also sent on the downstream channel an acknowledgment for each packet successfully received from the secondary hosts.

Because the secondary hosts transmitted packets in a decentralized fashion, collisions on the upstream channel inevitably occurred. This observation led Abramson to devise the pure ALOHA protocol, as described in this chapter. In 1970, with continued funding from ARPA, Abramson connected his ALOHAnet to the ARPAnet. Abramson's work is important not only because it was the first example of a radio packet network, but also because it inspired Bob Metcalfe. A few years later, Metcalfe modified the ALOHA protocol to create the CSMA/CD protocol and the Ethernet LAN.

a boorish partygoer who continues to chatter away regardless of whether other people are talking. As humans, we have human protocols that allow us not only to behave with more civility, but also to decrease the amount of time spent "colliding" with each other in conversation and, consequently, to increase the amount of data we exchange in our conversations. Specifically, there are two important rules for polite human conversation:

- *Listen before speaking.* If someone else is speaking, wait until they are finished. In the networking world, this is called **carrier sensing**—a node listens to the channel before transmitting. If a frame from another node is currently being transmitted into the channel, a node then waits until it detects no transmissions for a short amount of time and then begins transmission.
- *If someone else begins talking at the same time, stop talking.* In the networking world, this is called **collision detection**—a transmitting node listens to the channel while it is transmitting. If it detects that another node is transmitting an interfering

frame, it stops transmitting and waits a random amount of time before repeating the sense-and-transmit-when-idle cycle.

These two rules are embodied in the family of **carrier sense multiple access (CSMA)** and **CSMA with collision detection (CSMA/CD)** protocols [Kleinrock 1975b; Metcalfe 1976; Lam 1980; Rom 1990]. Many variations on CSMA and CSMA/CD have been proposed. Here, we'll consider a few of the most important, and fundamental, characteristics of CSMA and CSMA/CD.

The first question that you might ask about CSMA is why, if all nodes perform carrier sensing, do collisions occur in the first place? After all, a node will refrain from transmitting whenever it senses that another node is transmitting. The answer to the question can best be illustrated using space-time diagrams [Molle 1987]. Figure 5.12 shows a space-time diagram of four nodes (A, B, C, D) attached to a linear broadcast bus. The horizontal axis shows the position of each node in space; the vertical axis represents time.

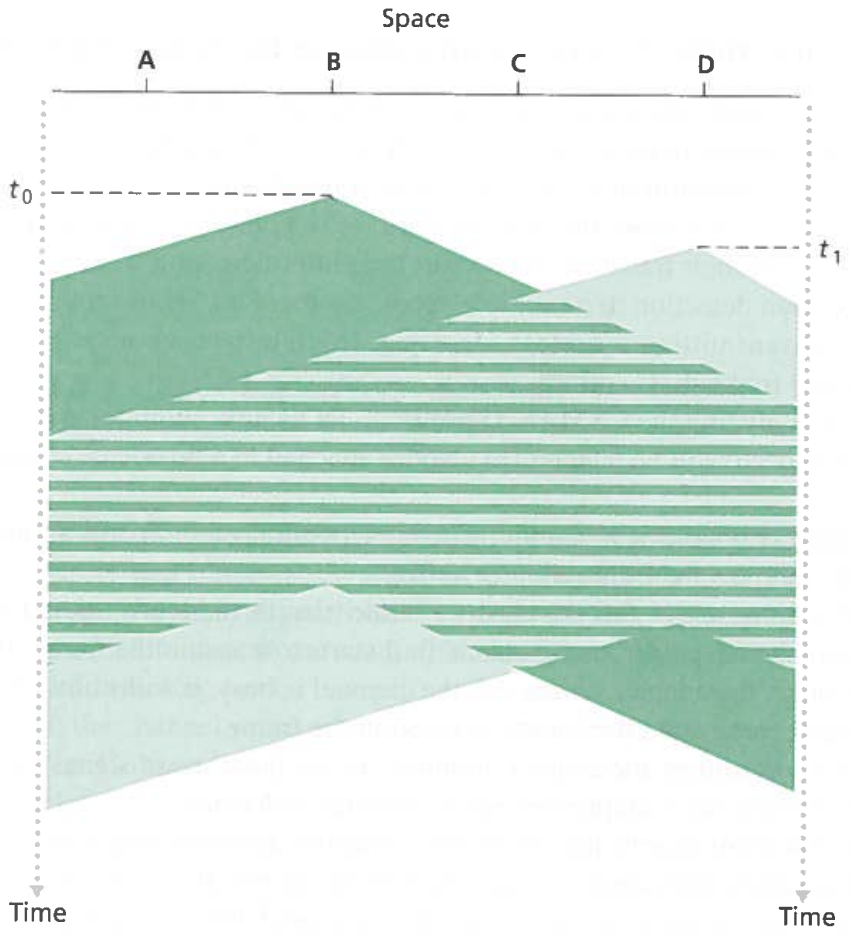


Figure 5.12 ♦ Space-time diagram of two CSMA nodes with colliding transmissions

At time t_0, node B senses the channel is idle, as no other nodes are currently transmitting. Node B thus begins transmitting, with its bits propagating in both directions along the broadcast medium. The downward propagation of B's bits in Figure 5.12 with increasing time indicates that a nonzero amount of time is needed for B's bits actually to propagate (albeit at near the speed of light) along the broadcast medium. At time t_1 ($t_1 > t_0$), node D has a frame to send. Although node B is currently transmitting at time t_1, the bits being transmitted by B have yet to reach D, and thus D senses the channel idle at t_1. In accordance with the CSMA protocol, D thus begins transmitting its frame. A short time later, B's transmission begins to interfere with D's transmission at D. From Figure 5.12, it is evident that the end-to-end **channel propagation delay** of a broadcast channel—the time it takes for a signal to propagate from one of the nodes to another—will play a crucial role in determining its performance. The longer this propagation delay, the larger the chance that a carrier-sensing node is not yet able to sense a transmission that has already begun at another node in the network.

Carrier Sense Multiple Access with Collision Dection (CSMA/CD)

In Figure 5.12, nodes do not perform collision detection; both B and D continue to transmit their frames in their entirety even though a collision has occurred. When a node performs collision detection, it ceases transmission as soon as it detects a collision. Figure 5.13 shows the same scenario as in Figure 5.12, except that the two nodes each abort their transmission a short time after detecting a collision. Clearly, adding collision detection to a multiple access protocol will help protocol performance by not transmitting a useless, damaged (by interference with a frame from another node) frame in its entirety.

Before analyzing the CSMA/CD protocol, let us now summarize its operation from the perspective of an adapter (in a node) attached to a broadcast channel:

1. The adapter obtains a datagram from the network layer, prepares a link-layer frame, and puts the frame adapter buffer.
2. If the adapter senses that the channel is idle (that is, there is no signal energy entering the adapter from the channel), it starts to transmit the frame. If, on the other hand, the adapter senses that the channel is busy, it waits until it senses no signal energy and then starts to transmit the frame.
3. While transmitting, the adapter monitors for the presence of signal energy coming from other adapters using the broadcast channel.
4. If the adapter transmits the entire frame without detecting signal energy from other adapters, the adapter is finished with the frame. If, on the other hand, the adapter detects signal energy from other adapters while transmitting, it aborts the transmission (that is, it stops transmitting its frame).
5. After aborting, the adapter waits a random amount of time and then returns to step 2.

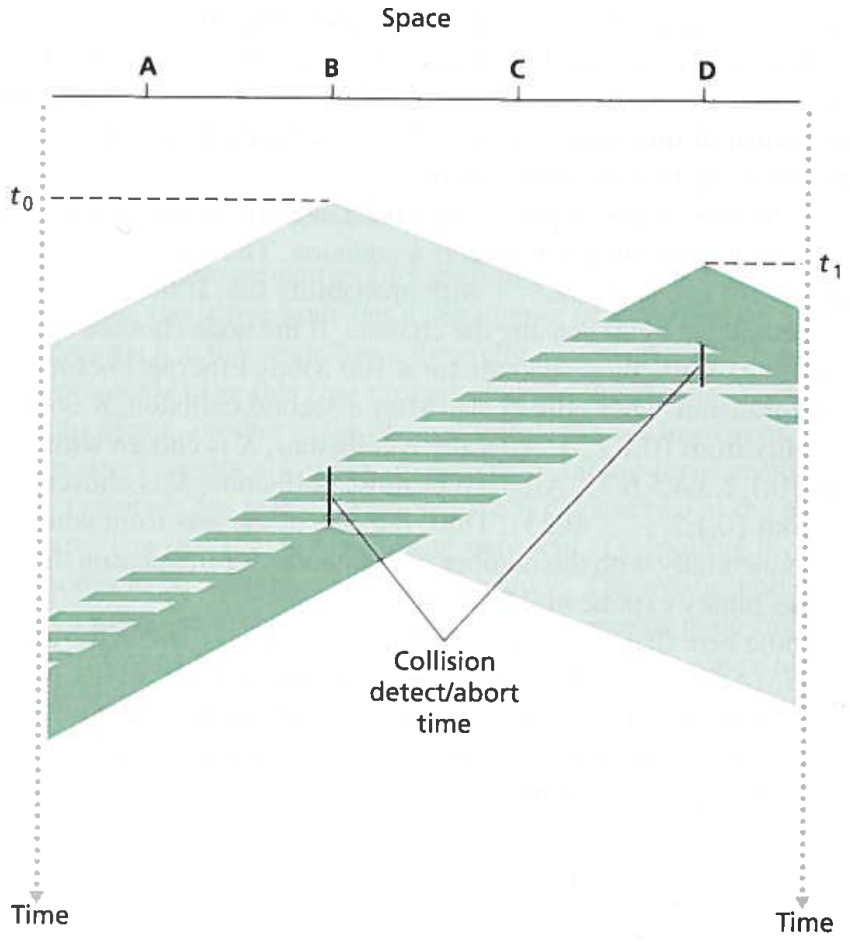


Figure 5.13 ♦ CSMA with collision detection

The need to wait a random (rather than fixed) amount of time is hopefully clear—if two nodes transmitted frames at the same time and then both waited the same fixed amount of time, they'd continue colliding forever. But what is a good interval of time from which to choose the random backoff time? If the interval is large and the number of colliding nodes is small, nodes are likely to wait a large amount of time (with the channel remaining idle) before repeating the sense-and-transmit-when-idle step. On the other hand, if the interval is small and the number of colliding nodes is large, it's likely that the chosen random values will be nearly the same, and transmitting nodes will again collide. What we'd like is an interval that is short when the number of colliding nodes is small, and long when the number of colliding nodes is large.

The **binary exponential backoff** algorithm, used in Ethernet as well as in DOCSIS cable network multiple access protocols [DOCSIS 2011], elegantly solves this problem. Specifically, when transmitting a frame that has already experienced

n collisions, a node chooses the value of K at random from $\{0, 1, 2, \ldots . 2^n-1\}$. Thus, the more collisions experienced by a frame, the larger the interval from which K is chosen. For Ethernet, the actual amount of time a node waits is $K \cdot 512$ bit times (i.e., K times the amount of time needed to send 512 bits into the Ethernet) and the maximum value that n can take is capped at 10.

Let's look at an example. Suppose that a node attempts to transmit a frame for the first time and while transmitting it detects a collision. The node then chooses $K = 0$ with probability 0.5 or chooses $K = 1$ with probability 0.5. If the node chooses $K = 0$, then it immediately begins sensing the channel. If the node chooses $K = 1$, it waits 512 bit times (e.g., 0.01 microseconds for a 100 Mbps Ethernet) before beginning the sense-and-transmit-when-idle cycle. After a second collision, K is chosen with equal probability from $\{0,1,2,3\}$. After three collisions, K is chosen with equal probability from $\{0,1,2,3,4,5,6,7\}$. After 10 or more collisions, K is chosen with equal probability from $\{0,1,2, \ldots , 1023\}$. Thus, the size of the sets from which K is chosen grows exponentially with the number of collisions; for this reason this algorithm is referred to as binary exponential backoff.

We also note here that each time a node prepares a new frame for transmission, it runs the CSMA/CD algorithm, not taking into account any collisions that may have occurred in the recent past. So it is possible that a node with a new frame will immediately be able to sneak in a successful transmission while several other nodes are in the exponential backoff state.

CSMA/CD Efficiency

When only one node has a frame to send, the node can transmit at the full channel rate (e.g., for Ethernet typical rates are 10 Mbps, 100 Mbps, or 1 Gbps). However, if many nodes have frames to transmit, the effective transmission rate of the channel can be much less. We define the **efficiency of CSMA/CD** to be the long-run fraction of time during which frames are being transmitted on the channel without collisions when there is a large number of active nodes, with each node having a large number of frames to send. In order to present a closed-form approximation of the efficiency of Ethernet, let d_{prop} denote the maximum time it takes signal energy to propagate between any two adapters. Let d_{trans} be the time to transmit a maximum-size frame (approximately 1.2 msecs for a 10 Mbps Ethernet). A derivation of the efficiency of CSMA/CD is beyond the scope of this book (see [Lam 1980] and [Bertsekas 1991]). Here we simply state the following approximation:

$$\text{Efficiency} = \frac{1}{1 + 5d_{\text{prop}}/d_{\text{trans}}}$$

We see from this formula that as d_{prop} approaches 0, the efficiency approaches 1. This matches our intuition that if the propagation delay is zero, colliding nodes will abort

immediately without wasting the channel. Also, as d_{trans} becomes very large, efficiency approaches 1. This is also intuitive because when a frame grabs the channel, it will hold on to the channel for a very long time; thus, the channel will be doing productive work most of the time.

5.3.3 Taking-Turns Protocols

Recall that two desirable properties of a multiple access protocol are (1) when only one node is active, the active node has a throughput of R bps, and (2) when M nodes are active, then each active node has a throughput of nearly R/M bps. The ALOHA and CSMA protocols have this first property but not the second. This has motivated researchers to create another class of protocols—the **taking-turns protocols**. As with random access protocols, there are dozens of taking-turns protocols, and each one of these protocols has many variations. We'll discuss two of the more important protocols here. The first one is the **polling protocol**. The polling protocol requires one of the nodes to be designated as a master node. The master node **polls** each of the nodes in a round-robin fashion. In particular, the master node first sends a message to node 1, saying that it (node 1) can transmit up to some maximum number of frames. After node 1 transmits some frames, the master node tells node 2 it (node 2) can transmit up to the maximum number of frames. (The master node can determine when a node has finished sending its frames by observing the lack of a signal on the channel.) The procedure continues in this manner, with the master node polling each of the nodes in a cyclic manner.

The polling protocol eliminates the collisions and empty slots that plague random access protocols. This allows polling to achieve a much higher efficiency. But it also has a few drawbacks. The first drawback is that the protocol introduces a polling delay—the amount of time required to notify a node that it can transmit. If, for example, only one node is active, then the node will transmit at a rate less than R bps, as the master node must poll each of the inactive nodes in turn each time the active node has sent its maximum number of frames. The second drawback, which is potentially more serious, is that if the master node fails, the entire channel becomes inoperative. The 802.15 protocol and the Bluetooth protocol we will study in Section 6.3 are examples of polling protocols.

The second taking-turns protocol is the **token-passing protocol**. In this protocol there is no master node. A small, special-purpose frame known as a **token** is exchanged among the nodes in some fixed order. For example, node 1 might always send the token to node 2, node 2 might always send the token to node 3, and node N might always send the token to node 1. When a node receives a token, it holds onto the token only if it has some frames to transmit; otherwise, it immediately forwards the token to the next node. If a node does have frames to transmit when it receives the token, it sends up to a maximum number of frames and then forwards the token to the next node. Token passing is decentralized and highly efficient. But it has its problems as well. For example, the failure of one node can crash the entire channel. Or if a node accidentally neglects to

release the token, then some recovery procedure must be invoked to get the token back in circulation. Over the years many token-passing protocols have been developed, including the fiber distributed data interface (FDDI) protocol [Jain 1994] and the IEEE 802.5 token ring protocol [IEEE 802.5 2012], and each one had to address these as well as other sticky issues.

5.3.4 DOCSIS: The Link-Layer Protocol for Cable Internet Access

In the previous three subsections, we've learned about three broad classes of multiple access protocols: channel partitioning protocols, random access protocols, and taking turns protocols. A cable access network will make for an excellent case study here, as we'll find aspects of *each* of these three classes of multiple access protocols with the cable access network!

Recall from Section 1.2.1, that a cable access network typically connects several thousand residential cable modems to a cable modem termination system (CMTS) at the cable network headend. The Data-Over-Cable Service Interface Specifications (DOCSIS) [DOCSIS 2011] specifies the cable data network architecture and its protocols. DOCSIS uses FDM to divide the downstream (CMTS to modem) and upstream (modem to CMTS) network segments into multiple frequency channels. Each downstream channel is 6 MHz wide, with a maximum throughput of approximately 40 Mbps per channel (although this data rate is seldom seen at a cable modem in practice); each upstream channel has a maximum channel width of 6.4 MHz, and a maximum upstream throughput of approximately 30 Mbps. Each upstream and downstream channel is a broadcast channel. Frames transmitted on the downstream channel by the CMTS are received by all cable modems receiving that channel; since there is just a single CMTS transmitting into the downstream channel, however, there is no multiple access problem. The upstream direction, however, is more interesting and technically challenging, since multiple cable modems share the same upstream channel (frequency) to the CMTS, and thus collisions can potentially occur.

As illustrated in Figure 5.14, each upstream channel is divided into intervals of time (TDM-like), each containing a sequence of mini-slots during which cable modems can transmit to the CMTS. The CMTS explicitly grants permission to individual cable modems to transmit during specific mini-slots. The CMTS accomplishes this by sending a control message known as a MAP message on a downstream channel to specify which cable modem (with data to send) can transmit during which mini-slot for the interval of time specified in the control message. Since mini-slots are explicitly allocated to cable modems, the CMTS can ensure there are no colliding transmissions during a mini-slot.

But how does the CMTS know which cable modems have data to send in the first place? This is accomplished by having cable modems send mini-slot-request frames to the CMTS during a special set of interval mini-slots that are dedicated

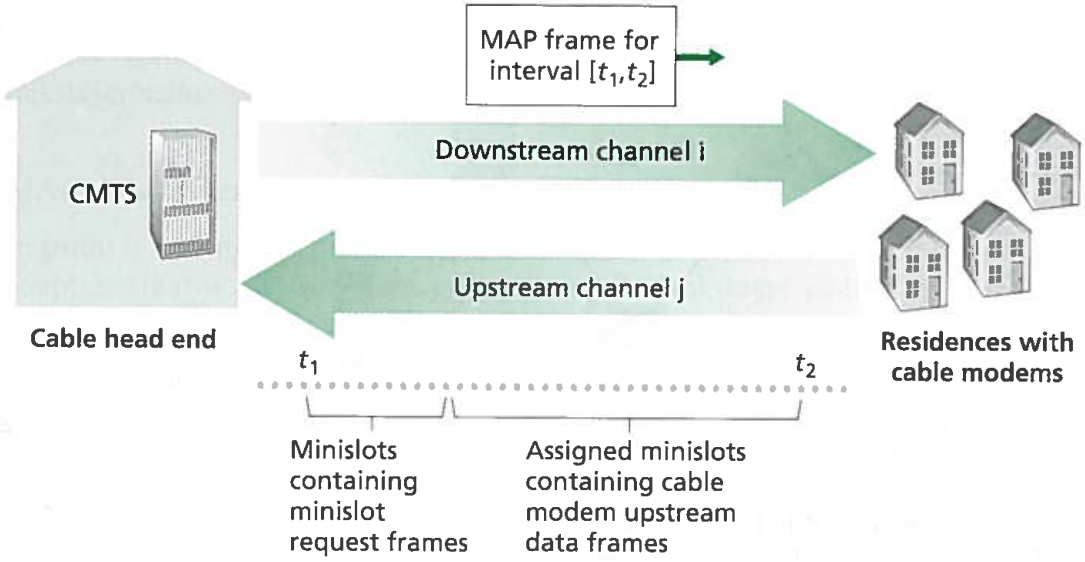


Figure 5.14 ♦ Upstream and downstream channels between CMTS and cable modems

for this purpose, as shown in Figure 5.14. These mini-slot-request frames are transmitted in a random access manner and so may collide with each other. A cable modem can neither sense whether the upstream channel is busy nor detect collisions. Instead, the cable modem infers that its mini-slot-request frame experienced a collision if it does not receive a response to the requested allocation in the next downstream control message. When a collision is inferred, a cable modem uses binary exponential backoff to defer the retransmission of its mini-slot -request frame to a future time slot. When there is little traffic on the upstream channel, a cable modem may actually transmit data frames during slots nominally assigned for mini-slot-request frames (and thus avoid having to wait for a mini-slot assignment).

A cable access network thus serves as a terrific example of multiple access protocols in action—FDM, TDM, random access, and centrally allocated time slots all within one network!

Obtaining a Host Address: The Dynamic Host Configuration Protocol

Once an organization has obtained a block of addresses, it can assign individual IP addresses to the host and router interfaces in its organization. A system administrator will typically manually configure the IP addresses into the router (often remotely, with a network management tool). Host addresses can also be configured manually, but typically this is done using the **Dynamic Host Configuration Protocol (DHCP)** [RFC 2131]. DHCP allows a host to obtain (be allocated) an IP address automatically. A network administrator can configure DHCP so that a given host receives the same IP address each time it connects to the network, or a host may be assigned a **temporary IP address** that will be different each time the host connects to the network. In addition to host IP address assignment, DHCP also allows a host to learn additional information, such as its subnet mask, the address of its first-hop router (often called the default gateway), and the address of its local DNS server.

Because of DHCP's ability to automate the network-related aspects of connecting a host into a network, it is often referred to as a **plug-and-play** or **zeroconf** (zero-configuration) protocol. This capability makes it *very* attractive to the network administrator who would otherwise have to perform these tasks manually! DHCP is also enjoying widespread use in residential Internet access networks, enterprise networks, and in wireless LANs, where hosts join and leave the network frequently. Consider, for example, the student who carries a laptop from a dormitory room to a library to a classroom. It is likely that in each location, the student will be connecting into a new subnet and hence will need a new IP address at each location. DHCP is ideally suited to this situation, as there are many users coming and going, and addresses are needed for only a limited amount of time. The value of DHCP's plug-and-play capability is clear, since it's unimaginable that a system administrator would be able to reconfigure laptops at each location, and few students (except those taking a computer networking class!) would have the expertise to configure their laptops manually.

DHCP is a client-server protocol. A client is typically a newly arriving host wanting to obtain network configuration information, including an IP address for itself. In the simplest case, each subnet (in the addressing sense of Figure 4.20) will have a DHCP server. If no server is present on the subnet, a DHCP relay agent (typically a router) that knows the address of a DHCP server for that network is needed. Figure 4.23 shows a DHCP server attached to subnet 223.1.2/24, with the router serving as the relay agent for arriving clients attached to subnets 223.1.1/24 and 223.1.3/24. In our discussion below, we'll assume that a DHCP server is available on the subnet.

For a newly arriving host, the DHCP protocol is a four-step process, as shown in Figure 4.24 for the network setting shown in Figure 4.23. In this figure, `yiaddr` (as in "your Internet address") indicates the address being allocated to the newly arriving client. The four steps are:

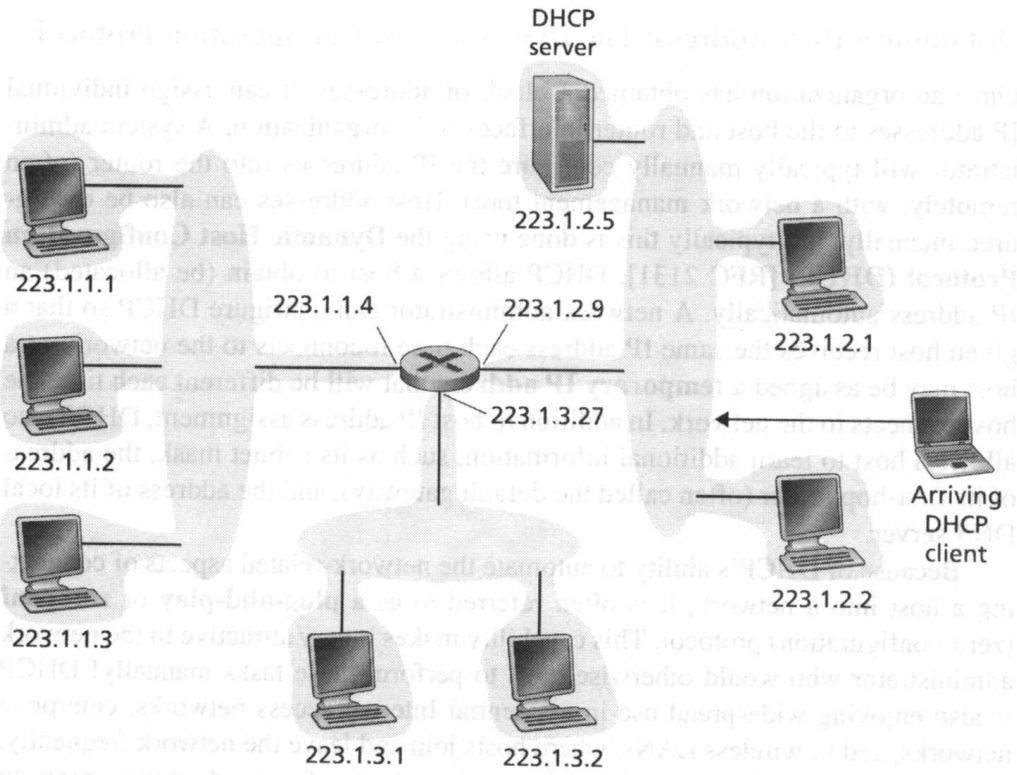


Figure 4.23 ♦ DHCP client and server

- *DHCP server discovery.* The first task of a newly arriving host is to find a DHCP server with which to interact. This is done using a **DHCP discover message**, which a client sends within a UDP packet to port 67. The UDP packet is encapsulated in an IP datagram. But to whom should this datagram be sent? The host doesn't even know the IP address of the network to which it is attaching, much less the address of a DHCP server for this network. Given this, the DHCP client creates an IP datagram containing its DHCP discover message along with the broadcast destination IP address of 255.255.255.255 and a "this host" source IP address of 0.0.0.0. The DHCP client passes the IP datagram to the link layer, which then broadcasts this frame to all nodes attached to the subnet (we will cover the details of link-layer broadcasting in Section 6.4).
- *DHCP server offer(s).* A DHCP server receiving a DHCP discover message responds to the client with a **DHCP offer message** that is broadcast to all nodes on the subnet, again using the IP broadcast address of 255.255.255.255. (You might want to think about why this server reply must also be broadcast). Since several DHCP servers can be present on the subnet, the client may find itself in the enviable position of being able to choose from among several offers. Each

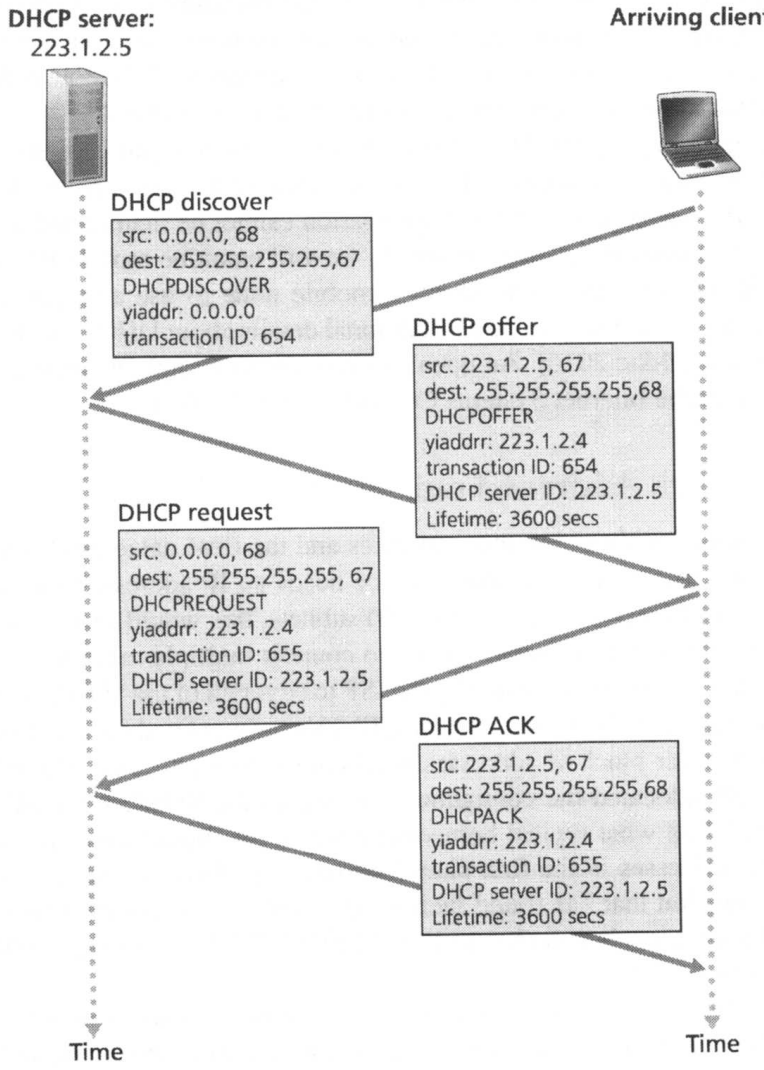


Figure 4.24 ♦ DHCP client-server interaction

server offer message contains the transaction ID of the received discover message, the proposed IP address for the client, the network mask, and an IP **address lease time**—the amount of time for which the IP address will be valid. It is common for the server to set the lease time to several hours or days [Droms 2002].

- *DHCP request.* The newly arriving client will choose from among one or more server offers and respond to its selected offer with a **DHCP request message**, echoing back the configuration parameters.
- *DHCP ACK.* The server responds to the DHCP request message with a **DHCP ACK message**, confirming the requested parameters.

Once the client receives the DHCP ACK, the interaction is complete and the client can use the DHCP-allocated IP address for the lease duration. Since a client may want to use its address beyond the lease's expiration, DHCP also provides a mechanism that allows a client to renew its lease on an IP address.

From a mobility aspect, DHCP does have one very significant shortcoming. Since a new IP address is obtained from DHCP each time a node connects to a new subnet, a TCP connection to a remote application cannot be maintained as a mobile node moves between subnets. In Chapter 6, we will examine mobile IP—an extension to the IP infrastructure that allows a mobile node to use a single permanent address as it moves between subnets. Additional details about DHCP can be found in [Droms 2002] and [dhc 2016]. An open source reference implementation of DHCP is available from the Internet Systems Consortium [ISC 2016].

Network Address Translation (NAT)

Given our discussion about Internet addresses and the IPv4 datagram format, we're now well aware that every IP-capable device needs an IP address. With the proliferation of small office, home office (SOHO) subnets, this would seem to imply that whenever a SOHO wants to install a LAN to connect multiple machines, a range of addresses would need to be allocated by the ISP to cover all of the SOHO's machines. If the subnet grew bigger (for example, the kids at home have not only their own computers, but have bought handheld PDAs, IP-capable phones, and networked Game Boys as well), a larger block of addresses would have to be allocated. But what if the ISP had already allocated the contiguous portions of the SOHO network's current address range? And what typical homeowner wants (or should need) to know how to manage IP addresses in the first place? Fortunately, there is a simpler approach to address allocation that has found increasingly widespread use in such scenarios: **network address translation** (**NAT**) [RFC 2663; RFC 3022].

Figure 4.22 shows the operation of a NAT-enabled router. The NAT-enabled router, residing in the home, has an interface that is part of the home network on the right of Figure 4.22. Addressing within the home network is exactly as we have seen above—all four interfaces in the home network have the same subnet address of 10.0.0/24. The address space 10.0.0.0/8 is one of three portions of the IP address space that is reserved in [RFC 1918] for a private network or a **realm** with private addresses, such as the home network in Figure 4.22. A *realm with private addresses* refers to a network whose addresses only have meaning to devices within that network. To see why this is important, consider the fact that there are hundreds of

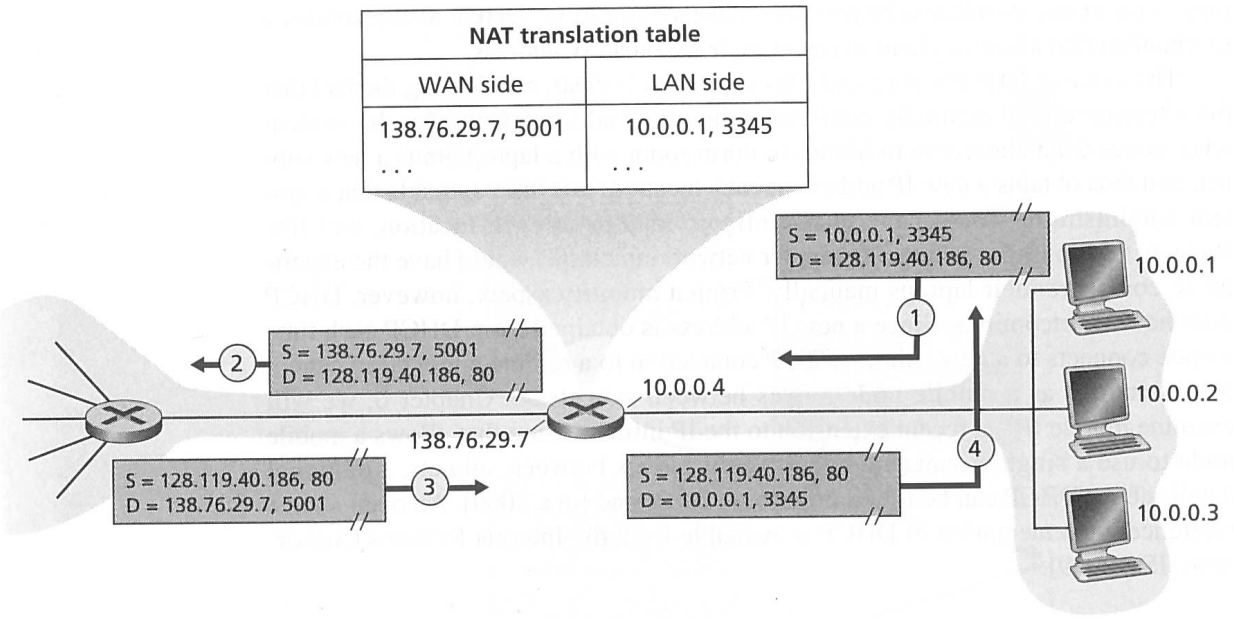


Figure 4.22 ♦ Network address translation

thousands of home networks, many using the same address space, 10.0.0.0/24. Devices within a given home network can send packets to each other using 10.0.0.0/24 addressing. However, packets forwarded *beyond* the home network into the larger global Internet clearly cannot use these addresses (as either a source or a destination address) because there are hundreds of thousands of networks using this block of addresses. That is, the 10.0.0.0/24 addresses can only have meaning within the given home network. But if private addresses only have meaning within a given network, how is addressing handled when packets are sent to or received from the global Internet, where addresses are necessarily unique? The answer lies in understanding NAT.

The NAT-enabled router does not *look* like a router to the outside world. Instead the NAT router behaves to the outside world as a *single* device with a *single* IP address. In Figure 4.22, all traffic leaving the home router for the larger Internet has a source IP address of 138.76.29.7, and all traffic entering the home router must have a destination address of 138.76.29.7. In essence, the NAT-enabled router is hiding the details of the home network from the outside world. (As an aside, you might wonder where the home network computers get their addresses and where the router gets its single IP address. Often, the answer is the same—DHCP! The router gets its address from the ISP's DHCP server, and the router runs a DHCP server to provide addresses to computers within the NAT-DHCP-router-controlled home network's address space.)

If all datagrams arriving at the NAT router from the WAN have the same destination IP address (specifically, that of the WAN-side interface of the NAT router), then how does the router know the internal host to which it should forward a given datagram? The trick is to use a **NAT translation table** at the NAT router, and to include port numbers as well as IP addresses in the table entries.

Consider the example in Figure 4.22. Suppose a user sitting in a home network behind host 10.0.0.1 requests a Web page on some Web server (port 80) with IP address 128.119.40.186. The host 10.0.0.1 assigns the (arbitrary) source port number 3345 and sends the datagram into the LAN. The NAT router receives the datagram, generates a new source port number 5001 for the datagram, replaces the source IP address with its WAN-side IP address 138.76.29.7, and replaces the original source port number 3345 with the new source port number 5001. When generating a new source port number, the NAT router can select any source port number that is not currently in the NAT translation table. (Note that because a port number field is 16 bits long, the NAT protocol can support over 60,000 simultaneous connections with a single WAN-side IP address for the router!) NAT in the router also adds an entry to its NAT translation table. The Web server, blissfully unaware that the arriving datagram containing the HTTP request has been manipulated by the NAT router, responds with a datagram whose destination address is the IP address of the NAT router, and whose destination port number is 5001. When this datagram arrives at the NAT router, the router indexes the NAT translation table using the destination IP address and destination port number to obtain the appropriate IP address (10.0.0.1) and destination port number (3345) for the browser in the home network. The router then rewrites the datagram's destination address and destination port number, and forwards the datagram into the home network.

NAT has enjoyed widespread deployment in recent years. But we should mention that many purists in the IETF community loudly object to NAT. First, they argue, port numbers are meant to be used for addressing processes, not for addressing hosts. (This violation can indeed cause problems for servers running on the home network, since, as we have seen in Chapter 2, server processes wait for incoming requests at well-known port numbers.) Second, they argue, routers are supposed to process packets only up to layer 3. Third, they argue, the NAT protocol violates the so-called end-to-end argument; that is, hosts should be talking directly with each other, without interfering nodes modifying IP addresses and port numbers. And fourth, they argue, we should use IPv6 (see Section 4.4.4) to solve the shortage of IP addresses, rather than recklessly patching up the problem with a stopgap solution like NAT. But like it or not, NAT has become an important component of the Internet.

Yet another major problem with NAT is that it interferes with P2P applications, including P2P file-sharing applications and P2P Voice over IP applications. Recall from Chapter 2 that in a P2P application, any participating Peer A should be able to initiate a TCP connection to any other participating Peer B. The essence of the problem is that if Peer B is behind a NAT, it cannot act as a server and accept TCP

connections. As we'll see in the homework problems, this NAT problem can be circumvented if Peer A is not behind a NAT. In this case, Peer A can first contact Peer B through an intermediate Peer C, which is not behind a NAT and to which B has established an ongoing TCP connection. Peer A can then ask Peer B, via Peer C, to initiate a TCP connection directly back to Peer A. Once the direct P2P TCP connection is established between Peers A and B, the two peers can exchange messages or files. This hack, called **connection reversal**, is actually used by many P2P applications for **NAT traversal**. If both Peer A and Peer B are behind their own NATs, the situation is a bit trickier but can be handled using application relays, as we saw with Skye relays in Chapter 2.

UPnP

NAT traversal is increasingly provided by Universal Plug and Play (UPnP), which is a protocol that allows a host to discover and configure a nearby NAT [UPnP Forum 2009]. UPnP requires that both the host and the NAT be UPnP compatible. With UPnP, an application running in a host can request a NAT mapping between its (*private IP address, private port number*) and the (*public IP address, public port number*) for some requested public port number. If the NAT accepts the request and creates the mapping, then nodes from the outside can initiate TCP connections to (*public IP address, public port number*). Furthermore, UPnP lets the application know the value of (*public IP address, public port number*), so that the application can advertise it to the outside world.

As an example, suppose your host, behind a UPnP-enabled NAT, has private address 10.0.0.1 and is running BitTorrent on port 3345. Also suppose that the public IP address of the NAT is 138.76.29.7. Your BitTorrent application naturally wants to be able to accept connections from other hosts, so that it can trade chunks with them. To this end, the BitTorrent application in your host asks the NAT to create a "hole" that maps (10.0.0.1, 3345) to (138.76.29.7, 5001). (The public port number 5001 is chosen by the application.) The BitTorrent application in your host could also advertise to its tracker that it is available at (138.76.29.7, 5001). In this manner, an external host running BitTorrent can contact the tracker and learn that your BitTorrent application is running at (138.76.29.7, 5001). The external host can send a TCP SYN packet to (138.76.29.7, 5001). When the NAT receives the SYN packet, it will change the destination IP address and port number in the packet to (10.0.0.1, 3345) and forward the packet through the NAT.

In summary, UPnP allows external hosts to initiate communication sessions to NATed hosts, using either TCP or UDP. NATs have long been a nemesis for P2P applications; UPnP, providing an effective and robust NAT traversal solution, may be their savior. Our discussion of NAT and UPnP here has been necessarily brief. For more detailed discussions of NAT see [Huston 2004, Cisco NAT 2009].

Network Working Group
Request for Comments: 3194
Updates: 1715
Category: Informational

A. Durand
SUN Microsystems
C. Huitema
Microsoft

November 2001

The Host-Density Ratio for Address Assignment Efficiency: An update on the H ratio

Status of this Memo

Copyright Notice

Abstract

This document provides an update on the "H ratio" defined in RFC 1715. It defines a new ratio which the authors claim is easier to understand.

1. Evaluating the efficiency of address allocation

A naive observer might assume that the number of addressable objects in an addressing plan is a linear function of the size of the address. If this were true, a telephone numbering plan based on 10 digits would be able to number 10 billion telephones, and the IPv4 32 bit addresses would be adequate for numbering 4 billion computers (using the American English definition of a billion, i.e. one thousand millions.) We all know that this is not correct: the 10 digit plan is stressed today, and it handles only a few hundred million telephones in North America; the Internet registries have started to implement increasingly restrictive allocation policies when there were only a few tens of million computers on the Internet.

Addressing plans are typically organized as a hierarchy: in telephony, the first digits will designate a region, the next digits will designate an exchange, and the last digits will designate a subscriber within this exchange; in computer networks, the most significant bits will designate an address range allocated to a network provider, the next bits will designate the network of an organization served by that provider, and then the subnet to which the individual computers are connected. At each level of the

hierarchy, one has to provide some margins: one has to allocate more digits to the region code than the current number of regions would necessitate, and more bits in a subnet than strictly required by the number of computers. The number of elements in any given level of the hierarchy will change over time, due to growth and mobility. If the current allocation is exceeded, one has to engage in renumbering, which is painful and expensive. In short, trying to squeeze too many objects into a hierarchical address space increases the level of pain endured by operators and subscribers.

Back in 1993, when we were debating the revision of the Internet Protocol, we wondered what the acceptable ratio of utilization was of a given addressing plan. Coming out with such a ratio was useful to assess how many computers could be connected to the Internet with the current 32-bit addresses, as well as to decide the size of the next generation addresses. The second point is now decided, with 128-bits addresses for IPv6, but the first question is still relevant: knowing the capacity of the current address plan will help us predict the date at which this capacity will be exceeded.

Participants in the IPNG debates initially measured the efficiency of address allocation by simply dividing the number of allocated addresses by the size of the address space. This is a simple measure, but it is largely dependent on the size of the address space. Loss of efficiency at each level of a hierarchical plan has a multiplicative effect; for example, 50% efficiency at each stage of a three level hierarchy results in a overall efficiency of 12.5%. If we want a "pain level indicator", we have to use a ratio that takes into account these multiplicative effects.

[.. omitted ..]

2. Definition of the HD-ratio

When considering an addressing plan to allocate objects, the host density ratio HD is defined as follow:

$$HD = \frac{\log(\text{number of allocated objects})}{\log(\text{maximum number of allocatable objects})}$$

This ratio is defined for any number of allocatable objects greater than 1 and any number of allocated objects greater or equal than 1 and less than or equal the maximum number of allocatable objects. The ratio is usually presented as a percentage, e.g. 70%. It varies between 0 (0%), when there is just one allocation, and 1 (100%), when there is one object allocated to each available address. Note that for the calculation of the HD-ratio, one can use any base for the logarithm as long as it is the same for both the numerator and the denominator.

[.. omitted ..]

3. Using the HD-ratio as an indicator of the pain level

In order to assess whether the H-Ratio was a good predictor of the "pain level" caused by a specific efficiency, RFC1715 used several examples of networks that had reached their capacity limit. These could be for example telephone networks at the point when they decided to add digits to their numbering plans, or computer networks at the point when their addressing capabilities were perceived as stretched beyond practical limits. The idea behind these examples is that network managers would delay renumbering or changing the network protocol until it became just too painful; the ratio just before the change is thus a good predictor of what can be achieved in practice. The examples were the following:

* Adding one digit to all French telephone numbers, moving from 8 digits to 9, when the number of phones reached a threshold of 1.0 E+7.

$$HD(FrenchTelephone8digit) = \frac{\log(1.0E+7)}{\log(1.0E+8)} = 0.8750 = 87.5\%$$

$$HD(FrenchTelephone9digit) = \frac{\log(1.0E+7)}{\log(1.0E+9)} = 0.7778 = 77.8\%$$

* Expanding the number of areas in the US telephone system, making the phone number effectively 10 digits long instead of "9.2" (the second digit of area codes used to be limited to 0 or 1) for about 1.0 E+8 subscribers.

$$HD(USTelephone9.2digit) = \frac{\log(1.0E+8)}{\log(9.5E+9)} = 0.8696 = 87.0\ \%$$

$$HD(USTelephone10digit) = \frac{\log(1.0E+8)}{\log(1E+10)} = 0.8000 = 80.0\ \%$$

* The globally-connected physics/space science DECnet (Phase IV) stopped growing at about 15K nodes (i.e. new nodes were hidden) in a 16 bit address space.

$$HD(DecNET\ IV) = \frac{\log(15000)}{\log(2^{16})} = 0.8670 = 86.7\ \%$$

From those examples, we can note that these addressing systems reached their limits for very close values of the HD-ratio. We can use the same examples to confirm that the definition of the HD-ratio as a quotient of logarithms results in better prediction than the direct quotient of allocated objects over size of the address space. In our three examples, the direct quotients were 10%, 3.2% and 22.8%, three very different numbers that don't lead to any obvious generalization. The examples suggest an HD-ratio value on the order of 85% and above correspond to a high pain level, at which operators are ready to make drastic decisions.

We can also examine our examples and hypothesize that the operators who renumbered their networks tried to reach, after the renumbering, a pain level that was easily supported. The HD-ratio of the French or US network immediately after renumbering was 78% and 80%, respectively. This suggests that values of 80% or less corresponds to comfortable trade-offs between pain and efficiency.

4. Using the HD-ratio to evaluate the capacity of addressing plans

Directly using the HD-ratio makes it easy to evaluate the density of allocated objects. Evaluating how well an addressing plan will scale requires the reverse calculation. We have seen in section 3.1 that an HD-ratio lower than 80% is manageable, and that HD-ratios higher than 87% are hard to sustain. This should enable us to compute the acceptable and "practical maximum" number of objects that can be allocated given a specific address size, using the formula:

number allocatable of objects
= exp(HD x log(maximum number allocatable of objects))
= (maximum number allocatable of objects)^HD

The following table provides example values for a 9-digit telephone plan, a 10-digit telephone plan, and the 32-bit IPv4 Internet:

	Reasonable HD=80%	Painful HD=85%	Very Painful HD=86%	Practical Maximum HD=87%
9-digits plan	16 M	45 M	55 M	68 M
10-digits plan	100 M	316 M	400 M	500 M
32-bits addresses	51 M	154 M	192 M	240 M

Note: 1M = 1,000,000

Indeed, the practical maximum depends on the level of pain that the users and providers are willing to accept. We may very well end up with more than 154M allocated IPv4 addresses in the next years, if we are willing to accept the pain.

[.. omitted ..]

9. References

[RFC1715] Huitema, C., "The H Ratio for Address Assignment Efficiency", RFC 1715, November 1994.

[IANAV4] INTERNET PROTOCOL V4 ADDRESS SPACE, maintained by the IANA, http://www.iana.org/assignments/ipv4-address-space

[DMNSRV] Internet Domain Survey, Internet Software Consortium, http://www.isc.org/ds/

[NETSZR] Netsizer, Telcordia Technologies, http://www.netsizer.com/

10. Full Copyright Statement

The Bellman–Ford algorithm. Suppose that node 1 is the "destination" node and consider the problem of finding a shortest path from every node to node 1. We assume that there exists at least one path from every node to the destination. To simplify the presentation, let us denote $d_{ij} = \infty$ if (i, j) is not an arc of the graph. Using this convention we can assume without loss of generality that there is an arc between every pair of nodes, since walks and paths consisting of true network arcs are the only ones with length less than ∞.

A shortest walk from a given node i to node 1, subject to the constraint that the walk contains at most h arcs and goes through node 1 only once, is referred to as a *shortest* ($\leq h$) *walk* and its length is denoted by D_i^h. Note that such a walk may not be a path, that is, it may contain repeated nodes; we will later give conditions under which this is not possible. By convention, we take

$$D_1^h = 0, \qquad \text{for all } h$$

We will prove shortly that D_i^h can be generated by the iteration

$$D_i^{h+1} = \min_j \left[d_{ij} + D_j^h \right], \qquad \text{for all } i \neq 1 \tag{5.1}$$

starting from the initial conditions

$$D_i^0 = \infty, \qquad \text{for all } i \neq 1 \tag{5.2}$$

This is the Bellman–Ford algorithm, illustrated in Fig. 5.31. Thus, we claim that the Bellman–Ford algorithm first finds the one-arc shortest walk lengths, then finds the two-arc shortest walk lengths, and so forth. Once we show this, we will argue that the shortest walk lengths are equal to the shortest path lengths, under the additional assumption that all cycles not containing node 1 have nonnegative length. We say that the algorithm terminates after h iterations if

$$D_i^h = D_i^{h-1}, \qquad \text{for all } i$$

The following proposition provides the main results.

Proposition. Consider the Bellman–Ford algorithm (5.1) with the initial conditions $D_i^0 = \infty$ for all $i \neq 1$. Then:

(a) The scalars D_i^h generated by the algorithm are equal to the shortest ($\leq h$) walk lengths from node i to node 1.

(b) The algorithm terminates after a finite number of iterations if and only if all cycles not containing node 1 have nonnegative length. Furthermore, if the algorithm terminates, it does so after at most $h \leq N$ iterations, and at termination, D_i^h is the shortest path length from i to 1.

Proof: (a) We argue by induction. From Eqs. (5.1) and (5.2) we have

$$D_i^1 = d_{i1}, \qquad \text{for all } i \neq 1$$

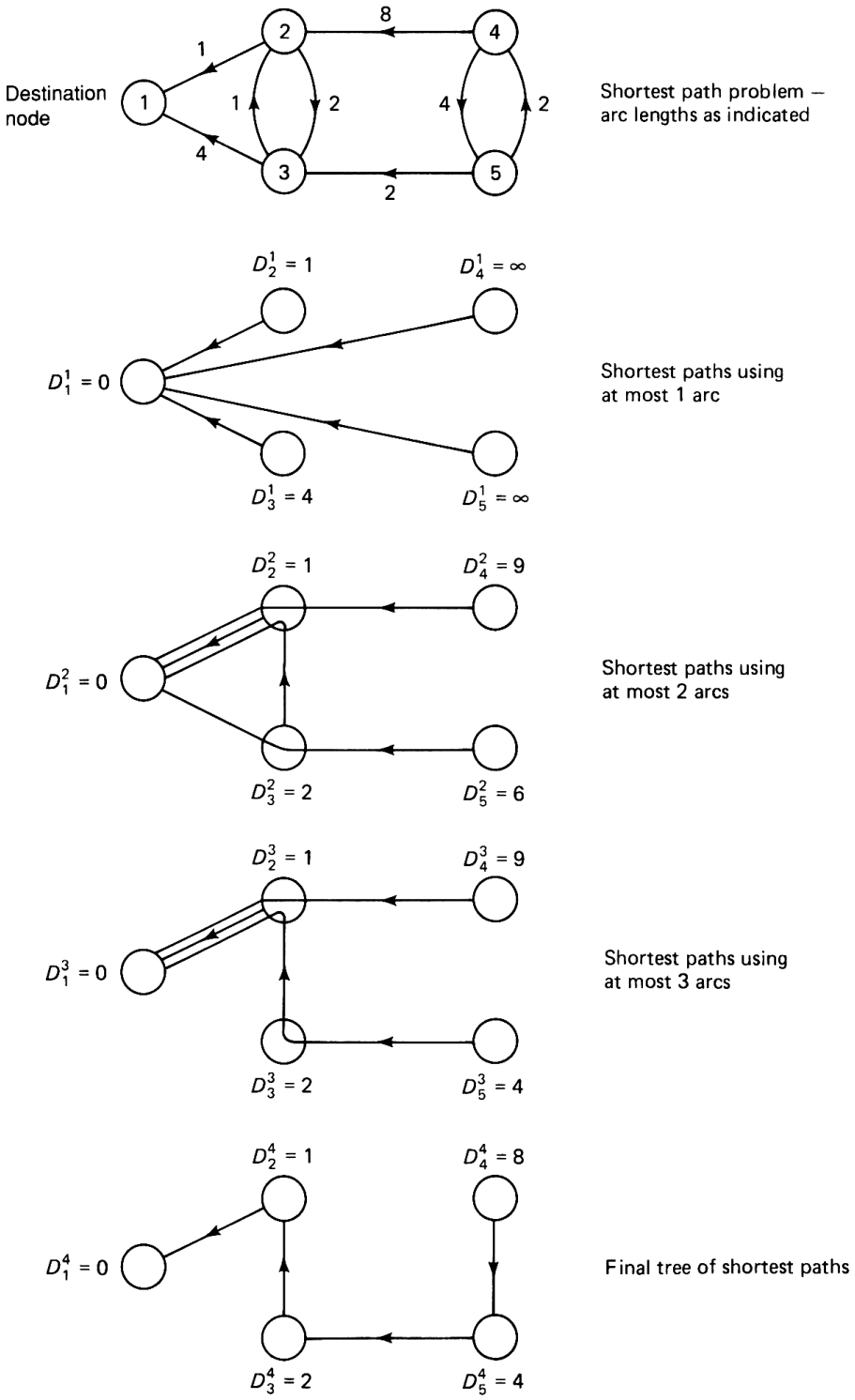


Figure 5.31 Successive iterations of the Bellman–Ford method. In this example, the shortest ($\leq h$) walks are paths because all arc lengths are positive and therefore all cycles have positive length. The shortest paths are found after $N - 1$ iterations, which is equal to 4 in this example.

so D_i^1 is indeed equal to the shortest (≤ 1) walk length from i to 1. Suppose that D_i^k is equal to the shortest ($\leq k$) walk length from i to 1 for all $k \leq h$. We will show that D_i^{h+1} is the shortest ($\leq h+1$) walk length from i to 1. Indeed, a shortest ($\leq h+1$) walk from i to 1 either consists of less than $h+1$ arcs, in which case its length is equal to D_i^h, or else it consists of $h+1$ arcs with the first arc being (i,j) for some $j \neq 1$, followed by an h-arc walk from j to 1 in which node 1 is not repeated. The latter walk must be a shortest ($\leq h$) walk from j to 1. [Otherwise, by concatenating arc (i,j) and a shorter ($\leq h$) walk from j to 1, we would obtain a shorter ($\leq h+1$) walk from i to 1.] We thus conclude that

$$\text{Shortest } (\leq h+1) \text{ walk length} = \min\left\{D_i^h, \min_{j\neq 1}\left[d_{ij} + D_j^h\right]\right\} \tag{5.3}$$

Using the induction hypothesis, we have $D_j^k \leq D_j^{k-1}$ for all $k \leq h$ [since the set of ($\leq k$) walks from node j to 1 contains the corresponding set of ($\leq k-1$) walks]. Therefore,

$$D_i^{h+1} = \min_j\left[d_{ij} + D_j^h\right] \leq \min_j\left[d_{ij} + D_j^{h-1}\right] = D_i^h \tag{5.4}$$

Furthermore, we have $D_i^h \leq D_i^1 = d_{i1} = d_{i1} + D_1^h$, so from Eq. (5.3) we obtain

$$\text{Shortest } (\leq h+1) \text{ walk length} = \min\left\{D_i^h, \min_j\left[d_{ij} + D_j^h\right]\right\} = \min\left\{D_i^h, D_i^{h+1}\right\}$$

In view of $D_i^{h+1} \leq D_i^h$ [cf. Eq. (5.4)], this yields

$$\text{Shortest } (\leq h+1) \text{ walk length} = D_i^{h+1}$$

completing the induction proof.

(b) If the Bellman-Ford algorithm terminates after h iterations, we must have

$$D_i^k = D_i^h, \qquad \text{for all } i \text{ and } k \geq h \tag{5.5}$$

so we cannot reduce the lengths of the shortest walks by allowing more and more arcs in these walks. It follows that there cannot exist a negative-length cycle not containing node 1, since such a cycle could be repeated an arbitrarily large number of times in walks from some nodes to node 1, thereby making their length arbitrarily small and contradicting Eq. (5.5). Conversely, suppose that all cycles not containing node 1 have nonnegative length. Then by deleting all such cycles from shortest ($\leq h$) walks, we obtain paths of less or equal length. Therefore, for every i and h, there exists a path that is a shortest ($\leq h$) walk from i to 1, and the corresponding shortest path length is equal to D_i^h. Since paths have no cycles, they can contain at most $N-1$ arcs. It follows that

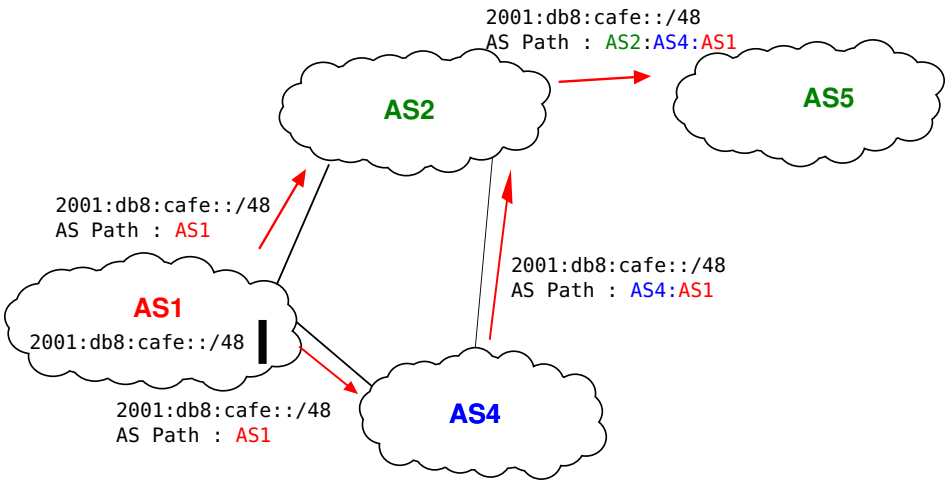
$$D_i^N = D_i^{N-1}, \qquad \text{for all } i$$

implying that the algorithm terminates after at most N iterations. **Q.E.D.**

The Border Gateway Protocol

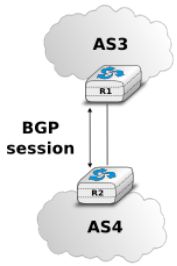
The Internet uses a single interdomain routing protocol : the Border Gateway Protocol (BGP). The current version of BGP is defined in **RFC 4271**. BGP differs from the intradomain routing protocols that we have already discussed in several ways. First, BGP is a *path-vector* protocol. When a BGP router advertises a route towards a prefix, it announces the IP prefix and the interdomain path used to reach this prefix. From BGP's point of view, each domain is identified by a unique *Autonomous System* (AS) number **[5]** and the interdomain path contains the AS numbers of the transit domains that are used to reach the associated prefix. This interdomain path is called the *AS Path*. Thanks to these AS-Paths, BGP does not suffer from the count-to-infinity problems that affect distance vector routing protocols. Furthermore, the AS-Path can be used to implement some routing policies. Another difference between BGP and the intradomain routing protocols is that a BGP router does not send the entire contents of its routing table to its neighbours regularly. Given the size of the global Internet, routers would be overloaded by the number of BGP messages that they would need to process. BGP uses incremental updates, i.e. it only announces the routes that have changed to its neighbors.

The figure below shows a simple example of the BGP routes that are exchanged between domains. In this example, prefix *2001:db8:1234/48* is announced by *AS1*. *AS1* advertises a BGP route towards this prefix to *AS2*. The AS-Path of this route indicates that *AS1* is the originator of the prefix. When *AS4* receives the BGP route from *AS1*, it re-announces it to *AS2* and adds its AS number to the AS-Path. *AS2* has learned two routes towards prefix *2001:db8:1234/48*. It compares the two routes and prefers the route learned from *AS4* based on its own ranking algorithm. *AS2* advertises to *AS5* a route towards *2001:db8:1234/48* with its AS-Path set to *AS2:AS4:AS1*. Thanks to the AS-Path, *AS5* knows that if it sends a packet towards *2001:db8:1234/48* the packet first passes through *AS2*, then through *AS4* before reaching its destination inside *AS1*.



Simple exchange of BGP routes

BGP routers exchange routes over BGP sessions. A BGP session is established between two routers belonging to two different domains that are directly connected. As explained earlier, the physical connection between the two routers can be implemented as a private peering link or over an Internet eXchange Point. A BGP session between two adjacent routers runs above a TCP connection (the default BGP port is 179). In contrast with intradomain routing protocols that exchange IP packets or UDP segments, BGP runs above TCP because TCP ensures a reliable delivery of the BGP messages sent by each router without forcing the routers to implement acknowledgements, checksums, etc. Furthermore, the two routers consider the peering link to be up as long as the BGP session and the underlying TCP connection remain up **[6]**. The two endpoints of a BGP session are called *BGP peers*.



A BGP peering session between two directly connected routers

In practice, to establish a BGP session between routers *R1* and *R2* in the figure above, the network administrator of *AS3* must first configure on *R1* the IP address of *R2* on the *R1-R2* link and the AS number of *R2*. Router *R1* then regularly tries to establish the BGP session with *R2*. *R2* only agrees to establish the BGP session with *R1* once it has been configured with the IP address of *R1* and its AS number. For security reasons, a router never establishes a BGP session that has not been manually configured on the router.

The BGP protocol **RFC 4271** defines several types of messages that can be exchanged over a BGP session :

- *OPEN* : this message is sent as soon as the TCP connection between the two routers has been established. It initialises the BGP session and allows the negotiation of some options. Details about this message may be found in **RFC 4271**
- *NOTIFICATION* : this message is used to terminate a BGP session, usually because an error has been detected by the BGP peer. A router that sends or receives a *NOTIFICATION* message immediately shutdowns the corresponding BGP session.
- *UPDATE*: this message is used to advertise new or modified routes or to withdraw previously advertised routes.
- *KEEPALIVE* : this message is used to ensure a regular exchange of messages on the BGP session, even when no route changes. When a BGP router has not sent an *UPDATE* message during the last 30 seconds, it shall send a *KEEPALIVE* message to confirm to the other peer that it is still up. If a peer does not receive any BGP message during a period of 90 seconds **[7]**, the BGP session is considered to be down and all the routes learned over this session are withdrawn.

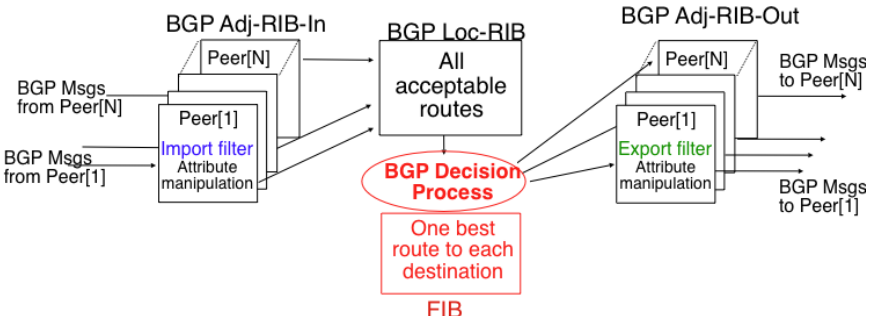
As explained earlier, BGP relies on incremental updates. This implies that when a BGP session starts, each router first sends BGP *UPDATE* messages to advertise to the other peer all the exportable routes that it knows. Once all these routes have been advertised, the BGP router only sends BGP *UPDATE* messages about a prefix if the route is new, one of its attributes has changed or the route became unreachable and must be withdrawn. The BGP *UPDATE* message allows BGP routers to efficiently exchange such information while minimising the number of bytes exchanged. Each *UPDATE* message contains :

- a list of IP prefixes that are withdrawn
- a list of IP prefixes that are (re-)advertised
- the set of attributes (e.g. AS-Path) associated to the advertised prefixes

In the remainder of this chapter, and although all routing information is exchanged using BGP *UPDATE* messages, we assume for simplicity that a BGP message contains only information about one prefix and we use the words :

- *Withdraw message* to indicate a BGP *UPDATE* message containing one route that is withdrawn
- *Update message* to indicate a BGP *UPDATE* containing a new or updated route towards one destination prefix with its attributes

From a conceptual point of view, a BGP router connected to *N* BGP peers, can be described as being composed of four parts as shown in the figure below.



Organisation of a BGP router

In this figure, the router receives BGP messages on the left part of the figure, processes these messages and possibly sends BGP messages on the right part of the figure. A BGP router contains three important data structures :

- the *Adj-RIB-In* contains the BGP routes that have been received from each BGP peer. The routes in the *Adj-RIB-In* are filtered by the *import filter* before being placed in the *BGP-Loc-RIB*. There is one *import filter* per BGP peer.
- the *Local Routing Information Base* (*Loc-RIB*) contains all the routes that are considered as acceptable by the router. The *Loc-RIB* may contain several routes, learned from different BGP peers, towards the same destination prefix.
- the *Forwarding Information Base* (*FIB*) is used by the dataplane to forward packets towards their destination. The *FIB* contains, for each destination, the best route that has been selected by the *BGP decision process*. This decision process is an algorithm that selects, for each destination prefix, the best route according to the router's ranking algorithm that is part of its policy.

- the *Adj-RIB-Out* contains the BGP routes that have been advertised to each BGP peer. The *Adj-RIB-Out* for a given peer is built by applying the peer`s *export filter* on the routes that have been installed in the *FIB*. There is one *export filter* per BGP peer. For this reason, the Adj-RIB-Out of a peer may contain different routes than the Adj-RIB-Out of another peer.

When a BGP session starts, the routers first exchange *OPEN* messages to negotiate the options that apply throughout the entire session. Then, each router extracts from its FIB the routes to be advertised to the peer. It is important to note that, for each known destination prefix, a BGP router can only advertise to a peer the route that it has itself installed inside its *FIB*. The routes that are advertised to a peer must pass the peer's *export filter*. The *export filter* is a set of rules that define which routes can be advertised over the corresponding session, possibly after having modified some of its attributes. One *export filter* is associated to each BGP session. For example, on a *shared-cost peering*, the *export filter* only selects the internal routes and the routes that have been learned from a *customer*. The pseudo-code below shows the initialisation of a BGP session.

```
def initialize_BGP_session( RemoteAS, RemoteIP):
  # Initialize and start BGP session
  # Send BGP OPEN Message to RemoteIP on port 179
  # Follow BGP state machine
  # advertise local routes and routes learned from peers*/
  for d in BGPLocRIB :
      B=build_BGP_Update(d)
      S=Apply_Export_Filter(RemoteAS,B)
      if (S != None) :
         send_Update(S,RemoteAS,RemoteIP)
  # entire RIB has been sent
  # new Updates will be sent to reflect local or distant
  # changes in routers
```

In the above pseudo-code, the *build_BGP_UPDATE(d)* procedure extracts from the *BGP Loc-RIB* the best path towards destination *d* (i.e. the route installed in the FIB) and prepares the corresponding BGP *UPDATE* message. This message is then passed to the *export filter* that returns NULL if the route cannot be advertised to the peer or the (possibly modified) BGP *UPDATE* message to be advertised. BGP routers allow network administrators to specify very complex *export filters*, see e.g. **[WMS2004]**. A simple *export filter* that implements the equivalent of *split horizon* is shown below.

```
def apply_export_filter(RemoteAS, BGPMsg) :
  # check if RemoteAS already received route
  if RemoteAS is BGPMsg.ASPath :
     BGPMsg=None
     # Many additional export policies can be configured :
     # Accept or refuse the BGPMsg
     # Modify selected attributes inside BGPMsg
  return BGPMsg
```

At this point, the remote router has received all the exportable BGP routes. After this initial exchange, the router only sends *BGP UPDATE* messages when there is a change (addition

of a route, removal of a route or change in the attributes of a route) in one of these exportable routes. Such a change can happen when the router receives a BGP message. The pseudo-code below summarizes the processing of these BGP messages.

```
def Recvd_BGPMsg(Msg, RemoteAS) :
    B=apply_import_filter(Msg,RemoteAS)
    if (B== None): # Msg not acceptable
       return
    if IsUPDATE(Msg):
       Old_Route=BestRoute(Msg.prefix)
       Insert_in_RIB(Msg)
       Run_Decision_Process(RIB)
       if (BestRoute(Msg.prefix) != Old_Route) :
          # best route changed
          B=build_BGP_Message(Msg.prefix);
          S=apply_export_filter(RemoteAS,B);
          if (S!=None) : # announce best route
            send_UPDATE(S,RemoteAS,RemoteIP);
          else if (Old_Route != None) :
            send_WITHDRAW(Msg.prefix,RemoteAS, RemoteIP)
     else : # Msg is WITHDRAW
         Old_Route=BestRoute(Msg.prefix)
         Remove_from_RIB(Msg)
         Run_Decision_Process(RIB)
         if (Best_Route(Msg.prefix) !=Old_Route):
           # best route changed
           B=build_BGP_Message(Msg.prefix)
           S=apply_export_filter(RemoteAS,B)
           if (S != None) : # still one best route towards Msg.prefix
              send_UPDATE(S,RemoteAS, RemoteIP);
           else if(Old_Route != None) : # No best route anymore
               send_WITHDRAW(Msg.prefix,RemoteAS,RemoteIP);
```

When a BGP message is received, the router first applies the peer's *import filter* to verify whether the message is acceptable or not. If the message is not acceptable, the processing stops. The pseudo-code below shows a simple *import filter*. This *import filter* accepts all routes, except those that already contain the local AS in their AS-Path. If such a route was used, it would cause a routing loop. Another example of an *import filter* would be a filter used by an Internet Service Provider on a session with a customer to only accept routes towards the IP prefixes assigned to the customer by the provider. On real routers, *import filters* can be much more complex and some *import filters* modify the attributes of the received BGP *UPDATE* **[WMS2004]** .

```
def apply_import_filter(RemoteAS, BGPMsg):
    if MysAS in BGPMsg.ASPath :
       BGPMsg=None
       # Many additional import policies can be configured :
       # Accept or refuse the BGPMsg
       # Modify selected attributes inside BGPMsg
    return BGPMsg
```

Note

The bogon filters

Another example of frequently used *import filters* are the filters that Internet Service Providers use to ignore bogon routes. In the ISP community, a bogon route is a route that should not be advertised on the global Internet. Typical examples include the documentation IPv6 prefix (*2001:db8::/32* used for most examples in this book), the loopback address (::1/128`) or the IPv6 prefixes that have not yet been allocated by IANA. A well managed BGP router should ensure that it never advertises bogons on the global Internet. Detailed information about these bogons may be found in **[IMHM2013]**.

If the import filter accepts the BGP message, the pseudo-code distinguishes two cases. If this is an *Update message* for prefix *p*, this can be a new route for this prefix or a modification of the route's attributes. The router first retrieves from its *RIB* the best route towards prefix *p*. Then, the new route is inserted in the *RIB* and the *BGP decision process* is run to find whether the best route towards destination *p* changes. A BGP message only needs to be sent to the router's peers if the best route has changed. For each peer, the router applies the *export filter* to verify whether the route can be advertised. If yes, the filtered BGP message is sent. Otherwise, a *Withdraw message* is sent. When the router receives a *Withdraw message*, it also verifies whether the removal of the route from its *RIB* caused its best route towards this prefix to change. It should be noted that, depending on the content of the *RIB* and the *export filters*, a BGP router may need to send a *Withdraw message* to a peer after having received an *Update message* from another peer and conversely.

Let us now discuss in more detail the operation of BGP in an IPv6 network. For this, let us consider the simple network composed of three routers located in three different ASes and shown in the figure below.

Utilisation of the BGP nexthop attribute

This network contains three routers : *R1*, *R2* and *R3*. Each router is attached to a local IPv6 subnet that it advertises using BGP. There are two BGP sessions, one between *R1* and *R2* and the second between *R2* and *R3*. A */127* subnet is used on each interdomain link (*2001:db8::4/127* on *R1-R2* and *2001:db8::0/127* on *R2-R3*) in conformance with the latest recommendation **RFC 6164**. The BGP sessions run above TCP connections established between the neighboring routers (e.g. *2001:db8::5 - 2001:db8::6* for the *R1-R2* session).

Let us assume that the *R1-R2* BGP session is the first to be established. A *BGP Update* message sent on such a session contains three fields :

- the advertised prefix

- the *BGP nexthop*
- the attributes including the AS-Path

We use the notation *U(prefix, nexthop, attributes)* to represent such a *BGP Update* message in this section. Similarly, *W(prefix)* represents a *BGP withdraw* for the specified prefix. Once the *R1-R2* session has been established, *R1* sends *U(2001:db8:1234::/48,2001:db8::5,AS10)* to *R2* and *R2* sends *U(2001:db8:5678:/48,2001:db8::6,AS20)*. At this point, *R1* can reach *2001:db8:5678::/48* via *2001:db8::6* and *R2* can reach *2001:db8:1234::/48* via *2001:db8::5*.

Once the *R2-R3* has been established, *R3* sends *U(2001:db8:acbd::/48,2001:db8::2,AS30)*. *R2* announces on the *R2-R3* session all the routes inside its RIB. It thus sends to *R3* : *U(2001:db8:1234::/48,2001:db8::1,AS20:AS10)* and *U(2001:db8:5678::/48,2001:db8::1,AS20)*. Note that when *R2* advertises the route that it learned from *R1*, it updates the BGP nexthop and adds its AS number to the AS-Path. *R2* also sends *U(2001:db8:abcd::48,2001:db8::6,AS20:AS30)* to *R1* on the *R1-R3* session. At this point, all BGP routes have been exchanged and all routers can reach *2001:db8::1234/48*, *2001:db8:5678::/48* and *2001:db8:abcd::/48*.

If the link between *R2* and *R3* fails, *R3* detects the failure as it did not receive *KEEPALIVE* messages recently from *R2*. At this time, *R3* removes from its RIB all the routes learned over the *R2-R3* BGP session. *R2* also removes from its RIB the routes learned from *R3*. *R2* also sends *W(2001:db8:acbd::/48)* to *R1* over the *R1-R3* BGP session since it does not have a route anymore towards this prefix.

Note

Origin of the routes advertised by a BGP router

A frequent practical question about the operation of BGP is how a BGP router decides to originate or advertise a route for the first time. In practice, this occurs in two situations :

- the router has been manually configured by the network operator to always advertise one or several routes on a BGP session. For example, on the BGP session between UCLouvain and its provider, **belnet** , UCLouvain's router always advertises the *2001:6a8:3080/48* IPv6 prefix assigned to the campus network
- the router has been configured by the network operator to advertise over its BGP session some of the routes that it learns with its intradomain routing protocol. For example, an enterprise router may advertise over a BGP session with its provider the routes to remote sites when these routes are reachable and advertised by the intradomain routing protocol

The first solution is the most frequent. Advertising routes learned from an intradomain routing protocol is not recommended, this is because if the route flaps **[8]**, this would cause a large number of BGP messages being exchanged

in the global Internet.

The BGP decision process

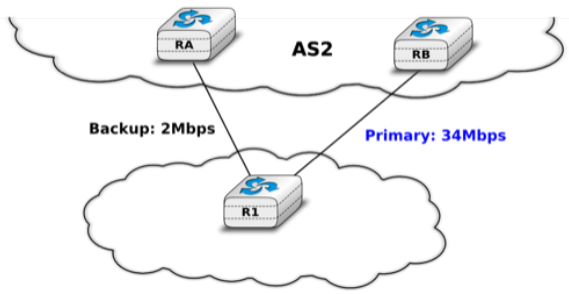
Besides the import and export filters, a key difference between BGP and the intradomain routing protocols is that each domain can define is own ranking algorithm to determine which route is chosen to forward packets when several routes have been learned towards the same prefix. This ranking depends on several BGP attributes that can be attached to a BGP route.

The first BGP attribute that is used to rank BGP routes is the *local-preference* (local-pref) attribute. This attribute is an unsigned integer that is attached to each BGP route received over an eBGP session by the associated import filter.

When comparing routes towards the same destination prefix, a BGP router always prefers the routes with the highest *local-pref*. If the BGP router knows several routes with the same *local-pref*, it prefers among the routes having this *local-pref* the ones with the shortest AS-Path.

The *local-pref* attribute is often used to prefer some routes over others.

A common utilisation of *local-pref* is to support backup links. Consider the situation depicted in the figure below. *AS1* would always like to use the high bandwidth link to send and receive packets via *AS2* and only use the backup link upon failure of the primary one.



How to create a backup link with BGP ?

As BGP routers always prefer the routes with the highest *local-pref* attribute, this policy can be implemented using the following import filter on *R1*

```
import: from  AS2 RA at R1 set localpref=100;
        from  AS2 RB at R1 set localpref=200;
        accept ANY
```

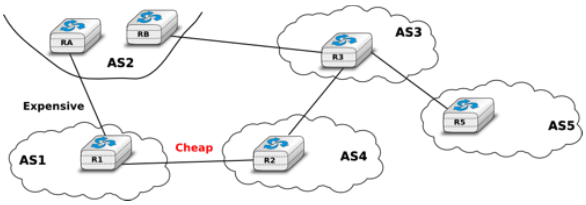
With this import filter, all the BGP routes learned from *RB* over the high bandwidth links are preferred over the routes learned over the backup link. If the primary link fails, the corresponding routes are removed from *R1*'s RIB and *R1* uses the route learned from *RA*.

R1 reuses the routes via *RB* as soon as they are advertised by *RB* once the *R1-RB* link comes back.

The import filter above modifies the selection of the BGP routes inside *AS1*. Thus, it influences the route followed by the packets forwarded by *AS1*. In addition to using the primary link to send packets, *AS1* would like to receive its packets via the high bandwidth link. For this, *AS2* also needs to set the *local-pref* attribute in its import filter.

```
import: from  AS1 R1 at RA set localpref=100;
        from  AS1 R1 at RB set localpref=200;
        accept AS1
```

Sometimes, the *local-pref* attribute is used to prefer a *cheap* link compared to a more expensive one. For example, in the network below, *AS1* could wish to send and receive packets mainly via its interdomain link with *AS4*.



How to prefer a cheap link over an more expensive one ?

AS1 can install the following import filter on *R1* to ensure that it always sends packets via *R2* when it has learned a route via *AS2* and another via *AS4*.

```
import: from  AS2 RA at R1 set localpref=100;
        from  AS4 R2 at R1 set localpref=200;
        accept ANY
```

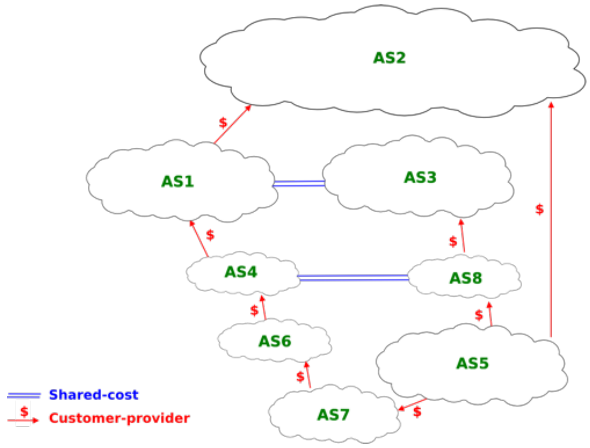
However, this import filter does not influence how *AS3* , for example, prefers some routes over others. If the link between *AS3* and *AS2* is less expensive than the link between *AS3* and *AS4*, *AS3* could send all its packets via *AS2* and *AS1* would receive packets over its expensive link. An important point to remember about *local-pref* is that it can be used to prefer some routes over others to send packets, but it has no influence on the routes followed by received packets.

Another important utilisation of the *local-pref* attribute is to support the *customer->provider* and *shared-cost* peering relationships. From an economic point of view, there is an important difference between these three types of peering relationships. A domain usually earns money when it sends packets over a *provider->customer* relationship. On the other hand, it must pay its provider when it sends packets over a *customer->provider* relationship. Using a *shared-cost* peering to send packets is usually neutral from an economic perspective. To take into account these economic issues, domains usually configure the import filters on their routers as follows :

- insert a high *local-pref* attribute in the routes learned from a customer

- insert a medium *local-pref* attribute in the routes learned over a shared-cost peering
- insert a low *local-pref* attribute in the routes learned from a provider

With such an import filter, the routers of a domain always prefer to reach destinations via their customers whenever such a route exists. Otherwise, they prefer to use *shared-cost* peering relationships and they only send packets via their providers when they do not know any alternate route. A consequence of setting the *local-pref* attribute like this is that Internet paths are often asymmetrical. Consider for example the internetwork shown in the figure below.



Asymmetry of Internet paths

Consider in this internetwork the routes available inside *AS1* to reach *AS5*. *AS1* learns the *AS4:AS6:AS7:AS5* path from *AS4*, the *AS3:AS8:AS5* path from *AS3* and the *AS2:AS5* path from *AS2*. The first path is chosen since it was learned from a customer. *AS5* on the other hand receives three paths towards *AS1* via its providers. It may select any of these paths to reach *AS1* , depending on how it prefers one provider over the others.

BGP convergence

In the previous sections, we have explained the operation of BGP routers. Compared to intradomain routing protocols, a key feature of BGP is its ability to support interdomain routing policies that are defined by each domain as its import and export filters and ranking process. A domain can define its own routing policies and router vendors have implemented many configuration tweaks to support complex routing policies. However, the routing policy chosen by a domain may interfere with the routing policy chosen by another domain. To understand this issue, let us first consider the simple internetwork shown below.

1.6 Networks Under Attack

The Internet has become mission critical for many institutions today, including large and small companies, universities, and government agencies. Many individuals also rely on the Internet for many of their professional, social, and personal activities. Billions of "things," including wearables and home devices, are currently being connected to the Internet. But behind all this utility and excitement, there is a dark side, a side where "bad guys" attempt to wreak havoc in our daily lives by damaging our Internet-connected computers, violating our privacy, and rendering inoperable the Internet services on which we depend.

The field of network security is about how the bad guys can attack computer networks and about how we, soon-to-be experts in computer networking, can defend networks against those attacks, or better yet, design new architectures that are immune to such attacks in the first place. Given the frequency and variety of existing attacks as well as the threat of new and more destructive future attacks, network security has become a central topic in the field of computer networking. One of the features of this textbook is that it brings network security issues to the forefront.

Since we don't yet have expertise in computer networking and Internet protocols, we'll begin here by surveying some of today's more prevalent security-related problems. This will whet our appetite for more substantial discussions in the upcoming chapters. So we begin here by simply asking, what can go wrong? How are computer networks vulnerable? What are some of the more prevalent types of attacks today?

The Bad Guys Can Put Malware into Your Host Via the Internet

We attach devices to the Internet because we want to receive/send data from/to the Internet. This includes all kinds of good stuff, including Instagram posts, Internet search results, streaming music, video conference calls, streaming movies, and so on. But, unfortunately, along with all that good stuff comes malicious stuff—collectively known as **malware**—that can also enter and infect our devices. Once malware infects our device it can do all kinds of devious things, including deleting our files and installing spyware that collects our private information, such as social

security numbers, passwords, and keystrokes, and then sends this (over the Internet, of course!) back to the bad guys. Our compromised host may also be enrolled in a network of thousands of similarly compromised devices, collectively known as a **botnet**, which the bad guys control and leverage for spam e-mail distribution or distributed denial-of-service attacks (soon to be discussed) against targeted hosts.

Much of the malware out there today is **self-replicating**: once it infects one host, from that host it seeks entry into other hosts over the Internet, and from the newly infected hosts, it seeks entry into yet more hosts. In this manner, self-replicating malware can spread exponentially fast. Malware can spread in the form of a virus or a worm. **Viruses** are malware that require some form of user interaction to infect the user's device. The classic example is an e-mail attachment containing malicious executable code. If a user receives and opens such an attachment, the user inadvertently runs the malware on the device. Typically, such e-mail viruses are self-replicating: once executed, the virus may send an identical message with an identical malicious attachment to, for example, every recipient in the user's address book. **Worms** are malware that can enter a device without any explicit user interaction. For example, a user may be running a vulnerable network application to which an attacker can send malware. In some cases, without any user intervention, the application may accept the malware from the Internet and run it, creating a worm. The worm in the newly infected device then scans the Internet, searching for other hosts running the same vulnerable network application. When it finds other vulnerable hosts, it sends a copy of itself to those hosts. Today, malware, is pervasive and costly to defend against. As you work through this textbook, we encourage you to think about the following question: What can computer network designers do to defend Internet-attached devices from malware attacks?

The Bad Guys Can Attack Servers and Network Infrastructure

Another broad class of security threats are known as **denial-of-service (DoS) attacks**. As the name suggests, a DoS attack renders a network, host, or other piece of infrastructure unusable by legitimate users. Web servers, e-mail servers, DNS servers (discussed in Chapter 2), and institutional networks can all be subject to DoS attacks. Internet DoS attacks are extremely common, with thousands of DoS attacks occurring every year [Moore 2001]. The site Digital Attack Map allows use to visualize the top daily DoS attacks worldwide [DAM 2016]. Most Internet DoS attacks fall into one of three categories:

- *Vulnerability attack.* This involves sending a few well-crafted messages to a vulnerable application or operating system running on a targeted host. If the right sequence of packets is sent to a vulnerable application or operating system, the service can stop or, worse, the host can crash.
- *Bandwidth flooding.* The attacker sends a deluge of packets to the targeted host—so many packets that the target's access link becomes clogged, preventing legitimate packets from reaching the server.

- *Connection flooding.* The attacker establishes a large number of half-open or fully open TCP connections (TCP connections are discussed in Chapter 3) at the target host. The host can become so bogged down with these bogus connections that it stops accepting legitimate connections.

Let's now explore the bandwidth-flooding attack in more detail. Recalling our delay and loss analysis discussion in Section 1.4.2, it's evident that if the server has an access rate of R bps, then the attacker will need to send traffic at a rate of approximately R bps to cause damage. If R is very large, a single attack source may not be able to generate enough traffic to harm the server. Furthermore, if all the traffic emanates from a single source, an upstream router may be able to detect the attack and block all traffic from that source before the traffic gets near the server. In a **distributed DoS (DDoS)** attack, illustrated in Figure 1.25, the attacker controls multiple sources and has each source blast traffic at the target. With this approach, the aggregate traffic rate across all the controlled sources needs to be approximately R to cripple the service. DDoS attacks leveraging botnets with thousands of comprised hosts are a common occurrence today [DAM 2016]. DDos attacks are much harder to detect and defend against than a DoS attack from a single host.

We encourage you to consider the following question as you work your way through this book: What can computer network designers do to defend against DoS attacks? We will see that different defenses are needed for the three types of DoS attacks.

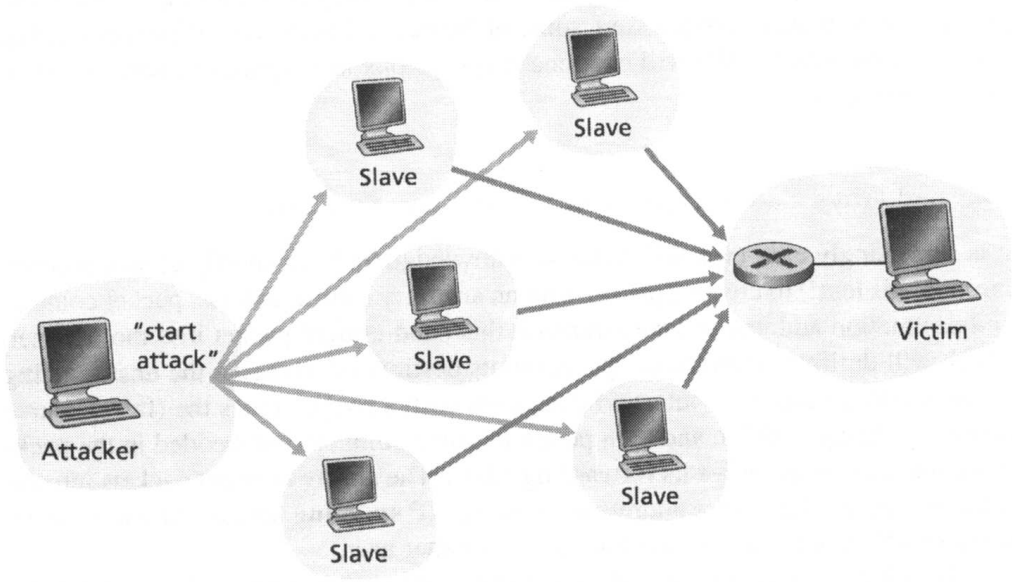


Figure 1.25 ♦ A distributed denial-of-service attack

The Bad Guys Can Sniff Packets

Many users today access the Internet via wireless devices, such as WiFi-connected laptops or handheld devices with cellular Internet connections (covered in Chapter 7). While ubiquitous Internet access is extremely convenient and enables marvelous new applications for mobile users, it also creates a major security vulnerability—by placing a passive receiver in the vicinity of the wireless transmitter, that receiver can obtain a copy of every packet that is transmitted! These packets can contain all kinds of sensitive information, including passwords, social security numbers, trade secrets, and private personal messages. A passive receiver that records a copy of every packet that flies by is called a **packet sniffer**.

Sniffers can be deployed in wired environments as well. In wired broadcast environments, as in many Ethernet LANs, a packet sniffer can obtain copies of broadcast packets sent over the LAN. As described in Section 1.2, cable access technologies also broadcast packets and are thus vulnerable to sniffing. Furthermore, a bad guy who gains access to an institution's access router or access link to the Internet may be able to plant a sniffer that makes a copy of every packet going to/from the organization. Sniffed packets can then be analyzed offline for sensitive information.

Packet-sniffing software is freely available at various Web sites and as commercial products. Professors teaching a networking course have been known to assign lab exercises that involve writing a packet-sniffing and application-layer data reconstruction program. Indeed, the Wireshark [Wireshark 2016] labs associated with this text (see the introductory Wireshark lab at the end of this chapter) use exactly such a packet sniffer!

Because packet sniffers are passive—that is, they do not inject packets into the channel—they are difficult to detect. So, when we send packets into a wireless channel, we must accept the possibility that some bad guy may be recording copies of our packets. As you may have guessed, some of the best defenses against packet sniffing involve cryptography. We will examine cryptography as it applies to network security in Chapter 8.

The Bad Guys Can Masquerade as Someone You Trust

It is surprisingly easy (*you* will have the knowledge to do so shortly as you proceed through this text!) to create a packet with an arbitrary source address, packet content, and destination address and then transmit this hand-crafted packet into the Internet, which will dutifully forward the packet to its destination. Imagine the unsuspecting receiver (say an Internet router) who receives such a packet, takes the (false) source address as being truthful, and then performs some command embedded in the packet's contents (say modifies its forwarding table). The ability to inject packets into the Internet with a false source address is known as **IP spoofing**, and is but one of many ways in which one user can masquerade as another user.

To solve this problem, we will need *end-point authentication,* that is, a mechanism that will allow us to determine with certainty if a message originates from

```
-----BEGIN PGP MESSAGE-----
Version: PGP for Personal Privacy 5.0
u2R4d+/jKmn8Bc5+hgDsqAewsDfrGdszX68liKm5F6Gc4sDfcXyt
RfdS10juHgbcfDssWe7/K=lKhnMikLo0+1/BvcX4t==Ujk9PbcD4
Thdf2awQfgHbnmKlok8iy6gThlp
-----END PGP MESSAGE
```

Figure 8.23 ♦ A secret PGP message

a *web of trust.* Alice herself can certify any key/username pair when she believes the pair really belong together. In addition, PGP permits Alice to say that she trusts another user to vouch for the authenticity of more keys. Some PGP users sign each other's keys by holding key-signing parties. Users physically gather, exchange public keys, and certify each other's keys by signing them with their private keys.

8.6 Securing TCP Connections: SSL

In the previous section, we saw how cryptographic techniques can provide confidentiality, data integrity, and end-point authentication to a specific application, namely, e-mail. In this section, we'll drop down a layer in the protocol stack and examine how cryptography can enhance TCP with security services, including confidentiality, data integrity, and end-point authentication. This enhanced version of TCP is commonly known as **Secure Sockets Layer (SSL)**. A slightly modified version of SSL version 3, called **Transport Layer Security (TLS)**, has been standardized by the IETF [RFC 4346].

The SSL protocol was originally designed by Netscape, but the basic ideas behind securing TCP had predated Netscape's work (for example, see Woo [Woo 1994]). Since its inception, SSL has enjoyed broad deployment. SSL is supported by all popular Web browsers and Web servers, and it is used by Gmail and essentially all Internet commerce sites (including Amazon, eBay, and TaoBao). Hundreds of billions of dollars are spent over SSL every year. In fact, if you have ever purchased anything over the Internet with your credit card, the communication between your browser and the server for this purchase almost certainly went over SSL. (You can identify that SSL is being used by your browser when the URL begins with https: rather than http.)

To understand the need for SSL, let's walk through a typical Internet commerce scenario. Bob is surfing the Web and arrives at the Alice Incorporated site, which is selling perfume. The Alice Incorporated site displays a form in which Bob is supposed to enter the type of perfume and quantity desired, his address, and his payment card number. Bob enters this information, clicks on Submit, and expects to receive (via ordinary postal mail) the purchased perfumes; he also expects to receive

a charge for his order in his next payment card statement. This all sounds good, but if no security measures are taken, Bob could be in for a few surprises.

- If no confidentiality (encryption) is used, an intruder could intercept Bob's order and obtain his payment card information. The intruder could then make purchases at Bob's expense.
- If no data integrity is used, an intruder could modify Bob's order, having him purchase ten times more bottles of perfume than desired.
- Finally, if no server authentication is used, a server could display Alice Incorporated's famous logo when in actuality the site maintained by Trudy, who is masquerading as Alice Incorporated. After receiving Bob's order, Trudy could take Bob's money and run. Or Trudy could carry out an identity theft by collecting Bob's name, address, and credit card number.

SSL addresses these issues by enhancing TCP with confidentiality, data integrity, server authentication, and client authentication.

SSL is often used to provide security to transactions that take place over HTTP. However, because SSL secures TCP, it can be employed by any application that runs over TCP. SSL provides a simple Application Programmer Interface (API) with sockets, which is similar and analogous to TCP's API. When an application wants to employ SSL, the application includes SSL classes/libraries. As shown in Figure 8.24, although SSL technically resides in the application layer, from the developer's perspective it is a transport protocol that provides TCP's services enhanced with security services.

8.6.1 The Big Picture

We begin by describing a simplified version of SSL, one that will allow us to get a big-picture understanding of the *why* and *how* of SSL. We will refer to this simplified

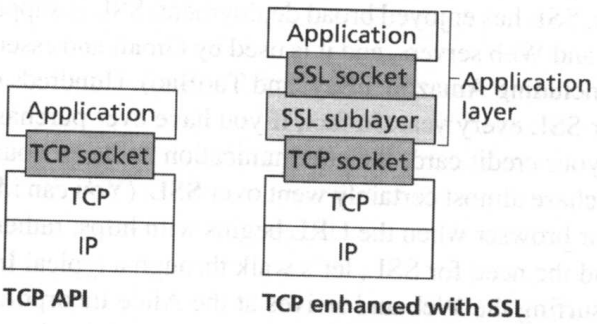


Figure 8.24 ♦ Although SSL technically resides in the application layer, from the developer's perspective it is a transport-layer protocol

version of SSL as "almost-SSL." After describing almost-SSL, in the next subsection we'll then describe the real SSL, filling in the details. Almost-SSL (and SSL) has three phases: *handshake*, *key derivation*, and *data transfer*. We now describe these three phases for a communication session between a client (Bob) and a server (Alice), with Alice having a private/public key pair and a certificate that binds her identity to her public key.

Handshake

During the handshake phase, Bob needs to (a) establish a TCP connection with Alice, (b) verify that Alice is *really* Alice, and (c) send Alice a master secret key, which will be used by both Alice and Bob to generate all the symmetric keys they need for the SSL session. These three steps are shown in Figure 8.25. Note that once the TCP connection is established, Bob sends Alice a hello message. Alice then responds with her certificate, which contains her public key. As discussed in Section 8.3, because the certificate has been certified by a CA, Bob knows for sure that the public key in the certificate belongs to Alice. Bob then generates a Master Secret (MS) (which will only be used for this SSL session), encrypts the MS with Alice's public key to create the Encrypted Master Secret (EMS), and sends the EMS to Alice. Alice decrypts the EMS with her private key to get the MS. After this phase, both Bob and Alice (and no one else) know the master secret for this SSL session.

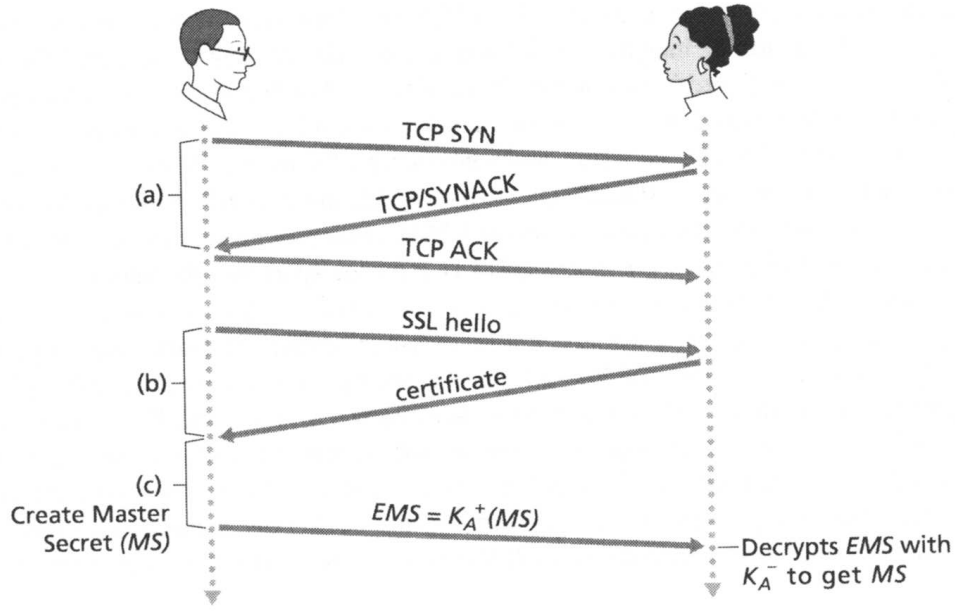


Figure 8.25 • The almost-SSL handshake, beginning with a TCP connection

Key Derivation

In principle, the MS, now shared by Bob and Alice, could be used as the symmetric session key for all subsequent encryption and data integrity checking. It is, however, generally considered safer for Alice and Bob to each use different cryptographic keys, and also to use different keys for encryption and integrity checking. Thus, both Alice and Bob use the MS to generate four keys:

- E_B = session encryption key for data sent from Bob to Alice
- M_B = session MAC key for data sent from Bob to Alice
- E_A = session encryption key for data sent from Alice to Bob
- M_A = session MAC key for data sent from Alice to Bob

Alice and Bob each generate the four keys from the MS. This could be done by simply slicing the MS into four keys. (But in *real* SSL it is a little more complicated, as we'll see.) At the end of the key derivation phase, both Alice and Bob have all four keys. The two encryption keys will be used to encrypt data; the two MAC keys will be used to verify the integrity of the data.

Data Transfer

Now that Alice and Bob share the same four session keys (E_B, M_B, E_A, and M_A), they can start to send secured data to each other over the TCP connection. Since TCP is a byte-stream protocol, a natural approach would be for SSL to encrypt application data on the fly and then pass the encrypted data on the fly to TCP. But if we were to do this, where would we put the MAC for the integrity check? We certainly do not want to wait until the end of the TCP session to verify the integrity of all of Bob's data that was sent over the entire session! To address this issue, SSL breaks the data stream into records, appends a MAC to each record for integrity checking, and then encrypts the record+MAC. To create the MAC, Bob inputs the record data along with the key M_B into a hash function, as discussed in Section 8.3. To encrypt the package record+MAC, Bob uses his session encryption key E_B. This encrypted package is then passed to TCP for transport over the Internet.

Although this approach goes a long way, it still isn't bullet-proof when it comes to providing data integrity for the entire message stream. In particular, suppose Trudy is a woman-in-the-middle and has the ability to insert, delete, and replace segments in the stream of TCP segments sent between Alice and Bob. Trudy, for example, could capture two segments sent by Bob, reverse the order of the segments, adjust the TCP sequence numbers (which are not encrypted), and then send the two reverse-ordered segments to Alice. Assuming that each TCP segment encapsulates exactly one record, let's now take a look at how Alice would process these segments.

1. TCP running in Alice would think everything is fine and pass the two records to the SSL sublayer.
2. SSL in Alice would decrypt the two records.

3. SSL in Alice would use the MAC in each record to verify the data integrity of the two records.
4. SSL would then pass the decrypted byte streams of the two records to the application layer; but the complete byte stream received by Alice would not be in the correct order due to reversal of the records!

You are encouraged to walk through similar scenarios for when Trudy removes segments or when Trudy replays segments.

The solution to this problem, as you probably guessed, is to use sequence numbers. SSL does this as follows. Bob maintains a sequence number counter, which begins at zero and is incremented for each SSL record he sends. Bob doesn't actually include a sequence number in the record itself, but when he calculates the MAC, he includes the sequence number in the MAC calculation. Thus, the MAC is now a hash of the data plus the MAC key M_B *plus the current sequence number*. Alice tracks Bob's sequence numbers, allowing her to verify the data integrity of a record by including the appropriate sequence number in the MAC calculation. This use of SSL sequence numbers prevents Trudy from carrying out a woman-in-the-middle attack, such as reordering or replaying segments. (Why?)

SSL Record

The SSL record (as well as the almost-SSL record) is shown in Figure 8.26. The record consists of a type field, version field, length field, data field, and MAC field. Note that the first three fields are not encrypted. The type field indicates whether the record is a handshake message or a message that contains application data. It is also used to close the SSL connection, as discussed below. SSL at the receiving end uses the length field to extract the SSL records out of the incoming TCP byte stream. The version field is self-explanatory.

8.6.2 A More Complete Picture

The previous subsection covered the almost-SSL protocol; it served to give us a basic understanding of the why and how of SSL. Now that we have a basic understanding of SSL, we can dig a little deeper and examine the essentials of the actual SSL protocol. In parallel to reading this description of the SSL protocol, you are encouraged to complete the Wireshark SSL lab, available at the textbook's Web site.

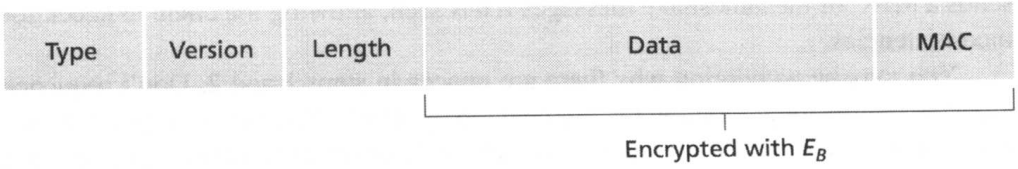


Figure 8.26 ♦ Record format for SSL

SSL Handshake

SSL does not mandate that Alice and Bob use a specific symmetric key algorithm, a specific public-key algorithm, or a specific MAC. Instead, SSL allows Alice and Bob to agree on the cryptographic algorithms at the beginning of the SSL session, during the handshake phase. Additionally, during the handshake phase, Alice and Bob send nonces to each other, which are used in the creation of the session keys (E_B, M_B, E_A, and M_A). The steps of the real SSL handshake are as follows:

1. The client sends a list of cryptographic algorithms it supports, along with a client nonce.
2. From the list, the server chooses a symmetric algorithm (for example, AES), a public key algorithm (for example, RSA with a specific key length), and a MAC algorithm. It sends back to the client its choices, as well as a certificate and a server nonce.
3. The client verifies the certificate, extracts the server's public key, generates a Pre-Master Secret (PMS), encrypts the PMS with the server's public key, and sends the encrypted PMS to the server.
4. Using the same key derivation function (as specified by the SSL standard), the client and server independently compute the Master Secret (MS) from the PMS and nonces. The MS is then sliced up to generate the two encryption and two MAC keys. Furthermore, when the chosen symmetric cipher employs CBC (such as 3DES or AES), then two Initialization Vectors (IVs)—one for each side of the connection—are also obtained from the MS. Henceforth, all messages sent between client and server are encrypted and authenticated (with the MAC).
5. The client sends a MAC of all the handshake messages.
6. The server sends a MAC of all the handshake messages.

The last two steps protect the handshake from tampering. To see this, observe that in step 1, the client typically offers a list of algorithms—some strong, some weak. This list of algorithms is sent in cleartext, since the encryption algorithms and keys have not yet been agreed upon. Trudy, as a woman-in-the-middle, could delete the stronger algorithms from the list, forcing the client to select a weak algorithm. To prevent such a tampering attack, in step 5 the client sends a MAC of the concatenation of all the handshake messages it sent and received. The server can compare this MAC with the MAC of the handshake messages it received and sent. If there is an inconsistency, the server can terminate the connection. Similarly, the server sends a MAC of the handshake messages it has seen, allowing the client to check for inconsistencies.

You may be wondering why there are nonces in steps 1 and 2. Don't sequence numbers suffice for preventing the segment replay attack? The answer is yes, but they don't alone prevent the "connection replay attack." Consider the following connection

replay attack. Suppose Trudy sniffs all messages between Alice and Bob. The next day, Trudy masquerades as Bob and sends to Alice exactly the same sequence of messages that Bob sent to Alice on the previous day. If Alice doesn't use nonces, she will respond with exactly the same sequence of messages she sent the previous day. Alice will not suspect any funny business, as each message she receives will pass the integrity check. If Alice is an e-commerce server, she will think that Bob is placing a second order (for exactly the same thing). On the other hand, by including a nonce in the protocol, Alice will send different nonces for each TCP session, causing the encryption keys to be different on the two days. Therefore, when Alice receives played-back SSL records from Trudy, the records will fail the integrity checks, and the bogus e-commerce transaction will not succeed. In summary, in SSL, nonces are used to defend against the "connection replay attack" and sequence numbers are used to defend against replaying individual packets during an ongoing session.

Connection Closure

At some point, either Bob or Alice will want to end the SSL session. One approach would be to let Bob end the SSL session by simply terminating the underlying TCP connection—that is, by having Bob send a TCP FIN segment to Alice. But such a naive design sets the stage for the *truncation attack* whereby Trudy once again gets in the middle of an ongoing SSL session and ends the session early with a TCP FIN. If Trudy were to do this, Alice would think she received all of Bob's data when actuality she only received a portion of it. The solution to this problem is to indicate in the type field whether the record serves to terminate the SSL session. (Although the SSL type is sent in the clear, it is authenticated at the receiver using the record's MAC.) By including such a field, if Alice were to receive a TCP FIN before receiving a closure SSL record, she would know that something funny was going on.

This completes our introduction to SSL. We've seen that it uses many of the cryptography principles discussed in Sections 8.2 and 8.3. Readers who want to explore SSL on yet a deeper level can read Rescorla's highly readable book on SSL [Rescorla 2001].

8.7 Network-Layer Security: IPsec and Virtual Private Networks

The IP security protocol, more commonly known as **IPsec**, provides security at the network layer. IPsec secures IP datagrams between any two network-layer entities, including hosts and routers. As we will soon describe, many institutions (corporations, government branches, non-profit organizations, and so on) use IPsec to create **virtual private networks (VPNs)** that run over the public Internet.

Before getting into the specifics of IPsec, let's step back and consider what it means to provide confidentiality at the network layer. With network-layer confidentiality between a pair of network entities (for example, between two routers, between two hosts, or between a router and a host), the sending entity encrypts the payloads of all the datagrams it sends to the receiving entity. The encrypted payload could be a TCP segment, a UDP segment, an ICMP message, and so on. If such a network-layer service were in place, all data sent from one entity to the other—including e-mail, Web pages, TCP handshake messages, and management messages (such as ICMP and SNMP)—would be hidden from any third party that might be sniffing the network. For this reason, network-layer security is said to provide "blanket coverage."

In addition to confidentiality, a network-layer security protocol could potentially provide other security services. For example, it could provide source authentication, so that the receiving entity can verify the source of the secured datagram. A network-layer security protocol could provide data integrity, so that the receiving entity can check for any tampering of the datagram that may have occurred while the datagram was in transit. A network-layer security service could also provide replay-attack prevention, meaning that Bob could detect any duplicate datagrams that an attacker might insert. We will soon see that IPsec indeed provides mechanisms for all these security services, that is, for confidentiality, source authentication, data integrity, and replay-attack prevention.

8.7.1 IPsec and Virtual Private Networks (VPNs)

An institution that extends over multiple geographical regions often desires its own IP network, so that its hosts and servers can send data to each other in a secure and confidential manner. To achieve this goal, the institution could actually deploy a stand-alone physical network—including routers, links, and a DNS infrastructure—that is completely separate from the public Internet. Such a disjoint network, dedicated to a particular institution, is called a **private network**. Not surprisingly, a private network can be very costly, as the institution needs to purchase, install, and maintain its own physical network infrastructure.

Instead of deploying and maintaining a private network, many institutions today create VPNs over the existing public Internet. With a VPN, the institution's inter-office traffic is sent over the public Internet rather than over a physically independent network. But to provide confidentiality, the inter-office traffic is encrypted before it enters the public Internet. A simple example of a VPN is shown in Figure 8.27. Here the institution consists of a headquarters, a branch office, and traveling salespersons that typically access the Internet from their hotel rooms. (There is only one salesperson shown in the figure.) In this VPN, whenever two hosts within headquarters send IP datagrams to each other or whenever two hosts within the branch office want to communicate, they use good-old vanilla IPv4 (that is, without IPsec services). However, when two of the institution's hosts

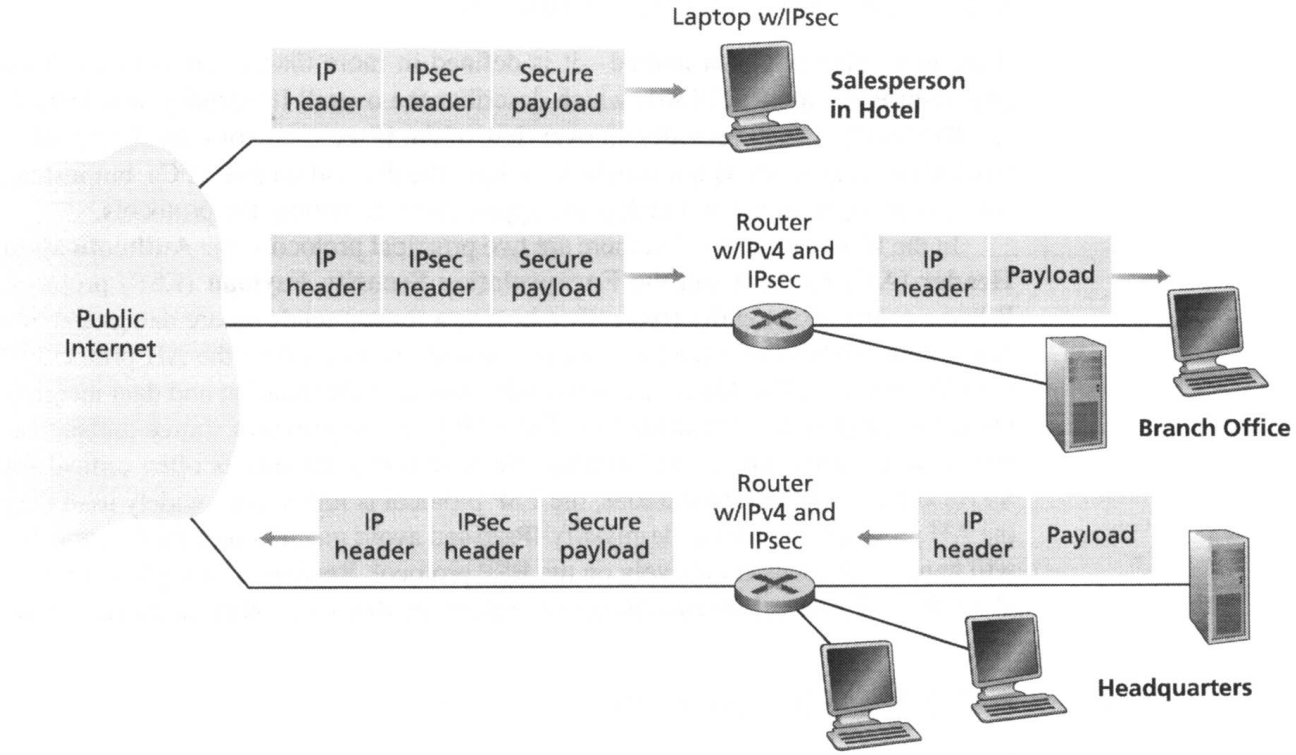


Figure 8.27 ♦ Virtual private network (VPN)

communicate over a path that traverses the public Internet, the traffic is encrypted before it enters the Internet.

To get a feel for how a VPN works, let's walk through a simple example in the context of Figure 8.27. When a host in headquarters sends an IP datagram to a salesperson in a hotel, the gateway router in headquarters converts the vanilla IPv4 datagram into an IPsec datagram and then forwards this IPsec datagram into the Internet. This IPsec datagram actually has a traditional IPv4 header, so that the routers in the public Internet process the datagram as if it were an ordinary IPv4 datagram—to them, the datagram is a perfectly ordinary datagram. But, as shown Figure 8.27, the payload of the IPsec datagram includes an IPsec header, which is used for IPsec processing; furthermore, the payload of the IPsec datagram is encrypted. When the IPsec datagram arrives at the salesperson's laptop, the OS in the laptop decrypts the payload (and provides other security services, such as verifying data integrity) and passes the unencrypted payload to the upper-layer protocol (for example, to TCP or UDP).

We have just given a high-level overview of how an institution can employ IPsec to create a VPN. To see the forest through the trees, we have brushed aside many important details. Let's now take a closer look.

8.7.2 The AH and ESP Protocols

IPsec is a rather complex animal—it is defined in more than a dozen RFCs. Two important RFCs are RFC 4301, which describes the overall IP security architecture, and RFC 6071, which provides an overview of the IPsec protocol suite. Our goal in this textbook, as usual, is not simply to re-hash the dry and arcane RFCs, but instead take a more operational and pedagogic approach to describing the protocols.

In the IPsec protocol suite, there are two principal protocols: the **Authentication Header (AH)** protocol and the **Encapsulation Security Payload (ESP)** protocol. When a source IPsec entity (typically a host or a router) sends secure datagrams to a destination entity (also a host or a router), it does so with either the AH protocol or the ESP protocol. The AH protocol provides source authentication and data integrity but *does not* provide confidentiality. The ESP protocol provides source authentication, data integrity, *and* confidentiality. Because confidentiality is often critical for VPNs and other IPsec applications, the ESP protocol is much more widely used than the AH protocol. In order to de-mystify IPsec and avoid much of its complication, we will henceforth focus exclusively on the ESP protocol. Readers wanting to learn also about the AH protocol are encouraged to explore the RFCs and other online resources.

8.7.3 Security Associations

IPsec datagrams are sent between pairs of network entities, such as between two hosts, between two routers, or between a host and router. Before sending IPsec datagrams from source entity to destination entity, the source and destination entities create a network-layer logical connection. This logical connection is called a **security association (SA)**. An SA is a simplex logical connection; that is, it is unidirectional from source to destination. If both entities want to send secure datagrams to each other, then two SAs (that is, two logical connections) need to be established, one in each direction.

For example, consider once again the institutional VPN in Figure 8.27. This institution consists of a headquarters office, a branch office and, say, n traveling salespersons. For the sake of example, let's suppose that there is bi-directional IPsec traffic between headquarters and the branch office and bi-directional IPsec traffic between headquarters and the salespersons. In this VPN, how many SAs are there? To answer this question, note that there are two SAs between the headquarters gateway router and the branch-office gateway router (one in each direction); for each salesperson's laptop, there are two SAs between the headquarters gateway router and the laptop (again, one in each direction). So, in total, there are $(2 + 2n)$ SAs. *Keep in mind, however, that not all traffic sent into the Internet by the gateway routers or by the laptops will be IPsec secured.* For example, a host in headquarters may want to access a Web server (such as Amazon or Google) in the public Internet. Thus, the gateway router (and the laptops) will emit into the Internet both vanilla IPv4 datagrams and secured IPsec datagrams.

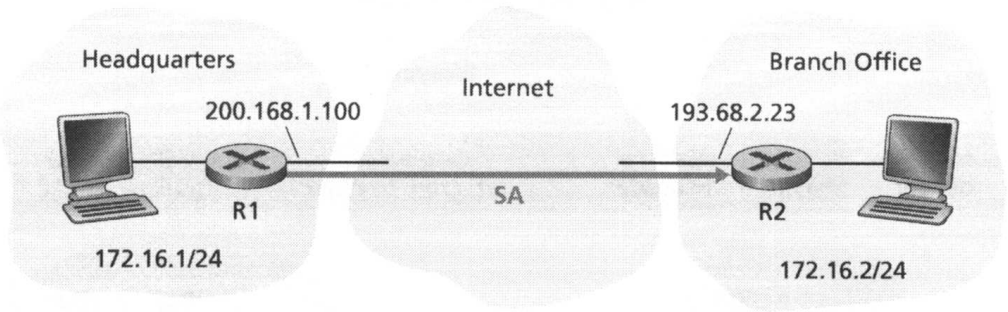


Figure 8.28 • Security association (SA) from R1 to R2

Let's now take a look "inside" an SA. To make the discussion tangible and concrete, let's do this in the context of an SA from router R1 to router R2 in Figure 8.28. (You can think of Router R1 as the headquarters gateway router and Router R2 as the branch office gateway router from Figure 8.27.) Router R1 will maintain state information about this SA, which will include:

- A 32-bit identifier for the SA, called the **Security Parameter Index (SPI)**
- The origin interface of the SA (in this case 200.168.1.100) and the destination interface of the SA (in this case 193.68.2.23)
- The type of encryption to be used (for example, 3DES with CBC)
- The encryption key
- The type of integrity check (for example, HMAC with MD5)
- The authentication key

Whenever router R1 needs to construct an IPsec datagram for forwarding over this SA, it accesses this state information to determine how it should authenticate and encrypt the datagram. Similarly, router R2 will maintain the same state information for this SA and will use this information to authenticate and decrypt any IPsec datagram that arrives from the SA.

An IPsec entity (router or host) often maintains state information for many SAs. For example, in the VPN example in Figure 8.27 with n salespersons, the headquarters gateway router maintains state information for $(2 + 2n)$ SAs. An IPsec entity stores the state information for all of its SAs in its **Security Association Database (SAD)**, which is a data structure in the entity's OS kernel.

8.7.4 The IPsec Datagram

Having now described SAs, we can now describe the actual IPsec datagram. IPsec has two different packet forms, one for the so-called **tunnel mode** and the other for the so-called **transport mode**. The tunnel mode, being more appropriate for VPNs,

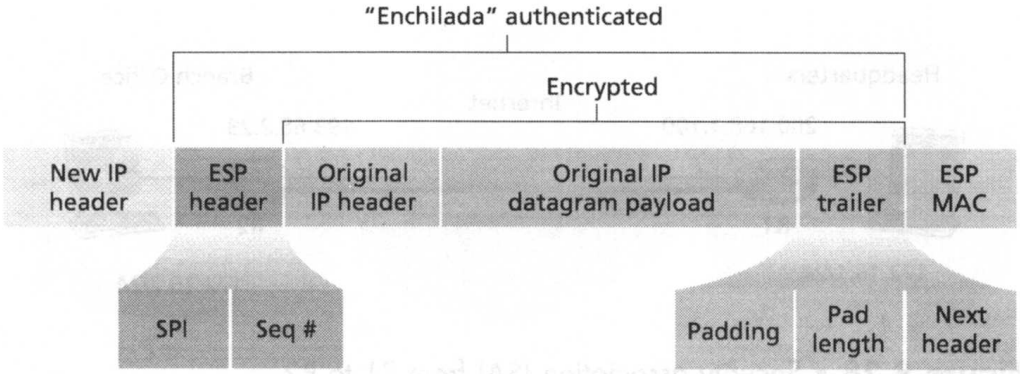


Figure 8.29 ♦ IPsec datagram format

is more widely deployed than the transport mode. In order to further de-mystify IPsec and avoid much of its complication, we henceforth focus exclusively on the tunnel mode. Once you have a solid grip on the tunnel mode, you should be able to easily learn about the transport mode on your own.

The packet format of the IPsec datagram is shown in Figure 8.29. You might think that packet formats are boring and insipid, but we will soon see that the IPsec datagram actually looks and tastes like a popular Tex-Mex delicacy! Let's examine the IPsec fields in the context of Figure 8.28. Suppose router R1 receives an ordinary IPv4 datagram from host 172.16.1.17 (in the headquarters network) which is destined to host 172.16.2.48 (in the branch-office network). Router R1 uses the following recipe to convert this "original IPv4 datagram" into an IPsec datagram:

- Appends to the back of the original IPv4 datagram (which includes the original header fields!) an "ESP trailer" field
- Encrypts the result using the algorithm and key specified by the SA
- Appends to the front of this encrypted quantity a field called "ESP header"; the resulting package is called the "enchilada"
- Creates an authentication MAC over the *whole enchilada* using the algorithm and key specified in the SA
- Appends the MAC to the back of the enchilada forming the *payload*
- Finally, creates a brand new IP header with all the classic IPv4 header fields (together normally 20 bytes long), which it appends before the payload

Note that the resulting IPsec datagram is a bona fide IPv4 datagram, with the traditional IPv4 header fields followed by a payload. But in this case, the payload contains an ESP header, the original IP datagram, an ESP trailer, and an ESP authentication field (with the original datagram and ESP trailer encrypted). The original IP datagram has 172.16.1.17 for the source IP address and 172.16.2.48 for the

destination IP address. Because the IPsec datagram includes the original IP datagram, these addresses are included (and encrypted) as part of the payload of the IPsec packet. But what about the source and destination IP addresses that are in the new IP header, that is, in the left-most header of the IPsec datagram? As you might expect, they are set to the source and destination router interfaces at the two ends of the tunnels, namely, 200.168.1.100 and 193.68.2.23. Also, the protocol number in this new IPv4 header field is not set to that of TCP, UDP, or SMTP, but instead to 50, designating that this is an IPsec datagram using the ESP protocol.

After R1 sends the IPsec datagram into the public Internet, it will pass through many routers before reaching R2. Each of these routers will process the datagram as if it were an ordinary datagram—they are completely oblivious to the fact that the datagram is carrying IPsec-encrypted data. For these public Internet routers, because the destination IP address in the outer header is R2, the ultimate destination of the datagram is R2.

Having walked through an example of how an IPsec datagram is constructed, let's now take a closer look at the ingredients in the enchilada. We see in Figure 8.29 that the ESP trailer consists of three fields: padding; pad length; and next header. Recall that block ciphers require the message to be encrypted to be an integer multiple of the block length. Padding (consisting of meaningless bytes) is used so that when added to the original datagram (along with the pad length and next header fields), the resulting "message" is an integer number of blocks. The pad-length field indicates to the receiving entity how much padding was inserted (and thus needs to be removed). The next header identifies the type (e.g., UDP) of data contained in the payload-data field. The payload data (typically the original IP datagram) and the ESP trailer are concatenated and then encrypted.

Appended to the front of this encrypted unit is the ESP header, which is sent in the clear and consists of two fields: the SPI and the sequence number field. The SPI indicates to the receiving entity the SA to which the datagram belongs; the receiving entity can then index its SAD with the SPI to determine the appropriate authentication/decryption algorithms and keys. The sequence number field is used to defend against replay attacks.

The sending entity also appends an authentication MAC. As stated earlier, the sending entity calculates a MAC over the whole enchilada (consisting of the ESP header, the original IP datagram, and the ESP trailer—with the datagram and trailer being encrypted). Recall that to calculate a MAC, the sender appends a secret MAC key to the enchilada and then calculates a fixed-length hash of the result.

When R2 receives the IPsec datagram, R2 observes that the destination IP address of the datagram is R2 itself. R2 therefore processes the datagram. Because the protocol field (in the left-most IP header) is 50, R2 sees that it should apply IPsec ESP processing to the datagram. First, peering into the enchilada, R2 uses the SPI to determine to which SA the datagram belongs. Second, it calculates the MAC of the enchilada and verifies that the MAC is consistent with the value in the ESP MAC field. If it is, it knows that the enchilada comes from R1 and has not been tampered with. Third, it checks the sequence-number field to verify that the datagram is

fresh (and not a replayed datagram). Fourth, it decrypts the encrypted unit using the decryption algorithm and key associated with the SA. Fifth, it removes padding and extracts the original, vanilla IP datagram. And finally, sixth, it forwards the original datagram into the branch office network toward its ultimate destination. Whew, what a complicated recipe, huh? Well no one ever said that preparing and unraveling an enchilada was easy!

There is actually another important subtlety that needs to be addressed. It centers on the following question: When R1 receives an (unsecured) datagram from a host in the headquarters network, and that datagram is destined to some destination IP address outside of headquarters, how does R1 know whether it should be converted to an IPsec datagram? And if it is to be processed by IPsec, how does R1 know which SA (of many SAs in its SAD) should be used to construct the IPsec datagram? The problem is solved as follows. Along with a SAD, the IPsec entity also maintains another data structure called the **Security Policy Database (SPD)**. The SPD indicates what types of datagrams (as a function of source IP address, destination IP address, and protocol type) are to be IPsec processed; and for those that are to be IPsec processed, which SA should be used. In a sense, the information in a SPD indicates "what" to do with an arriving datagram; the information in the SAD indicates "how" to do it.

Summary of IPsec Services

So what services does IPsec provide, exactly? Let us examine these services from the perspective of an attacker, say Trudy, who is a woman-in-the-middle, sitting somewhere on the path between R1 and R2 in Figure 8.28. Assume throughout this discussion that Trudy does not know the authentication and encryption keys used by the SA. What can and cannot Trudy do? First, Trudy cannot see the original datagram. If fact, not only is the data in the original datagram hidden from Trudy, but so is the protocol number, the source IP address, and the destination IP address. For datagrams sent over the SA, Trudy only knows that the datagram originated from some host in 172.16.1.0/24 and is destined to some host in 172.16.2.0/24. She does not know if it is carrying TCP, UDP, or ICMP data; she does not know if it is carrying HTTP, SMTP, or some other type of application data. This confidentiality thus goes a lot farther than SSL. Second, suppose Trudy tries to tamper with a datagram in the SA by flipping some of its bits. When this tampered datagram arrives at R2, it will fail the integrity check (using the MAC), thwarting Trudy's vicious attempts once again. Third, suppose Trudy tries to masquerade as R1, creating a IPsec datagram with source 200.168.1.100 and destination 193.68.2.23. Trudy's attack will be futile, as this datagram will again fail the integrity check at R2. Finally, because IPsec includes sequence numbers, Trudy will not be able create a successful replay attack. In summary, as claimed at the beginning of this section, IPsec provides—between any pair of devices that process packets through the network layer—confidentiality, source authentication, data integrity, and replay-attack prevention.

8.7.5 IKE: Key Management in IPsec

When a VPN has a small number of end points (for example, just two routers as in Figure 8.28), the network administrator can manually enter the SA information (encryption/authentication algorithms and keys, and the SPIs) into the SADs of the endpoints. Such "manual keying" is clearly impractical for a large VPN, which may consist of hundreds or even thousands of IPsec routers and hosts. Large, geographically distributed deployments require an automated mechanism for creating the SAs. IPsec does this with the Internet Key Exchange (IKE) protocol, specified in RFC 5996.

IKE has some similarities with the handshake in SSL (see Section 8.6). Each IPsec entity has a certificate, which includes the entity's public key. As with SSL, the IKE protocol has the two entities exchange certificates, negotiate authentication and encryption algorithms, and securely exchange key material for creating session keys in the IPsec SAs. Unlike SSL, IKE employs two phases to carry out these tasks.

Let's investigate these two phases in the context of two routers, R1 and R2, in Figure 8.28. The first phase consists of two exchanges of message pairs between R1 and R2:

- During the first exchange of messages, the two sides use Diffie-Hellman (see Homework Problems) to create a bi-directional **IKE SA** between the routers. To keep us all confused, this bi-directional IKE SA is entirely different from the IPsec SAs discussed in Sections 8.6.3 and 8.6.4. The IKE SA provides an authenticated and encrypted channel between the two routers. During this first message-pair exchange, keys are established for encryption and authentication for the IKE SA. Also established is a master secret that will be used to compute IPSec SA keys later in phase 2. Observe that during this first step, RSA public and private keys are not used. In particular, neither R1 nor R2 reveals its identity by signing a message with its private key.
- During the second exchange of messages, both sides reveal their identity to each other by signing their messages. However, the identities are not revealed to a passive sniffer, since the messages are sent over the secured IKE SA channel. Also during this phase, the two sides negotiate the IPsec encryption and authentication algorithms to be employed by the IPsec SAs.

In phase 2 of IKE, the two sides create an SA in each direction. At the end of phase 2, the encryption and authentication session keys are established on both sides for the two SAs. The two sides can then use the SAs to send secured datagrams, as described in Sections 8.7.3 and 8.7.4. The primary motivation for having two phases in IKE is computational cost—since the second phase doesn't involve any public-key cryptography, IKE can generate a large number of SAs between the two IPsec entities with relatively little computational cost.

8.9 Operational Security: Firewalls and Intrusion Detection Systems

We've seen throughout this chapter that the Internet is not a very safe place—bad guys are out there, wreaking all sorts of havoc. Given the hostile nature of the Internet, let's now consider an organization's network and the network administrator who administers it. From a network administrator's point of view, the world divides quite neatly into two camps—the good guys (who belong to the organization's network, and who should be able to access resources inside the organization's network in a relatively unconstrained manner) and the bad guys (everyone else, whose access to network resources must be carefully scrutinized). In many organizations, ranging from medieval castles to modern corporate office buildings, there is a single point of entry/exit where both good guys and bad guys entering and leaving the organization are security-checked. In a castle, this was done at a gate at one end of the drawbridge; in a corporate building, this is done at the security desk. In a computer network, when traffic entering/leaving a network is security-checked, logged, dropped, or forwarded, it is done by operational devices known as firewalls, intrusion detection systems (IDSs), and intrusion prevention systems (IPSs).

8.9.1 Firewalls

A **firewall** is a combination of hardware and software that isolates an organization's internal network from the Internet at large, allowing some packets to pass and blocking others. A firewall allows a network administrator to control access between the outside world and resources within the administered network by managing the traffic flow to and from these resources. A firewall has three goals:

- *All traffic from outside to inside, and vice versa, passes through the firewall.* Figure 8.33 shows a firewall, sitting squarely at the boundary between the administered network and the rest of the Internet. While large organizations may use multiple levels of firewalls or distributed firewalls [Skoudis 2006], locating a firewall at a single access point to the network, as shown in Figure 8.33, makes it easier to manage and enforce a security-access policy.
- *Only authorized traffic, as defined by the local security policy, will be allowed to pass.* With all traffic entering and leaving the institutional network passing through the firewall, the firewall can restrict access to authorized traffic.
- *The firewall itself is immune to penetration.* The firewall itself is a device connected to the network. If not designed or installed properly, it can be compromised, in which case it provides only a false sense of security (which is worse than no firewall at all!).

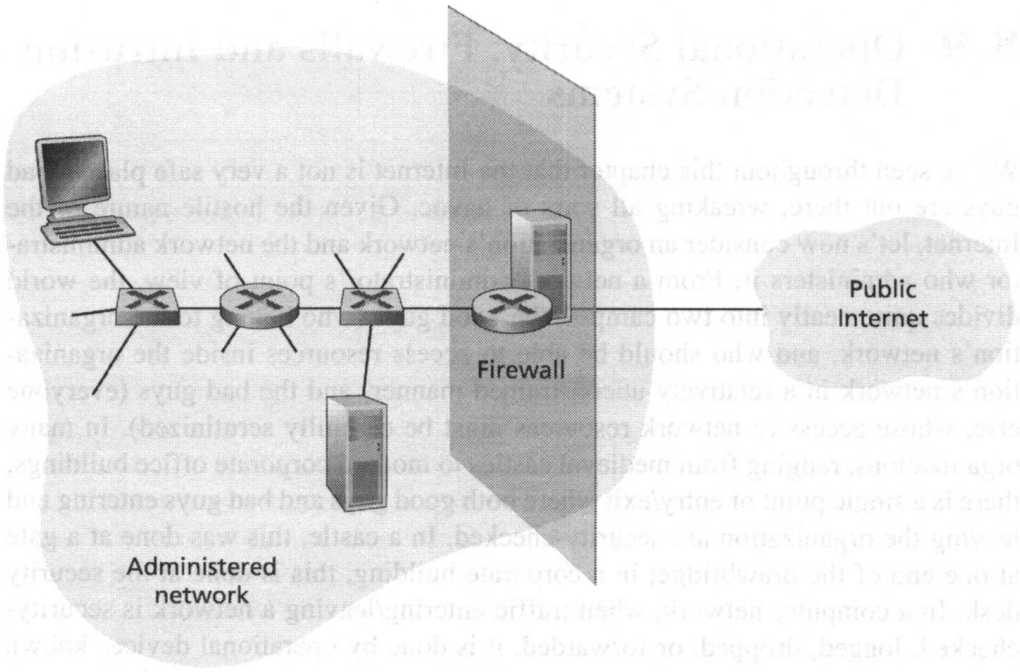


Figure 8.33 ♦ Firewall placement between the administered network and the outside world

Cisco and Check Point are two of the leading firewall vendors today. You can also easily create a firewall (packet filter) from a Linux box using iptables (public-domain software that is normally shipped with Linux). Furthermore, as discussed in Chapters 4 and 5, firewalls are now frequently implemented in routers and controlled remotely using SDNs.

Firewalls can be classified in three categories: **traditional packet filters**, **stateful filters**, and **application gateways**. We'll cover each of these in turn in the following subsections.

Traditional Packet Filters

As shown in Figure 8.33, an organization typically has a gateway router connecting its internal network to its ISP (and hence to the larger public Internet). All traffic leaving and entering the internal network passes through this router, and it is at this router where **packet filtering** occurs. A packet filter examines each datagram in isolation, determining whether the datagram should be allowed to pass or should be dropped based on administrator-specific rules. Filtering decisions are typically based on:

- IP source or destination address
- Protocol type in IP datagram field: TCP, UDP, ICMP, OSPF, and so on
- TCP or UDP source and destination port

Policy	Firewall Setting
No outside Web access.	Drop all outgoing packets to any IP address, port 80.
No incoming TCP connections, except those for organization's public Web server only.	Drop all incoming TCP SYN packets to any IP except 130.207.244.203, port 80.
Prevent Web-radios from eating up the available bandwidth.	Drop all incoming UDP packets—except DNS packets.
Prevent your network from being used for a smurf DoS attack.	Drop all ICMP ping packets going to a "broadcast" address (eg 130.207.255.255).
Prevent your network from being tracerouted.	Drop all outgoing ICMP TTL expired traffic.

Table 8.5 • Policies and corresponding filtering rules for an organization's network 130.207/16 with Web server at 130.207.244.203

- TCP flag bits: SYN, ACK, and so on
- ICMP message type
- Different rules for datagrams leaving and entering the network
- Different rules for the different router interfaces

A network administrator configures the firewall based on the policy of the organization. The policy may take user productivity and bandwidth usage into account as well as the security concerns of an organization. Table 8.5 lists a number of possible polices an organization may have, and how they would be addressed with a packet filter. For example, if the organization doesn't want any incoming TCP connections except those for its public Web server, it can block all incoming TCP SYN segments except TCP SYN segments with destination port 80 and the destination IP address corresponding to the Web server. If the organization doesn't want its users to monopolize access bandwidth with Internet radio applications, it can block all not-critical UDP traffic (since Internet radio is often sent over UDP). If the organization doesn't want its internal network to be mapped (tracerouted) by an outsider, it can block all ICMP TTL expired messages leaving the organization's network.

A filtering policy can be based on a combination of addresses and port numbers. For example, a filtering router could forward all Telnet datagrams (those with a port number of 23) except those going to and coming from a list of specific IP addresses. This policy permits Telnet connections to and from hosts on the allowed list. Unfortunately, basing the policy on external addresses provides no protection against datagrams that have had their source addresses spoofed.

Filtering can also be based on whether or not the TCP ACK bit is set. This trick is quite useful if an organization wants to let its internal clients connect to external servers but wants to prevent external clients from connecting to internal servers.

action	source address	dest address	protocol	source port	dest port	flag bit
allow	222.22/16	outside of 222.22/16	TCP	> 1023	80	any
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK
allow	222.22/16	outside of 222.22/16	UDP	> 1023	53	—
allow	outside of 222.22/16	222.22/16	UDP	53	> 1023	—
deny	all	all	all	all	all	all

Table 8.6 ♦ An access control list for a router interface

Recall from Section 3.5 that the first segment in every TCP connection has the ACK bit set to 0, whereas all the other segments in the connection have the ACK bit set to 1. Thus, if an organization wants to prevent external clients from initiating connections to internal servers, it simply filters all incoming segments with the ACK bit set to 0. This policy kills all TCP connections originating from the outside, but permits connections originating internally.

Firewall rules are implemented in routers with access control lists, with each router interface having its own list. An example of an access control list for an organization 222.22/16 is shown in Table 8.6. This access control list is for an interface that connects the router to the organization's external ISPs. Rules are applied to each datagram that passes through the interface from top to bottom. The first two rules together allow internal users to surf the Web: The first rule allows any TCP packet with destination port 80 to leave the organization's network; the second rule allows any TCP packet with source port 80 and the ACK bit set to enter the organization's network. Note that if an external source attempts to establish a TCP connection with an internal host, the connection will be blocked, even if the source or destination port is 80. The second two rules together allow DNS packets to enter and leave the organization's network. In summary, this rather restrictive access control list blocks all traffic except Web traffic initiated from within the organization and DNS traffic. [CERT Filtering 2012] provides a list of recommended port/protocol packet filterings to avoid a number of well-known security holes in existing network applications.

Stateful Packet Filters

In a traditional packet filter, filtering decisions are made on each packet in isolation. Stateful filters actually track TCP connections, and use this knowledge to make filtering decisions.

source address	dest address	source port	dest port
222.22.1.7	37.96.87.123	12699	80
222.22.93.2	199.1.205.23	37654	80
222.22.65.143	203.77.240.43	48712	80

Table 8.7 ♦ Connection table for stateful filter

To understand stateful filters, let's reexamine the access control list in Table 8.6. Although rather restrictive, the access control list in Table 8.6 nevertheless allows any packet arriving from the outside with ACK = 1 and source port 80 to get through the filter. Such packets could be used by attackers in attempts to crash internal systems with malformed packets, carry out denial-of-service attacks, or map the internal network. The naive solution is to block TCP ACK packets as well, but such an approach would prevent the organization's internal users from surfing the Web.

Stateful filters solve this problem by tracking all ongoing TCP connections in a connection table. This is possible because the firewall can observe the beginning of a new connection by observing a three-way handshake (SYN, SYNACK, and ACK); and it can observe the end of a connection when it sees a FIN packet for the connection. The firewall can also (conservatively) assume that the connection is over when it hasn't seen any activity over the connection for, say, 60 seconds. An example connection table for a firewall is shown in Table 8.7. This connection table indicates that there are currently three ongoing TCP connections, all of which have been initiated from within the organization. Additionally, the stateful filter includes a new column, "check connection," in its access control list, as shown in Table 8.8. Note that Table 8.8 is identical to the access control list in Table 8.6, except now it indicates that the connection should be checked for two of the rules.

Let's walk through some examples to see how the connection table and the extended access control list work hand-in-hand. Suppose an attacker attempts to send a malformed packet into the organization's network by sending a datagram with TCP source port 80 and with the ACK flag set. Further suppose that this packet has source port number 12543 and source IP address 150.23.23.155. When this packet reaches the firewall, the firewall checks the access control list in Table 8.7, which indicates that the connection table must also be checked before permitting this packet to enter the organization's network. The firewall duly checks the connection table, sees that this packet is not part of an ongoing TCP connection, and rejects the packet. As a second example, suppose that an internal user wants to surf an external Web site. Because this user first sends a TCP SYN segment, the user's TCP connection gets recorded in the connection table. When

action	source address	dest address	protocol	source port	dest port	flag bit	check conxion
allow	222.22/16	outside of 222.22/16	TCP	> 1023	80	any	
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK	X
allow	222.22/16	outside of 222.22/16	UDP	> 1023	53	—	
allow	outside of 222.22/16	222.22/16	UDP	53	> 1023	—	X
deny	all	all	all	all	all	all	

Table 8.8 ♦ Access control list for stateful filter

the Web server sends back packets (with the ACK bit necessarily set), the firewall checks the table and sees that a corresponding connection is in progress. The firewall will thus let these packets pass, thereby not interfering with the internal user's Web surfing activity.

Application Gateway

In the examples above, we have seen that packet-level filtering allows an organization to perform coarse-grain filtering on the basis of the contents of IP and TCP/UDP headers, including IP addresses, port numbers, and acknowledgment bits. But what if an organization wants to provide a Telnet service to a restricted set of internal users (as opposed to IP addresses)? And what if the organization wants such privileged users to authenticate themselves first before being allowed to create Telnet sessions to the outside world? Such tasks are beyond the capabilities of traditional and stateful filters. Indeed, information about the identity of the internal users is application-layer data and is not included in the IP/TCP/UDP headers.

To have finer-level security, firewalls must combine packet filters with application gateways. Application gateways look beyond the IP/TCP/UDP headers and make policy decisions based on application data. An **application gateway** is an application-specific server through which all application data (inbound and outbound) must pass. Multiple application gateways can run on the same host, but each gateway is a separate server with its own processes.

To get some insight into application gateways, let's design a firewall that allows only a restricted set of internal users to Telnet outside and prevents all external clients from Telneting inside. Such a policy can be accomplished by implementing

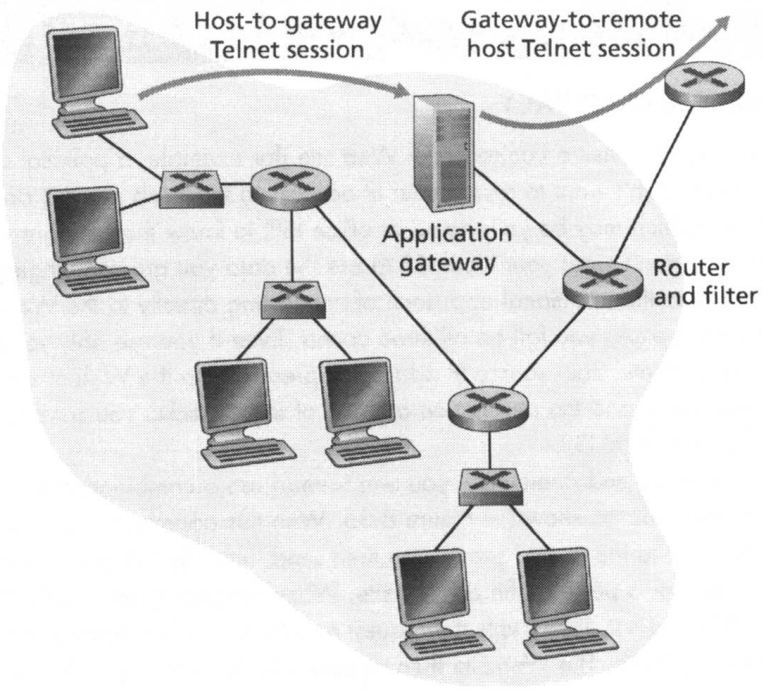


Figure 8.34 ♦ Firewall consisting of an application gateway and a filter

a combination of a packet filter (in a router) and a Telnet application gateway, as shown in Figure 8.34. The router's filter is configured to block all Telnet connections except those that originate from the IP address of the application gateway. Such a filter configuration forces all outbound Telnet connections to pass through the application gateway. Consider now an internal user who wants to Telnet to the outside world. The user must first set up a Telnet session with the application gateway. An application running in the gateway, which listens for incoming Telnet sessions, prompts the user for a user ID and password. When the user supplies this information, the application gateway checks to see if the user has permission to Telnet to the outside world. If not, the Telnet connection from the internal user to the gateway is terminated by the gateway. If the user has permission, then the gateway (1) prompts the user for the host name of the external host to which the user wants to connect, (2) sets up a Telnet session between the gateway and the external host, and (3) relays to the external host all data arriving from the user, and relays to the user all data arriving from the external host. Thus, the Telnet application gateway not only performs user authorization but also acts as a Telnet server and a Telnet client, relaying information between the user and the remote Telnet server. Note that the filter will permit step 2 because the gateway initiates the Telnet connection to the outside world.

CASE HISTORY

ANONYMITY AND PRIVACY

Suppose you want to visit a controversial Web site (for example, a political activist site) and you (1) don't want to reveal your IP address to the Web site, (2) don't want your local ISP (which may be your home or office ISP) to know that you are visiting the site, and (3) don't want your local ISP to see the data you are exchanging with the site. If you use the traditional approach of connecting directly to the Web site without any encryption, you fail on all three counts. Even if you use SSL, you fail on the first two counts: Your source IP address is presented to the Web site in every datagram you send; and the destination address of every packet you send can easily be sniffed by your local ISP.

To obtain privacy and anonymity, you can instead use a combination of a trusted proxy server and SSL, as shown in Figure 8.35. With this approach, you first make an SSL connection to the trusted proxy. You then send, into this SSL connection, an HTTP request for a page at the desired site. When the proxy receives the SSL-encrypted HTTP request, it decrypts the request and forwards the cleartext HTTP request to the Web site. The Web site then responds to the proxy, which in turn forwards the response to you over SSL. Because the Web site only sees the IP address of the proxy, and not of your client's address, you are indeed obtaining anonymous access to the Web site. And because all traffic between you and the proxy is encrypted, your local ISP cannot invade your privacy by logging the site you visited or recording the data you are exchanging. Many companies today (such as proxify.com) make available such proxy services.

Of course, in this solution, your proxy knows everything: It knows your IP address and the IP address of the site you're surfing; and it can see all the traffic in cleartext exchanged between you and the Web site. Such a solution, therefore, is only as good as the trustworthiness of the proxy. A more robust approach, taken by the TOR anonymizing and privacy service, is to route your traffic through a series of non-colluding proxy servers [TOR 2016]. In particular, TOR allows independent individuals to contribute proxies to its proxy pool. When a user connects to a server using TOR, TOR randomly chooses (from its proxy pool) a chain of three proxies and routes all traffic between client and server over the chain. In this manner, assuming the proxies do not collude, no one knows that communication took place between your IP address and the target Web site. Furthermore, although cleartext is sent between the last proxy and the server, the last proxy doesn't know what IP address is sending and receiving the cleartext.

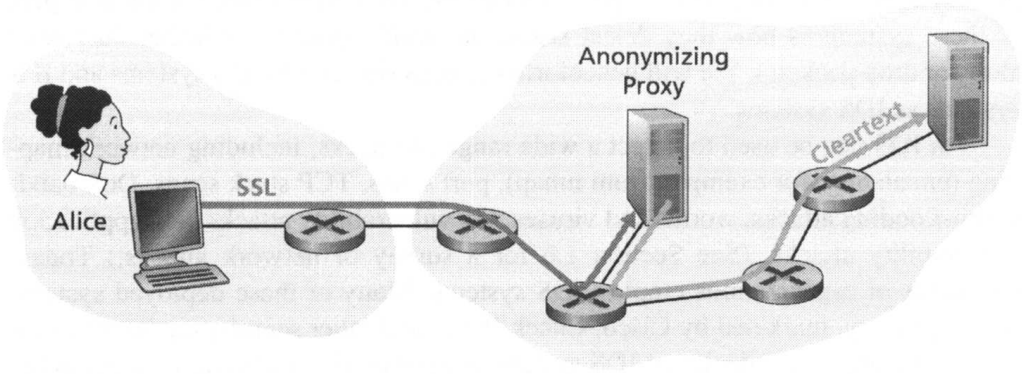


Figure 8.35 ♦ Providing anonymity and privacy with a proxy

Internal networks often have multiple application gateways, for example, gateways for Telnet, HTTP, FTP, and e-mail. In fact, an organization's mail server (see Section 2.3) and Web cache are application gateways.

Application gateways do not come without their disadvantages. First, a different application gateway is needed for each application. Second, there is a performance penalty to be paid, since all data will be relayed via the gateway. This becomes a concern particularly when multiple users or applications are using the same gateway machine. Finally, the client software must know how to contact the gateway when the user makes a request, and must know how to tell the application gateway what external server to connect to.

8.9.2 Intrusion Detection Systems

We've just seen that a packet filter (traditional and stateful) inspects IP, TCP, UDP, and ICMP header fields when deciding which packets to let pass through the firewall. However, to detect many attack types, we need to perform **deep packet inspection**, that is, look beyond the header fields and into the actual application data that the packets carry. As we saw in Section 8.9.1, application gateways often do deep packet inspection. But an application gateway only does this for a specific application.

Clearly, there is a niche for yet another device—a device that not only examines the headers of all packets passing through it (like a packet filter), but also performs deep packet inspection (unlike a packet filter). When such a device observes a suspicious packet, or a suspicious series of packets, it could prevent those packets from entering the organizational network. Or, because the activity is only deemed as suspicious, the device could let the packets pass, but send alerts to a network administrator, who can then take a closer look at the traffic and take appropriate actions. A device that generates alerts when it observes potentially malicious traffic is called an **intrusion detection system (IDS)**. A device that filters out suspicious traffic is called an **intrusion prevention system (IPS)**. In this section we study

both systems—IDS and IPS—together, since the most interesting technical aspect of these systems is how they detect suspicious traffic (and not whether they send alerts or drop packets). We will henceforth collectively refer to IDS systems and IPS systems as IDS systems.

An IDS can be used to detect a wide range of attacks, including network mapping (emanating, for example, from nmap), port scans, TCP stack scans, DoS bandwidth-flooding attacks, worms and viruses, OS vulnerability attacks, and application vulnerability attacks. (See Section 1.6 for a survey of network attacks.) Today, thousands of organizations employ IDS systems. Many of these deployed systems are proprietary, marketed by Cisco, Check Point, and other security equipment vendors. But many of the deployed IDS systems are public-domain systems, such as the immensely popular Snort IDS system (which we'll discuss shortly).

An organization may deploy one or more IDS sensors in its organizational network. Figure 8.36 shows an organization that has three IDS sensors. When multiple sensors are deployed, they typically work in concert, sending information about

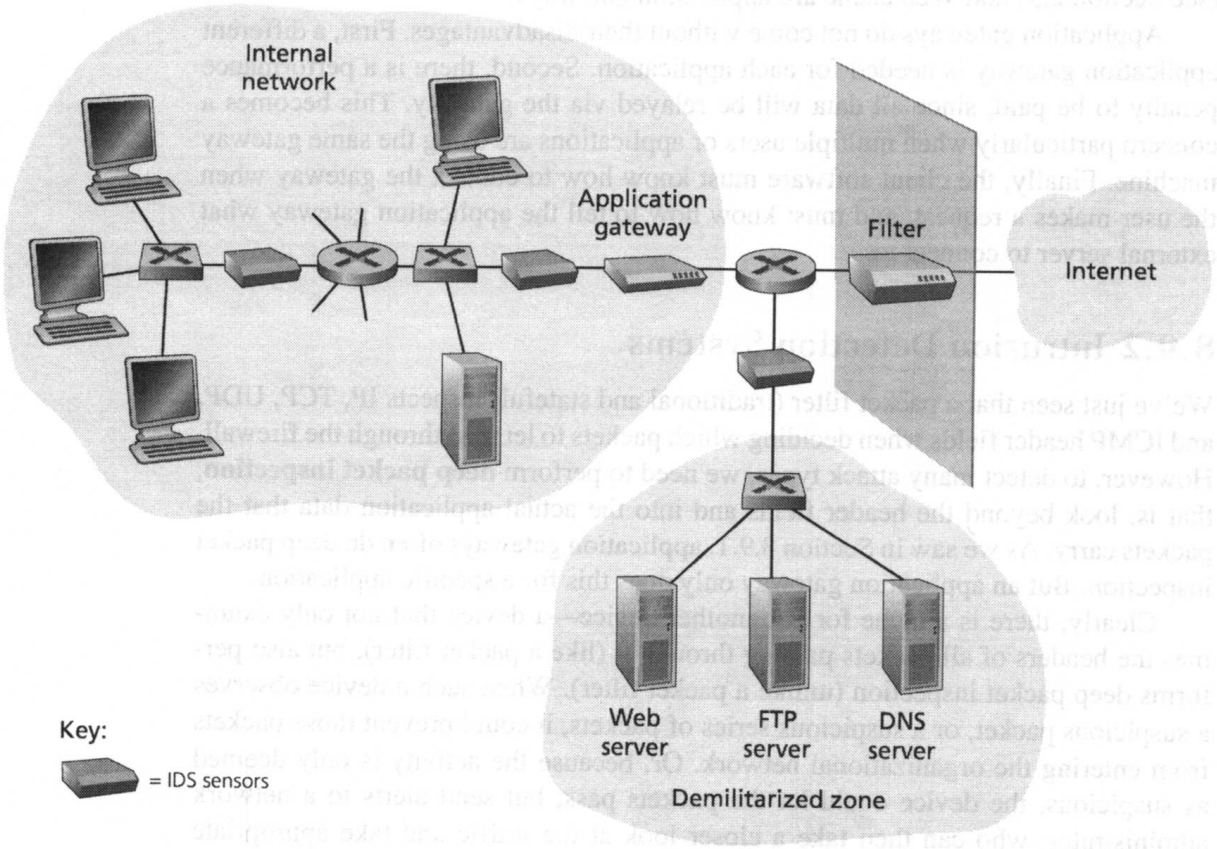


Figure 8.36 ♦ An organization deploying a filter, an application gateway, and IDS sensors

suspicious traffic activity to a central IDS processor, which collects and integrates the information and sends alarms to network administrators when deemed appropriate. In Figure 8.36, the organization has partitioned its network into two regions: a high-security region, protected by a packet filter and an application gateway and monitored by IDS sensors; and a lower-security region—referred to as the **demilitarized zone (DMZ)**—which is protected only by the packet filter, but also monitored by IDS sensors. Note that the DMZ includes the organization's servers that need to communicate with the outside world, such as its public Web server and its authoritative DNS server.

You may be wondering at this stage, why multiple IDS sensors? Why not just place one IDS sensor just behind the packet filter (or even integrated with the packet filter) in Figure 8.36? We will soon see that an IDS not only needs to do deep packet inspection, but must also compare each passing packet with tens of thousands of "signatures"; this can be a significant amount of processing, particularly if the organization receives gigabits/sec of traffic from the Internet. By placing the IDS sensors further downstream, each sensor sees only a fraction of the organization's traffic, and can more easily keep up. Nevertheless, high-performance IDS and IPS systems are available today, and many organizations can actually get by with just one sensor located near its access router.

IDS systems are broadly classified as either **signature-based systems** or **anomaly-based systems**. A signature-based IDS maintains an extensive database of attack signatures. Each signature is a set of rules pertaining to an intrusion activity. A signature may simply be a list of characteristics about a single packet (e.g., source and destination port numbers, protocol type, and a specific string of bits in the packet payload), or may relate to a series of packets. The signatures are normally created by skilled network security engineers who research known attacks. An organization's network administrator can customize the signatures or add its own to the database.

Operationally, a signature-based IDS sniffs every packet passing by it, comparing each sniffed packet with the signatures in its database. If a packet (or series of packets) matches a signature in the database, the IDS generates an alert. The alert could be sent to the network administrator in an e-mail message, could be sent to the network management system, or could simply be logged for future inspection.

Signature-based IDS systems, although widely deployed, have a number of limitations. Most importantly, they require previous knowledge of the attack to generate an accurate signature. In other words, a signature-based IDS is completely blind to new attacks that have yet to be recorded. Another disadvantage is that even if a signature is matched, it may not be the result of an attack, so that a false alarm is generated. Finally, because every packet must be compared with an extensive collection of signatures, the IDS can become overwhelmed with processing and actually fail to detect many malicious packets.

An anomaly-based IDS creates a traffic profile as it observes traffic in normal operation. It then looks for packet streams that are statistically unusual, for example, an inordinate percentage of ICMP packets or a sudden exponential growth in

port scans and ping sweeps. The great thing about anomaly-based IDS systems is that they don't rely on previous knowledge about existing attacks—that is, they can potentially detect new, undocumented attacks. On the other hand, it is an extremely challenging problem to distinguish between normal traffic and statistically unusual traffic. To date, most IDS deployments are primarily signature-based, although some include some anomaly-based features.

Snort

Snort is a public-domain, open source IDS with hundreds of thousands of existing deployments [Snort 2012; Koziol 2003]. It can run on Linux, UNIX, and Windows platforms. It uses the generic sniffing interface libpcap, which is also used by Wireshark and many other packet sniffers. It can easily handle 100 Mbps of traffic; for installations with gibabit/sec traffic rates, multiple Snort sensors may be needed.

To gain some insight into Snort, let's take a look at an example of a Snort signature:

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any
(msg:"ICMP PING NMAP"; dsize: 0; itype: 8;)
```

This signature is matched by any ICMP packet that enters the organization's network (`$HOME_NET`) from the outside (`$EXTERNAL_NET`), is of type 8 (ICMP ping), and has an empty payload (dsize = 0). Since nmap (see Section 1.6) generates ping packets with these specific characteristics, this signature is designed to detect nmap ping sweeps. When a packet matches this signature, Snort generates an alert that includes the message "`ICMP PING NMAP`".

Perhaps what is most impressive about Snort is the vast community of users and security experts that maintain its signature database. Typically within a few hours of a new attack, the Snort community writes and releases an attack signature, which is then downloaded by the hundreds of thousands of Snort deployments distributed around the world. Moreover, using the Snort signature syntax, network administrators can tailor the signatures to their own organization's needs by either modifying existing signatures or creating entirely new ones.