The McGraw·Hill Companies

The Nuts and Bolts of e-Commerce

The McGraw·Hill Companies

The Nuts and Bolts of e-Commerce

Dhruv Nath
Professor
Management Development Institute
Gurgaon

Tata McGraw-Hill Publishing Company Limited
NEW DELHI

McGraw-Hill Offices
New Delhi New York St Louis San Francisco Auckland Bogotá Caracas
Kuala Lumpur Lisbon London Madrid Mexico City Milan Montreal
San Juan Santiago Singapore Sydney Tokyo Toronto

The McGraw·Hill Companies

Tata McGraw-Hill
*A Division of The **McGraw·Hill** Companies*

ISBN 0-07-043625-8

Published by Tata McGraw-Hill Publishing Company Limited,
7 West Patel Nagar, New Delhi 1 10 008, typeset at Script Makers,
19, A-1B Block, DDA Market, Pashchim Vihar, New Delhi 110 063 and printed at
Sheel Print-N-Pack, D-132, phase II Extension, NOIDA

Cover design: Angeline Sunderraj, Bangalore

RZDYCRBARCRYX

The McGraw·Hill Companies

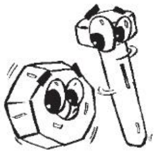
To

Mom and Pop

For everything...........and more !

The rest of the message is unsaid

A Few Words Before You Begin...



Dear Reader,

Now that you have picked up this book, it's only fair that you understand why it was written. And then decide whether or not to buy it. And still more important, whether or not to read it.

Well, first of all, this is not "Yet another book on e-Commerce".

No sir ! There are too, too many of that kind in the market for me to add one more. So I am not talking about all the things you can do with e-Commerce, such as buying and selling, and chatting and e-mail. I'm not trying to help you figure out whether or not you should put up banners on search engines. I'm definitely not harping on and on about how the Internet has revolutionised society. Because there are more than enough books that cover those areas.

So, what am I talking about?

To understand this, you must first understand how the book was thought of. It was actually born out of frustration.

That's right! For years, I have come across normal human beings like you and me tearing their hair apart, trying to understand how e-Commerce worked. Not how great it was, but how it worked. So they would hear from their computer savvy friends how TCP/IP had transformed the world. Without the foggiest notion of whether TCP/IP was a new vaccine or simply the initials of India's latest satellite in space. Or they would read an ad in the papers about how a terrific new application server had been launched, with no clue of what an application server was and how it would help cure all ills that might befall humanity. Or again, listen spellbound to someone harping about digital signatures, wondering whether these were normal signatures written in ink, but scanned into a computer.

And that's when I realised that there was probably no book catering to this need. The need was strong—hundreds of frustrated men and women testified to this—but strangely enough, no one had ever attempted to address this need. (Well, I may be wrong—there could perhaps be a book in Spanish and selling in Argentina, but at least in the English speaking world I have not come across anything like this.)

And one fine day, I decided to tackle the problem. And hopefully relieve all those wonderful men and women of their frustrations. And this book was born.

So this book, dear reader, is about "Truly understanding e-Commerce". About understanding how databases work. And how they are normalised. About understanding Java. And how it manages to work across all kinds of computers. About understanding firewalls, and Domain Name Servers, and GIFs and JPEGs. And so much, much, much more– – .– – – – – – – – –

If you're a manager who wishes to understand what's going on before implementing your e-Commerce system, this book is for you. If you're a student trying to grapple with the concepts of e-Commerce, and can only find stuff that is written for computer professionals, this book is for you. If you're a computer professional, but have not picked up e-Commerce, this book is also for you. In fact, if you are just about anybody—from a housewife, to a retired insurance agent, to a salesman, to a shopkeeper—who wants to understand the subject without picking up a PhD in Computer Science, just go ahead and read this book. And don't forget to let me know how you found it !

Remember These Thoughts… … … … .…

Since you have reached this far, I assume you have decided to buy the book. First of all, welcome to the gang. And secondly, I must share with you some thoughts before you proceed.

First of all, I must clarify that I am not attempting to describe or compare specific products and vendors. Why? Well, you must know that the technology you are now attempting to understand, changes very fast. Between the time I write this piece and the time the book comes out in print, the world would have marched ahead. And of course most vendors would have brought out newer and better versions of their products. So any comparisons I might make at this stage would be quite meaningless. That is best done on the Net itself—simply go to the vendor's site and locate the product that is of interest to you.

Instead, I have attempted to clear up concepts. And fortunately, concepts last a lot longer than products and versions. So you may see many, many versions of products coming through, but the concepts I describe in this book will probably remain valid for some time.

Secondly, in the interest of clarity, I have tried to keep things simple. In fact, in some cases I have taken the liberty to oversimplify concepts—my apologies in advance. I have also left out details wherever I believed they were not required in a book of this nature. Remember, this book is not intended to be a "Bible" of e-Commerce. It is a book that attempts to make a complex subject simple.

And finally, I must admit that in the many examples I have used in this book, at the risk of offending our lady readers, I have taken a man's viewpoint. Our lady readers would, no doubt, realise that these examples are merely for expression and not intended to hurt their sensibilities. However, if there is sufficient demand, I could consider bringing out another edition of this book–

And Finally, the True Story…………………………..

Before I end, I must tell you the real reason for writing the book (but please keep it strictly to yourself).

You see, I have always wanted to write a funny book. For years I have lived under the delusion that I had a sense of humour. People used to laugh at my jokes. And I used to get decent marks in English at school. The logical conclusion was that as a writer I could be a brilliant humourist.

There was only one problem—I couldn't even dream of competing with my childhood heroes such as Richmal Crompton (William), Stephen Leacock, and George Mikes. No, that would have been sacrilege. Completely unacceptable!!

So I had to write a humorous book with a difference. And that's when I realised that I had to have a subject around which to write the book.

There began my search—and one night I had a brainwave: Why not e-Commerce. After all, my job involves e-Commerce. I have been a consultant to several companies in the area of e-Commerce (and they continue to make the blunder of paying me for my advice).

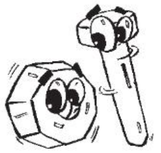
And so my decision was made: I would write a humorous book on e-Commerce.

And here, dear reader, I must make my position very, very clear—I am first and foremost writing a humorous book. It just happens to be on the subject of e-Commerce. (It could so easily have been on the subject of Drip Irrigation except for one minor problem—I know next to nothing about the subject.)

So if you do not understand the concepts of e-Commerce from this book, I would perhaps be a little upset. But if at the end of reading the book you turn around and tell me it was not funny, I would be very, very disappointed. Either way, do let me know at dhruvn55@hotmail.com or dhruv@mdi.ac.in.

So go ahead, and have fun. And I hope you have as much fun reading the book as I had writing it !!!

Acknowledgements



Dear reader,

When I had finished writing this book, I sat down to write this section on Acknowledgements. And I realised with a start that it wasn't really my book !

Yes, it does have my name on the cover. And of course, all the jokes are strictly mine. But it would be unfair to call it my book. Simply because there have been many, many people who have contributed in a huge way, to making it possible.

Here are some of them– – – – – – – – .

First of all, R.S.Pawar or simply RSP—better known as the Chairman of NIIT, who has been more of a friend and guide to me than a boss. For his faith in me, for the opportunities he has given me at every stage (which is where I learnt what I am telling you). And most of all for the freedom he gave me to pursue directions I was keen on—such as writing this book.

My seniors at NIIT, Vijay Thadani, Arvind Thakur, and P Rajendran, who, along with RSP, have built up a terrific organisation called NIIT. They've helped me to learn at every stage. And they have not only put up with, but positively encouraged the strange behaviour of a person who, after 18 years in an organisation, suddenly comes up and says he wants to get into Academics.

Dr. Devi Singh, Director—Management Development Institute, who has been extremely encouraging ever since I joined MDI. He has taken keen personal interest in seeing this book come out—so much so, that I will now be writing a whole series of them.

Several people have helped me understand the subject of e-Commerce, but amongst them one man stands out—my colleague and guru, Chetan Juneja—terrific clarity, huge depth of knowledge, and if ever I asked him a question to which he didn't have an answer (which by the way was extremely rare), he'd have it ready the next morning.

Many, many colleagues and excolleagues have contributed to teaching e-Commerce to a novice like me so that I could pass it on to all of you. I wish I could name all of them, but that would mean a few more pages (and I'm reasonably certain you would refuse to bear that cost). So a truncated list would be Dr Sugata Mitra, Deepak Prabhakar, Vijay Ghei, Suranjan Choudhury, Himanshu Sud, Deb Bhattacharji, Deepak Mishra, Abraham Tharakan, Arvind Mehrotra, Suresh Kumar, Sumit Madan– – – – ..

Prof. SN Maheshwari, at the Department of Computer Science and Engineering, IIT Delhi, has been my guru from the time I was a young, uncertain PhD student. Among several other things, he taught me how to write, re-write, and re-re-rewrite till I almost fought with him. And in the process, produce something that was truly worth reading.

Tata McGraw-Hill has been a great publisher, and I must thank Dr N Subrahmanyam, V Deepa, Chandra Sekhar, and Nidhi Sharma. It's been great fun writing this book with them, and I believe the book is far better because of their active support and involvement. I can imagine the guts it must have taken on the part of the leading publisher of technical books, world-wide, to approve a funny, completely non-serious book. And that too, with cartoons on every page !

And of course, the man who created the cartoons—my friend Arka, the terrific artist—who is probably the reason why you picked this book off the shelves in the first place.

My colleague V.M. Johnson, for doing a great job in supporting a somewhat eccentric boss—through drafts and printouts and 11 levels of back-up !

My children, Malvika and Siddharth, deserve a special mention. I am delighted to tell you that at the ripe old ages of 15 and 12 respectively, they are far better humorous writers than their father—and therefore all the funny parts in this book are the joint effort of father, daughter, and son. Incidentally, had it not been for their presence, this book would have been completed in one-tenth the time. But it would also have been one-tenth as much fun.

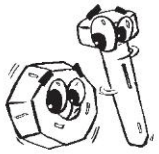
And finally, if you've seen movies (I'm sure you have), you would know that the most important person is mentioned last of all—the Director. The one who is not seen on the screen, but has a key hand in everything that is done. And that, dear reader, is Rajni—the only girl friend I've ever had—and who I married seventeen years ago. It is not easy to manage a home and family and kids almost single-handedly, when your husband is doing a full-time job, and has also got it into his head to write some silly book. But Rajni has come out of this with flying colours! In addition to being a patient punching-bag during my frequent explosions.

Strictly between you and me, she doesn't know she's made it so easy for me, that I'm now planning to write a whole series of books !

Thank you everyone—for all that you've done. And in case you didn't know it, I'm writing many books like this one—so you'll get a great chance to help again!

The McGraw·Hill Companies

Contents



1

The Wonderful World Wide Web

So you have finally got down to reading this book—I assume that's what you are doing since you are on this first page.

Obviously, you have gone through the preface. If for any reason you haven't, please go through it immediately. I did not take the trouble of writing it simply to add a few extra pages to this book !

You therefore realise that I expect you to know something about the Internet already—No, not too much, but basic stuff. For instance, you have already been surfing comfortably and therefore understand **browsers** and **hyperlinks**. You probably also know what **HTML** stands for and does. And you may just be aware that **HTTP** is a protocol for transmitting HTML pages.

However, it's always a good idea to refresh your memory. Further, I would like to ensure that your understanding of the terms used here and my understanding of the same terms is similar. For example, both of us should agree that a browser is something you use to get information from the Internet, and not a person who stands idly in a bookshop, flipping pages at random, without any intentions of buying the book.

Therefore, in this chapter we would refresh our existing knowledge about the Internet, about HTML, about the **world wide web**, and so on. If you've heard all this before, just skip it and go on to the next chapter.

But if you haven't, then just read on.........

HYPERLINKS AND HTML

When you sit down in front of your PC and log on to the Internet, what's the first thing you see?

The browser naturally (Fig. 1.1)—which is essentially a piece of software that runs on your PC and helps you get pages of information from the Internet. It is the interface between you and the Internet. And as you are already aware, it is fairly simple to use.

Fig 1.1
A browser

What do you do with the browser? Well, obviously you type in the address of the page of information that you want to see—such as amazon.com, or to make it a little more exciting, government_forms.nic.in. This page could, of course, be found anywhere on the Internet. Your good old faithful browser locates this page for you and displays it on your PC (how this address works and where this page comes from will be discussed later).

Now, is that the end of your expedition on the Internet?

Of course not! The page you have just brought into your browser will itself have several pointers or links to other pages (Fig. 1.2).

Each of these links shows up as a picture or text on your page. But what is not visible to you is that each link has an address associated with it. You could choose any of these links, click on it, and the browser will automatically fetch the page pointed to by the corresponding address—you don't need to type in the address.

You would also know that these links are called **hyperlinks.** Therefore, as you can see, a page of information on the Internet includes text and hyperlinks—which are nothing but text or even small pictures pointing to other pages. And since computer people love to make simple things sound complex—to ensure the survival of their species of course—they call this **hypertext**.

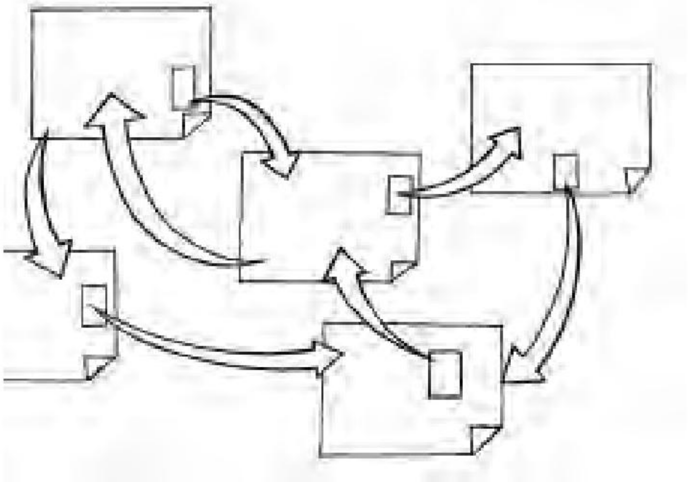


Fig. 1.2 *Pages on the Internet, linked through hyperlinks*

And now we need a language—a language in which pages on the Internet can be written, which the browser understands and therefore displays. This language is called the **HyperText Markup Language**, or good old **HTML** for short. Incidentally, HTML was originally invented by Tim Berners-Lee, a scientist working at CERN, the European Laboratory for Particle Physics in Geneva.

So now we begin to get a little specific. The browser fetches and displays HTML pages on your PC. And each page contains hypertext, which is a combination of text and hyperlinks.

Fig. 1.3(a) depicts a simple series of HTML statements, and Fig. 1.3(b) shows the corresponding web page that these statements generate on the browser.

You would notice that HTML is composed of statements such as the following:

```
<TITLE>This is a sample</TITLE>
<H1>Dhruv Nath's Home Page</H1>
<P>This is a very simple example of HTML</P>
```

Do you see a pattern here?

Sure you do! Each HTML statement is composed of pairs such as,

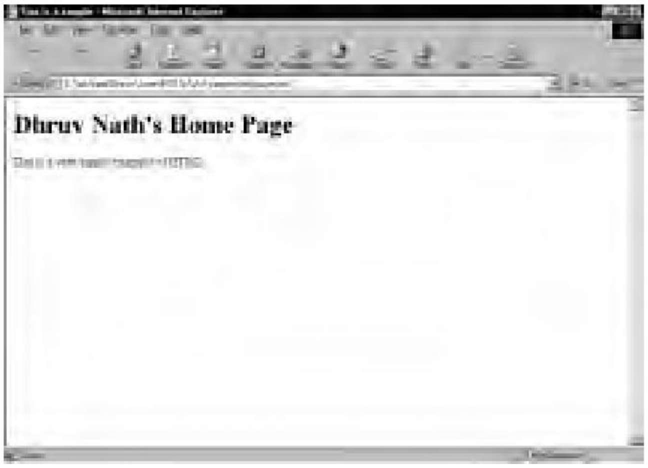
```
<TITLE>......</TITLE> and
<H1>......</H1>
```

Each such pair is called an HTML tag. For instance, the pair

Fig. 1.3(a) *A simple series of HTML statements*

```
<HTML>
<HEAD>
<TITLE>This is a sample</TITLE>
</HEAD>
<BODY>
<H1>Dhruv Nath's Home Page</H1>
<P>This is a very simple example of HTML</P>
</BODY>
</HTML>
```

Fig. 1.3(b) *The corresponding HTML page*



Dhruv Nath's Home Page

This is a very simple example of HTML

<TITLE>......</TITLE>

indicates that whatever is written between them becomes the title of the page (which appears right at the top of the browser page in the blue area).

Similarly the tag

<H1>......</H1>

indicates that whatever is written between the pair becomes the heading on the web page.

I wish I had the time to describe HTML in more detail—and you had the time to learn it in the same detail. Unfortunately, that would probably need another book—and you would need to change your job to that of a full time programmer! For the moment, therefore, let's just live with the fact that HTML permits you to format text in various ways and display it on a browser.

Now there is one question which must have occurred to you: For such a simple HTML page, the program appears rather cumbersome. What happens with more realistic (and therefore more complex) pages?

Naturally, the HTML code in those cases is far longer. And far, far more tedious to write.

So is there a solution?

Yes, there is! Several vendors have created software tools called **HTML editors,** which are easy to use, and which generate the required HTML code. All you need to do is to give the required instructions to this software tool, and then go to sleep—the tool does the rest. Life for you is simplified, and the software vendors make a bit of money—vendors such as Microsoft, who supply an editor called Front Page, Adobe with Page Mill, and Macromedia—who supply DreamWeaver !!!

WEB SERVERS

Okay. We have seen that when you type the address of an HTML page into your browser, the browser gets you the page.

The question is, where does the page come from? And how is it brought to your PC?

Action !! Enter the good old **web server** !!

But wait. Let's go back to the beginning.

You may or may not be surprised to know that the world of the Internet is broadly divided into two parts—those who serve, and those who are served. It's like a restaurant, where you have customers or clients, who are served, and waiters, cooks, cashiers, and even the durban who salutes you—who provide these clients with the service.

In exactly the same way, on the Internet, there are computers that provide a service, and there are other computers who get this service. And now the most brilliant deduction of all—those who provide the service are called **servers**, and those who use or ask for the service are called **clients**. This mode of working is popularly known as **client-server** computing.

So when you use a PC to access an HTML page, your PC is the client and the computer where this page comes from is the server. Similarly, when you click on a hyperlink, the next page also comes from a server, and so on.........

There is one more term that you need to learn here. Notice that these hyperlinks connect pages to each other, all around the world. Since all these pages are stored on various servers, the servers are also indirectly linked. If you let your imagination run wild, you could view this as a huge web across the world, rather like a spider's web (Fig. 1.4).

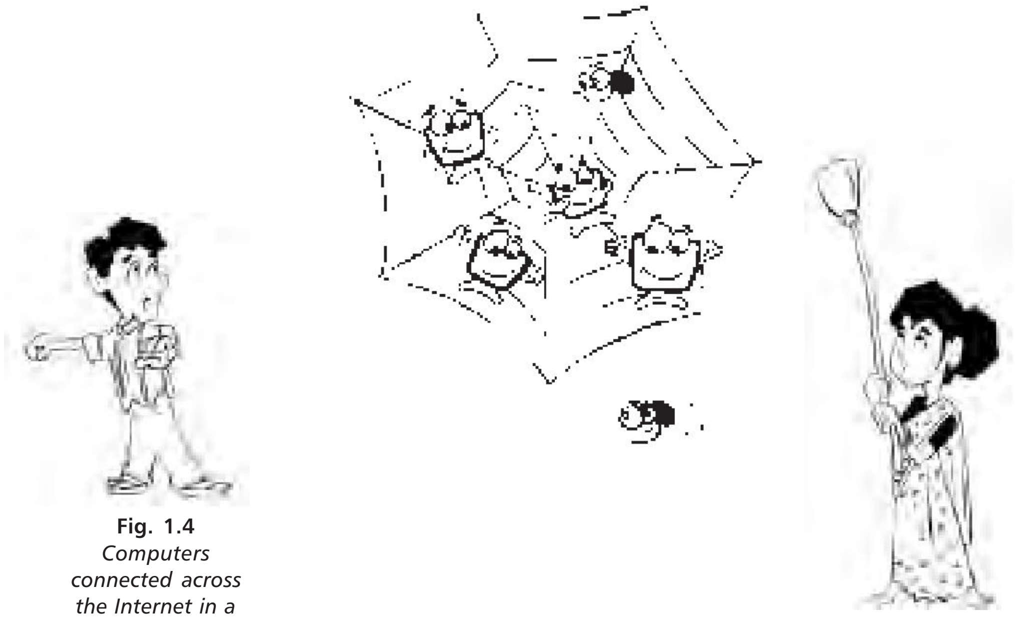


Fig. 1.4 *Computers connected across the Internet in a huge web*

And that, my friend, is the **world wide web**, or **WWW**. The collection of all the servers and the associated software that store HTML pages, and make these available to client PCs on request.

What are these servers called? **Web servers** naturally!!! What else did you think? And equally naturally, HTML pages are also often called **web pages**.

Incidentally, it is the world wide web that has led to the boom in the usage of the Net, whether it is housewives, school teachers, or retired insurance agents. Simply because this whole business of "point and click" has made it very, very easy to use. Earlier, the Net was the preserve of academicians, researchers, and scientists, with at least two degrees against their names. But now, it's child's play—just point and click.

Interestingly, the world wide web is a subset of the Internet—there are enough computers on the Net that do not operate on HTML—but it is by far the most popular subset. And when people like you and me speak about the Net, we generally refer to the world wide web.

One issue that confuses a lot of people is this: The term server is used for both hardware and software. So the hardware that stores and dishes out web pages is called a web server. And the software that runs on it, which permits it to dish out these pages, is also called a web server. Similarly, the PC from where you access the Internet is the client, and the browser that runs on your PC and enables it to access the Net (or now more accurately, the world wide web), is also called the client.

What do we do about this confusion?

Actually nothing—both uses of these terms are common—and as you get more and more familiar with the subject, you will be able to figure out the difference from the context.

Some of the popular web servers in existence are the Internet Information Server or IIS from Microsoft, and the Netscape Enterprise Server from Netscape (now SUN Microsystems). There is also the Apache Web Server, which is not supported by commercial organisations like Microsoft or Netscape, but is supported and evolved by a group of software professionals world-wide. Interestingly, IIS only runs on the Microsoft Operating System (Windows/NT, and now of course Windows 2000), whereas Apache and Netscape run on both Windows/NT and UNIX.

Let us now complicate life a bit more. Suppose your HTML page (or web page as it is popularly known) had graphics, animation, and maybe even audio in it. How would these fit in with HTML?

Actually HTML has a special tag which permits you to use pictures or images. Each such graphic image is stored as an independent file. And each such tag gives the location of this file on the web server.

So how does it work?

Simple ! When the browser makes a request for a web page, the server only returns the HTML page and not the associated graphic image files. The browser now scans through the page and realises that for each such tag it needs to send out a further request for the associated graphic file. Subsequently these files are downloaded from the web server and inserted into the HTML page at the appropriate location (you may have noticed this when you surf the Net—the HTML page itself comes in early and the graphics follows). And incidentally, exactly the same logic is valid for any other kind of file such as sound or video.

OUR FIRST PROTOCOL FOR THE NET—HTTP

Hey, that sounds funny ! Protocols on the Internet? But aren't protocols something that diplomats follow?

Sure they are. So if the President of Upper Moravia were to come to your country on a five-day state visit, and you gallantly asked the junior-most secretary in the Department of Animal Husbandry to welcome him with a booming 21-gun salute at the airport, the President is not likely to talk to you ! And that's protocol ! He must be met by someone of appropriate rank, so that a few egos (largely the President's) do not get bruised.

And that's the key—for the two countries to talk to each other, a protocol must be followed.

In simple terms therefore, a protocol is a previously agreed to arrangement which enables two people, or two computers, or even two countries, to communicate with each other. Since it is agreed to, *this is the only way these two communicate, and it must be followed every time.*

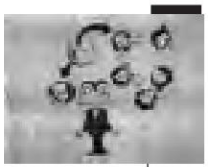
Okay, so where does this fit in with our web servers and browsers?

Well, the browser and the web server have a pre-arranged protocol which permits them to exchange information, whether it is a request from the browser for a page, or the page being sent back by the server. And this protocol is called the **HyperText Transfer Protocol (HTTP)** for obvious reasons.

How is this protocol implemented? After all, there must be someone who understands it and ensures that it is followed.

As usual, it is software that does this job. Software that is part of the browser at the user end, and part of the web server at the server end.

SUMMARY



So there we are !

In case you hadn't realised it, you have just taken your first step towards understanding e-Commerce. Let's now create a simple diagram of what an e-Commerce system looks like (Fig. 1.5). We'll use the term **architecture** to describe this diagram. In fact, we'll see this architecture in subsequent chapters as well, where we'll keep adding pieces to make it more and more complete.

In the figure, you of course are the handsome young man sitting at the PC at the extreme left, and you access the Internet through the browser. Or to be a bit more accurate, you access the world wide web through the browser.

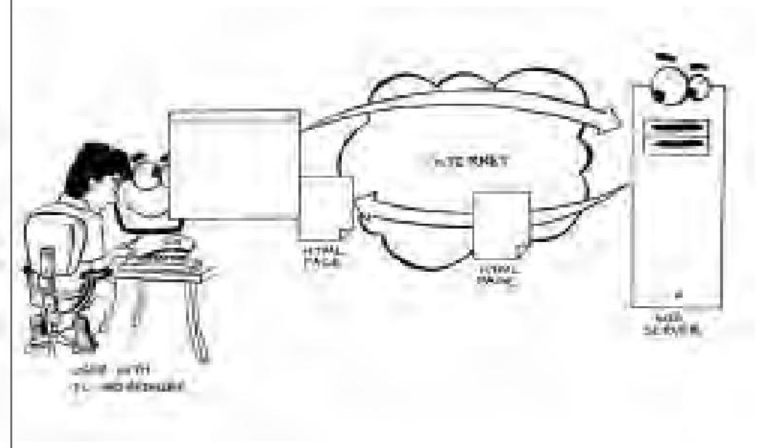


Fig. 1.5 *Our very first e-Commerce architecture*

Through the browser, you send out a request for a web page, using the HTTP protocol. And you reach the web server that contains, or "hosts" this web page. The web server sends the page back to you, once again using the same HTTP protocol.

Each web page may have further links to other web pages, which may be on the same web server, or elsewhere. These links are called hyperlinks. And the web page, of course, is written in something called HTML, or Hyper Text Mark-up Language.

So now you have a simple understanding of the Internet, of HTML and HTTP. And you are aware of what browsers do and how hyperlinks work. And of course when someone uses the term "web", there is a reasonable chance that you will not think of spiders but of the world wide web.

That's enough technical terms for now. Please go to sleep—perhaps dreaming of spiders ! We meet again in the next chapter.

2

Internet Addressing, the Hard Way: The IP Story

So you have learnt all about web servers and browsers. And about HTML and HTTP. I am sure you're feeling like a techie already—oozing technology from every pore ! Let me tell you that these are great terms to drop at a party......and even more so, if you know what they mean.

But is that enough?

Obviously not. In fact, we haven't yet addressed a very basic question.

As you are aware, there are millions and millions of web servers on the Internet. So how do you pick out the one that you want?

It's actually just like locating the house of your girl friend or boy friend—out of the millions and millions of houses in the city, how do you specify the address of his or her house? Similarly, how do you specify the address of the web server that you want to reach?

Good question, my friend. Have patience, go through this chapter, and hopefully you'll get your answer.

INTERNET ADDRESSING USING IP ADDRESSES

All computers on the Internet have a unique address, just like the postal system where every house has an address. And this address is a number such as 230.43.109.77—in other words four numbers, each between 0 and 255, and separated by dots. This addressing scheme is popularly known as the **IP addressing scheme** —IP standing for **Internet Protocol**, naturally. And the address itself is known as the **IP address**.

For those of you who are somewhat more IT-friendly, this address is actually a 32-bit address, split into four numbers, each 8 bits long. And you would recall from good old binary mathematics that an 8-bit number can have a value that is anywhere between 0 and 255. But if that sounds like Greek to you, just ignore it. You can still make tons of money from the Internet without understanding the first thing about binary mathematics).

So that's it !!! When you want to get a page from sillysite.com, you simply type in 45.234.200.245 (assuming that is the IP address of the web server that hosts the site).

Isn't that correct?

No, of course not. You type in sillysite.com!!!

But then, how is sillysite.com converted to 45.234.200.245?

Good question—in fact it is such a good question that it deserves a separate chapter—so wait till Chapter 4. For the moment, simply assume that you will continue to use the IP address.

One other thing—do only servers on the Internet have IP addresses? What about your own PC? It is certainly not a server. All it has is a browser, which you use to surf the Internet and possibly send e-mail. So does it need an IP address?

Of course it does ! Remember, not only do you need to locate the web server, the web server also needs to locate you. Otherwise how else would it send you the web page you asked for?

OK. So how does your PC get its IP address?

Well, that's a complex question which depends on the kind of connection you have (among other things). You may be accessing the Internet from a dial-up connection obtained from an ISP (Internet Service Provider) such as VSNL or Mantra On–Line. Or you may be accessing it via your cable operator. Or even from the local area network in your office, which in turn has a leased line to VSNL. How your PC gets its IP address depends on all these and more.

For instance if you have a dial-up connection, your PC has no permanent IP address. Instead, your ISP gives you a temporary one when you log on, and then takes it back when you have finished surfing.

In short, every computer on the Internet *must* have an IP address (see Fig. 2.1). Some have temporary addresses, and some have permanent ones—but each of them must have one. And just to add to your collection of Internet terms, we will call such computers **hosts.** A host therefore, is any computer (whether client or server) that is connected to the Internet, and therefore has an IP address.

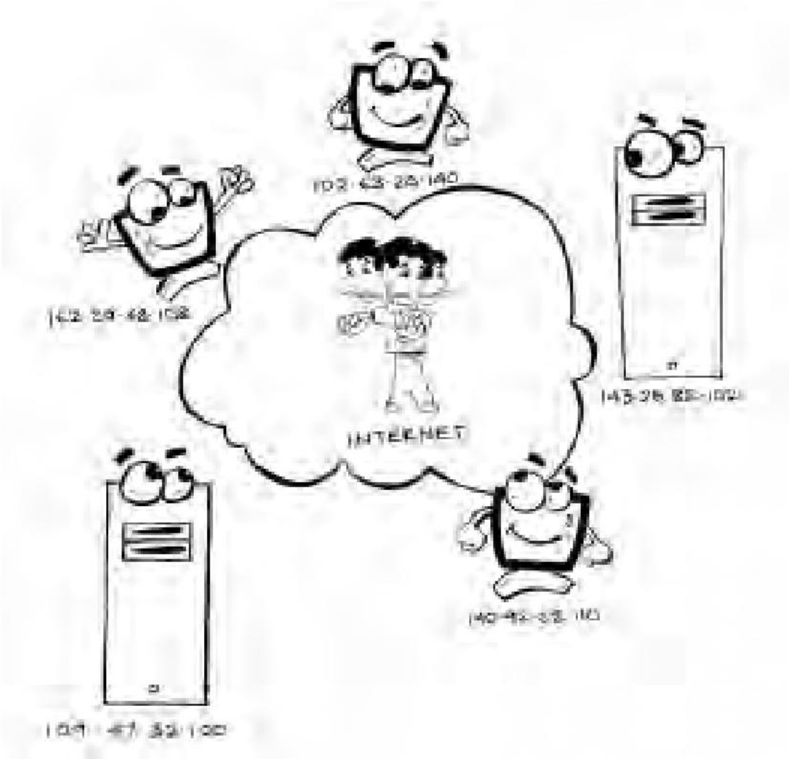


Fig. 2.1 *Hosts on the Internet, each with a unique IP address*

IPv6

Before we leave the subject of IP addressing, I must bring in one last complication—how many computers (or hosts) can be connected to the Internet at any time?

“Aha !” you say, “I knew we would finally get into advanced binary mathematics.”

Actually it’s not so complex. If you notice, any IP address is 32 bits long. And it is fairly straightforward to figure out that these 32 bits give you approximately 4 billion unique combinations or 4 billion unique IP addresses in all.

But, is that enough?

Well, at this point there are a few hundred million hosts on the Internet, and therefore we are able to manage. Moreover, we will probably continue to manage for maybe a couple of years.

But wait! Haven’t you heard of Internet devices—air conditioners, refrigerators, stereo systems, TVs, and maybe even light bulbs? Devices which are on the Internet, and can therefore communicate with any other Internet device in the world? For instance your refrigerator may figure out that it has no more jam (or cheese—it doesn't really matter which), and therefore call up the local grocer, who promptly supplies the jam—with you blissfully unaware of what has happened till you get the bill !

Now, do you think we can still manage when these Internet devices start proliferating? With maybe 10, 20, maybe even 50 Internet devices per household? And far, far more in each office?

Suddenly the number of potential IP addresses appears tiny, doesn't it? How terrible—32 bits which halt the progress of mankind (and definitely halt the progress of the refrigeration industry).

But wait. Mankind doesn't give up so easily. Mankind has gone ahead and created a new version of the IP address, called **IPv6**, which is 128 bits long. And therefore potentially gives us over two hundred million million million million million million IP addresses of the new variety (incidentally the current version, which has 32 bits, is IP version 4, or simply IPv4). IPv6 is not yet in place on the Internet, but believe me—it will be.

Summary

That was short and sweet, my friend. A quick peek into the world of IP addresses—IP of course standing for Internet Protocol. IP addresses are 32-bit addresses that are unique to any computer (or host) on the Internet. And all hosts in today's world are ultimately located using these addresses.

But what about tomorrow's world? Where your refrigerator and all the electric bulbs at home may all be on the Internet?

In tomorrow's world, my friend, we need far, far more hosts on the Internet—and therefore a new version of IP addressing, which is 128 bits long, has emerged.

So you can go ahead and construct your dream house, without worrying. You can have all the light bulbs chatting away with the cash register of your neighbourhood grocer, if you so wish.

Because technology, good old, ever reliable technology, will support you !

3

TCP/IP: The Initials of the Internet

We have just seen the wonderful world of Internet addressing in the last chapter. If you recall, we actually looked at two problems—the problem of locating your girl friend's or boy friend's house, and the related problem of locating a computer on the Internet.

Sadly, we were able to give a solution to only the second problem. But anyway, having figured out how to address each host, we have another key question to answer. Namely, how is the request (or the web page) actually transmitted between the client and the web server?

That, dear reader, is the equally wonderful world of packet switching and TCP/IP. And that, by the way, is the subject of this chapter.

PACKET SWITCHING

Let us first get down to basics. When the browser and web server interact, what they are doing is sending information, or a message, from one to the other.

That's rather like the telephone, isn't it—where you pick up the phone and talk to someone on another phone anywhere in the world? So to understand how communication takes place on the Internet, let us first understand our good old telephone system.

If you think carefully, there is a very fundamental problem with our phone system. When you call up a friend, the telephone network establishes a connection between both of you. And this connection remains reserved for you till you hang up.

What does this mean? Suppose you were having a fight with your girl friend or boy friend and you said something really mean to her (or him). The next several seconds are spent in silence—you are wondering how you had the guts to say something like that, and she (or he) is wondering how to react. So you are both silent, thinking. But the winner is our good old Telecom Department—the phone line is still reserved for both of you and therefore you pay even for the long pauses.

In other words, reserving a connection for two parties wanting to communicate with each other is expensive and a waste of resources.

Let's take another analogy. I am sure you've often been stuck in your little Maruti car at a level crossing (if you are lucky enough to own a Mercedes instead of a Maruti, please loan it to me for a month, so I can check whether the logic still holds). Now the great thing about level crossings is that once the gate is closed, it remains closed till the entire train passes by—unless of course there is a suicide pact between someone on the road and the guard at the crossing. And if you are unfortunate enough to arrive just after the gate has been closed, too bad !!!!

So you sit there in your little Maruti (Fig. 3.1), and curse the two thousand wagons going past (at least it feels like two thousand).

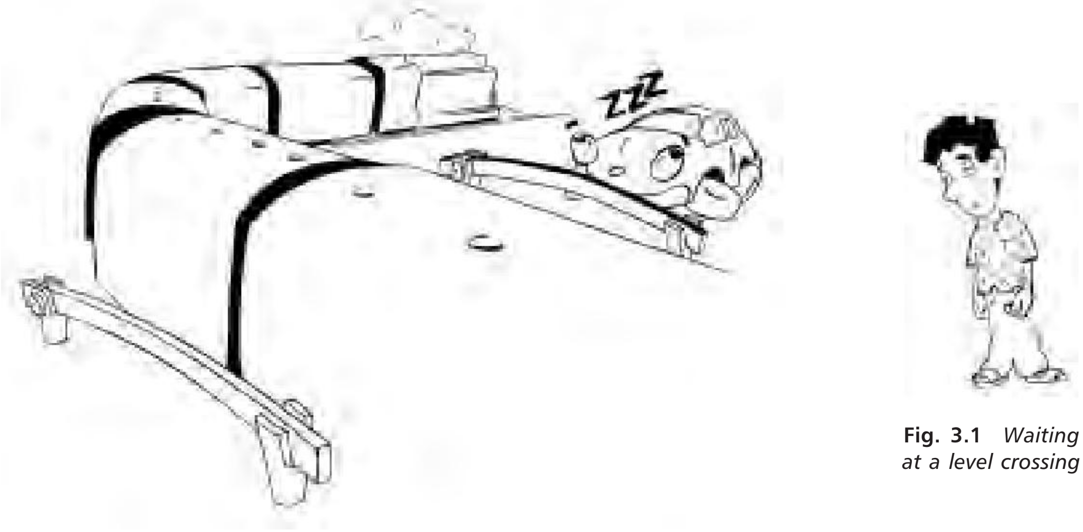


Fig. 3.1 *Waiting at a level crossing*

Now let me give you some hope. Suppose there were a way of de-coupling all the wagons and attaching an engine to each wagon. Now you would have trains which were just one wagon in length. As long as there were no cars waiting at the level crossing , all these mini trains would pass through one by one. But the moment you come along in your little Maruti (Fig. 3.2), the guard at the level crossing asks the trains to wait just a bit, and opens

Fig. 3.2 *Converting the train into packets*

the gate so as to allow you to go through. After this he shuts the gate and the trains continue. So the time you spend waiting in your car is minimised.

Try and understand what we have done here. We have converted the train into a lot of little "packets", each packet consisting of one wagon and its associated engine. Each packet goes across the track, but if there is someone else wanting to move, he doesn't have to wait much (incidentally, that someone is also a packet—in this case a Maruti packet).

Can you see how this concept can be extended to the Internet? (If you can't, just go back to the beginning of this chapter and start all over again!)

Long messages are split into packets. Each packet has an additional section called the header (like the engine) which contains information such as the IP address of the sender and receiver. And each packet moves independently over the Internet (see Fig. 3.3). No long message is able to hog the Internet, so no one is forced to wait for ever while a lo-o-o-o-o-o-o-o-o-o-o-ong message is being sent by someone else. And of course, it makes optimal use of the network because no one blocks resources.

My friend, let me now share something truly momentous with you. If you have understood this example, you have understood a very fundamental concept on the Internet. And this concept is called **packet switching.**

But there is more—after all you must hold forth at your next party—so read on!

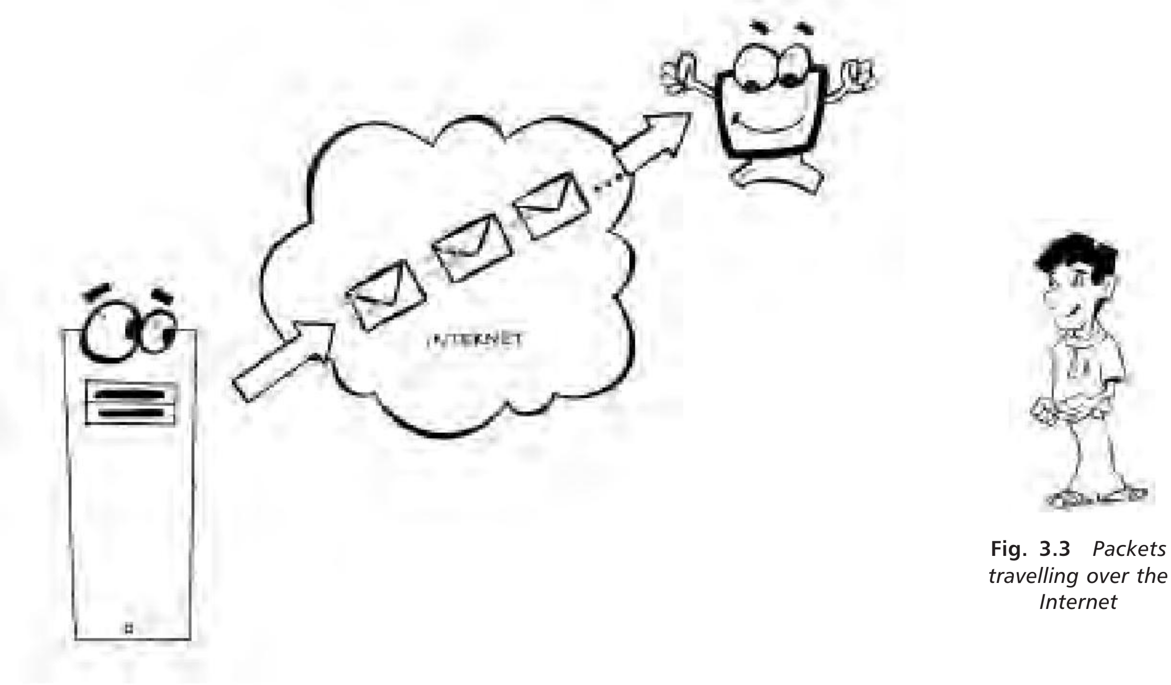


Fig. 3.3 *Packets travelling over the Internet*

HANDLING TRANSMISSION ERRORS THROUGH PACKET SWITCHING

Those of you who have had the good fortune from time to time, of picking up a phone and making a call, would know that a large amout of time is spent in saying

"Hello,....... Hello,........ Hello,.......... Hello,........ HELLO!!!!"

And in doing so, you are not being over-friendly but are simply attempting to understand what the other person is saying over a phone line that is very, very unclear.

In fact most of the time, you probably hear everything except the person you want to hear at the other end, simply because the phone lines are so horribly noisy.

Add to this the fact that the Internet too uses similar communication lines, and you can see the problem. You can transmit a message over the Net, but the chances are that there would be a huge number of errors in this transmission (because of the noise, a zero would be mistaken for a one and so on).

So what do you do?

Obviously it is fairly complex to find out where exactly the error occurred.

(A noisy phone conversation might look something like this:

"Hey I couldn't catch the fourth word in your last sentence."

"What?"

"The fourth word in your last sentence."

"The fourth word? Do you expect me to keep track of every word I say?"

"No, no, but just give me the fourth word again."

"Well, let me see if I remember............"

etc. etc. etc.)

So you have only one choice—transmit the entire message again.

And this is where the fun starts. You have probably heard of something called probability theory. It's the branch of mathematics where you figure out the chances of your wife being at the same dance party as you, when you go there with your girl friend! Most mathematicians use far more mundane examples but the concepts are the same.[1]

Probability theory also states that the chances of error in a message increase dramatically if the message is long. So if the probability of an error in a 1-bit message is 20%, the probability of an error in a 2-bit message is 36%, the probability of an error in a 3-bit message is about 49%, and so on and so forth. The exact percentages here are unimportant—they have been put in simply to let you know that I understand probability theory. But what is important is that the chances of error go up if the message is longer.

Not only that, in long messages, you also have a high probability of getting not just one error but several of them, just to add to the fun (Fig. 3.4).

So now what happens when you have a long message transmitted over the Internet?

Very simple—you get lots of errors!

You re-transmit the same message—and good old probability theory hits you with several errors all over again, ... and again, ... and again, ...

But hold it. You've just seen that probability theory is lenient with short messages—and therefore the chances of an error in such messages are fairly low.

[1] For our lady readers, as well as for the lucky, unmarried ones, similar examples could be constructed to explain probability theory, but in the interest of brevity I'll leave those to you.

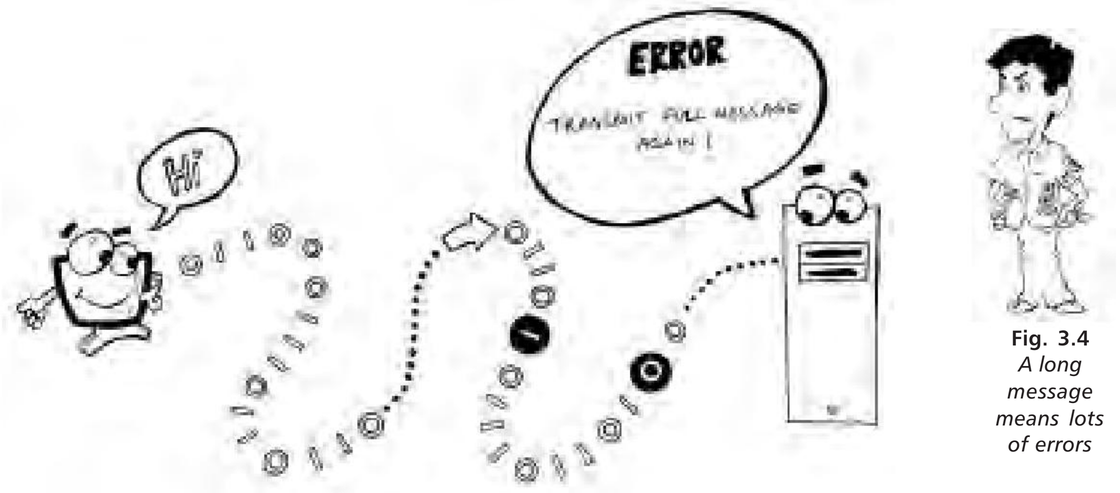


Fig. 3.4 *A long message means lots of errors*

So now you bring in our good old packets. Split a long message into little packets, and transmit each of the packets independently. Now all you need to do at the receiving end is to check each packet for errors. Given good old probability theory, most packets would arrive in good shape, and only a few of them would have an error—or errors (Fig. 3.5).

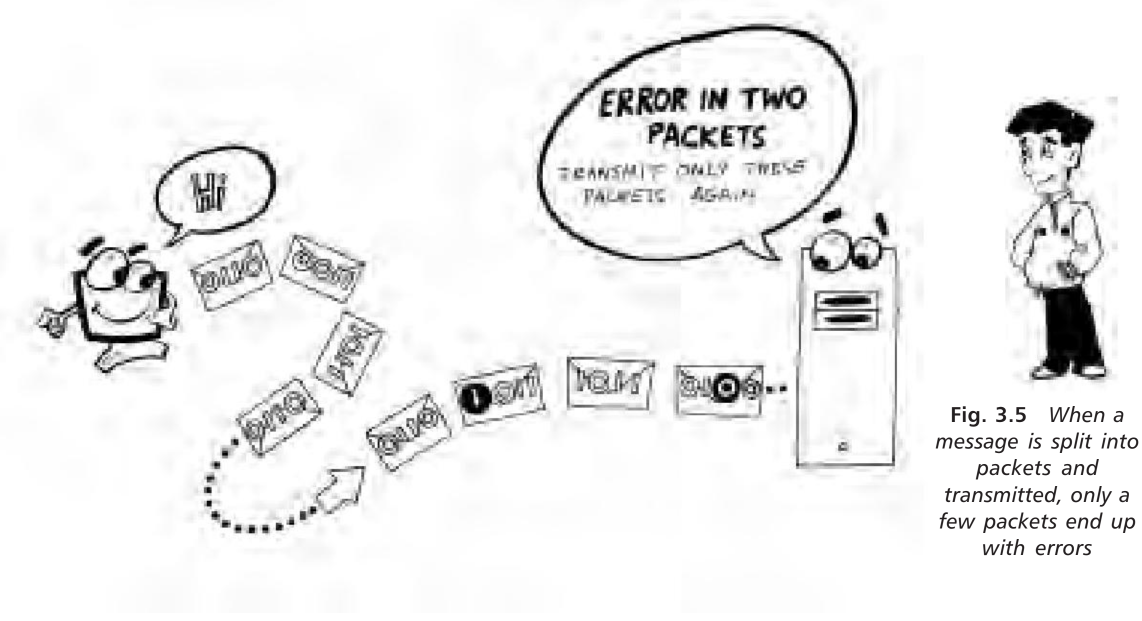


Fig. 3.5 *When a message is split into packets and transmitted, only a few packets end up with errors*

What do you do with the few packets that have errors in them? No problem—just transmit them again. *But not all packets—only those that have an error in them.* Once again, you may have errors in some of these re-transmitted packets, but that would again be a small fraction of them, so perhaps in a couple of such attempts you have got the entire message transmitted across perfectly.

(Don't you wish you had such a simple probability-based solution to the problem of your wife catching you at a party with your girl friend??)

So far so good. Now how do you figure out whether or not a packet has an error in it? For instance when you transmit 11001100 and the message received is 11001101, how does the system figure out that there is an error? (This could be fairly critical—11001100 could be the code transmitted by the US President, saying, "Dismantle Nuclear Warheads", whereas 11001101 could mean "Fire away, me lads!!!)".

Incidentally, the following section is not too critical to understanding e-Commerce, so if you are happy with the fact that there is a way to check for errors, just skip it and go on to the next section. But if you are the strong, silent type, determined to show the world, just go ahead............

To understand how errors can be detected, let's take a simple message being transmitted, such as '5648'. Now let us find the sum of all the digits in this message. In this case, we get the number 23. Take the last digit of this sum, which happens to be '3'. We will call this additional digit a **checksum** because it is derived from the **sum** of all the digits in the message, and is used to **check** for errors in transmission. Now append this '3' to the original message, so as to get a slightly longer message, namely '3-5648'. Finally, transmit this composite message.

At the receiving end, repeat the same procedure. Namely, separate out the original 4-digit message, and add the digits again, to get a new checksum.

And now we come to the key. Suppose there is an error in the message during transmission, and one of the digits in the message is changed. For instance we receive the message 3-5647 instead of 3-5648. Then the new checksum calculated at the receiving end would not be '3' but '2', which is different from the checksum transmitted.

In other words, if the two checksums tally, there is no error in transmission. But if they don't, we have a error...................and therefore the system requests a re-transmission of this packet (Fig. 3.6).

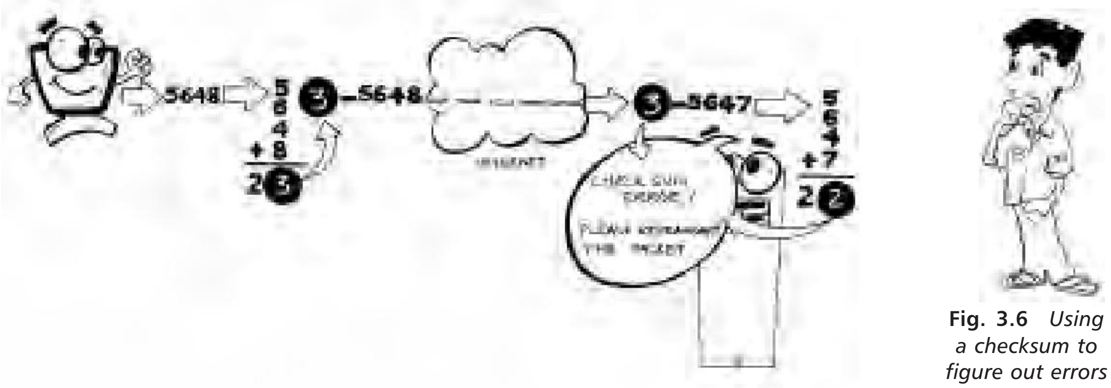


Fig. 3.6 *Using a checksum to figure out errors in transmission*

EFFICIENT ROUTING USING PACKET SWITCHING

Is that all there is to packet switching?

Well, not quite—there is still a little more. After all, networking engineers don't get the fat salaries they do get for nothing!

Let us take another analogy. Suppose you were in Delhi, and were travelling with lots and lots of friends from Greater Kailash to Connaught Place (These are two well known colonies in Delhi. If you are familiar with them, great. If not, too bad! The purpose of this book is to explain e-Commerce and not the geography of Delhi. However, do not despair. I strongly suspect that even if you live in Jhumri Talaiyya, you can still build the same logic to understand packet switching).

So to get back to our trip, let us also assume that each one of you had your own car. Now to reach Connaught Place you would normally take a road called Dr Zakir Hussain Road. So you go ahead and merrily hit this road.

But wait—you almost hit a bull as well. This bull believes that it is siesta time and that it has sole rights to that wonderful bed called Dr Zakir Hussain Road. And naturally, no one has the guts to disturb the bull. So two of the three lanes available for traffic are blocked.

What's the result? Naturally, traffic moves at a snail's pace.

So what do you do?

Well, you would do exactly what the cowboys do in Western movies. Quick as a flash, you reach for your holster and whip out your mobile phone. And assuming you don't get hauled up by a friendly neighbourhood policeman for using a mobile phone while driving, you call your friends (who incidentally, have had the good fortune of finding a traffic signal that didn't work and are therefore stuck well before Dr Zakir Hussain Road).

If your friends have any sense (sometimes in life we do need to make such sweeping assumptions), they would now avoid Dr Zakir Hussain Road, and instead come via Barakhamba Road.

However, after three friends have gone through, this road too, gets choked (remember we are talking about Delhi, not Jhumri Talaiyya) so the next four friends come via the Andrews Ganj crossing. And so on......................

I do hope you are getting the idea. There are multiple paths that you can take between two points, and different cars take different paths depending on the traffic.

Do you see light?

No?

Okay, let us assume each car were a packet and these packets were travelling on the Internet instead of Delhi roads.

Now do you see light?

Of course you do! When a message is split into packets, different packets can travel to the same destination via different routes depending on the traffic conditions (see Fig. 3.7). That's the other beauty of packet switching—if you have a large message, the packets could take different paths so the message arrives faster, instead of all packets of that message travelling on one path (thereby choking it and slowing down transmission).

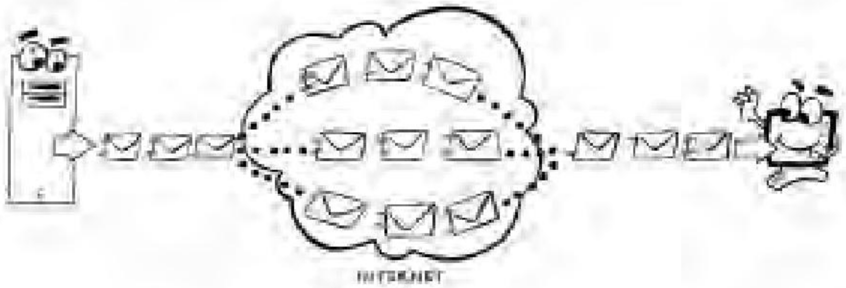


Fig. 3.7
Different packets taking different routes towards the same destination

Incidentally, every node in the Internet has a special computer called a **router,** which decides the path each packet takes—after looking at the destination address of the packet.. Some routers are **static,** which means that as long as the end destination is the same, they will always send the packet along the same route. Therefore static routers do not re-route packets based on current traffic conditions. Other routers are **dynamic routers,** which means that they check traffic conditions and then determine the path for a packet. Naturally, **dynamic routers** make more efficient use of the Internet.

TRANSMISSION CONTROL PROTOCOL, OR SIMPLY 'TCP'

Wonderful! We have just solved the problem of traffic congestion. On the Net of course. But that naturally leads us to another problem (why is life so full of problems?). The first car to leave Greater Kailash may be the fifth one to arrive at Connaught Place, the second car to leave may be the third to arrive, and so on.

No problem with cars. But what about packets on the Internet? Can you imagine what would happen to a wonderful message like

"I will marry you! Tell your parents not to worry."

if the packets (and therefore the words) were to arrive in the wrong sequence, such as:

"I will not marry you! Tell your parents to worry."

So you get the idea!

And therefore at the destination end, there needs to be someone who looks at the packets as they arrive, and re-assembles them in the correct sequence. So you get the original message all over again (and therefore get happily married in this case).

Terrific. You have understood packet switching, my friend. At the same time you would also have noticed that it requires several functions to be performed. Namely:

(a) Each message needs to be split into small packets
(b) A checksum needs to be computed and appended to each packet
(c) At the receiving end, each packet needs to be checked for errors
(d) If there is an error in a packet, it needs to be re-transmitted
(e) Someone needs to check that all packets transmitted have in fact arrived. If not, the lost packets need to be re-transmitted
(f) And finally, the packets need to be put together in the right sequence, so that the original message reaches safe and sound.

Lots of things to do!

So who does them?

What we need is a method of doing all this, which has been agreed to by everyone (whether server or client) on the Internet. In other words, we need a protocol which works at both ends and performs all these functions. And this one is called **TCP,** or the **Transmission Control Protocol.**

On the Internet, TCP and IP go hand in hand. So TCP is a protocol that creates packets, checks for errors and re-transmits them if necessary, and recombines them in the correct sequence to get the original message all over again. And IP makes sure that each packet goes to the correct destination, namely the destination IP address.

Since these two always work together on the Net (rather like Laurel and Hardy), they are usually referred to as **TCP/IP**. However, TCP/IP is not just one single protocol. It is actually a set of protocols, usually known as the **TCP/IP suite** of protocols, which we will meet in the next section.

THE TCP/IP SUITE OF PROTOCOLS

One last thing. We have seen three protocols so far, namely HTTP, TCP, and IP. How are these related to each other?

Let us digress for a moment and talk about receptionists and secretaries (much more fun, isn't it?)

Assume that you are a manager and wish to talk to a manager in another organization. Would you pick up the phone and dial?

No way! You would phone up your secretary and tell her (or him, if you are unlucky enough to be married to a tough, no-nonsense wife).

Now what does your secretary do? If she is senior she would not call up either. She would ask the receptionist (the junior-most employee in the office who keeps polishing her nails and dreaming of her boy friend), and the receptionist responds by calling up the receptionist in the other organisation.

This is critically important. *Protocol demands that the receptionist talks to the other receptionist and not to the secretary or the manager.* Now that the two receptionists have got over the initial hurdle of getting through the telephone company and establishing the phone connection, they pass the call on to the respective secretaries. The secretaries, in turn, do a bit of chit-chat, discuss the latest gossip, and then pass on the call to their bosses, who then talk (or shout, depending on the current state of relations between the two organisations).

Notice that we have three levels of activity at both ends (see Fig. 3.8). The bosses talk business, but for that they first need to get their secretaries to get them on the line. The secretaries ensure that each boss is in the office and is willing to talk at this point in time (for instance, the called boss should not be on the other line with his "other" girl friend). But they in turn need the receptionists to fight the real battle with the telephone company and make the actual phone call.

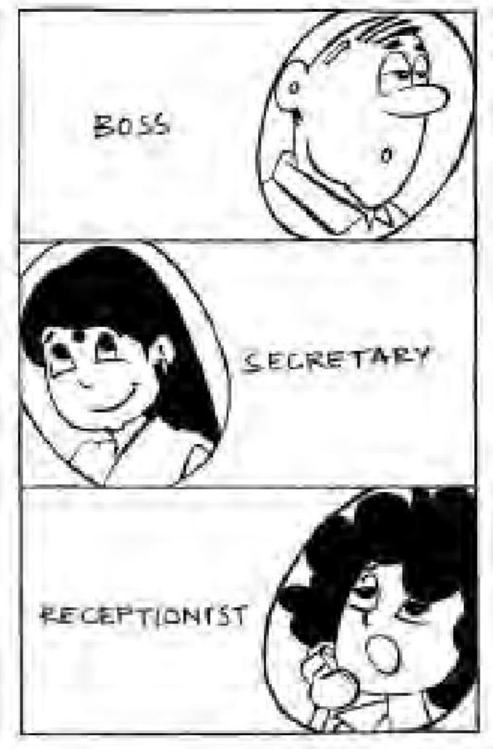


Fig. 3.8 *Three levels of protocol—the boss, secretary, and receptionist*

Three different people. At three different levels. Each required to perform a different function. And each talking only to his or her counterpart at the other end.

Now finally let us name each of these three people. We call the boss "HTTP", the secretary "TCP", and the receptionist "IP".

So we have three levels of protocol, each taking care of one function, and communicating with the corresponding protocol at the other end.

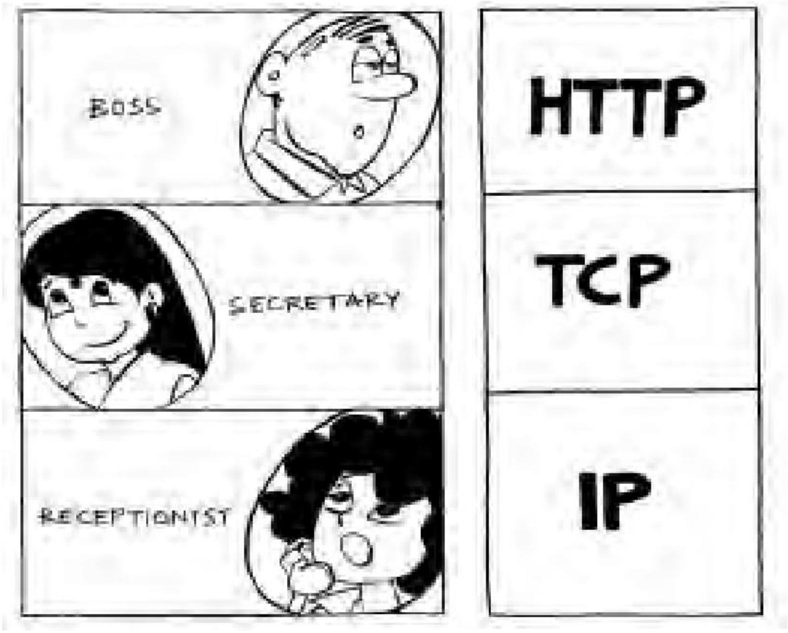


Fig. 3.9 *Three levels of protocol—HTTP, TCP, and IP*

Get the idea?

That, my friend, is the TCP/IP protocol suite (Fig. 3.9) !!

Actually, we're not quite finished. There is a little more to the TCP/IP suite.

Let us go back to our example of bosses, secretaries, and receptionists. Let us also assume we're making a trunk call, across cities. And let us also go back several years in time when we did not have automated telephone exchanges. So when the receptionist had to make a trunk call to another city, she did not dial a number. Instead, she would call up the nearest telephone exchange, speak to the operator in at the exchange, and ask her for the desired number. This operator, in turn, would call up the exchange operator in the called city to establish the connection. This second operator would then call the office of the called party and speak to the receptionist. And so on.........

Therefore, we actually had four levels of protocol. Namely the boss, the secretary, the receptionist, and the telephone operator. And each would talk to his or her counterpart at the other end.

By the same logic, on the Internet we again have a lower level of protocol. We call it the **Physical layer** of protocol. This layer has to do with hard core Electronics Engineering—what voltages are required to be transmitted, what kind of cabling is required for the communication to happen, etc.

Since it is Electronics Engineering, I assume your interest has flagged. Therefore we will not cover this subject in this book. However the point is that we actually have four levels of protocol, as shown in Fig. 3.10.

And that's it. That completes our TCP/IP suite—simplified as usual of course. As you can see, it is not one single protocol but a set of protocols, sitting one on top of the other, and communicating only with the corresponding protocol at the other end.

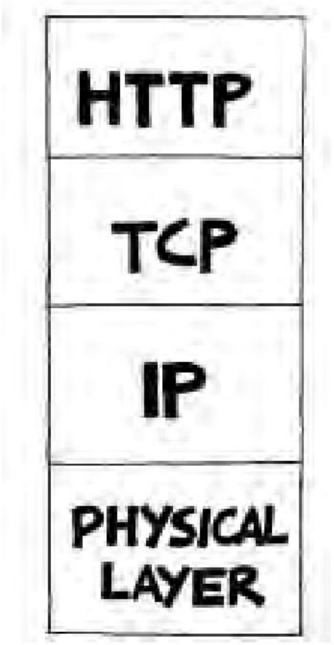


Fig. 3.10
Four levels of protocol—HTTP, TCP, IP and the physical layer

SUMMARY

So we come to the end of the chapter. Where you learnt all about packet switching—the process of splitting up a message into small packets, transmitting these packets independently, and then putting them together again at the receiving end.

We saw that packet switching permits us to share communication lines instead of blocking them for one sender and receiver—thereby using them very efficiently. We also saw that this cuts down the need to re-transmit messages because of errors during transmission. And finally, we realised that different packets in a message can take different routes to the same destination—ensuring that problems due to traffic congestion are minimised.

All this gets managed by a protocol called TCP, or the Transmission Control Protocol.

Surely you also remember that TCP and IP go hand in hand (rather like you and your girl friend) and are referred to as TCP/IP. In fact, TCP/IP is a suite of protocols, which includes the physical level as well as the highest level protocol, namely HTTP.

At this stage, if we were to re-visit our e-Commerce architecture, it would look something like Fig. 3.11.

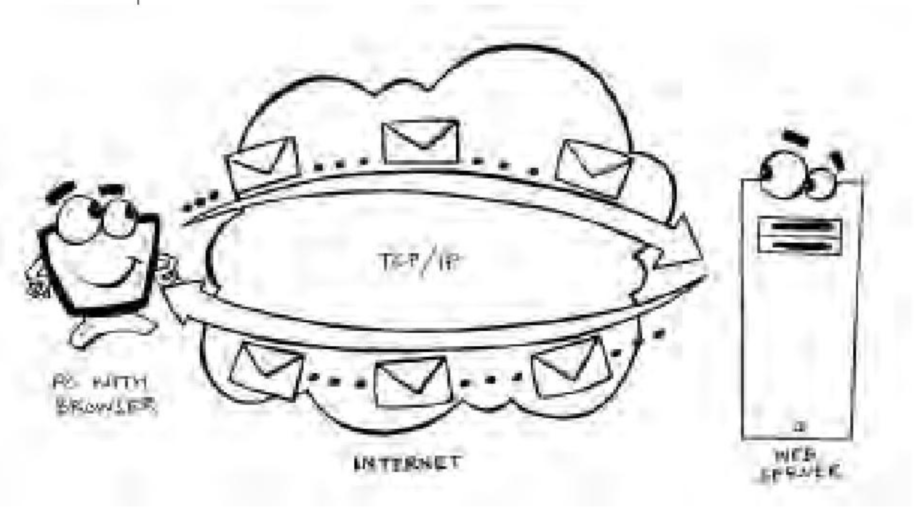


Fig. 3.11 *The e-Commerce architecture with TCP/IP*

And that, my friend, is all we have for you in this chapter. Close the book (don't forget to put in a bookmark or at least fold the page) and get back to sleep. Just make sure you wake up in time for the next chapter!!!

4

Internet Addressing, the Easy Way—The Domain Name System

You would recall from Chapter 2 that computers on the Internet are addressed using IP addresses. If you remember that, please proceed. If you don't, you need to go right back and go through the chapter all over again......and again......and again, until you remember it.

Assuming you do remember this astounding fact, let me ask you a question:

Are IP addresses easy to remember?

Imagine for moment what would happen if we human beings were to have IP addresses as names! A typical conversation may run something like this:

Visitor : "What have you named the little one?"
Proud mother : "Oh, we call him 27.34.200.144"
Visitor : "How sweet ! What does the name mean?"
Proud mother : "Actually it's a Sanskrit name, meaning wisdom"
Visitor : "But how do you remember his name?"
Proud mother : "We're planning to tattoo it on his arm"

Obviously it's impossible to remember more than one or two such IP addresses. It's much, much easier to remember a name like amazon.com or cricinfo.org or tomcruise.co.uk (especially tomcruise.co.uk, if you happen to be one of our lady readers).

Fortunately, for once the computer people and the human beings agreed. So that is the kind of addressing people on the Internet use—cricinfo.org or amazon.com. Such a name is called a **domain name.** And for the next several pages, that is exactly what we are going to discuss—domain names and how they work !

THE DOMAIN NAME SYSTEM OR 'DNS'

By now you must have realised that I am very fond of analogies. And to understand the domain name system, that's exactly what we will use—yet another analogy.

Assume that you lived in Jhunjhunu in Rajasthan, and you had to get your sister married (or your cousin, or niece, or aunt—it doesn't really matter which one !). And someone has told you of a wonderful young man called Rajan Sinha, who is currently working with the YetAnother-Cola company in their branch office in San Fernando, a city in Venezuela. He is tall, fair, *very* handsome, highly qualified, earns in five figures, no bad habits—a great match for your sister who is also *very* fair, extremely beautiful, a double MA, a wonderful cook, with a terrific family background, etc. etc. etc.........what a unique couple the two would make!

So as a dutiful brother, what's the first thing you would do? Naturally, locate this young man and check him out with someone who knows him.

How would you do this?

Well, your immediate impulse would probably be to catch the next flight to Venezuela (remember all those Miss Universe contests?)

But that's slightly impractical isn't it? So the next best thing is to ask someone.

And that's easy—you go across to the nearest YetAnother-Cola distributor in Jhunjhunu, and ask him,

"Sharmaji, aap YetAnother-Cola company mein kisi Rajan Sinha ko jaante hain? (Sharmaji, do you know anyone named Rajan Sinha who works in the YetAnother-Cola company?)"

"*Kaun?* (Who?)"

"*Rajan Sinha.*"

"*Kahan rehte hain woh?* (Where does he live?)"

"Venezuela !"

"*Yeh kahaan hai—Dilli ke paas hai*? (Where is this place—Is it anywhere near Delhi?)"

So you get the idea. There is a remote possibility that the distributor in Jhunjhunu will know of Rajan Sinha in Venezuela, but it's very, very unlikely. All you can expect him to know is the names and whereabouts of his own employees in Jhunjhunu!

So what do you do?

Actually it turns out that people in the YetAnother-Cola company are very helpful and therefore you don't need to do anything. The distributor at Jhunjhunu, on his own, passes on the enquiry to his boss, the branch manager of the Rajasthan branch office in Jaipur (see Fig. 4.1).

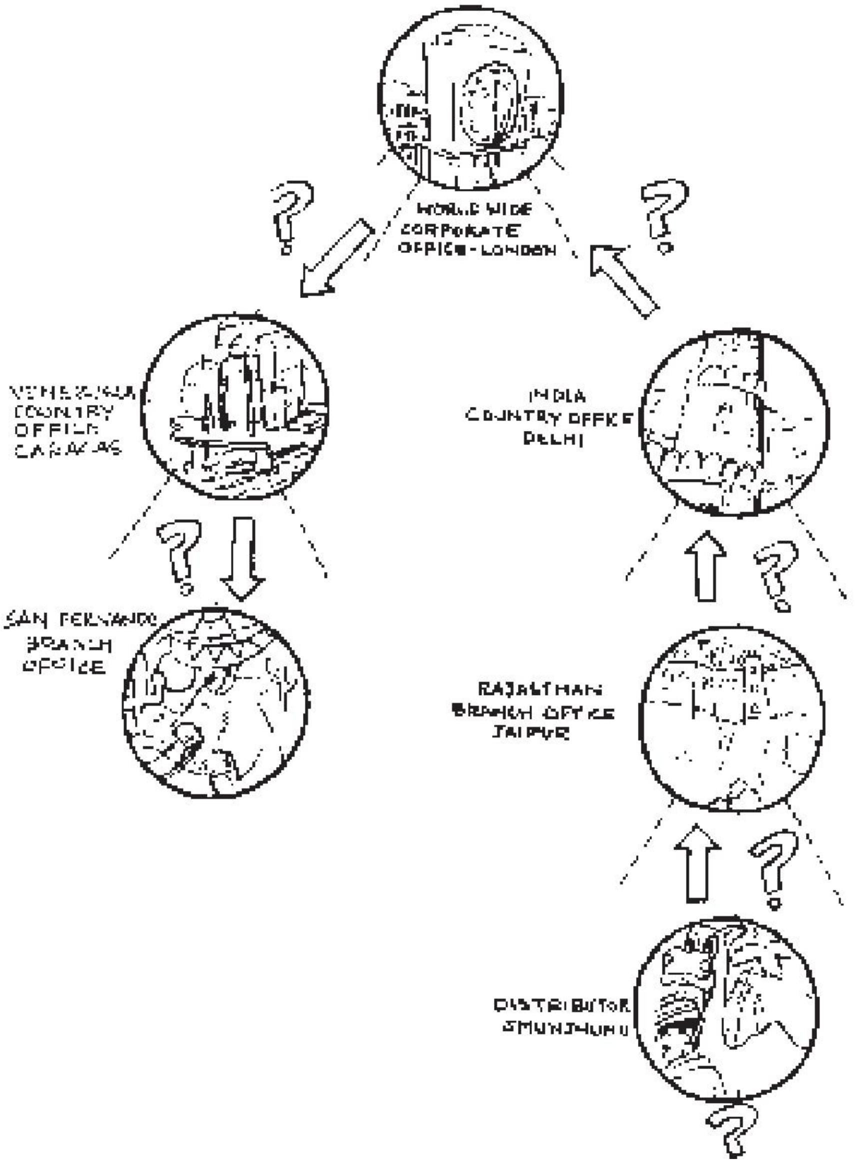

WORLD WIDE CORPORATE OFFICE-LONDON
VENEZUELA COUNTRY OFFICE CARACAS
SAN FERNANDO BRANCH OFFICE
INDIA COUNTRY OFFICE DELHI
RAJASTHAN BRANCH OFFICE JAIPUR
DISTRIBUTOR JHUNJHUNU

Fig. 4.1
Searching for Rajan Sinha

The branch manager is equally helpful (after all, it is a question of honour—a daughter of Rajasthan marrying someone in the YetAnother-Cola company). But unfortunately he hasn't heard of Venezuela either (you see, his wife doesn't allow him to watch all those "distracting" Miss Universe contests). He only has information about all the YetAnother-Cola distributors in Rajasthan (who report to him). And of course he has information about all the YetAnother-Cola employees who work with him in his branch office.

So he passes on the query to *his* boss, namely the country manager of the India head office in Delhi. Fortunately this gentleman is also keen to help. Unfortunately, he only maintains information about branch offices in India, and about each employee in the India head office in Delhi.

He therefore informs the branch manager at Jaipur that he (the country manager) was never too strong at geography in school, but that his (the branch manager's) request was worthy of action, and that he (the country manager) would do his (the country manager's) best to help. He (the country manager) added that he (the branch manager) need not worry and that he (the country manager) would pass on the query to his (the country manager's) boss, who happened to be the head of YetAnother-Cola world-wide, and sat in their world-wide corporate office in London.

Strangely, the world-wide head of YetAnother-Cola finds the time to help in this major matrimonial initiative. Naturally, he cannot be expected to know the names of all the employees in all offices world-wide (of course, being a good human manager, he does know all the employees in the corporate office). But he does the next best thing. He obviously knows where Venezuela is, and he passes on the request to his subordinate there, namely the country manager of YetAnother-Cola, Venezuela, at Caracas (the Capital of Venezuela, in case you have forgotten your class VII geography).

The country manager at Caracas is obviously aware of the San Fernando branch office but he in turn cannot be expected to know the names of all the employees there (once again he knows the employees in his own country office). So he directs the query in turn to his subordinate, the branch manager at San Fernando.

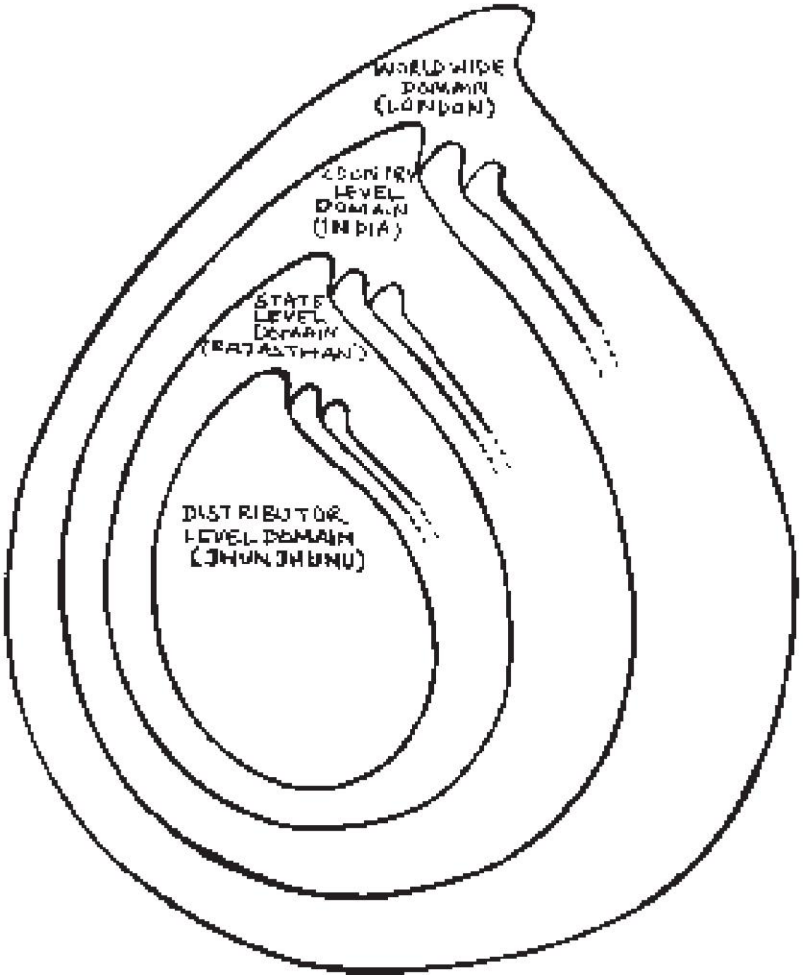
And here you finally get lucky. The branch manager at San Fernando does know Rajan—after all he works in his branch office. So within a few days you get a response from this branch manager:

> "Yes, Rajan Sinha does work for our San Fernando branch office. An excellent chap, doing very well, and likely to be promoted this year."

"He lives with his girl friend who is from Paraguay, and they have two very cute children. We hear they plan to marry this year !!"

Now, dear reader, you could have one of two possible reactions to this message. If you're like most humans, you would probably feel relieved that you got to know all about Rajan Sinha before your sister married him. On the other hand, if you are completely non-sentimental, and your only interest is in learning e-Commerce, you would realise something very fundamental here.

The YetAnother-Cola company, or for that matter any other large organization, is structured as follows (see Fig. 4.2):


WORLD WIDE DOMAIN (LONDON)
COUNTRY LEVEL DOMAIN (INDIA)
STATE LEVEL DOMAIN (RAJASTHAN)
DISTRIBUTOR LEVEL DOMAIN (JHUNJHUNU)

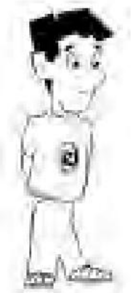


Fig. 4.2 *The YetAnother-Cola company structured into domains*

At the highest level, there is the world-wide organisation itself—so that forms the largest group. Let's call it a **domain**. This domain contains several smaller country-level domains, such as India, China, and Venezuela. Next, each country-level domain contains several smaller state-level domains. For instance, the India domain would contain state level domains such as Rajasthan, Punjab, and Tamil Nadu. Each state-level domain contains several distributor-level domains—such as Jhunjhunu in the Rajasthan domain.

Importantly, the organisation has many, many employees, and these could be located in any domain. And grouping the organisation into domains gives us an interesting way to specify the address of any employee. For instance, if we had an employee called Chauhan working with the distributor in Jhunjhunu, you could specify his address as follows :

Chauhan . Jhunjhunu . Rajasthan . India

As you can see, we give the name of the employee, followed by the domain where he is located, followed by the next level domain, and so on. Similarly, we could refer to our friend Rajan Sinha by using the following address :

RajanSinha . San Fernando . Venezuela

You would also notice (I hope you do!) that each domain has an office with a manager. This manager maintains information about all the employees in his office. And all these managers form a kind of hierarchy, with the manager of any one domain reporting to his boss, the manager of his next-level domain.

So how do you locate an employee?

Well, you start with the domain where you are located, and speak to the manager there. So if you happen to be in Jhunjhunu, you catch the distributor there and ask him about Rajan Sinha. Since Rajan Sinha does not work in his office, he directs you to his boss—namely the branch manager of the Rajasthan domain. Since this manager does not maintain information about Rajan Sinha either, he directs you to *his* boss, namely the country manager of the India domain. And so on till you reach the manager of the San Fernando branch office, where the manager does know all about Rajan.

Can you now see why we created these multi-level domains?

Obvious, isn't it?

So that information about any employee need only be stored in the office where he works. Otherwise information about all employees across the world, would have to be stored in the London corporate office. What a phenomenally huge corporate office that would require !!! And all for no reason—after all, the world-wide corporate office really doesn't need to maintain information on the employees at the branch office in Jaipur. Similarly, the India head office at Delhi does not need to maintain information about all the distributors' sales people in every state.

Further, you might be in the branch office in Coimbatore in Tamil Nadu, India, but to find out information about another employee in the same branch office, you would need to send your query all the way to London! In other words, the primary function of the corporate office in London changes from running a successful company, to answering matrimonial and other queries.

Now let's shift from the earlier marriage proposal to the Internet.

Can you see light ?

Of course you can !!!

Just replace the word *employee* with *server* and you've cracked the problem. Servers on the Internet are grouped into domains in exactly the same way as employees were in the YetAnother-Cola company. And they are also addressed in the same way. So for instance, the site

hrithik . bollywood . co . in

refers to the web server called *hrithik*, which is located in the domain *.bollywood*, which is further located in the domain *.co*, which is finally located in the domain *.in* (see Fig. 4.3).

This method of naming servers, so you don't need to tattoo the names on the user's arm, is called, rather unimaginatively, the **domain name system** or **DNS** for short. So when you want to create your web site, you would need to think of an appropriate domain name, hope that no one has thought of it before you, and get it registered. Registration is done by several organisations, but ultimately they must go to one of a few, such as InterNIC.

There is another key reason why domain names are more useful than IP addresses. You may, for some reason, change the web server on which you have hosted your site. (The reasons could be several—you may have found a cheaper hosting service. Or had a fight with the current host. Or simply didn't like his face).

Naturally, this new web server would have a different IP address from the earlier one. So what would you do when this happens? How would you inform all your customers? And even if you did inform them, they had a tough enough time remembering the old IP address, without having to re-learn it every time you changed your hosting service !

But the domain name? Ah, that my friend, can remain the same. Hosting services may come and hosting services may go, but the domain name stays on for ever!

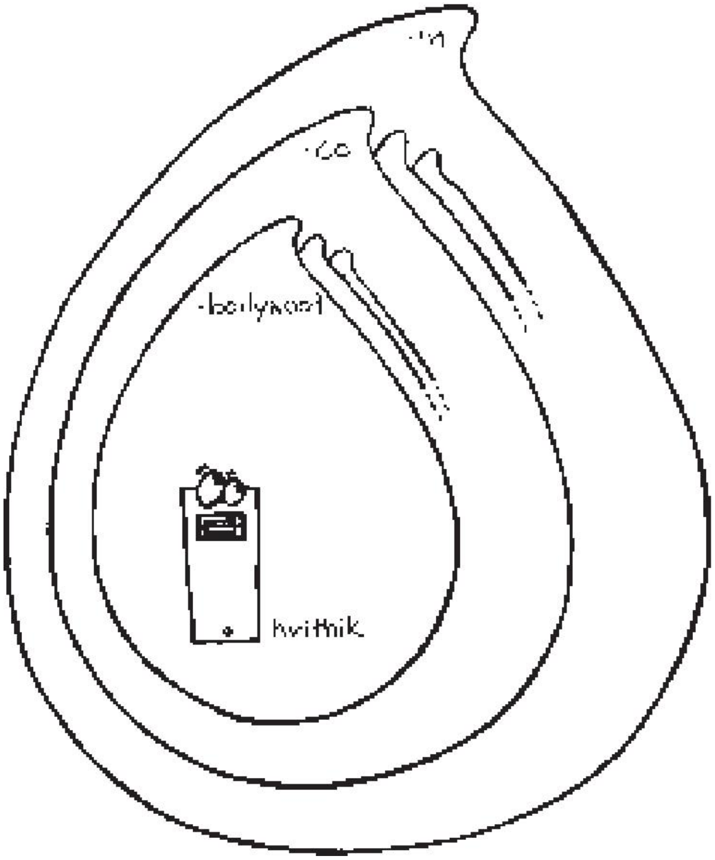


Fig. 4.3 *Domains in the Internet*

THE DNS SERVER

Next question. Who plays the role of the managers at various levels? Naturally it cannot be humans in this case, so they are, as expected, special servers. Once again, without using too much imagination, they have been called **DNS servers** or simply **name servers**.

And what do these managers do? Instead of maintaining details about employees, as in the matrimonial example earlier, they maintain details of servers, which in this case means the IP address.

So how do you locate a domain name and therefore find out its IP address?

Simple. Your browser goes to the domain name server closest to you and checks. If the domain name you have requested is found there, great ! If not, the request is passed on to the domain name server in the next-level domain—and so on. Exactly the way you searched for Rajan Sinha. Incidentally, this process of finding the IP address for a given domain name is called **domain name resolution**.

Finally, when the IP address corresponding to this domain name is located, it is sent back to the browser, which then puts out an HTTP request for this page.

As you can see (I hope you can), getting a page on the web is actually a two-step process. In the first step we resolve the domain name and get the IP address. In the second step we get the web page from this IP address.

TOP-LEVEL DOMAIN NAMES

If you are a reasonably avid Internet surfer, you would have realised that the largest (rightmost) domains in any domain name are standardised, and there are just a few that you can choose from. For instance .com, .edu, .mil, .in, or .uk. These are called **top-level domain names** (since they refer to the largest possible domains), and they are of two kinds. First of all there are the country specific top-level domain names, as shown in Table. 4.1.

Table 4.1 *Top-level domain names for each country*

Domain Name	*Country*
.uk	United Kingdom
.in	India
.jp	Japan
.sg	Singapore
.my	Malaysia
.fr	France
.eg	Egypt

In addition, there is another category of top-level domain names, which refers to different types of sites. These are shown in Table 4.2. As you can see, sites in the USA do not use the top-level domain for the USA. Instead they use domain names such as .edu, .mil, and .gov.

Table 4.2 *Top-level domain names for categories of sites*

Domain Name	*Description*
.com	Commercial company
.org	Non-profit organisation
.edu	Educational institute (college or university)
.gov	US government organisation
.mil	US military
.net	Network service, such as an ISP

URLs

You have obviously heard this term several times. And you know it has something to do with the Internet. But what?

To understand the term, let's look at a simple URL:

http://www.tomcruise.co.uk

(I am not sure this is the site of the actor, but for the sake of understanding e-Commerce, let us assume it is so).

This URL is divided into several parts. Let us start by examining the first of them, namely "http://".

What does "http://" here mean? Simple. It specifies the protocol that is being used. Which also means that there are protocols other than HTTP, which are used in the same layer of the TCP/IP suite as HTTP.

For instance, if we were sending e-mail over a TCP/IP network, we would not require functionality such as hyperlinks, and therefore would not use HTTP. Instead we would use a protocol called **SMTP** (which stands for **simple mail transfer protocol**—what else?).

In other words, the highest layer of the TCP/IP suite of protocols will vary depending on the kind of application we are running—whether a web application or an e-mail application. For this reason, this highest layer is also called the **application layer**.

Next we have "www".

That's simple isn't it? It simply refers to the fact that the host we want to access is on the world wide web.

But where else can it be?

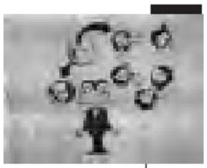
Actually, it could be anywhere. It could even be on your company's local area network or LAN, which is not even connected to the Internet. Obviously therefore, in this case you would not put in the letters "www".

Next we have "tomcruise.co.uk", which is our good old domain name. I'm sure you understand this much by now.

And that brings us to the full form of the abbreviation **URL**. It's **uniform resource locator**. In other words, it is a way to specify the location of any resource. Whether it's on a web server, a mail server, or any other server. Whether it's on the web, or on the local area network in your office.

Actually, if you look at the address line in your browser, you'll see the URL of the page you've reached. Interesting, isn't it?

SUMMARY



And that brings us to the end of the chapter on Tom Cruise.

Well not quite ! It's the chapter where we figured out how to reach the web site of Tom Cruise. Which, for most of us, is as close as we're likely to get to the actor!

This chapter brought out the concept of domain names. Which are English-like names (sometimes very pleasant sounding ones such as Tom Cruise) with which you and I refer to web sites. Of course, each domain name would need to be converted into the corresponding IP address of the web site.

Who does this conversion? We saw that it was done by a series of servers on the Internet, called domain name servers. So when you ask for tomcruise.co.uk, your request goes through a series of domain name servers, till it reaches the server that has heard of Tom Cruise. And gives you the appropriate IP address. After which, of course, you ask your browser to take you to the web site corresponding to this IP address.

How does this change our e-Commerce architecture?

Fairly simple—the modified architecture is now seen in Fig. 4.4. Where the browser and web server remain, but there is an additional series of domain name servers, used to locate the IP address of the web server you want to reach.

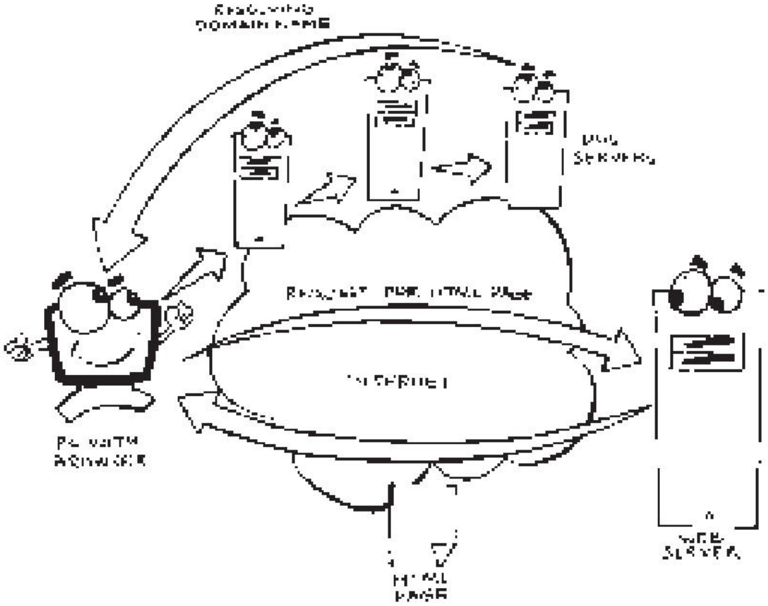


Fig. 4.4 *e-Commerce architecture, with domain name servers added on*

And finally, we looked at the concept of the uniform resource locator, or URL.

Now if you've spent some time surfing you would have noticed that many URLs are as simple as the one we saw in this chapter, but some are far more complex.

And my advice to you here is very simple:

Forget about them—after all you're not a Systems Programmer with Oracle Corporation—you're just a human being wanting to learn e-Commerce.

So forget about them, my friend.

Just remember http://www.tomcruise.co.uk !

5

The Stunning World of Graphics—and Other Media

So you've managed to reach Chapter 5, have you? And that means you're becoming something of an expert !

But do you see something missing? Don't you see that everything we have talked about so far has been text and data? No pictures, no sound, no video?

How boring! We must have pictures, mustn't we? So this chapter, my friend, is dedicated to the wonderful and colourful world of graphics on the Net.

GRAPHICS: THE BANDWIDTH PROBLEM

Who is your favourite film star? Madhuri Dixit? Shah Rukh Khan? Tom Cruise? Clint Eastwood? Hrithik Roshan? Aishwarya Rai?

Instead of getting drawn into a lengthy and unending debate on the pluses and minuses of both Bollywood and Hollywood stars, let us assume it is Hrithik Roshan (for me this has the added benefit of keeping both my wife and my children happy).

So assume that you have created a wonderful web site, and when the user keys in the address, the home page comes with a full blown photograph of Hrithik Roshan—including all those muscles ! Wow !!!

How does this happen? Fairly simple (remember Chapter 1?). You have typed in the address of the site, which points to a web page on some web server. This page is naturally in the HTML format. On the same web server, we have a 'graphic file'containing a scanned picture of Hrithik Roshan.

When the HTML page is downloaded onto your browser, the browser realises that it has a special HTML tag pointing to this graphic file (in other words, the tag would point to the URL of the graphic file).

So the next step is that the browser requests this file, gets it downloaded, and Hey Presto—you have Hrithik on your screen.

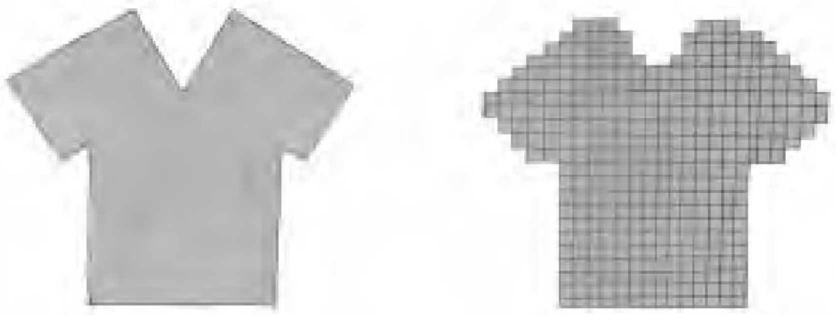
Simple isn't it?

Well not really—there's more, so read on.

How much time does it take to download the graphic, as it is often called?

Let us do a quick calculation (if you always scraped through in Maths at School, or worse, did not manage to scrape through, just skip these calculations and go on to the end of the section).

First of all, remember that any picture in the computing or Internet world is split up and stored in the form of pixels (see Fig. 5.1).



(a) Original picture

(b) Picture split into pixels

Fig 5.1 *All pictures are split into a series of pixels*

A pixel is the smallest unit of a picture—it is a little like the tiny dots you see in newspaper photographs. Each pixel has one colour, and by stringing together a series of pixels, we get a complete graphic. Also remember that effects such as light and shade lead to a change in colour, so it is sufficient to simply talk in terms of colours without getting into complexities of light and shade.

So far so good. In our example, therefore, young Hrithik Roshan would be split into a series of pixels (sounds terrible doesn't it?). And when the graphic file needs to be transmitted from the web server to the browser, all the pixels in the graphic would need to be transmitted—one by one.

How much time does that take?

Well, a small photograph would be about 300 pixels by 300 pixels (this is an approximate number—we could easily have taken a more accurate figure such as 297 pixels by

312 pixels but we would simply have complicated the Maths without, I believe, gaining any additional clarity). So that gives us 90,000 pixels in all, or a round figure of 1,00,000.

Now there is a common standard in the computing industry, which specifies that we need 3 bytes, and therefore 24 bits, to store each pixel.[1]

How many bits does that make the photograph?

It is actually 1,00,000 × 24, or 24,00,000 bits.

And now the fun starts. What is the download speed available to you at the browser end? In other words, what's the bandwidth available (bits that can be transferred per second)?

Most people are delighted if they get a bandwidth of 4 Kbps, or 4000 bits per second on the Internet. And are quite satisfied with a bandwidth of 2 Kbps or 2000 bits per second (it's a question of what you are used to. For instance, take Mumbai traffic at peak time—cars there have a natural aversion to motion of any kind, and people are quite happy if they are able to chug along at about the speed of a brisk pedestrian).

To get back to Hrithik Roshan, therefore, we are talking about downloading a picture containing 24,00,000 bits at a download speed of about 2000 bits per second—and it doesn't require very advanced Maths to convert that into time—1200 seconds. Doesn't sound much till you convert it into minutes—that's about 20 minutes.

Let's pause for a minute to let that sink in. ....

Even the most die-hard fans of Hrithik are unlikely to wait that long—although I must check this out with my children Malvika and Siddharth tonight.

Of course, if you are one of that breed which the world calls a manager, and are therefore fully conversant with modern management techniques such as scheduling and task allocation, you would have a very simple solution to this problem—just have a bath in the meantime ! It would take you about 20 minutes anyway!

[1] For the mathematics whiz kids, this means that each pixel can have any one of 2^{24} or approximately 16 million possible values and therefore colours. For the non-mathematically inclined, it simply means that the number of colours possible is large enough to depict any scene we wish to. In fact this standard is sometimes called "True Colour".

But what if you have to download several such pictures? Where does that leave you? You would look very silly, wouldn't you, going into the bathroom every half an hour, and staying there for 20 minutes each time?

So, my friend, if you want to download terrific muscles of Hrithik Roshan, you have a problem. And no management technique can solve this problem.

COMPRESSING GRAPHICS WITH 'GIF'

Fortunately, however, you also have a solution. It's a technical solution, not a managerial one. And it's called compression.

To understand the concept of compression, we must (unfortunately) leave the subject of Hrithik Roshan for a while and take up a simpler graphic. Let's assume you need to transmit the graphic shown in Fig. 5.2 (why you would want to transmit a silly graphic like this, I haven't the foggiest idea, but let's skip that question for the moment).

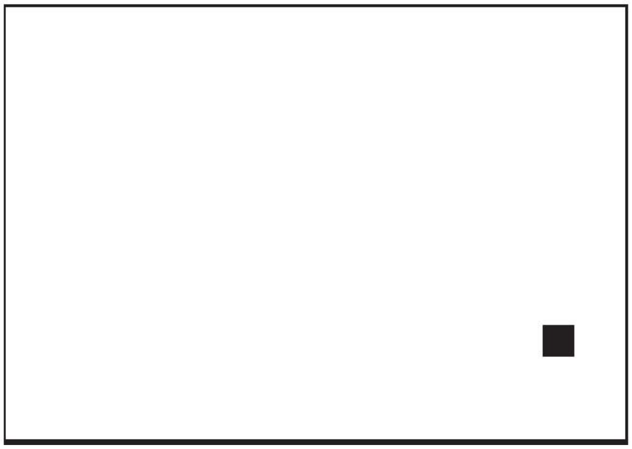


Fig. 5.2 *Simple graphic to be transmitted*

For the purpose of argument, let us assume that the black dot within this picture is exactly one pixel in size—although it may seem larger.

Now if you are smart (obviously you are—you bought this book, didn't you?) you can see that there are two ways of transmitting this graphic.

The obvious one is by splitting it into all its pixels—about 50,000 of them—and transmitting them one by one.

But do you notice something? Most of these pixels have exactly the same colour—and therefore it's the same information that gets transmitted again and again and again.........! Isn't that a terribly inefficient way of transmitting the graphic?

So what's an efficient way? Simply this—transmit the location of the four corners of the figure, and the fact that everything within this is one single colour (in this case white). Also the exact location of the dot, and the fact that it is black—or blue, or yellow or whatever colour it might be.

That's all !!!

At the other end, the entire graphic can be constructed from this information.

And that, my friend, is the key to compression. You don't have to send every pixel across, provided you can send all the information required to reconstruct the original graphic at the receiving end.

Let us now complicate the mechanism a bit, and at the same time complicate the graphic you wish to transmit (Fig. 5.3).

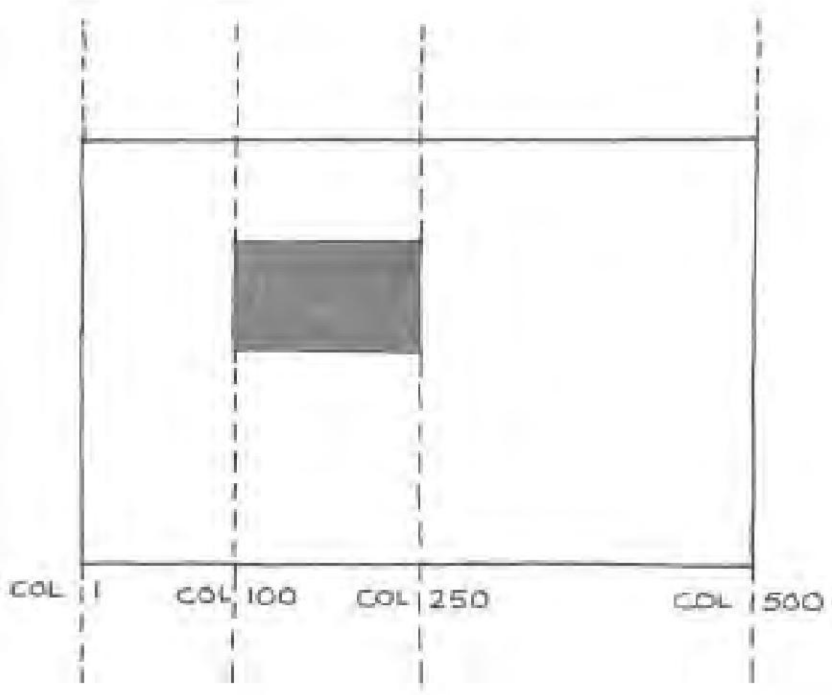


Fig. 5.3 *A little more complex graphic to be transmitted*

In the first row of pixels, since all the pixels are the same, you simply need to transmit the position of the first and the last pixel in the row—in this case columns 1 and 500—as well as the colour.

The same is true for the next several rows.

And after this, from column 1 to 99, you have white pixels, so you transmit the information '1, 99, white'. From column 100 to 249, all the pixels are the same shade of grey, say, shade 4567, so you transmit the information '100, 249, grey shade 4567'. And so on ....

I presume you get the idea. We don't send all pixels, but only sufficient information so that the graphic can be reconstructed accurately at the other end.

This technique of compression forms the basis of the extremely popular **GIF** format, or the **Graphic Interchange Format**, and is accepted as a standard world–wide. Many graphic files that you download on the Net are in the GIF format.

Who does the compression? Simple. When the web page is designed, you would design the graphic using some graphics software such as Corel Draw, or Adobe Illustrator. Most of these standard graphics packages allow you to save the graphic file in the GIF format. Of course, at the other end, you need to reconstruct the original graphic, and that is done by the browser.

Before we move away from the fascinating subject of GIFs, let me ask you one last question. Suppose we had the two pictures shown in Fig. 5.4 on the next page. The pictures are both of the same size and therefore have the same number of pixels. So which one would lead to better compression?

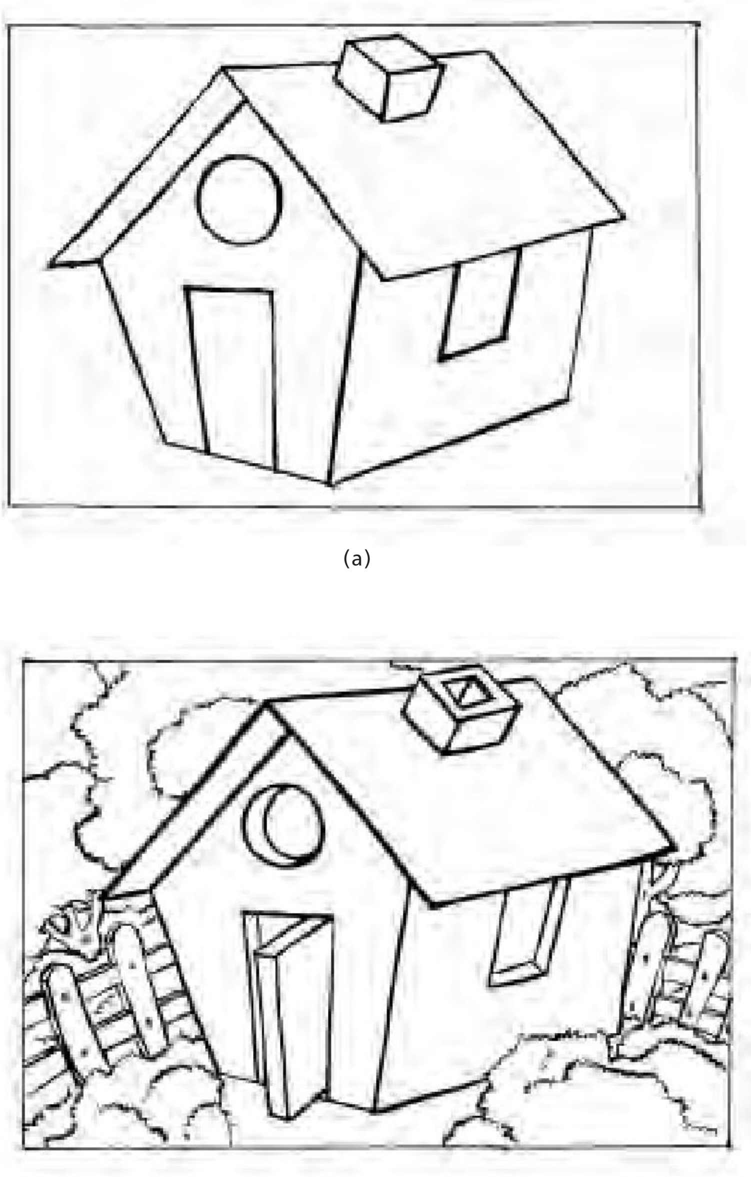
The one on the top, isn't it? Why? Because it has less variations in colour, and by the very nature of the GIF technique, lower variations mean better compression. So next time tell that to your graphics designer—the chap who gives you those phenomenal works of art on your web page—if only you had the patience to download them !!!

COMPRESSING PHOTOGRAPHS WITH 'JPEG'

Wonderful. So now you start downloading a photograph of Hrithik Roshan, all rippling with muscles. And you start nodding your head,.........and then you nod your head some more,.........and more,.........and finally you fall asleep, oblivious to all those muscles.

Why does this happen?

Let's look at a typical Hrithik Roshan muscle. Question: Is it the same colour all through? (Strictly speaking, to answer this question, you'll have to get real close to Hrithik,



(a)

(b)

Fig. 5.4 *Two graphics to be transmitted*

which I'm sure many of you would not mind. But to understand GIF compression, a decent photograph—or even your imagination—will do).

Remember, any photograph, unlike a drawing or a cartoon, will never have one constant colour. And even muscles that ripple have subtle variations in colour (see Fig. 5.5).

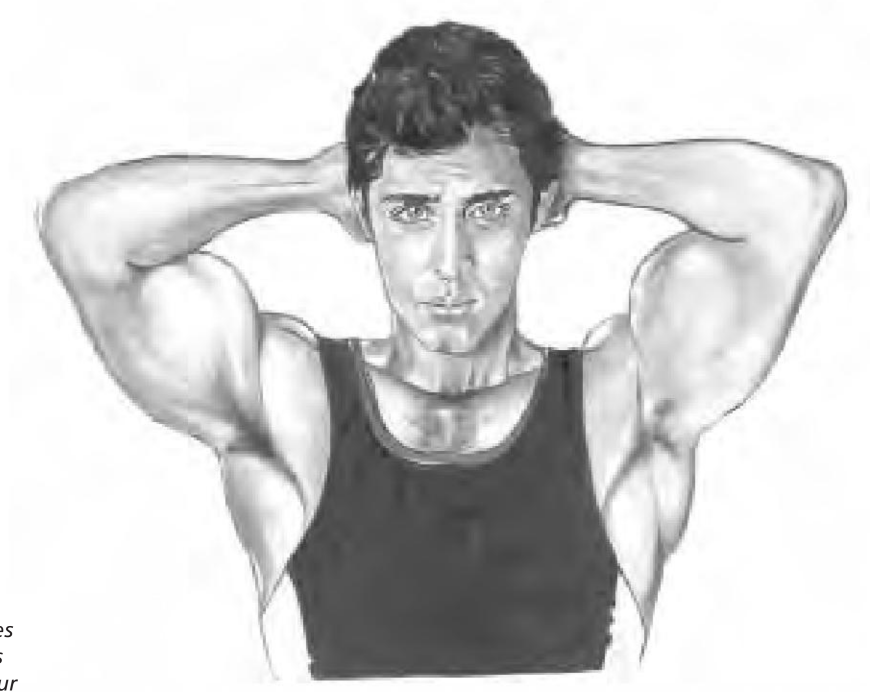


Fig. 5.5 *Muscles have continuous variations in colour*

So now will our good old GIF compression technique work? GIF is based on the premise that several pixels in a row have the same colour and can therefore be compressed. Not so with photographs. So while you can certainly use GIFs, the compression you will be able to achieve (or the reduction in file size) will be extremely small indeed.

Does that mean photographs are not downloaded on the Net? Of course not, but then the compression technique is different. It is a technique where variations of colour can be compressed well, provided the variations are not too drastic. It's called the **JPEG** format, and it stands for the **Joint Photographic Experts Group**, named quite naturally after the group of people who developed the format.

Am I going to explain this technique? Well it really depends on your interest. If I were to provide a simple Mathematical explanation using Discrete Cosine and Fourier Transforms, I am sure you would understand perfectly. A piece of cake isn't it? But then, you would almost certainly be a PhD in Mathematics.

So let us not get into JPEGs in this book. Just remember that GIFs are meant for drawings, sketches, or cartoons. And JPEGs are meant for photographs. Use the right one in the right place.

And next time you meet Hrithik Roshan, you might want to tell him, "Hrithik, you have terrific JPEGs!!!"

VIDEO COMPRESSION

Okay, now what about video? You have lots and lots of sites that have a little video clip on the home page—maybe the Chairman's speech, or even a little clip that shows you how to use a product, for which you are asked to pay an exhorbitant price on the next page.

I hope you understand the basic principles of video. You have several frames, one after the other, each frame being one photograph or one graphic. Each frame is slightly different from the previous one, and so on. And by seeing all these frames in quick succession, you get the feeling that the photograph is actually moving, when it is only different frames rapidly seen one after the other (isn't that a great way to fool the public?) For instance, in Fig. 5.6, you get the feeling that the ball in the child's hand is bouncing.

Strictly speaking, if you have such frames changing at least 20 times a second, it's enough to fool the human eye. And this is called the persistence of vision.

(At this stage you may want to invest ten rupees and buy one of those fat little books where, when you flip the pages in rapid succession, you get a small fish being swallowed up by a bigger fish, which in turn gets swallowed up by a still bigger fish, and so on—rather like politicians. As far as I am aware, Tata McGraw-Hill does not publish such advanced literature, but I'm sure you can find it freely on most pavements of Delhi or Mumbai. And believe me, it is by far the best method I know, of understanding how video works.)

How much time does it take to download video?

Well, for this we need a different example. And we need motion, so let's look at a video of Sachin Tendulkar hitting a six.

The whole process of hitting a six takes about 10 seconds, from the time the bowler starts his run-up, to the time the poor chap in the stands—who is hit on the head—is carried away to hospital.

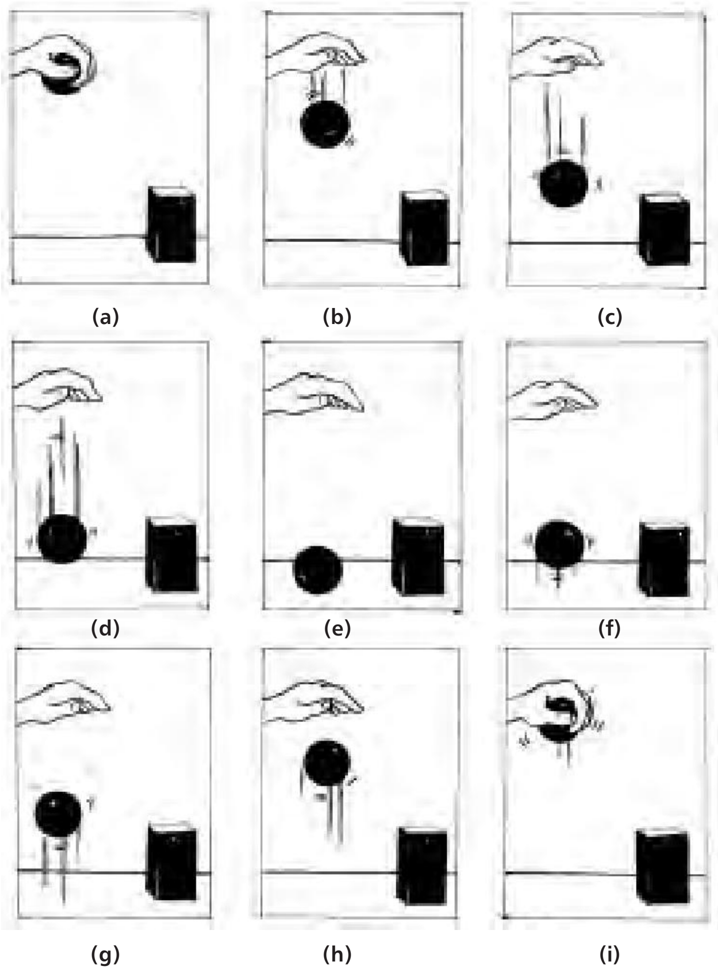


Fig. 5.6 *Video: Several frames, seen in quick succession, give the impression of movement*

Let us once again take a video where the size is 300 × 300 pixels. Each frame therefore contains about 1,00,000 pixels and has a file size of 1,00,000 × 24 bits or 24,00,000 bits. Exactly as before.

But now can you see the difference? To see this video smoothly, you need 20 frames per second. And you have 10 seconds of the video, so that gives us 24,00,000 × 20 × 10 or about 48,00,00,000 bits. And at the same download speed of 2 Kbps, that's about 2,40,000 seconds. Doesn't seem much, does it? Just a measly 3 days !!!

The match would be over by then, and maybe even the next match.

So what's the answer?

Good old compression of course !

Look carefully at Fig. 5.6. Can you see how the logic behind GIF can be extended to video?

Of course you can. In the case of GIF, we transmitted the differences between one pixel and the next. Where the pixel was the same colour, we did not transmit it again. In video on the other hand, we transmit differences between one frame and the next. So in Fig. 5.6, the picture of the hand needs to be transmitted only once. It is just the ball which needs to be sent across every time.

Wonderful thing isn't it?

I hate to disappoint you, but compression in video is somewhat more complex. However, the basic principles are essentially what we have discussed so far.

Two commonly used formats or techniques for video compression are the **AVI** or **Audio Video Interleaved** format, and the **MPEG** or **Motion Picture Experts Group** format.

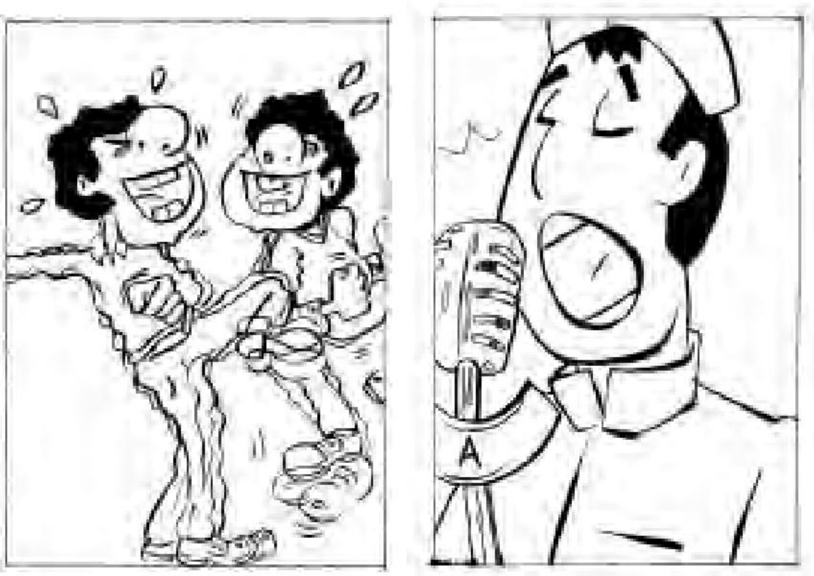
Once again, you would use video editing software to create any of these formats and store it at the web server end. And the browser would decompress this to recover the original video, which would then be displayed on your screen.

One last question: If you have two different videos of the same size and duration—one which shows a very vigorous dance, and the other one with someone making a boring speech (Fig. 5.7), which of these is likely to be compressed better? In other words, which of these is likely to lead to a smaller file size?

Naturally the second one ! Because that has less variations between successive frames.

So remember, the next time you need to put a video recording of your Chairman's speech on your site, and your Chairman is the kind who likes to wave his arms around while talking, you need additional manpower to shoot the video : One cameraman to film the shot, one person for the lighting and effects,.........

.........and one person to stand behind your Chairman and hold his hands still !!!



(a) (b)

Fig. 5.7 *Two different videos: which one leads to better compression?*

CACHING

Finally, we have one more solution to the problem of speed. And this one is not linked to compression in any way. It is a solution that we call caching.

Let us for a moment get back to our matrimonial search problem with the YetAnother-Cola company in Chapter 4. Assume that good old Rajan Sinha was a highly eligible bachelor (at least that's what several unsuspecting parents of would-be brides think). And therefore there is a string of proposals for him from—you've guessed it—Jhunjhunu.

Now the distributor at Jhunjhunu realises that a large chunk of management time of the YetAnother-Cola company is going into getting information about Rajan Sinha. Obviously his boss, the branch manager, doesn't find it funny, so the distributor decides to help. He keeps information on the whereabouts of Rajan with himself. And therefore, after the first such request, all subsequent requests get answered at his level rather than being sent up to his boss.

Do you see something similar in the Internet world?

Of course you do !!

You are connected to some ISP. Obviously there would be several other users connected to the same ISP. Now assume that all of you happen to have a soft corner for

Cindy Crawford, and access the same Cindy Crawford web page, but at different times. The ISP server realises this early on, and therefore stores this web page (see Fig. 5.8). So when a new user asks for this page, the request does not have to go half-way across the globe. It simply goes up to a server at your ISP, from where the page is picked up and sent back to you.

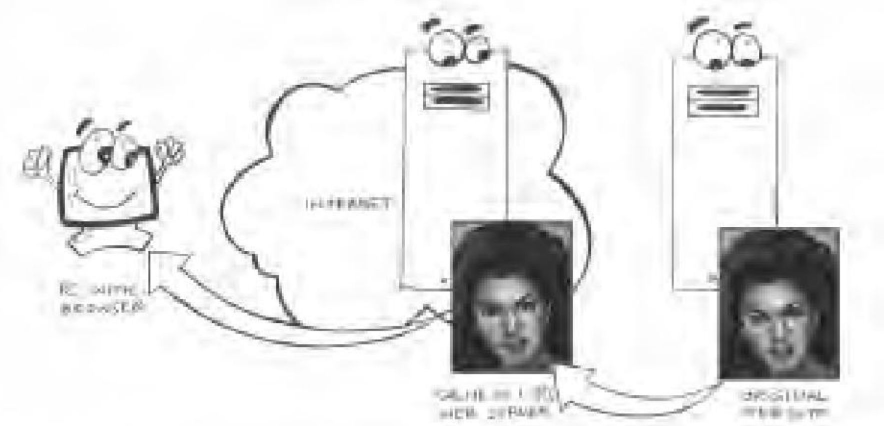


Fig. 5.8
The process of caching

There is a name that is used for this. No, not for Cindy Crawford (I think her name is just fine), but for the process of storing commonly used pages in the ISP's server, thereby increasing speed of access. It's called **caching**. Incidentally, caching is relevant not only for graphics and video but for just about any web page—even one with only text. The only thing is that the time saved is higher with highly graphic pages.

Summary

This was the most colourful chapter of the lot. Where we looked at muscles, more muscles, and still more muscles. And dances, and chairmen making boring speeches.

But if you look beyond these, we actually examined the world of media on the Internet. We saw that transmitting anything other than pure text or data meant large files travelling across the Net—and therefore required tremendous bandwidth. And of course, bandwidth, like money, is always in short supply !

But then we saw how to compress large graphic files into smaller ones using a technique called GIF compression. Which is a method where we do not transmit all the pixels in a graphic, but instead transmit only as much information as is required to reconstruct the graphic at the browser end.

How does this change the e-Commerce architecture we been building up, chapter by chapter?

Actually it doesn't. Except that web pages from the web server to the browser, now come along with additional files, which could be graphic, video, or audio files (Fig. 5.9).

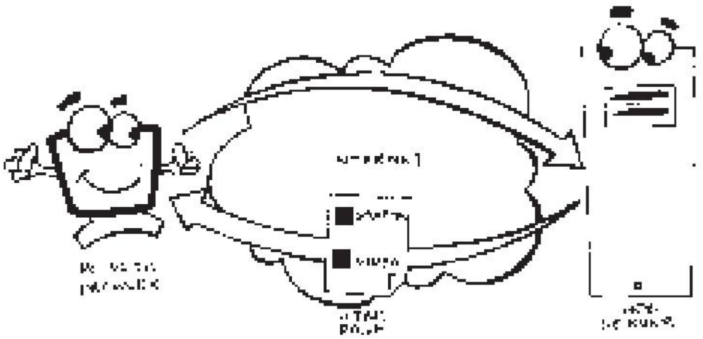


Fig. 5.9
The earlier e-Commerce architecture, with graphic and video files transmitted along with an HTML page

We also took a brief look at JPEG compression in the case of photographs. And MPEG and AVI in the case of video. With the fundamental principle remaining the same—transmit only what is required to reconstruct the photograph or video at the other end.

And finally, we took a look at caching. Specifically, we saw that if someone accesses pages from Cindy Crawford's web site, his ISP stores or "caches" these pages, to make it much faster for other people who ask for the same pages.

So remember the principle of caching, my friend. And be grateful for all your friends who also have a soft corner for the lady. Because every time they reach her on the Net, they make it that much easier for you.

6

Building Dynamic Sites—The Database Way

So far we have learnt all about the web, web servers, and web pages. We have looked at how communication takes place on the Internet, and how English-like domain names are resolved to the more complex IP addresses.

Now it's time to complicate life a bit (after all, we must understand why computer people get all those fat salaries and perks, isn't it?). So in this chapter we examine the fascinating area of dynamic sites and databases.

THE NEED FOR DATABASES

Whichever city or town you live in, I am sure you have a favourite *sabziwala,* who comes around to your house every day with his cart piled up with greenish potatoes, stale onions, sorry looking peas, and beans that may have been fresh a week ago. However that is not the point. The point is that he sells you something every day. And for argument's sake let's call him Haribhai.

Now guess what? Haribhai need not visit you any more. Why? Because he has launched his own web site—sabziwala.com. And whatever you order on the Net is delivered by his young nephew, who is under training to become a proud *sabziwala* in his own right some day (obviously with the consequent smart rise in his net worth in the dowry market).

Haribhai has engaged a leading American multinational (headquartered in Boston, Massachusetts) to create his e-Commerce software. Which is naturally lots and lots of HTML pages, each page listing some vegetable, with package sizes, prices for each, time to deliver, and so on and so forth.

Suddenly one day there is a rumour that the transporters of the area may go on strike. And our good old, honest Haribhai realises with a smirk that he can now charge twice as much for the same old stale vegetables.

But when he goes to the multinational, he is in for a shock.

"Sorry, sir, but we'll have to change the HTML pages. And that means you need our programmers all over again. And that will mean……let me see…….. about $10,000. But since you are a valued customer and a major account for us, we'll cut it down to just $8,000 !"

Haribhai is shattered. How can he possibly make up this money even if he charges three times the normal rates? And then he gets another terrible thought: Tomorrow, when the rumour is squashed by the appropriate government official, he would have to bring the prices down again. So he would again need to go back to the multinational—$8,000 once again. And indeed, every time he changes the price of his vegetables, the gainer would only be the software company !

Apart from feeling sorry for Haribhai, can we think of a solution?

Yes we can !!!

Indeed, there has to be a solution, because the data that you put out on your HTML page needs to be changed all the time. In fact, all software companies use this solution. And that solution is to use a **database**.

DYNAMIC AND STATIC SITES

What is a database? Very simply, it is a collection of data, organised in some manner. And you can "query" this database to get whatever data you want. Databases are an essential component of any web site today, and of any computer based application, whether on the Internet or otherwise.

Let's take a look at Haribhai's vegetable database. Assume he needs to store three pieces of information—name of the vegetable, size of packing, and price. The database would then be in the form of a table (see Table 6.1).

Table 6.1 *The vegetable database in sabziwala.com*

Name of vegetable	*Size of pack (kg)*	*Price of pack (Rs)*
Onion	0.5	20
Onion	1.0	38
Potato	1.0	15
Potato	2.0	28
Tomato	0.5	15
Tomato	1.0	28
Tomato	2.0	54

Notice that this table defines a relationship between *Name of vegetable, Size of pack* and *Price of pack*. Consequently it doesn't take much imagination to see why such a database is called a **relational database**. And in fact, the table itself is often called a relation![1]

Now let us see how we can query such a database. If we wanted to get the price of a half kg pack of onions, we would give a command to the database which would look something like this:

'Get *price of pack* for the row where *name of vegetable* is 'Onion', and *size of pack* is '0.5 kg'

And naturally, the result will be another table—this time a smaller one, which looks something like this (Table 6.2):

Table 6.2 *Table formed as a result of the query*

Price of pack (Rs)
20

[1] When I wrote this piece, my wife Rajni was laughing away at my complete ignorance of vegetable prices today. But it is not ignorance at all. No sir ! Haven't you heard of a cute little thing called inflation? And by the time this book gets into print—who knows—these prices may be a reality!

And that's the result of the query, isn't it?

Notice that the result of the query doesn't need to give the first two entries (*name of vegetable* and *size of pack*). Because you know them anyway—after all you were the one who asked the question. It's like an exam where the student doesn't have to write down the entire question along with the answer—just the answer will do, as long as he writes down the question number.

Can you now see Haribhai's problem getting solved?

Of course you can!

Because now, when Haribhai needs to change prices, he doesn't need to change any of the HTML pages. He doesn't need a programmer. Most of all, he certainly doesn't need the Boston based multinational. All he needs is someone who understands simple English, and who can enter this data into the database every time it changes. In fact, his 12-year-old niece, who goes to an English medium school, could do the job. (The big bonus of course is that Haribhai could explain this to her father as terrific training for her, and therefore not pay her at all.)

The other brilliant thing is that it gives you one more set of technical terms that you can brag about, namely **dynamic sites** and **static sites**. Naturally a static web site is one which consists of HTML pages which are permanent. Any changes in such a site require re-programming of some of the pages. Therefore such a site is useful only when changes are infrequent.

On the other hand, dynamic sites are those where the web page is created dynamically, as in our example, through a database, instead of being permanently coded in HTML. As discussed, these sites are useful when some of the HTML pages need data that keeps changing frequently. The term we use for this is that database-driven sites are more maintainable (in other words changeable, if the need arises) than sites which are simply based on static HTML pages.

Incidentally, where is the database stored? Usually on a separate server called the **data server**. And, the data server is generally connected to the web server through a LAN or Local Area Network link (see Fig. 6.1).

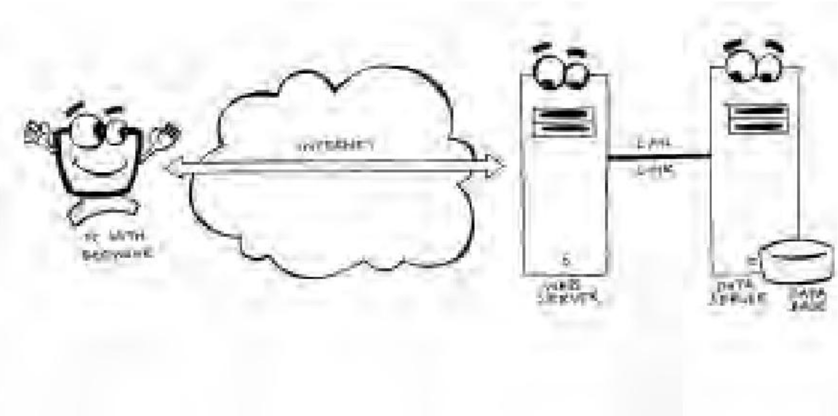


Fig. 6.1 *The data server and web server are connected through a LAN link*

DATABASE NORMALISATION

Surely there is more to databases than this. After all, this couldn't be what database specialists are paid fabulous sums of money for!!!

Yes there is. Let us complicate life just a little bit and see what happens. Supppose our friend Haribhai now introduces a delivery charge as well. But remember, Haribhai is smart, so he speaks to a logistics consultant, based in Geneva. This consultant advises him on how delivery charges should be computed. To cut a long story short, the consultant tells him to keep the delivery charges dependent only on the weight of the consignment, and not on the contents. In other words, the delivery charges for 2 kg of potatoes, tomatoes, or any other vegetable, would be the same. But the delivery charges for a 1 kg pack would be different.

How do we represent these delivery charges?

Obviously, one simple way is to write them down as part of the HTML page.

But do you see a problem here? What if tomorrow, Haribhai's nephew were to acquire a girl-friend with the consequent quadrupling of his expenses? And what if he were to pluck up the courage to come to his uncle and demand a raise? Naturally Haribhai, being the hard-nosed businessman that he is, would need to raise delivery charges. And that means...........

You've guessed it. It means still more dollars for our Boston-based multinational!

So what's the way out?

Simple—add an extra column to the database. So now our vegetable database would look something like Table 6.3.

Table 6.3 *The new database with delivery charges included*

Name of vegetable	*Size of pack (kg)*	*Price of pack (Rs)*	*Delivery charges (Rs)*
Onion	0.5	20	4
Onion	1.0	38	3
Potato	1.0	15	3
Potato	2.0	28	2
Tomato	0.5	15	4
Tomato	1.0	28	3
Tomato	2.0	54	2

So now, when Haribhai's customer, sitting at the inevitable browser, asks the question, "What's the price of a half kg pack of onions, along with the delivery charges", the software translates this into the query:

'Get *price of pack* and also *delivery charges* for the row where *name of vegetable* is 'Onion', and *size of pack* is '0.5 kg'.

And the resulting table would look like this (Table. 6.4):

Table 6.4 *Result of the second query, which includes delivery charges*

Price of pack (Rs)	*Delivery charges (Rs)*
20	4

Wonderful! Of course we are making the somewhat simplistic assumption that Haribhai delivers only one packet at a time, so delivery charges cannot be amortised over a large consignment of several packets (I could of course discuss such complex scenarios as well. In fact, I do in a semester long e-Commerce course that I teach at the Management Development Institute—and all of you are most welcome to attend that course).

Now let's complicate life still further. We have mentioned that the delivery charge is dependent on only the weight of the pack and not on the vegetable, or the price.

Can you see a problem with this?

No?

Okay, I'll give you a hint. What if Haribhai has just read *Marketing Management* by Philip Kotler, and therefore decides to remove all 0.5 kg packs for the moment (he probably feels that a person who buys a half-kg pack would be willing to buy a 1 kg pack as well). So he removes all the entries in the database in Table 6.3 that correspond to half-kg packs.

The database now looks something like this (Table 6.5):

Table 6.5 *The vegetable database with half kilo packs removed*

Name of vegetable	*Size of pack (kg)*	*Price of pack (Rs)*	*Delivery charges (Rs)*
Onion	1.0	38	3
Potato	1.0	15	3
Potato	2.0	28	2
Tomato	1.0	28	3
Tomato	2.0	54	2

Now do you see the problem?

No?

Okay, you get one last hint ! Haribhai decides to go 'upmarket' and introduce exotic vegetables, starting with broccoli. Now broccoli is expensive, and it is likely that most people would not want more than a half-kg pack.

So what would the delivery charges be? Clearly the database doesn't specify anything, because there are no rows for half-kg packs. And Haribhai has a poor memory for figures, so he doesn't remember either. In other words, there is no record kept anywhere, of the fact that a half-kg pack has an associated delivery charge of Rs 4.

And that, my friend, is the problem. Because now Haribhai will have to go back to our Geneva-based logistics consultant, shell out the appropriate Swiss Francs once again, and get them to work out delivery charges for half-kg packs all over again.

Some of you may find these arguments simplistic. But just replace Haribhai with a large consumer products retailer such as Wal-Mart, who deals with perhaps 100,000 products or more, each in a unique pack size and shape, and suddenly the problem becomes real.

What's the key issue?

Very simple ! We have two different kinds of information that we are dealing with here. First of all, there is the information about each kind of vegetable—onions, potatoes, etc., along with its pack sizes and associated prices. And secondly, pack size related information such as the delivery charges for a pack size, which is independent of the vegetable that is delivered.

And the fundamental rule of relational databases is that we must separate out information that is logically independent.

So what do we do?

Very simple. Use one relation for information about vegetables, and a separate one for information about delivery charges (Table 6.6).

Table 6.6 *The database split into two distinct relations*

Relation A: *Vegetable related information*

Name of vegetable	*Size of pack (kg)*	*Price of pack (Rs)*
Onion	0.5	20
Onion	1.0	38
Potato	1.0	15
Potato	2.0	28
Tomato	0.5	15
Tomato	1.0	28
Tomato	2.0	54

Relation B: *Pack size related information*

Size of pack (kg)	*Delivery charges (Rs)*
0.5	4
1.0	3
2.0	2

The data about delivery charges will now remain in the database, even if Haribhai removes all half-kg packs of vegetables!!!

Now how do we query such a database? Well, it is a little more complex than before, because it's a multi-step process. The query now looks something like this :

'Get the entire row from relation A, where *name of vegetable* is 'Onion', and *size of pack* is '0.5 kg',.

The result would appear as in Table 6.7.

Table 6.7 *The entire row of information from relation A, where name of vegetable is 'Onion' and size of pack is '0.5 kg'*

Name of vegetable	*Size of pack (kg)*	*Price of pack (Rs)*
Onion	0.5	20

The next step in the query is to say:

'For this *size of pack*, get *delivery charges* in rupees from relation B, and attach it to the same row'.

And that gives us the relation in Table. 6.8.

Table 6.8 *The relation in Table 6.7, with delivery charges attached*

Name of vegetable	*Size of pack (kg)*	*Price of pack (Rs)*	*Delivery charges (Rs)*
Onion	0.5	20	4

And finally, we would perform the third step of the query, where we say:

From the relation in Table 6.8, get me just *price of pack* and *delivery charges*. And lo and behold, we have the final relation (Table. 6.9).

Table 6.9 *Final result of the query*

Price of pack (Rs)	*Delivery charges (Rs)*
20	4

Now two things need to be clarified:

(a) First of all, the way we have been framing our queries (such as, 'Get *price of pack* and also *delivery charges* for the row where *name of vegetable* is 'Onion', and *size of pack* is '0.5 kg'), is rather informal. The purpose was to explain a concept to you. In actual practice, programmers would use a somewhat more formal **query language** and the most commonly used one is called **Structured Query Language** or **SQL**.

(b) Secondly, what we have done is to split a database into separate relations, with distinct information in different relations. This process is known as **database normalisation**. And the resultant database is said to be in the **3rd Normal Form**, or sometimes just **3NF**.

But why 3rd Normal Form? Why not 1st or 2nd? Actually those are simply intermediate forms before we arrive at the 3rd Normal Form, and are never used independently. Therefore, unless you have a Computer Science exam to take, don't worry. Just stay with the 3rd one!

COMMON GATEWAY INTERFACE OR CGI

You have just seen that dynamic sites need access to databases. And that means queries need to be written in some query language such as SQL.

In addition, when the database returns a table as a result of a query, it is not in the HTML format. After all, databases were here much, much before the world wide web. Even today, databases are used for several reasons, not just to feed web pages.

And finally, even if databases were to give results in HTML format, these results would still need to be integrated into the rest of the web page. In our current example, for instance, the web page would include things like a title, a nice picture (maybe of Haribhai and his wife as the chief shareholders of the venture) contact addresses of Haribhai, several clickable icons which lead to 'check out', 'feedback', 'chat forum', etc. etc. etc. In fact there would be lots and lots of things on the page, in addition to vegetable prices. And of these, vegetable prices are the only thing that needs to come from the database.

In other words, there are several things that need to be done, when a web page is created based on access to a database:

(a) Get the appropriate data from the database by using a query
(b) Convert this into HTML format
(c) Integrate this with the static HTML portion which already exists with the web server

So who does all these tasks?

It actually needs a program to do all these (what else?), and it could be written in many, many languages—some of which are C, C++, and a unique language called PERL. The only thing is that this program must communicate with the web server using a standard format. And that standard format is called **common gateway interface** or **CGI**. For obvious reasons, the program is called a **CGI script**. Figure 6.2 shows all these pieces interacting with each other.

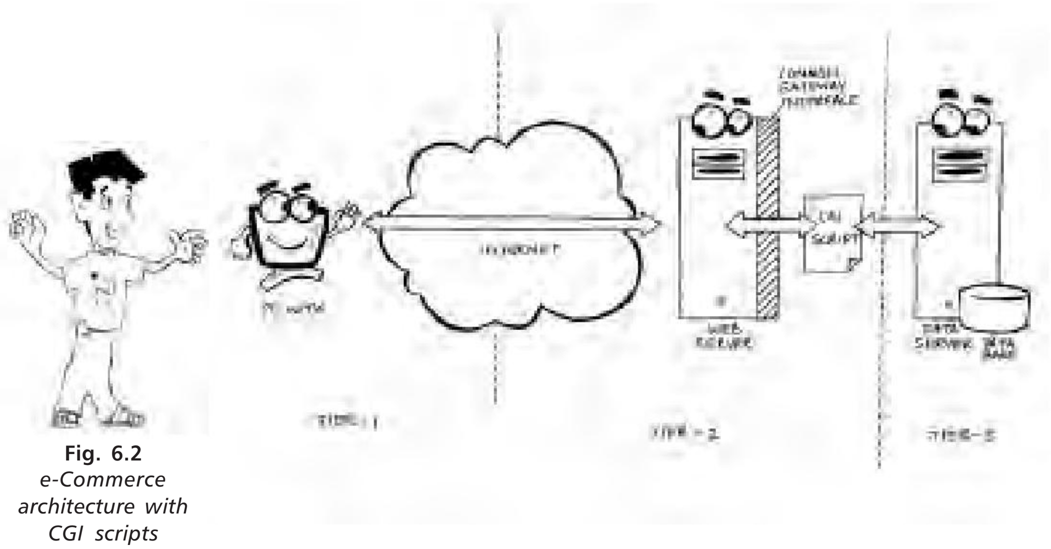


Fig. 6.2 *e-Commerce architecture with CGI scripts*

Incidentally, CGI scripts are an example of **server side computing**, because the CGI programs run on the server, as against the client PC. And before I forget, CGI scripts can be used to do any processing at the web server end, not just database access. As an example, for Haribhai's Japanese clients, CGI scripts could be used to convert all pricing information from the database into Yen, and only then send it to the browser as an HTML page.

Is CGI the only way to access databases on the Internet?

Well, no. In fact CGI scripts are seldom used today and have now been replaced by newer technologies such as Java, Active Server Pages (ASP), and Java Server Pages (JSP), which are described in subsequent chapters. But it was the pioneer in the area, and many sites still use the concept.

THE 3-TIER ARCHITECTURE

Take a look at Fig. 6.2 again. Do you notice that our e-Commerce Architecture has been clearly defined in **layers**, or **tiers**?

Tier 1 is the client tier, which is your PC and the browser running on it. This tier normally contains nothing but the browser. During access to the Net, HTML pages and their associated programs would get loaded onto this layer and executed.

Tier 2 is the presentation and business logic layer or tier. All processing of data is done here. All business rules (such as creation of shopping carts) are implemented in this layer. In addition, all pages are formatted here before final delivery to your browser in HTML format.

And finally, Tier 3 is the database layer, which contains the databases and all the associated data.

The good thing about this layering is that we have split up the problem of designing a web site into three distinct parts, instead of one huge complex problem. The other interesting thing here is that we have made the site more maintainable. For instance, you can change the database tier without affecting the middle tier, or vice versa. Obviously this would not have been possible had the layers been mixed up !

DYNAMIC SITES AND SPEED

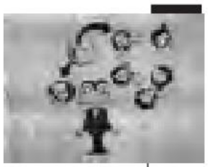
One last question. If databases are such a great thing, why don't we use them everywhere? We've just seen that database-driven sites are far more maintainable than those with pure static HTML pages. So what's the problem?

The problem is very simple—databases tend to slow down the site (isn't that obvious?). If pages were to be stored in final HTML format on the web server, getting any page into your browser is just a two step process—send in the query, and pick up the page. And is therefore quick! Instead, if it needs access to a database, we are talking of several steps—namely making a call to another server (the data server), searching and processing various tables in the data base, getting out the appropriate data, sending it back to the web server, converting it into HTML format, and finally inserting it into the HTML page for sending to the browser.

All this would take a lot of time wouldn't it?

So the moral of the story is that database-driven sites are maintainable but slow. In other words, use databases where you expect changes on a regular basis. But don't use them all over the place, just because you think you've mastered normalisation. Good old static HTML is still the fastest technology going!

SUMMARY



And that was our friend Haribhai. Interesting character, isn't he?

Haribhai is not aware that he has been instrumental in your learning all about dynamic and static web sites. And relational databases. Which are essentially in the form of relations or tables.

Haribhai has helped you figure out that some sites, where data keeps changing, are best created with databases. Instead of static HTML pages. Unwittingly, he has also told you all about the need for normalisation—specifically the 3rd Normal Form. Because good database design requires that we separate out information that is logically independent. In other words, keep independent information in different relations.

He also showed you how the web server accesses the database and converts the data into HTML format, ready to send to the browser. For this, we need an interface called CGI, or Common Gateway Interface. Which would permit us to use just about any language, such as Basic, or C++, or PERL to write the program to access the database. And before I forget, these programs are called CGI scripts.

This has, quite naturally, added one layer to the e-Commerce architecture that we've been building up. And in the process, has created the 3-tier architecture shown in Fig. 6.2.

And at this stage, I must leave you. Because I just heard my doorbell ring.

I suspect it's Haribhai's nephew, who has come to deliver the vegetables I ordered.

So I'll leave you to mull over databases. And dynamic sites. And CGI scripts.

Think about them. And don't get too confused. There's still a lot more to learn !

7

The Magic of Java

JAVA !

The magic word of the late nineties. The word that seemed to change the shape of the entire computer industry. If there was one word that almost certainly led to a Green Card in the USA, it was Java. Rumour had it that programmers who knew Java commanded a significantly higher dowry in the marriage market than those poor souls who survived on miserable languages such as C, C++, and Visual Basic. Of course, I could never confirm this rumour, largely because I was already married by the time Java came on the scene.

But what was or is Java? And how is it different from the hundreds of other languages that are simply oozing out of the software industry?

Well, if you really want to know, there is a very obvious solution isn't there?

Simply read this chapter !

SERVER SIDE COMPUTING—THE PROBLEMS

Why do we need Java?

To understand the answer, we need to look at an example. Let us assume that you are a stock market enthusiast. It doesn't matter whether or not you have pots of money, but you play the share market. So on the days when the OPEC decides to hike petroleum prices and the Bombay Stock Exchange Sensex crashes by 100 points, you lose your appetite. Conversely, when the Prime Minister announces yet another dose of liberalisation, and the Sensex goes up by 200 points, you suddenly start beaming at all your neighbours—even the ones you fought with last week.

Naturally, since you are such a share buff, you need solid amounts of information before you decide to put in your hard-earned Rs 1000 into such and such shares. So you go to a site such as sharecrash.com, which helps you buy and sell shares over the Internet, or simply get the latest information about the stock market. And you click on the relevant hyperlink on their home page to get the intra-day share price movement of Hindustan Lever (for those uninitiated souls who do not understand this jargon and would prefer to put their Rs 1000 elsewhere, this is simply a minute-by-minute log of how the share price of Hindustan Lever moved during the day).

You click—and lo and behold you get the following table (Table 7.1):

Table 7.1 *Intra-day share price movement of Hindustan Lever*

Time (hrs. min)	*Share price (Rs.)*
10.00	220.40
10.01	220.00
10.02	219.50
10.03	219.05
10.04	218.90
10.05	218.50
10.06	212.00
10.07	208.00
10.08	197.75
10.09	197.50
10.10	197.50
10.11	197.50
10.12	197.50
10.13	197.80
10.14	200.25
10.15	203.75
10.16	206.50
10.17	208.90
10.18	212.15
10.19	217.20
10.20	225.50
10.21	232.80
10.22	238.85
10.23	245.60

Wonderful isn't it?

Or is it?

Do you really want to go through such a massive table with so many detailed entries?[1]

Of course not !! That's silly. You would much rather see a graph—something like the one in Fig. 7.1:

Great! But now there is a small question to be answered. Who draws the graph?

As you might have guessed, the data that we have shown in Table 7.1 is available in some database which has been populated right through the trading day. But someone has to take this data and draw the graph.

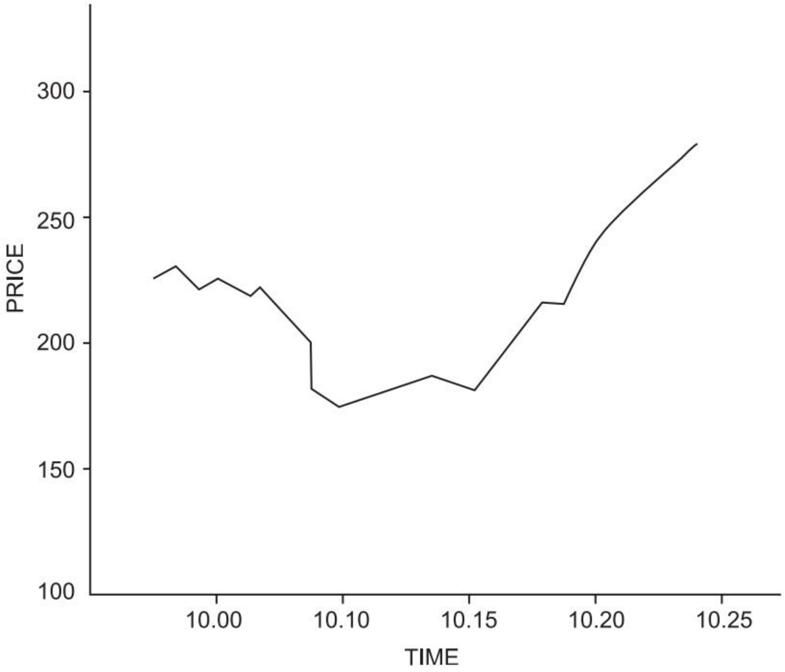


Fig. 7.1 *Graph of intra-day share price movement of Hindustan Lever*

[1] For each day, there are probably many, many more rows in this table. But after detailed and very animated discussions with the publisher of this book, I decided to show you simply one sample page, thereby cutting down the total cost of the book.

Naturally that someone is a program. And if you recall Chapter 6, it is our good old CGI script.

So what's the problem?

Very simple. In fact there are two problems.

First of all, CGI scripting suffers from a very cute disorder. It can only be used by one user at a time.

"Aha", you might say, "But I am not the only user accessing the site. My friend from Patiala might be trying out his luck with shares at exactly the same time as me. Or my other friend from Mumbai. In fact there might be hundreds and hundreds of people accessing sharecrash.com at the same time. Maybe not for Hindustan Lever shares but for Reliance, or Glaxo, or NIIT, or any other shares. And several of them might be asking for the Intra-day share movement graph at exactly the same time. So how does the CGI script manage?"

Wonderful, my friend. You've hit the nail on the head ! CGI scripts cannot manage more than a user at a time. So they do the next best thing. They create copies of themselves, like a photocopier (Fig. 7.2). And each user runs one of these copies of the program[2].

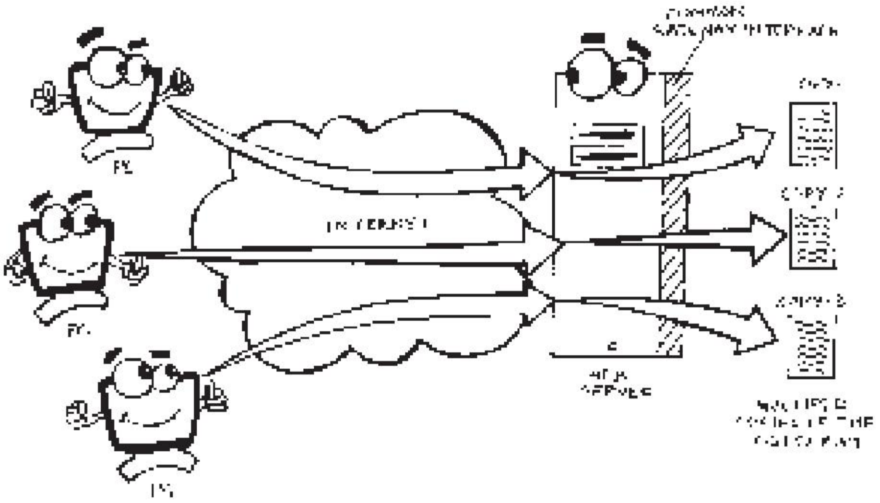


Fig. 7.2
Multiple copies of a program— one for each user

[2] Strictly speaking what is created is several copies of the *process* but unless you are planning to do an M.Tech in Computer Science, the explanation given above is fine. And it is certainly enough to impress your boss, your girl friend, or even to hold forth at your next party!

Can you now see the problem? If you have a hundred users logged on to sharecrash.com at a given time, you have 100 copies of the Intra-day CGI script running. Imagine the amount of RAM and disk space the web server requires! Also imagine the load on the poor web server, which has to execute 100 copies of the same program simultaneously. You would need a huge and very expensive web server to manage all this. And even after you've got one of the most muscular ones available in the market, you would still get poor response times. After all, you are using one poor, overworked server to run more than 100 programs simultaneously.

What's the other problem?

Simply this: Once the graph of daily share movements has been created, it has now become a graphic or picture. And you know what that means—a BIG file—and therefore lots of time to download it (see Fig. 7.3).

Which means that you have poor response time because of the load of hundreds of programs running on the web server. And you also have poor response time because you now need to transmit a graphic file from the web server to your PC. Therefore, dear reader, sharecrash.com is in grave danger of putting several users to sleep.

So now we have the perfect scenario—two problems, leading to heavy investments in the web server, and also leading to poor response time.

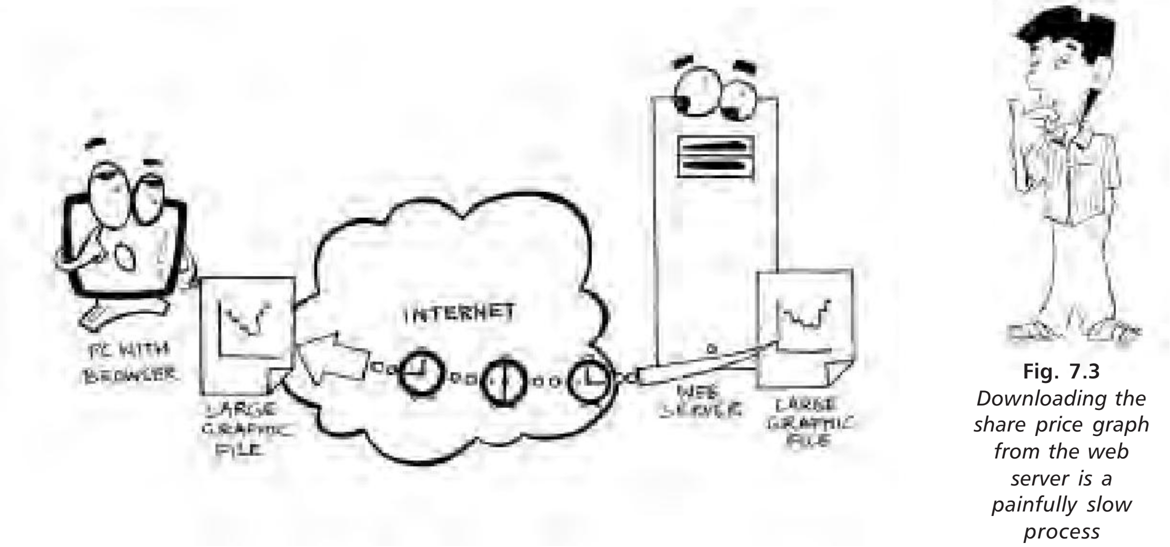


Fig. 7.3
Downloading the share price graph from the web server is a painfully slow process

Is there a solution?

Sure there is ! Read on, dear reader............read on to get the solution!

CLIENT SIDE COMPUTING

If you think carefully (that's right, I do expect you to think from time to time—this is not a novel), you would notice that both these problems have the same solution:

Don't create the graph in the web server. Instead, send the data over to your PC and create it there.

So we don't need to create multiple copies of the CGI script. Thereby, we lessen the load and the memory requirements of the web server. Moreover, we don't transmit a graphic file from the web server to your PC, but only transmit data. Hopefully, therefore, we have a smaller file size and therefore much lower bandwidth requirement.

In other words, what we are looking for is something called **client side computing**—running programs on the client PC instead of the server.

"But where is the program", you ask? "The program to draw the graph is not stored in my PC or in my browser?"

Absolutely correct. And the solution is, very simply, to send the program to your PC from the web server. In other words, when you click on an icon asking for the Intra-day share movement graph, the following things happen (Fig. 7.4):

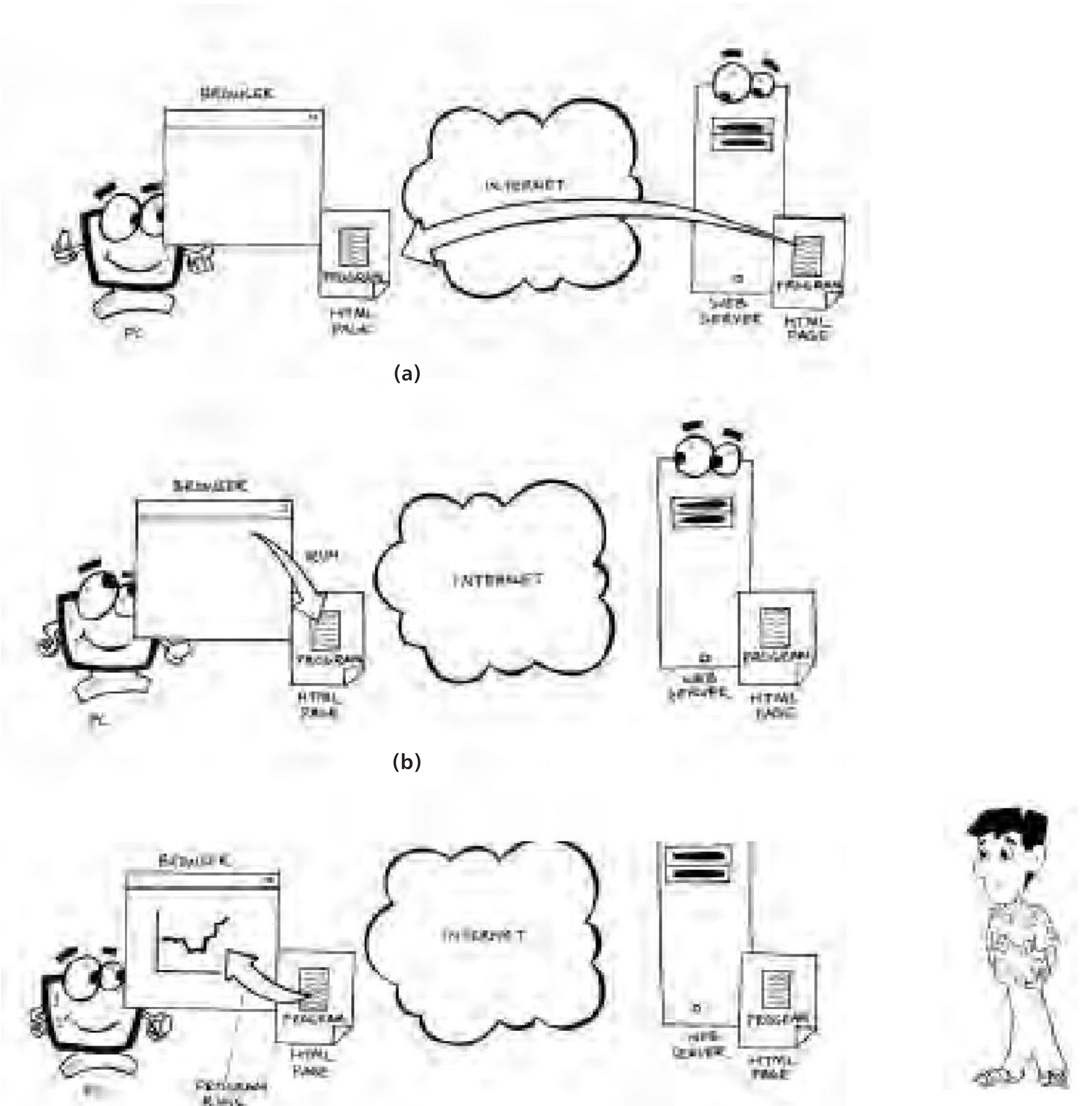
(a) The appropriate HTML page is sent by the web server to the browser. Along with this HTML page, the data, and a program that will use the data to create a graph, is also sent to the browser.

(b) The browser recognises this program and asks it to run.

(c) Thereby the graph is created and displayed as a graphic within the HTML page on your screen.

After you click on something else and go to a fresh HTML page, or close the browser, the program is deleted from your PC. After all, you wouldn't want all kinds of programs to clutter up your PC's memory, would you? And if you wanted to use the program again, you could click on the same icon all over again.

Of course, based on this, you may or may not buy Hindustan Lever shares, but that is irrelevant to the subject of this book.



(a)

(b)

(c)

Fig. 7.4 *Client side computing*

Can you see light dawning now?

Can you see the benefit of getting the program over to your PC or your browser, instead of running it on the web server?

Of course you can. And since you find it so simple, let me give you one more situation.

I assume you have heard of the term "validation". No it is not the annual day of a school. It is the process of checking whether or not the data you have entered is correct. For instance, on a site such as amazon.com, when you are asked to enter the number of copies of a book that you wish to buy, do you naturally enter "GRV"? If so, either you wish to keep the number of copies a closely guarded secret (from amazon, naturally) or you've left your reading glasses somewhere and have typed in letters of the alphabet, where you had to type in digits.

Or do you type in "106" when you are asked to type in your age, which is unlikely to be factually correct, but instead may simply be the result of a hangover from last night's delightful party?

So, my friend, there are many, many situations where you could enter wrong data into the form that is open on your browser. And naturally, someone needs to check this out!

The obvious way is again by using a CGI script. However, this solution has the same problem as before—namely that of speed and load on the server.

In addition, there is one more problem. Suppose you have keyed in your age as "PT". The entire form is now sent to the web server, where you have a CGI script, which does this checking. And if you are lucky, it gets back to you in about 30 seconds, saying, *"Error. Invalid age, Enter only digits."* Remember however, that people on the Internet are impatient people (rather like most car drivers). They don't want to wait 30 seconds or longer to find out a silly thing like this. Essentially, *they do not want the entire form to be transmitted half-way across the world to the web server, only to get back 30 seconds later with an error message*. They would naturally want this message from their own PC, if that were possible.

There—you've got the answer. Just as the graph was plotted by a program which was downloaded onto the client machine, so also programs that validate your data entry can be downloaded and run on the client machine—by using the browser, naturally!

So now you have at least two kinds of problems which have the same solution, namely downloading a program onto the browser, and running it there.

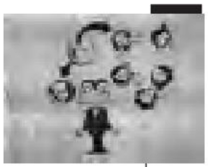
And that, my friend, is where a great company called SUN Microsystems came up with an equally great solution—JAVA !!!

Java is a language in which you write programs and store them on the web server, along with the corresponding web page. The wonderful thing about them is that they do not run on the server. They run on the client—in other words on your PC. So when your browser asks for the web page, the Java program is transferred to the browser along with the page, and then run.

What else could we do with Java?

Actually lots. Any time we require heavy ongoing activity at the browser end, HTML would not be able to handle it, and we would typically look at Java. For instance, tickers running across your browser, giving stock prices, may use Java. Similarly, a lot of chat sites are based on Java running on your browser.

SUMMARY



So that's Java!

We've seen how Java can dramatically cut down the load on web servers. And thereby increase speeds.

We have also seen how it permits us to avoid sending large files across the Net, and instead create these files on your PC.

And finally, we've seen how Java can be used to instantaneously validate data that is entered into the browser. Without spending time waiting till the page is sent to the web server, half-way across the globe !

Java is a truly revolutionary concept, isn't it?

And this is where I am sure you have several questions: How is Java stored? How does it run? Does it have any disadvantages...........?

We could easily have answered these questions right away. But I notice that you are already looking tired. It's been a long chapter, with lots of new concepts. So why don't you take a break—we'll meet again in the next chapter !

8

How Java Works

Now that you are back from the previous chapter, I assume you remember the key issues we discussed there. Namely, the fact that Java is a language in which programs are written and stored in the web server, along with the corresponding web page. And when the browser asks for the web page, the Java program is transferred to the browser along with the page, and then run on the client PC.

You would also recall that we raised some questions—How does Java work? How is it stored, etc. etc. etc....................?

It's now time to answer each of these questions.

So read on, my friend. And begin to grasp the magic of Java!

THE PORTABILITY PROBLEM

What's the key property of a language like Java?

Well, simply this—You write programs in Java which are stored on the server but transferred to and therefore run on the client PC.

And this is where life really begins to be fun. If you think carefully my friend, can you see a magnificent, juicy problem rearing its head?

What's the problem? Well, how does the web server know which PC you have? You may have a Macintosh from Apple. You may have a PC that runs good old Windows. Or you may be a nerd and have a PC that runs UNIX. Or best of all, you may be a needy college student who must use free software and therefore are a disciple of Linux.

In fact, you may have any kind of computer running any kind of operating system. And the wonderful thing about computer systems is that programs designed to run on one kind of computer do not run on the others.

So what does the poor web server do? Does it need to keep versions of every program—one for each kind of client PC it might encounter? And what happens tomorrow if there is a new kind of computer in the market with a new operating system? *Would we need to create new versions of all the programs residing on all web servers?*

Dear reader, that would be a terrific opportunity for our software companies (look at the amount of business they would get). But alas! That is not to be. When SUN Microsystems created the language Java, they worked out a brilliant solution which effectively killed this potentially great business opportunity. Because the wonderful thing about Java is that it can run, at least theoretically, on any computer and any operating system.

Great!

Next question: How does Java manage this? Especially when it was not possible with any other language earlier?

To answer this question, let us first understand the problem in a little more detail.

I am sure you are aware that computers work in binary language—strings of 0s and 1s. So any program for any computer would ultimately need to be a set of instructions in the form of 0s and 1s. For instance,

```
0101000011101011
1010100110000001
1010100101111111
1111111000000000
0001110000111100
```

may be a perfectly acceptable binary program (also called a machine language program).

Also, you would know that human beings have a decided aversion to writing programs in 0s and 1s. For obvious reasons they would much rather write programs in languages called **high level languages,** such as C, C++, PASCAL, or BASIC, which are all closer to English.[1] So for instance, the following may be an acceptable high level language program:

```
for x = 1 to 20 step 1 do
    begin
    a := a+1;
    b(x) := b(x)*2
    end
```

Notice that this program is far simpler to write and understand than a binary or machine language program.

Also notice that since the computer does not understand such a high level language, we need some sort of translator program or **compiler** (Fig. 8.1). The compiler would translate this high level language program into machine language, which the computer can then understand and run.[2]

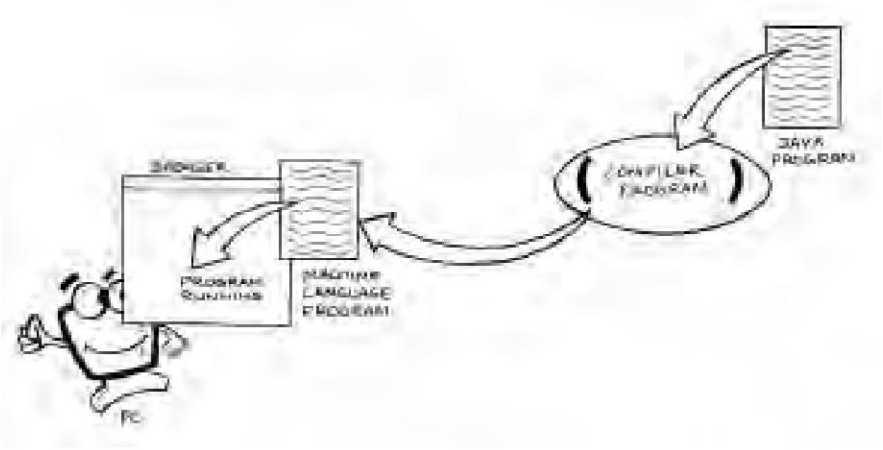


Fig. 8.1
Compiling a high level language program

[1] When I make sweeping statements like this one, I naturally refer to the large majority of programmers on planet Earth. I am sure there exist some wonderful people who would drool at the thought of writing machine language programs—although I have not had the good fortune of meeting anyone from this tribe.

[2] This is only a very cursory explanation of the concept of high level languages and compilers. The assumption here is that the reader is already reasonably familiar with the area. If not, there is a good case for another book in this series.

Therefore, when a program is written in Java, or for that matter in any other high level language, it would need to be compiled by a Java compiler into machine language. But here's the catch—the machine language of a SUN computer is different from the machine language of a COMPAQ PC, which in turn is different from the machine language of a Macintosh....................

And that is why, simply compiling and storing Java programs on the web server is not good enough. Because then you would need to store one version for each and every distinct kind of computer that accesses this web server.

Same question again: How do we solve this problem?

Let us take an analogy. Assume we had a multinational company based in the UK, and they had transferred one of their British managers as customer training manager, Indian operations. This company markets fertilizers, and the job of this training manager is to give instructions to farmers on good practices in fertilizer usage. These farmers then follow, or execute, these instructions.

The company has decided to focus on the Hindi belt (Haryana, Rajasthan, Madhya Pradesh, UP, and Bihar). And naturally, farmers in these states do not speak or understand too much English, so the instructions need to be translated into Hindi.

What do we do?

Simple—all we need is an interpreter who listens to the training manager's clipped British accent and translates the instructions into Hindi.

But this is where the fun really starts.

Have you had the good fortune of travelling across these states?

I have. And I can tell you, my friend, that these different regions do not even speak the same version of Hindi. For instance, the dialect of Hindi spoken in North Bihar is very different from our good old Haryanvi Hindi, which in turn is very different from the dialect in Jaisalmer in Rajasthan, and so on.

Can you now see how the problem has got complicated?

We now have one set of instructions in English, to be translated into multiple dialects of Hindi depending on the region. And therefore you need multiple interpreters, each of whom understands the Queen's English (including the clipped British accent) and also understands one of these dialects.

Dear reader, you could search and search...................., and still not find such a person—and if you do, you would probably pay the moon for him.

Is there a solution?

Yes there is. The solution is to use a two step translation process. In step one, we get an interpreter who converts British English into standard Hindi (for the purpose of our discussions here, let us assume that the language used in the Hindi news bulletin on Doordarshan is standard Hindi—although I often get the feeling that it is closer to Sanskrit). And in each region of the country, we get an interpreter who translates standard Hindi into the local dialect of Hindi.

In a sense, therefore, standard Hindi acts as some kind of intermediate language. And since it is much closer to each Hindi dialect than it is to British English, the corresponding *"Hindi to Local Hindi"* interpreter has a reasonably simple job.

Problem solved?

Wonderful!

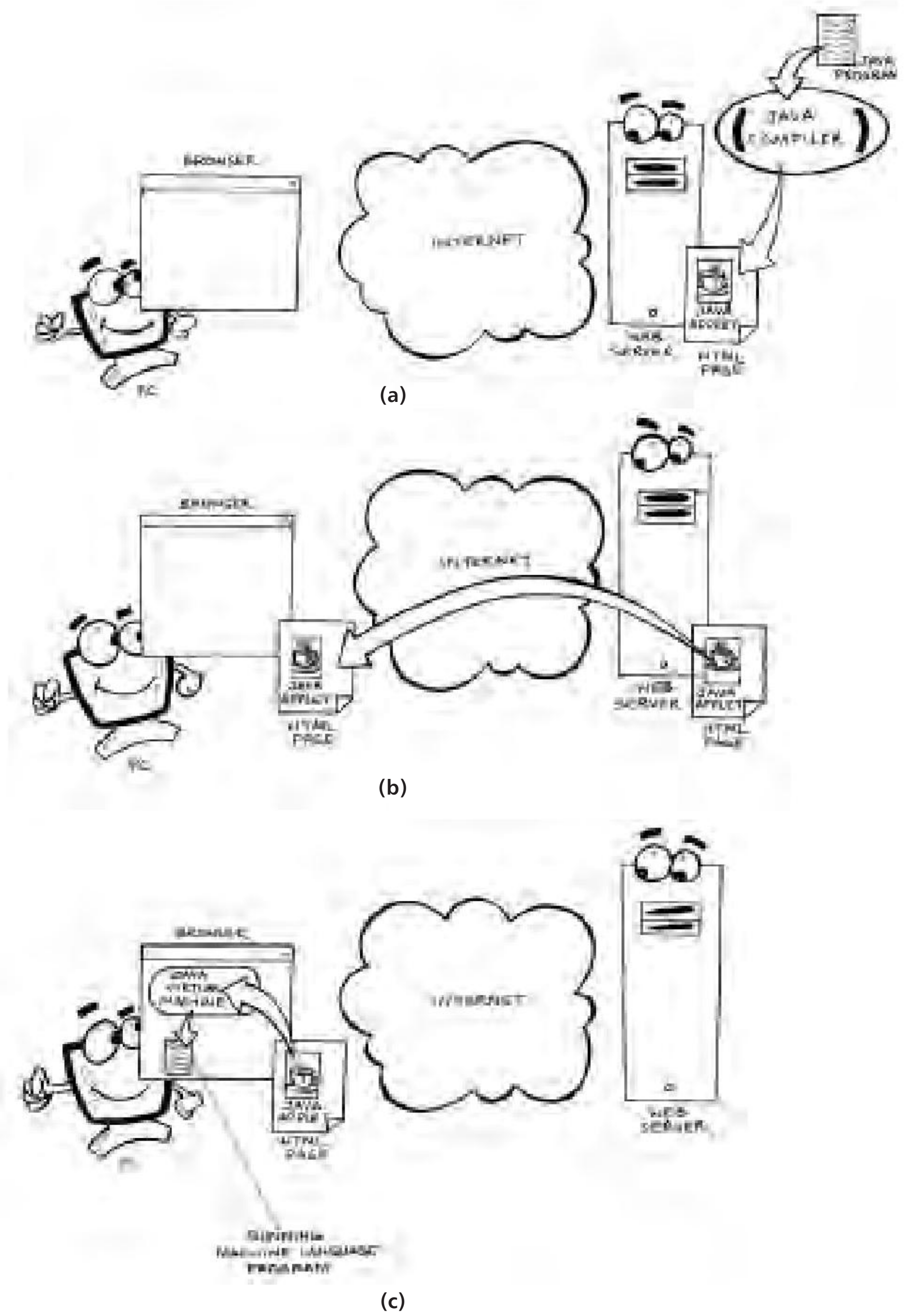
Now if you are awake at this point, can you see the same process in Java? (if you can't, you're probably asleep, so insert a bookmark into this page—or fold the page—and try again later).

For those who are awake, just take the preceding description and do the following:

(a) Replace "*customer training manager*" with "*server*"
(b) Replace "*farmer*" with "*client PC*"
(c) Replace "*English*" with "*Java*"
(d) Replace "*local Hindi dialect*" with "*machine language of the Client PC* "
(e) Create a kind of intermediate language between Java and the machine language, and replace "*standard Hindi*" with this language

Eureka! Now we have truly solved the problem!!

The Java program is translated into a standard intermediate language and stored on the web server. We call this language **Java byte code,** and the program after translation is called a **Java applet** or simply an **applet**. When a browser running on a client PC asks for this applet, it is sent across to the PC. The browser now contains another translator program which converts this applet into *the machine language of that specific computer,* and then executes or runs the applet (Fig. 8.2).



(a)

(b)

(c)

Fig. 8.2
A Java Applet downloaded from the web server, and running on the client PC

This program is called a **Java virtual machine,** or simply a **JVM,** because for all practical purposes it has given us a computer that has Java byte code as its machine language. The beauty of all this is that Java byte code is fairly similar to any machine language, so you have a reasonably simple and quick translation job at the browser end.

HOW STANDARD IS JAVA?

And that brings me to the juicy bit—the political issues with Java. This is the fun part, the part where we computer professionals sit back, pick up a beer, and watch the action.

Who are the big daddies of the computer world? Well, at the time of writing this book, three of the real big daddies were Microsoft, SUN Microsystems and Oracle.

Let's first look at Microsoft's vision of this world. It is a world where Microsoft is king. So every user machine runs Windows, whether it is Win/NT or Windows 2000, or any other version. In such a world, the problem we have described with web servers does not arise, because all the machines are essentially the same.

Now Java is from SUN, which is a competitor. So what does Java attempt to do?

Simple. It ensures that all computers are capable of accessing the Internet and running the same program, whether or not they run Windows. In other words it removes the need for Microsoft Windows as a standard. So if a user likes a "Mac" he can happily buy a "Mac".

Did Microsoft like this? You bet your life they didn't. So what did they do? They did something real smart. They added additional features to Java. SUN had created what they called 100 % pure Java (rather like Milkfood in India created 100 % ice cream some time back), but Microsoft added to this. And the additional features were such that they were supported only on Microsoft's own Internet Explorer browser running on a Windows machine!

Interestingly, many web site developers started using these additional features because it gave them a more powerful site.

Aha! So suddenly Java was not the standard it was once touted to be. And therefore its usefulness came down. In fact several web developers have now cut down on using Java applets for this reason. (the other reason is that Java applets tend to be large and complex and therefore require large download time, thereby slowing down user response.)

Incidentally, a recent US Court ruling has prevented Microsoft from modifying Java, so at least the current round of the battle has gone to SUN. But battles in the IT industry are never ending affairs—so you can continue to expect good, clean entertainment!

THE NEED FOR JAVASCRIPT

One of the shortcomings of Java is that it tends to be a fairly complex language. Which means the following things:

1. Writing programs in Java is somewhat complex
2. Paying salaries to Java programmers is an expensive proposition
3. Java byte code turns out to have fairly large file sizes, and therefore it takes a good deal of time to download it from the web server onto the browser.

Of course, from the point of view of the Java programmer, nothing could be better—it gives him that exalted status, and significantly enhances his monthly paycheck.

But the IT industry was naturally not too happy about this, and therefore wanted a solution.

Fortunately, it did find a solution—and that solution is called **JavaScript,** from Netscape Communication Corporation.

JavaScript is a far simpler language than Java. It actually looks somewhat similar, and therefore the name is also similar. It cannot do all the things Java can, such as drawing complex graphs, but it can certainly do things like validations (and other invaluable things like changing the colour of a picture when you move your mouse over it).

JavaScript is not converted into Java byte code or any such intermediate language. It is in fact too simple for us to get into such complications. It is simply placed in the HTML page on the web server as a JavaScript program, and is transmitted to the client PC as it is, along with the rest of the HTML page (Fig. 8.3). Once it reaches the browser, it is translated and executed in one shot. Incidentally, this also means that the browser must have the capability to translate and execute it—which most browsers do.

One last word—JavaScript came from Netscape—which is part of the SUN gang. So what did Microsoft do?

Very simple. It created its own version of JavaScript and called it JScript. And the fun part is that JavaScript and JScript have compatibility problems as well, just as the different versions of Java do. So as you would expect, JavaScript works best with the Netscape

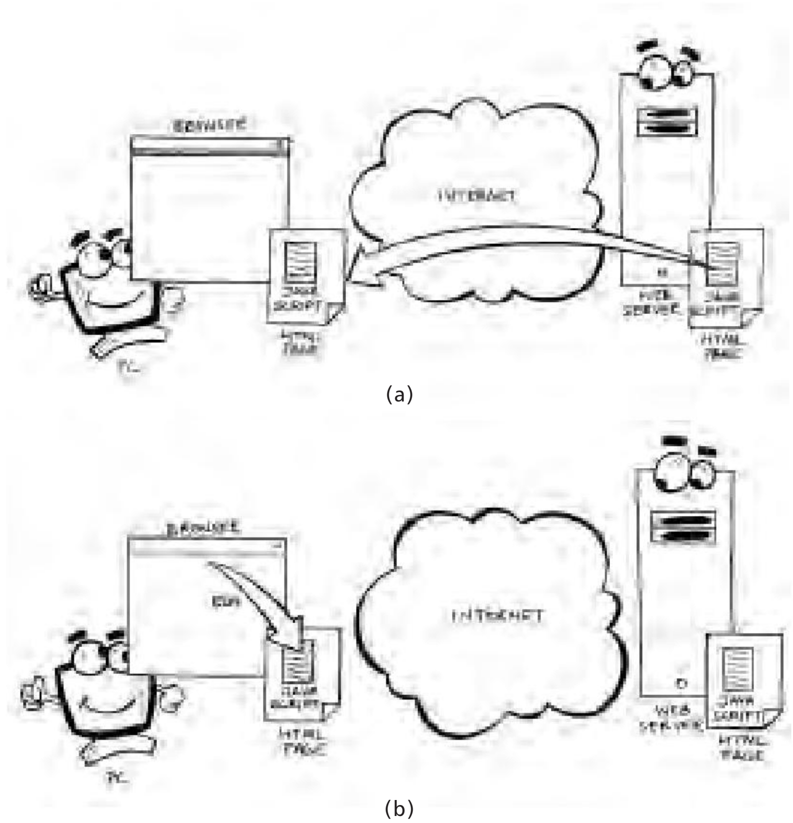


Fig. 8.3
How JavaScript works

browser, and JScript works best with the Microsoft Internet Explorer browser. Although most features of both languages work on all browsers, you do have the occasional cute message on your browser screen, saying "*JavaScript Error*".

Microsoft actually went one step further. They created another language of this kind and called it VB Script. Once again, VB Script works best on the Internet Explorer browser. After all, Microsoft is in this business for business, isn't it?

SUMMARY

And that, ladies and gentlemen, is how Java works.

Java programs need to be brought to the client machine and then run. And naturally, you have no control over what kind of client machine the user is using.

To overcome this problem, Java is compiled and stored in the form of byte code on the web server. This, incidentally, is also called a Java applet. Any browser that downloads Java applets, has a special, built-in program called a Java virtual machine or JVM, which does the job of translating the Java applet into the binary machine language of that particular client machine, and then executing it.

A truly revolutionary concept, which attempted to make Java a standard language for client side computing. I say attempted, because it didn't quite turn out that way: There are today multiple versions of Java from different vendors. But the world still carries on happily !

Of course Java itself tends to be somewhat slow, largely because downloading an applet from the web server to the browser takes time. And that led to the design of the much simpler scripting languages, such as JavaScript, VBScript, and JScript. Which are not as powerful as Java, but can do simple things like validations, much faster.

Before we end this chapter, let us take a look at our e-Commerce architecture again (Fig. 8.4).

As you might have guessed, it remains a 3-tier architecture as before. The only difference is that we now have Java applets and JavaScript, both of which reside in the

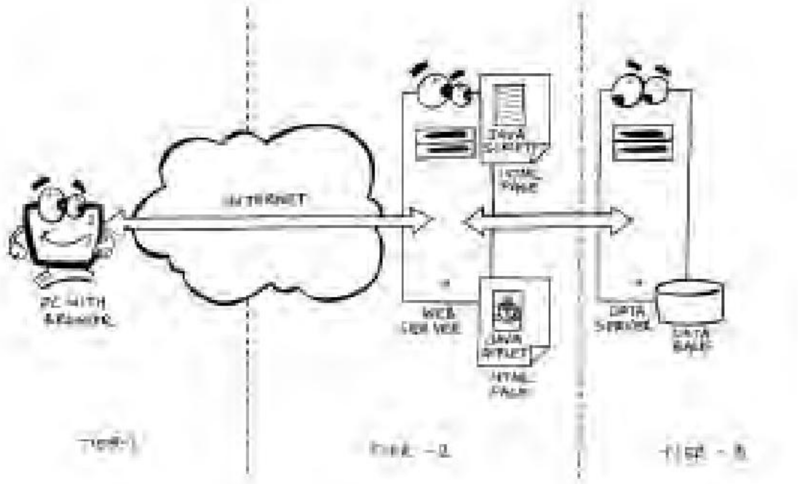


Fig. 8.4 *The 3-tier e-Commerce architecture*

middle tier, but are downloaded to the first tier and executed. As before, these take care of the presentation and business logic of the site.

So that, my friend, is Java. A great concept! It could have provided the magical portability that the world was looking for, but it's a little short at this time.

What will happen in the future?

Who knows? We have Microsoft, we have Oracle, and we have SUN. And any of them can win in the long run.

So who do we bet on?

Well, would you like to know where I've put my hard-earned money?

My money, dear reader, is on none of them. It's safely in my pocket !!!

9

Back to the Web Server—Server Side Languages

Welcome my friend. Welcome back from the world of Java ! Of applets and portability ! Of standardisation but no standardisation ! Of Microsoft and SUN and Oracle.

We saw that Java permits you to move away from server side computing, thereby cutting down the load on the web server.

But guess what? In this chapter, we take you right back to where we started—server side computing all over again. No, not to CGI scripts—those are now more or less obsolete, but to a new breed of server side languages.

So read on....................

THE NEED FOR SERVER SIDE COMPUTING

Java is a great language. It set out to do something truly noble, and to a large extent it succeeded. But it still has one fundamental drawback.

In certain situations it tends to be slow.

Why?

Let's take a very simple analogy from real life. Suppose reading this book had made you hungry, and you wanted to order something to eat. Some distance away from your house, there is a restaurant called Madras Café, which serves mouth watering *dosas*. And what is more, they make these *dosas* to your specifications.

Being the active person that you are, you do not want to walk all the way to Madras Café to get the *dosa* (or even drive—that really stresses the toe muscles). Instead you decide to telephone and ask them to deliver it to your house. In the process of course, you give your specifications—for instance you may want bamboo shoots as the main filling, in addition to Mexican beetroot and African blue onions (how on earth they would get African blue onions I do not know, but I am sure they'll find a way).

Now Madras Café has two choices. The obvious one is to make the *dosa* at their outlet and then have it sent across to you (including a trunk call to somewhere in the Sahara desert to get the blue onions, but that's irrelevant to our study of e-Commerce).

But wait. There is also another approach. Suppose Madras Café were to take their battery of cooks, all the ingredients that they required—including desiccated African blue onions—charter a bus, and have them all reach your house. Where they greet you with a beaming smile and say, "*Vanakkam* (Greetings) Sir, just let us use your kitchen, and we'll have the *dosa* ready for you in no time !!!"

Theoretically this is highly possible. But there is an obvious problem—transporting the entire '*dosa creating mechanism*' from Madras Café to your house. And it seems so silly doesn't it? It would take far too much time, in addition to the cost, of course.

But hang on a minute !

There are situations where you may want to do this. Haven't you organised parties at home where you called the cook (*or halwai*) home? And asked him to set up his infrastructure on your terrace, and make *dosas* right there?

Why did you do this? Simply because you had a party and therefore wanted lots of food. Equally important, each guest obviously wanted his food at different points in time, because each of them took different quantities of alcohol (measured in gallons). And naturally, each guest wanted his *dosas* piping hot—in other words, freshly made.

Wouldn't it be horrendously complicated, time consuming, and expensive to call up the restaurant separately for each *dosa,* and ask them to deliver it independently?

You could of course, place one single composite order for 300 *dosas,* to be delivered one *dosa* at a time at five minute intervals, but I am sure you see the associated problems!

Therefore you call the cook home.

Simple!

Now it's time to step back from the (slur-r-r-r-p) world of African blue onions and use strategic management techniques to figure out what has happened.

If you think carefully (aha, work for you again), you first need to check your requirements—do you need one *dosa,* multiple *dosas,* multiple *dosas* at different points in time, etc. etc. And the crucial bit is to figure out which option is more expensive and time consuming—transporting the *dosas*, or transporting the *dosa*-making mechanism !

Since you are a master at management techniques, you would obviously choose the option that is faster and cheaper !

How fast can you make the switch from *dosas* to the Internet? Well, you've got to do it right now. Because the way Java is used—or should be used—is very similar.

Just look upon the program as the *dosa*-making mechanism. And the results that the program generates as the *dosas*. Now the same management technique works here as well. Namely, *figure out which option is more expensive and time consuming—transporting the program across the Net, or transporting the results*. And naturally, choose the better one !

For instance, assume you were visiting a health site, where you wanted to check your lifestyle habits, and their impact on your longevity (or lack of it). You would then typically answer several questions on the web page such as the number of chickens eaten in a day, bottles of whisky consumed, minutes walked every morning, and so on.................... And the result, naturally, would be a long sermon on how terrible your lifestyle was, how unlikely it was that you would live beyond a few months, and how pathetic those few months would be.

You would probably be shocked to discover this (as if you hadn't known it already) and would attempt to answer the same questionnaire again, perhaps cutting down the number of chickens eaten in a day, to see if that impacted longevity. Leading to probably another long sermon, but perhaps a somewhat different one.

And so you go on and on and on attempting to make changes....................

....................until you finally arrive at the perfect lifestyle. And then of course, you sit down with a drink in front of your TV, satisfied in the knowledge that you can do it !

Can you see a parallel between the world of chickens and the world of *dosas*? There is a program that takes as inputs details of your lifestyle, and creates an appropriate sermon—the results of the program. So the key issue is the cost of getting the program across the Internet versus the cost of getting the sermons across. In this case, since there is one program but several sermons, one for each combination of number of chickens and bottles of whisky, the program is likely to be easier and quicker to transmit. And therefore Java may be the answer.

On the other hand, if the designer of the web site expects people to use the site only once and then give up in disgust, there would be one long program but perhaps a shorter sermon, delivered once. Transporting the results here would be quicker than transmitting the program. And therefore, **server side programming** may be a better option.

To generalise : If the program is likely to be used only once by each individual, it probably makes sense to use server side computing, to avoid sending a big program over the Internet. But if it is going to be used repeatedly, the bandwidth required to transmit the results of the program would probably be far higher. And therefore, a Java applet downloaded onto the browser and running constantly, is probably the answer.[1]

As another example, you could recall Chapter 7, where it made sense to transfer a Java applet to your PC when you wanted to plot a graph of the Hindustan Lever share price. Simply because sending a complete graph (and therefore a picture) would consume too much bandwidth: The applet in this case would probably be much faster to send than the graph. Therefore in this case it made sense to use client side computing with Java, and not server side computing.

What's the moral of the story?

Simply this— Java applets are a great option in some cases. But there are several situations where you would rather have a program running on the server !

But hang on. Doesn't that bring us back full-circle? To the world of CGI? Which we said was slow?

Dear reader, you are both smart and wide awake. You are absolutely right. We do need a solution that runs on the web server, like CGI. But unlike CGI, this solution needs to be fast.

And that's exactly what we'll see in the next section.

ACTIVE SERVER PAGES AND JAVA SERVER PAGES

In Chapter 6 we saw that CGI scripts running on the web server had a very fundamental problem: They were only usable by one user at a time. So, every time a new user wanted to access a CGI script, the web server would create a new copy and run it. And we saw that this increased the load on the server tremendously, and therefore made the whole process very, very slow.

But now we have come back to saying that in some cases we do need programs running on the web server.

[1] There are, of course, other factors. We have simplified the logic as usual. But in the interest of clarity I'm sure you don't mind.

What's the solution?

Obvious, isn't it? We need to find a way of writing and running programs on the server so that they do not create copies of themselves. Instead, the same copy is accessed by multiple browsers simultaneously. If this were possible, the load on the server would come down greatly and therefore speeds would go up.

And now the good news. This *is* possible. In fact there are several technologies available for running such programs on the web server. And most of these come from either good old Microsoft, or equally good old SUN.

The first of these technologies is called **Active Server Pages** or **ASP**. ASP is a method of embedding programs in the HTML page kept on the web server. When the page is called by the browser, *these programs are executed before the page is sent* (see Fig. 9.1).

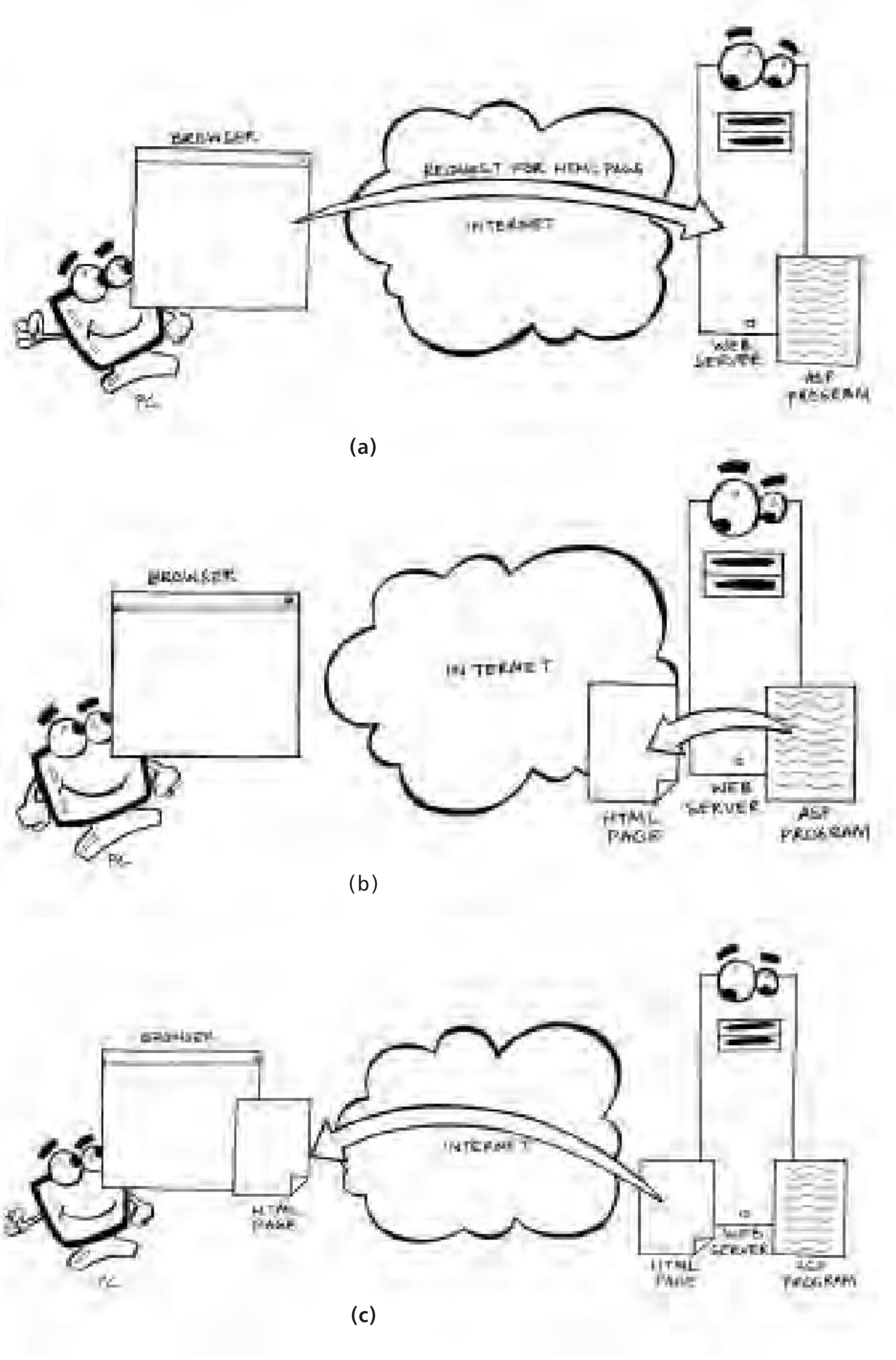
Like CGI scripts, ASP programs can be used to access data bases, the results of which are placed in the HTML page before it is sent to the browser. Also, like CGI scripts, they could be used to perform calculations before the HTML page is sent. For instance, picking up Hindustan Lever share prices from a database, computing your total wealth based on the number of shares you hold, and putting the good (or bad) news in the HTML page for you to see.

How is this done?

It's actually quite neat. Recall Chapter 1 where we mentioned that HTML consists of tags. We also mentioned that each tag formats the text (or pictures) in a certain way. ASP inserts special **processing tags** which do not format text, but instead call a program, which is then run on the web server. And the results of this program are inserted into the appropriate space in the HTML page.

The fundamental difference between ASP and CGI scripts of course, is that ASP does not create copies of itself (among other differences of interest to, perhaps, Computer Science students). All browsers access the same copy, so the load on the server is minimised.

The other interesting thing, which ASP makes possible, is personalised web pages. For instance, assume that you have reached the web site of shadynews.com, the most authentic news site world-wide, for news on all kinds of subjects. Now you have decided that your interests are in the area of movies, and of course the magic word "SALE" wher-



(a)

(b)

(c)

Fig. 9.1 How "Active Server Pages" works

ever it occurs. On the other hand, politics and sports simply do not interest you. So on your home page you would like headlines from movies and clearance sales, without a single mention of what such and such minister said yesterday (of course that would make it a most boring site, but then you asked for it).

Now shadynews.com has headlines in all areas that it covers. But by using ASP, it selects only those headlines which you want, puts them into the HTML format, and hey presto, gives you your home page (Fig. 9.2).

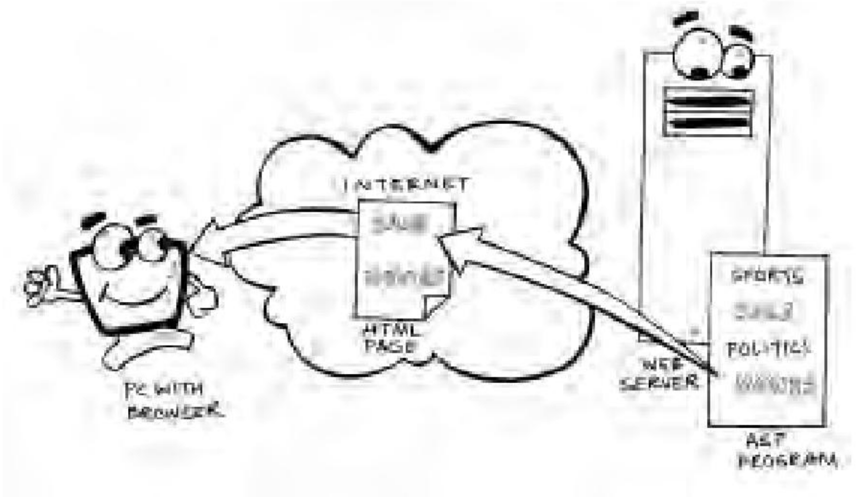


Fig 9.2 *Using ASP to select headlines and create a customised page*

You must have seen this kind of personalisation on many, many sites. And ASP is one of the technologies that makes it possible.

The only issue is that ASP runs only on Microsoft's web server (currently IIS or Internet Information Server). No other web server supports the language.

Question: If Microsoft has done it, can SUN be far behind?

Obviously not ! So SUN has come out with a similar technology, and, rather unimaginatively, called it **Java Server Pages**, or — you've guessed it — **JSP** for short. Like ASP, JSP also runs on the web server, and has special executable tags that call Java programs.

And the rest of course you know.........................

Summary

Let's summarise.

We have several options for programming web pages :

- We started off with server side programming using CGI scripts. We also mentioned that this led to a lot of load on the web server.
- We then went to Java which moves programs to client PCs as required, and then executes them there. This could again be slow in some cases, because of the time required to download the Java applet.
- And finally, we saw how to go back to server side computing using ASP and JSP, but at the same time overcome the speed problems of CGI scripts.

At this stage it's a good idea to take a fresh look at the e-Commerce architecture that we have been building up so far (see Fig. 9.3).

As you can see, the three tiers or layers remain intact, but we have added some more options to each tier.

So, in tier 2, which is the web server tier, we have the option of ASP or JSP, in addition to the CGI scripts we had earlier. As before, these take care of the presentation and business logic of the site.

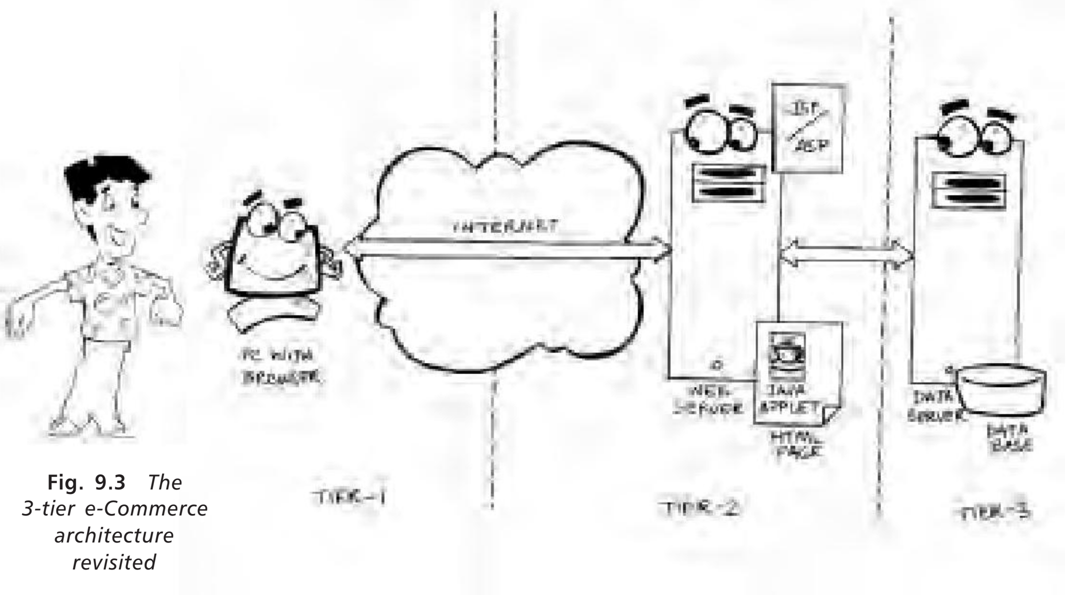


Fig. 9.3 *The 3-tier e-Commerce architecture revisited*

And now the final question. We have seen lots and lots of options for programming web pages. The question is, which one do you use?

Obvious, isn't it?

When you have a huge program with small amounts of data, you would probably use server side programming. But when it's a small program, with lots of data to be transferred, Java would be a good option.

There is of course another major issue to be checked before you take this momentous decision.

In today's world, who is the most unstable employee in your organisation?

It's your good, old programmer, of course ! Because he gets a new job offer every week.

So pamper him. Keep him happy. And ask him, in all humility, "Sir, which technology would you like to use?"

10

The Building Blocks of the Net—Component Based Development

We have looked at a lot of new technological concepts in this book. And this is probably a good time to check on whether you're still with me or have got lost somewhere a few chapters ago.

If you're lost, I suggest you go right back and re-learn. But if you are still with me, great. Let's continue!

You would have noticed that we have been talking about a number of different programming languages, such as Java, JSP,......................

Now, in the good old days when many of us used to do computer programming, most of us would take pride in the number of *lines of code* we could write in a day. There were the laggards, there were the so-so programmers, and then there were the Olympic champions.

In today's world of Java and JSP however, this expertise is dying out—a little like the expertise in writing Hieroglyphics died out. No one expects you to write everything from scratch any more. Today, people expect you to use components.

What are components?

Good question, my friend. Read on, and you will discover...........

THE NEED FOR COMPONENTS

There are two well-known facts about the software industry:

First, software takes time to develop—lots and lots of time. Any reasonably complex system would take months to develop and get operational, if not more.

And if there is one thing that companies today do not have, it is time. They are under pressure from competition, from the market, from customers, and from just about everyone else.

Secondly, software is expensive. It requires expensive programmers, more expensive designers, and still more expensive analysts to develop it. And naturally, the longer it takes to develop, the longer these analysts and programmers need to be paid, and therefore the more expensive the software.

Again, if there is one more thing that companies do not have enough of, it is money. You may be aware that most companies have shareholders, and most of the bigger ones have made the abominable blunder of offering their shares to the public. What happens at their Annual General Meetings, where the shareholders turn up in hordes, is not funny: If the company has not made a fat profit, it is torn apart. If it *has* made a big profit, the shareholders clap—and simultaneously raise their expectations for the next year. Which means............you've guessed it ....they tear the company apart next year, naturally !

So we have a time problem and we have a money problem. And yet we insist on wanting software!

There is, of course, one simple solution to this problem—don't use software!

But it is probably a bit late for that. Had you been a senior manager in a multinational firm around the time of Akbar the Great, that may have been a wise decision. But today you probably don't have a choice.

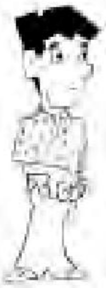
So what's the answer?

"The answer, my friend, is blowing in the wind" (with due apologies to Bob Dylan). It's all around us, if only we could see it.

And to figure out this answer, let us take an example from the world around us.

Most of you who do not have the good fortune of living in the serene and tranquil environs of the Himalayas would, at some time or the other, have installed a cooler in your home. (If you haven't, please go ahead and install one, or get a complete dump from a friend who has. Also, please let me know the secret of your survival in the cool months of May and June, in the plains of India).

Those enlightened readers who have installed a cooler would know that coolers have several components or parts (Fig. 10.1). For instance, there is the pump, the body, the fan, the stand, the regulator, the water tank, etc. etc.............



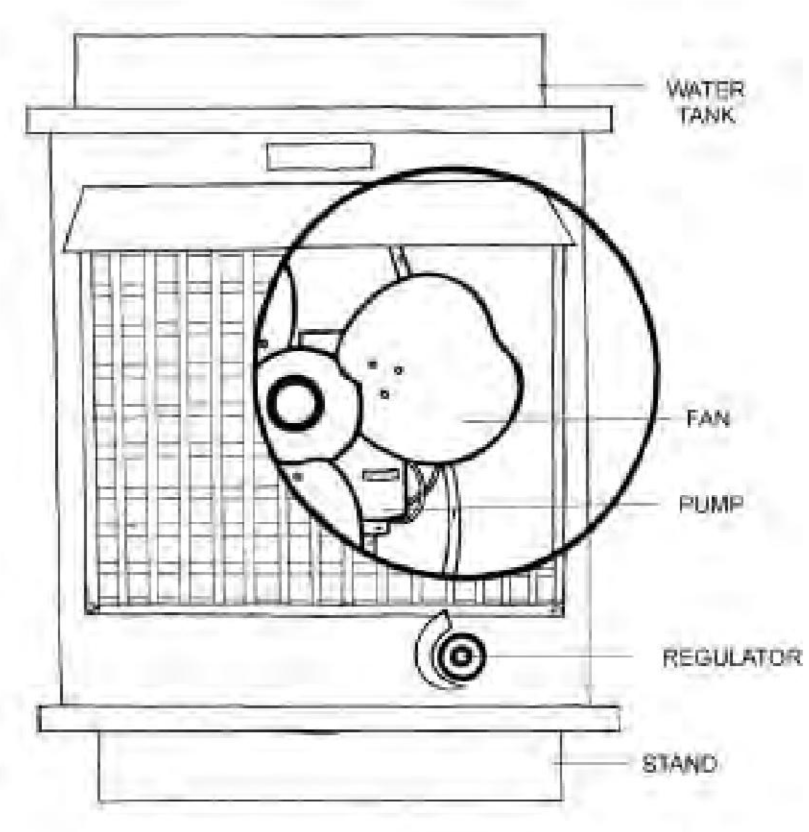


Fig. 10.1 *Components of a cooler*

Question: Do you build the pump from scratch? In other words, do you sit down with a metal sheet, wires of various specifications, and ferrite cores, and start by winding the wires around the ferrite cores?[1] Of course not. That would be quite silly, wouldn't it? You would simply go out and buy the pump—unless of course you had made a ten rupee bet with a friend, and were desperate to win those ten rupees.

Similarly, you would not make the fan, or the regulator, or even the pipes. Each of these is a **component** of the cooler that you would buy from the market.

[1] Practising Electrical Engineers should forgive me if there are any factual errors in the preceding statements. I used to be an Electrical Engineer about a quarter of a century back, but today I'm a mere Software professional. In any case this book is on e-Commerce and not on the subject of winding pumps and motors!

Now each of these components has some functionality—in other words it does something that it is supposed to. So the fan pushes out air, the pump raises the water to a certain level, and so on. And each one of them works with the other components to form the complete system—in this case a cooler.

Notice therefore, that for each component, we need to specify the following :

(a) What the component is supposed to do—for instance, the pump is supposed to raise water, the regulator is supposed to vary the speed of the fan and has three different speeds, etc.

(b) How the component is supposed to work with other components—so the regulator would vary the voltage that is applied to the fan, which would then vary its speed. Or the fact that if you have a 36 inch fan, you cannot possibly have a 20 inch body—if you did, where would you fit the fan?

(c) How the component does its job—for instance the fact that the pump has a single phase induction motor. However, this part is really not for you and me. It is more for the techie—the electronics freak who has no girl friends but has plenty of electric motors.

In other words, you as the master assembler of the cooler would need two kinds of information for each component of the cooler:

(a) What the component is supposed to do

(b) How it is to be used in conjunction with other components

Hopefully, if all manufacturers agree, we could decide on a common format or framework in which to specify both of these. And this common format is what we call a **component framework**.

Dear reader, after reading this treatise on Electrical Engineering, if you are still awake, you might have noticed the following things about components:

(a) They dramatically cut down the time, effort, skills, and cost of creating (or shall we say developing) complete systems or products. For instance, you do not need the expertise and skills to wind a motor, and you don't need to spend the time either.

(b) They can be bought from different manufacturers, as long as the manufacturers agree on a common format for specifying how their components will work with others.

(c) A component created for one purpose can also be used elsewhere. So the same pump could be used in your cooler, or in the water fountain in your farmhouse (I assume you are one of the lucky few who own a farmhouse. Or if you are not so lucky, to pump up water to your third floor flat). In other words, components are re-usable, provided of course you use them for the purpose for which they were built. Of course, you would need several pieces of the same pump, but it's the same model, the same make, etc. etc.

And now for the magic !

If all this is crystal clear to you, just do something very simple. Substitute the word *software* for the word *cooler*. And you have cracked the problem.

Just as we have mechanical or electrical components such as pumps and fans, we also have **software components**. Each such software component is written to perform some function. For instance, you could have a component that takes as input, details of your income, and computes your tax liability. Or you could have a component that takes as input, daily price variations in some share, and produces as output a graph which shows this price movement, minute by minute.

Isn't it natural that software components will give you the same kinds of benefits that the other *physical* components do? So, for instance,

(a) They dramatically cut down the time, effort, skills, and cost of developing complete software applications, because you use pre-created pieces

(b) They can be bought from different software vendors, as long as these vendors follow a common component framework.

(c) A software component created for one purpose can also be used elsewhere. So a component that plots a graph of share price movements across the day, can also be used by the Meteorological Department, to plot a graph of the inches of rainfall, minute by minute, on the day you go out for your annual picnic.

Figure 10.2 shows you how software can be built using components. And once again, you as the software developer will need two kinds of information for each component that you pick up and use in your software:

(a) What the component is supposed to do

(b) How it is to be used in conjunction with other components. In the software world, this means the data that needs to be exchanged between this component and the others it interacts with, the format of this data, etc. etc. etc......................

As an example, the component that plots share price movement as a graph would require for each minute, the share price, possibly the number of shares sold at this price, the number of shares that were offered but could not be sold, etc. What would also be

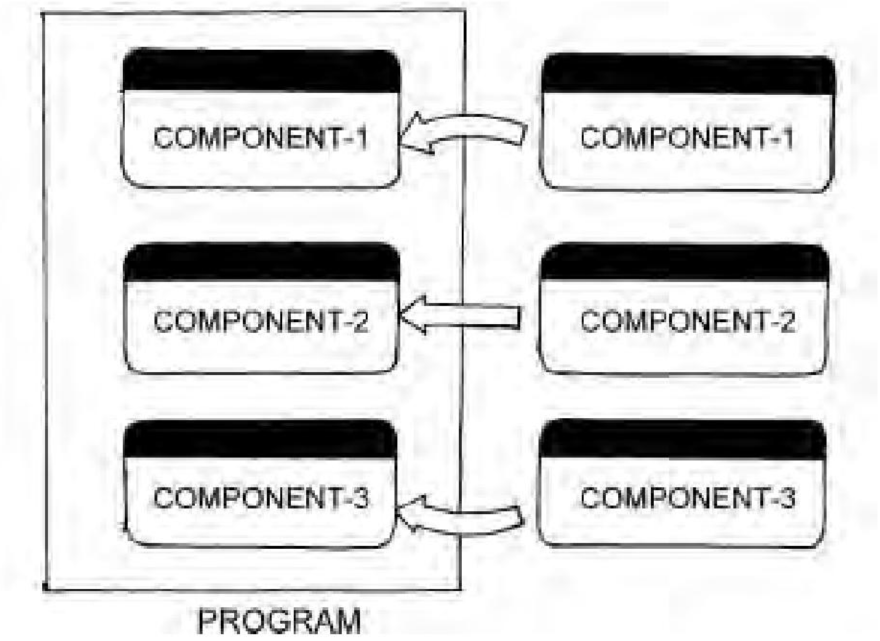


Fig. 10.2
Software built using components

specified is whether this share price is in rupees or in thousands of rupees, whether it is correct to the nearest paisa, or simply rounded off to the nearest rupee, etc.

Once again we need a common method of specifying this information about software components, and this is also called a **component framework**.

Incidentally, components need not always be bought. Consider your cooler again: You may have a terrific vision of a cooler stand with carved legs.[2] Obviously you will have to get someone to build (develop) these carved legs for you. Similarly, in the software world, there are several things you may want, for which no one has developed an appropriate component as yet. Or perhaps someone has, but he lives in the southern-most tip of Chile, and for some strange reason you are not aware of his existence. In other words, you will need to write a program to perform this particular task. And in the process, you create your very own component. Which can be re-used elsewhere if so desired (just as you can

[2] Now, I know of no existing cooler stand with carved legs. Neither do I see any reason for carved legs, because the legs would be outside the house and not even visible. But then, it's your money and you have a right to spend it the way you want.

use the carved legs in your sofa, your bed, or even—provided you have terrific imagination—the bathroom sink).

JAVA BEANS

One great example of a component framework is the **Java beans** framework (I have been trying to figure out whether there is any connection between Java beans and coffee beans, or even the vegetable, beans, but have failed so far. Let's just leave it as an interesting name for a most useful concept). This framework permits us to create components called, once again, **Java beans**. So the framework and the components are both known by the same name. Naturally, Java beans are written in Java. And they satisfy all the requirements we have looked at in this chapter. Namely:

(a) Each Java bean specifies what it is supposed to do

(b) It also specifies the interface with the outside world, or with other beans.

For a moment, let us go back to Chapter 8 on Java, where we talked about Java applets. As discussed, a Java applet is a Java program that has been compiled into Java byte code, so as to run on a Java virtual machine.

If it's a simple applet, the programmer would probably write out the program himself. But if it is a complex applet, he may want to pick up pieces which have already been created by other programmers, put them together, add some programming of his own, and thereby create the applet.

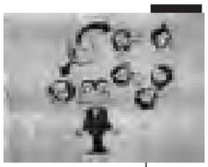
Now doesn't that remind you of something? Isn't it exactly like building a cooler, where you take a ready-made motor, fan, regulator, etc. and put them together, instead of constructing them from scratch?

In other words, doesn't it make eminent sense to create complex applets by putting together components—some of which may have been created by other programmers, or even other organisations?

And that is where Java beans come in ! To combine components into one big piece of software requires that you know exactly what each component does. And that is provided by the Java beans framework. So take the appropriate beans (if you can find them somewhere), put them together along with the beans that you yourself have created, and you have a larger Java bean !

Compile it down to byte code, and you have your applet ! And incidentally, creating JSP is very similar !

SUMMARY



This chapter was all about coolers, wasn't it?

But hang on a minute—if you've been reading it carefully, we actually went beyond coolers. To the world of components. Where we said that components were a wonderful way to create software, because you could simply pick up ready-made components and put them together to create an application. You could even re-use your own components in different applications.

Of course, you would need a common framework within which components are written, so that different components are able to understand and communicate with each other. We called this a component framework, and we also saw that Java beans formed an example of such a component framework.

And that, my friend, is the solution to the problem of both time and cost in the software industry. Beg, borrow or steal components, and write programs from scratch only where you have no choice (meaning no components available).

Or prepare to face the same fate that befell our poor ancient Egyptian experts in the Hieroglyphic script.

11

Enter the Big Guns—Application Servers

Dear reader, we have come a long, long way from the first chapter of this book. And hopefully you are at least marginally wiser than you were when you started. Terms such as Java, CGI scripts, web servers, HTTP, domain name servers, and a variety of similar animals probably sound more English than Greek to you now.

And this is just the right time to bring in the real killer. A term that would put any of these other miserable technologies to shame. A term that would catapult you into the league of *Gurus* at your next party.

Yes sir ! We are talking about the hottest subject this side of the Nile—the application server. A subject that is always talked about, but generally mis-understood. And you my friend, will hopefully both understand and be able to talk about it.

So fasten your seat belts, and let's go!!!

APPLICATION SERVERS

First things first. What is an application server?

To understand the term, let us for a moment go back to the cooler we had examined in the previous chapter. If you recall, we had several components which were used in building up the cooler, namely the pump, the fan, the water tank, the regulator, the various pipes, etc.

Now let's ask you a question: Are there any common services, which are used by these various components? And which make the components themselves simpler to design and use?

Sure there are !

For example there is the electric power supply. Which is used by the pump, the fan, and the regulator. None of these components generates its own electricity. Instead they all assume that a common power supply exists and each of them uses it as required. Can you imagine how horrendously complex it would be if the pump were to first generate its own electricity? Or the fan for that matter?

Similarly, there is the water supply—the pump, the various pipes, and the tank all assume that the Municipal Corporation provides this water supply—none of them picks up water from the river, purifies it, and then carries it home !

In other words, there are a few common functions that most components require. Getting each component to individually perform these functions is expensive, involves a huge amount of replication, and makes it far tougher to design these components. Instead, providing some of this functionality as common services makes the components simpler, easier to design, and much, much cheaper.

Cut to the Internet.

Can you see a parallel?

We have seen that many of the programs that run on the web server are written in languages such as JSP. And these programs are generally created using software components such as Java beans. Now surely there must be some functions that are common to several components, just like our power supply and water supply. And if we can find such common functions, wouldn’t it make sense to provide these as common services to all components? So that our software components can now be made simpler, and far cheaper?

Yes there are !

And all these common services are provided by a special piece of software that we call an **application server.**

Let us look at some of these common services.

REMEMBER ME?

For a start, let's go back to the site sabziwala.com run by our friend Haribhai, which sells fresh (and not so fresh) vegetables on the Internet.

Assume you have logged on to the site and have filled in unnecessary personal details such as your name, address, and preferred time for delivery, along with critical information such as your father's name, date and place of birth, names of your grandmother and grandfather, highest degree earned, etc. All this has been done by filling in details on a form thoughtfully provided by Haribhai (I believe Haribhai was considering asking you to fill in this form in triplicate, in keeping with Government regulations. But the Boston-based consultant who was advising Haribhai on e-Commerce, convinced him against the idea).

At the end of the form, assuming your telephone link has not been disconnected, you click your mouse, and Voila ! The information is sent to the site as one HTTP request.

Now you get to the part where you have to select the vegetables you wish to buy. So you start by choosing green potatoes (that's the only kind Haribhai sells. And incidentally, green potatoes are a delicate, acquired taste—so by now you probably cannot eat the regular variety. Haribhai calls this "building customer loyalty").

After selecting these potatoes, you naturally click on the submit button, and wait for the potatoes to be added to your shopping cart. So a second HTTP request goes off to the web server.

But this is where the fun begins! *The protocol we use over the web, namely HTTP, does not remember the past.* So the fact that a form has been filled up with green potatoes, along with whatever information you had put into that form, (such as quantity) is sent to the web server all right. But the web server has no way of linking it to the earlier request. In other words it does not know who ordered the potatoes ! Both HTTP requests are treated as independent !

You see the problem, don't you? And next, when you order sorry-looking cauliflower, the past has again been forgotten. Therefore the web server gets another form with details of the cauliflower you keyed in, but it has no way of finding out that the potatoes and cauliflower are part of the same order, and from the same customer.

Now Haribhai's site is quite popular. All the neighbours and indeed people from neighbouring colonies are onto the site at the same time. So what does Haribhai see? Lots and lots of forms, each being one HTML page. And no connection between these forms.

Mass confusion, because he has no way of figuring out who has ordered what!

Terrible isn't it?

There has to be a solution.

One obvious solution is to type in your user name (or user id) and password on every page that you send. While this is a terrific way to exercise your fingers, it's a little silly isn't it? Wouldn't it be better if there were some other way of remembering which page came from which PC?

Fortunately there is. And it uses a cute thing called a **cookie**—no, not something to eat—but a software cookie.

When you first log on to a web site and give your personal details, the web server keeps all these details in a series of tables in its database (remember our friend the database from Chapter 6?) And it also creates a pointer to your details in this database. A pointer is simply the location of your personal details in the database (similar to the address of a house in a colony), so by using this pointer, the web server can get to your specific data.

Wonderful. What next?

This pointer is now sent to your PC and is stored in a special folder there. And this pointer is what we call a cookie. In actual practice, cookies can contain a lot more than just pointers—but that's of interest to Computer Science students. For our purposes, a cookie is a pointer!

For some strange reason, the folder where all cookies in a PC are stored, is not called a cookie jar, but we'll leave that debate out of the purview of e-Commerce for the moment. Incidentally, this is a simplified explanation as usual.

Now what happens? Every time you access the site, the web server asks for the cookie. It reads the cookie, uses it to access the database and says, "Aha, so Feroze A. Darukhanawala is back, or whatever it is your name is !!".

In fact, every new page you click on and submit to the web server, goes along with the cookie. So the server is able to link up all the pages you send, and figure out that all these vegetables have been bought by Feroze Darukhanawala.

Now let us get slightly technical. Each interaction of yours with one web page is called a **session**. And the session ends when you receive the page from the web server. What we are saying in technical terms, is that HTTP has no way to link sessions with each other—in other words, there is no **session persistence**. Cookies are a wonderful way to overcome this problem, so that sessions can actually be related to each other.

Obviously, managing session persistence is a commonly required service. Which involves adding your details into the data base, creating a cookie, sending it to your PC, asking for it next time, using it to access the data base, etc. etc. etc.

Complex, isn't it? So instead of writing this part of the program afresh for every component, why not make it available as a shared service? In other words, we could have a common piece of software that takes care of session persistence, cookie creation, cookie management, etc. So each programmer will not need to create this program afresh (see Fig. 11.1).

And this common software is part of......................

...........................you've guessed it—an application server.

Rather like the electric and water supply, isn't it?

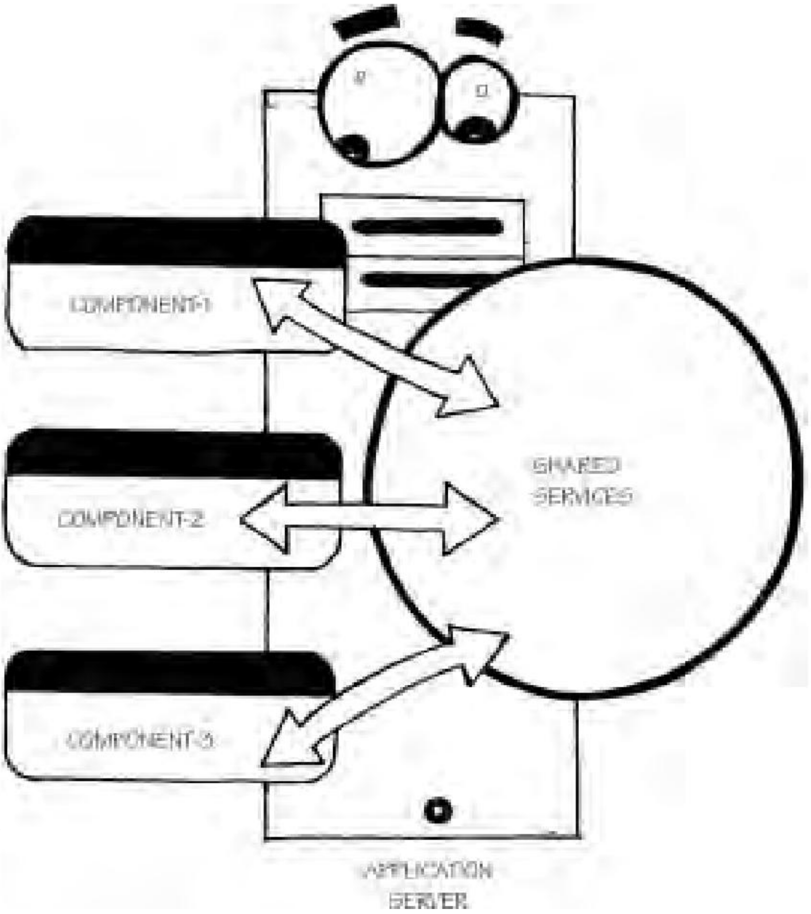


Fig. 11.1
Components sharing services provided by the application server

THE 4-TIER ARCHITECTURE

Now watch what is happening very carefully.

We are actually dealing with two kinds of components.

The first kind is components which do not assume that any common services exist, other than the JVM of course. These are the Java beans kind. As discussed earlier, they could be used to create either Java applets for execution on the client machine, or JSP programs for execution on the web server.

And then we have those components that assume the presence of common services provided by the application server. In the Java world, these components are called Enterprise Java beans, or simply EJBs. And, the framework which supports these EJBs is called the J2EE framework. Incidentally, these application servers are also called J2EE compliant application servers. Commonly available servers of this kind are WebLogic, WebSphere, and iPlanet.

Figure 11.2 shows how the application server fits into our 3-tier architecture that we have been discussing so far. Notice that it actually forms a separate layer between the web server and the data server. And therefore the 3-tier architecture has now become a 4-tiered one.

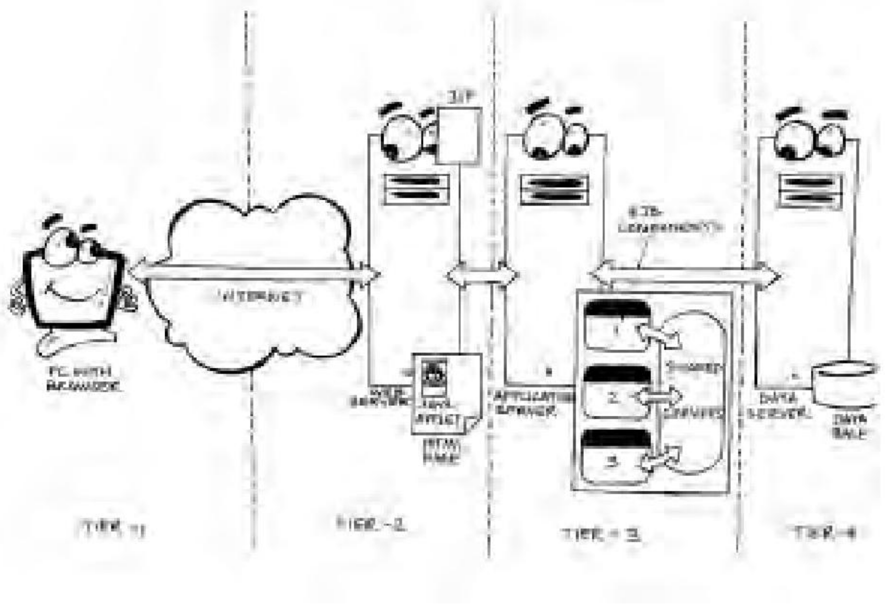


Fig. 11.2 *The 4-tier e-Commerce architecture*

The application server usually handles the business rules of the web site, and the web server takes care of the formatting and presentation (which would perhaps be done using JSPs as before). In other words, the application server does the database access and calculations, and the web server puts everything into the appropriate HTML pages for the user to see.

SIMPLIFYING DATABASE ACCESS

Is that all that application servers do? Or are there other common services that they provide to components?

Of course there are, so let us look at another information-related problem. The problem of accessing the database where all the information is stored—about vegetables, about customers, and about Rupee to Yen exchange rates (Haribhai has some Japanese customers, you see). And here, as you would probably have guessed, I'll take yet another analogy.

What happens when you hire a car?

You would typically land up at the parking lot of a car rental agency—and as an example I take one of the most well known ones—"Smashed Car Rental". Next, you would fill up one form (if you are lucky—I have filled up four at times).

After this you would be asked to show your driving licence, perhaps your passport, and your credit card. The card would be swiped into a card reader, after which one of two things may happen. Either the transaction gets authorised, in which case you get the car. If not, you would probably hear police sirens within three minutes, because someone they've been attempting to track down for months, has finally been located.

We hope it is the former. After all, you need to be out of jail to implement your great new e-Commerce system, don't you?

How much time does all this take? Well, it would probably depend on who the representative of Smashed Car Rental is. If she is a young woman, and you are a normal, healthy young man in the age group of 70 to 90, it may take a couple of hours—and at the end of it, you may even have a passenger for your car. On the other hand, if it's a grumpy old man, and you are a business traveller in a tearing hurry, it is likely to be shorter (and much less pleasant). But it would still take a significant amount of time because it's a complex operation, with lots of data capture and verifications!

Is there a solution? Actually there is, but to my knowledge, no one has implemented it so far. Suppose we had a middleman between Smashed Car Rental and you. This middleman takes up, say, ten cars and says, "I will take care of all the paperwork. And incidentally, I know several of these customers, so the paperwork actually gets cut down." So you pick up the car from the middleman, and not from Smashed Car Rental.

Which naturally means that the time to pick up a car is reduced (once again subject to the earlier condition—the representative should not be a young woman, and you should not be a young man in the age group mentioned earlier).

And now we come to the really crucial bit—when you return the car, the middleman does not return it to Smashed Car Rental ! Instead, he retains it, so that when the next customer walks in, he can quickly rent out the car to him.

Can you see the crucial role the middleman plays in this case? He holds a few cars with himself, so that it is quicker and more efficient to rent out a car to the next customer.

Now let's look at a similar problem in e-Commerce: The problem of getting access to the database. When you want to check the prices of vegetables on Haribhai's site, you would need access to the database. Normally that would require checking your user id and password, figuring out whether you have the right to access the database, and also which parts of the database you have access to. For instance customers in Chile may not have access to highly perishable vegetables because of problems in transporting them from India. In addition, we would also need to check what you are allowed to do with the database (as a customer you can only see prices, but Haribhai and a few other privileged people can change the prices), etc. etc. etc. !

Do you see a pattern?

Of course you do. Getting access to a database is not very different from hiring a car, and it is complex and time consuming.

And therefore a middleman would help.

And pray, who's the middleman in this case?

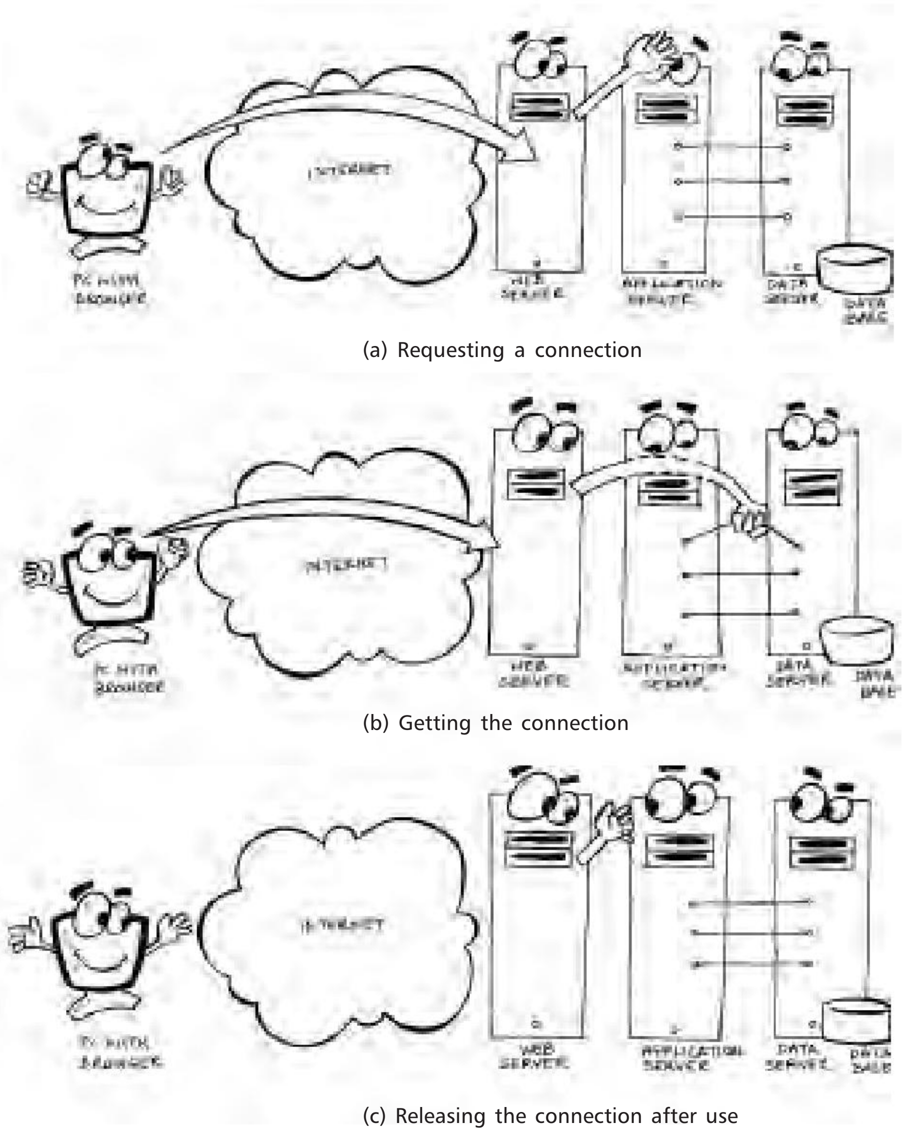
No guesses. Our very own application server of course !

Here I am again constrained to introduce one more technical term. Let's go back to our car rental middleman for a minute, and try and understand what he is doing.

Actually, he is creating some sort of car pool, where he has picked up a certain number of cars, which he keeps circulating amongst his customers. The application server is doing something very similar. It is taking up a certain number of **connections** to the database, and these are shared amongst the users of the system.

There is an obvious term used for such a function performed by the application server—and it's called **database connection pooling** !

Once again, database connection pooling is a common service that the application server offers to all components. Instead of getting programmers to write the code that performs this function repeatedly for each component (see Fig. 11.3).



(a) Requesting a connection

(b) Getting the connection

(c) Releasing the connection after use

Fig. 11.3 *Database connection pooling*

Before I forget, there is something else that connection pooling does. Let's go back to Smashed Car Rental and see what is happening inside.

The owner has decided that he doesn't want to employ more than three salesgirls. He has also realised that each salesgirl can deal with four customers simultaneously, but not more (how on earth she can deal with four, I have no idea, but that's the owner's problem, not yours or mine). Now, if you were to take out your calculator (the advanced one which gives you accuracy up to 25 decimal places), it would not take you more than an hour or so to figure out that this translates to 12 customers simultaneously, at the agency.

What happens if there is a rush one day? Perhaps everyone's personal car has been involved in a smash-up, and therefore suddenly everyone needs a car on hire. And they all land up at Smashed Car Rental at the same time.

What happens to the poor salesgirls? How do they deal with a hundred different customers, all hollering together?

Simple. They don't. They just give up and sip their Cola (or Diet Cola, depending on certain key measurements).

And what happens to the customers? Huge delays, lo-o-o-o-o-o-o-o-o-ng wait times, and frayed tempers, naturally.

What is the solution?

Our good old middleman of course !!!

He takes care of the paperwork. He takes care of managing hundreds of customers simultaneously (how he does that would probably be terribly exciting to someone who is planning to set up a car rental agency, but it's not relevant to this book, so we'll skip it for the moment). Most important, *he ensures that not more than twelve people from his office talk to the Smashed Car Rental salesgirls at a point in time*—so the salesgirls do not have a problem.

Switch back to the Internet. Can you see some kind of similarity? Databases are traditionally not designed to work with too many users. In fact, most database vendors would charge you on the basis of the number of users who were likely to use the database simultaneously. This figure would usually be in the range of 5 to 100.

And there's the problem: On a typical Internet site, you may have thousands, maybe tens of thousands of users accessing the site simultaneously. So how does this poor, overworked database manage so many simultaneous users? After all, it is not designed to manage more than ten or at the most a hundred of them?

You know the answer of course. It's our good old middleman, the application server. Using connection pooling once again. The application server manages the connections of a huge number of users simultaneously on the Internet (assuming of course that people spend a lot of time on-line but most of this time is spent thinking and reading—the actual time spent in accessing the database is very tiny). And on the other side, it ensures that these connections are shared amongst the few connections that the database provides (Fig. 11.4).

So that is one more common service provided to components by the application server!

Easy isn't it?

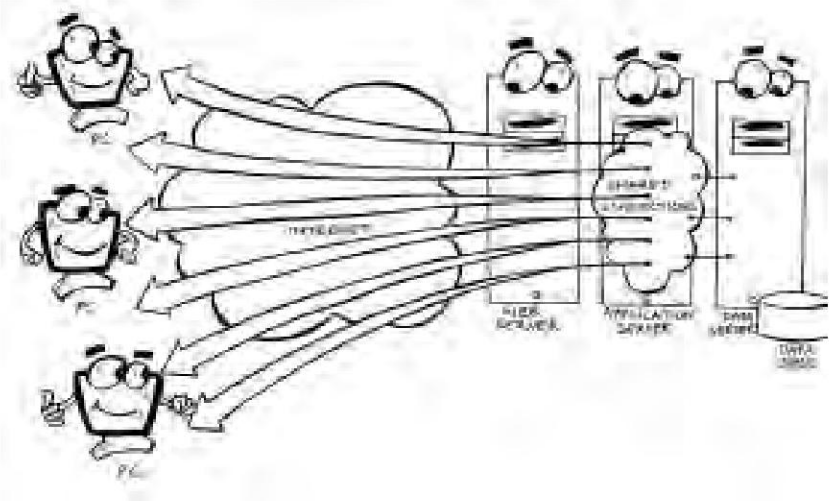


Fig. 11.4 *Many people on the Internet sharing a few database connections*

LOAD BALANCING

There are, in fact, several other common services that application servers provide. But we'll take a look at just one more.

Have you ever been to McDonald's? Of course you have. And you have obviously noticed that at the counter they have several smiling young people whose job is to take your order and execute it.

Why do they have several young people? (Actually it could even be several surly old people—our e-Commerce learning would still be the same.) For the simple reason that McDonald's expects a large number of customers, and having just one single person to serve them would be interminably slow.

Since each of these young people is assigned the job of serving customers, we could easily call him or her a server. And all of them together would probably be called a **cluster of servers.**

Cut to the Internet. In exactly the same way, servers on the Internet can either exist alone or in clusters. And the purpose is exactly the same, namely to push up speeds. So you could have a cluster of web servers or application servers or even data servers.

Switch back to McDonald's once again. What happens if the queue in front of one of the servers is much longer than the queue in front of the others? Would the customers in the long queue wait that much longer than the other customers?

Obviously not ! You would probably have a manager hanging around, whose job is to ensure that some people from the longer queue switch to the shorter ones. So that everyone gets attended to in roughly the same amount of time.

We call this process **load balancing**, and it is identical to what happens on the Internet when you have a cluster of servers. In other words, you have a manager who switches requests to different servers within the cluster, so as to equalise loads and therefore response times.

And for both clusters of application servers, as well as data servers, the manager in question is the application server itself.

Once again, a common service which every programmer does not have to build into his components.

DCOM

All that we have discussed so far refers to the world of Java. But what about Microsoft?

Actually, the logic we have been using remains virtually the same when you shift to the world of Microsoft. So, just as we have EJB compliant application servers in the Java world, we have **DCOM** compliant application servers in the Microsoft world. And components written for these servers are called **DCOM** components. In fact the latest version from Microsoft is **COM+** components.

Naturally, these components handle the business logic on the application server. With ASP programs handling the presentation and formatting on the web server, as before. So we have a 4-tier architecture as before, but this time with Microsoft technologies (see Fig. 11.5).

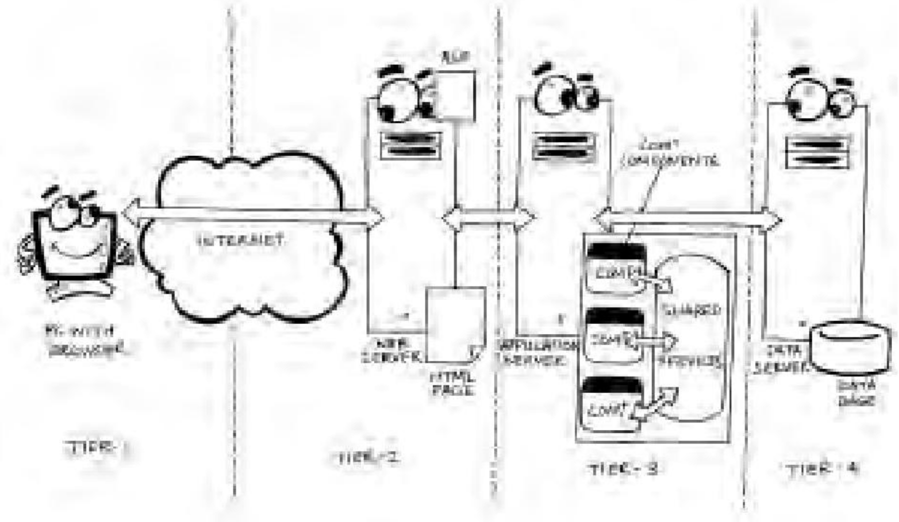


Fig. 11.5 *4–tier architecture using Microsoft technologies*

Summary

And that, my friends, is the wonderful application server ! Which provides a set of ready-made services, to be used by any component running within the application server. For instance, it is able to keep track of transactions from one HTTP request to another by using cookies. So if you ordered a book on-line by clicking on one web page, and like a good, honest citizen, you made payment for this book by clicking on another web page, the site would agree to deliver the book to you and not to your friend (who also happened to be passing through the same book site around the same time).

Or, we could share a database, which permits only 50 or so simultaneous connections, across several thousand users, by using connection pooling. And of course, we could also do load balancing across servers, so that no transaction had to wait too long for its turn.

We looked at EJB components, which run on J2EE compliant application servers. And then we looked at COM+ components from Microsoft, which run on COM+ compliant servers.

Interestingly, we saw that application servers create a fourth tier in our earlier 3-tier e-Commerce architecture, by coming in between the web server and the data server. And therefore our e-Commerce architecture becomes a 4-tiered one (Figs. 11.2 and 11.5).

One last comment. You would have realised from the preceding discussions, that application servers are useful in large, complex web sites, which have many concurrent users. In other words, application servers help you get scalability–which simply means that the same site can be used by more and more people without any significant reduction in speeds and performance.

But what if you have a small site? Where scalability is not required at all?

As you can imagine, most of the functions that application servers provide are not even required now—functions such as load balancing, connection pooling, and so on.

So the answer is simple–no application server required. Stick to the web server and the data server. And stick to ASP or JSP or even good old CGI scripts. And be happy you did not have to spend on an application server—which, by the way, could be expensive.

And that's a good thought with which to end this chapter. Application servers are good for large sites, which need to be scalable. But for small sites...................... use your judgement my friend ! The same, sound judgement you used when you invested in this book.

12

Internet Security Made Simple

What is the first thing you think of when you hear the word security? Spies in thick, dark overcoats with upturned collars? Black Cat commandos? Or huge security guards with handle-bar moustaches and a double barrelled shot gun?

Well, on the Internet, while security is a big issue, I suspect none of these will help. Instead, you have innocuous sounding things like sand boxes, firewalls, and certificates.

What are these? And how do they solve security related problems?

To get the answers to these questions, simply read on ....

SECURITY CONCERNS ON THE INTERNET

Let's start at the very beginning. Naturally, with a question: Why is security such an issue on the Internet?

Actually there are several reasons, but the biggest by far is the huge connectivity that the Internet provides.

Look at it this way: If you were at a railway station with, say, twenty passengers, it is unlikely that you would have your pocket picked (if you do, on such a sparsely populated platform, you probably deserve it anyway). But if you have a few thousand people on the same railway platform, all pushing and shoving each other, you have several hundreds of people who come in contact with you. And of these, anyone can be a pickpocket, can't he?

The Internet is similar. You have hundreds of millions of computers connected together on the Internet, each potentially able to reach all the others. Some of these are manned by good, honest citizens like you and me, but many of them are not. And each of them is able to reach you, whether you are a server or a client.

That's a sobering thought, isn't it?

Unfortunately, I cannot give any solutions to the security problem on railway platforms (other than advising you to travel strictly by one of the two following modes—by air, or by bullock cart). But I can and will share with you some commonly used solutions to the Internet security problem.

To start with, remember that there are essentially three elements in the Internet, namely:

- The client
- The various servers (web server, data server, and application server)
- Transactions or messages between clients and servers

Given these three elements, it doesn't take too much wisdom to conclude that there is a security risk associated with each of them.

In fact, each of these three elements requires very distinct security solutions. So let's put on our thickest black overcoat, turn up our collar, wear our darkest goggles, and get into the world of security.

But first, a note of warning. The purpose here is not to make you an expert in security, but to give you enough examples so that you can talk sensibly to a vendor who is selling you security solutions. At the same time, we assume you are familiar with simple things like passwords (if not, please switch on a Windows machine, and you will come to know what a password is). So in the interest of cutting down the pages of this book, we will skip such topics.

CLIENT END SECURITY

What are the risks that clients on the Internet face?

There are actually several, but let us look at a couple of the common ones:

(a) Active Content

This sounds real techy doesn't it?

Well actually, it is fairly simple. And to understand it, let's go back to Chapter 7 on Java, where we saw that web pages loaded onto a browser had essentially two parts: First of all, the HTML content, which is simply text formatted in different ways depending on the HTML tags used. Secondly, there is the **active content**.

“Aha !”, you obviously say, “What’s this active content?”

Actually it is not very different from human beings. There are the passive variety who lie on their sofas all day, their primary activity being switching TV channels using the remote control. And then there are the active ones like your neighbour’s son, who are forever jumping up and down, or cycling, or practising cricket with a ball in an old sock till well past midnight. In other words, they keep doing something.

Active content is very similar. It refers to a program embedded in an HTML page, which does something. In fact it may not even be embedded in an HTML page. It could come to your machine as an e-mail attachment, or even as a plug-in for your browser. The key is that it comes to your machine and starts running or executing there.

I’m sure you are familiar with such active content. In fact, Java applets do precisely this. They come to the browser and run faithfully. Exactly the way their master (the programmer who wrote them) asked them to run !

And that, my friend, is where the problem begins. *They run in exactly the way their programmer intended them to.*

Because their programmer could be untrustworthy !

For instance, what if his primary purpose were to see the confidential files on your computer and delete them? Or even send them to his own computer? Or change them, so that poor you are very happy with wrong data on, say, your tax planning?

Doesn’t that get your blood boiling? Sure it does ! It’s like a thief getting access to your home and doing whatever he wanted. Worse, in this case you have actually invited the thief in, by downloading the web page !

Incidentally, such a malicious program, which gets access to your computer under the guise of a totally different purpose, is called a **Trojan horse.** If you remember your ancient Greek history, you would immediately see the connection. If you don’t—no problem ! Just remember the name and stick to e-Commerce.

The long and the short of it is that we need a solution. Yes sir, we definitely need a solution !

Fortunately, solutions exist. And we will look at one possible solution here. This one is from the world of Java.

Java categorises applets into two types: Trusted and non-trusted. **Trusted applets** are those that come from a trusted source. And **non-trusted applets** are those that come from an unknown source (it may be perfectly honest and well-intentioned, but it is unknown and therefore not to be trusted yet!)

How do you know the source from which the applet is coming?

Actually, identifying the source of a web page is a subject in itself. It uses something real exotic sounding, namely **digital certificates**. Since digital certificates are key to the subject of security, we have dedicated an entire chapter to them—Chapter 15. For the moment, let's just assume that there is a way.

Obviously, if you know that the applet is from a trusted source, you would probably permit it to execute freely on your client machine. It's like trusted guests in your house—you certainly wouldn't follow them all over the house, would you, just to check on what they were putting into their pockets?

But untrusted applets—now that's a completely different story. You just don't know what they might do under the garb of drawing a graph of the intra-day share price movement on your PC. They may even run away with your salary details. Worse, they may pass them on to the Income Tax department !

So how do you prevent this from happening?

The solution is actually derived from a very quaint analogy. Controlling an untrusted Java applet is rather like controlling a mischievous child. When you leave such a child alone, you never know what trouble he will get into. So you would ideally like to construct walls around him—or box him in, so he does not go outside these walls ! And since he will probably howl the place down, you would need to give him something to play with—and what better than a heap of sand? In other words you put him into a **sand box.** He can do anything he wants to within the sand box, but he can't climb out of it. So for instance he cannot access those juicy apples growing on the trees of your surly neighbour. Or even the wonderful glass statue in the lawn, which is only a teeny-weeny bit fragile.

Great solution isn't it?

And that's exactly what Java does. It asks untrusted applets to run in a sand box (Fig. 12.1). These applets have no access to any resources on your PC ! So they cannot access any of your files, whether it is to read them or delete them or modify them. They simply operate on data that comes in from the web server from which they were downloaded.

Eureka ! That solves the problem, doesn't it? So trusted applets run freely on your PC. But untrusted ones run without any access to your files. And therefore the

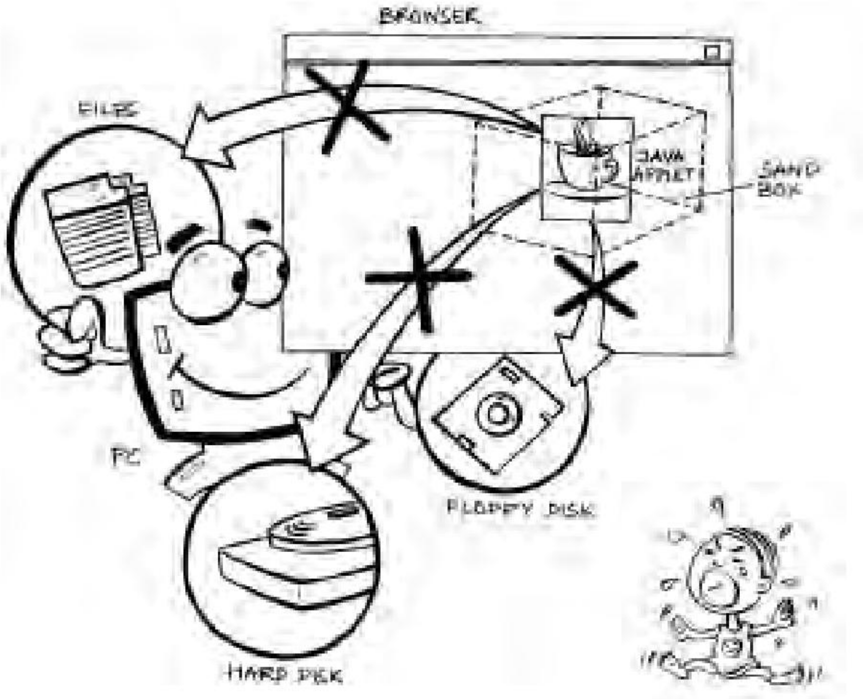


Fig. 12.1 *Java applets in a sand box environment*

magnificent dreams of the programmer who wrote them—the dreams of being a top-notch hacker—unfortunately remain fulfilled.

More important, Java continues to be used without fear, by all and sundry.

(b) Virus Protection

Who hasn't come face to face with viruses?

There are all those wonderful people around the world, whose sole ambition is to make life difficult for others by creating newer and newer viruses. These viruses either delete your files, or make your programs go haywire, or do something as elementary as calling a halt to your PC.

Fortunately there are organisations that develop what is called anti-virus software. Which scans your PC for several known viruses and "disinfects" it.

Virus protection is a subject in itself (maybe I'll write a book on it later). But just remember—you've got to have anti-virus software running on your PC.

What if you have downloaded anti-virus software once? Is that enough?

I wish it were. But alas, the wonderful people who create these viruses keep creating newer and newer types of viruses, day by day. Just to see the havoc they wreak on the poor, unsuspecting, Internet-linked world.

So what do you do?

Fortunately, the organisations that develop anti-virus software are equally smart. Every time a new virus is invented, these organisations get to work—and determine a brand new antidote for it !

Which simply means that you need to keep upgrading your anti-virus software on a regular basis. And of course follow a few simple rules. Like not installing pirated software (which is often highly virus prone). Or not inserting diskettes into your PC, unless you were 800% sure of their contents. Or not reading e-mails from unknown sources—especially those with attachments.

Many a brave man has flouted these rules. Only to see his PC, his hard disk, and his files eaten up by yet another of those prowling organisms.

So be careful my friend. You can never be too safe in the world of viruses!

SERVER END SECURITY: THE WONDERFUL WORLD OF FIREWALLS

And now from your PC (the client) we go all the way to the other end. And look at server end security.

Whichever server you use, there is one watchdog that you are bound to use. And that is a wonderful thing called a firewall.

But what is a firewall?

Actually it has no relation to fire eaters (who are generally found in circuses), but instead is a very fundamental concept in security. And as always, I will take an analogy to explain the concept.

Assume that you were in the habit of storing large sums of money at home. Say, a couple of millions of dollars in used currency notes in your pillow, one more million just under the toilet flush, and a few millions scattered across the kitchen cupboards.

Now I have two questions for you. First of all, how did you manage to get so much money? Would you be willing to share it (the method of getting it, not the money) with friends such as me? I am sure the publisher of this book would be more than happy to get you to write a book on the subject.

However, all that is irrelevant to the subject of this book. What is far more critical is the second question: Namely, how will you ensure security?

There is at least one answer to this question which should be fairly obvious—build a huge wall around your house. Robbers, thieves, dogs etc. would all have to stay out, thanks to your wall.

There is also the problem of all that wonderful money being burnt up in a fire—so the wall must be fire-proof as well. In other words, what you really need is a **firewall** (Fig. 12.2).

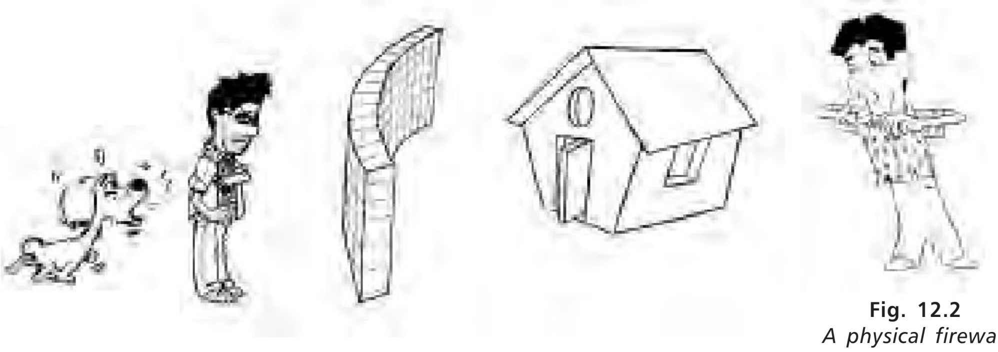


Fig. 12.2
A physical firewall

Now there is one problem with this firewall. Unless you have plans to become a hermit (many people do—nothing wrong with the ambition), you would need to allow at least some people to cross this firewall of yours and come into your house. For instance, your mother in-law. Or your friends. Or your boss (except if he is coming to check whether or not you are genuinely sick, after applying for sick leave).

In other words, the firewall needs to be intelligent (like the person who lives behind the firewall). It has to protect you from unknown and therefore untrusted visitors. But it must permit known and therefore trusted visitors (unfortunately, in this world the largest number of visitors are those who are known and untrusted, but we will ignore this fact of life. We will simply define two categories of visitors—the trusted ones, and everyone else).

Cut to the Internet. Let us take, say, the database server. Would you like every Tom, Dick and Harry to access your database? Obviously not—you may have terribly confidential information in it. Pricing information for instance. Or information about all your customers, which a competitor would just love to get his hands on. And therefore you need a firewall in front of your database server (see Fig. 12.3). Only, this firewall is not a physical wall. It's a piece of software running on a server.

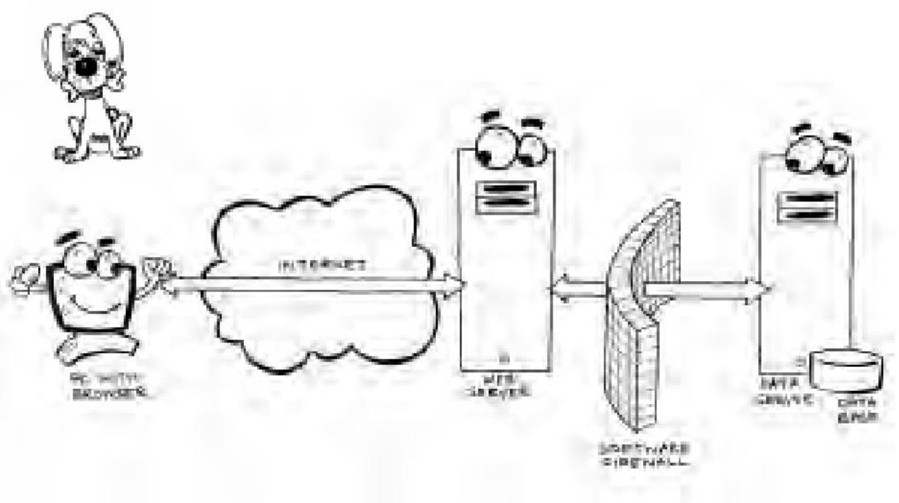


Fig. 12.3
A software firewall

But do you want to block everyone? If no one can access the database, what use is it anyway?

Valid point. You would definitely want the web server to be able to access the database (so that Haribhai's customers, for instance, can see the prices of the vegetables they are buying). But no one else from the Internet should be allowed access.

How do you ensure this?

One of the simplest methods is to use the IP address. The firewall could be programmed to pass through any packet coming from the web server's IP address. In other words the firewall checks the source IP address of each incoming packet. If this is the IP address of the web server, it is allowed to go through. All other packets are blocked.

What about the web server? Do we need a firewall in front of it as well?

Let's ask a question: Are we going to debar some PCs on the Net from accessing the server?

No, obviously not ! Unless you have some sort of family enmity and want to keep out that family—but we'll leave out that possibility for the moment. So you want every PC on the Net to be able to access your site.

But for what purpose?

You are quite happy if they access your site to see a web page, or perform any of the actions that they may be permitted to on this page—such as making purchases. But do you want them to change the web page on the server? Obviously not ! That's a right that you can only give to the web administrator—the person in charge of the web site.

Once again you have a firewall, but this time a different type of firewall. This one permits anyone to make a request using the HTTP protocol, but not using any other protocol ! HTTP does not permit the user to modify what's on the web server—for that you would need other protocols—and so you are safe !

The interesting thing you would have noticed is that we have looked at firewalls at two different levels. One is at the IP level in the TCP/IP suite (remember Chapter 3?). And the other one is at the application (or HTTP) level.

Naturally, firewalls can be far more complex, and generally are. But the principle is very similar—define rules which permit some people and debar others. And then implement those rules.

SUMMARY

So that was the cloak and dagger stuff—security. For a start, there is client end security. Which means, among other things, protection against viruses. It also means protection against active content. Which is content—or programs—downloaded from somewhere and running on your PC. Such as Java applets. Naturally, if this content runs on your PC, it gives the programmer who wrote the content a wonderful chance in life. Because he can now say, "Aha ! I can see whatever you have on your PC." Better still, he can steal some of what you have, such as your credit card number and other insignificant details.

So we created the concept of a sand box environment for a language like Java. And said that if an applet was downloaded from an unknown (and therefore untrusted) source, it would run within the sand box on your PC, and would not get access to any of the other resources on the PC, such as your files !

Then of course we looked at server end security, and examined the concept of the firewall. Which acts as a kind of barrier to anyone not authorised to cross the barrier.

Firewalls could be constructed at multiple levels : They could be at the TCP/IP level —so that messages coming from specific IP addresses are allowed to go past the firewall, but messages coming from other IP addresses get blocked. Alternatively, firewalls could be at the application protocol level. So for instance, HTTP requests to the web server are permitted from anywhere (and therefore any user can access a web page on this server). But no one can change a web page—except of course the poor sop who is in charge of the web site—and I'm sure you wouldn't grudge him this privilege !

How does this change our e-Commerce architecture? Well, if you go back to the 4-tier architecture we had described in Chapter 11, we now have firewalls at various points. For instance, we could have a firewall in front of the web server. And we could also have another firewall in front of the data server (Fig. 12.4).

So we've looked at security at the client end. And we've also looked at security at the server end. And now the biggest question of the lot—how do you ensure security of a message that is flowing across the Net? In other words, how do you ensure security of transactions?

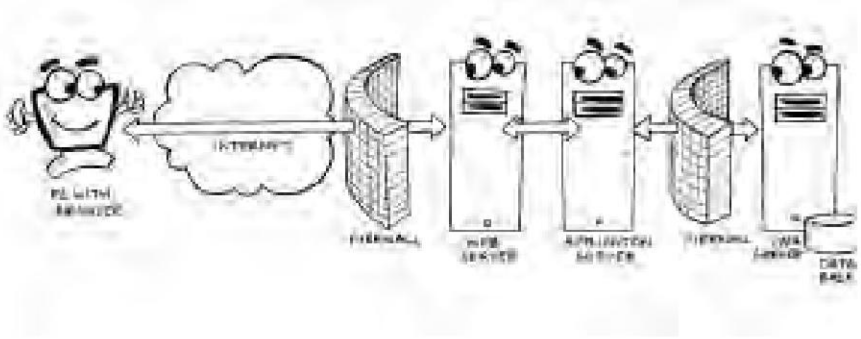


Fig. 12.4 *The e-Commerce architecture with firewalls*

Good question. And it obviously deserves a good answer. Therefore we have devoted three whole chapters to it—the next three, in fact.

So shut the book and relax. We'll meet again in the next chapter.

13

Internet Security Made "Not So Simple"–Encryption

Welcome back from your well-deserved break.

Since you have taken a look at both client end security and server end security in the previous chapter, it is now time to examine the third one. Namely, message or transaction security.

THE FOUR PROBLEMS OF MESSAGE SECURITY

Let us start with an example as usual. You get a message over the Internet from your girl friend, saying :

"Darling, I love you, and will do anything to marry you".[1]

Now this is an extremely commonplace message and I am sure most of you would have received it many times in your life—from several different sources of course.

However, as you may have guessed, it does have certain security risks.

For instance, someone else other than you may read the message. No real problem, unless that someone else happens to be another girl friend. Or worse, it could be your wife—and then, boy, are you in trouble !!!

So your first concern is to ensure that only you, as the person to whom the message has been sent, are able to read the message. We call this, the problem of **confidentiality** or **secrecy**.

[1] I have used this message at the risk of offending lady readers. However, as you go through this section, you would immediately realise that it could easily be a message from a gentleman to a lady. As you might imagine, the impact on your knowledge of e-Commerce would probably be the same, either way.

Next, how can you be sure that your girl friend (and no one else) has sent this message?

It could even be your wife, who is out to check on your activities, and simply wants proof before setting a friendly Doberman or Alsatian after you. So my friend, it is vital to figure out whether the person (or for that matter the site) that you are dealing with, is actually the person (or site) he claims to be. If not, you are face to face with the problem of **spoofing—**someone pretending to be someone else.

Therefore you must first authenticate the person or site. And this is the second problem with security, namely the problem of **authentication**.

What's the third issue?

Well, suppose someone were to intercept the message mid-way and change it. In this case it may be a competitor (her other boy friend) who may not be able to read the message but simply changes it to something more pleasant, such as:

"You rogue ! I hate you and I never want to see your stupid face again."

Here two possibilities come to mind : If you genuinely wanted to marry this young lady, the changed message might even drive you to suicide. On the other hand if you did not, and were actually contemplating two other women, you might heave a sigh of relief, and thank God the problem sorted itself out.

For the purposes of explaining e-Commerce, we will assume the former situation. In other words it is important that the message received is actually the message sent. And this is the problem of **integrity**.

And finally, let's look at the most tragic scenario of all. Your girl friend (your favourite girl friend, that is) does send you this message. And it does reach you without getting modified in any way. So the first three problems do not exist. In your excitement, you ditch all your other girl friends and land up at her house, all set to take her to the marriage registrar (or Pundit, whoever you have more faith in).

But by the time you reach, she discoveres the most "*gorgeous hunk of mankind*"— and that too, unmarried. So she says to you,

> "Message? What message? I never sent you any message. Anyway, I have a date now—good night and good-bye !!!"

Other than deeply sympathising with you or whatever is left of you, what's the moral of the story?

Simply this: There is a fourth problem that needs to be taken care of in such situations. The problem where one person, or site, or organization, denies having sent the message he did send. And we call this the problem of **non-repudiation.**

So that's it. Four meaty, juicy problems for us to chew over. Namely:

(a) Confidentiality
(b) Authentication
(c) Integrity
(d) Non-repudiation

Now you may be thinking, "But hold on. This situation is highly artificial. It doesn't happen like this. And anyway, how on Earth is it connected with e-Commerce?"

Well, that's not difficult to answer. You may be applying for a job through a placement site on the Net. Do you want your boss to see it? Or you are carrying out a major deal with a client, through your e-Commerce site. Do you want your competitor to see it? No? Well then, you have understood the issue of confidentiality.

Next, you get a message from the web site of your placement agency, congratulating you on landing the job. But is this truly their site? Or is it a rival placement agency, wanting to make sure you quit your job, so they could then start shoveling jobs at you. You see? The problem of authentication !

Or, take the biggest business deal of your life. Your competitor changes the terms in the proposal you are sending to your prospective client through the Net. The client is surprised, and happily agrees.......................leaving you to howl over your losses. Can you see the problem of integrity in action?

And finally, you go to the web site of your good old share broker, and ask him to buy 1,000 shares of Hindustan Lever Ltd. at Rs 300 a share. But the very next day the price goes up to Rs 500. And your good, honest broker says, "But you never placed such an order !" Simply because he wants to pocket the fat profit made. Conversely, the share price crashes to Rs 150, and you deny that you had ever given the broker any message to buy these shares—leaving the poor broker in the lurch. So that's the fourth problem, namely, non-repudiation.

Wonderful. We have just tackled the first part of the problem. In other words, you have at least accepted the problems. And the rest is easy—we simply have to solve them !

CONFIDENTIALITY AND ENCRYPTION

First of all, let us take up the issue of confidentiality. How do you send a secret message (because that is what confidentiality means)?

Very simply, by **encrypting** it. In other words by doing something to the message so that it doesn't make sense till the reverse of this "something" is done to it.

As a very simple example, let's take a message such as,

"Buy one thousand Reliance Shares !"

A simple (though not smart) way to encrypt this message would be to replace every letter by the next letter in the alphabet. So you would replace the letter "a" by "b", "b" by "c", "c" by "d", etc. And that gives us the following scrambled message :

"Cvz pof uipvtboe Sfmjbodf Tibsft !

Terrific, isn't it? Anyone intercepting and reading this message would come to one of the following conclusions :

(a) It is a message in Russian (or maybe Hebrew)

(b) Most of the vowel keys in your keyboard have got stuck, or

(c) You are boiling mad, and these are swear words that some child proofing software on the Internet has edited.

Who would ever guess that you wanted to buy Reliance Shares !!!

Now all you need to do at the receiving end, is to reverse the operation (which we call **decryption**). In other words, replace each letter by the one preceding it in the alphabet, and Hey Presto, you've got the original message all over again.

The only problem with this is that the moment someone guesses that you have replaced each letter by the next one in the alphabet, your entire secrecy is gone. Everyone in the neighbourhood knows that you want to buy Reliance shares (not that there is anything illegal or immoral about your actions !)

This was a very simple, even trivial, example of encrypting a message. And the science of encryption is called **cryptography**. It usually employs several mathematicians, especially the long-bearded variety from St. Petersburg in Russia. And these are usually the kind of mathematicians who absolutely drool over the fact that they have finally discovered two 200-digit numbers which are mutually prime. Not that it makes any difference to mere mortals such as you and me !

Of course, it is vital to any communication, and therefore e-Commerce over the Internet.

At this point, I would like to introduce a new term : I'll use the term **encryption algorithm** for describing the process of replacing each letter with the next one in the alphabet. (Incidentally, that's not a spelling mistake: algorithm is a perfectly valid English word—check it up in the dictionary). Naturally, encryption algorithms are usually much more complex than the simple one described here—and are therefore much harder for a hacker to crack. Sometimes such an algorithm is also called a **cipher** (not to be confused with cider, which is a kind of drink).

So the next time you need to encrypt your message before sending it across the Internet, what would you do?

Get a new cipher, naturally. And equally naturally, it needs to be different from ciphers used by anyone else—otherwise this other person would be able to read your confidential message !

Aha! Do you see a problem coming up?

No?

Okay. Let's take an analogy. Notice that encryption is rather like locking up a door or a suitcase. No one can open it unless he has been allowed to do so by the person who locked it. Taking the analogy further, suppose you had to buy a lock. You would typically go to your neighbourhood hardware store and ask the shopkeeper,

"I want to buy a lock."

What if he were to say,

"OK, give me your specifications—how many levers, level of security required, etc. etc."

And immediately thereafter, he rushes off to the back of his shop and calls up the company that manufactures these locks. Only to return after ten minutes or so, beaming and saying,

"Great, sir, it's all fixed. It will only cost you ten thousand Rupees, and will be ready in a month. You can give me five thousand Rupees as advance right now !"

What happened?

Simply this: *The shopkeeper asked the manufacturer to make a new lock for you*—and naturally that would be both costly and time consuming.

But is that what happens in real life?

Of course not ! Lock manufacturers would go out of business in that case. And thieves would suddenly find business highly profitable.

What the manufacturer does instead, is to create many, many locks of the same kind, but with different keys. So they all look the same, and feel the same, but the key to one lock will not open another lock of the same kind. And the basic idea obviously, is to cut down the effort, time and cost of making a special lock for every suitcase.

Now do you get the idea?

Of course you do !

Developing an encryption algorithm is a complex job (don't be fooled by the highly simplified example I have taken). And therefore doing this afresh for every message or every individual is like making a lock all over again. So the natural solution is to create one algorithm but make available multiple keys for this algorithm !

To continue with our previous example, the algorithm or cipher might say,

"Replace each letter by the letter which is "X" positions away in the alphabet."

"X" here is the key and in this case it has the value "1".

Now, another person may have exactly the same algorithm but his key may have the value "3" so he cannot read your messages, and vice versa.

My God, you might say—I thought I had picked up a book on e-Commerce. But this is Algebra ! And I have just come away from breaking my head with my daughter's class VII Algebra book.

But remember, my friend—remember what your class II teacher used to tell you, "Whatever you do in life, you cannot escape from Mathematics." Remember?

And therefore, much against my wishes, I am constrained to bring in this level of Algebra into e-Commerce.

To summarise, therefore, encryption has two components (see Fig. 13.1):

1. An encryption algorithm or cipher, which could be common to a huge number of people. In fact, ciphers are publicly known. Some of the commonly used ones are called DES, and the stronger Triple DES (it certainly sounds stronger, doesn't it?)
2. A key, which is unique to you, or to a site, or to a software application. Naturally, it will never be 100% unique (so for instance, you may have a web site in Brazil with the same key) but for all practical purposes it is. Importantly, this key is a secret between you and the receiver of the message.

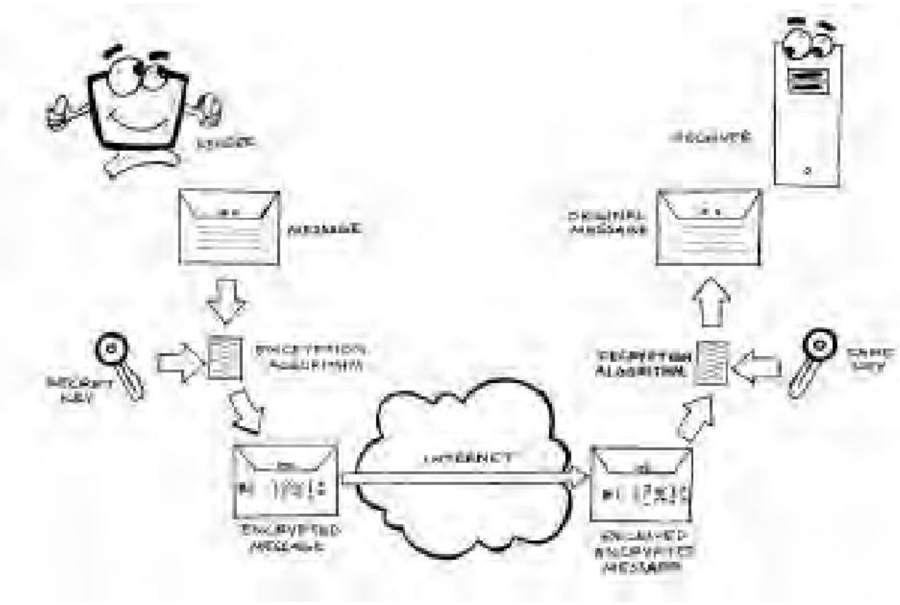


Fig. 13.1 *Encryption using a key*

PUBLIC KEY CRYPTOGRAPHY

Now, as per our usual practice, let us add one more level of complexity. You as the sender of the message, are sending an encrypted message to someone else, who we call the receiver. The algorithm is publicly known, but the key is uniquely known only to you.

Have we forgotten something? Your friend at the other end needs to decrypt the message, which means he needs your key.

No problem—just send the key to him over the Internet (or over the phone, or fax, or any other communication medium for that matter).

But is it so simple? Now the key itself could be intercepted. And once the intruder, whoever he is, has access to your key, he has access to all your confidential messages.

What's the solution?

Naturally you encrypt the key—using some other key, which in turn could get hacked.

As you can see, we are ending up in a never-ending cycle, from which there is no escape (doesn't this feel a little like life?)

But my friend, there is an escape. There is actually a most wonderful method of escape devised, as usual, by mathematicians. And the method is called **public key cryptography**.

What is public key cryptography? It's a concept similar to that of Noah's Ark. Namely, everything in life must be paired up, otherwise it doesn't work. In this case, keys need to be paired up. And one of them is called the **public key**, while the other one is called the **private key.** Believe it or not, the public key is the one which is freely accessible to everyone, whereas the private key is known only to the owner of the key pair.

Special encryption algorithms have been devised which work with these key pairs, in the following manner (see Fig. 13.2) :

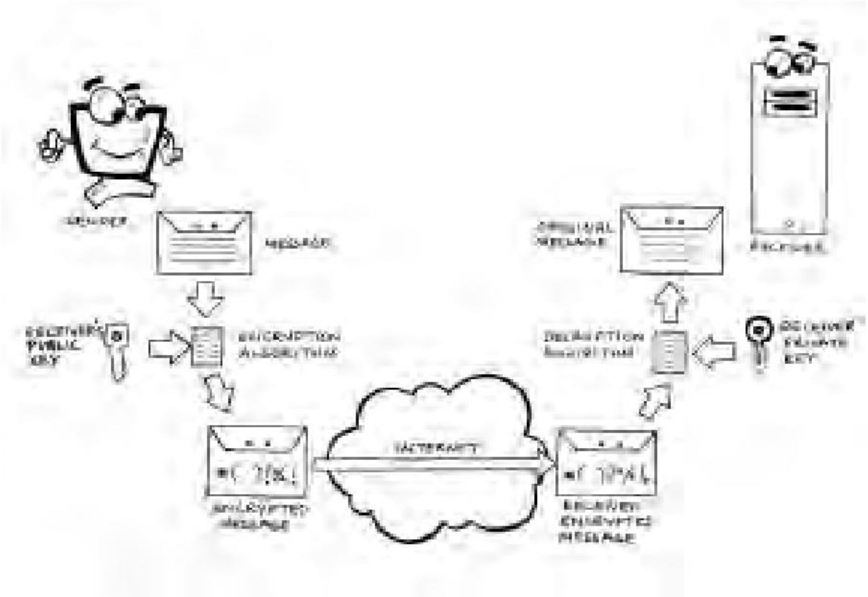


Fig. 13.2 *Encryption with a public key and decryption with the corresponding private key*

The message being transmitted is encrypted using the public key. *And it can only be decrypted using the corresponding private key.* This is vitally important, so I must repeat it, just to ensure that you are awake: *The message cannot be decrypted with the same public key.* It now needs the corresponding private key to decrypt it.

And then there was light !!!

Because all you need to do is to encrypt your message using the receiver's public key. In any case this is freely known to everyone. The receiver then uses his private key to decrypt the message. Since he is the only person who has access to his own private key, confidentiality is automatically ensured !

Eureka !!! Isn't that a simply wonderful solution? If two people need to exchange a message now, all that needs to be done, is that the receiver makes his public key available to the rest of the world—through any means. He could put it up on a public web site, or advertise in the newspapers, or even have it put up on huge hoardings in the middle of the city (of course with the mandatory, smiling, bikini-clad model, to ensure people look in that direction).

But wait ! He needs to do something else as well. He needs to keep his private key to himself. However, that is something we are used to, isn't it? It's like your e-mail password, or your bank account number, or your ATM PIN number, which you would guard with your life—or at least you should.

Incidentally the original process, where we used the same key both for encryption and decryption, is called **symmetric key cryptography**, since it uses the same key at either end.

SUMMARY

And that, dear reader, was your first foray into the subject of message security. Which is all about four different problems that we face when sending messages across the Internet. Namely :

(a) Confidentiality : No unauthorised person should read the message

(b) Integrity : The message should not be tampered with

(c) Authentication : You should know who you are dealing with

(d) Non-repudiation : You should not be able to deny sending a message that you did send

We saw that you needed something wonderful called encryption to ensure confidentiality—which of course means that you scramble the message to get something that (hopefully) only the intended receiver can unscramble. We called these two processes encryption and decryption, and understood the concept of a key—which both the sender and the receiver needed to have.

And then of course we saw that using one single key at both the sending and receiving ends caused a problem—because then you would have no way to send the key securely. So we discovered the even more wonderful public key encryption, where you actually work with key pairs. One key in this pair is the private key, which is known only to the owner of the key pair. And the other one is the public key, which is known to everyone !

A message encrypted with a public key can only be decrypted with the corresponding private key. And that, quite naturally, solves the problem of confidentiality. All you need to do now, is to encrypt a message with the receiver's public key, and he decrypts it with his private key—no one else can ! Unless of course he has been stupid enough to put up his private key on the notice board—but in such cases I firmly believe a person deserves the problems he's got.

Does this change our e-Commerce architecture?

Actually no. It just means that messages which pass between different hosts on the Internet could be encrypted (Fig. 13.3).

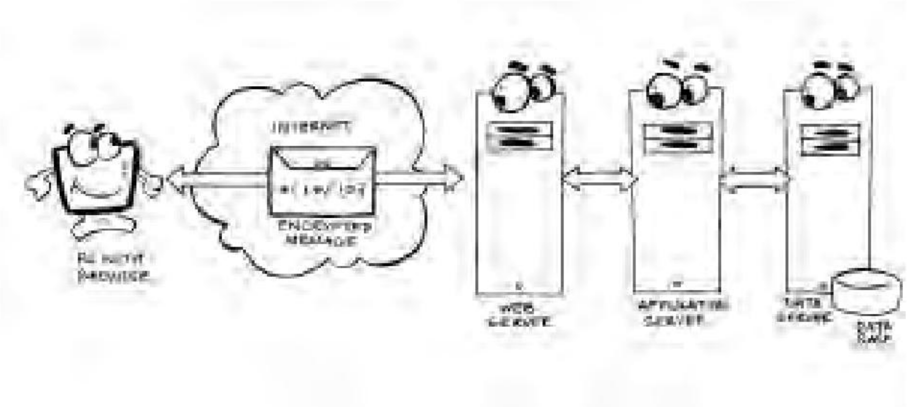


Fig.13.3 *e-Commerce architecture, with messages encrypted*

Oh by the way, I almost forgot an interesting sidelight. Namely, public key cryptography works in reverse as well. In other words, if you encrypt a message with a private key, it can only be decrypted with the corresponding public key (Fig. 13.4) !!!

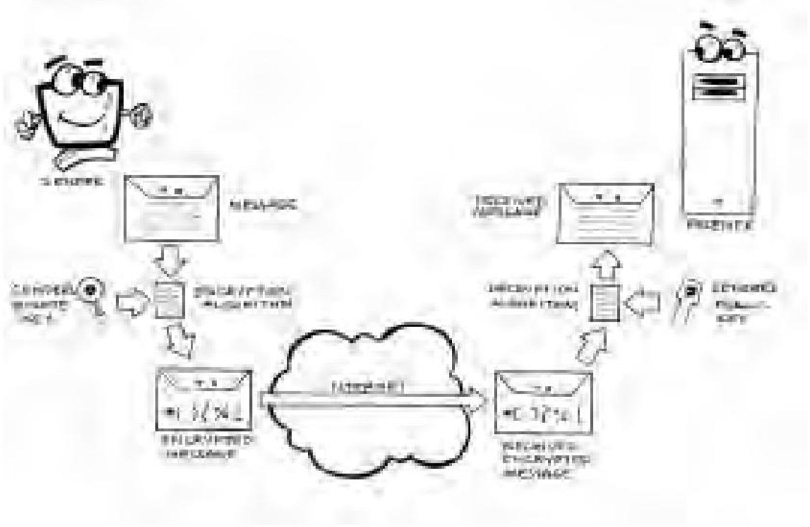


Fig. 13.4 *Encryption with a private key, and decryption with the corresponding public key*

Big deal ! What's the use of that? Everyone has access to the public key !!!

Strangely enough, this is as important to security as encryption using the public key. In fact, all the three problems that we haven't tackled so far, namely authentication, integrity, and non-repudiation, are based on this strange fact.

How?

My friend, haven't you racked your brains enough for one session? Wait till we meet again—in the next chapter of course !

14

The Wonderful World of Digital Signatures

Have you ever been to a party where one of the guests was highly opinionated on just about everything under the sun—from George W. Bush, to Tutankhamon, to the merits of drip irrigation, to the state of the Indian economy? And all the ladies in the party were surrounding him, with poor old you nursing a quiet, miserable beer in a corner, wondering whether Tutankhamon was the latest Ayurvedic wonder drug, or simply a hybrid variety of cabbage............

Well cheer up! Now there is hope for you and many, many others like you. You, my friend, are going to learn all about digital signatures and certificates. And I can guarantee that at the next party you will have all the ladies drooling over every word of yours.

To start with therefore, let's take off from where we stopped in the previous chapter, and examine the problem of integrity.

THE INTEGRITY PROBLEM

As described in the previous chapter, integrity of a message means that no one has tampered with it. In other words, the message you receive is actually the message that was sent. So in the example in the previous chapter, if your girl friend says she wants to marry you, she does...................... it is not a desperate attempt by her ex-boy friend to get her into trouble.

How is integrity ensured in the physical world? Very simply, by using a signature. For instance, if you are sending a letter to someone (and let us for the moment assume that it's a typed letter), it is very easy to remove the letter and replace it with another one. Unless of course it is signed by the sender.[1]

The key issue here is that the receiver of the message has some way of recognising the signature. In other words, if you get a message from Sherlock Holmes, and it is signed by him, you know that the signature actually belongs to Mr. Holmes and not to Professor Moriarty! You may not have met Sherlock Holmes in person (neither have I, for that matter), but you have some way of figuring out that it is, indeed, his signature.

Wonderful! Now let us look at the digital equivalent of this signature. Which, for some strange reason, we will call a **digital signature**.

This digital signature, whatever shape it takes, needs to satisfy one simple criterion. Namely, only the true sender of the message can use it to sign the message.

Let us now apply a bit of magic to our original romantic message—my favourite:

"Darling, I love you, and will do anything to marry you"

We ask your girl friend to create a synopsis of this message by taking every third letter. And for this purpose, we will assume things like a blank between two words, and even commas, are all letters. That gives us the equally romantic message,

"DlgIo ua ldatntmry"

(This of course appears as though your girl friend has a cold, but of course you and I know better). Now you would notice that we have chopped up the message mercilessly. Computer Scientists, who are known for their ingenuity, have compared this process to that of preparing hash brown potatoes by chopping and mashing them. We therefore say that we have applied a **hash function** to the original message sent by your girl friend.

Further, since the result of applying a hash function is a synopsis of the original message, we will call it a **message digest**.

And now for the interesting part: Ask your girl friend to send you both the original message and also the message digest (Fig. 14.1).

Great ! So you receive these two at your end. What next?

Simple. Just take the original message received and apply to it the same hash function that she had applied. Therefore you generate a message digest of the message you have

[1] Here it is assumed that physical signatures cannot be forged. In actual practice they can, but the probability of that happening is low, and therefore the world assumes that when it sees a signature belonging to someone, it does actually belong to that individual.

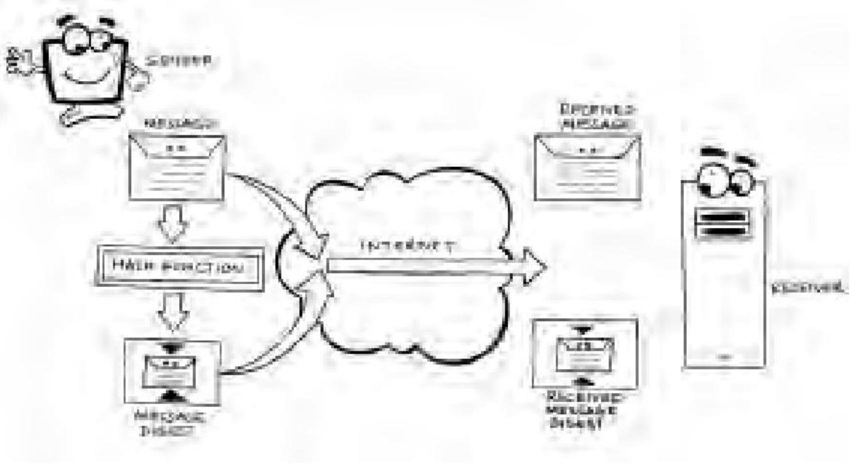


Fig. 14.1 *Creating and sending the message along with the message digest*

received (Fig. 14.2). Again remember that these hash functions, like most other kitchen recipes, are common knowledge, and therefore publicly known (unlike private keys).

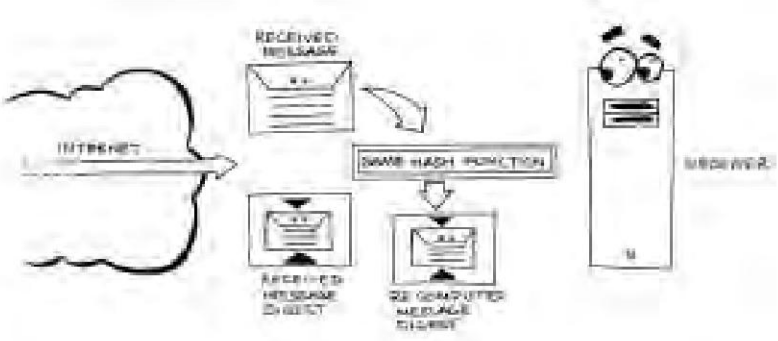


Fig. 14.2 *Applying the same hash function to the original message at the receiving end, to get a second message digest*

Now there is only one thing left to do. And if you are still awake, that one thing should be obvious—compare the two message digests with you. One digest has been sent by your girl friend. And the other one has been computed at your end by applying the hash function to her message (Fig. 14.3).

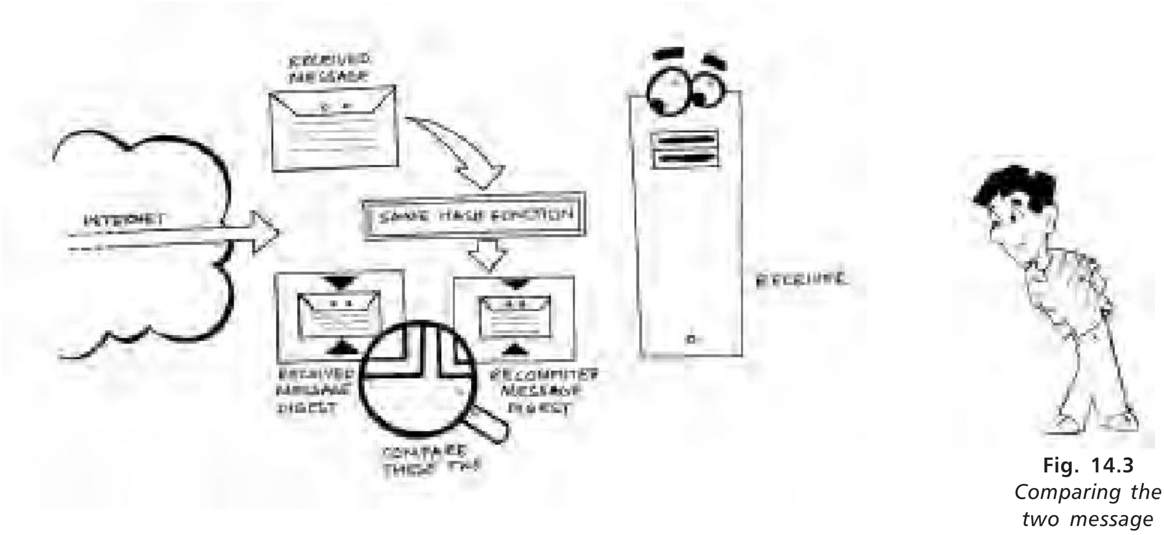


Fig. 14.3 *Comparing the two message digests*

Isn't it clear now?

No?

Then I would suggest you close this book, get yourself a cup of strong black coffee, and think!!!

Think, my friend. You have just obtained two message digests by two different methods. Now what if a hacker had tampered with the original message during transmission? Naturally, the message would have been changed, and therefore, the message digest you generate at your end is different from the digest your girl friend had sent you!

So when you compare the two message digests at the end of the process, they would not tally.

Eureka!!!

All you therefore need to do, is to compare the two message digests. If they are the same, your message has come through unscathed. And if they don't tally, there is an integrity problem.

Simple, isn't it?

THE DIGITAL SIGNATURE

But wait! Our friend, the hacker, has seen through this. And he has worked out a perfect method of beating the system.

What he does is to change the original message and then apply the same hash function to this new (changed) message, thereby getting a new message digest.

Now all he has to do is to remove the original message digest and replace it with the new one. *In other words, he sends you a changed message and also an appropriately changed message digest*!

When you get the message, you will apply the hash function and get the same message digest that the hacker has generated! So it tallies with what he has sent!

Trouble, isn't it?

What's the solution?

Try this: After your girl friend creates a message digest, ask her to *encrypt this message digest using her private key, before sending it to you* (this is vital—the magic works only if it is her private key). Figure 14.4 depicts this process.

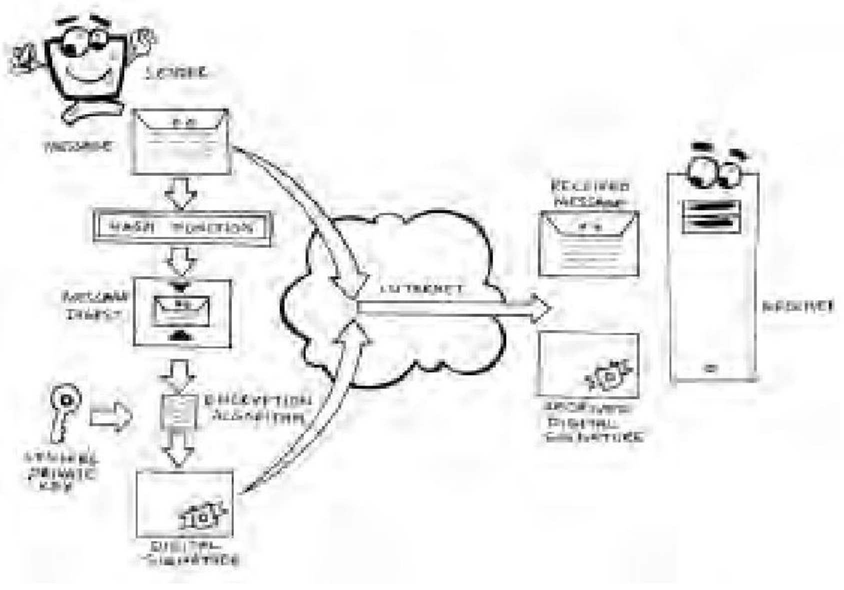


Fig. 14.4 *Sending the original message, and an encrypted message digest*

And that, ladies and gentlemen, is her **digital signature**. Which she sends you along with the original message as before.

What do you do after receiving these two?

Simple. We ask you to *decrypt the signature using your girl friend's public key* (again this is critical—it must be her public key). This is shown in Fig. 14.5.

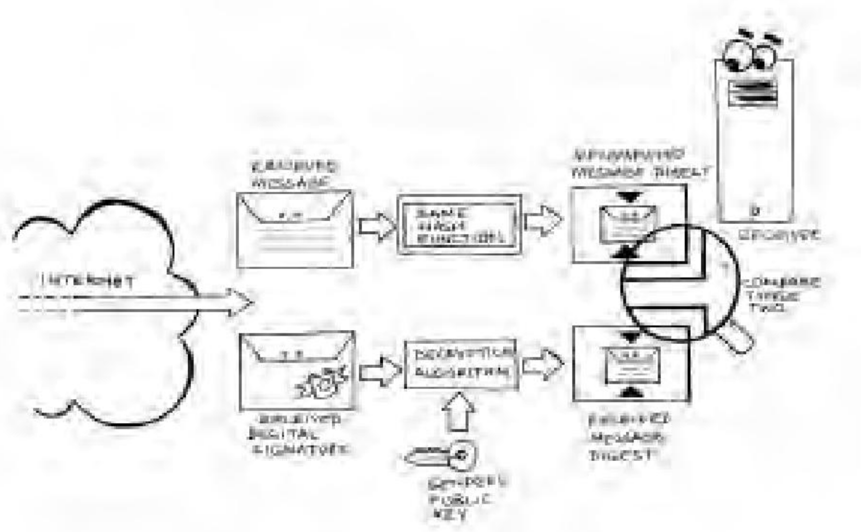


Fig. 14.5
At the receiving end, creating a message digest as before, but now comparing it with the decrypted digital signature

And as before, you pick up the transmitted message and apply the hash function to it, to get a second message digest. Once again, compare the two message digests. Now let us assume the hacker has done his job and tampered with the message. He has also created a new message digest for the modified message. But here's the catch. He does not have your girl friend's private key, and therefore cannot create her digital signature. *Any signature applied by him would be using his own private key and not your girl friend's private key.* Therefore this signature cannot get decrypted using her public key.

You get the message? Your ability to successfully decrypt the message using someone's public key means that *only he (or she) could have signed the message. So no tampering was done at any stage.*

And when you decrypt the fraudulent message digest, what you get will not tally with the message digest you have computed at your end.

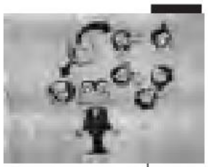
But if there is no tampering, the two digests tally. Which means you've got the right message.

Eureka again! And this time, it is genuine!!!

Question: Why do we use a message digest and not the entire message? Well, the answer should be obvious—the digest is much shorter than the original message, and therefore in doing so we conserve valuable bandwidth (and of course time).

One more comment before we close the subject of digital signatures. In the preceding discussion, we have taken care of the problem of integrity but not confidentiality. But that problem is easily rectified. Simply ask your girl friend to encrypt the complete message (her original message along with the digital signature) with your public key, as in the previous chapter. Now only you can decrypt the message, because only you have your own private key. (Of course if you have lost it or shared it with someone, you are in trouble, but then you deserve to be. It is like sharing the secret PIN number of your ATM card—can't be done!!!)

Summary



So now you are the proud possessor of a unique piece of know-how. You know how a digital signature helps us solve the problem of integrity.

To ensure integrity of a message, the sender needs to apply a hash function to the message, so as to get a message digest. Next, he encrypts this message digest with his private key, and then sends it along with the original message.

At the receiving end, you do two things : First, you apply the same hash function to the original message to get a new message digest. And then of course, you decrypt the digital signature with the sender's public key. That gives you two message digests, both arrived at by different means. If the two match, you're in business—meaning that the message has reached intact, without any tampering on the way. But if they don't, well, there's something fishy going on.

And incidentally, this doesn't really change our e-Commerce architecture. It simply means that messages passing between hosts could have digital signatures along with them (see Fig. 14.6).

So that's the glorious world of digital signatures.

Wonderful aren't they? After all, they help you to keep your love life intact.

But your love life needs something more—it needs authentication. Has the message truly come from your girl friend, or is it your wife who is spoofing? And authentication requires the other biggie, namely digital certificates.

What are digital certificates?

Good question. For the answer, just turn the page and get to the next chapter. But—and this is a sincere suggestion—do it tomorrow. Right now, go and take a nice, lo-o-o-o-o-o-o-ng, walk. Enough technology for today.

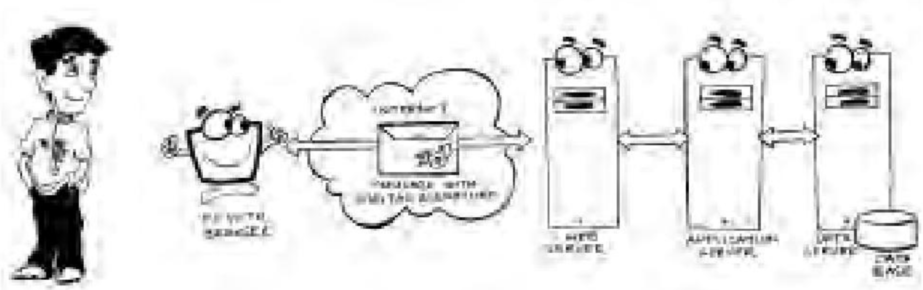


Fig. 14.6
e-Commerce architecture with digital signatures

15

The Final Frontier–Digital Certificates and More

Sounds terrific doesn't it: digital certificates! Just roll them over and over your tongue and get used to them. You can't escape them. Because digital certificates (or simply certificates) are a critical component of any security solution on the Internet.

So put on your thinking cap all over again, and jump into this chapter. We're going to look at certificates......................

AUTHENTICATION AND DIGITAL CERTIFICATES

To understand what a digital certificate is, let us go back to the third problem we had described in Chapter 13, namely that of authentication. How do we know that the site we have reached, for instance sabziwala.com (run by our good friend Haribhai from Chapter 6) is truly sabziwala.com? It could be an imposter who is simply out to collect your personal data, which he then sells to all and sundry. So you are flooded with e-mails asking you to buy tummy flattening equipment, or inflatable sofas, or simply pirated music CDs. Worse, it could be Haribhai's nephew who is just dying to get his hands on your credit card details, so that he can upgrade the venue of his "dates" to a five star hotel.

As you can see, in the cyber world, just as in the real world, it is vital to identify who you are dealing with. In fact it is even more important than it is in the real world, where you can see and talk to him. Here you can only see his site!

Let us therefore solve the problem of authentication. And as usual, we'll tackle one thing at a time. So we will first try and solve the authentication problem alone, without worrying about confidentiality and integrity. As you have seen in the previous chapters, these can always be added on.

How is authentication done in the real world? Usually through some document, isn't it? For example, when you travel from one country to another, you prove your identity through your passport, which has your name and photograph, along with several other details such as your date and place of birth, father's name, permanent address, etc. etc.

And above all, your passport is signed by the passport officer in the Ministry of External Affairs.

This last bit is critical: Someone (in this case the passport officer) has verified the details that you have given, and this "someone" has signed on your passport. Notice that this "someone" meets two critical conditions :

(a) He is trusted—both by you and by the visa officers in the countries you are planning to visit.

(b) He is also neutral—he is not personally interested in you or in your trip, or for that matter in the country you are about to visit, and can therefore be expected to be unbiased (unless he happens to be your uncle, or even your wife's first cousin's son in-law—but we will ignore that possibility for the moment).

A very logical term for this person is therefore, **trusted third party**.

And now we get back to cyber space. Just as we had a passport in the physical world, we need some document in the cyber world. And this document is what we have been hearing about all this while—the **digital certificate**, or sometimes simply a **certificate**.

Every site such as sabziwala.com or microsoft.com, or rediff.com would get a digital certificate. The certificate would need to be obtained from a trusted third party, and would contain details such as :

(a) Name and address of the organisation that owns the site
(b) Domain name of the site
(c) Period of validity, etc.

And finally, the certificate would be signed by the trusted third party (sometimes called **TTP**).

How?

By using his digital signature of course!

So when you go to sabziwala.com, you would see an icon on the home page, which represents the digital certificate. Clicking on this icon would take you to the certificate, where you can actually see all the details listed earlier. Importantly, since the certificate is signed by the trusted third party, you know that it has not been tampered with.

Is that all?

How disappointing!

Fortunately that is not all. There is one thing that we have left out. And that one thing is crucial.

Let's step back for a moment. We said that we needed to authenticate the site, so we know we are sending our message to the correct site, and not to an imposter. However, having identified the site, we also want to ensure that *this site and only this site* can read our message—the problem of confidentiality described in Chapter 13.

If you recall, confidentiality requires that you encrypt your message with the public key of the receiver—in this case, the public key of the site you wish to access. The key question is, *how do you ensure that you encrypt the message with the public key of this site and no other?*

There is one way. And it's a ridiculously simple way. All we need to do is to ensure that the digital certificate contains the public key of the site as well. And since it is digitally signed by the TTP, you know that no one has tampered with it (remember integrity from Chapter 14?). So the public key that you receive is actually the public key of the site you are accessing. See Fig. 15.1 to understand what a digital certificate looks like.

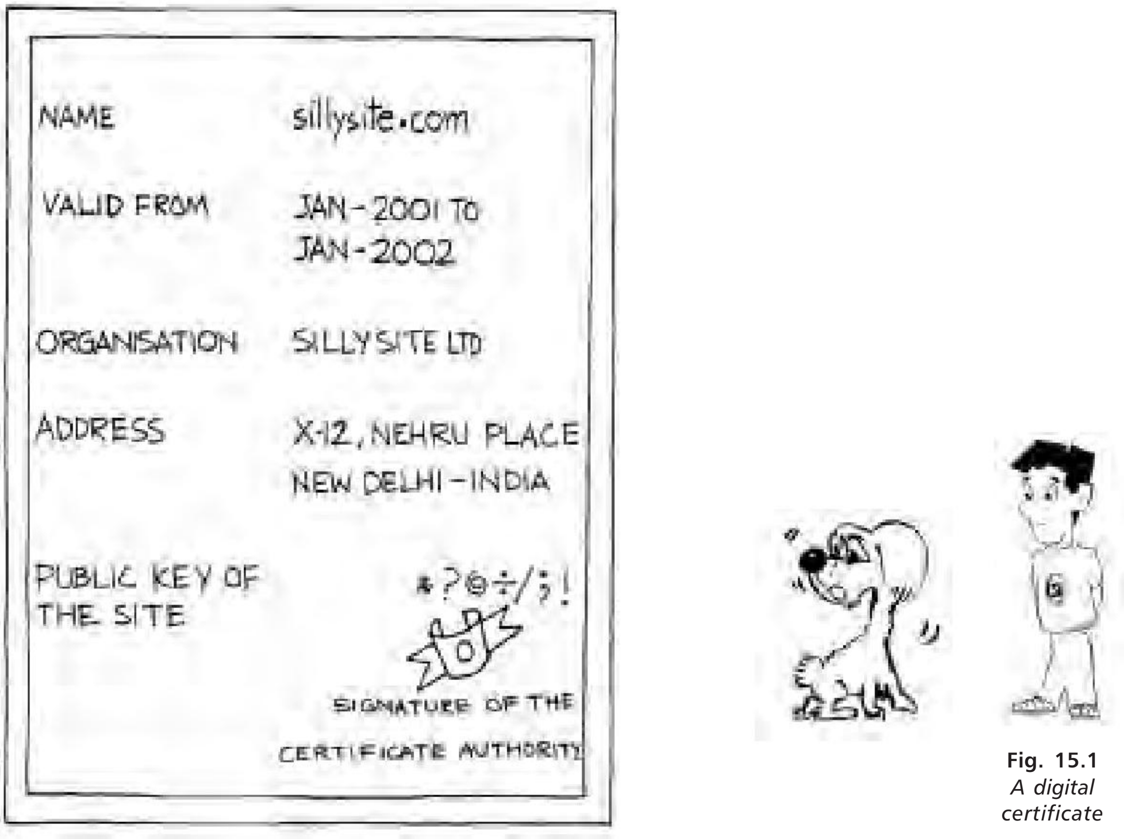
Next question: Do only servers have digital certificates? What if the site needs to authenticate you before doing business with you?

Sure, you also have client side certificates, so that two way authentication is possible. But to avoid giving you a headache, I will not get into details here.

Incidentally, the trusted third party is usually an organisation. And since this organisation issues the certificate, it is often referred to as the **certificate authority** or **CA**. And yes, before I forget, the CA is the one who issues the key pair to the site (it is also possible for you to generate the key pair using special programs, and then have the public key sent to the CA so that he can include it in your certificate). All for a fee of course. In fact for an annual fee. After all, CAs don't live on charity !!!

Verisign and GTE are two of the most popular CAs world-wide. Naturally, they are recognised and trusted across the globe.

But you may also have CAs who are specific to a country. These CAs are recognised and trusted as TTPs within their country of operation, but may not even be known outside.



NAME	sillysite.com
VALID FROM	JAN-2001 TO JAN-2002
ORGANISATION	SILLYSITE LTD
ADDRESS	X-12, NEHRU PLACE NEW DELHI-INDIA
PUBLIC KEY OF THE SITE	*?©÷/;!

SIGNATURE OF THE
CERTIFICATE AUTHORITY

Fig. 15.1 *A digital certificate*

Either way, the CA is a crucial player in e-Commerce. Because when you are dealing with an unknown web site, the CA is the only one who can tell you whether it is genuine.

NON–REPUDIATION

And that, ladies and gentlemen, brings us to the most complex problem of all—that of non-repudiation. If you recall from chapter 13, how does your share broker, who buys Hindustan Lever shares for you, ensure that you do not go back on your word when the share price suddenly crashes?

Let's pause for a minute. What does non-repudiation require?

It actually requires the following :

(a) The message itself
(b) Proof that the message came from you—or your certificate
(c) Proof that the message has not been tampered with—or your digital signature

All your broker needs to do is to store these, and his problem is solved. He now has proof that this message came from you.

And of course, when the broker confirms that he has bought your shares, he sends you exactly the same combination, so that he is not able to go back on his word either !

That's it ! Non-repudiation ! Making sure that each party to a transaction has proof of what the other party said.

Vital for any e-Commerce transaction, isn't it?

SECURE SOCKETS LAYER (SSL)

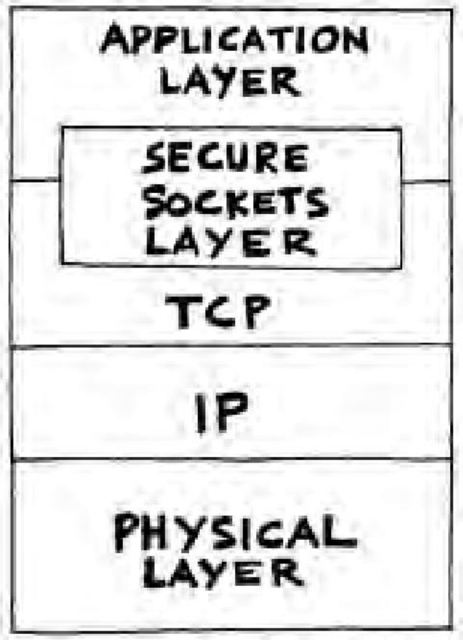
Let me now take you all the way back to chapter 1, where we had introduced the term protocol. We had defined the term as an agreed to arrangement for two computers to exchange information. Based on what we have just learnt about security, we therefore need a protocol which permits encryption, digital signatures, and certificates, to be used and exchanged between hosts on the Net.

The most popular protocol of this kind for the Internet, was developed by Netscape. It is called the **Secure Sockets Layer** (or **SSL** for short). SSL takes care of all the issues we have raised, namely confidentiality, integrity, and authentication. It insists on a certificate for the server, but does not insist on a certificate with the user or client.

Aha ! One more protocol !!

But we saw in Chapter 3 that the TCP/IP suite of protocols is a 4-layered protocol? So where does this new one fit in?

Interestingly, it fits in very neatly. It actually sits between the TCP layer and the application layer, or the HTTP layer (see Fig. 15.2). Incidentally, SSL is somewhat more involved than our simple description indicates, but that's okay for you and me.



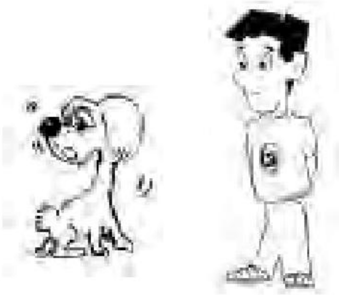


Fig. 15.2 *The TCP IP suite with SSL*

Question: How do you distinguish between a site that supports the SSL protocol and one that doesn't?

Answer: Easy—It's all in the URL. For instance if the site sabziwala.com were to support the SSL protocol for security, you would type in the URL :

“https://sabziwala.com”

instead of the conventional http://sabziwala.com. In other words, “http” gets replaced by “https”.

And incidentally, when you are connected to such a site, the browser will display a small lock at the bottom—just to indicate that you are on a secure site !

Before we move on, you may have noticed that SSL takes care of confidentiality, integrity, and authentication. But hang on ! What about the fourth problem, namely non-repudiation?

You are right—it doesn't take care of non-repudiation.

So is there a protocol that does?

Acutally there is, but we will need to understand the subject of payment first. And therefore we will meet this new protocol in Chapter 16.

40 BIT AND 128 BIT KEYS

I am sure you're dying to ask a very key question—how difficult is it for an intruder to hack into an encrypted message?

To answer this question, we must first understand what the term "hack" means. Essentially, it is the process of discovering the private key, so that the message can be decrypted.

And now let us take an analogy again.

I assume all of you are familiar with the concept of a combination lock. What happens in a combination lock? Essentially, you have only one combination of digits which will open the lock, whereas all other combinations will fail. Clearly the correct combination of digits is the key.

Now assume you had a combination lock with only one digit. A potential thief (or hacker) would only need to try out 10 different digits to hit upon the correct key. But if the lock had two digits, he would need to try all possible combinations, which in this case works out to 100. Of course if you had three digits, which is what most combination locks do have, the thief needs to try out all possible combinations of three digits, which in this case is 1000. And by the time he has tried out a fraction of these, you have probably come back from the toilet or wherever it is you had left your suitcase and gone !

Get the message? The longer the key, the tougher it is to crack, and therefore the more time it takes. Not only that, as you add one digit to the key, the time to crack it goes up by a factor of 10 (something that mathematicians call an 'exponential increase').

In the computer industry, as you know, everything works in bits. And therefore we have key lengths for encryption measured in bits. The two most commonly used standards are 40 bit and 128 bit keys. 128 bit encryption is obviously much safer, but then it also takes much more time to encrypt and decrypt, as well as to transmit.

Like everything else in life, you don't get something for nothing.

WHERE SHOULD WE USE SECURITY? AND WHERE SHOULDN'T WE?

We have been discussing threadbare, the whole issue of security, of digital certificates, of non-repudiation, etc. etc. etc. for some four chapters now. And it does seem as though encryption, digital signatures, certificates, and what not, are the solution to all problems in life, doesn't it?

But is there a flip side? Do we lose something when we go in for all these fancy security mechanisms?

Let us take an analogy and see. Suppose you had to purchase a magazine. So you go to one of the pavement shops in Connaught Place (if you are in Delhi), or in Churchgate (in Mumbai), or Chowringhee (in Kolkata) or anywhere else for that matter, and you pick up your copy.

Now suddenly the shopkeeper demands your driving licence !

Naturally, you are horrified. "Why?" you ask, or more likely, shout.

"I need your identification. You may be using forged currency notes !"

So you pull out your driving licence and show it to him.

And what does he do with it? He turns to a photocopier kept behind him, and takes a copy. And then of course, asks a Gazetted Officer to "attest" the copy he's just taken.

Why? Because of non-repudiation. If he discovers later, that the fifty rupee note you gave him was forged, you shouldn't go back on your word—hence the photocopy, duly attested by the Gazetted Officer!

And at end of all this (which by the way, has taken about 10 minutes), you are the proud owner of a 20 rupee copy of the magazine.

Does this make sense?

Of course not. For such a small, insignificant transaction, it only makes the process highly cumbersome, irritating (you are not likely to come back to this pavement tomorrow), and above all, very, very slow.

In exactly the same manner, encryption, hash function computation, digital signatures, etc. tend to slow down the process of e-Commerce and make it very cumbersome. Because of which, it is sometimes preferable to take a risk (if the risk is small) and avoid some or all of these security mechanisms.

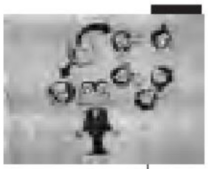
For instance, many web sites are hosted on two distinct servers, one which supports security and one which doesn't. And the site gives you the option to choose either of them. If you choose the secure site, the transaction is likely to be significantly slower than the non-secure site.

How do you choose the secure or the non-secure site? Well, if it is SSL security, you have just seen how to do it in the preceding section—if the site sabziwala.com had both a secure server and a non-secure server option, you would type in the URL "https://sabziwala.com" for the secure server, and for the non-secure site it would be "http://sabziwala.com". The only difference is in the "https" as against the "http" protocol.

In most cases, you would probably go to the non-secure site first, do your browsing, select what you want to buy and put it in the shopping cart, and only when you are about to make the payment would you choose between these two options. In which case, most sites would give you two hyperlinks—one for each option. But of course if you are truly paranoid, and do not want any snooper to see what you are even browsing, then my friend you need SSL right from the word go.

You also perhaps need a psychiatrist !!!

Summary



And that's it ! The digital certificate ! To identify yourself in the real world, you would need a document such as a passport or driving licence. Similarly, in the cyber world you need a digital certificate, or simply a certificate. You get these from a trusted third party—also called a CA or a certificate authority. This certifies that you are who you claim to be. And anyone visiting your site simply needs to click on this certificate to be sure that he's dealing with the right person.

And then we have non-repudiation, which really requires three things, namely :

(a) The message itself

(b) Proof that the message came from you, or your certificate

(c) Proof that the message has not been tampered with, or your digital signature

SSL is one popular protocol which takes care of confidentiality, integrity, and also authentication. But not non-repudiation.

What this does to our e-Commerce architecture is that it permits hosts on the Internet to use certificates to identify themselves. And it also potentially permits SSL as a protocol for sending messages across the Internet (Fig. 15.3).

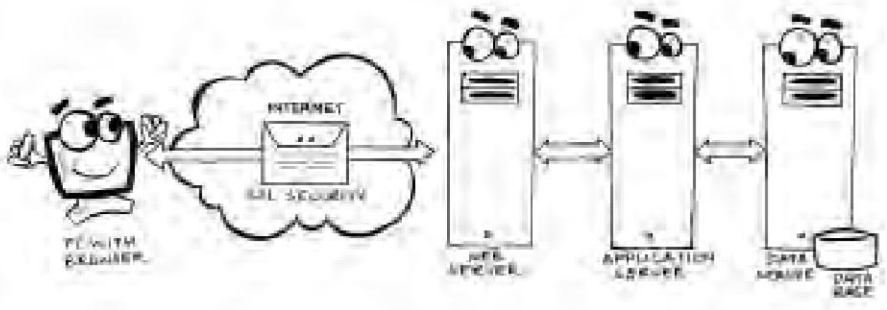


Fig. 15.3 *e-Commerce architecture with SSL security*

And with that, at the end of four gruelling chapters, we have finally come to the end of our most critical subject—Internet security !

Security solutions are available for most problems. But don't jump into them. Take what you need, but don't overdo it.

Remember, the more the security, the slower the system !!!

16

Money, Money, Money—Payment on the Net

Finally, after all these pages and pages, we have come to something close to everyone's heart—money !

Whatever you may do in e-Commerce, ultimately you will either need to pay or to receive money. This could be tiny amounts of perhaps a few rupees, or it could be huge, say, thousands or even millions of dollars.

So fasten your seat belts, and let's get into the world of MONEY !

CREDIT CARDS IN THE PHYSICAL WORLD

As a consumer, if you get onto a shopping site on the Internet, the chances are that you would pay by credit card. So let us first understand how such a credit card transaction takes place.

We will start by taking a look at how credit cards are processed in the physical world (the world of shops and great salesgirls and of course traffic jams). Then we'll move this to the cyber world and see the difference. For simplicity, we will first look at the bare essentials of the process, and subsequently add in complexities.

What happens when you buy, say, a camera from a shop (also called merchant), and use your credit card to pay?

The first thing is that the salesgirl (or salesman—we can't always be lucky can we?) takes your credit card and swipes it in a strange looking device. Since this device is used in the shop, or in other words at the point of actual sale, we call it a **point-of-sale terminal**. Fig. 16.1 shows you the full cycle of what happens in such a transaction.

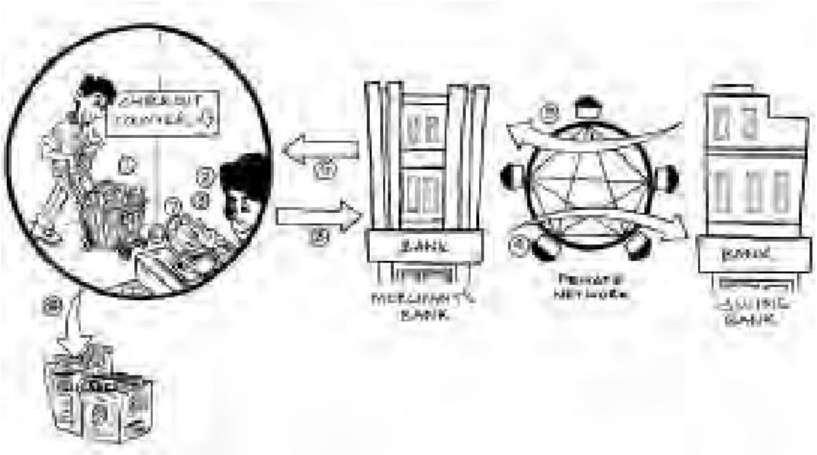


Fig. 16.1 *Buying goods with a credit card*

Why is the salesgirl swiping your card? Because it contains magnetically encoded information about your credit card account. Armed with this information, the point-of-sale terminal phones up the **merchant's bank** (the bank that the shop keeps its accounts with) to get authorisation for the purchase. Incidentally, the merchant's bank is often called an **acquiring bank**, because it "acquires" the credit card transaction from the merchant.

Here we obviously have a problem. Because you as the card holder have got your card issued from some other bank (we will call this the **issuing bank**), which is different from the merchant's bank. And the issuing bank is the one that maintains your account—what your credit limit is, how much you have already spent, etc. So how can the merchant's bank possibly authorise the transaction? It is not even aware of your existence (unless of course you happen to be the current Miss Universe or the captain of the Indian cricket team or something similar).

Which means that the merchant's bank must somehow contact the issuing bank and get authorisation from them.

How is this done?

Well, if credit cards had been invented during the time of Archimedes, it is likely that they would have used a highly reliable carrier pigeon service. But in today's modern times, unfortunately, we do not have the luxury of waiting for the week or two that the carrier pigeon would take to go from one bank to the other and return. After all, you are waiting at the shop, camera in hand, and a couple of weeks may not be acceptable (unless of course you have well and truly fallen for the salesgirl). Therefore, we are forced to rely on something the modern world has given us—a computer network.

In other words, we need a computer network that links up the merchant's bank with the issuing bank.

But are these the only two banks that are linked up through this network?

The merchant next door (where you buy bullets for your revolver—remember, we are living in modern times) may be banking with a third bank, the restaurant where you take the salesgirl (from the first merchant) for a date, may bank with a fourth bank,....................

Further, your uncle may have got his credit card issued from a completely different bank, his aunt may have got hers from yet another bank, and so on....................

Get the idea? In the good old days, you had only one bank—your own pillow—and perhaps your mattress for variety. Or even a hole dug in your garden. But in today's world we have banks and banks and more banks. So you need a large, complex network, that spans all banks.

Could the Internet be used for this purpose?

Actually, there are a couple of issues here.

First of all, you must know that credit cards have been around for many, many years. And the Internet has become freely available only in the past few years.

Secondly, the great thing about banks is that they are paranoid about security. Naturally—they have all your millions at stake ! So banks tend to be fairly conservative and risk-averse, when it comes to adopting new technologies.

Now if the Internet were to be used as the network that connects all banks and all branches together, can you see the security risk? Every user on the Internet (and there are hundreds of millions of them) has access to all these banks. And you can just see all those hackers drooling over the opportunity.

So friends, the network that connects banks is not the Internet, and is not likely to be the Internet. It is actually a private network provided by VISA and MasterCard (see Fig. 16.2). Which all those wonderful hackers have no access to !

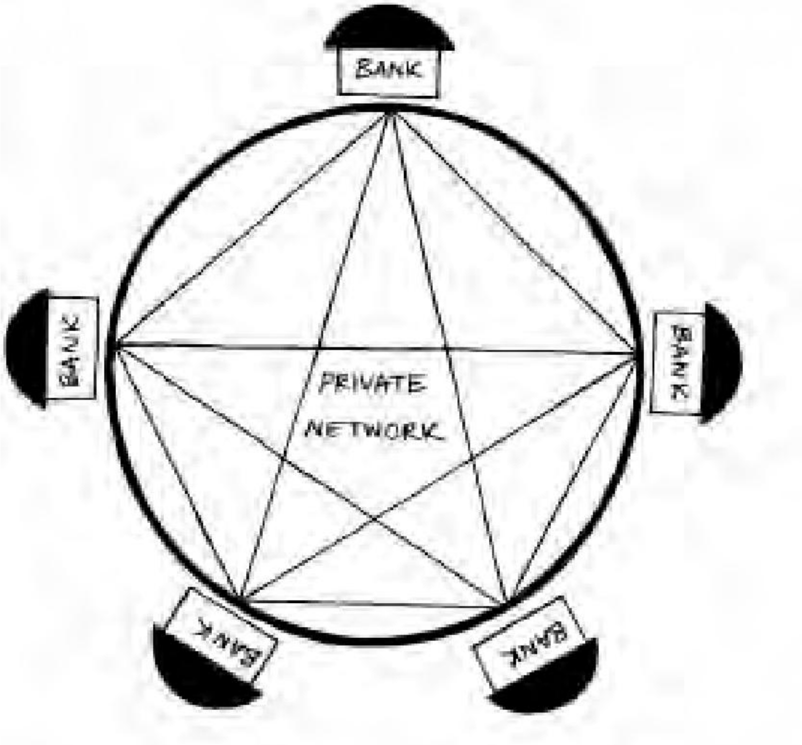


Fig. 16.2 *Private network connecting banks*

Now let's get back to our card that has been swiped. The information on your card, as well as the transaction amount, goes through this network and ultimately reaches your issuing bank. If your credit limit with the bank has not been crossed, the bank authorises the transaction, and therefore promises to pay the merchant subsequently. The authorisation comes back through the same network, and ultimately gets printed out as a little yellow slip on the same point-of-sale terminal (sometimes it's blue and sometimes it's pink, but believe me, it makes no difference at all to e-Commerce.)

If on the other hand you have already exceeded your credit limit with the bank, the bank turns down the request, so you don't get the camera. (We also have the last possibility where the bank says, "Aha. Caught you. The handcuffs are on their way !" This possibility of course arises if you steadfastly refuse to pay your past bills, or if the card has been reported stolen, but fortunately it is not too common).

All this happens automatically, without any manual intervention (except of course the handcuffs—at the time of writing this book there is no known way to get handcuffs coming out of the point-of-sale terminal and grabbing you by the wrist, but it may be an interesting area for research).

Whether the transaction is approved or denied, therefore, the bank sends back an appropriate message. Assuming it has been approved, you are now asked to sign on the yellow slip, and the salesgirl compares your signature with the signature on the credit card—unless of course she finds your face more interesting.[1]

And so you get your camera. What the shop has got is proof of the transaction having been authorised by the issuing bank, and proof that you have presented the card and agreed to pay.

Obviously the merchant needs to get his money as well, and you need to finally pay up but we'll see how that happens in a later section.

Let us summarise these steps in a table. The left hand column of Table 16.1 lists these steps for physical credit card transactions. Of course you can cheat and look at the right hand column as well, but since it would only confuse you at this stage, I suggest you follow my advice.

CREDIT CARDS IN THE CYBER WORLD

Now let us get into the cyber world ! Isn't it logical that most of the steps in the physical world, also need to be performed in the cyber world? For instance, details of the transaction need to be entered in both cases. Credit card details are also required in both cases. Your bank needs to check and make sure your credit limit has not been exceeded, etc. etc. etc.

In fact, much of the process followed in the cyber world and the physical world is common. Figure 16.3 shows you how the transaction proceeds in the cyber world, and the steps are listed in the right hand column of Table 16.1 (I must emphasise that this procedure is not standardised but varies depending on the country or the payment service you use. However, it is close enough to most commonly used procedures and is therefore good enough to understand the concept).

[1] In some older systems, particularly in remote towns with no access to the network, there is an alternative step. You would have noticed that your credit card has some of your details embossed on it, such as your name, card number, and expiry date. The salesgirl phones up the bank to get the transaction authorised. And then she takes an impression of this embossing on the invoice (or bill). She therefore has a physical copy of your credit card details, on which she take your signature.

Table 16.1 *The steps that a credit card transaction goes through*

Step	*Physical world*	*Cyber world*
1.	Items are put into the shopping cart	Items are put into the shopping cart by clicking on each icon, and then the check-out button is clicked.
2.	The salesgirl keys in the total amount into the point-of-sale terminal	The total amount is computed by the merchant's web server and displayed on your PC
3.	The credit card is presented to the salesgirl (and therefore the merchant), who swipes it into the point-of-sale terminal, thereby sending details to the merchant's bank	The credit card details are entered into the web page, from where they go to the web server,[2] and on to the merchant's bank, the last step through a payment gateway
4.	The merchant's bank sends details of the transaction and credit card to the issuing bank, through the private VISA/MasterCard network	The merchant's bank sends details of the transaction and credit card to the issuing bank, through the same private network
5.	The issuing bank authorises the transaction and informs the merchant's bank through the same private network	The issuing bank authorises the transaction and informs the merchant's bank through the same private network
6.	The merchant's bank informs the merchant through the point-of-sale terminal. A yellow slip is printed out for you	The merchant's bank informs the merchant's web server,[2] which sends a message to your browser. No slip is printed out.
7.	The salesgirl takes your signature on the yellow slip	No equivalent
8.	The items are packed and handed over to you	The items are packed and shipped to you

There are however, a few differences between credit cards in the real world and the cyber world.

[2] In simple sites, it would be the web server. In larger sites with a 4-tier architecture, this would be the web server along with the application server.

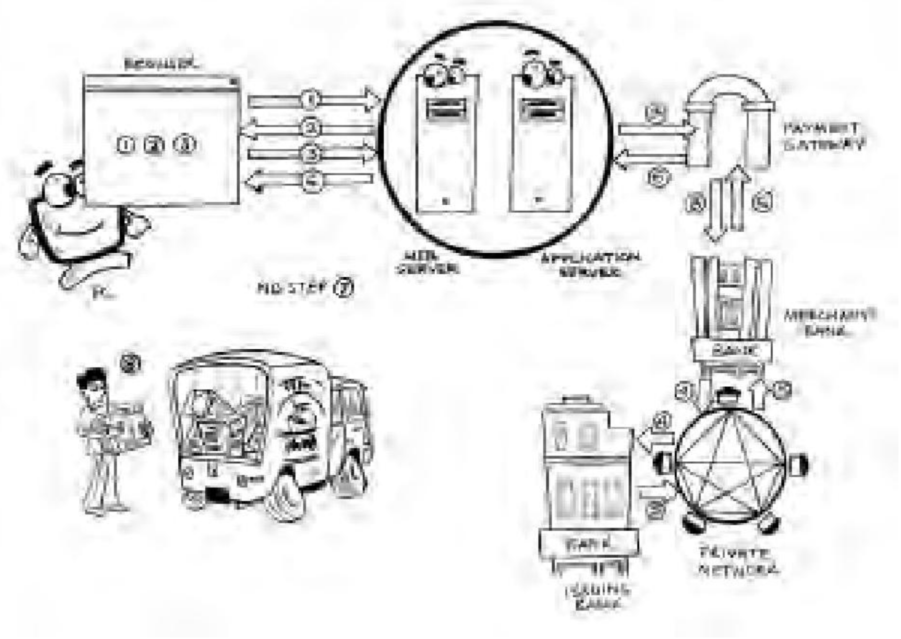


Fig. 16.3 *Credit card transaction in the cyber world*

First of all, in the cyber world you do not go to the shop (with the added disadvantage that you do not get to meet the salesgirl). You therefore do not have a physical point-of-sale terminal where your card can be swiped, but some kind of virtual point-of-sale terminal which only exists as an HTML page on your browser.

And this fact has major implications on security. Because there is now no way for the merchant to check the fact that you have a physical credit card with you. You may simply have picked up the card number from someone (in fact you may be a salesman with some other merchant, and your rather profitable hobby happens to be collecting credit card numbers along with names). And of course, the salesgirl cannot take your signature on the invoice or on the yellow slip!

Interestingly, the Western world has been using credit cards on the phone for several years. Now you can immediately see that giving a credit card number on the phone has exactly the same potential for fraud as doing it over the Internet. So the Net has not made things any worse.

And therefore countries like the USA accept a certain percentage of credit card frauds over the Internet. They bank on a strong legal system to track them down, wherever possible. For instance, if you were to suddenly get a bright idea and order an expensive book using someone else's credit card, you would have it delivered to your home, wouldn't you? So the delivery address is known—and you can still be tracked down ! It is not always easy and in many cases not possible, but the world lives with it.

However in India, banks have been more careful. Many banks that offer payment services on the Net ask you to key in some secret identification code as well—something like a PIN number. Someone else may pick up your credit card number, but hopefully he won't be able to get your PIN number !

The other difference between the real and the cyber worlds is that there is one additional element we have introduced in the cyber world—namely the **payment gateway**.

What's a payment gateway? In fact, what's a gateway?

Well, when you go from a busy road into a park (preferably along with your current flame) you pass through a gate, or a gateway. When you go into a historical monument, you go through a gateway. When you go to prison from the free world outside, you go through a huge, forbidding, and probably not very cheerful gateway (I guess so—I have no first-hand experience). In other words, **a gateway** is required where two different environments meet, to enable you to go from one of these environments to the other.

Importantly, the two environments can be very different from each other. For instance the environment inside a prison would have far higher security than the world outside (at least I hope so). Also, the protocols there would be very different—you would probably have to call all the officials "Sir" and all your colleagues in prison "Dada". Whereas in the outside world, you can call anyone by name without fear of getting beaten up.

Similarly, in the virtual world, a gateway connects two different kinds of networks. In this case, a payment gateway connects the highly secure private banking network with the far more public Internet. Therefore you would have far higher security behind the gateway than you would on the Internet side (by using things like firewalls). Plus, the protocols being followed on the Internet side may be completely different from those on the side of the banking network. The payment gateway therefore performs whatever is required to make this connection happen (such as security, protocol conversion, etc.). And as you might imagine, the gateway is a server with special software loaded on it to perform all these tasks.

Incidentally, payment gateways are usually put up by acquiring banks, and also by organisations that have specialised in payment systems, such as CyberCash.

WHAT HAPPENS BACK-END

So far so good. For you that is. Because you have got your transaction authorised and have also picked up whatever it is you had bought.

But what about the merchant? He needs to be paid, doesn't he? And if he is paid, where does the money come from? Ultimately from you, isn't it?

Therefore we have only seen one part of the story. Your part. We now need to see the other part where the merchant and the bank get paid.

As before, we'll first take a look at what happens in the physical world.

At regular intervals, say at the end of each day, the merchant sends the various slips he has accumulated during the day, to his bank (the merchant's bank). The merchant's bank in turn contacts the issuing bank, which transfers the amount to the merchant's bank. This amount then goes to the merchant's account. So that takes care of our friend the merchant.

But that has not yet taken care of the issuing bank ! This bank needs to recover the amount from you. So at the end of the month the issuing bank goes back to you as the credit card holder, and gives you a monthly statement. This statement lists all the purchases you have made during the month, and asks you to pay up. We hope and pray that you do actually pay up, otherwise what happens next is beyond the scope of this book (it would probably be within the scope of a book on criminology).

So the merchant gets paid, the issuing bank gets paid, and you have finally paid for your purchase. Figure 16.4 shows this process pictorially, and the left hand column of Table 16.2 lists the steps.

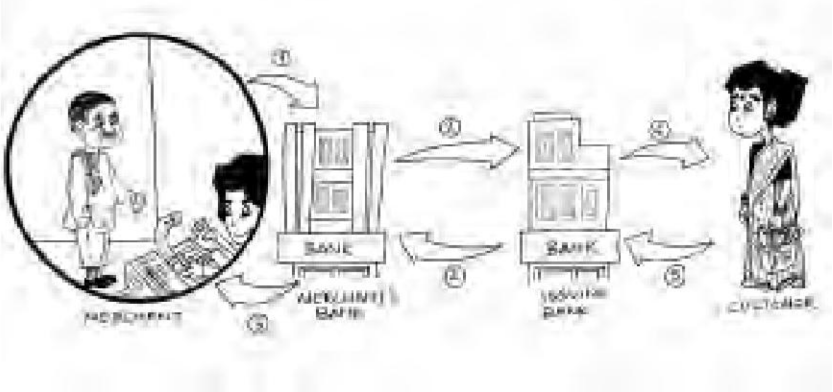


Fig. 16.4 *What happens back-end in a credit card transaction in the physical word*

Table 16.2 *The steps involved back-end, in a credit card transaction*

Step	*Physical world*	*Cyber world*
1	The merchant sends details of all the purchases to the merchant's bank	The merchant sends details of the purchases to the merchant's bank
2	The merchant's bank gets the payment from the issuing bank	The merchant's bank gets the payment from the issuing bank
3	The merchant's bank makes the payment to the merchant	The merchant's bank makes the payment to the merchant
4	The issuing bank gives you a list of transactions that you have performed during the month, and asks you for the payment	The issuing bank gives you a list of transactions that you have performed during the month, and asks you for the payment
5	You make the payment to the issuing bank, maybe by cheque	You make the payment to the issuing bank, maybe by cheque

What happens in the cyber world? Actually, it is almost identical. However, the difference is that in the physical world, the merchant sends to his bank paper copies of the transaction (the bill, the authorisation slip, and your signature). But these paper copies do not exist in the cyber world. Instead therefore, he sends soft copies of these, as well as the earlier authorisation he had obtained.

THE ELECTRONIC WALLET

There is still a problem with credit cards. Let's see what it is.

Suppose you were employed somewhere and your boss went on leave. What would you do?

Probably one of the first things you would do is to go to the Internet, and log on to all those shopping sites that you would never dare to, in his presence.

And on one of these sites, suppose you discovered the most charming pair of purple socks. Obviously you would be desperate to buy them. So the site would ask you to key in your name, how you want your socks shipped (by air, courier, special delivery, or simply delivered at your door-step along with the local brass band playing "For he's a jolly good fellow"), your shipping address, details of your credit card, and possibly several other personal details. In addition to, of course, clicking on the purple socks.

You have just finished keying in all this, when the connection breaks !

What happens now?

Nothing much—except that you key in all these details all over again.

Now you go to another site (remember, your boss is on leave and you have all the time in the world, sitting in office) and discover a terrific hat which makes you look the spitting image of Tutankhamon. Again, you must buy it. So once again you key in your name, how you want your hat shipped, the shipping address, credit card details,....................

Haven't we heard all this before?

Sure we have ! Every time you go to a new site, you would need to enter the same details all over again.

Any solution?

Yes, of course. There is always a solution !

Many vendors have provided a piece of software called a **wallet** (sometimes called an **electronic wallet** or simply e-wallet) that you load onto your PC. Into this wallet you enter your personal details, including your credit card details. And every time you get onto a site where a form needs to be filled up—Hey Presto—your wallet does it for you. No more monotonous keying in over and over again. It's rather like your real-world wallet, which contains your credit cards, your driving licence, and other particulars about you.

Wallets are quite common nowadays, and several organisations such as Microsoft and CyberCash provide them.

SECURITY ONCE AGAIN

What's the second most important thing you are worried about when dealing with money? (The first, obviously, is that you never seem to have enough of it—but that subject is beyond the scope of this book).

Security, of course !

So how do you ensure that your credit card details don't leak out? How do you prevent a hacker from breaking into a payment transaction and increasing the amount you are paying? Or taking it and using it for a purchase he is making? Or mixing up the message so that your payment doesn't reach—and neither do your wonderful purple socks?

One of the things that is used to ensure security is, of course, good old SSL (remember Chapter 15?). Because of the fact that SSL is simple to implement and use, it is among the most popular mechanisms in use for on-line payment.

However, SSL does have its problems. It ensures that the server has a digital certificate, and therefore authentication of the merchant is taken care of. However, it does not insist on client side certificates. Therefore there is no authentication of the client. Also, non-repudiation is not built into the SSL protocol.

The other issue is that SSL permits the merchant to see your credit card details.

Now wait a second.......................Is that really required? Does the merchant need to see details of your credit card? Why? Isn't it enough for him to see details of the purchase, with the banking system getting details of your credit card?

Similarly, the banking system needs to see details of your credit card, but not details of the transaction (the bank may start violating privacy norms by getting data on buying habits of the customer).

In other words, information should be made available to different parties purely on a *need to know* basis.

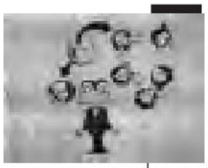
And that's where **SET**, or **Secure Electronic Transactions** comes in. SET is a protocol developed specifically by VISA and MasterCard (along with several other companies such as Microsoft, IBM, Netscape, GTE, etc.) to handle credit card transactions.

It insists on client side certification, and therefore takes care of security problems from both ends. It also takes care of non-repudiation.

Further, SET does not let the merchant see details of your credit card. Instead, these details are seen only by the banking system. Conversely, details of your purchase are only seen by the merchant, and not by the bank.

For the limited area of credit card transactions, SET is an extremely powerful and comprehensive security mechanism. Unfortunately, it is complex to implement, and is therefore used only where really tight security is required. Most sites on the Internet are quite happy with SSL.

Summary



We have just taken a look at one of the most commonly used mechanisms for making payment over the Internet—namely credit cards. We first took a look at credit cards in the real world. And then saw how the picture changed in the cyber world. We saw that in both worlds, the process of making payment using a credit card is more or less the same, but with some key differences. For instance, in the real world, you have a slip printed out when a transaction is authorised, and this slip is signed by the card holder. But in the cyber world this is obviously not possible, so we do without the signature.

The other thing we saw in the cyber world, was the fact that we had two completely different kinds of networks, possibly with completely different sets of protocols, connected to each other : The highly public Internet, and the high-security banking network. And discovered the concept of a payment gateway that acted as the interface between these two.

We actually saw two distinct processes when it comes to credit cards—first of all, the process of submitting your card (or entering details on your PC), getting the transaction authorised, and therefore collecting whatever it is that you have purchased. And then the back end process, where the various banks settle the transaction amongst themselves, and you, as the proud possessor of the card, are forced to pay up at the end of the month (by the bank that made the blunder of issuing you the card in the first place).

And finally, we took a look at a special security protocol called SET, which has been developed specifically to take care of credit card transactions. Unlike SSL, SET has in-built capability to take care of non-repudiation. Further, unlike SSL, it ensures that confidential information is seen only by the person or organisation that needs to see it—and by no one else. So for instance, the merchant does not get to see your credit card details. Conversely, the bank only gets to see your card details, and not what you are purchasing.

All this adds substantially to the e-Commerce architecture we have been building up. Because the application server is now connected to the payment gateway—and from there onwards to the banking network (see Fig. 16.5).

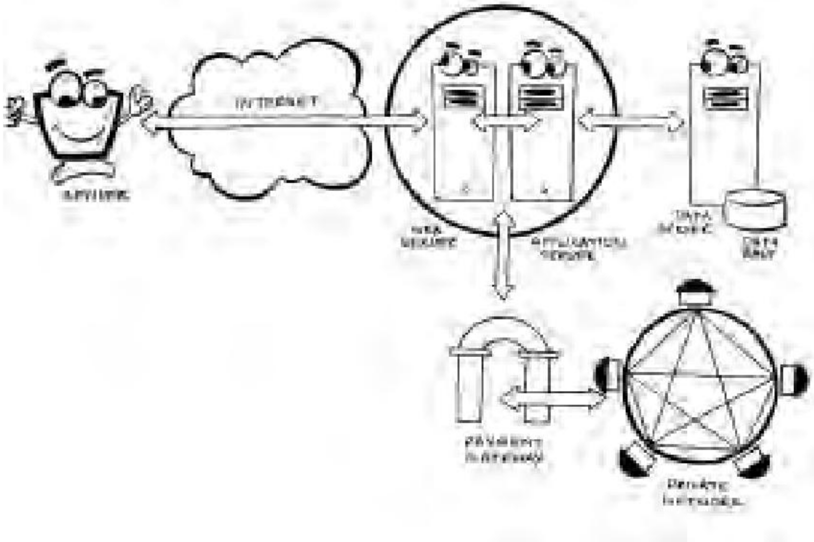


Fig. 16.5 *Our 4-tier e-Commerce architecture, connected to the banking network through the payment gateway*

So that's it my friend. Credit cards on the Internet. Nice and easy, aren't they?

Next time you casually present you credit card after taking your girl friend out for a date, think about it. There's a lot that is happening back there just to keep her happy!!!

17

And This is What We Have Learnt

Well, my friend, you have finally managed to wade through the entire book. And have reached the end (assuming of course that you did not skip chapters in between).

Wouldn't you now like to get a quick, bird's eye view of whatever we have learnt in this book?

Well, whether or not you do, I'm giving it to you...................

A BIRD'S EYE VIEW

As I'm sure you are aware, everything on the Internet starts and ends with the browser. You access the Internet or the world wide web through the browser (Chapter 1). Which in turn communicates with the web through the HTTP protocol.

Through your browser, you send out a request for a web page, giving the domain name of the web server that hosts this page—such as cindycrawford.co.uk, or tomcruise.com (of course, which of these sites you are more likely to go to, would depend on who you are).

You do know that this domain name needs to be converted into an IP address (Chapter 2). And this is done by a special series of servers called domain name servers (Chapter 4).

You also saw that all messages on the Internet are actually split into small packets and then re-combined at the other end, using a cute technique called packet switching. Which in turn is managed by another protocol, namely TCP or Transmission Control Protocol (Chapter 3).

The web page itself is written in a language called HTML or Hyper Text Markup Language. This language can show you text, or data, and can even include audio, video, and graphics (Chapter 5). But if you are designing a web site, you had better ensure that you use some sort of compression technique—else you'll have your viewers falling asleep waiting for a page to download !

So far so good. Now what happens at the web server end?

Actually, it could be one of several things. The web server could have what we call static pages, which are permanently written in HTML, and therefore do not change. So it simply dishes these pages out in response to the browser's request.

Or if the pages need to change frequently, you could use dynamic pages. Which means, for instance, that a web site giving stockmarket quotes, updates itself every minute to keep pace with the (hopefully) rising and (unfortunately) falling millions you have bet your life on!

And these dynamic pages, as good, old, Haribhai of sabziwala.com has discovered, get generated from a database (Chapter 6). Accessed of course, through CGI scripts.

Since CGI scripts tend to be slow, we brought in Java (Chapters 7 and 8). Which, as we have seen, has had far reaching effects on mankind (including the worth of mankind in the dowry market). So you would store Java applets in the web server, but transmit them along with the web page to the browser, and execute them there.

Or, you could continue with server side computing. Except that you could now use ASP or JSP, which are much faster than CGI scripts (Chapter 9).

And then we went beyond the web server. And looked at component based development—along with looking at room coolers. We said that components were a wonderful way to create software, because you could simply pick up ready-made components and put them together to create an application (Chapter 10). Of course, you would need to use a component framework such as Java beans, so that different components are able to understand and communicate with each other.

But then we saw a number of common services that were required across different components—and it certainly didn't make sense for programmers to re-write the programs for these common services every time.

And so was born the wonderful application server! Which provides a set of ready-made services, such as database connection pooling, or managing session persistence, or load balancing, to be used by any component running within the application server (Chapter 11).

And finally we looked at the cloak and dagger stuff—security. Which we said was a three–part subject. First, we had client end security (Chapter 12), where we examined viruses. We also looked at the concept of the sand box environment, which prevents Java applets from accessing any resources on your PC.

Then of course, we looked at server end security, and examined the concept of the firewall, which acts as a kind of barrier to anyone not authorised to cross the barrier.

And then of course, came the biggie—message security. In four parts. Namely :

(a) Confidentiality : No unauthorised person should read the message

(b) Integrity : The message should not be tampered with

(c) Authentication : You should know who you are dealing with

(d) Non–repudiation : You should not be able to deny sending a message that you did send

We saw that you needed something called public key encryption, to ensure confidentiality (Chapter 13). And using this, we also looked at digital signatures, to ensure integrity of a message (Chapter 14).

Authentication, of course, is real simple. Just as you need a passport or a driving licence to identify yourself in the real world, in the cyber world you need a digital certificate, or simply a certificate (Chapter 15).

We looked at the Secure Sockets Layer, or SSL, as a simple protocol to implement most of these message security mechanisms, except of course, the problem of non-repudiation.

And finally, we came to the real clincher. The part where everything else falls into place. The part where you actually part with your hard-earned rupees or dollars or pounds (or your father's hard-earned rupees, dollars or pounds). And make payment for buying something on the Internet (Chapter 16).

We took a look at one of the most commonly used mechanisms for making payment over the Internet, namely credit cards. Where we saw that you had two distinct processes involved : First of all the process of entering your credit card details into your PC, getting the transaction authorised, and therefore collecting whatever it is that you have purchased. And then the back-end process, where the various banks settle the transaction among themselves, and you, as the proud possessor of the card, pay up at the end of the month.

We also took a look at a special security protocol called SET, which has been developed specifically to take care of credit card transactions. SET is more powerful than SSL, but is also more complex and slower. Incidentally, SET implements non-repudiation as well.

And on that wonderful note—naturally what could be more wonderful than money—we concluded our journey through the fascinating world of e-Commerce.

GOOD BYE

So that's it !

I've told you all I wanted to about e-Commerce for the moment (there is of course, a lot more, but that will have to wait till I write another book, and you buy it).

I do hope you've learnt something from this book. And even more, I hope you had fun in the process.

As we now part company, I can only leave you with one word of advice. When you are on the Internet, keep your eyes and ears open. And go on learning. Or you'll forget whatever you have learnt so far.

Of course, whenever you wish to get back to this book, I'll be truly happy to meet you again.

Till then—or till we meet in my next book.....................

Good Bye. And Good Luck !!!

The McGraw·Hill Companies