Chapter 1

Introduction

The decade of the 1990s has seen unprecedented growth in wide-area data networks. This period has seen the graduation of the Internet from an experimental network used primarily by the research community to a commercial network used by millions of users. This rapid growth has been fueled primarily by one application, namely the World Wide Web (the "Web" for short). The easy to use point-and-click interface coupled with the appealing graphical content have made the Web an increasingly popular medium for publishing and accessing information. The dominant status of the Web in the Internet today is underscored by recent measurements which show that Web traffic constitutes over 70% of the total on wide-area backbone links in the Internet [102].

Information in the Web is arranged as *pages* stored on *servers*. Each page typically consists of text and other data such as *inline images*, with *hyperlinks* leading to other pages. Figure 1.1 shows the snapshot of a Web page. Users interact with the Web by browsing Web pages and following hyperlinks to other pages with a Web client program called the *browser*. Being able to *download* and browse Web pages quickly is crucial for good user-perceived performance.

The *Hypertext Transfer Protocol* (HTTP) [10] defines the way Web clients and servers communicate with one another. HTTP is layered on top of the *Transmission Control Protocol* (TCP) [89],

Figure 1.1 Snapshot of a Web page showing text, inline images and hyperlinks.

the de facto standard protocol for reliable data transport in the Internet. The structure of Web information and the manner in which users access it interact poorly with TCP. *Addressing the performance challenges that arise because of the mismatch between the Web application and the TCP protocol forms the core of this thesis*. We preview the challenges here, and defer a detailed discussion to later sections.

1. The transmission of a Web page from a server to a client involves the transfer of multiple distinct components, such as inline images, that comprise the page. A component is often of some value to the user independent of other components of the same page. So it is desirable to transfer the components concurrently and independently of each other to minimize user-perceived latency. However, TCP has historically been more attuned to applications such as remote login (e.g., Telnet [90]) and bulk file transfer (e.g., FTP [91]) which, to first order, transfer an undifferentiated, linear sequence of bytes.

2. Since Web browsing is an interactive activity driven by a human user, it involves bursts of activity corresponding to page download interspersed with idle periods corresponding to *user think time*. So Web traffic is bursty. The amount of data transferred during each burst is usually a few tens of kilobytes [65], which is much larger than a single telnet packet containing a few key-

strokes, but is also much shorter than typical bulk transfers. This poses a dilemma in the context of Web data transfer. On the one hand, we would like each burst transferred as quickly as possible. On the other hand, a burst may be too large to be transmitted without first probing the network to determine the available bandwidth. But the latter is often a time-consuming process (as it is in TCP).

3. Web data transfer tends to be highly asymmetric with an order-of-magnitude more data flowing from servers towards clients than in the opposite direction [65][1]. This asymmetry is partly responsible for the deployment of *asymmetric access networks*, which provide (relatively) low-cost, high-bandwidth connectivity from the network core towards end users (*downstream*), but much slower-speed connectivity in the opposite direction (*upstream*). Bandwidth asymmetry adversely impacts the performance of TCP data transfer in the downstream direction because the limited bandwidth in the upstream direction disrupts the flow of receiver feedback needed to sustain data transfer.

The rest of this chapter is organized as follows. We begin in Section 1.1 with a discussion of the rapid growth of the Internet in general and the Web in particular, which underscore the practical relevance of the problem we are trying to solve. In Section 1.2, we discuss the philosophy and design of the TCP/IP protocol suite that have played a key role in enabling this growth. In Section 1.3, we discuss the distinguishing characteristics of Web data and traffic. We follow this up with an overview of TCP in Section 1.4. In Section 1.5, we identify specific challenges that arise from the mismatch between the Web application and the TCP protocol. In Section 1.6, we discuss the impact of heterogeneous access network technologies on performance. Having enumerated the performance challenges, we outline the techniques that we have developed to address these challenges in Section 1.7. In Section 1.8, we discuss the design principles that have guided our work. We conclude in Section 1.9 with an outline of the rest of this thesis.

1.1 Growth of the Internet

The Internet, as we know it today, has evolved from the ARPANET, a research network sponsored by the U.S. Department of Defense. The original configuration in December 1969 consisted of

1. Such asymmetry is understandable given that it is much easier for individual users to consume information than to generate information.

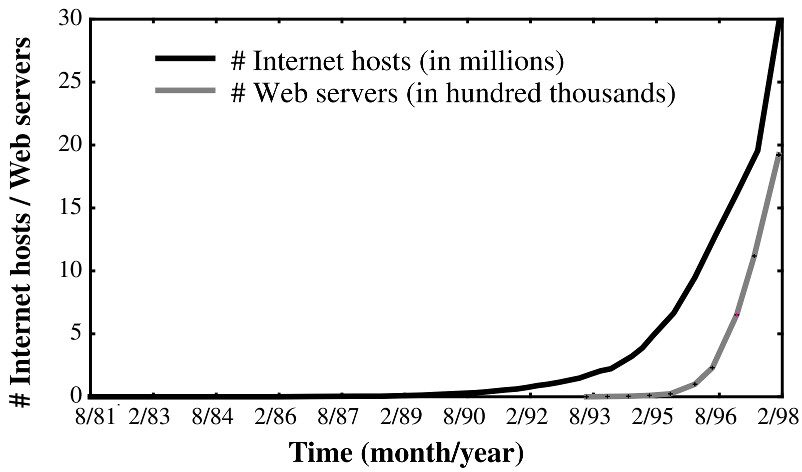

Figure 1.2 The exponential growth in the number of Internet hosts and the number of Web servers. (Sources: [79, 40])

four nodes (*hosts*), all located in the U.S. Today the Internet is estimated to have about 30 million hosts spread across more than 180 countries. This growth has sustained an exponential rate and shows no signs of slowing down. Figure 1.2 provides a glimpse of this rapid growth.

Two factors have contributed to this growth. The first factor is the ability of the Internet protocol architecture (colloquially referred to as the *TCP/IP protocol architecture*) to accommodate and assimilate heterogeneous network technologies. This architecture also enables the network to be grown in a decentralized manner with minimal involvement of a central controlling authority. This has facilitated the addition of new networks to the Internet on a daily basis.

The second factor that has fueled the growth of the Internet is the large number of useful applications it has supported and continues to support. Historically, these have included electronic mail, network news, file transfer, remote login, etc. In recent years, Web browsing has overtaken these to become by far the most popular application in the Internet, as evident from Table 1.1. Just as the total number of Internet hosts, the number of Web servers is also growing at an exponential rate, as depicted in Figure 1.2. In fact, it is the immense popularity of the Web that is largely responsible for the rapidly increasing numbers of hosts and users in the Internet. The Web is poised for even greater growth in the future as it increasingly becomes the method of choice for interfacing to a wide-range of devices such as printers, cameras, etc. The dominant position of the Web makes the subject of this thesis interesting and relevant. Next, we discuss the TCP/IP protocol architecture in some detail.

Protocol/Application	% packets	% bytes
HTTP (Web)	70	75
FTP (file transfer)	3	5
SMTP (e-mail)	5	5
NNTP (network news)	1	2
Telnet (network terminal)	1	< 1
DNS (name service)	3	1

Table 1.1 The fraction of traffic measured at a busy backbone link in August 1997 corresponding to various application-level protocols. (Source: [102])

1.2 TCP/IP Protocol Architecture

The Internet, as any other network, depends on a variety of protocols for its operation. Since networks comprise multiple distributed entities, network protocols are needed to establish a common basis for communication between these entities. Network protocols are analogous to a *lingua franca* that enables a group of people to communicate with each other.

It has long been recognized that a network protocol architecture should subdivide the communication problem into *layers* and have well-defined interfaces between these layers [111, 20]. Such a subdivision simplifies protocol design. However, many approaches to layering have been proposed and how function should be assigned to layers has been an ongoing controversy. Among the many approaches, the *OSI (Open Systems Interconnection) Reference Model* [111] is perhaps the most elaborate, and provides a useful framework to structure discussion of other approaches to layering as well.

The OSI model defines a seven-layer network protocol stack. Figure 1.3 lists the seven layers. Each layer provides a set of services that builds on the services provided by the layers lower in the stack. Briefly, the physical and data link layers include functionality needed for basic connectivity. The network layer enables the routing of data packets from one host to another via intermediate nodes. The transport layer provides services such as reliable end-to-end data delivery between a

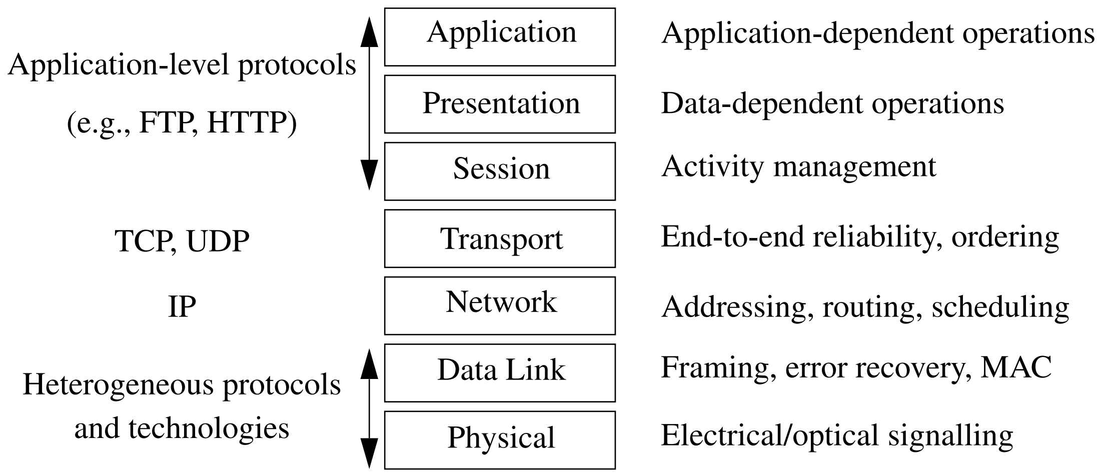

Figure 1.3 The 7-layer OSI network protocol stack. The annotations on the right indicate the typical role of each layer. The annotations on the left show how Internet protocols map on to the OSI model.

pair of hosts[1]. The session through application layers provide enhanced end-to-end services, such as data format conversion, that are often data-specific or application-specific.

It is useful to consider how the TCP/IP protocols [20] map on to the seven-layer OSI model. As shown in Figure 1.3, the TCP/IP protocol stack has fewer layers than the OSI model. The lowest two layers in the OSI model, physical and data link, operate within the confines of a single link. These layers provide a means for higher layers to move data from one end of a link to the other. As such, there is minimal interaction between the design of these layers and the TCP/IP protocols, the latter being concerned with only higher-layer issues. *IP* (*Internet Protocol*) is a network-layer protocol that provides addressing and routing services. Both these services exploit hierarchy to facilitate the growth of the network in a decentralized manner. *TCP* (*Transmission Control Protocol*) and *UDP* (*User Datagram Protocol*) are the two key transport-layer protocols in the Internet architecture. Finally, the top three layers in the OSI model are usually subsumed in application-level protocols, which tend to be as diverse as applications themselves. Examples include *HTTP* (*Hypertext Transfer Protocol*) for the Web, *FTP* (*File Transfer Protocol*) for file transfer and *SMTP* (*Simple Mail Transfer Protocol*) for electronic mail.

1. Unless otherwise specified, we only refer to *unicast* transport protocols in all of our discussion.

While layering is a convenient design tool, it can often lead to inflexibility and performance inefficiency when carried over to implementation [21]. This is primarily because the fixed function of each layer and the rigid interfaces between layers often result in a situation where one layer performs suboptimally because the information it needs for optimal operation is only available to another layer. In other words, there is a *mismatch* between the two layers.

Efficient Web data transport is difficult, in part, because of a mismatch between the application layer and the TCP layer. In this thesis we analyze this mismatch and the performance problems that it causes. We then propose a solution, *TCP session*, which resolves these problems by redefining the relationship between the functions performed by the TCP layer and the interface it provides to the application layer. Over the next three sections, we discuss the key characteristics of the Web application and the TCP protocol, and identify challenges arising from their mismatch.

1.3 World Wide Web

The World Wide Web is a large distributed system consisting of servers and clients spread across the Internet. Web servers are repositories of information. Web clients request information from servers, usually on behalf of human users.

Web clients and servers communicate using the *Hypertext Transfer Protocol* (*HTTP*) [10]. HTTP defines several *methods* that a client can specify in requests that it sends to the server. The most commonly used method is *GET*. The client issues a GET request to retrieve a *resource* from the server. A resource can be a static file or the output of a script executed at the server. The GET request identifies the resource with a *Uniform Resource Locator* (*URL*) [8].

We discuss the characteristics of Web data and Web traffic, which are very different from those of traditional applications and which lead to a mismatch with TCP.

1.3.1 Web Data Characteristics

As noted before, Web data is served in logical units known as *pages*. A Web page is usually composed of multiple components such as text, images, video, audio, etc. that are embedded in it. Table 1.2 provides a sampling of the diverse data types that occur in Web pages and their frequency of occurrence, as reported in [107]. The individual components are embedded in a Web page using the *Hypertext Markup Language* (*HTML*) [9].

Web Data Type	Description	Frequency of Occurrence
HTML	Hypertext Markup Language	76.3%
TXT	ASCII text	2.2%
PS	Postscript	1.8%
AU	Sun audio	0.7%
WAV	Microsoft WAVE	0.3%
GIF	Graphical Interchange Format	61.7%
JPEG	Joint Pictures Expert Group	7.8%
MPEG	Motion Pictures Expert Group	0.3%
MOV	QuickTime Movie	0.2

Table 1.2 A subset of Web data types and the frequency of their occurrence in Web pages. (Source: [107])

Users typically *download* entire Web pages from servers, rather just an individual component of a page. They use a *browser* program to download pages. During a download, the browser uses HTTP to first retrieve the HTML file corresponding to the Web page. It then parses the HTML to determine which components, if any, are embedded in the page. Finally, it retrieves each of the embedded components using HTTP. Although not necessary, the components of a Web page usually reside on the same server as the HTML file.

The browser renders the components in a manner that is convenient for the user to perceive. For instance, image components are displayed on a screen while audio components are played on an audio device. The browser may invoke a separate helper application to render components of data types that it is unable to process.

The interactive nature of Web browsing makes it critical to minimize latency. However, an important point is that the browser can often render a component while still in the process of retrieving it. *So the latency perceived by the user is often determined by the entire process of rendering the Web page, and not necessarily the time to download the entire page*. For instance, rendering a set of (progressively-coded) inline images concurrently may reduce user-perceived latency compared to rendering them sequentially, even if the total download time is the same in both cases.

A Web page component may be more or less valuable to the user than other components. Perceived latency would improve if the valuable components are rendered more quickly, even if the less valuable ones are slowed down as a result. For instance, the user may be more interested in viewing the headline image on a news page than an advertisement banner.

Thus, Web data transport involves the transfer of multiple, independent components. It is desirable to have the flexibility to control the progress of the individual transfers depending on the (possibly dynamically varying) preferences of the user. These characteristics make Web data transfer fundamentally different from the traditional bulk data transfer. The latter, to first order, is only concerned with the transfer of an undifferentiated, linear sequence of bytes.

1.3.2 Web Traffic Characteristics

We now discuss the characteristics of the traffic between Web servers and clients in aggregate, without being concerned with the internal structure of the traffic that we discussed in the previous section.

A user's Web interactions are often in the form of *browsing sessions*. During a browsing session, the user may download several pages from the same server. The presence of hyperlinks pointing to others pages residing on the server contributes to this locality of reference. There is often an idle period, called the *user think time*, between successive Web page downloads within a browsing session. This period corresponds to the time that the user spends perusing the Web pages retrieved previously or engaged in other activity. The length of the idle period typically ranges from a few seconds to several minutes; [65] reports a median user think time of 15 seconds.

As a result, Web traffic tends to be bursty. Each Web page download constitutes a burst. Individual Web pages tend to be small in size. Several studies have indicated that Web page sizes are typically on the order of a few tens of kilobytes; [65] reports an average page size of 26-32 KB and average component size of 8-10 KB. Therefore, the bursts tend to be intermediate in length compared to those in traditional applications — much longer than the 1-2 KB bursts in Telnet, but at the same time, much shorter than bulk transfers that may be hundreds of kilobytes long.

To minimize latency, it is important that a Web page transfer utilizes the available bandwidth effectively. However, the bursty nature of Web traffic makes this task difficult. We discuss this mismatch in Section 1.5, after outlining the TCP protocol in the next section.

1.4 Transmission Control Protocol (TCP)

TCP is the predominant transport protocol in the Internet. TCP builds on the service that IP provides of transporting data packets from a source host to a destination host. The distinguishing characteristic of the transport layer compared to layers beneath it in the stack is that it operates *end-to-end*, i.e., it involves peer-to-peer communication between processes at the communicating end-hosts. Before discussing TCP in detail, we outline the functions that the transport protocols may perform in general:

- *Demultiplexing*: A network level address, such as an *IP address*, only identifies a host. The transport layer could provide a mechanism to address individual processes on a host, and demultiplex incoming data packets to the correct recipients.
- *Reliable data delivery*: Packets may be dropped due to resource constraints on the network path from the source host to the destination host. Indeed, this is a deliberate design choice made in the IP service model [20]. The transport layer could perform *loss recovery* to mask packet drops from the higher layers.
- *In-order data delivery*: The network layer could deliver packets to the receiving host in a different order than they were sent in. The transport layer could again mask such reordering from the higher layers.
- *Flow control*: This is a mechanism to keep a sender from transmitting data packets faster than the receiver is able to consume them.
- *Congestion control*: This is a mechanism to keep a sender from overwhelming shared network resources (such as link bandwidth and router buffer space) by sending too fast.

TCP is a full-fledged transport protocol that performs *all* of the functions listed above. The fundamental abstraction in TCP is a *connection*, which provides reliable, ordered byte-stream service to a higher layer, i.e., a user of TCP. In this model, the sender transmits a sequence of bytes over a connection and *all* of those bytes are delivered to the receiver in the *same order* they were sent in. This simple abstraction for reliable communication greatly simplifies the task of an application developer. This has led to the adoption of TCP as the basis for a wide range of applications, including remote login, file transfer, and more recently, the Web.

TCP performs the functions listed above at the granularity of a connection. Each TCP connection transfers an ordered byte-stream independently of other connections. TCP provides no message or record boundaries within a connection. Each connection also performs congestion control and loss recovery independently of other connections. The tight coupling of the ordered-byte stream service abstraction with congestion control and loss recovery mechanisms leads to suboptimal performance for Web transfers, as we will discuss in Section 1.5.

Since TCP provides reliable data delivery on top of a network that could drop packets, it includes mechanisms to detect data loss and recover from it. The TCP receiver sends *acknowledgement* packets (*acks*) to the sender to indicate which packets it has received. The TCP sender buffers all unacknowledged data and infers data packet loss by observing acks. When it detects a loss, the sender retransmits the corresponding data from its buffer.

The TCP sender detects packet loss either based on acks it receives in response to data it had transmitted (*data-driven loss recovery*) or the absence of an ack for data sent well in the past (*timer-driven loss recovery*). In either case, loss recovery happens independently for each connection. Data-driven loss recovery is the more desirable alternative from the viewpoint of quick recovery because as discussed in [108], timers need to be set conservatively to contend with potential variability in feedback (i.e., ack) delay.

Data-driven loss recovery is more effective the greater the length of the data stream. Intuitively, each data packet provides the sender a means to probe the receiver because of the response it elicits in the form of an ack. The greater the number of probes, the easier it is for the sender to reliably determine which packets have reached the receiver and which have not.

Acks also play a critical role in TCP congestion control. Since the available network bandwidth is not known *a priori*, a TCP connection starts with a low data rate. As acks are received, indicating the successful delivery of data to the receiver, the connection transmits new data packets (*ack clocking*). It also ramps up its data rate (*slow start* [51]). The connection may eventually congest the network and cause packet loss. When a packet loss happens, the connection backs off by halving its data rate.

1.5 Mismatch Between the Web Application and the TCP Protocol

Having discussed the key characteristics of Web data and traffic, and the salient features of the TCP protocol, we now discuss the performance problems arising from their mismatch.

First, a Web page download involves multiple, logically-separate transfers between the server and the client. TCP in itself does not provide any means to demarcate sub-streams within a connection. So, with the HTTP/1.0 protocol [10], Web client and server applications map each transfer onto a separate TCP connection. Doing so provides a separate data stream for each transfer, which is desirable. However, the concurrent connections compete in an uncoordinated manner, which aggravates congestion and causes unpredictable performance for each connection. A connection carrying valuable data may perform worse than one carrying less important data. Furthermore, data-driven loss recovery suffers because each connection is typically short in length (only a fraction of the length of the entire Web page transfer). Therefore, the first challenge of Web data transport is:

Challenge #1: *Supporting multiple, concurrent data streams between a server and a client efficiently.*

Second, it is difficult for Web transfers to utilize the available bandwidth effectively because of the bursty nature of Web traffic. At the onset of a burst, the sender is unaware of the level of load in the network, so it is forced to probe the network to discover how much bandwidth is available. But this probing requires time. In particular, TCP's slow start process may require several round trip times (RTTs) to ramp up the data rate to match the available bandwidth. As a result, the latency for the short burst increases and its bandwidth utilization suffers.

To avoid these problems, a connection may start with a high data rate. However, doing so increases the chance that the network will become congested, especially when several connections start with a high data rate simultaneously. Network congestion could lead to heavy packet loss and degraded performance for all connections. Clearly, this is not a desirable outcome.

Therefore, the second challenge of Web data transport is:

Challenge #2: *Enabling short and bursty transfers to utilize the available network bandwidth effectively, without adversely impacting the stability of the network.*

Before we present our solutions to these challenges, we discuss a third challenge, which arises due to the characteristics of certain access network technologies.

1.6 Impact of Heterogeneous Access Network Technologies

Access networks provide connectivity to remote locations such as residences and small offices. Access network technologies span a broad spectrum, as summarized in Table 1.3. The traditional and still dominant technology is dialup phone lines. However, its extremely limited bandwidth has spurred the development and deployment of higher-speed alternatives such as cable modem networks. However, some of these new access network technologies interact poorly with TCP, and hence cause performance problems for Web transfers. We list below the key characteristics of these networks and the performance problems these cause:

1. *Bandwidth asymmetry:* This arises in a variety of networks, including those based on ADSL, cable modem and, satellite technologies. The ratio of downstream to upstream bandwidth can range from 10 to 1000 depending on whether the network is inherently two-way, or requires a separate upstream channel such as a dialup phone line. Although Web browsing has asymmetric bandwidth requirements, bandwidth asymmetry could adversely impact the performance of Web downloads in the downstream direction. This is so because congestion in the upstream direction could disrupt the flow of acks, which is critical to sustaining good TCP throughput in the downstream direction. Performance is especially impacted when there is other concurrent traffic in the upstream direction.

Access Technology	Physical Medium	Bandwidth	Latency
Dialup modem	PSTN	28-56 Kbps D, 28-33 Kbps U	50-100 ms
ISDN	PSTN	64-128 Kbps	10 ms
ADSL	PSTN	1.5-9 Mbps D, 16-640 Kbps U	
1-way cable modem	Coax cable/HFC	10-30 Mbps D, 28-56 Kbps U	
2-way cable modem	Hybrid fiber coax	10-30 Mbps D, 0.096-4 Mbps U	
MMDS	Wireless (2.5 GHz)	10-30 Mbps D, 28-56 Kbps U	
LMDS	Wireless (28 Ghz)	>100 Mbps	
CDPD	Wireless (850 Mhz)	19.2 Kbps	
Packet radio	Wireless (915 MHz)	30-40 Kbps	400 ms
Satellite	Wireless (Ka, Ku)	16 Kbps-60 Mbps D 16 Kbps-2 Mbps U	20-400 ms

Table 1.3 A summary of various access network technologies. "D" and "U" refer to the downstream and upstream directions, respectively.

2. *Large delay-bandwidth product:* This situation arises in particular in geostationary satellite networks, in which the latency is large (several hundred milliseconds) and so is the bandwidth (several Mbps). Because of TCP slow start, short Web transfers utilize the available bandwidth very poorly. A similar situation can arise even in highly asymmetric terrestrial networks when there is a bidirectional flow of traffic (Chapter 9).

3. *Constrained bandwidth:* At the other extreme are bandwidth-constrained access networks such as dialup modems and the mobile data networks with bandwidths of the order of a few tens of Kbps. In such networks, multiple simultaneous data streams of a Web transfer can cause severe congestion and degrade performance.

The trend towards extremely high-speed network backbones makes the access network a critical factor in end-to-end performance. This leads to our third challenge:

Challenge #3: *Achieving high bandwidth utilization and low latency over a wide range of access network technologies, including asymmetric-bandwidth, large bandwidth-delay product and bandwidth constrained networks.*

1.7 Addressing the Challenges

In this thesis, we develop several new techniques to address the three challenges that we have identified in this chapter.

We first developed a new connection abstraction for HTTP, called *persistent-connection HTTP* (*P-HTTP*). The key ideas are to share a *persistent TCP connection* for the transfer of multiple Web page components and to *pipeline* the transfers of these components. By avoiding the competition among connections that afflicts HTTP/1.0, P-HTTP reduces latency significantly. The ideas in P-HTTP, developed by us in 1994 [85], have been adopted by the HTTP/1.1 protocol [30].

However, a fundamental drawback of P-HTTP is that the service abstraction of a TCP connection imposes strict ordering on Web transfers that correspond to possibly unrelated components of a Web page, as illustrated in Figure 1.4(a). This violates the *application-layer framing* (*ALF*) principle enunciated in [21] and can degrade user-perceived latency.

This drawback of P-HTTP led us to develop a comprehensive solution, which has two components. The first component, *TCP session,* was proposed by us in [82] and refined in [6]. TCP session decouples TCP's ordered byte-stream service abstraction from its congestion control and loss recovery mechanisms. It treats data belonging to the set of concurrent connections between a server and a client as a single, unified data stream for the purposes of congestion control and loss recovery, thereby *integrating* the latter mechanisms across the set of connections. However, the individual connections retain their independent byte-stream abstractions. *Thus, TCP session combines the flexibility of separate connections with the performance efficiency of a shared connection.* The integration of two TCP connections into a session is illustrated in Figure 1.4(b). TCP session decreases download time by up to a factor of ten compared to HTTP/1.0 layered on standard TCP.

TCP session has the important attribute that it is backward compatible with the existing TCP protocol. TCP session does not alter TCP's messaging semantics, so deployment only involves local

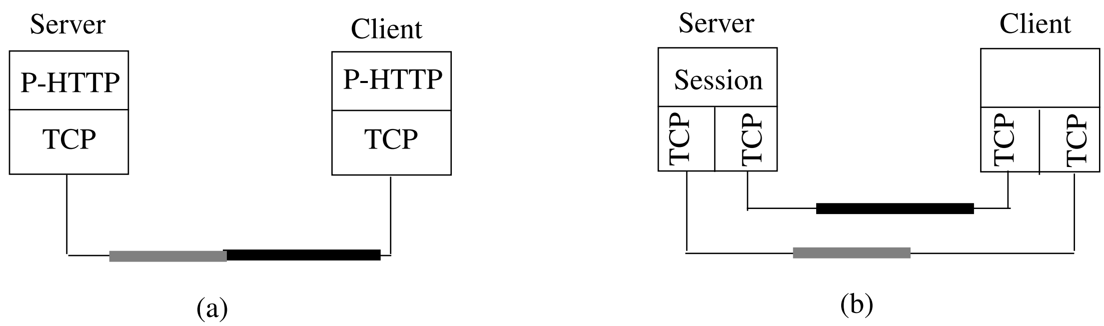

Figure 1.4 The progress of two logically-separate Web transfers with (a) P-HTTP and (b) TCP session. P-HTTP imposes a linear order on the two transfers while TCP session does not. P-HTTP operates both at the client and the server whereas TCP session is confined to the server.

changes at the sender. In the context of the Web, the senders are primarily the servers. And servers are probably easier to upgrade than the more numerous clients.

The second component of our solution is *TCP fast start*, proposed by us in [84]. The key idea is using *persistent state*. TCP fast start improves bandwidth utilization for short transfers by reusing information about the network conditions cached in the recent past. Thus, it can decrease the cost of probing for bandwidth significantly, especially in networks with a large bandwidth-delay product.

To avoid adverse effects in case the cached information is stale, TCP fast start exploits priority dropping at routers. By marking packets sent during fast start as low-priority, the sender minimizes the adverse impact that fast start could have on other connections in the network. TCP fast start also augments TCP's loss recovery mechanisms to quickly detect and abort a failed fast start attempt.

In addition to TCP session and TCP fast start, we have developed a set of end host-based and router-based techniques to improve performance substantially in asymmetric networks [5]. These techniques reduce both the disruption in the flow of receiver feedback (in the form) of acks and TCP's dependence on such feedback. In certain situations, these help decrease Web page download time by a factor of fifteen.

1.8 Design Principles

Before concluding this chapter, we discuss the principles that have influenced the design of the techniques developed in this thesis:

1. *Robustness*: While the techniques we design may be targeted to addressing performance problems that arise in specific situations (such as short Web transfers or asymmetric-bandwidth networks), they should not cause performance degradation in other situations or create new performance problems of their own.

2. *Wide applicability*: Our solution should be as widely applicable as possible across networks and applications. For instance, techniques designed to improve the performance of Web browsing should also help other applications, such as distributed transaction processing, that have similar communication requirements. In particular, it undesirable to have a solution that is tied to a specific application-level protocol, such as HTTP.

3. *End-to-end principle*: To the extent possible, our solution should only require the participation of end-hosts in the network, such as servers and clients. There should be minimal additional burden placed on routers in the interior of the network. This results in a scalable design because the additional work entailed in supporting a new set of hosts gets distributed across them. This principle, first formulated in [96], has been a cornerstone of the growth and success of the Internet.

4. *Incremental deployment*: It should be possible to deploy our solution incrementally since the size and the heterogeneity of the Internet preclude a global deployment in a short span of time. Even a partial deployment should yield significant performance benefits. In this context, a partial deployment could mean that only subset of the solution is deployed and/or its deployment is confined to a subset of the network.

We believe that these represent good engineering principles that have contributed to the success of the Internet. In Chapter 10, we evaluate how successful we have been in conforming to these principles.

1.9 Thesis Outline

The remainder of the thesis is organized as follows. In Chapter 2, we present background information on the TCP and HTTP protocols, and survey related work, pointing out its relationship with our work.

In Chapter 3, we discuss our research methodology. This chapter includes a discussion of our experimental network testbed, traffic traces, simulation and implementation environments, and performance evaluation metrics.

In Chapter 4, we present an analysis of HTTP performance. We develop a simple mathematical model for HTTP. We also analyze the interactions between HTTP and TCP using traffic traces from the 1996 Olympics Web server run by IBM.

This analysis leads into Chapter 5, where we introduce our initial solution, *persistent-connection HTTP* (*P-HTTP*). We discuss two key ideas, persistent connections and pipelining. We demonstrate the significant performance benefits of P-HTTP via experimental evaluation. We also point out its limitations, which set the stage for the comprehensive solution presented in Chapter 6 through Chapter 8.

In Chapter 6, we present *TCP session*. We discuss the three components of TCP session: integrated congestion control, connection scheduling, and integrated loss recovery. We discuss how TCP session combines the strengths of both HTTP/1.0 and P-HTTP, and present a detailed performance analysis.

In Chapter 7, we discuss some advanced issues pertaining to TCP session, including exposing connection scheduling to applications, and adapting integrated congestion control to environments with Web proxies.

We present *TCP fast start* in Chapter 8. We discuss the benefits and potential pitfalls of reusing cached information. We explain how TCP fast start avoids the pitfalls by exploiting priority dropping in routers, and by employing augmented loss recovery procedures. Our performance analysis includes experiments over the DirecPC satellite network [27].

In Chapter 9, we discuss a set of end-to-end and router-based techniques to address the problems arising in asymmetric networks. We present performance evaluation of both unidirectional and bidirectional traffic. Our analysis includes measurements over a cable modem network from Hybrid, Inc [47].

We conclude this thesis with a summary of our work and contributions in Chapter 10. We revisit the design principles listed in Section 1.8 and evaluate how successful we were in conforming to them. We also provide directions for future work.

In Appendix A we discuss the software tools we used for our analysis and evaluation. These tools include both existing ones and new ones that we developed.

In Appendix B, we discuss the implementation of the integrated loss recovery component of TCP session in detail.