4 Web Engineering and XML Processing– A Closer Look[1]

[1] Co-authored by Kamran Khalid

4.1 Introduction

This chapter deals with the detailed Web design and engineering issues that could not be covered in Chapter 2 ("Web, XML, Semantic Web, and Web services") of the Middleware Module. The topics include a closer look at HTTP and Web performance and workload characterization issues. In addition, a treatment of different aspects of XML processing is given including a discussion of the tradeoffs between storing XML data in relational databases versus XML databases.

4.2 HTTP Design – A Closer Look

HTTP is at the foundation of Web design and engineering. This section reviews HTTP server design considerations and then highlights the developments in HTTP 1.1. Since it augments the HTTP discussion in Chapter 2 of the Middleware Module, a review of HTTP in Chapter 2 is highly recommended before proceeding.

4.2.1 HTTP Server Design

HTTP servers, commonly known as ***Web servers***, are responsible for servicing requests from the Web browsers over HTTP. Many HTTP servers are commercially available at the time of this writing. Examples are Apache and Tomcat servers from the Apache Group (www.apache.org), and the IIS server from Microsoft. Figure 4-1 shows a conceptual view of HTTP server. The overall operations of an HTTP server are:

- Open a socket: This opens a TCP socket to receive the incoming traffic.
- Read HTTP request. When a client issues a request by invoking a URL, this instruction is executed at the server side
- Parse the HTPP request and determine whether the request is authorized. The client prepares and then sends an HTTP header along with a request. This step parses the header (we will review HTTP headers later)
- Read/update the specified file as indicated by the request. This may, for example, involve reading an HTML page or invoking a CGI gateway.
- If the file could not be accessed, then issue a 404 message indicating a file not found error.
- If there are no errors, then the server constructs a response along with some header information.
- Write to the socket so that the message is transmitted back to the browser.
- Go back to HTTP read for the next request.

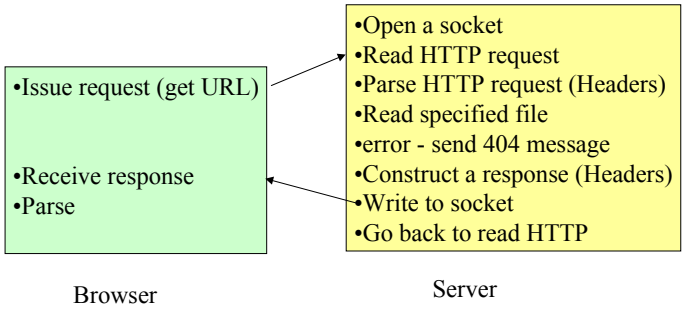


Figure 4-1: HTTP Server Design

At some point in handling a request, the Web server may create a log entry to record information about the request. For example, a server may create a log entry for security failures. The server can also create and use

cookies in these steps. For example, a server can create a client side cookie when it writes to the client the first time or it may create a server side cookie after authenticating the client.

A Web server, as discussed in Chapter 2 of the Overview Module, typically handles multiple client requests at the same time. These requests must share access to server files, processor, memory, and network interfaces. Web server architectures typically follow two approaches to handle large number of requests:

- Event driven architectures that use a single process that alternates between servicing different requests. In this type of design, a single process handles requests one at a time. Event driven servers are simple and easy to develop. But they can be very slow especially if a back-end system has to be accessed. Consider, for example, the situation where a server process handles a database query. In this case, the server process is free only after all the steps (connection to the database, issuing the query, collecting the results, and closing the connection) have been completed.
- Process oriented architectures where a service is allocated to a separate process. In the process-driven architectures, one process per request (user) is created. As more users issue requests, more processes are created. This design can handle several users simultaneously. Typically a master process listens for requests and invokes a process for each user. This design is very well suited for large scale Web servers where the number of processes can increase with number of users without running out of system resources.

Naturally, a hybrid server architecture would combine the two approaches. Different commercially available Web servers use different architectural approaches. It should be noted that these architectural approaches are by no means new – they have been used to design other servers such as database servers.

Several companies build Web servers. In fact, it is possible to develop a simple Web server in a classroom environment[2]. Development of a Web server requires several skills. To start, the developer must know how to issue TCP/IP socket calls because the server must be able to open, read, write, and close the TCP/IP sockets. as shown in the above figure. The book "TCP/IP Network Programming" by Stevens (Prentice Hall, latest edition) is considered a classic reference for this type of work. The Java developer site (http://developer.java.sun.com/developer/codesamples/examplets/java.net) shows how Java can be used to develop socket level programs. In addition, knowledge of issuing operating system calls to read/write directories and files is needed. Finally, some knowledge of building schedulers that can handle multiple requests is highly desirable. These topics, if you have not guessed yet, are beyond the scope of this book.

4.2.2 HTTP Methods and Headers

Web clients and servers exchange ***messages*** over HTTP. HTTP messages contain either the request or the response. The HTTP request message is sent from the client and contains the Universal Resource Identifier (URI), a request method, protocol version, request modifiers, client information, and possible body contents. The response message is sent from the server and contains the response, naturally, along with other information such as the type of server (e.g., Apache), time, and the size of entity returned. The two key players in HTTP messages are the methods and the headers. The general format of an HTTP message is:

Method

Zero or more headers

Body

Before describing HTTP methods and headers, let us briefly talk about client/server systems design in general. Most clients and servers pass information to each other through ***signatures*** which represent the parameters exchanged. Consider, for example, the case where a client wants to read an html file joe.html and does not want to use caching. In this case, the client could issue the following command:

GET (joe.html, no-cache, buffer area for Joe's file)

[2] My students at University of Pennsylvania developed a simple Web server as part of an assignment in the Web Technologies course.

The signature for the GET method in this case is (file name, cache-options, buffer area) where the first two parameters are input parameters and the third parameter is a return area where the server puts the file read. Another approach is to use a very simple GET method and pass any additional information by using headers. In this case, the GET command will look something like this:

```
GET joe.html
Header1: No-cache
Header2: buffer area for Joe's file return
```

HTTP uses headers while many client/server systems like CORBA use signature. Thus as more options are added in HTTP, it translates to more HTTP headers. This is the main reason why a great deal of activity in HTTP evolution is centered around the headers. For example, HTTP 1.1 has added almost 40 more headers to the family of headers that existed in HTTP 1.0.

4.2.2.1 HTTP Methods

An HTTP method is the most important part of a message because it shows the type of operation that needs to be performed. Examples of the common methods used in HTTP 1.0 are GET, HEAD, and POST.

GET Method. This is most commonly used and indicates that a document needs to be retrieved from the server. This method either retrieves the document specified or a default document. Web servers return a default page if you do not specify a document. Web browsers use the GET method regularly to download HTML pages. Here are some examples:

```
GET http://www.w3.org   http/1.0
GET http://www.w3.org/cgi-bin/query?q=wml      http 1.0
```

HEAD method. This method is used just to get meta information (headers) without transferring the entire document. This method is typically used to test hypertext links for validity, recent modifications, and accessibility. The server only retrieves the metadata about a resource. For example, consider the following method:

```
HEAD   http://www.me.com/myhome.html   http/1.0
```

This returns the following message that may be used for debugging, or security:

```
HTTP/1.0 200 OK
Content-length: 3200
Last-modified: Monday, April 7, 2002
Content-Type: text/html
```

POST Method. This method is used to update or provide input to a process on server (e.g., a shopping cart, credit card processor). It can be used as an alternate to get to pass parameters to a CGI program: here is an example:

```
POST http://www.w3.org/cgi-bin          HTTP 1.0
Content-length : 40
CRLF
query?q=wml
```

4.2.2.2 HTTP Headers

In addition to methods, HTTP relies heavily on **headers** to exchange information between Web clients and servers. An HTTP header is a free-format character string that is used to "parameterize" the HTTP requests or responses. HTTP headers play a key role in Web by specifying the information that needs to be accessed through HTTP GET and by describing the information that is actually retrieved. HTTP headers come in different flavors:

- General headers that can be used in request as well as response

- Request headers that can be used in request only
- Response headers that can be used in responses only

General headers are used in requests as well as responses. HTTP 1.0 defined only two general headers: **Date** that specifies the date and time when the message is originated, and **Pragma** that tells the receiver to do something (e.g., do not use cache). Here is an example of a GET that specifies the date and requests no caching.

```
GET /faculty.html HTTP/1.0  (request line)
Date: Wed, 25 sept 2001 (General headers)
pragma: no-cache
```

Request headers are used by the clients to provide information and/or directions to the server. For example, the following request headers specify the client:

```
From: aumar1@hotmail.com (request headers)
user-agent: netscape
```

Response headers are used to provide additional information about the responses and the server that produced the response.

As we will see in the next section, HTTP 1.1 introduced several new methods as well as headers.

4.2.3 HTTP 1.1 Highlights

HTTP 1.1 was developed to address many issues with HTTP 1.0. Some of these issues are:

- Depletion of IP addresses due to the popularity of the Internet
- Inefficiencies of using TCP for short responses typical in Web (i.e., TCP connects, transfers and disconnects for every response).
- Statelessness of HTTTP (i.e., HTTP cannot maintain sate across sessions)
- Security concerns such as sending passwords in clear text
- Dealing with proxies effectively and efficiently

The following capabilities of HTTP 1.1 were developed to address these issues:

Virtual Web hosting (Internet Address Conservation). Let us suppose that a serious businessman (Zombie Jr.) wants to run a Web site. His Web hoster (www.graveyard.com) can give him a site www.graveyard.com/zombie.html by simply putting zombie.html in the Web server directory. Suppose that the good Mr. Zombie insists on his own URL (www.zombie.com). Here are his choices.

With HTTP 1.0, there is a problem with resolving multiple site names to a single IP address. Simply stated, it cannot be done. For example, the URL: www.graveyard.com/zombie.html is translated to

```
GET zombie.html HTTP 1.0
```

Note that the server name is stripped from the GET request. Thus, you cannot put more than one server on the same IP address. The ISP has to obtain multiple IP addresses (one per server) on the same machine to support multiple servers. In HTTP 1.0, DNS found the IP address and the operating system matched IP address to the server.

HTTP 1.1 solves this problem by retaining host name through another header "Host". Thus the same URL: www.graveyard.com/zombie.html is now translated to the following:

```
GET zombie.html HTTP 1.1
Host: www.graveyard.com
```

Because of this, a Web host can install multiple servers on the same IP address (www.graveyard.com, www.zombie.com, etc.).

Hop by Hop Mechanism. In most cases, an HTTP message between end-points passes through several intermediaries ("hops") such as proxies. HTTP 1.0 used same techniques (e.g., compression) between end-

points across all hops. HTTP 1.1 allows different techniques between intermediaries so that you can use different compression with a local proxy. This is implemented through a connection header (connection header1). By allowing different compression techniques between intermediaries, there is a possible positive impact on the traffic between end nodes.

Messages and Headers. HTTP 1.1 offers many new features, refinements and new error codes. These include (see section 7.2 of [Krishnamurthy 2001] for a very detailed treatment of this topic):

- Several new request methods are introduced in HTTP 1.1. HTTP 1.0 had only three methods (GET, HEAD, POST) that have been widely used. Some other HTTP methods (e.g., PUT and DELETE) were not well specified in HTTP1.0. In HTTP 1.1, GET, HEAD, and POST have been retained; PUT and DELETE have been formally specified, and three new methods (OPTIONS, TRACE, CONNECT) have been added for more flexibility.
- New headers are added for flexibility. As stated previously, HTTP headers are the primary means of accessing metadata and to change the behavior of a request. HTTP 1.0 had only two general headers: "Date" to indicate message generation date and time, and "pragma" to indicate message directive. HTTP 1.1 adds several new headers (e.g., cache-control, connection, trailer, warning). In addition, several new request and response headers are added in HTTP 1.1.
- New response codes have been added for clarity. For example, response codes are used to indicate the type of error. HTTP1.0 had 16 response codes but HTTP1.1 supports 41 response codes.

Caching: There are many options to control caching in HTTP 1.1. For example, you can control requests and responses. You can also specify no-cache, only-if-cached (access cached only), maximum age (do not use cache older than this), etc. A new ***"entity tag" (ETAG)*** has been introduced that specifies the information to be accessed (e.g., PDF document time and date). This allows comparison of cached information with new -- thus new information overrides cache (you do not have to hit the "reload" button to refresh the cache). There are many, many other caching options.

Proxies. In HTTP 1.1, proxies are formally recognized and facilitated accordingly. Proxies can be used for conversion of HTTP 1.0 and HTTP 1.1 messages. Moreover, you can add information (e.g., through ETAGs) that is specific for proxy processing (i.e., only some documents with certain ETAGS will be processed by the proxies. HTTP 1.1 proxies also can handle improved caching, connection management, and security.

Bandwidth optimization: To reduce workload, HTTP 1.1 minimizes resends -- it can send deltas or parts of documents by using the "RANGE" header. For example, you can use the following RANGE header to send partial documents (e.g., pages 10-30 of a PDF file):

```
GET bigone.pdf HTTP 1.1
Host: www.junk.com
Range: 10-30
```

You can also use transforms for sophisticated compression. You can also use "Expect/continue" to ask server what to expect. In other words, you do not send at all if receiver cannot handle it.

Connection Management. HTTP 1.1. offers several facilities for better connection management. For example, the following statements can keep a connection alive:

```
GET /home.html html 1.1
Connection: Keep-alive (keep connections alive)
```

Many proposals in HTTP 1.1 deal with persistent connections and pipelining (multiple requests). There are several improvements in this area.

Message transmission: HTTP 1.1 blocks ("chunks") the messages for transmission efficiency. HTTP 1.0 sent long messages without chunking. HTTP 1.0 used message length to ensure receipt safely. HTTP 1.1 uses "chunking" to send long messages (breaks messages into chunks and sends a zero length to indicate end).

4.2.4 Current Activities in HTTP

At the time of this writing, HTTP/1.1 is a stable specification. In addition, an HTTP extension framework has been specified to coordinate the "tension" between private agreement and public specification of HTTP. This framework also accommodates extension of HTTP clients and servers by software components.

W3C has closed the HTTP Activity because the Activity has achieved its goals of developing a successful standard that addresses the weaknesses of earlier HTTP versions. Work on efforts such as HTTP-NG (HTTP Next Generation) that considered reengineering of the HTTP protocol architecture has been stopped. The main area of continuing work is to track implementation experience with HTTP/1.1 and HTTP extensions. In addition, W3C staff is maintaining an Overview page on HTTP on the W3C website.

W3C has started new protocol-related work in the XML Protocol Activity. A new working group, called the XML Protocol Working Group, is defining an HTTP binding for XML Protocol, which is a higher-level protocol..

4.3 Web Performance and Workload Measurement

This section gives an overview of Web performance and workload measurement by first presenting a simple performance model and then discussing how the parameters of this model can be measured and characterized.

4.3.1 A Simple Web Performance Model

Let us introduce a simple procedure for estimating the performance of a Web application. Simply stated, response time of an application is the elapsed time between request submission (clicking of a URL on the browser) and completion (display of entire page on the browser). In the simplest case, the total response time RT of a Web application is given by the sum of all processing and queuing delays: Without queuing, RT, the response time, is given by

$$RT = \sum_I N(i)*S(i) \qquad \text{…….. (3.1)}$$

where S(i) = time needed for completion of service i, N(i) = number of times service i is needed, and service i represents any activity needed to complete a Web request (e.g., transmission of the messages between Web clients and servers over a networks, processing of the message at the server to produce the results, and transmission of the results back to the user). S(i) and N(i) can be easily measured through experimentation. In these formulas, * is used to indicate multiplication. Let us consider the following example to illustrate the usefulness of this simple formula.

Example 1: Web client of a Web application sends a 20 byte customer account number to the Web server. The server searches a customer database for the account number and then sends the customer information (2000 bytes) to the client. The Web server resides on computer M that is connected to users through 5600 bps WAN lines. M can complete 20 customer retrievals per second on the average

Three services are needed to complete this application: For M, , we get:

S(1) = transmit time (transaction input) = 20 x 10/5600 = 0.03 secs
S(2) = transmit time (transaction output) = 2000 x 10/5600 = 3 secs
S(3) = time per service = 1/20 = 0.05 secs
Total response time RT for N1 = S(1) + S(2) + S(3) = 0.03 + 3 +.05 = 3.1 secs

In this case, the bottleneck (the service where most time is consumed) is the communication line.

Although the best case estimates are a good starting point, they ignore the impact of queuing on response time calculations. Queues are formed due to two reasons: the device providing the service may be busy or it

may be locked by another activity. The first condition is an indication of workload (too many services requested) and the second condition is a result of resources being reserved (e.g., a file being updated) by one activity. In this section, our primary focus is on queuing due to workload.

.

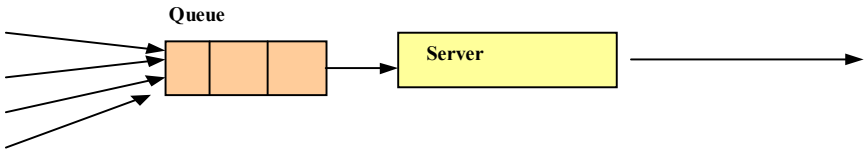


Figure 4-2: A Conceptual View of Queuing System

We need to introduce another parameter, A(i), to handle queuing. A(i) = arrival rate of requests for service i. For example, if 10 browsers send 5 queries per hour to a Web server, then A(i) = 50 per hour for the Web server.

The following formula shows utilization U(i) of a server i:

U(i) = server i utilization = A(i) * S(i) --- (3.2)

A rule of thumb used in queuing calculations is that U(i) should be kept below 0.7 to avoid queuing. The theoretical foundation for this rule of thumb is the following well known M/M/1 (Markovian arrival, Markovian service time, 1 server) formula [Kleinrock 1976]:

Queue length at server i = Q(i) = U(i)/1-U(i) ---------- (3-3)

Where Q(i) shows the number of customers in the system, including the one being served. Thus Q(i)=1, if U(i)=0.5; Q(i) reaches infinity if U(i)=1. Table 4-1 shows the impact of utilization on system performance. The key point is that U must be kept below 0.5 to keep queuing low and thus run system at optimal performance. If U is too high, then you could do the following to improve performance:

- Reduce service time by using faster servers
- Reduce arrival rate A by adding more servers

Table 4-1: Impact of Utilization on Queue Length and System Performance

Utilization U	Queue Length Q=U/1-U	Impact on System Performance
0.1	0.1	No queuing – system should performing optimally
0.5	1	On average one customer in queue, including one being served – some delays expected
0.7	2.3	On average two customers. Response time may double
0.8	4	Response time could be four times causing serious performance problems
0.9	9	Response time could be nine times causing disastrous performance problems
1.0	Infinite	Forget it

The basic assumptions of the M/M/1 queuing formula are:

- Arrivals at the server are independent of each other.
- Service times are independent of each other.

It is not necessary for the users to know that these two assumptions are based on stochastic processes and queuing theory. For example, these arrival and service time patterns are called Poisson and Exponential distributions, respectively, in stochastic processes. Poisson arrival rates and Exponential server times are referred to as Markovian behavior in queuing systems.

The net effect of queuing is that the service time increases due to queuing. The service time S(i) is replaced with S'(i):

$$S'(i) = \text{service time at server i after queuing} = S(i) + S(i)*Q(i) \quad \text{---------- (3-4)}$$

The example shown in Example 3-2 illustrates the impact of server utilization on queue length and response time. This example essentially reworks the example shown in Example 3-1 by adding queuing calculations.

So far we have focused on queuing for a single server. In most practical situations, a queuing network is formed where output of one server becomes an input to another server. Jackson's Theorem [Kleinrock 1976], shows that the following very useful results apply as long as Poisson arrival and Exponential server assumptions hold at each server in the queuing network.

- Each server can be treated independently
- Arrival rate at a server is the sum of all arrivals from all sources

Jackson's theorem is used heavily because it applies to stochastic routing also, that is, if a message is routed to server i with probability p(i). In fact, even when the Poisson arrival and Exponential service time assumptions are not satisfied, this formula is used because there is no other straightforward method for the analytical calculations of response times in queuing networks. Example 3-3 gives another example to illustrate queuing calculations in a network.

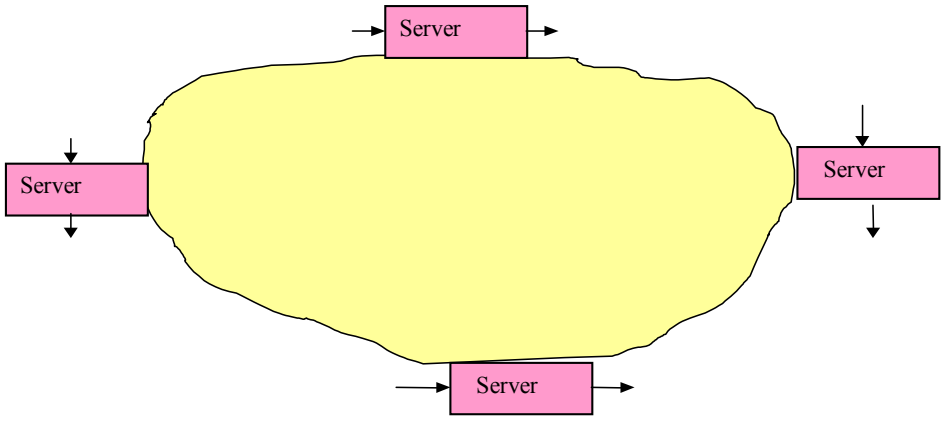


Figure 4-3: A Queuing Network System

Response Time Estimation: The Key Points

- Response time can be roughly estimated in terms of three easily observable parameters :
  S = Service time of a device
  A = Arrival rate at the device
  N = the number of devices in the network
- Utilization U, simply given by A*S, is a good indicator of server congestion. U should be kept less than

70 % to avoid excessive queuing.

- In a network of devices, the device with the largest U is the bottleneck.
- U can be reduced by decreasing the arrival rate A, decreasing the service time S, or both. For example, in the customer file example described in Tables 2.5 and 2.6, A can be reduced by putting the customer file at more than one computer and S can be reduced by purchasing a faster disk.

The following procedure may be utilized to estimate the performance of an application component allocation (see the sidebar "Response Time Estimation: The Key Points"):

- Perform best case analysis by ignoring queuing. In this case, only S and N are needed for computations (see equation (3-1)). These calculations can be used as a starting point.
- Determine if performance constraints are satisfied. If not, then there is no need for queuing analysis because if a system does not satisfy the best case calculations, then it will not satisfy conditions with workload.
- Study the effect of queuing and workload by estimating arrival rates A. The estimate A may be at peak time or average time. Estimate total time including queuing by using the equations (3-2) through (3-4).
- Try to reduce U to less than 0.7 for most devices in the network. In addition, determine the bottleneck device (devices with largest U). Try to reduce U by decreasing A and S.For example, if U of a database server is too high, then the following steps can be taken:
- Reduce arrival rates A by adding another server that may contain the entire database or most queried data items.
- Reduce service time S by getting a faster machine or by eliminating other work being done on the server.
- If detailed analysis of a configuration is needed, then you may need to simulate.

The following example illustrates this procedure.

Example 2 Impact of Queuing On Response Time

Consider the same customer database example in Example1. .We want to consider the impact of queuing on thw Web server residing on computer M. We assume that 30 clients will use this application and each client will generate the customer retrieval message 22 times per minute. This gives us arrival rate A(3) at the server machine = 30x22/60 = 11 per second.

Let us compute new service time at M. The following table shows the utilization, the average queue length, the average wait time and the total service time at the Web server for two workloads: 11 per second, 15 per second and 20 per second We have dropped the index in this table for simplicity.

Workload	Arrival Rate A at server	Service Time S	Utilization U= A*S	Queue Length Q=U/1-U	WaitTime = S*Q	New (Total) Service Time S'=S+S*Q
Workload 1	11 Per Sec	0.05 Sec	0.55	1.2	0.06 Sec	0.11 Sec
Workload 2	15 Per Sec	0.05 Sec	0.75	4	2 Sec	2.05 Sec
Workload 3	20 per second	0.05 Sec	1	infinite	infinite	infinite

It can be seen that increasing the arrival rate (workload) increased the queue length dramatically and thus caused serious performance problems. This is a common occurrence in Web performance – you are doing fine but all of a sudden one user issues many requests (increases A on the Web server) and everything halts.

To keep reasonable performance, the utilization U must be reduced to less than 0.5. In other words, A*S < 0.5. To keep U under control, you have basically two choices:

- Reduce the service time S, i.e., put the Web server on a faster machine, and/or

- Decrease the arrival rate A by splitting the traffic, i.e., install two Web servers on two different machines thus cutting the traffic to half. :

4.3.2 Performance Measurements

The previous discussion suggests that the key things to measure for Web performance are:

- Arrival rate A of the Web services that arrive at the Web server
- Service time S for each service at the Web server

These measurements, as we will see, do require some effort. The motivation for these measurements that determine the depth of measurements vary with the players. For example, the content creators are interested to know how many people are accessing their content (they want to know A for their content), Web hosters want to know the level of service they are providing (they want to know A and S for the hosting sites), network operators want to know the workload on the network pipes for the purpose of capacity planning (they want to know A and S on the network pipe), and Web/network researchers want to buid analytic/simulation models for performance predictions. The sources of measurement are:

- Web server logs. Each log entry in a Web server log corresponds to an HTTP request handled by the server, including information about the requesting clients, the request time, and the request and response messages. Most of the server logs are coarse grain (time is coarse in seconds while a service may take less than a second). Thus it is difficult to determine the service time S. It is also difficult to assign service request to users because of the use of proxies and dynamic address allocations. A Common Log Format (CLF), although not based on a formal standard is followed by most Web server vendors (see Table 4-2).
- Proxy logs that are similar to the server logs. They also follow the Common Log Format.
- Client logs can be turned on at each Web client site. These logs can be very detailed and can have much more information than the server logs. There is no common log format for Web client logs.
- TCP/IP logs. These logs are kept by routers. These logs keep track of routing information and can be very very detailed.

There are several issues with the current logs. For example, the granularity is too coarse in some cases (e.g. time in seconds in server logs). In addition, the data you need may be absent (e.g., no service time recorded in server logs). Moreover, encrypted data is difficult to log. There is always a tradeoff between details of the logs versus performance of the system (more detailed log keeping can degrade the performance of the system). Due to these limitations, you may need to capture your own data and may have to keep your own logs.

Despite a great deal of activity, no formal standards dictate the log formats. The Common Log Format (CLF) shown in Table 4-2 is used in servers and proxies. Although the seven filed shown are commonly recorded, the format may differ between implementations.

The main challenge lies in processing the various logs and drawing useful inferences from them. For example, how can you connect the client logs, the server logs, and the network logs to build a complete behavior model of the system. Many research efforts, mostly at universities have been pursuing this goal (see, for example, [Arlett97, Duska97]).

Table 4-2: Common Log Format (CLF) (Source: [Krishnamurthy 2001])

Field	Meaning
Remote Host	Hostname or IP address of a requesting client
Remote identity	Account associated with connection on client machine
Authenticated user	Name provided by user for authentication
Time	Date/time associated with the request
Request	Request method, request URI, and protocol version
Response code	Three-digit HTTP response code
Content length	Number of bytes associated with the response

4.3.3 Role of HTTP and TCP in Web Performance

4.3.3.1 Compression and Performance

Compression can have a major impact on the performance of HTTP and Web. In particular, compression reduces the size of the message being sent and thus reduce the service time S in the aforementioned model. Consider, for example, a dialup line that uses PPP (Point-to-Point Protocol). These lines are filled up with data and reducing the number of bytes transmitted quickly leads to higher performance. Various performance studies have shown the effect of compression on the performance of HTTP 1.1. Examples are the impact of case sensitivity on some compression algorithms, the effect of compression on a LAN and how it may interact with TCP slow start and delayed ACK algorithms, and the effect of HTML compression on PPP modem lines.

4.3.3.2 HTTP and TCP Interactions

Most of performance work on HTTP has been devoted to improving the interactions between HTTP and TCP. Although HTTP does not depend on TCP, almost all implementations of HTTP use TCP. Since TCP and HTTP were designed for different audiences in mind, there are several performance implications of TCP/HTTP interactions. For example, the earlier applications of TCP such as FTP differ from HTTP because HTTP is designed for shorter and interactive requests and responses.

The main consideration is that TCP uses timers heavily to trigger operations such as retransmission of lost packets, i.e., TCP sets a timer for transmission and resends packets if the timer expires. This is not desirable for HTTP because the browser can wait for a while before a needed file from the Web server is read. This wait can cause TCP timer to expire unnecessarily resulting in resends that can slow down the response to users. A larger timer parameter may be used to overcome this problem.

In addition, Web browsers can establish multiple TCP connections to a server for simultaneous downloads of embedded images. Proxies can also have multiple TCP connections to represent multiple browsers. Handling multiple connections between Web clients and servers causes some problems for other users of TCP (i.e., some Telnet or FTP users may not be able to connect because many TCP sessions are being occupied by a few HTTP clients). Web servers also have to consider the implications of managing multiple connections. To handle large no of requests for Web servers, the servers may reject some requests and/or close idle connections.

One parameter that has been discussed at great length is whether ***Nagle's algorithm*** affects HTTP performance in a negative way. The Nagle algorithm, named after its inventor John Nagle at Ford Aerospace and Communications Corporation, is used to automatically concatenate a number of small buffer messages into a larger packet that is suitable for transmission. This process (called nagling) decreases the number of packets that must be sent, thus increasing the network efficiency. Without nagling, an application generating data one byte at a time, could cause the network to be overloaded with packets because one byte of data originating from an application could result in the transmission of a 41 byte packet consisting of one byte of useful information and 40 bytes of header data. Nagling is used commonly at present in most TCP/IP routers. The question has been raised about the impact of nagling on HTTP traffic. The results show that in some cases, the last segment in a packet cab be delayed up to 200ms. See (http://www.w3.org/Protocols/HTTP/Performance/Nagle/) for details.

In addition, using a single TCP connection for multiple downloads instead of one TCP connection per request has a significant impact on the amount of overhead produced in Web applications. See the www.w3.org site for details.

4.3.3.3 TCP Analysis Tools

A variety of tools have been developed over the years for TCP performance. One of the most commonly used tool is tcpdump. This tool runs on Unix systems and is surrounded by numerous converters for specialized processing. For example, tcpdump2xplot is a Perl program that is included in the xplot package for converting tcpdumps to xplot format. Another program, getdata invokes tcpdumps from programs and automates the process of taking snapshots. iter.pl is a Perl program that scans the tcpdumps and extracts the detailed summary. For non-Unix environments, netmon is a Microsoft Windows utility that facilitates analysis of TCP performance analysis.

4.3.4 Workload Characterization

Workload consists of the set of all inputs a system receives over a period of time and shows what needs to be serviced (e.g., arrival rates from all users). It is important to characterize workload correctly because the system must be designed for a particular workload. The main question is: what is the real workload and how can it measured in a Web environment. The typical approach used is to develop a worst case, best case, and average case workload and then design a system around these cases. For better understanding, benchmarks are developed to represent typical profiles of users. For example, different benchmark scan be developed for customers purchasing online, for users accessing online customer support, for marketing reps accessing the corporate Web site remotely, and for business partners exchanging purchase orders.

Workload measurements are used in performance and simulation models to determine how many servers will be needed to handle the expected traffic, evaluate new servers and or proxies, and evaluate protocol efficiency. Workloads for Web mining (clickstream mining) is an active area of work in customer relationship management (CRM). ;;;;;

To summarize, evaluation of Web protocols and Web software components requires an understanding of the characteristics of Web workloads. You also need an approach to build workload models from the data that is captured in the various Web logs. We have given a brief overview of the subject matter. For an extensive discussion of this topic, see chapters 8 and 9 of [Krishnamurthy 2001].

4.4 Transmission Control Protocol (TCP) Considerations

TCP is a reliable and connection-based protocol that resides above IP. It is most widely used (all Web applications use TCP because HTTP is built on top of TCP). Although TCP is usually discussed as a protocol on top of IP, TCP is a general purpose protocol that can be used on other delivery systems. For example, TCP can directly run on top of an Ethernet LAN, dial up telephone lines, a high speed fiber optic network, a packet switching system or a slow speed serial connection. The main strength of TCP is the large number of delivery systems it can run on (this, however, requires global addressing). It should be emphasized that TCP is a protocol and not a software package. Basically, TCP defines a set of rules and message formats for reliable service which have been implemented in the TCP software packages.

TCP allows many applications on one host to communicate concurrently with other applications on another host. The concept of a ***port***: is introduced to specify an endpoint in a host. Each port is a unique address within a host. Thus the TCP traffic sender and receiver addresses are given by the host Internet address plus the port number within the host. TCP has many ports reserved for specific services while other ports are assigned dynamically to the applications. Table 4-3 shows some common port numbers in TCP. It can be seen that FTP, SMTP, Telnet, and several other applications have pre-assigned TCP port numbers.

Table 4-3: Examples of Assigned TCP port Numbers

Port No	Keyword	Description

0		Reserved
1-4		Not assigned
5	RJE	Remote job entry
7	Echo	Echo port
11	Users	No. of active users
13	Daytime	Daytime
15	Netstat	Network Status
20	FTP Data	FTP data send/receive
21	FTP	FTP session management
23	Telnet	Terminal logon/logoff
25	SMTP	Simple Mail Transport Protocol
53	Domain	Domain name server
80	HTTP	HTTP default port for Web

The two endpoints (ports) are essential in TCP because TCP is a connection oriented protocol (UDP also has ports). The applications on both sides must establish a connection between the two ports before TCP traffic can be initiated. Communication between two applications using TCP commonly involves the following steps:

- Application A at host H1 performs a "passive open" by indicating that it will accept an incoming connection. This request is sent to the operating system at H1 which assigns a port number at H1. The passive open can be specific (listen for a specific user) or unspecific (listen to any user such as print or file user).
- Application B at host H2 performs an "active open" to establish a connection with A at H1.
- The TCP software modules at H1 and H2 establish and verify the connection.
- The application A can now send and/or receive data from B.

TCP divides the application data into segments, where each segment has a sequence number and travels as an Internet datagram. To ensure reliable communication, TCP uses a positive acknowledgement with timeout algorithm. This algorithm consists of the following steps:

- The sender sends a message.
- The receiver receives a message and sends an acknowledgement.
- The sender receives the acknowledgement and sends the next message.
- If the sender does not get an acknowledgement within a specified time period, it times out and resends the message.

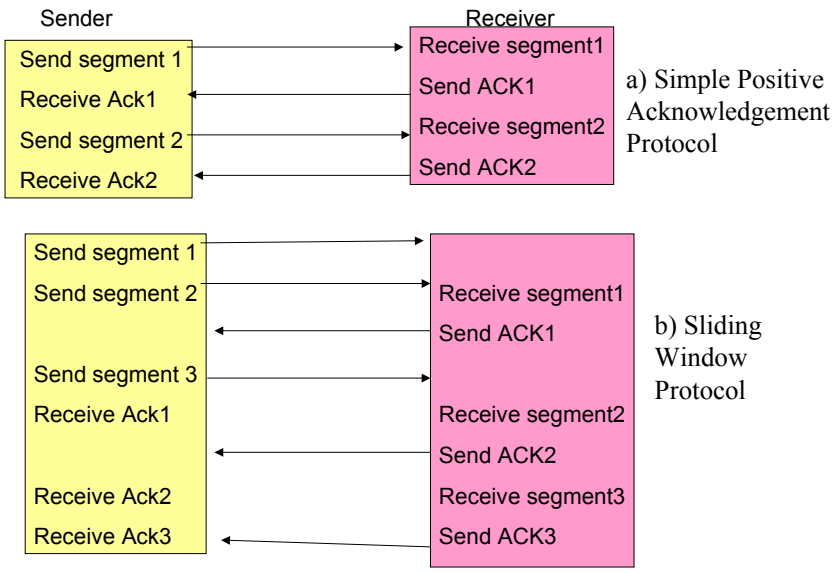


Figure 4-4: Sliding Window Versus Simple Acknowledgement

To improve efficiency of network communication, TCP uses a sliding window protocol. Simply stated, a sliding window allows the sender to keep on sending n data units (n is the sliding window size) before waiting for an acknowledgement. Figure 4-4 illustrates the difference between a positive acknowledgement (stop-and-wait) and sliding window protocol. This is a much more efficient protocol because the sender does not have to wait for acknowledgement of a message before sending the next message. In the limiting case when the window size is 1, the sliding window protocol becomes a simple positive acknowledgement algorithm.TCP uses a variable size sliding window for flow control (see appendix A). For example, if the receiver buffer is getting full, it may give the sender a small window size to minimize incoming traffic. In the limiting case, a sliding window size of zero stops incoming traffic. Adequate flow control is essential in an Internet environment because in such an environment several machines with different speeds and capacities are interconnected.

As stated previously, TCP uses segments as the basic unit of information transfer between TCP modules. Segments are exchanged to establish connection, to send data, to send acknowledgements, to send window size, and to close a connection. The format of a TCP segment is shown in Figure 4-5. The main fields of interest in the TCP segment are as follows:

- The Source Port and Destination Port show the sender and receiver application programs.
- The Sequence Number shows the starting position of data in the sender's segment and the Acknowledgement Number shows the position of the last byte received (next byte expected). These two fields are used to synchronize the positions of data sent and received.
- The Flags are used to show the type of data being sent in the segment (information data, acknowledgement, command, etc.).
- The Window parameter is used for flow control. This parameter sets the sliding window size and is used to tell the sender how much data to send.
- The Urgent indicator is used to tell TCP to deliver this segment before others.

TCP does leave several issues for implementers of TCP software. For example, an implementer can choose to use piggybacking of acknowledgements with data, use timers to retransmit unacknowledged data and send one byte at a time instead of maximum segment size, etc. Additional technical details about TCP are given in the DARPA RFC 793 by Postel. For detailed coverage, the books [Comer 2000, Tannenbaum 1996] are recommended.

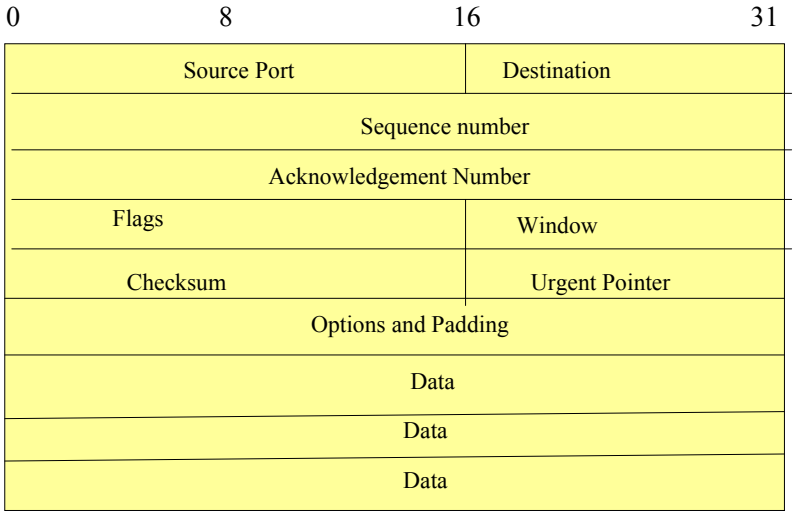


Figure 4-5: Format of a TCP Segment

4.5 Application Layer Protocols in TCP/IP

4.5.1 Overview

A rich set of higher-level (application) protocols run on top of TCP or UDP. These include, as indicated previously, the older basic protocols such as FTP and Telnet plus the newer more popular protocols such as HTTP. All higher-level protocols have some common characteristics:

- They can be standardized and shipped with the TCP/IP product. For example, the TCP/IP Suite includes application protocols such as Telnet, FTP and SMTP. These are the most widely implemented application protocols, but many others exist. Each particular TCP/IP implementation includes a set of application protocols.
- They use UDP or TCP as a transport mechanism. Recall that UDP is unreliable and offers no flow-control; so in this case, the application has to provide its own error recovery and flow-control routines. It is easier to build applications on top of TCP, a reliable, connection-oriented protocol. Most application protocols use TCP, but many applications are built on UDP especially for higher performance of connectionless services.
- Most use the client-server model of interaction in which one host acts as a client and the other as a server. The client hosts send a request over the Internet that is received and processed by the server. The tasks performed by a server can be simple or complex. For example, a time-of-day server simply returns the current time whenever it receives a client request; a file server receives requests to perform file reads/writes and returns the results.

Section **Error! Reference source not found.** (Attachment A) takes a closer look at the traditional Internet application protocols such as FTP, Telnet, and NFS. The following section highlights the end-user aspects through an example.

4.5.2 An Example of Traditional Internet Applications

Figure 4-6 shows a simplified view of an IP network. The network consists of an Ethernet IP LAN and an ATM IP WAN, interconnected through a router. A Unix minicomputer and a PC-Windows desktop are connected to the LAN and an MVS machine is connected to the WAN (there may be several other devices, but we are showing these three just to highlight key points). Each computer ("host") on this network has an IP address and also has been assigned a domain name (e.g., joe.college1.edu, sam.college1.edu,

Tom.bank3.com). This IP network is very heterogeneous (different computers, different physical networks). However, to the users of this network, it provides a set of uniform TCP/IP services (TCP/IP hides many details). Let us illustrate the use of this network.

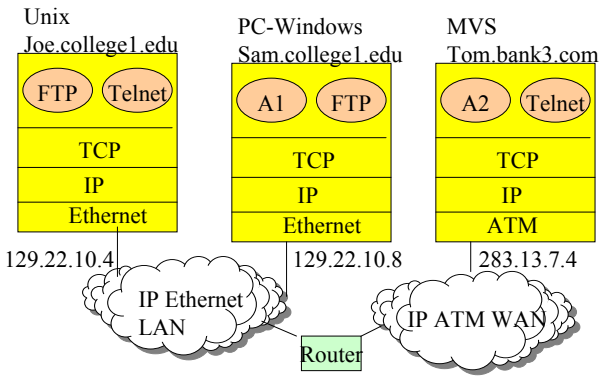


Figure 4-6: A Sample Internet Configuration

Let us assume that the Unix user needs to transfer a file from the PC ("sam.college1.edu"). The user would use the following steps (the steps are explained through comments in /* */):

```
joe> ftp sam.college1.edu       /* invoke FTP. Could have typed " ftp 129.22.10.8"*/
sam> enter logon: umar          /* prompt from sam for logon ID. umar is ID */
sam> password: xxxx             /* prompt from sam for password */
sam>  get file1  file2          /*  FTP file transfer command */
sam> quit                       /* quit  FTP */
```

Now let us assume that the Unix user needs to remotely logon to the machine "tom" to run a program "account". The user would use the following steps (the steps are explained through comments in /* */):

```
> telnet tom                    /* invoke Telnet. Could have typed " telnet 283.13.7.4".*/
tom> enter logon: umar          /* prompt from tom for logon ID. umar is ID */
tom> password: xxxx             /* prompt from tom for password */
tomr>  account          /*  run the program "account"   */
tom> quit                       /* quit  telnet */
```

A client/server application runs between the PC and the MVS machine (A1 is the client and A2 is the server). A1 and A2 use TCP/IP sockets to exchange information over the TCP/IP network. To use this application, the following steps would be needed:

start A2 on tom

start A1 on sam

use A1 (this usage will automatically send the requests to A2)

Notice that the user does not know anything about the underlying network technologies to remotely logon, transfer files and invoke client/server applications. The user only needs to know the address of the host to perform interactions.

4.5.3 Telnet (Remote Logon)

Telnet is the protocol used to provide terminal access to hosts. Telnet runs on top of TCP and provides minimal terminal support. Telnet client software usually allows the user to specify the server address as Internet address or domain names. This capability allows users to use Telnet with or without the domain naming services. By using Telnet, a user in New York can remotely logon to a computer located in London or Singapore as long as he knows the domain name or IP address of the computer.

Figure 4-7 shows a generic Telnet session. It should be emphasized that Telnet implementations vary slightly between systems. As shown in Figure 4-7, the user activates Telnet by typing "telnet" on a terminal or workstation. Telnet software prompts the user to identify a host and then goes through a logon process. After the logon process, the user workstation or terminal behaves as a terminal connected to the remote host.

Figure 4-7 also shows how Telnet actually operates. The Telnet client reads the keyboard data, passes it to the server and concurrently receives the server response and displays it on the user display unit. The Telnet server basically reads the client requests (keyboard data) and passes them to the local operating system. However, the Telnet server must have the capability to handle multiple, concurrent sessions. Usually, a main server monitors the new connections and initiates new programs to handle the new connections.

The Telnet protocol is based on three ideas: the concept of a Network Virtual Terminal (NVT), the principle of negotiated options, and a symmetric view of terminals and processes. An NVT is a virtual device, which provides the basic functionalities of a standard terminal. Each host maps its own terminal characteristics to an NVT and assumes that the other host will do the same. Telnet uses the principle of negotiated options because most hosts provide many services beyond those available with the NVT. Servers and clients negotiate the options to establish their Telnet connection (e.g., an option controls whether data passed across the connection is binary or ASCII text). The symmetry of the terminals or processes allows every host to negotiate options. To begin the negotiation, hosts verify their mutual understanding, using standard syntax. After a minimum of understanding, they can use sub-negotiation under free syntax. This allows Telnet to connect two application programs, where one acts as a client and the other as a server.

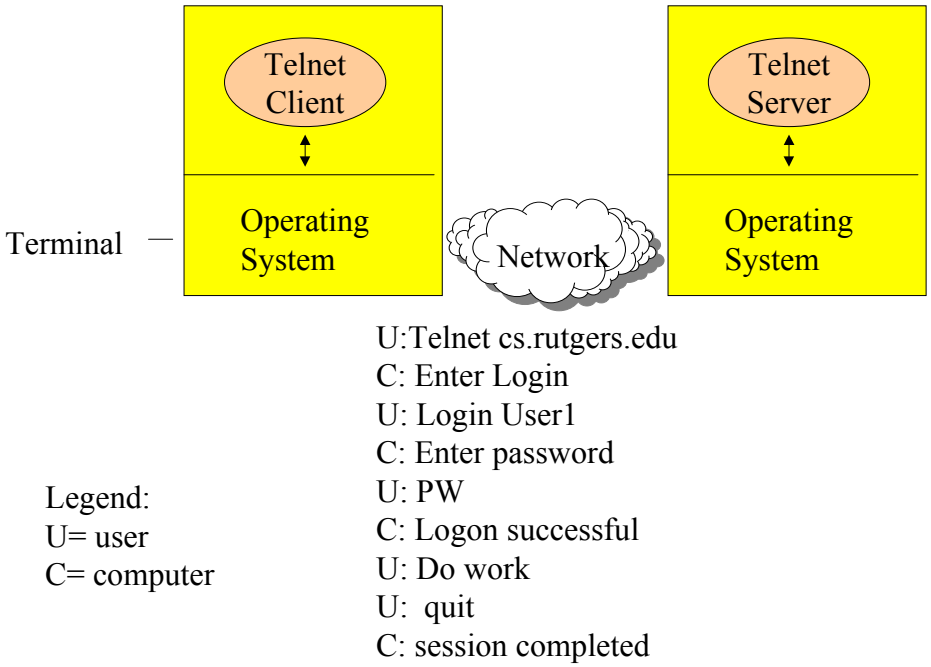


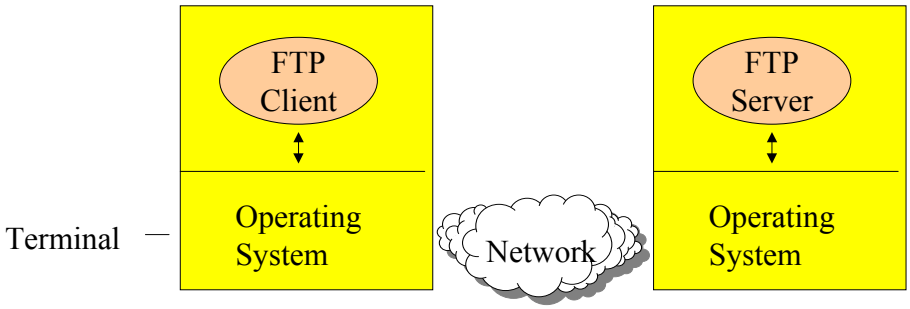
Figure 4-7: Telnet Operations

4.5.4 The File Transfer Protocol (FTP)

The File Transfer Protocol (FTP) provides a way to transfer files between hosts on the Internet. Copying files from one machine to another is one of the most frequently used operations. To support this essential function, a reliable end-to-end protocol like TCP is used. The data transfer between client and server can be in either direction. The client may send a file to the server machine, it may also request a file from this server. FTP can also be used to exchange files between programs. Many FTP implementations provide statistics on file transfer rates.

FTP allows authorized users to log into a remote machine, identify themselves, list remote directories, copy files to and from the remote system, and execute a few simple commands. FTP understands a few basic file formats and translates between different types of character codes such as ASCII and EBCDIC texts. It gives the ability to require IDs and passwords in order to access files, thereby making it practical to use with private files. To access remote files, the user must provide identification to the server which is responsible for authenticating the client before it allows the file transfer. Figure 4-8 shows a typical FTP session. It can be seen that the FTP initiation is similar to Telnet (logging on to remote host, etc.). We have used a different logon format for the purpose of illustration (FTP as well as Telnet can directly establish a session with a host in the activation command such as "Telnet Host1" and "FTP Host1"). After logon, the FTP commands of GET and PUT are used to transfer files.

The FTP link is connection-oriented; TCP/IP must be up and running on both hosts in order to establish a file transfer. FTP uses two connections: one for login to the remote host and one for managing the data transfer. The login connection employs Telnet protocol. To use the login connection, the user must have a user name and a password to access remote files and directories. The user who initiates the connection becomes a client and the remote host assumes the server function.



U:FTP cs.rutgers.edu
C: Enter Login
U: Login User1
C: Enter password
U: PW
C: Logon successful
U: put file1.sys1 file1.sys2
C: File transfer done
U: quit

Legend:
U= user
C= computer

Figure 4-8: FTP Operations

Figure 4-8 shows a typical FTP connection. The client establishes two connections with the server: one to transfer the file data and the other to pass the control information (e.g., file name). The server waits for the connection at the FTP port (a pre-assigned TCP port 21). When a connection arrives on this port, the server initiates a control process C to handle this connection and goes back to wait for the connection. The process C communicates with the client over the control connection for logon authentication and for control commands such as list remote directories and specify the file name to be transferred. After the authentication and identification of file name to be transferred, C opens another process T to actually transfer the file selected. This transfer is conducted over the other connection.

The example shown in Figure 4-8 presents a conceptual overview of FTP operations. The FTP software commonly provides a wide variety of options, as illustrated by the following sequence of operations:

- Login to Foreign Host. To execute a file transfer, the user has to start with a login operation. Login may allow change of the data representation during the transfer.
- Define a Directory. After the login is complete, the user may have to execute a change directory (CD) in order to manage space for data (e.g., CD userid).
- Define the File to be Transferred. The DIR subcommand of FTP displays the filenames within a directory. The most frequent operations performed on the files are copying a file from the remote host into the local file system, using the GET subcommand; and copying a file from the local system file to the remote host, using the PUT subcommand.
- Define Mode of Transfer. FTP subcommands allow transformation of data representation between dissimilar systems. The user can decide on: i) the way the bits will be moved from one place to another and ii) the different representations of data upon the system's architecture. Examples of the subcommands are:
  - Block mode parameter preserves the logical record boundaries of the file. Stream mode parameter, the default transfer mode, is the most efficient mode of transfer because no data block of information is transferred. Any type of data representation can be transferred in stream mode.
  - TYPE with ASCII, EBCDIC, IMAGE parameters shows the data representation. ASCII, the default transfer type, is normally used between ASCII hosts and EBCDIC is used for transferring data between hosts using this data representation. IMAGE data is sent as contiguous bits packed in 8-bit bytes. This transfer type is the most efficient for transferring and storing binary data.

- Site Command. The SITE subcommand is used with PUT to specify how data is to be stored on the remote host. The parameters of this subcommand are used by the foreign host system. Sending information to the remote system may be accomplished with the subcommands SENDSITE and SITE.
- End the Transfer Session. Issue a QUIT, which will disconnect the hosts running FTP. When issuing a CLOSE subcommand, FTP stays active and you can OPEN a new session.

4.5.5 The SUN Network File System (NFS) Protocol

This protocol, originally developed by the SUN Microsystems, has become a de facto standard for use in building distributed file systems. NFS runs on top of UDP, however a few implementations of NFS on TCP also exist. NFS provides the ability for host computers and workstations running NFS to transparently share files across the entire Internet. For example, an IBM PC could logically have files resident on a mainframe; Apollo or Sun; however the file location is totally transparent to the user. In general, an NFS user at computer C1 can access a file F in the network without knowing the location of F. The user also accesses a remote file as if it were local.

Figure 4-9 shows a sample NFS session. The user of NFS basically issues a MOUNT command to mount a remote directory from a remote host to the local host. The user can search the remote directories by using the LOOKUP command and can retrieve and delete remote files by using GET and REMOVE commands. The user can also display the attributes of remotely located files and access them as local files. An UNMOUNT makes the remotely located files unavailable to the local host.

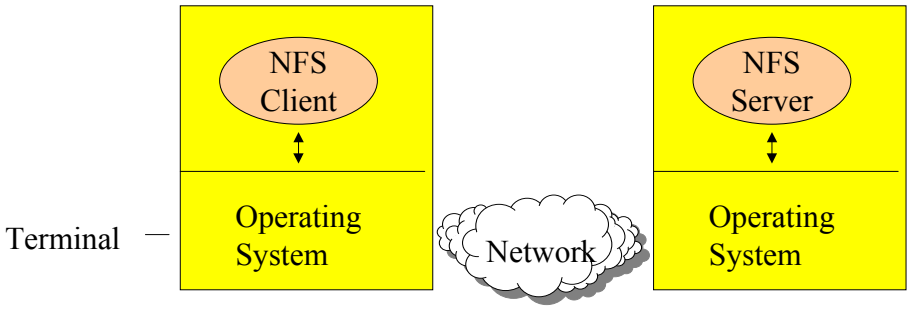
Figure 4-9 also shows the operational view of NFS. The NFS client C sends the file access commands to the NFS server. The server at host H receives requests from authorized clients at C to access the files at H as if they were local to C. The server at H must have the facilities to concurrently receive and process many client requests. The server at H also provides the facilities for file creation at H. The files at H appear as extensions to the local disk at C.

The steps in using NFS are as follows:

- Assume that the client site C has files F1, F2, F3
- Logon to remote site H from the client
- Issue a MOUNT command to make a remote directory at H available. Assume that the remote directory contains files F4, F5. The remote directory is mounted to C and can be accessed as a local directory.
- Issue reads to F1, F2, F3, F4, and F5 from C. The read requests do not have to specify the location of any of these files.

Let us review some of the NFS protocols. NFS uses two basic protocols: the MOUNT protocol and the NFS Protocol.

The MOUNT protocol specifies the remote host and the file system to be accessed. This protocol supports a MOUNT command which mounts a remote directory and returns a file handle pointing to the directory. This file handle is used later by the client to access the remote files. Some implementations may encrypt the file handle for security reasons. Other commands supported by this protocol are UNMOUNT and UNMTALL to unmount one or all remote directories, respectively.



Legend:
U= user
C= computer

U:Mount host1 dir1
C: directory dir1 mounted
U: lookup file1
C: f1 is the file handle for file1
U: showattr f1
C: attributes of file1 shown
U: unmount
C: directory unmounted

Figure 4-9; NFS Operations

The NFS protocol performs the actual file operations after it has been requested through a MOUNT command. Some of the commands supported by NFS are as follows:

- LOOKUP: searches for a file in the mounted directory and, if found, returns a file handle
- READ and WRITE: basic file I/O operations
- RENAME: rename a file
- REMOVE: delete a file
- MKDIR and RMDIR: creation/deletion of subdirectories
- GET and SET-ATTR: retrieve and set file attributes.

These commands correspond to the local file operations. If the file to be accessed is a remote file, the operating system just reroutes these calls to a remote host. This makes all file operations look alike, independent of the site where they are located.

NFS is designed to be machine, operating systems and transport protocol independent. This independence is achieved by using the Remote Procedure Call (RPC) facility on top of the UDP. NFS is currently available on most TCP/IP based mini and microcomputer systems. IBM has announced NFS support on its mainframes under MVS operating systems. This announcement has expanded the availability of NFS to all ranges of computing systems.

4.5.6 TCP/IP Berkeley Sockets

TCP/IP Berkley Sockets, simply called sockets, allow new applications and protocols to be developed on top of TCP/IP. For example, many of the Internet application layer protocols use sockets. In addition, middleware packages such as CORBA internally use sockets. Sockets are application programming interfaces (APIs) that can be used by C programs residing on two IP hosts to communicate with each other. Let us briefly review TCP/IP sockets.

A socket is an addressed endpoint of communication which conceptually resides above TCP. The addresses associated with the sockets are commonly the IP physical addresses (32 bit host number, 16 bit port number). There are several types of sockets, grouped according to the services they provide. The services include stream sockets, which provide duplex, sequenced flow of data, with no record boundaries; datagram

sockets which transfer messages of different sizes in both directions and which are not promised to be reliable and sequenced; and sequenced packet sockets which are similar to stream sockets, with the difference that record boundaries are preserved. Applications written for UNIX environments use stream (reliable connection) or datagram mode, forking which allows several processes to be initiated by one process, and/or mailboxes which allow an intermediate file for message transmission.

The sidebar "Berkeley Socket Command Summary" shows a summary of the socket commands for stream or datagram services. The commands, available as UNIX system calls, are shown generically because different versions of UNIX support different command verbs for the same activity through different facilities. It can be seen that sockets support relatively few commands. The first five commands are initialization commands. SOCKET creates a socket; BIND is used by the servers to register their well-known address to the system so that the clients can connect and transfer information; CONNECT is used by the client to establish a connection with the server after the server has issued a BIND; and LISTEN and ACCEPT are issued by the server to indicate that it is willing to accept connections from the clients and then to put the connections on a queue for later processing. Commands 6 and 7 are the main information transfer commands for connection-based (TCP) client-server systems and commands 8 and 9 are the main information transfer commands for connectionless (UDP) client-server systems.

Berkeley Socket Command Summary (basic Commands)

1. SOCKET = creates a socket and specifies socket type (TCP, UDP)
2. BIND = assigns a name to an unnamed socket (handle)
3. CONNECT = establishes a connection between local and remote server
4. LISTEN = server is willing to accept connections
5. ACCEPT = accept a connection and put it on queue
6. WRITE = send data on TCP socket
7. READ = read data from TCP socket
8. SENDTO = send data on UDP socket
9. RECVFROM = read data from UDP socket

Figure 4-10 shows the pseudo code for a TCP based socket program. Commands 1 through 4 represent the Initiation State of the server by using the SOCKET, BIND and LISTEN commands. Instruction 1 is used to create a socket. A socket number is chosen which is not already being used by the standard Internet services such as FTP and Telnet. The BIND command is used to make the address of the server available to the Internet ("register the server socket with the network") and LISTEN and ACCEPT indicate that the server is willing and waiting for a client to issue a connect. Instruction 4 processes a CONNECT instruction from the client and puts it on a queue for processing. Instructions 5 and 6 create a socket and issue a CONNECT from the client (Initiation State of the client) which is "ACCEPTed" by the server. After this, a connection between the two processes to pass messages is established. Basically, the server process specifies half of the association by issuing an ACCEPT system call and then passively listening on its socket for a CONNECT. The association is completed by the client process when it issues a CONNECT call to the server's socket. The next commands are used to send and receive information in the Information Transfer State. In this state, the data is sent and received by using the READ and WRITE commands. The CLOSE and EXIT commands constitute the Termination State of the data transfer phase of the application.

This pseudo code can be translated into programs by using C, C++, Java, or assembler compilers with proper support libraries. The main limitation with socket API is that it does not provide any primitives for command

controls such as locking, commit, roll-backs, etc. Each programmer designs his/her own control verbs, which are sent as data by Unix through the READ/WRITE and RECVFROM/SENDTO commands. This can lead to applications that are not portable across networks. In fact, most application developers do not write socket level code. Commercially available middleware packages such as CORBA, DCOM, and Sun RPC provide higher level programming support for developing distributed applications (these middleware packages internally use sockets).

A detailed description of Berkeley Sockets with numerous examples is given in the Stevens book, "Unix Network Programming, Vol 1 and 2 ", [Stevens 1997].

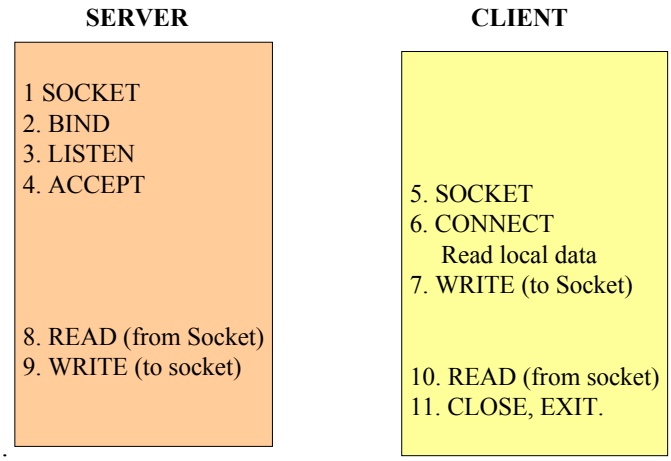


Figure 4-10: Psuedo Code of Program in Berkeley Sockets

4.5.7 Miscellaneous Application Layer Protocols

In addition to the aforementioned protocols, many other application protocols are available on TCP/IP Suite. Some of these protocols are described here briefly.

The Trivial File Transfer Protocol. The Trivial File Transfer Protocol (TFTP) also transfers files between hosts, but with less functionality. TFTP is much less complex to implement than FTP, it runs on top of UDP, and it avoids the complexities of TCP. TFTP does not provide any authorization mechanism, so it typically can only be used with publicly available files.

The Simple Mail Transfer Protocol. The Simple Mail Transfer Protocol (SMTP) is the Internet electronic mail exchange mechanism. This TCP based protocol provides a way for dissimilar hosts to exchange mail. It is responsible for transporting mail and is not concerned with mail format. An SMTP implementation is usually built beneath the native mail system in the host. SMTP manages the mail connection such as establishing the sender's credentials and ensures the recipient mailbox. It provides forwarding and deferred delivery and supports a variety of message formats and native mail systems.

The Domain Name Services. The Domain Naming service can be used to assign symbolic names to the Internet addresses. Symbolic names are preferable to Internet addresses because they are easier for humans to work with, and because an address may change if a host is moved or a network is reorganized. The Domain Naming system defines a hierarchical naming structure. At the highest level is the organization type (EDU for educational, COM for commercial, GOV for governmental, etc.). Below that is the organization name. Further levels may be needed in large organizations. For example, EE.MIT.EDU identifies an educational computer that resides in the EE department of MIT. The mapping between a domain name and an Internet address is performed by name server machines in each domain. For example, one name server at MIT may contain the mapping between domain names and IP addresses in the MIT domain. This avoids proliferation of independent address tables in every host. A protocol, run on top of UDP, is used to query the

name server, to determine the IP address for a particular domain name. The name server may in turn ask another name server if it does not know. This distribution of names means that each server need not know all of the hosts in an Internet. In addition, if a host is assigned a new IP address, only the name server local to that host needs to be updated. The change will then be available to other name servers and hosts as needed.

The Simple Network Management Protocol (SNMP). This network management protocol reflects the DARPA/OSI short range approach to network management. The focus of this protocol is on the network monitoring and control functions of networks. The OSI based network management standard CMIP (Common Management Information Protocol) includes many other network management functions such as network performance, accounting and security. SNMP uses polling where the manager repeatedly polls the components about their status. SNMP is based on a series of RFCs (e.g., RFC 1067) and is available from a substantial number of vendors. Due to its simplicity and vendor support, SNMP has furthered the popularity of TCP/IP in managing heterogeneous networks. In addition, it is being considered as a transition path to CMIP by using the following scenario: 1) install SNMP over TCP/IP for network management, 2) convert to CMOT (CMIS over TCP/IP) by changing the application layer, and 3) convert to CMIP by replacing the underlying TCP/IP with OSI.

4.6 XML Processing

4.6.1 Highlights of XML

XML is a subset of SGML (Standard Generalized Markup Language) developed by World Wide Web Consortium. It is a data description Meta language and its main purpose is to provide platform independent data exchange format. Unlike HTML, it does not have its own fixed vocabulary. It is a Meta language and it can be used to define markup languages specific to the needs of users e.g., WML (wireless Markup Language), MathML, chemML, FpML etc. -- all XML variants.

The following is an XML example of a customer:

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE customer SYSTEM "customer.dtd">
<customers>
<customer cid= "12">                    <name>
                            <first>Joe</first>
                            <last>Blatt</last>
                    </name>
<address>
                            <street>731 Oak St</street>
                            <city>Philadelphia</city>
                            <state>PA</state>
</address></customer>
<customer cid= "12"><name>
                            <first>Jim</first>
                            <last>Zokar</last>
</name><address>
                            <street>221 Market st</street>
                            <city>Philadelphia</city>
                            <state>PA</state></ddress> </customer>
```

```
</customers>
```

Figure 4-11:XML data

XML documents can be data-centric or document centric. Document centric XML documents contain irregular structure and larger grained data like letters, memos etc while data centric means that the document contains discrete data such as customer record, order, invoice, etc. Data in XML is delimited by tags for identification (see Figure 4-11*)*. We will concentrate on data centric documents in this discussion.

4.6.2 XML Versus Other Description Languages

4.6.2.1 Overview

In a very short time period, XML has gained great popularity as a description language and is being used in many areas. Here are some examples of where XML is being used:

- E-Commerce/eBusiness. XML is being used in e-commerce for representing the purchase orders, invoices etc. A standard called ebXML is being developed for using XML in eBusiness.
- Third generation distributed applications. XML Web Services, also known as Web Services, use XML to communicate between remotely located application components. XML Web Services are at the core of Microsoft's Dot Net and Sun's J2EE – the two platforms positioned for 3G distributed computing.
- Web publishing. XML is being used to publish Web content (company information, products, etc).
- Online banking. XML is being used to represent transactions for online banking. An example is Open Financial Exchange (OFX) specification. XML documents in OFX show account information, monthly payments, bank statements etc.
- Scientific publishing. XML is the foundation of several markup languages for scientific publishing. Examples are the Mathematical Markup Language (developed by W3C) and Chemical Markup Language.
- Airline Reservation Systems. XML is being used in the airline industry to represent flight information, arrival times, departure times, etc.

Many groups are defining new formats for information interchange in XML. The number of XML applications is growing rapidly. We discuss two topics; XML versus EDI and XML versus HTML.

4.6.2.2 XML vs EDI (for e-commerce)

Data exchange between disparate systems especially without human intervention has always been a problem. The technologists had been trying for a long time for a technique that could be used to exchange information seamlessly. One attempt was EDI (Electronic Data Interchange). EDI was developed primarily to interchange data between computer applications of the trading partners (e.g., B2B ecommerce) using an agreed upon standard to structure. EDI was developed before the advent of Web and was basically developed for private networks. Although EDI proved quite effective in automating secure business processes but there were some pitfalls in the technology that needed to be addressed and were addressed in XML. The comparison of XML and EDI technologies for ecommerce is given in the following table:

EDI	XML
Requires costly program development	Requires only style sheets and scripts
Requires dedicated server that costs a lot.	Can be used on Web server for ecommerce through www
Used in private networks. cost per message is much more than XML based e-commerce	Uses internet connection
Emphasis is on size of the messages. Compressed messages are used	Emphasis is on ease in data transmission and data application development

EDI does not cover advertising. It can only be used for certain business processes like purchase orders, invoices etc.	XML can be used for product publishing and advertising

4.6.2.3 XML vs. HTML

HTML has been the Web markup language from beginning but it has certain limitations. HTML primarily addresses the data presentation in the Web browser. Although HTML was introduced for Web applications but the main advantage of XML over HTML is that XML concentrates on data description and data is kept separate from the presentation formats and business logic.

HTML has fixed tags while XML has no particular tags of its own. It can be used to define data tags relevant to the interests of a particular group of users.

Selected XML References

Books

- Williams,Kevin, "Professional XML Databases", Wrox press ltd, December 2000
- Hunter, David, "Beginning XML", Wrox press ltd., August 2000

Websites

- **www.w3c.org** (official site of world wide Web consortium)
- **www.oracle.org** (Oracle Website)
- **www.tamino.com** (native XML Database)
- **www.xml.com**

Papers

- Schoning Harald Dr., "Tamino – a DBMS designed for XML", IEEE 2001
- Zisman A., "An overview of XML", computing & control Engineering Journal", August 2000
- Roy Jaideep etal, "XML Schema Language: taking XML to the Next Level", IT Pro, IEEE 2001
- Bourret Ronald, "XML Databases", http://www.rpbourret.com/xml/XMLAndDatabases.htm, 2000
- Bourret Ronald, "XML Database Products", http://www.rpbourret.com/xml/XMLDatabaseProds.htm, 2000

4.7 XML Technologies

XML standards and technology cannot be considered without its associated technologies shown below. A quick scan of these technologies ("XML Family") is presented as a refresher. :

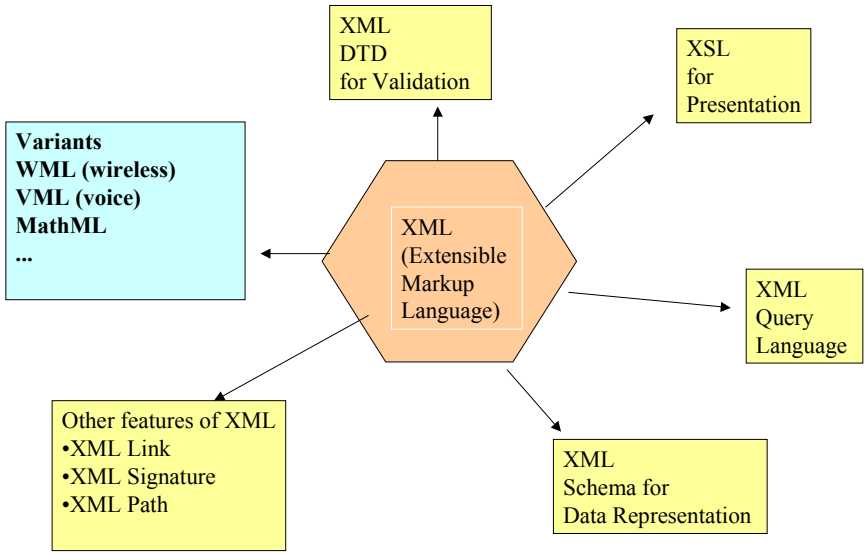


Figure 4-12: XML Family

4.7.1 DTDs and XML Schema

DTDs as mentioned above define data vocabulary that can be used in the XML documents. Through DTDs, we can define elements, sub-elements, and attributes of elements, allowed cardinality for attributes and sub-elements, whether a particular data can be null or not etc. *it helps control the authenticity of xml documents in a particular environment.*

Normally, companies or parties who want to communicate with each other for a particular purpose agree upon a common DTD. But it is also possible to communicate with slightly different DTDs and transformation of one DTD to another is possible through XSLT (extensible style language transformation)

DTDs are quite useful but they have following limitations i.e.,

- Data lengths cannot be specified
- Data types are not supported.
- Only document structure is validated; actual data can't be validated through DTD

W3C has developed a new standard -- XML Schema -- to provide a better way to validate XML documents. Unlike DTD, XML Schema file is an XML derivative.

XML schemas have the following additional capabilities that make them a better choice:

- Can declare data types. Data types supported by schemas are quite similar to the primitive data types of high level programming languages.
- Customized simple and complex data types can be defined and referred to.
- Attribute constraints like maximum length, formatting pattern etc can be defined.

4.7.2 XSL(eXtensible Style Language)

XML separates content from presentation. Once authored, an XML document can be viewed in any desired format by using suitable XSL (eXtensible Style Language). XSLT is a subset of XSL, a transformation language that converts one XML document to another e.g., an XML document can be converted to WML, HTML etc. HTML is not the same as XML but a transformation can be done as long as a document can be modeled as an XML tree.

It is beyond the scope of this tutorial to describe XSL. The following two links contain XSL tutorials:

- http://www.xml101.com/xsl/xsl_server.asp
- http://nwalsh.com/docs/tutorials/xsl/

4.7.3 XML Query Languages

Many XML query languages have been and are being defined. the purpose is to treat XML document not just as an intermediate data format but also as a database. Xml Query languages are the same to XML as SQL is to databases. Some examples are Quilt, XQL, XML Query Language and now W3C is concentrating on XML Query.

4.7.4 XLink, XPointer

The XML Linking Working Group is defining a standard way to represent links between XML resources. In addition to simple links, like HTML's <A> tag, XML has mechanisms for links between multiple resources and links between read-only resources. X-Pointer describes how to address a resource and X-Link describes how to associate two or more resources. These two methods can be used to define links between objects: "external" links to locate an XML document and "internal" links to locations within XML documents.

4.7.5 How XML Works: XML Data Validation

Figure 4-13 shows a high level view of XML processing. Broadly speaking, an XML document goes through a document processor that performs three distinct functions: the document is first received, then parsed/validated by invoking an XML parser/validator, and then passed on to the next program. Consider, for example, the situation when you want to display an XML document on your browser, say the Internet Explorer (IE). The browser, in this case is the document processor that receives the XML document, invokes a parser imbedded in IE, and then passes the validated document to a program that converts XML to HTML for display. The conversion from XML to HTML is accomplished through an XSL.

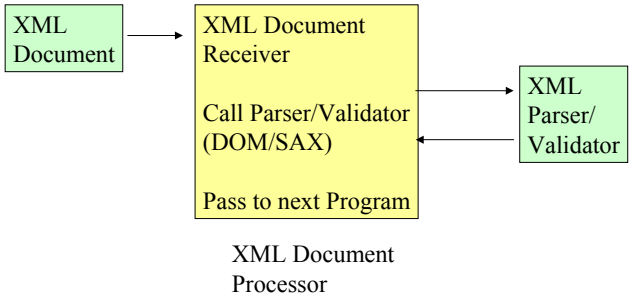


Figure 4-13: XML Document Processing – Another View

Let us look at this processing in a bit more detail. XML parser (a software program) is used to validate XML documents and is a key player XML processing. If a document is structured according to the basic rules as defined by the XML standard, it is called ***well-formed*** document. A ***valid*** document is a well-formed document that also conforms to the rules specified in the data definition document (DTD) or Schema. A parser determines if an XML document is well formed and valid. A wide range of XML parsers are commercially available at the time of this writing. Figure 4-14 shows how XML parsing and validation works.

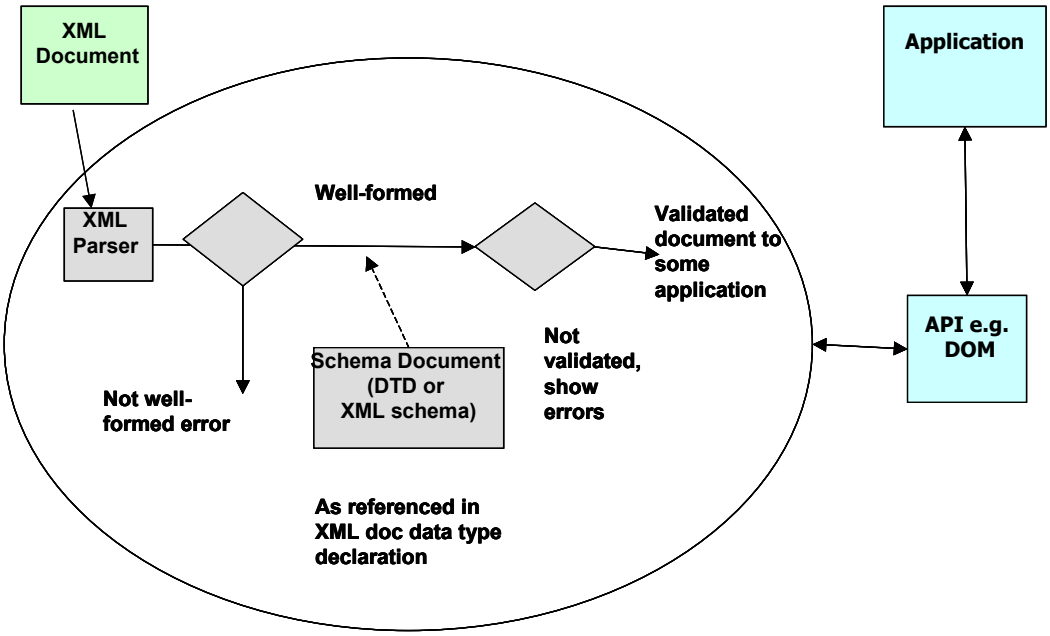


Figure 4-14: How XML Processing Works

Let us explain this processing by considering the following XML description.

```
<?xml version="1.0" standalone="no"?>
< -- customer example -- >
<customer>
            <name>
                  <first>Joe</first>
                  <last>Blatt</last>
            </name>
            <address>
                  <street>721 Oak street</street>
                  <city>Philadelphia</city><state>PA</state>
            </address>
</customer>
```

The first statement in XML starts with "<?xml" to indicate an XML document. The XML document consists of elements and sub-elements that define various tags. Each element starts with a <tag> and ends with a </tag>. The above example describes the customer element with sub-elements name and address. The sub-elements of name are first and last, etc. Comments in XML begin with <! -- And end with -->. XML allows definition of optional fields, repeating fields, etc. If a parser does not see this syntax (<tag> ended by </tag>) then you get a parsing error otherwise the document is "well formed". A well-formed document is then processed against a DTD or Schema and then validated.

4.8 Client Versus Server Side XML Processing and XSL

4.8.1 Overview

Suppose you want to display an XML document that contains a customer record on a Web browser or a cellular phone. How can this be done? To present the XML information for viewing, you need to apply an XSL (XML Stylesheet language) transformation to the XML document. As stated previously, XSL can be used to transform XML into a wide range of displayable and other formats. The application of XSL transformations can be done in two ways:

- Client-Side transformation: both the data XML and display XSL are downloaded from the web server, and then the XSL transformation is applied at the client side (the web browser).
- Server-Side Transformation: The XSL transformation is applied at the web server side, then the transformed data (e.g. in HTML format, or other formats) is sent to the browser.

XSL plays a key role in this transformation. As stated previously, the following two links contain XSL tutorials:

- http://www.xml101.com/xsl/xsl_server.asp
- http://nwalsh.com/docs/tutorials/xsl/

Naturally, there are advantages and disadvantages of client-side versus server-side processing for XML.

4.8.2 Client Side Processing:

Client-Side Transformation can be done through CSS or XSL. The Cascading Style Sheets (CSS), is an older technology to describe the rendering of particular XML elements in a web page. XSL is a newer technology. To render with XSL, you need to prepare an XSL style sheet to render the XML data file to a more sophisticated HTML format. . XSL is much more powerful than CSS – it supports sorting, IF-conditions, and other processing functions.

The advantages of client side processing are:

- Very simple because most browsers (IE v5+) support XML client side processing
- No need to install a particular transformation engine running on the web server
- Server has lighter work because the processing is done on client side

The main disadvantages of client-side processing are:

- Different browsers have different level of XML support (e.g., Netscape versus IE). Some browsers do not support XML and XSL (non-XSL aware browser). And even when you use JavaScript to do the transform, not all browsers support an XML parser which JavaScript needs to use to transform XML.
- Security issues because cannot hide the XML document -- the client is able to see the whole XML document.
- Download sizes -- you have to send a big XML document while the user only wants to see a small part of it.
- No performance benefit if XML is dynamically-generated

4.8.3 Server Side Processing

XML is converted to HTML or other formats on the server side and then sent to the browser. Server-side transformation is supported by many Web servers. For example, Tomcat Web Server is an open source web server that supports Java Servlets (see http://java.sun.com/products/servlet/index.html). In addition, Apache

Cocoon is an open source server-side XSL transformation engine from the Apache organization. Cocoon does not function as a standalone server and runs on top of Apache or Tomcat.

The advantages of server side processing are:

- Applicable to all browsers. Since not all browsers support XML and XSL, you can transform the XML document on the server and send it as pure HTML to the browser.
- More efficient. It is better to generate the transformed data in different formats by writing only one XSL file on the server
- More secure. It is more secure to generate the transformed data at server side than at the client side because you can hide sensitive fields and keep them on the server.

The server side disadvantages are:

- Server side processing is more complex because you have to decide the web server and the transformation engine used by the server. The transformation engine (e.g., Cocoon) has to be installed.
- Server has heavier work load because all transformations are done on the server.

4.9 Handling XML Data by Using Databases

XML data can be handled at present in two different ways:

- Use relational databases (RDBs) to store XML data
- Use Native XML databases

These two options are discussed in this section.

4.9.1 Using Relational Databases -Mapping between XML and Relational Databases

4.9.1.1 Representation of relational data in XML - Concepts

Storing XML data in RDB requires translation between RDB and XML formats. XML represents data in hierarchical tree like form whereas data in databases is stored in relations arranged in columns or fields. The question is how can we map data from one form to the other. Following are the main approaches:

Tables. We can represent an RDB table as an XML 'Element'. Elements can be empty elements that contain only attributes and no sub-elements or text value (PCDATA or CDATA), elements containing sub-elements only, containing text value only or mixed elements that can contain both text value of their own and sub-elements at the same time. *(See 'Beginning XML' and 'XML databases' by WROX press for details)* It is better to use elements containing sub-elements where sub-elements represent attributes (e.g., see the following code):

```
<A>
<B>"ac"</B>
<C>"w"</C>
</A>
```

'A' representing a table and 'B' and 'C' represents attributes). Another way is to use 'empty element' to represent a table (e.g., <A B="ac", C="w"/>).

Columns or Fields. This aspect has been explained while discussing representation of tables in XML i.e., columns can be expressed in terms of XML attributes or text only elements. In the examples given above, B and C are column/ field names and 'ac' and 'w' represents their value for a single tuple.

Primary Keys. One way to represent a primary key of RDB in XML is to define ID attributes. ID attributes in XML uniquely identify the contents of an element but the drawback is that there cannot be more than one id attribute for an element.

Relationships. Relationships can be expressed as relationships between ID and IDREF/s attributes or through containment i.e., treating one table as a sub-element of an element For example, there is one to many relationship between Table A and B. Suppose one record (shown below) of A corresponds to two records of B. B can be a sub-element of A or a separate element.

```
Case1) using containment                    case 2) using ID / IDREFs
-------------------------------             ----------------------------
<A (set of attributes for A)>               <A idattrib ="1"/>
<B (set of attrib)>                         <B idrefattrib = "1"/>
<B(set of attrib)>                          <B idrefattrib ="1"/>
</A>
```

Both cases have certain limitations e.g., in case 1, we cannot specify a B record for which no record of A exists. In 2nd case, although both the tables have been structured separately but it is difficult to navigate from the referenced table i.e., we can't get corresponding records of B for a given record of A but we can get A from B.

Data types. Data types can be specified in XML Schema This is not supported in DTDs. Almost all the primitive data types supported by most of the high level programming languages is supported. Data types as defined by W3C for XML schema are mentioned on w3c Web page i.e., http://www.w3.org/TR/xmlschema-0/#SimpleTypeFacets

Default Value. Default value can be provided in DTD and XML schema. And the parser provides this value if that particular attribute has not been not mentioned in the XML document.

Unique Constraint. If we are representing attributes as XML attributes, the only way to impose unique constraint is to specify the said attribute as ID but the drawback is that we can't have more than one ID attributes for an Element. But if we specify attributes as sub-elements, we can impose unique constraint on individual attributes and not for a group of attributes.

Not Null Constraint. Not null constraint can be mentioned in a DTD by adding '#Required' clause in attribute definition e.g., as described in section 2 XML example, 'id' attribute for customer element is defined as required.

<!ATTLIST **customer** id CDATA #REQUIRED>

Attribute can be defined as optional by using '#IMPLIED' clause.

4.9.1.2 EXAMPLE: Sample Representation of relational Data into XML

Consider the following simple relational schema:

Department

Dept_id	Dept_name

Instructor

Inst_first_name	Inst_last_name	Inst_mid_name	e_mail	Dept_id

Course

Crs_num	Crs_name	Dept_id	credits

This relational schema can be represented in DTD as shown below. . The DTD shows use of ID & IDREF

```
<!ELEMENT COURSE EMPTY>
<!ATTLIST COURSE
crs_num CDATA  ID #REQUIRED
crs_name CDATA #IMPLIED
credits CDATA   #IMPLIED
dptIDREF IDREF #REQUIRED>
<!ELEMENT DEPARTMENT >
<!ATTLIST DEPARTMENT
dept_id ID #REQUIRED
dept_name #implied>
<!ELEMENT INSTRUCTOR
<!ATTLIST INSTRUCTOR
INST_ID ID #REQUIRED
inst_first_name #REQUIRED
inst_last_name #REQUIRED
inst_mid_name #IMPLIED
e_mail #IMPLIED
deplDREF IDREF #IMPLIED>
```

In the above DTD, we can see that we had to introduce a new attribute INST_ID as the ID (identifying attribute for Instructor Element since the primary key was a composite key and any element in XML can hold only a single identifying attribute)

APIs for XML (For translation from XML to RDBMS and vice versa). Several APIs are available for processing XML documents. Some are standard APIs like DOM (developed by W3C) and some are vendor specific APIs that the vendors have built in their products to make things easier. These APIs can be used to translate XML data to RDBMS and vice versa. We will discuss these APIs as part of the XML Native databases in Section 4.9.2.

4.9.1.3 XML-enabled Database Products .

Some RDBMS claim to provide XML support e.g., SQL SERVER 2000 and ORACLE 8i/9i etc. In this discussion, we will see what kind of XML support these vendors are providing:

a) SQL Server 2000. (www.microsoft.com)

Microsoft SQL Server 2000 supports XML in following ways:

- **FOR XML clause in SELECT statements** The FOR XML **clause** has three options i.e., RAW, AUTO, EXPLICIT, which specify how the SELECT statement is mapped to XML. Explicit option requires additional information to be specified e.g., in what way the fields in the resulting table be shown i.e., they are elements, attributes, Ids or IDREF etc. Addition of XMLDATA clause will also

develop XML Schema in the output. This clause was used in the hands on experiments with SQL Server 2000. I used evaluation edition of SQL SERVER2000 and sample Northwind database for experimenting with the SQL SERVER functions. See Section 6 for Experiments conducted

- **OpenXML. Function** . The OpenXML function uses a table-based mapping to extract part of an XML document as a table and use it. In most places a table name can be used, such as the FROM clause of a SELECT statement.
- **Updategrams**. Inserts, updates, and deletes are done through specially formatted XML documents called "updategrams". These contain the before and after data (both in the case of update, only after data in the case of insert, and only before data in the case of delete). By default, updategrams use table-based mappings. They can use object-relational mappings by specifying an annotated schema.

4.9.1.4 Oracle 8i, 9i (www.oracle.com).

Oracle 8i can store XML documents in three different ways: in the Internet File System (iFS), using the XML SQL Utility for Java, and as a BLOB that can be searched using the Oracle Intermedia XML Search. Oracle 8i also includes a number of other XML-related tools, the most interesting of which is the XML Class Generator, which can generate Java classes from a DTD.

With iFS, one or more type definition (mapping) files define an object-relational mapping from the XML document to the database. iFS uses these mapping files both to construct tables in which the XML document can be stored, and to transfer data between XML documents and the database. Of interest, iFS can return data from the database directly as objects instead of XML documents, which is useful to many applications. iFS also supports content management features such as check-in/check-out and versioning.

Oracle Intermedia XML Search is a utility that can, "Automatically index and search XML Documents and Document Fragments of any size up to 4 Gigabytes each. [It has] powerful XML document searching including hierarchical element containership, doctype discrimination, and searching on XML attributes."

Oracle 9i (announced 2 Oct, 2000) includes "Native XML Database Support (XDB)", which introduces a new object data type (XMLType) and "features ... 'navigational' access and search for XML documents."

4.9.2 Data Storage in Native XML format

4.9.2.1 Overview

Native XML databases fall into two broad categories:

- Text-based storage Store the XML document in text form and provide database functionality in accessing the document. A simple approach for this may store the document as a BLOB (binary large object) in a relational database or as a file in a file system and provide XML indexes over the document. Another approach is to store the document in a specialized data store with indexes and transaction support.
- Model-based storage. It Store a binary model of the document (based on DOM) in an existing data store. For example, this could map the DOM to relational tables such as Elements, Attributes.

These two approaches give different storage and performance for native XML databases. But how do these native XML databases differ from the XML-enabled databases discussed in the last section. There are at least three differences:

Native XML databases can keep the XML structure intact (entity usage, CDATA sections, etc.) as well as comments, Processing Instructions, DTDs, etc. While XML-enabled databases can do this in theory, this is generally not done in practice. Native XML databases can store XML documents without knowing their schema (DTD). However, XML-enabled databases could generate schemas.

The only interface to the data in native XML databases is XML query languages, and related technologies, such as XPath, the DOM, or SAX. However, XML-enabled databases can directly support SQL.

Native XML Databases differ from XML Servers and XML Application Servers because these systems are generally front-ends to native XML databases.

4.9.2.2 Products

Many XML database system products are being produced. The main products are:

- Tamino, developed by AG software (www.tamino.com)
- Excelon, developed by excelon corporation (www.excelon.com)

4.9.2.3 XML Query Languages

Several query langauges are being developed to query XML native databases. Examples are XQL (XML Query Language), XQL, Quilt and others (see the XML Query Working Group Web site in w3c--www.w3.org). Some of these are based on XSL (e.g., XQL) while others extend SQL to include XML features (e.g., Quilt).

4.9.2.4 APIs for XML Access

Once we have information in XML format, we need means to manipulate this information to meet our purpose. For this purpose, several APIs are available for processing XML documents. Some are standard APIs like DOM (developed by W3C) and some are vendor specific APIs that the vendors have built in their products to make things easier. We will discuss DOM, SAX and techniques used in Oracle 8i/9i and SQL Server 2000.

4.9.2.4.1 DOM. (Document Object Model)

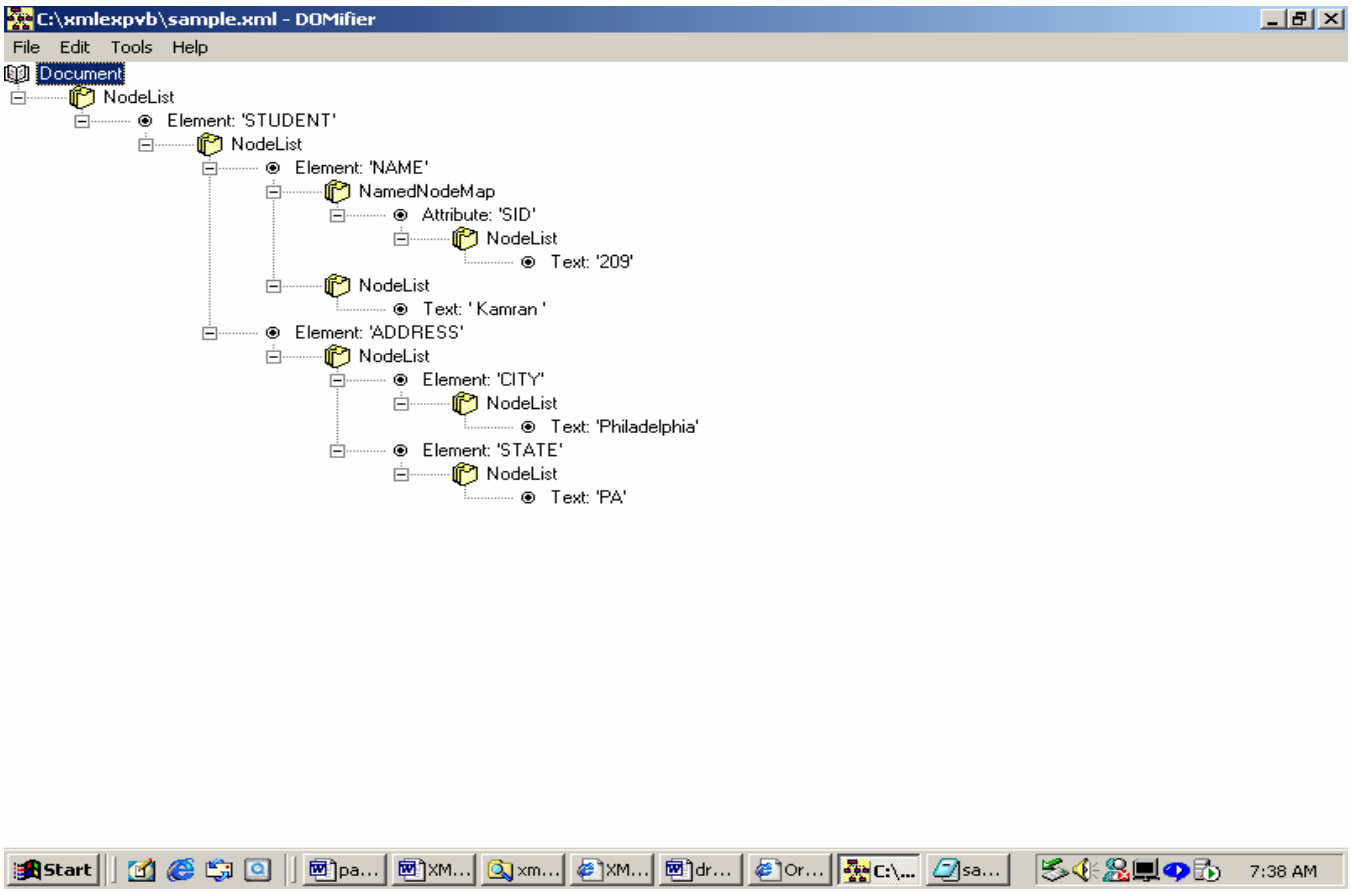
According to W3C, "The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents". DOM allows us to create documents, move, copy and remove parts of the document, add or modify attributes.

DOM (Document Object Model) treats XML documents as object models with methods and properties associated with each type of node. Node can be an element, attribute etc. DOM reads the XML document in memory and then we can manipulate it i.e., we can retrieve specific data or modify data etc.

DOM is used as an additional layer between XML parser and the application that needs the information in that document. To work with DOM, programmers need a parser, a DOM implementation e.g., object library MSXML by Microsoft includes a DOM implementation that can be used in vbscript, vb, javascript etc.

```
<STUDENT>
        <NAME SID="209"> Kamran </NAME>
        <ADDRESS>
                <CITY>Philadelphia</CITY>
                <STATE>PA</STATE>
        </ADDRESS>
</STUDENT>
```

e.g., above given xml data when loaded in DOM can be graphically represented as follows:



DOM can be used both for HTML and XML documents. The drawback of using DOM is that before we can use it to traverse a document, it has to build a massive in-memory map of the document that is a heavy tax on the memory and it is inefficient if we have to extract a small amount of information from a large document.

4.9.2.4.2 SAX.

SAX ("Simple API for XML") was developed by an informal group of participants of the XML-DEV mailing list. It was developed to enable more efficient analysis of large XML documents. SAX uses event-driven approach and it uses less memory than DOM. Unlike DOM, SAX do not make a map of the whole xml document but events such as start element tag, end element tag, etc. these events are used to manipulate the xml data.

4.9.2.5 Hands-on Experiments Conducted.

The following discussion is based on hands-on experiments conducted on an SQL Server 2000, an XML enabled RDBMS. The experiments show the use of "FOR XML" clause added to SQL statements. This statement , with its variants, produces an XML document. The SQL Server query analyzer was used to query the database.

1). Use of XML FOR with raw option. This option produces raw XML. For example, consider the following statement:

```
select
lastname as LN,
firstname as FN
```

```
from employees where firstname like 'a%'
for xml raw
```

This statement produces the following output:

```
<row LN="Fuller" FN="Andrew"/>
<row LN="Dodsworth" FN="Anne"/>
```

2). Use of XML FOR with Auto option. Given the following statement:

```
select
lastname as LN,
firstname as FN
from employees where firstname like 'a%'
for xml auto
```

This statement produces the following output:

```
<employees LN="Fuller" FN="Andrew"/>
<employees LN="Dodsworth" FN="Anne"/>
```

3). Use of Explicit option with XML FOR. This statement is very powerful and generates well-formed XML documents with multiple elements.

```
select 1 as tag,
null as parent,
lastname as [employee!1!lastname!element],
firstname as [employee!1!FName!element]
from employees
order by [employee!1!lastname!element]
for xml explicit
```

The above statement produces the following output:

```
<employee>
<lastname>Buchanan</lastname>
<FName>Steven</FName>
</employee>

<employee>
```

```
<lastname>Callahan</lastname>
<FName>Laura</FName>
</employee>

<employee>
<lastname>Davolio</lastname>
<FName>Nancy</FName>
</employee>
```

4.9.3 General Comments

This discussion addresses the issue of using XML as databases. In other words, how can XML be stored in relational databases (RDBs) or how can XML documents be treated as databases. Many approaches have been proposed to map data from XML to RDB and vice versa. The impedance mismatch does exist and it is not easy always to map data between these two formats. New versions of some RDBMS have included XML support to provide some mapping. Another idea is to store XML in native form in XML databases and avoid the conversion altogether. However, XML databases need a great deal of work before becoming a viable alternative to RDBs.

It should be also kept in mind that XML is not magic. Programs need to be written to extract, parse, validate and process the XML data. XML technologies are in their infancy stage at the time of this writing. New XML standards and products are appearing at a dizzying pace. For example, different XML document querying languages (XML QL, XQL, QUILT, and X Query) have emerged in a short period of time. Hopefully, the cloud of XML technologies will be cleared and reliable technologies will emerge in the next few years.

4.10 Sources of Additional Information

Arlett, M., and Williamson, C., “Internet Web Servers: Workload Characterization and Performance Implementations”. IEEE/ACM Transactions on Networking, Oct. 1997

Berners-Lee, T., et al, “The Semantic Web”, The Scientific American ,special issue May 2001.

Berners-Lee, T., “Weaving the Web”, Harper San Francisco, 1999

Berners-Lee, T. and , R. Cailliau, "World Wide Web" Computing in High Energy Physics 92, Anney, France, 1992.

Comer, D., "Internetworking with TCP/IP", Two Volumes, Prentice Hall, latest edition

December, J. and Randall, N., "The World Wide Web Unleashed", Sams Net Book, latest edition

Duska, B., et al, “The Measured Access Characteristics of World Wide Web Proxy Caches”, USENIX Symposium on Internet Technologies and Systems, Dec. 1997.

Gibson, W., "Neuromancer", Ace Books, New York, 1984.

Goldfarb, C. “XML by Example : Building E-Commerce Applications”, Series on Open Information Management, 1998

Graham, I., "HTML Source Book", latest edition, Wiley

Hahn, H., "Internet: Complete Reference", latest edition

Hunter, D. “Beginning XML”, Wrox Press, 2000

Kevin, W., “Professional XML Databases”, Wrox press LTD, December 2000

Krishnamurthy, B., and Rexford, J., "Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching, and Traffic Measurement", Addison Wesley, 2001.

McGrath, S., “XML by Example : Building E-Commerce Applications”, Charles F. Goldfarb Series on Open Information Management), Paperback - June 1998 .

Stevens, R., "UNIX Network Programming", Prentice Hall, latest edition

Tittel, E. and James, S., "HTML for Dummies", IDG Books, latest edition

Webref: http://info.cern.ch/hypertext/WWW/Protocols/HTTP/HTTP2.html - T. Berners-Lee (1994) "HTTP: A Protocol for Networked Information" CERN, IEFT Internet Draft, original version 1991.