[Unit 3: Protocols and Client/Server Applications]
Internet Technology (CSC-402)

Jagdish Bhatta

Central Department of Computer Science & Information Technology
Tribhuvan University

<u>SMTP (Simple Mail Transfer Protocol) :</u>

Simple Mail Transfer Protocol (**SMTP**) is an Internet standard for electronic mail (e-mail) transmission across Internet Protocol (IP) networks. SMTP uses TCP port 25. SMTP is a connection-oriented, text-based protocol in which a mail sender communicates with a mail receiver by issuing command strings and supplying necessary data over a reliable ordered data stream channel, typically a Transmission Control Protocol (TCP) connection. An *SMTP session* consists of commands originated by an SMTP client (the initiating agent, sender, or transmitter) and corresponding responses from the SMTP server (the listening agent, or receiver) so that the session is opened, and session parameters are exchanged. A session may include zero or more SMTP transactions. An *SMTP transaction* consists of three command/reply sequences. They are:

1. **MAIL** command, to establish the return address, a.k.a. Return-Path, 5321.From, mfrom, or envelope sender. This is the address for bounce messages.
2. **RCPT** command, to establish a recipient of this message. This command can be issued multiple times, one for each recipient. These addresses are also part of the envelope.
3. **DATA** to send the *message text*. This is the content of the message, as opposed to its envelope. It consists of a *message header* and a *message body* separated by an empty line. DATA is actually a group of commands, and the server replies twice: once to the *DATA command* proper, to acknowledge that it is ready to receive the text, and the second time after the end-of-data sequence, to either accept or reject the entire message.

SMTP is a delivery protocol only. It cannot *pull* messages from a remote server on demand. Other protocols, such as the Post Office Protocol (POP) and the Internet Message Access Protocol (IMAP) are specifically designed for retrieving messages and managing mail boxes. However, SMTP has a feature to initiate mail queue processing on a remote server so that the requesting system may receive any messages destined for it. POP and IMAP are preferred protocols when a user's personal computer is only intermittently powered up, or Internet connectivity is only transient and hosts cannot receive message during off-line periods.

The basic SMTP works well, but it is limited in several respects. **It does not include authentication**. This is quite useful for sending spam. **Another limitation is that SMTP transfers ASCII messages, not binary data.** This is why the base64 MIME content transfer encoding was needed. However, with that encoding the mail transmission uses bandwidth inefficiently, which is an issue for large messages. **A third limitation is that SMTP sends messages in the clear. It has no encryption to provide a measure of privacy against prying eyes.** To allow these and many other problems related to message processing to be addressed, SMTP was revised to have an extension mechanism. This mechanism is a mandatory part of the RFC 5321 standard. The use of SMTP with extensions is called **ESMTP** (**Extended SMTP**).

POP (Post Office Protocol):

The **Post Office Protocol** (**POP**) is an application-layer Internet standard protocol used by local e-mail clients to retrieve mail from a remote server over a TCP/IP connection. POP and IMAP (Internet Message Access Protocol) are the two most prevalent Internet standard protocols for e-mail retrieval. Virtually all modern e-mail clients and servers support both. The POP protocol has been developed through several versions, with version 3 (POP3) being the current standard. Most webmail service providers such as Hotmail, Gmail and Yahoo! Mail also provide IMAP and POP3 service.

Like it seems everything on the internet, mail retrieval is a client-server application. The Post Office Protocol defines how your email client should talk to the POP server. The POP is a very simple protocol. This makes it easy to implement, has earned the Post Office Protocol widespread adoption and makes it very robust, but it also means the Post Office Protocol provides only basic functionality.

Things that can be done via the POP include:

- Retrieve mail from an ISP and delete it on the server.
- Retrieve mail from an ISP but not delete it on the server.
- Ask whether new mail has arrived but not retrieve it.
- Peek at a few lines of a message to see whether it is worth retrieving.

Usually the POP server listens to port 110 for incoming connections. Upon connection from a POP client (your email program) it will hopefully respond with *+OK pop.philo.org ready* or something similar. The *+OK* indicates that everything is â€” OK. Its negative equivalent is *-ERR*, which means something has gone wrong. Maybe your email client has already shown you one of these negative server replies.

Clients that leave mail on servers generally use the UIDL (Unique ID Listing) command to get the current association of message-numbers to message identified by its unique identifier. The unique identifier is arbitrary, and might be repeated if the mailbox contains identical messages. In contrast, IMAP uses a 32-bit unique identifier (UID) that is assigned to messages in ascending (although not necessarily consecutive) order as they are received. When retrieving new messages, an IMAP client requests the UIDs greater than the highest UID among all previously-retrieved messages, whereas a POP client must fetch the entire UIDL map. For large mailboxes, this can require significant processing

An informal proposal had been outlined for a *"POP4"* specification, complete with a working server implementation. This *"POP4"* proposal added basic folder management, multipart message support, as well as message flag management, allowing for a light protocol which supports some popular IMAP features which POP3 currently lacks.

IMAP (Internet Message Access Protocol):

IMAP stands for Internet Message Access Protocol. It is a method of accessing electronic mail or bulletin board messages that are kept on a (possibly shared) mail server. In other words, it permits a "client" email program to access remote message stores as if they were local. **For example, email stored on an IMAP server can be manipulated from a desktop computer at home, a workstation at the office, and a notebook computer while traveling, without the need to transfer messages or files back and forth between these computers.**

IMAP's ability to access messages (both new and saved) from more than one computer has become extremely important as reliance on electronic messaging and use of multiple computers increase, but this functionality cannot be taken for granted: the widely used Post Office Protocol (POP) works best when one has only a single computer, since it was designed to support "offline" message access, wherein messages are downloaded and then deleted from the mail server. This mode of access is not compatible with access from multiple computers since it tends to sprinkle messages across all of the computers used for mail access. Thus, unless all of those machines share a common file system, the offline mode of access that POP was designed to support effectively ties the user to one computer for message storage and manipulation.

Key goals for IMAP include:

1. Be fully compatible with Internet messaging standards, e.g. MIME.
2. Allow message access and management from more than one computer.
3. Allow access without reliance on less efficient file access protocols.
4. Provide support for "online", "offline", and "disconnected" access modes
5. Support for concurrent access to shared mailboxes
6. Client software needs no knowledge about the server's file store format.

IMAP supports both on-line and off-line modes of operation. E-mail clients using IMAP generally leave messages on the server until the user explicitly deletes them. This and other characteristics of IMAP operation allow multiple clients to manage the same mailbox. Most e-mail *clients* support IMAP in addition to POP to retrieve messages; however, fewer e-mail *services* support IMAP. IMAP offers access to the mail storage. Clients may store local copies of the messages, but these are considered to be a temporary cache.

To use IMAP, the mail server runs an IMAP server that listens to port 143. The user agent runs an IMAP client. Incoming e-mail messages are sent to an e-mail server that stores messages in the recipient's e-mail box. **The user retrieves the messages with an e-mail client that uses one of a number of e-mail retrieval protocols. Some clients and servers preferentially use vendor-specific, proprietary protocols, but most support the Internet standard protocols, SMTP for sending e-mail and POP and IMAP for retrieving e-mail, allowing interoperability with other servers and clients.**

When using POP, clients typically connect to the e-mail server briefly, only as long as it takes to download new messages. When using IMAP4, clients often stay connected as long as the user interface is active and download message content on demand. For users with many or large messages, this IMAP4 usage pattern can result in faster response times.

The POP protocol requires the currently connected client to be the only client connected to the mailbox. In contrast, the IMAP protocol specifically allows simultaneous access by multiple clients and provides mechanisms for clients to detect changes made to the mailbox by other, concurrently connected, clients.

PGP (Pretty Good Privacy)

Pretty Good Privacy (PGP) is a data encryption and decryption computer program that provides cryptographic privacy and authentication for data communication. PGP is often used for signing, encrypting and decrypting texts, e-mails, files, directories and whole disk partitions to increase the security of e-mail communications.

PGP is a public key encryption package to protect e-mail and data files. It lets you communicate securely with people you've never met, with no secure channels needed for prior exchange of keys. It's well featured and fast, with sophisticated key management, digital signatures, data compression, and good economic design. **The actual operation of PGP is based on five services: authentication, confidentiality, compression, e-mail compatibility, and segmentation.**

- PGP provides authentication via a digital signature scheme.
- PGP provides confidentiality by encrypting messages before transmission using RSA schemes.
- PGP compresses the message after applying the signature and before encryption. The idea is to save space.

- PGP encrypts a message together with the signature (if not sent separately) resulting into a stream of arbitrary 8-bit octets. But since many e-mail systems permit only use of blocks consisting of ASCII text, PGP accommodates this by converting the raw 8-bit binary streams into streams of printable ASCII characters using a radix-64 conversion scheme. On receipt, the block is converted back from radix-64 format to binary.
- To accommodate e-mail size restrictions, PGP automatically segments email messages that are too long. However, the segmentation is done after all the housekeeping is done on the message, just before transmitting it. So the session key and signature appear only once at the beginning of the first segment transmitted. At receipt, the receiving PGP strips off all e-mail headers and reassemble the original mail.

HTTP

HTTP is the protocol to exchange or transfer hypertext. ***H****yper****T****ext* ***T****ransfer* ***P****rotocol*, the underlying protocol used by the World Wide Web. HTTP defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands. For example, when you enter a URL in your browser, this actually sends an HTTP command to the Web server directing it to fetch and transmit the requested Web page. HTTP is the framework for how browsers will display and use file formats. When you enter in a URL with HTTP at the beginning, you are requesting a web page which can contain other elements (such as pictures) and links to other resources. HTTP utilizes TCP port 80 by default, though other ports such as 8080 can alternatively be used.

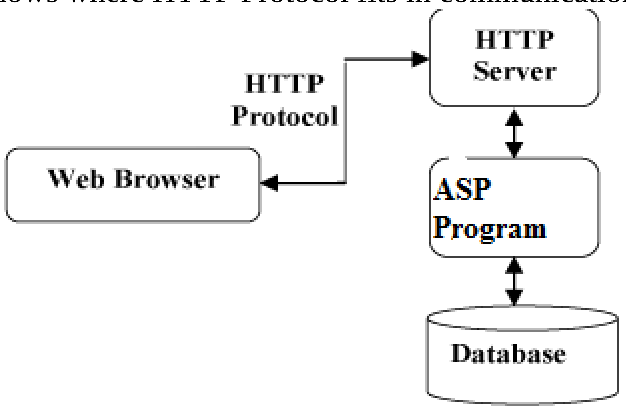
The current version of HTTP in widespread use - HTTP version 1.1 - was developed to address some of the performance limitations of the original version - HTTP 1.0. HTTP 1.1 is documented in RFC 2068.

There are three important things about HTTP of which you should be aware:

- **HTTP is connectionless:** After a request is made, the client disconnects from the server and waits for a response. The server must re-establish the connection after it processes the request.
- **HTTP is media independent:** Any type of data can be sent by HTTP as long as both the client and server know how to handle the data content. How content is handled is determined by the MIME specification.
- **HTTP is stateless:** This is a direct result of HTTP's being connectionless. The server and client are aware of each other only during a request. Afterwards, each

forgets the other. For this reason neither the client nor the browser can retain information between different requests across the web pages.

Following diagram shows where HTTP Protocol fits in communication;



HTTP is an application layer network protocol built on top of TCP. HTTP clients (such as Web browsers) and servers communicate via HTTP request and response messages. **The three main HTTP message types are GET, POST, and HEAD.**

An HTTP session is a sequence of network request-response transactions. **An HTTP client initiates a request by establishing a Transmission Control Protocol (TCP) connection to a particular port on a server (typically port 80). An HTTP server listening on that port waits for a client's request message. Upon receiving the request, the server sends back a status line, such as "HTTP/1.1 200 OK", and a message of its own**. The body of this message is typically the requested resource, although an error message or other information may also be returned

HTTP Request Methods:

HTTP defines nine methods (sometimes referred to as "verbs") indicating the desired action to be performed on the identified **resource**. What this resource represents, whether pre-existing data or data that is generated dynamically, depends on the implementation of the server. Often, the resource corresponds to a file or the output of an executable residing on the server.

HEAD: Asks for the response identical to the one that would correspond to a GET request, but without the response body. This is useful for retrieving meta-information written in response headers, without having to transport the entire content.

GET: Requests a representation of the specified resource. Requests using GET should only retrieve data and should have no other effect. (This is also true of some other HTTP methods.) The W3C has published guidance principles on this distinction, saying, "Web application design should be informed by the above principles, but also by the relevant limitations."

POST: Submits data to be processed (e.g., from an HTML form) to the identified resource. The data is included in the body of the request. This may result in the creation of a new resource or the updates of existing resources or both.

PUT: Uploads a representation of the specified resource.

DELETE: Deletes the specified resource.

TRACE: Echoes back the received request, so that a client can see what (if any) changes or additions have been made by intermediate servers.

OPTIONS: Returns the HTTP methods that the server supports for specified URL. This can be used to check the functionality of a web server by requesting '*' instead of a specific resource.

CONNECT: Converts the request connection to a transparent TCP/IP tunnel, usually to facilitate SSL-encrypted communication (HTTPS) through an unencrypted HTTP proxy.

PATCH: Is used to apply partial modifications to a resource.

HTTP servers are required to implement at least the GET and HEAD methods[13] and, whenever possible, also the OPTIONS method.

FTP:

File Transfer Protocol (FTP) lives up to its name and provides a method for transferring files over a network from one computer to another. More generally, it provides for some simple file management on the contents of a remote computer. It is an old protocol and is used less than it was before the World Wide Web came along. Today, its primary use is uploading files to a Web site. It can also be used for downloading from the Web but, more often than not, downloading is done via HTTP. Sites that have a lot of downloading (software sites, for example) will often have an FTP server to handle the traffic. If FTP is involved, the URL will have *ftp:* at the front.

The File Transfer Protocol is used to send files from one system to another under user commands. Both text and binary files are accommodated and the protocol provides features for controlling user access. When a user wishes to engage in File transfer, FTP sets up a TCP connection to the target system for the exchange of control messages. These allow used ID and password to be transmitted and allow the user to specify the file and file action desired. Once file transfer is approved, a second TCP connection is set up for data transfer. The file is transferred over the data connection, without the overhead of headers, or control information at the application level. When the transfer is complete, the control connection is used to signal the completion and to accept new file transfer commands.

FTP can be run in active or passive mode, which determines how the data connection is established. In active mode, the client sends the server the IP address and port number on which the client will listen, and the server initiates the TCP connection. at the condition when the client is behind a firewall and unable to accept incoming TCP connections, passive mode may be used. In this mode the client sends a PASV command to the server and receives an IP address and port number in return. The client uses these to open the data connection to the server. Data transfer can be done in any of three modes:

- Stream mode: Data is sent as a continuous stream, relieving FTP from doing any processing. Rather, all processing is left up to TCP. No End-of-file indicator is needed, unless the data is divided into records.

- Block mode: FTP breaks the data into several blocks (block header, byte count, and data field) and then passes it on to TCP.

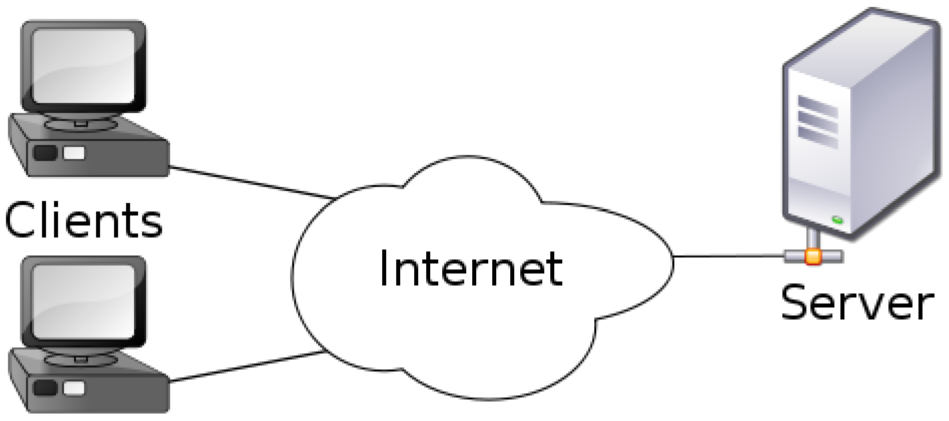
- Compressed mode: Data is compressed using a single algorithm (usually run-length encoding).

Client/Server Model:

The **client–server model** is a computing model that acts as distributed application which partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server machine is a host that is running one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests.

Client/Server Architecture:

Client server network architecture consists of two kinds of computers: clients and servers. Clients are the computers that that do not share any of its resources but requests data and other services from the server computers and server computers provide services to the client computers by responding to client computers requests. Normally servers are powerful computers and clients are less powerful personal computers. Web servers are included as part of a larger package of internet and intranet related programs for serving e-mail, downloading requests for FTP files and building and publishing web pages.



Advantages

- The client/ server architecture reduces network traffic by providing a query response to the user rather than transferring total files.
- The client/ server model improves multi-user updating through a graphical user interface (GUI) front end to the shared database.
- Easy to implement security policies, since the data are stored in central location
- Simplified network administration

Disadvantages

- Failure of the server causes whole network to be collapsed
- Expensive than P2P, Dedicated powerful servers are needed
- Extra effort are needed for administering and managing the server.

Client/Server architecture can be of different model based on the number of layers it holds. Some of them are;

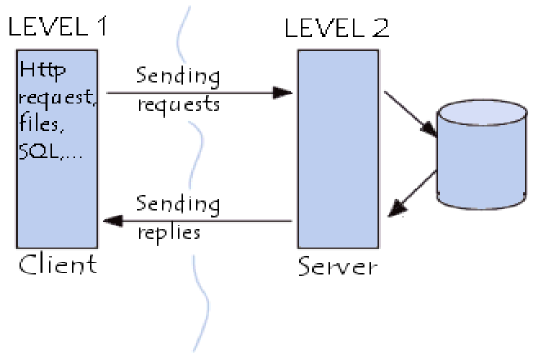
- **2-Tier Architecture**

  2-tier architecture is used to describe client/server systems where the client requests resources and the server responds directly to the request, using its own resources. This means that the server does not call on another application in order to provide part of the service. It runs the client processes separately from the server processes, usually on a different computer:

  – The client processes provide an interface for the customer, and gather and present data usually on the customer's computer. This part of the application is the presentation layer

  – The server processes provide an interface with the data store of the business. This part of the application is the data layer

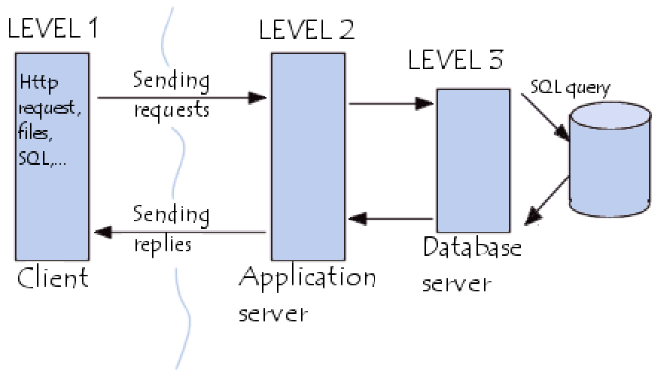
  – The business logic that validates data, monitors security and permissions, and performs other business rules can be housed on either the client or the server, or split between the two.
    - Fundamental units of work required to complete the business process
    - Business rules can be automated by an application program.



- **3-Tier Architecture**

In 3-tier architecture, there is an intermediary level, meaning the architecture is generally split up between:

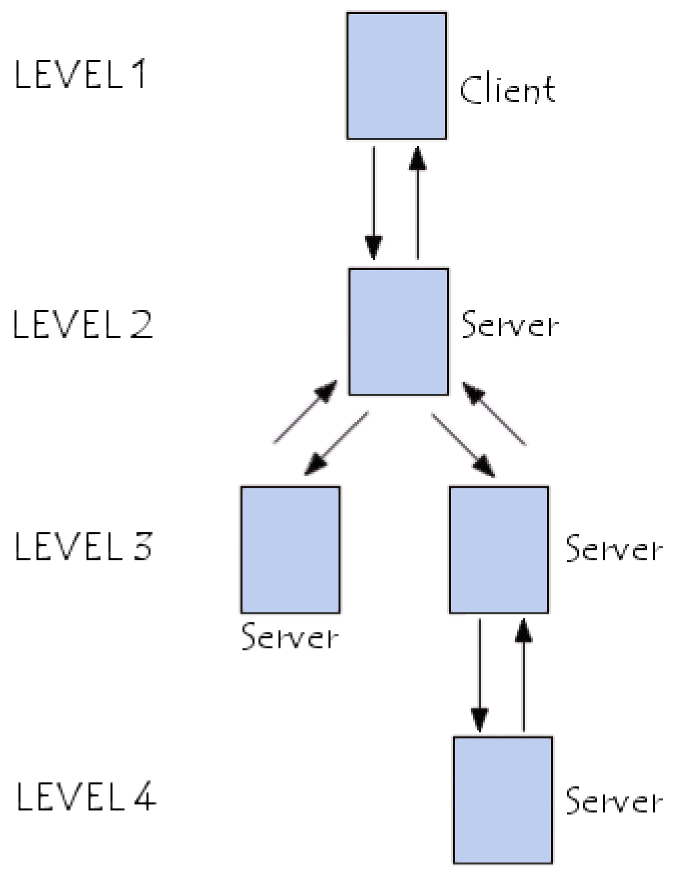
  - A client, i.e. the computer, which requests the resources, equipped with a user interface (usually a web browser) for presentation purposes
  - The application server (also called **middleware**), whose task it is to provide the requested resources, but by calling on another server
  - The data server, which provides the application server with the data it requires



- **N-Tier Architecture (multi-tier)**

N-tier architecture (with N more than 3) is really 3 tier architectures in which the middle tier is split up into new tiers. The application tier is broken down into separate parts. What these parts are differs from system to system. The following picture shows it:

The primary advantage of N-tier architectures is that they make load balancing possible. Since the application logic is distributed between several servers, processing can then be more evenly distributed among those servers. N-tiered architectures are also more easily scalable, since only servers experiencing high demand, such as the application server, need be upgraded. The primary disadvantage of N-tier architectures is that it is also more difficult to program and test an N-tier architecture due to its increased complexity.



Advantages of Multi-Tier Client/Server architectures include:

- Changes to the user interface or to the application logic are largely independent from one another, allowing the application to evolve easily to meet new requirements.

- Network bottlenecks are minimized because the application layer does not transmit extra data to the client, only what is needed to handle a task.

- The client is insulated from database and network operations. The client can access data easily and quickly without having to know where data is or how many servers are on the system.

- Database connections can be 'pooled' and thus shared by several users, which greatly reduces the cost associated with per-user licensing.

- The organization has database independence because the data layer is written using standard SQL which is platform independent. The enterprise is not tied to vendor-specific stored procedures.

- The application layer can be written in standard third or fourth generation languages, such as ASP, PHP with which the organization's in-house programmers are experienced.

What kind of systems can benefit?

Generally, any Client/Server system can be implemented in an 'N-Tier' architecture, where application logic is partitioned among various servers. This application partitioning creates an integrated information infrastructure which enables consistent, secure, and global access to critical data. A significant reduction in network traffic, which leads to faster network communications, greater reliability, and greater overall performance is also made possible in a 'N-Tier' Client/Server architecture.

Universal Internet Browsing: (Discussed in class)

Multiple Protocol Support:

Complex data communication systems no longer utilize a single protocol to handle all transmission tasks. Instead, they require a set of protocols, called a protocol suite. The reason for using multiple protocols is to make them less complicated; this simplifies dealing with problems that arise when machines communicate over a network. such problems include the following;

- **Hardware failure:** when a router or host hardware fails, the protocol software needs to detect the failure and recover from it.

- **Network congestion:** the protocol needs to detect when the network capacity has been exceeded and arrange a way to handle the congestion.

- **Packet delay or loss:** the protocol software needs to adapt to long delay in order not to lose packets that were significantly delayed.

- **Data corruption:** the protocol software needs to detect and recover from transmission errors and corruption due to transmission impairments of hardware failures

- **Data duplication or sequence error:** networks that offer multiple routes may deliver data out of sequence or deliver duplicates of packets. The protocol software needs to reorder packets and remove duplicates.

It is difficult or undesirable to write a single protocol that will handle everything in the network, particularly since many networks are heterogeneous. Hence multiple protocols are needed in network communications. There has been lots of protocols adapted till the date.