Assignment 1

Georgia Koloniari
Salteas-Kalogeras Panagiotis

1. *Goal 1:* The web offers a simple, consistent and uniform model of distributed documents. The main type of web-based documents is HTML documents. Besides HTML and XML there are many other type of documents expressed in the form of a MIME type (Multipurpose Internet Mail Extensions). MIME makes a distinction between top-level types and subtypes. Top-level types include

- text (plain, HTML, XML)
- image (gif, jpeg)
- audio (basic, tone)
- video (mpeg, pointer)
- application (octet-stream, postscript, pdf)
- multipart (mixed, parallel)

The main problems derived by sharing are that of security and unwanted communication. Most of the security issues in the Web deal with setting up a secure channel between a client and server. The Secure Socket Layer (SSL) is the predominant approach for setting up a secure channel in the Web. Most clients and servers support it. An update of SSL is the Transport Layer Security (TLS) protocol. TLS is an application-independent security protocol that is logically layered on top of a transport protocol, usually TCP. TLS is organized into two layers and the secure channel setup proceeds into two phases. At the first phase the cryptographic algorithm and compression method are chosen and at the second phase the authentication takes place. Furthermore, the web with the use of proxies and firewalls provides means to filter out unwanted communication and offers some additional security. Finally, web-based applications like web browsers allow us to disable JavaScripts and protect ourselves from malicious code or annoying pop-up windows.

Goal 2: The web is actually a front-end that offers great transparency and allows the end user to concentrate on the particular task she is interested in without being concerned or aware of how that task is implemented by the lower layers of the system.

- Access Transparency: Web offers this kind of transparency since a user can access resources independently of the way these resources are manipulated by the server's operating system. For example a user can view a file without knowing the naming conventions used by the file system of the web server providing the file and all the necessary manipulations are hidden from her.
- Location Transparency: Web uses a naming scheme that is based on logical names therefore it provides location transparency. A user can access any resource by simply knowing its logical name - which does not reveal any information about its location - without knowing where that resource is actually stored.
- Migration Transparency: The naming scheme used also provides no information about whether a resource has always been at the same web server or has just moved there, supporting migration transparency as well. We can

access a web file by using its name, even if that file has moved to a new site since the last time we have accessed it.

- Relocation Transparency: Relocation transparency is only partially supported by the Web. For example, we can access a file that is located on a server of a mobile computer even if the user of that computer is moving without being aware of that movement. On the other hand, if we wish to move a file from one sever to another while this file is being accessed by some user, either this action will not be allowed and we will have to copy the file, or the will be aware of the change since the connection will be lost.
- Replication Transparency: Replication Transparency is also supported by the Web. Once again the naming scheme hides the existence of many replicas that may be located at different sites. However, the Web does not give any guarantees about the consistency of these replicas. For example, an update in one replica does not assume the automatic update of the other replicas.
- Concurrency Transparency: One of the main characteristics of the Web is that users can simultaneously access the same resource without realizing that other users are using it. The Web can be considered as being a read-mostly system, where updates are generally done by a single person without introducing conflicts that would hinder concurrency transparency. However, some applications that require writes may cause conflicts that require locking mechanisms so as to be resolved. If the application does not provide these mechanisms the Web takes no extra provision so concurrency transparency is lost. For example, if two users wish to change an object at a database, which does not provide concurrency control, through a web service an error is very likely to occur.
- Failure Transparency: The Web does not support failure transparency. The problem of hiding failures is difficult because of the inability to distinguish between a dead connection and a very slow one. Although many browsers or other web-based applications repeatedly try to contact a server if it is not responding, they will eventually time out and report that the resource is unavailable.
- Persistence Transparency: The Web finally offers persistence transparency. The user is unaware of the fact that a server might move a file from disk to main memory or vice versa to complete a given task applied by the user.

Goal3: An open system is a system that offers services according to standard rules that describe the syntax and semantics of those services. Since all the messages that are exchanged through the web are defined by the HTTP protocol Web satisfies the definition for an open system. Also the Web supports interoperability since its components from different manufacturers co-exist and co-operate together. Also portability is supported since applications developed for a sub-system can easily be brought to another without modifications since their interfaces are clearly defined. However as far as Web services are concerned, although the syntax has been clearly defined through SOAP, RDF and WSDL the semantic part is still undefined thus bringing up problems to the extensibility of the Web. A user will not use a service just because she knows how, if she does not know what this service does.
The protocols used be the Web can be divided into three categories:

- Data Representation Protocols – HTML (HyperText Markup Language) that provides keywords in order to structure a document and instruct the browser how to present it.

- Naming Scheme Protocol – URI (Uniform Resource Identifier) is the single naming scheme that is used to refer to documents. URIs have two forms, the Uniform Resource Locator (URL) that identifies a document by including information on how and where to access the document and the Uniform Resource Name (URN) that acts a true unique identifier.
- Communication Protocol – HTTP (Hypertext Transfer Protocol), which is a simple client-server protocol. Other protocols that are used are FTP for file transfer, telnet for remote login, NNTP (Network News Protocol) for Usenet news and Gopher a simple variation of HTTP.

An example of a web mechanism and policy is that concerning security. A Web browser provides facilities for determining the level of security we wish to enforce. A user can decide whether she wishes to disable JavaScripts to run on her machine, how to handle cookies, accept or reject them. Also a user can define some trusted sites from which it can receive cookies or download code and data without danger. The tuning of these parameters the browser offers define its user's own security policy.

Goal 4: Web is scalable in all the three dimensions, with respect to size, geographically and administratively. Web nowadays supports a large number of users it is spread all over the world and it contains many separate administrative domains. The Web employs the three well-known techniques of scaling in order to achieve its scalability in all the dimensions. For geographical scalability, web applications use asynchronous communication in order to conceal network latencies. For example, a web browser opens multiple threads to download a web file and issues a request for every object in the file without waiting for the previous request to be satisfied. Furthermore Web also uses the technique of reducing communication by moving some of the processing at the client side. For example, a form is filled in at the client, checked and then shipped to the server as a whole and not field by field.

The technique of distribution is also used both for geographical and size scalability. To most users web appears as an enormous document-based system where each document has its own unique name. It may even appear that there is only a single server. However, the Web is physically distributed across a large number of servers, each handling a number of documents and the name of the server that handles the document is encoded into the URL (the document's name). The DNS naming service is another example of distribution in the Web. The name space is hierarchically organized into a tree of domains, which are divided into non-overlapping zones. Each server is responsible for a single zone. Thus the name resolution process is divided among many components of the system and the load is balanced so the system can scale efficient in all the three dimensions.

Replication and caching also improve scalability. By adding replicas of a resource we increase its availability so that more users can use it and thus we can improve scalability with respect to size. By caching we bring data closer to a user reducing communication costs and thus scaling up both geographically and again with respect to size. Of course keeping all these replicas up-to-date presents another problem of immediate propagation of updates that can be an obstacle to efficient scaling.

Finally the mechanisms for security that the Web provides enable us to extend it to many administrative domains where each of them enforces its own policy and they can co-operate together.

2.

(a) The paper presents a design principle that helps guide placement of functions among the modules of a distributed computer system. The principle, called the end-to-end argument, suggests that some functions can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system. Performing the function at the lower level may cost more for two reasons: Firstly, since the lower level subsystem is common to many applications, those applications that do not need the function will pay for it anyway. Second, the low-level subsystem may not have as much information as the higher levels, so it cannot do the job as efficiently. However some effort at the lower levels to improve network reliability can have a significant effect on the application performance but it should be seen as an engineering tradeoff between performance rather than a requirement for correctness. This direction should not be pushed very far since the function that the lower layer provides will have to be implemented by the application as well, since it poses the global knowledge. Using this argument requires a subtle analysis of application requirements in order to identify correctly the end-points in the communication. Thus the end-to-end argument is not an absolute rule, but rather a guideline that helps in application and protocol design analysis; one must use some care to identify the end points to which the argument should be applied. Examples discussed in the paper include bit error recovery, security using encryption, duplicate message suppression, recovery from system crashes, and delivery acknowledgement. Low level mechanisms to support these functions are justified only as performance enhancements.

(b) The end-to end argument introduces an overhead for the applications of the end points, which will have to deal with problems such as error correction and message integrity. Therefore applications will spend more time addressing these problems and new delays will be also introduced. Especially for end-points with limited computing capabilities this problem can become very serious.

Also an overhead for the application designer is also introduced. Each designer will have to address problems that were probably met by many others as well and will have to solve them on his own. If a lower layer were responsible fort this task the designer would have much more time to spend with designing the actual application and not things that were common for all.

The end-to-end is mainly target for a general-purpose environment or situations where developers may not know ahead of time what an application need to do or requires. Consider an isolate high speed, highly reliable system designed specifically for a military use, this is where the systems/networks are only used on fewer purposes. They are highly customized systems. This is where special or specific functions may be able to build in to low levels for variety different reasons including performance, security so on.

Furthermore, for an unreliable network where the possibility of a package to be lost or corrupted is high, the end-to-end argument fails, since the whole file that the application need would have to be transmitted before the error could be identified and the procedure would have to start from scratch. By applying some reliability checks per package by the lower layer and by the intermediate nodes as well we can improve the performance of the application significantly.

Finally when we require that confidential data be not exposed to anyone by misbehaving users these data must be encrypted and only read by authorized users. This can only be achieved if all the data that are transmitted through the network are

automatically encrypted and this violates the end-to-end argument since no application can be responsible for this task. The transmission layer must encrypt the data before letting them in the network.
Another disadvantage is the problem of correctly defining the end-points to which one should apply the end-to-end argument.

(c) An example of a distributed system that violates the end-to-end argument is the Web and basically any system that is implemented on top of the TCP layer. Although all of the functions mentioned in the paper are such as delivery guarantees, and FIFO message delivery are handled by the two end-points of the communication, these functions are not implemented at the application layer but at the communication subsystem and in particular, by the transport layer (TCP). So the Web does support the argument in the extent that it uses IP datagrams that are only processed by the two end-points of a communication and the intermediate nodes have no idea or control over their content. However it also violates the argument, as the TCP is responsible for all the functions we have mentioned and not the application itself.
Another example is distributed systems that use proxies or firewalls. Proxies and firewalls filter out unwanted communication or suspicious messages before their reaching their destination, that is the end-point. In this case, an intermediate node does process a message and rejects it at will.