An evaluation of the WebDAV extensions to the HTTP protocol

Björn Nilsson

Civilingenjörsprogrammet
Institutionen för Systemteknik
Avdelningen för Datorkommunikation
An evaluation of the WebDAV extensions to the HTTP protocol

Master’s Thesis in Computer Science
Björn Nilsson
January 2000
Abstract

The HyperText Transfer Protocol (HTTP) is used for the most popular service on the Internet today – World Wide Web. WebDAV is a new extension to the HTTP protocol, which makes it possible to write, edit, and share information across intranets and the Internet. In this Master's Thesis, the WebDAV extensions are examined. A comparison between HTTP with WebDAV and the existing Internet protocols FTP and POP3 is done. Also a behavioral analysis of existing WebDAV applications is made. The conclusions are that HTTP with WebDAV extensions can replace both FTP and POP3. When using HTTP’s pipelining and persistent connections, better performance than with FTP is achieved. It seems like WebDAV can be used as a universal protocol for client-server solutions, gaining the advantages of HTTP such as encryption and caching.
Preface

This Master's Thesis has been made as the final part of my Master of Science degree in Computer Science and Engineering at Luleå University of Technology (LTU). The work has been carried out from September 1999 to January 2000 at Telia ProSoft AB in Malmö.

I would like to thank my supervisor at Telia ProSoft, Anders Jönsson, for assistance and guidance throughout the work. I would also like to thank Lars-Åke Larzon at LTU who has been my examiner.

Björn Nilsson
Malmö, January 2000
# Table of Contents

## 1 INTRODUCTION

1.1 DESCRIPTION OF THE ASSIGNMENT .............................................................. 1
1.2 PURPOSE OF THE ASSIGNMENT ................................................................. 1
1.3 LIMITATIONS ............................................................................................... 2
1.4 STRUCTURE OF THE REPORT ..................................................................... 2

## 2 BACKGROUND

2.1 HTTP - HYPERTEXT TRANSFER PROTOCOL .................................................. 3
  2.1.1 Overview .............................................................................................. 3
  2.1.2 Methods ............................................................................................... 3
  2.1.3 Persistent connections and pipelining .................................................... 5
  2.1.4 Byte range operations ........................................................................... 6
  2.1.5 Authentication ...................................................................................... 6
  2.1.6 Integrity ............................................................................................... 11
  2.1.7 Confidentiality ..................................................................................... 11
  2.2 FTP – FILE TRANSFER PROTOCOL .......................................................... 12
  2.2.1 Overview ............................................................................................ 12
  2.2.2 Commands ......................................................................................... 13
  2.2.3 Authentication ..................................................................................... 14
  2.2.4 Integrity ............................................................................................. 15
  2.2.5 Confidentiality .................................................................................... 15
  2.3 POP3 – POST OFFICE PROTOCOL ............................................................ 15
  2.3.1 Overview ............................................................................................ 15
  2.3.2 Commands ......................................................................................... 16
  2.3.3 Security issues .................................................................................... 17

## 3 WEBDAV

3.1 OVERWRITE PREVENTION ....................................................................... 18
3.2 PROPERTIES .............................................................................................. 21
3.3 NAME SPACE MANAGEMENT ................................................................. 23
3.4 WORK IN PROGRESS .............................................................................. 25
  3.4.1 Advanced Collections ......................................................................... 25
  3.4.2 Versioning ......................................................................................... 27
  3.4.3 Searching and Locating ...................................................................... 28
  3.4.4 Access Control .................................................................................. 29
3.5 WEBDAV SERVERS .................................................................................. 31
3.6 SECURITY CONSIDERATIONS ............................................................... 31

## 4 ANALYSIS OF WEBDAV

4.1 FILE TRANSFERS ..................................................................................... 33
  4.1.1 Testbed setup ....................................................................................... 33
  4.1.2 Performance ....................................................................................... 33
  4.1.3 Functionality ....................................................................................... 39
  4.1.4 Security ............................................................................................. 40
4.2 MAIL READING ....................................................................................... 40
  4.2.1 Overview ............................................................................................ 40
  4.2.2 Properties ......................................................................................... 40
  4.2.3 Functionality of the client ................................................................... 41
4.3 BEHAVIOR OF COMMON WEBDAV APPLICATIONS .............................. 42
  4.3.1 Web Folders ...................................................................................... 42
  4.3.2 Office 2000 ....................................................................................... 46

## 5 CONCLUSIONS

......................................................................................................................... 50

## 6 REFERENCES

.......................................................................................................................... 51

## APPENDIX A – GLOSSARY

.......................................................................................................................... 53
1 Introduction

From 1994 up until today, the World Wide Web has developed from a forum in which scientists exchanged their research reports into a center where everyone has a Web page. A large part of the television commercials today include a URL. In order for this to happen, the architecture behind Internet has expanded tremendously from its original specification. The core of the Web is the Hypertext Markup Language (HTML), a markup language to show and format text in web browsers. Originally, the language only supported linking to other pages, inclusion of pictures and text formatting like bold, italics and headlines. In several steps it expanded with functions such as frames, forms, script languages and style sheets. Now it is common to create connections between Web pages and databases to show the contents dynamically.

Behind the markup language is a protocol that fetches the wanted Web pages. The name of the protocol is the Hypertext Transfer Protocol (HTTP). From the beginning of 1990, it was a simple protocol that only sent HTML text documents. In the latest version, it is a generic transfer protocol, which handles all kind of data. It also encompasses functionality for authentication, caching and encryption.

The latest features to HTTP are extensions for Distributed Authoring and Versioning (WebDAV). With WebDAV, it becomes feasible to write, edit and share information across intranets and the Internet. The main features are overwrite prevention, properties and namespace management. Overwrite prevention makes it possible to avoid the "lost update problem" that occurs as changes to a document are lost when several authors access and attempt to edit a file at the same time. Properties is a way to store and retrieve information about a web document such as the author's name, copyright, publication date and keywords that Internet search engines use to find and retrieve relevant documents. The namespace management capabilities enable users to manage Internet files and directories, including the ability to move and copy files.

Several major software vendors are currently developing and adopting WebDAV. For example, Microsoft supports WebDAV in Office 2000, Internet Information Server (IIS) 5.0 and Internet Explorer 5.0. Microsoft has created a function in their operating systems called "Web Folders". With it, it is possible to map and navigate in folders on Web servers as ordinary folders on the hard drive. Some people state that the functionality in WebDAV can replace existing protocols like POP3, FTP and IMAP.

1.1 Description of the assignment

This Master's Thesis will examine WebDAV from some different views. First, it will be examined if WebDAV makes it possible to replace other protocols. FTP and POP3 will be examined. The comparison will focus on performance (only FTP), functionality and security. When comparing security, three different perspectives are covered: authentication (who can access the data?), integrity (is it possible to detect manipulated data?) and confidentiality (how to make the data unreadable to eavesdroppers). Second, the behavior of existing WebDAV applications are examined. The applications that will be examined are Microsoft's Web Folders and Office 2000. The analysis will study how they use the HTTP protocol and the WebDAV extensions.

1.2 Purpose of the assignment

The purpose of this Master's Thesis is to evaluate WebDAV and its possibilities. Today, much of the work at Telia ProSoft is to create web based solutions. The most concrete example of this is Telia's Instant Office, an effort to
create a virtual office on the Internet. With it, customers can exchange documents and messages over the Internet. Other systems built at Telia ProSoft often use the web as an interface to its users. Due to the rapid development with different web techniques, this Master’s Thesis is a way to examine one aspect of new technology that could be important for the way Internet is used in the future.

1.3 Limitations

This task has been done in association with Telia ProSoft in Malmö, which means that the study has focused on aspects of importance for them. A problem with the existing work on WebDAV is that many important parts are still Internet-Drafts, some at an early stage in the development. Some drafts, not mentioned here, suggest changes to the current RFC about WebDAV. Since people at Microsoft wrote some of them, it is possible that these changes are already implemented in Microsoft’s HTTP server. An important area where this is the case is DASL (DAV Search and Locating). In some cases there did not exist an implementation to test a specific function in WebDAV. When this was the case, I have copied the examples from the Internet-Drafts and updated the XML syntax.

In the tests of FTP file transfers, typical traffic was not simulated in a model. Instead the tests was designed to make the differences between HTTP and FTP obvious.

1.4 Structure of the report

In Chapter 2 is a description of the common Internet protocols HTTP, FTP and POP3. Chapter 3 explains WebDAV and its different components. Chapter 4 concerns the comparison between WebDAV and the other protocols, and the analysis of existing applications. In Chapter 5, some conclusions from this study are found and all references are in Chapter 6.
2 Background

This chapter will describe the different protocols that are discussed in the report. First is a description of HTTP, the transfer protocol that is used for HTML documents. Then are explanations of the File Transfer Protocol (FTP) and the Post Office Protocol (POP3), which are the most common used protocols for file transfers and fetching mail.

2.1 HTTP - Hypertext Transfer Protocol

HTTP is the protocol for Internet’s most popular service – World Wide Web. In this section is a description of the protocol [1] and some examples. Authentication, integrity and confidentiality are also discussed [2].

2.1.1 Overview

The Hypertext Transfer Protocol (HTTP) is an application-level stateless protocol. It is based on a request/response paradigm. When the client wants to retrieve a document, it connects to a server and sends a request to the server in the form of a request method, the URI, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content. The response from the server is a status line, including the message’s protocol version and a status code, followed by a MIME-like message containing server information, entity meta information, and possible body content.

The status code is a three-digit integer, which makes it easy for computers to interpret the status. It is followed by a text explanation of the status for human users. The first digit of the status code defines the class of the response. There are five values for the first digit, number 1 - 5. Some usual responses are: 200 OK, 403 Forbidden and 404 Not Found.

On the Internet, HTTP communication generally takes place over TCP/IP connections. HTTP can be implemented on top of other protocols, but it presumes a reliable transport. The default port is TCP 80, but other ports can be used. A picture of HTTP in relation to the other Internet protocols is shown in Figure 1. The current version of HTTP is 1.1. Its predecessors are HTTP/0.9 and HTTP/1.0. Version 1.0 of the protocol is still used, but 0.9 is obsolete.

2.1.2 Methods

When the client wants to do something with a web server, it sends a request containing a method. The possible methods are listed in Table 1. It is also possible to invent own methods and use them. The protocol standard talks about allowing the client to send an “extension-method”. This is of course only useful if the new methods are widely implemented in web servers around the world. WebDAV is an example of a suite of new methods to extend the functionality of web servers.
An evaluation of the WebDAV extensions to the HTTP protocol

Background

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Retrieves the specified resource.</td>
</tr>
<tr>
<td>HEAD</td>
<td>Retrieves all information about the specified resource without returning the actual resource.</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Asks the server about the capabilities of a specific resource or about the server in general.</td>
</tr>
<tr>
<td>POST</td>
<td>Allows the client to send information to a server for processing.</td>
</tr>
<tr>
<td>PUT</td>
<td>Requests that the enclosed entity be stored under the supplied URI.</td>
</tr>
<tr>
<td>DELETE</td>
<td>Requests that the server delete the specified resource.</td>
</tr>
<tr>
<td>TRACE</td>
<td>The TRACE method is used for debugging purposes at the application level.</td>
</tr>
<tr>
<td>CONNECT</td>
<td>This method is reserved for use with a proxy that can dynamically switch to being a tunnel.</td>
</tr>
</tbody>
</table>

Table 1 HTTP/1.1 Methods.

If the client tries to use a method that is not allowed on the specified resource, the server answers with the status code 405 (Method Not Allowed). If the method is unrecognized or not implemented by the server, it returns 501 (Not Implemented). The only methods that the server must understand are GET and HEAD. All the other methods are optional.

>>Request
OPTIONS /poweredby.gif HTTP/1.1
Host: dav.malmo.trab.se

>>Response
HTTP/1.1 200 OK
Date: Wed, 27 Oct 1999 14:43:17 GMT
Server: Apache/1.3.9 (Unix) DAV/0.9.12
Content-Length: 0
MS-Author-Via: DAV
Allow: OPTIONS, GET, HEAD, POST, DELETE, TRACE, PROPFIND, PROPPATCH, COPY, MOVE, PUT, LOCK, UNLOCK
DAV: 1,2
Content-Type: text/plain

Figure 2 OPTIONS example.

The OPTIONS method is used to examine which methods that are allowed on a HTTP server. Figure 2 is an example of its use. The client connects to port TCP 80 at the server. Then it sends the request message. The first line includes the method, which resource it wants to apply the method to, and the HTTP version. The next line is the Host header. It tells the server which Internet host it wants to talk to. In HTTP/1.1, it is possible to assign multiple web servers (and host names) to the same IP number. Instead the Host header separates them. The response from the server starts with the status line. The status code is 200 and expressed in text, it means "OK". Then follows the current date and time, information about the server and the length of the message body. Since the OPTIONS method only gives information in the response header, the content length is zero. The next header, Allow, contains the information, a list with all the methods that can be used. It seems that most of the methods found in Table 1 is acceptable. Some of these methods are DAV specific (PROPFIND, PROPPATCH, COPY, MOVE, LOCK and UNLOCK). These methods are examined in Chapter 3. The MS-Author-Via header is not part of the standard, but invented by Microsoft. It is used by their applications, where both FrontPage extensions and WebDAV are available. In this case, the header means that the server only supports WebDAV. The DAV header tells us which compliance
classes the resource supports. The last line is a Content-Type header. It specifies which kind of data is included in the body. This information is not relevant, since the OPTIONS method do not use the body.

```
>>Request
GET /poweredby.gif HTTP/1.1
Host: dav.malmo.trab.se

>>Response
HTTP/1.1 200 OK
Date: Wed, 27 Oct 1999 14:33:21 GMT
Server: Apache/1.3.9 (Unix) DAV/0.9.12
ETag: "22116-719-38144dcf"
Accept-Ranges: bytes
Content-Length: 1817
Content-Type: image/gif

... binary data ...
```

Figure 3 GET example.

In Figure 3, an example of the GET method is shown. This is probably the most commonly used method. The request specifies which resource is asked for and the host. The response contains a successful status code. The Last-Modified header indicates the date and time at which the server believes the resource was last modified. The meaning of the header depends on the type of the resource. For files, the file system’s last-modified time is often used. For entities with dynamic content, it may be the most recent of the set of last-modify times for its component parts. The ETag header contains the entity tag for the resource. An entity tag is used to indicate when a resource has been changed. It is used by the HTTP cache to discover whether its content has expired or not. The Last-Modified header is not enough, because it only counts in whole seconds. When using the entity tag, it must be unique for every resource on the server. It must also be changed when the resource is modified. The next header, Accept-Ranges, indicates that the server can deliver parts of files. One can, for example, request only 100 bytes of the file. The last headers, Content-Length and Content-Type, tell us that we are going to receive an image in gif format that is 1817 bytes.

2.1.3 Persistent connections and pipelining

Among the serious problems with HTTP/1.0 was the user demand for faster loading of pages. A common method to increase the loading speed was to make multiple connections to a single server. The high number of connections led to bandwidth and server overload at times. With persistent connections, the server does not automatically close the connection after replying to a client request. Instead the connection is open until the client or server explicitly requests that it should be closed. Persistent connections have several advantages:

- By using fewer TCP connections, CPU time is saved in routers and hosts (clients, servers, proxies, gateways, tunnels, or caches). Memory used for TCP protocol control blocks can be saved in hosts.
- HTTP requests and responses can be pipelined on a connection. Pipelining allows a client to make multiple requests without waiting for each response. This way, a single TCP connection can be used much more efficiently, with much lower elapsed time.
• Network congestion is reduced by reducing the number of packets caused by TCP socket opens, and by allowing TCP to use sufficient time to determine the congestion state of the network.

  Latency on subsequent requests is reduced since there is no time spent in TCP’s connection opening handshake.

### 2.1.4 Byte range operations

HTTP/1.1 allows a client to request that only a part of the given resource shall be included in the response. A common problem when downloading large files is that an interrupt in the connection forces you to start over again. If the file is large it is very time consuming to download it from the beginning, if only the last 10MB is wanted. Now, the user agent can just ask for the last 10MB of the resource instead of asking for the entire resource again. This can improve response time. When the client is requesting only a part of the resource, it makes a request as normal, but includes a `Range` header specifying the byte range the resource is to return. The client may also specify multiple byte ranges within a single request. In that case, the server returns the resource as a `multipart/byteranges` media type.

```
>>Request
GET /README HTTP/1.1
Range: bytes=500-999
Host: dav.malmo.trab.se

>>Response
HTTP/1.1 206 Partial Content
Date: Thu, 28 Oct 1999 09:59:43 GMT
Server: Apache/1.3.9 (Unix) DAV/0.9.12
Last-Modified: Thu, 28 Oct 1999 09:50:00 GMT
ETag: "22117-1efc2-38181c48"
Accept-Ranges: bytes
Content-Length: 500
Content-Range: bytes 500-999/126914
Content-Type: text/plain

... 500 bytes text ...
```

**Figure 4 Byte Range Request.**

In the example (Figure 4) is a request for 500 bytes of a larger file. The server responds in the `Content-Range` header that byte 500 - 999 of 126 914 is returned. It returns status code 206 (Partial Content).

### 2.1.5 Authentication

Access authentication is used to force a user agent to supply a password, if it wants to request a protected resource. In HTTP, access authentication can be done in two different ways. Basic authentication was introduced already in HTTP/1.0 [3], and the more secure approach, digest authentication is introduced with HTTP/1.1. Both methods are simple challenge-response mechanisms.

#### 2.1.5.1 Basic Authentication

Figure 5 is a request to GET a protected resource. The client requests a resource that is located at a protected space on the server. The server responds with status code 401 (Authorization Required) and includes the `WWW-Authenticate` header. This response is used to challenge the authorization of the client. We are asked to use “Basic” authentication. The realm value is a string, generally assigned by the server’s administrator. A combination of a realm value and an URL defines the protection space. These realms allow the protected resources on a server to be
partitioned into a set of protected spaces, each with its own authentication scheme or authorization database.

```
>>Request
GET /secret/Rebecka.jpg HTTP/1.1
Host: dav.malmo.trab.se

>>Response
HTTP/1.1 401 Authorization Required
Date: Thu, 28 Oct 1999 13:26:03 GMT
Server: Apache/1.3.9 (Unix) DAV/0.9.12
WWW-Authenticate: Basic realm="Secret resource..."
Transfer-Encoding: chunked
Content-Type: text/html

1de
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>401 Authorization Required</TITLE>
</HEAD><BODY>
<H1>Authorization Required</H1>
This server could not verify that you are authorized to access the document requested. Either you supplied the wrong credentials (e.g., bad password), or your browser doesn’t understand how to supply the credentials required.<P>
<HR>
<ADDRESS>Apache/1.3.9 Server at dav.malmo.trab.se Port 80</ADDRESS>
</BODY></HTML>
```

**Figure 5** Client tries to GET a protected resource but fails.

When the client receives the 401 status code from the server, it probably asks the user about user name and password. Then the client composes a string containing that information and sends it back to the server.

```
    Username : Password
```

**Base64 Encoded**

**Figure 6** Basic Authentication String.

Figure 6 shows how the authentication string is encoded. Base64 encoding is not encryption. Instead, it is a different way to encode characters [4]. The Base64 encoding is designed to represent arbitrary sequences of octets in a form that need not be humanly readable. A 65-character subset ([A-Za-z0-9+/=]) of US-ASCII is used, enabling six bits to be represented per printable character. This means that the user name and password are sent as clear text in the transmission. An example of the second try, with the Authorization header, is shown in Figure 7.
An evaluation of the WebDAV extensions to the HTTP protocol

Background

2.1.5.2 Digest Authentication

The other way to authenticate a user is with digest authentication. Like Basic Access Authentication, the Digest scheme is based on a simple challenge-response paradigm. The Digest scheme challenges with a nonce value.

Nonce is a server-specified data string that is uniquely generated each time a 401 (Authorization Required) response is made. This string is base64 encoded. The contents of the string depend on the implementation. An example implementation is described in [2] (See Figure 8). Here, time-stamp is a server-generated time, ETag is the value of the HTTP ETag header associated with the requested entity, and private-key is data known only to the server. With a nonce of this form a server would recalculate the hash portion after receiving the client authentication header and reject the request if it did not match the nonce from that header or if the time-stamp value is not recent enough. In this way the server can limit the time of the nonce's validity.

Figure 9 is an example of a user trying to GET a resource that demands digest authentication. The server responds with the WWW-Authenticate header and status code 401 (Authorization Required). Realm is the same thing as in Basic authentication. The ETag from the resource prevents replay request for an updated version of the resource. Algorithm is a string that indicates a pair of algorithms used to produce the digest and the checksum. By default the MD5 algorithm [5] is used and that is the only algorithm described in [2]. Domain is a quoted, space-separated list of URIs that defines the protection space. The qop directive is optional, but is included for backward compatibility with RFC 2069 [6]. It is a quoted string of one or more tokens indicating the "quality of protection" values supported by the server. The value "auth" indicates authentication.
An evaluation of the WebDAV extensions to the HTTP protocol

Background

Figure 9 Client tries to get protected resource and is asked for digest authentication.

After receiving the 401 (Authorization Required) status code, the client asks the user for username and password. Then, the answers are collected in an Authorization header. The new fields in the Authorization header are described below. The response field is a string of 32 hex digits, which proves that the user knows its password. It is composed in a couple of steps.

First, two help strings are concatenated; A1 and A2. If the "algorithm" directive's value is "MD5" or is unspecified, then A1 looks as in Figure 10. If the directive's value is "MD5-sess", then A1 is calculated only once. It happens on the first request by the client following receipt of a WWW-Authenticate challenge from the server. Then, A1 contains a hash of what is shown in Figure 10 concatenated with nonce and cnonce (cnonce is described below). This creates a session key for the authentication of subsequent requests and responses, which is different for each authentication session.

If the "qop" directive is "auth" or is unspecified, then A2 contains the information shown in Figure 11. Is the value "auth-int" instead, A2 also contain the hash of the messages' body. The "Method" value is the HTTP request method (for
example: `GET`). The "Uri" value is the request URI from the request line. This may be "*", an absolute URL or an absolute path. It must agree with the request URI. When A1 and A2 are created, all the components needed in the response field are generated (shown in Figure 12).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5 Hash</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MD5 Hash</td>
</tr>
</tbody>
</table>

**MD5 Hash**

**Figure 12 Response field.**

The next field, `cnonce`, is an opaque quoted string value provided by the client and used by both client and server to avoid chosen plaintext attacks, to provide mutual authentication and to provide some message integrity protection. Nc (or nonce-count) is the hexadecimal count of the number of requests that the client has sent with the nonce value in this request. The purpose of this directive is to allow the server to detect request replays by maintaining its own copy of this count - if the same nc-value is seen twice, then the request is a replay.

```plaintext
>>Request
GET /top_secret/Climbing.jpg HTTP/1.1
Host: dav.malmo.trab.se
Authorization: Digest realm="Top Secret resource",username="bjorn",uri="/top_secret/Climbing.jpg",nonce="X KAZOA==d6f57266a0e6088ef53d3d7147fa4200f252157a",response="be99fbc c4a885fc038c58d82453cde5db",algorithm="MD5",cnonce="61cb2dc2f55dd49 c804d134d48ba2259",qop="auth",nc="00000003"

>>Response
HTTP/1.1 200 OK
Date: Fri, 29 Oct 1999 13:28:05 GMT
Server: Apache/1.3.9 (Unix) DAV/0.9.12
Authentication-Info: rspauth="673825ee59964728015182008def8262",cnonce="61cb2dc2f55dd49c804d134d48ba2259",nc="00000003",qop="auth"
Last-Modified: Fri, 29 Oct 1999 09:57:34 GMT
ETag: "5a0b0-39f14-38196f8e"
Accept-Ranges: bytes
Content-Length: 237332
Content-Type: image/jpeg

... Binary data ...
```

**Figure 13 Retrieval of resource with digest authentication.**

A request with a valid `Authorization` header is shown in Figure 13. Now, the server replies with another new header, `Authentication-Info`.

The server uses this header to communicate some information regarding the successful authentication in the response. The “rspauth” field is an optional response digest. It is used by the server to show that it knows the user’s secret. The value is basically calculated in the same way as for the “response” field in the `Authorization` header. Finally, the `cnonce` and `nc` values are the same as the client sent in its request.
2.1.6 Integrity

Integrity is achieved by using Digest Authentication. When the client makes a request for a protected resource, the server includes the WWW-Authenticate header in the response. One of the fields in that header is qop, "quality of protection". If the server indicates that "auth-int" is a valid value, then authentication with integrity protection is feasible.

As described earlier, when "qop=auth-int", the A2 field will be calculated in a different way.

2.1.7 Confidentiality

The common way today to achieve authentication, integrity and confidentiality is by using the Secure Sockets Layer (SSL) protocol [7]. Netscape Communications Corporation developed the protocol in 1994. An updated version, SSL 3.0, was released 1996. SSL uses encryption and decryption to ensure that data is transmitted securely. This is done with public key encryption from RSA [8]. The server contains a private and public key received once from a Certificate Authority. A Certificate Authority (CA) is an organization, which issues keys to companies. When a client connects to a server, it receives its certificate (with information about the company and its public key). Then the server and the client use public key encryption during the SSL handshake. In that procedure, the client authenticates the server and they decide which cryptographic methods to use and which session keys. After the handshake is completed, they switch to the more efficient symmetric key encryption for the reminder of the transaction. Symmetric key encryption is done with a single key, a shared secret. That key is used to both encrypt and decrypt data. A different session key is used for each transaction between a client and a server. The session key automatically expires after 24 hours.

The Transport Layer Security (TLS) Protocol [9] is a successor to SSL. The differences between TLS 1.0 and SSL 3.0 are not dramatic, but they are significant enough that the protocols do not interoperate. The large difference is instead that TLS is a result of an effort in an IETF Workgroup and soon will become an Internet Standard. SSL is the best current practice on the Internet, but a standard developed Netscape. Another IETF Workgroup, the Web Transaction Security WG, is working on a specification for the provision of security services to web transactions using HTTP. In August 1999, they released an Experimental RFC [10], called "The Secure HyperText Transfer Protocol" (S-HTTP). S-HTTP is designed to be intermixed on the same TCP port (80) as an ordinary HTTP server. They have also proposed to extend the HTML tags to support the protocol [11]. The protocol uses the HTTP protocol to negotiate security information and allows the user to choose the security specifications it wants for each transaction between client and server. It provides signatures that can be archived for later retrieval to support non-repudiation services. The encrypted objects can also be archived for later decryption.

All confidentiality methods discussed so far is about the transport layer. It is also possible to protect the data at the network layer. Such an approach is the IP Security Protocol [12]. The IPSEC Protocol ensures secure private communication over IP networks. IPSEC parameters between two devices are negotiated with the Internet KeyExchange [13]. IKE can use digital certificates for device authentication to enable the creation of large encrypted networks. It also allows security to be embedded at the network layer. IPSEC implements network layer encryption and authentication providing an end-to-end security solution in the network architecture itself. This means that the end systems and applications do not need changes in order to take advantage of strong security.
2.2 FTP – File Transfer Protocol

FTP is the common protocol for file transfers over the Internet. It is an Internet Standard from 1985. The purpose of the protocol was to make it possible to share files between computers and to encourage use of remote computers for different purposes.

2.2.1 Overview

The FTP service can be shown in a diagram (Figure 14) [15]. In the model, the client system initiates the control process. The control process connects to the server with the Telnet protocol [14]. At the initiation of the user, standard FTP commands are sent from the client’s control process to the server’s control process. Standard replies are sent from the server process to the client. As in HTTP, the server replies with a status code of three digits and a text string to be read by humans. Some common reply codes are: 200 (Command okay), 500 (Syntax error; command unrecognized) and 502 (Command not implemented). The FTP commands specify the parameters for the data connection (data port, transfer mode, representation type, and structure). It also specifies the nature of file system operation (store, retrieve, append, delete, etc). The client’s Data Transfer Process should listen on the specified data port, and the server should initiate the data connection and data transfer in accordance with the specified parameters. The data port that is used for the data transfer do not have to be on the same host that initiates the FTP commands via the control connection, but the client and its user must ensure that someone listen on the specified data port.

It is also feasible to transfer files between two hosts, neither of which is a local host. Then, the user has to set up control connections to the two servers and then arrange for a data connection between them.

The protocol requires that the control connections be open while data transfer is in progress. It is the responsibility of the user to request the closing of the control connections when finished using the FTP service, while it is the server who takes the action. The server may abort data transfer if the control connections are closed without any command.

An example of an FTP session is shown in Figure 15. A user makes an anonymous login to an FTP server and retrieves a file.
An evaluation of the WebDAV extensions to the HTTP protocol

Background

```
ftp> open dav.malmo.trab.se
Connected to dav.malmo.trab.se.
220 malmo.trab.se FTP server (Version wu-2.5.0(1) Tue Aug 10 17:10:42 MDT 1999) ready.
Name (dav.malmo.trab.se:root): anonymous
--- USER anonymous
331 Guest login ok, send your complete e-mail address as password.
Password:
--- PASS XXXX
230 Guest login ok, access restrictions apply.
--- SYST
215 UNIX Type: L8
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd pub
--- CWD pub
250 CWD command successful.
ftp> ls
--- PORT 131,115,49,103,4,9
200 PORT command successful.
--- LIST
150 Opening ASCII mode data connection for /bin/ls.
total 154
drwxr-xr-x   6 root     root         1024 Nov 10 15:54 ..
-rw-r--r--   1 root     ftp         56507 Nov 12 09:24 ftpd.ps
-rw-r--r--   1 root     ftp         65892 Nov 12 09:24 ftpaccess.ps
-rw-r--r--   1 root     ftp         27689 Nov 12 09:24 ftpd.ps
226 Transfer complete.
ftp> bin
--- TYPE I
200 Type set to I.
ftp> get ftp.ps
local: ftp.ps remote: ftp.ps
--- PORT 131,115,49,103,4,10
200 PORT command successful.
--- RETR ftp.ps
150 Opening BINARY mode data connection for ftp.ps (56507 bytes).
226 Transfer complete.
56507 bytes received in 0.0884 secs (6.2e+02 Kbytes/sec)
ftp> close
--- QUIT
221-You have transferred 56507 bytes in 1 files.
221-Total traffic for this session was 57995 bytes in 2 transfers.
221-Thank you for using the FTP service on malmo.trab.se.
221 Goodbye.
ftp> bye
```

![Figure 15 FTP Transfer.](image)

### 2.2.2 Commands

When the client has established a control connection to the server, it can send different commands. The commands are divided into three categories, depending on what they do. The categories are Access Control Commands, Service Commands and Transfer Parameter Commands.

#### 2.2.2.1 Access Control Commands

The first category, Access Control Commands, is about authentication and navigation in the directory tree. When a user connects to an FTP server, it starts by sending the `USER` and `PASS` commands, with its login name and password. Without a correct login, the user cannot access the server's file system. The category also contains two commands to navigate the file system – `CDW` and `CDUP`. `CDW` changes the working directory and `CDUP` changes to the parent directory. An optional command, `SMNT`, allows the user to mount a different file system data structure on the server. Commands to create a secure connection are also found here. These are described in RFC 2228 [16]. Currently, these are not implemented in the common FTP daemons, due to export restrictions on cryptographic software.
2.2.2.2 Service Commands

The FTP Service Commands define the file transfer or the file system function requested by the user. These commands may be used in any order except that a "rename from" (RNFR) command must be followed by a "rename to" (RNTO) command. The data transferred in response to the commands, is always sent over the data connection, except for certain informative replies. With the commands, a user can retrieve files (RETR) and directory listings (LIST/NLST). It is possible to store files on the server (STOR/STOU) and append to existing files (APPE). Remote file management can be done by creating directories (MKD) and deleting directories (RMD) and files (DELE). One can get information about the server’s operating system (SYST) and the operation in process (STAT).

The restart command (REST) is used to continue an interrupted transfer. It must be followed by the interrupted service command (e.g. STOR or RETR). Another important issue about the REST command is that it is only defined for the block and compressed modes of data transfer. These modes have markers to keep track of what has been sent. The same functions do not exist in stream mode. Even if the restart function for stream mode has been undocumented for a long time, it is commonly implemented in FTP servers around the world. A formal description is found in an Internet-Draft [17].

2.2.2.3 Transfer Parameter Commands

The Transfer Parameter Commands are used for transfer of data between hosts. They specify the format of the data sent via the data connection. Many of the parameters that were defined in RFC 959 are obsolete on the modern Internet. Usually, they are unimplemented in existing ftp daemons. The PORT command is used to specify the IP number and port of the receiver. If the server should listen on a data port and wait for a connection, the PASV command is used. The format of the data to be sent is modified by the representation type (TYPE), file structure (STRU) and transfer mode (MODE). Among the representation types, ASCII and binary (Image) are the common alternatives. ASCII is used for directory listing and navigation through the file system, while binary is used for file retrieval. The file structure alternative is usually set to File. The other options (Record and Page) are seldom used. Transfer modes specify how the data is formatted before it is sent through the data channel. Stream mode is the default and the most simple. The data is transmitted as a stream of bytes. Other alternatives are Block mode (the data is divided into blocks with headers) and Compressed mode (the data is compressed before it is transmitted).

2.2.3 Authentication

To access the content on an FTP server, the user needs to supply a username and password. A common form of login is anonymous access. The user uses the username “anonymous” and sends its email address as password. The usual limitation here is the number of open connections the server can handle. The next level of security is when every user that is allowed to access the server has its own username and password. When the user login, it sends these parameters with the USER and PASS commands. The password is sent in clear text. The FTP Standard supports another type of login, with the ACCT command. With this approach, each user needs a special account on the server (besides username and password). It can also be used in special cases, for example when the user wants to save files on the server. The ACCT command is seldom used on the Internet nowadays.

With the FTP Security Extensions [16], the authentication takes place in a couple of steps. First, the client tells the server what security mechanism it wants to
use with the \texttt{AUTH} command. The server will either accept this, or reject it in some way. If the server does not completely reject the request, the client can try again with other arguments (this is a rudimentary form of negotiation). Depending on the reply codes from the server, the client might be forced to send additional data for the security mechanism to interpret. These are sent with the \texttt{ADAT} command. During this phase, the authentication process is eventually performed. It depends on which security mechanism that is used. Irrespective of the mechanism, a username is required.

\subsection{Integrity}

In ordinary FTP (according to RFC 959), integrity is not supported. But with the Security Extensions, it might be possible. It depends on the security associations negotiated between the client and the server. Some security mechanisms support integrity, some do not. If the chosen mechanism supports it, integrity protection must be used on the control channel as soon as the negotiations are finished. It is possible to turn off the protection of the control channel with the \texttt{CCC} command. When commands are sent integrity protected, then the command to be executed shall be protected and base64 encoded. The command \texttt{MIC} is used to send the base64 encoded command string to the server. If both integrity and confidentiality is needed, then instead the \texttt{ENC} command shall be used (in the same way).

To set the integrity level for the data channel, the client uses the \texttt{PROT} command. As default, the data channel is clear (i.e. without any protection). If the protection level is set to Safe, then the channel will be integrity protected. Another alternative is to set the protection level to Private. Then both confidentiality and integrity protection will be applied to the data transfer.

\subsection{Confidentiality}

The Internet Standard about FTP does not mention confidentiality issues. In the RFC about FTP Security Extensions, some new commands to handle confidentiality are presented. When the security mechanisms for the session are specified, the client and the server determine the size of the data blocks (with the \texttt{PBSZ} command). The protection level for the data channel is set with the \texttt{PROT} command. If it is set to Confidential, then confidentiality protection will be enabled. If both confidentiality and integrity protection is desired, then the protection level should be set to Private.

To achieve confidentiality on the command channel, commands shall be encrypted, base64 encoded and sent with the \texttt{CONF} command. If integrity is also wanted, the \texttt{END} command shall be used instead (in the same manner).

It could be possible to use both SSL and TLS to achieve confidentiality. These protocols are independent of the application protocol (i.e. ftp). This is not implemented in any ftp daemons yet. Also IPSEC is possible to use to secure FTP sessions.

\section{POP3 – Post Office Protocol}

The Post Office Protocol - Version 3 [18] is intended to allow a workstation to dynamically retrieve mail from a server. The protocol is not intended to provide extensive manipulation operations of mail on the server. It is only developed for simple operations such as downloading mail and deleting them from the server.

\subsection{Overview}

A server running the POP3 service listens to TCP port 110. When a client wants to use the service, it connects to that port. The server starts the conversation
by sending a greeting. The client and server then exchange commands and responses until the connection is closed or aborted. An example of a POP3 session is shown in Figure 16.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STAT</td>
<td>Returns number of messages and their size in the mailbox.</td>
</tr>
<tr>
<td>LIST</td>
<td>Gives information about each message. The number of the message is given followed by its size. If a number is given as argument to the command, only the chosen message is listed. The response ends with a &quot;.&quot;.</td>
</tr>
<tr>
<td>RETR</td>
<td>Retrieves a specified message.</td>
</tr>
<tr>
<td>DELE</td>
<td>Marks a message as deleted. Any future references to the message in the session results in an error. The server doesn’t delete the message until the Update state.</td>
</tr>
<tr>
<td>NOOP</td>
<td>The server does nothing, but replying with a positive response.</td>
</tr>
<tr>
<td>RSET</td>
<td>If any messages have been marked as deleted by the server, they are unmarked.</td>
</tr>
</tbody>
</table>

Table 2 POP3 Transaction commands.
2.3.3 Security issues

The authentication process in POP3 happens in the Authorization state. In the POP3 RFC [18], two methods are presented. The simplest method is authentication with \texttt{USER/PASS}. Both username and password is sent in clear text.

Another method is to send the \texttt{APOP} command. A server that implements the \texttt{APOP} command includes a timestamp in its greeting message. On a UNIX system, the syntax of the timestamp might be: \texttt{<process-ID.clock@hostname>}, where “process-ID” is the decimal value of the process’s PID, clock is the decimal value of the system clock, and hostname is the fully-qualified domain-name corresponding to the host where the server is running. The client issues the \texttt{APOP} command together with its username and a digest. The digest is a MD5 digest of the timestamp concatenated with the user’s password.

An optional command that is used for authentication is \texttt{AUTH} [19]. If the server accepts the command, it is possible to use the Simple Authentication and Security Layer (SASL) mechanism [20]. SASL is a method for adding authentication support to connection-based protocols. The client and server negotiate the use of a security layer. If they use it, the layer is inserted between the protocol and the connection. Then integrity and confidentiality is achieved.

The TLS protocol can also be used with POP3 [21]. If the server supports the \texttt{STLS} command, it is possible to use TLS to handle authentication, integrity and confidentiality.
WebDAV is an extension to HTTP/1.1, which allows clients to perform remote web content authoring operations. It encompasses new methods, headers and entity body formats for requests and responses. DAV is an abbreviation for “Distributed Authoring and Versioning”. The core functionality of the protocol is described in RFC 2518; “HTTP Extensions for Distributed Authoring – WebDAV” [22]. The core encompasses locking, properties and namespace management. In the requirements document for WebDAV [23], versioning is described as an important part. Originally versioning was supposed to be included in the WebDAV RFC, but instead it was transferred to a new IETF working group, Delta-V [24]. The goals of the Delta-V Working Group are to extend WebDAV with versioning and configuration management. Another IETF Working Group is working on a closely related topic; DAV Searching and Locating (DASL) [25]. The goals are to create a HTTP-based search protocol to transport queries and result sets. This allows clients to make use of server-side search facilities. Neither Delta-V nor DASL has finished their work yet. A deeper review of their work, based on their Internet-Drafts, can be found below in the “Work in Progress” section.

A difference between WebDAV methods and HTTP/1.1 methods are the way parameter information is encoded. In HTTP/1.1, this is done in HTTP headers. WebDAV on the other hand, encodes the information either in an Extensible Markup Language (XML) [26] request entity body, or in an HTTP header. The advantages by using XML are the ability to add extra XML elements to existing structures, and by XML’s ability to encode information in a way that provides internationalization support. Usually, parameters are encoded in XML bodies when they have unbounded length, or when they may be shown to a human user. Otherwise, parameters are encoded within HTTP headers. XML is also used in WebDAV to encode the responses from methods.

### 3.1 Overwrite prevention

A usual scenario, when discussing the possibilities with WebDAV is when two people are editing a web site together. Suppose that they sit in different locations and simultaneously decide to edit the same web page. With the technique of today, they probably would use HTTP and the GET method to load the page into their HTML editor. When their work is finished, they would save it with HTTP’s PUT method, or by using FTP. FTP is harder to use, since the user needs to log on to the FTP server, find the correct directory and finally STOR the file. The problem is not that it can be complicated to store the document. Instead it is a large risk that one person will overwrite the other’s work. This is called the “lost update” problem. WebDAV solves these issues by introducing write locks. Two new methods are used to handle locks: LOCK and UNLOCK.

There are two different types of locks; exclusive and shared. An exclusive lock is where the access rights are only granted to a single user. When using this lock, there is no need to merge work from users. However, sometimes the purpose of a lock is not to exclude others from using their access rights. Instead, it can be used as a mechanism to show that someone intends to work with a resource. Here, it is more appropriate with shared locks. A shared lock allows multiple users to receive a lock. The working group has to coordinate their work in a different manner (for example: email, telephone conversation, etc). The intent of a shared lock is to let collaborators know who else may be working on a resource. The locks are included, because experience from web distributed authoring systems has shown that exclusive locks are too rigid. When using exclusive locks, one tries to enforce a certain behavior:
acquire an exclusive lock, read the resource, edit it, write the resource, release the lock. The problem with this process is that locks are not always properly released. When a program crashes or when a lock owner leaves without unlocking a resource, the other participants in the working group cannot write to the resource. While both timeouts and administrative action can be used to remove an offending lock, neither mechanism may be available when needed.

The lifecycle of a lock is shown in Figure 17. To lock a resource, the LOCK method is used. The method is also used when a user already has locked a resource, but wants to keep it locked for a longer time. The locked resource can be unlocked in three different ways. Explicit when a user sends an UNLOCK request, or when the system administrator removes the lock. It can also be unlocked when the timeout value reaches zero.

```
>>Request
LOCK /TeamWork/Readme.txt HTTP/1.1
Host: dav.malmo.trab.se
Timeout: Second-86400
Depth: infinity
Content-type: text/xml
Content-length: 276

<?xml version="1.0"?>
<A:lockinfo xmlns:A="DAV:"
   <A:locktype><A:write/></A:locktype>
   <A:lockscope><A:exclusive/></A:lockscope>
</A:lockinfo>
```

```
>>Response
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Thu, 25 Nov 1999 14:53:36 GMT
Content-Type: text/xml
Content-Length: 455

<?xml version="1.0"?>
<a:prop xmlns:a="DAV:"
   <a:lockdiscovery>
      <a:activelock>
         <a:locktype><a:write/></a:locktype>
         <a:lockscope><a:exclusive/></a:lockscope>
         <A:owner xmlns:A="DAV:">
         </A:owner>
         <a:locktoken>
         </a:locktoken>
      </a:activelock>
   </a:lockdiscovery>
</a:prop>
```

![Figure 17 State Diagram for locks.](image-url)
An evaluation of the WebDAV extensions to the HTTP protocol

WebDAV

The LOCK request looks as in Figure 18. The client specifies which resource to lock and the name of the server. The Timeout header specifies how long timeout the client wants. If this is not specified, the server assigns a default value. Clients must assume that locks may disappear at any time, regardless of the value given. For example, an administrator may remove a lock at any time or the system may crash in such a way that it loses the record of the lock's existence. The next header, Depth, specifies that resource along with all its internal members all the way down the hierarchy are to be locked. In the body of the request is XML information about the lock that the client wants. The DAV:lockinfo contains three elements: DAV:locktype specifies which kind of lock the client wants. At present, only the write lock is defined. The next element, DAV:lockscope, specifies the type of the lock, in this case exclusive. Finally is some information about the user. The server replies with status code 200 (OK). Included in the XML body is the DAV:prop element. It is a container for properties defined on resources, in this case properties on the locked file. The DAV:lockdiscovery property contains a listing of who has a lock (DAV:owner), what type of lock he has (DAV:locktype and DAV:lockscope), the timeout type and the time remaining on the timeout (DAV:timeout), and the associated lock token (DAV:locktoken).

The lock token must be unique across all resources for all time. That allows lock tokens to be submitted across resources and servers without fear of confusion. The lock token in the example begins with the string "opaquelocktoken". This means that it is created in a special way to ensure that it is unique. Having a lock token provides no special access rights. Anyone can find out anyone else's lock token by looking at the DAV:lockdiscovery property. Locks must be enforced based upon the authentication mechanism used by the server, not based on the secrecy of the token values.

>>Request
MOVE /TeamWork/Readme.txt HTTP/1.1
Host: dav.malmo.trab.se
Overwrite: F
Destination: http://draco/TeamWork/Docs/Readme.txt
If: (<opaquelocktoken:A398A421-0720-45EA-BC05-AF19C74DE7B4:214748364855>)
Content-Length: 0

>>Response
HTTP/1.1 201 Created
Server: Microsoft-IIS/5.0
Date: Mon, 29 Nov 1999 10:31:16 GMT
Location: http://draco/TeamWork/Docs/Readme.txt
Content-Type: text/xml
Content-Length: 0

Figure 19 Working with locked resources.

When a resource is locked, the lock token must be included when modifying the resource. This is done in the If header. In Figure 19, a locked resource is moved to a new location. The action is successfully completed.

>>Request
UNLOCK /TeamWork/Readme.txt HTTP/1.1
Host: dav.malmo.trab.se

>>Response
HTTP/1.1 204 No Content
Server: Microsoft-IIS/5.0
Date: Thu, 25 Nov 1999 14:54:06 GMT
Content-Length: 0

Figure 20 UNLOCK example.

To unlock a resource, the client use the UNLOCK method (see Figure 20). After the method name, the resource is specified. The Lock-Token header is
included in the request with the lock token received when locking the resource. In
the response, the status code 204 (No Content) is used instead of 200 (OK) because
there is no response entity body.

3.2 Properties

In another scenario, a person is creating a new document on the Web. The
person sends it to the server, but also wants to set some attributes for the document
(author, title, type of document, subject, etc). These attributes can be used later in
searches. Sometimes she may also want to create catalog entries for documents that
are not available in electronic form. Then, there will be no content for these
documents, just attributes.

In WebDAV, the attributes are called properties. The properties are data about
data and describe the state of a resource. The WebDAV property model consists of
(name, value) pairs. The name of a property identifies the property's syntax and
semantics. It also provides an address by which to refer to its syntax and semantics.

There are two types of categories: “live” and “dead”. A “live” property has its
syntax and semantics enforced by the server. It includes cases where the value of a
property is read-only and maintained by the server, or where the value is maintained
by the client but the server performs syntax checking on submitted values. Some
examples of live properties are shown in Table 3. A “dead” property has its syntax
and semantics enforced by the client. The server only keeps track of its value.

<table>
<thead>
<tr>
<th>&quot;Live&quot; property</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAV:creationdate</td>
<td>Records the time and date the resource was created.</td>
</tr>
<tr>
<td>DAV:displayname</td>
<td>Provides a name for the resource that is suitable for presentation to a user.</td>
</tr>
<tr>
<td>DAV:getcontentlength</td>
<td>Contains the Content-Length header returned by a GET without accept headers.</td>
</tr>
<tr>
<td>DAV:getcontenttype</td>
<td>Contains the Content-Type header returned by a GET without accept headers.</td>
</tr>
<tr>
<td>DAV:getlastmodified</td>
<td>Contains the Last-Modified header returned by a GET method without accept header.</td>
</tr>
</tbody>
</table>

Table 3 Live Properties.

The name of a property is a universally unique identifier that is associated with
a schema that provides information about the syntax and semantics of it. To make
the name unique, XML’s namespace mechanism is used [27]. The property
namespace is flat, which implies that no hierarchy of properties is possible. It is not
possible to define the same property twice on a single resource, as this would cause a
collision in the resource’s property namespace. The value of a property is expressed
as well formed XML code.

To deal with properties, WebDAV contains two new methods: PROPFIND
and PROPPATCH. PROPFIND retrieves properties from a resource and
PROPPATCH changes them.

A PROPFIND request is found in Figure 21. The method retrieves properties
declared on the resource identified by the Request-URI. The Depth header that is
included in the request specifies how to treat collections. The value “0” specifies that
only the named resource shall be included in the response. In the entity body, the
client specifies which properties it wants. Here, all the properties are wanted.
An evaluation of the WebDAV extensions to the HTTP protocol

WebDAV

Figure 21 PROPFIND example.

In the response, the status code is 207 (Multi-Status). Then follows a DAV:response XML element for each resource, if it is a collection. Since the request only wanted properties from one resource, we have only one DAV:response element. The following properties are defined on the resource: DAV:getcontentlength, DAV:creationdate, DAV:displayname, DAV:getetag, DAV:getlastmodified, DAV:resourcetype, DAV:supportedlock, DAV:ishidden, DAV:iscollection and DAV:getcontenttype. In the response a namespace called b is used. This URN is used for defining data type for searchable resources. It is used on servers that understand DASL (see 3.4.3 Searching and Locating).
To change, set or remove properties, the **PROPPATCH** method is used (see Figure 22). The request body must contain the **DAV:propertyupdate** element. Inside that element all **DAV:set** and **DAV:remove** elements should be. All these elements that are included must be successfully processed, or the changes will be undone. In the example, the following properties are set:

- [http://draco.trab.se/props/Author](http://draco.trab.se/props/Author)
- [http://draco.trab.se/props/Title](http://draco.trab.se/props/Title)

The response is the status code 207 (Multi-Status). It also contains a list of the new properties on the resource.

### 3.3 Name space management

When working with distributed resources on a remote server, it is necessary with bookkeeping functions, rather similar to the own operating system. It should be possible to create directories, delete, copy and move files.

On a web server, the HTTP URL namespace model is used. It is a hierarchical namespace where the hierarchy is delimited with the "/ " character. The namespace is similar to a file system on an ordinary computer. It has the same hierarchical structure. One important difference is the terminology that is used. In a file system, the components are called directories and files, but in the HTTP namespace model it is called collections and resources. The reason for this terminology is that the server might not be using directories and files to store the resources that can be accessed on the server. It can fetch the data from databases, be a document management system or some kind of repository.
The formal definition for a collection is a “resource whose state consists of at least a list of internal member URIs and a set of properties, but which may have additional state such as entity bodies returned by GET” [22]. To create a collection, the MKCOL method is used. It should be used instead of the existing HTTP PUT or POST method. The PUT method is a bad choice, because if a description of a collection that omitted some existing resources were PUT to a server, this might be interpreted as a command to remove those members. This would change the semantics of PUT to also perform DELETE functionality. POST is also a bad choice. It is easy to create a POST command, which creates a collection but then it would be difficult to separate access control for collection creation from other uses of POST.

>>Request
MKCOL /TeamWork/New%20Folder HTTP/1.1
Host: draco
Content-Length: 0

>>Response
HTTP/1.1 201 Created
Server: Microsoft-IIS/5.0
Date: Mon, 29 Nov 1999 10:30:58 GMT
Location: http://draco/TeamWork/New%20Folder/
Content-Length: 0

Figure 23 MKCOL example.

To handle the file management with WebDAV, two new methods are added: COPY and MOVE. The HTTP/1.1 methods PUT and DELETE are also used to upload and delete files.

>>Request
COPY /TeamWork/Rebecka.jpg HTTP/1.1
Destination: http://draco/TeamWork/New%20Folder/Rebecka.jpg
Overwrite: F
Host: draco
Content-Length: 0

>>Response
HTTP/1.1 201 Created
Server: Microsoft-IIS/5.0
Date: Mon, 29 Nov 1999 10:31:30 GMT
Location: http://draco/TeamWork/New%20Folder/Rebecka.jpg
Content-Type: text/xml
Content-Length: 0

Figure 24 COPY example.

The COPY method creates a duplicate of the source resource, identified by the Request-URI, in the destination resource (see Figure 24). The destination, where the resource is to be copied is specified with the Destination header. The exact behavior of the COPY method depends on the type of the source resource. When the resource to be copied is not a collection, the result of the COPY method is the creation of a new resource at the destination. Then all properties from the source are copied to the destination. When a collection is copied, the result depends on the Depth header. If the header is not included, the request is treated as “Depth: Infinity”. In that case, the collection is copied to the destination and all its internal member resources are copied to a location relative to it, recursively through all levels of the collection hierarchy. In the other case, “Depth: 0”, only the collection and its properties but not resources identified by its internal member URIs are copied. The Overwrite header specifies if the request should overwrite the destination, if it already exists. The valid values are T (true) and F (false). In the example, it is set to false. The status code from the server, 201 (Created), indicates that the resource was successfully copied.

The MOVE method is similar to COPY. When applied on a resource, the resource is copied to the new location and then deleted from the previous location.
The operations are performed atomically. The server also performs a consistency maintenance step, which perform updates caused by the move, such as updating all URIs other than the Request-URI which identify the source resource, to point to the new destination resource. A MOVE on a collection works as a “Depth: Infinity” header was applied. The server tries to move all internal members in the collection. The MOVE method also uses the Overwrite header, to decide whether to overwrite the destination or not. In Figure 25, a MOVE example is shown. The server responds with 201 (Created) to show that the resource has been properly moved.

```
>>Request
MOVE /TeamWork/Tendens.doc HTTP/1.1
Overwrite: F
Destination: http://draco/TeamWork/New%20Folder/Tendens.doc
Host: draco
Content-Length: 0

>>Response
HTTP/1.1 201 Created
Server: Microsoft-IIS/5.0
Date: Mon, 29 Nov 1999 10:31:16 GMT
Location: http://draco/TeamWork/New%20Folder/Tendens.doc
Content-Type: text/xml
Content-Length: 0
```

Figure 25 MOVE example.

### 3.4 Work in progress

All the feasibility that people see with WebDAV is not yet finished. Many important functions are still under development, for example the versioning support that was intended to be part of the original document. This section is an overview of the parts of WebDAV that at present only exists in a couple of Internet-Drafts. Some of them have expired (see the References chapter), but the topics are still important and the work will continue.

#### 3.4.1 Advanced Collections

WebDAV provides basic support for collections, which makes it possible to create and list unordered collections. Since many applications need more powerful collections, extensions on collections are under development. The requirements document [28] talks about extending the functionality in two different directions: resource sharing and collection ordering.

##### 3.4.1.1 Ordered Collections

In many cases, it is convenient for clients to be able to impose orderings on collections at the server [29]. It can, for example, be a scenario where a manufacturing company develops and maintains its product maintenance manuals on the Web. There is a separate collection for each product manual. The sections of each manual may be ordered so that they can be printed together as a book. A collection ordering may sometimes be based on property values, for example an ordering that is alphabetical by author’s last name. An ordering can also be custom. A teacher may order a collection of course readings in the sequence that makes sense to coordinate them with his lectures. There is an important distinction between server-maintained and client-maintained orderings. A server-maintained ordering is when the server enforces the semantics of the ordering by placing each collection member at the appropriate position in the ordering. The client cannot change it. In client-maintained orderings, the client places each resource in the ordering based on its understanding of the semantics of the ordering. The server does not validate it.
An evaluation of the WebDAV extensions to the HTTP protocol

When a collection is created, the client decides if it should be an ordered collection or an ordinary unordered collection. An ordered collection is created with the MKCOL method and the new Ordered header (See Figure 26). The Ordered header can take three different values: DAV:custom, DAV:unordered and an URI. Using DAV:unordered gives the same result as not using the Ordered header at all, that is an ordinary unordered collection is created. If a URI is specified it might identify a server-maintained ordering. It may also specify a semantics for a client-maintained ordering, which provides information to a human user, who need to insert a new member in an intelligent manner. The value can also be set to DAV:custom, to indicate that the collection is ordered, but the semantics of the ordering is not advertised.

>>Request
MKCOL /Order/ HTTP/1.1
Host: draco
Ordered: DAV:custom

>>Response
HTTP/1.1 201 Created

Figure 26 Creating an ordered collection.

When a new resource is added to a collection with client-maintained ordering, its position can be set with the new Position header. That header allows the client to specify that the resource should be first, last, before or after, related to the other resources in the collection. After a collection is created, it is possible to change the ordering of the resources. This is done with the new ORDERPATCH method. If one has an ordered collection with the following resources...

1. Introduction
2. WebDAV
3. Background

... and wants to move the second resource last in the collection, it can be done with the ORDERPATCH request described in Figure 27.

>>Request
ORDERPATCH /Exjobb/ HTTP/1.1
Host: draco
Content-Type: text/xml
Content-Length: 155

<?xml version="1.0" ?>
<d:order xmlns:d="DAV:"
  <d:ordermember>
    <d:href>WebDAV</d:href>
    <d:position>
      <d:last/>
    </d:position>
  </d:ordermember>
</d:order>

>>Response
HTTP/1.1 207 Multi-Status
Content-Type: text/xml
Content-Length: 178

<?xml version="1.0" ?>
<d:multistatus xmlns:d="DAV:"
  <d:response>
    <d:status>HTTP/1.1 200 OK</d:status>
  </d:response>
</d:multistatus>

Figure 27 Working with ordered collections.
Ordered collections are a new compliance class in WebDAV (see 3.5 WebDAV Servers). If the server supports the function, a response to an OPTIONS request on a collection can include the DAV header with the value orderedcoll.

### 3.4.1.2 Shared Resources

Associations make it possible to share resources. A resource can be physically located on one server and be pointed at from different locations on the same server or on other servers. Since only one physical copy of the resource need to exist, changes made to it are visible from all other collections that share it. An example of the kind of sharing is a mathematics department at one university that wants a collection of information on fractals. It contains some local resources, but also accesses resources at several other universities. For many reasons, to conserve disk space or to respect copyright constraints, it may be undesirable to make physical copies of the shared resources on the local server.

WebDAV has two solutions to address this problem: Redirect Reference Resources [30] and Bindings [31]. A redirect reference resource is a resource in a collection that responds to most requests by redirecting the requests (using an HTTP/1.1 302 Moved Temporary response) to a different resource, possibly in a different collection. Bindings are a different approach, where mappings of URIs to resources are created.

The two approaches have very different characteristics. A redirect reference is a resource, which can have properties of its own. Information such as who created the reference, when, and why can be stored on the redirect reference resource. Since they are implemented using status code 302 responses, it generally takes two round trips to submit a request to the intended resource. Redirect references work equally well for local resources and for resources that reside on a different server from the reference. On the other hand, bindings do not create a new resource, but simply makes a new URI for submitting requests to an existing resource available. The new URI can be used like any other URI to submit a request to a resource. Only one round trip is needed to submit a request to the intended resource. Because servers are required to enforce the integrity of the relationships between the new URIs that the clients create and the resources associated with them, binding requests will not work between different servers.

Each approach to shared resources has its own compliance class, methods and headers. If a WebDAV server supports redirect references resources, it should include redirectrefs in the DAV header as a response to an OPTIONS request. If it supports bindings, it should include bindings. Bindings are created with the BIND method and redirect references resources with MKRESOURCE.

### 3.4.2 Versioning

Version management was one of the original parts of WebDAV that is now developed in the IETF Delta-V working group. WebDAV’s versioning consists of: automatic versioning support for version-unaware clients, linear versioning and support for parallel development and configuration management [32]. The versioning support is divided into separate parts, one Core extension and some supplementary extensions: Activity, Merging, Configuration, Versioned Collection and Baseline versioning support. The Core support provides versioning of largely independent resources. It allows authors to concurrently create and access distinct revisions of a resource. Activity support includes logical change tracking and management through activities. Merging support encompasses conflict detection and resolution through merging. Configuration support extends Core support with the creation of sets of consistent revisions. Versioned Collection makes it possible to version the URL namespace. Finally, Baseline support extends Configuration and Versioned
Collection support with the ability to create and compare configurations of all revisions in a URL sub tree.

When working with versioned resources, the resources first have to be checked out to create an editable working resource. This is done with the `CHECKOUT` method. The resource is placed in the user's workspace together with other versioned resources that he needs. After editing, the resource is checked in, with the `CHECKIN` method. The resource is stored with a new revision number.

### 3.4.3 Searching and Locating

The DAV Searching and Locating (DASL) Protocol is an extension to support efficient searching for resources based on WebDAV properties and content [33]. With HTTP and WebDAV extensions, it is possible to perform client-side search but not server-side search. A client can fetch a document with the GET method to examine its contents and use the PROPFIND method to look at its properties. Having retrieved a resource's properties and/or content, the client can compare them to its search criteria to determine whether the resource is of interest. This search behavior makes inefficient use of network resources. A client must retrieve properties and content for each resource under consideration. Furthermore, it does not take advantage of the server's power. Using server-side searching, methods such as internal caching and content indexing can be utilized.

DASL allows for server-side searching. It allows the client to formulate a query and have the server perform the task of selecting the resources that match the criteria. The benefit is a searching solution that scales well, while the cost is that the server software becomes more complex. DASL encompasses the `SEARCH` method, the DASL response header, the `DAV:searchrequest` XML element, the `DAV:queryschema` property, the `DAV:basicsearch` XML element and query grammar, and the `DAV:basicsearchschema` XML element.

When a client want to search for something using DASL, it first constructs a query using the `DAV:basicsearch` grammar. Then, the client invokes the `SEARCH` method on a resource that will perform the search (the search arbiter) and includes a text/xml request entity that contains the query. On the server-side, the search arbiter performs the query. The result is sent back to the client in the response.

```xml
<?xml version="1.0"?>
<d:searchrequest xmlns:d="DAV:"
<d:basicsearch>
    <d:select>
        <d:allprop/>
    </d:select>
    <d:from>
        <d:scope>
            <d:href>/TeamWork/</d:href>
            <d:depth>infinity</d:depth>
        </d:scope>
    </d:from>
    <d:where>
        <d:eq>
            <d:prop><d:iscollection/>
        </d:eq>
        <d:prop><d:isliteral>
        <d:literals>1</d:literals>
        </d:prop>
    </d:where>
</d:basicsearch>
</d:searchrequest>
```

Figure 28 SEARCH for collections.
An example of a search request is found in Figure 28. The client use the SEARCH method and includes a DAV:searchrequest using DAV:basicsearch. The DAV:select element specifies that all the properties on the found resources shall be returned. The scope of the search is specified in DAV:from. The directory /TeamWork/ and all its subfolders will be searched. The DAV:where element specifies the search criteria. In the example, all collections in the scope are returned.

DASL is implemented in Microsoft's Internet Information Server 5. Since it is hard to find any documentation about the implementation, it is not possible to tell whether they comply with the Internet-Drafts or not. According to the responses from the server (on a OPTIONS request), it seems like the only supported query language is DAV:sql. DAV:basicsearch, which must be supported according to the drafts, is not included.

### 3.4.4 Access Control

The ordinary access control in HTTP is an authentication mechanism that grants or denies access to directories. If a user can send the correct username and password to the server, he can access the directory. Depending on the configuration of the server, the user may also have write access.

In WebDAV, a more complicated access control system is needed. According to the goals documents [34], three different scenarios have to be handled. These are: Different authors on each document, Denying to member of a group and Delegation. In the first scenario, a person owns a directory of documents. Each of the documents has its editor. WebDAV's Access Control system must allow the person the set access permissions individually for each document; so only the correct editors have write access to each document. In the second scenario, another person administrates a couple of files, which can be read by all members of a group. Since one document contains a secret that one of the group members should not know, the administrator must be able to set the permissions on that document such that even though the group is allowed to access the document one single member is denied. In the final scenario, a systems administrator wants to delegate some administration rights of a directory to another person. That person should be able to edit the access rights on the directory and its resources, but it should not be possible to take over entirely by removing all permissions from the systems administrator.

<table>
<thead>
<tr>
<th>Right</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>All the rights a resource can have.</td>
</tr>
<tr>
<td>Read</td>
<td>Read right to a resource and its properties.</td>
</tr>
<tr>
<td>write</td>
<td>Write right to a resource.</td>
</tr>
<tr>
<td>delete</td>
<td>Access to the DELETE method.</td>
</tr>
<tr>
<td>createchild</td>
<td>Controls the ability to PUT internal members of a collection.</td>
</tr>
<tr>
<td>deletechild</td>
<td>Controls the ability to DELETE internal members of a collection.</td>
</tr>
<tr>
<td>writeowner</td>
<td>The ability to change the value of the owner right.</td>
</tr>
<tr>
<td>Readacl</td>
<td>The ability to read the ACL property.</td>
</tr>
<tr>
<td>writeacl</td>
<td>The ability to alter the ACL property.</td>
</tr>
</tbody>
</table>

*Table 4 WebDAV Access Control Rights.*

The proposed protocol for access control is currently an Internet-Draft, called “WebDAV ACL Protocol” [35]. The basic model for access control states that who you are determines how you can access a resource. To decide who is accessing a
resource, an identifier is used. The ways a user can access a resource is determined by an "access control list" (ACL) associated with a resource. An ACL contains Access Control Elements (ACE), which specifies a set of users or groups, a set of granted rights, and a set of denied rights. When a new resource is created, it inherits a set of default ACL properties from another resource, which is specified as an ACL source. The inheritance can be static, which means that subsequent changes to the ACL source do not affect the new resource's ACL properties, or it can be dynamic. With dynamic inheritance, changes in the ACL source are reflected in the new resource's ACL properties. When accessing properties on a resource, the resource dynamically inherits from its ACL source. It is also possible to assign an ACL to an individual property. The rights that are possible to set on resources belong in the namespace http://www.ietf.org/standards/acl/. A listing of them is found in Table 4.

```
>>Request
ACL /TeamWork/Secret/ HTTP/1.1
Host: draco
Content-Type: text/xml
Content-Length: 0

>>Response
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: 725

<?xml version="1.0" ?>
<a:acl xmlns:D="DAV:" xmlns:A="http://www.ietf.org/standards/acl/">
  <a:inheritance>dynamic</a:inheritance>
  <a:owner>bjorn</a:owner>
  <a:ace>
    <a:principal>domain/joebob</a:principal>
    <a:grant/>
    <a:deny><a:all/></a:deny>
  </a:ace>
  <a:property>
    <d:prop><d:creationdate></d:prop>
    <a:acl>
      <a:inheritance>dynamic</a:inheritance>
      <a:owner>blah</a:owner>
      <a:ace>
        <a:principal>domain/joebob</a:principal>
        <a:grant><a:all/></a:grant>
        <a:deny/>
      </a:ace>
    </a:acl>
  </a:property>
  <a:property>
    <d:prop>
      <d:displayname/>
      <d:get-content-length/>
      <d:get-content-type/>
      <d:get-etag/>
      <d:get-lastmodified/>
    </d:prop>
    <a:acl>
      <a:inheritance>dynamic</a:inheritance>
    </a:acl>
  </a:property>
</a:acl>
```

Figure 29 Working with Access Control Lists.

To deal with Access Control Lists, a new HTTP method is used - ACL. The method's request body is used to define alterations to the ACL of the resource and its properties. The Request-URI of the ACL method identifies the resource whose ACL information is to be retrieved and possibly altered. In Figure 29, a use of the ACL method is shown. Since the request is empty, no changes will be made. Instead the response contain all the relevant values. The resource obtains its own ACL dynamically from its parent. It also overrides the inherited ACL by defining its own owner. One user, domain/joebob, is denied all rights. While the ACL for
creation date is also inherited it has its own owner, blah, and has an additional ACE for joebob. All the rest of the properties have their ACLs inherited from the resource.

Currently no work is done with the Access Control Lists Protocol, but it is a necessary part of WebDAV, and will be finished sometime in the future.

### 3.5 WebDAV Servers

WebDAV servers can be of different compliance classes, which specify how much of the specifications they support. At present, only two classes are defined in RFCs 1 and 2. A class 1 compliant server follows all “MUST” requirements in [22]. Practically, it means namespace managements and properties. A class 2 server also supports locks (overwrite prevention). Compliance classes can have other identifiers than numbers. The Internet-Drafts introduces the following classes: orderedcoll, redirectrefs, and bindings.

The most popular web servers on the Internet will support WebDAV. Microsoft’s IIS 5 server is a class 2 WebDAV server. It has also support for searching with DASL. The server uses the Windows 2000 file system as a repository to store properties and resources. It is also a tight integration between the operating system and the server. When a file is remotely locked with WebDAV, it is also locked in the file system for a local user. IIS5 also uses Windows 2000 user and access-control lists to determine whether a WebDAV user has access to a particular file. There is no separate Web access-control mechanism used by the HTTP server.

The most commonly used web server, Apache, also has a module that provides class 2 WebDAV support: mod_dav. The module also uses a file system repository, but requires that the Apache server owns all WebDAV authorable files. This prevents local access to files. It avoids the need to assume root privileges under UNIX to change the ownership of files (which is a security risk). WebDAV users are created with authoring privileges, which makes local system accounts unnecessary.

### 3.6 Security Considerations

Using a HTTP server with WebDAV extensions implies greater security considerations than the ones that exist for HTTP. Below is a summary of the most important security risks found in [22].

**Authentication of Clients**

In WebDAV, authentication is an important issue. Many functions concern data manipulation on the server. HTTP’s basic authentication must only be used on secure channels. A secure channel can be a server running TLS or SSL with encryption, or a physically secured network with restricted access.

**Denial of Service**

Three kinds of Denial of Service attacks are possible:

- **PUT**ting large files to the server could attack the underlying storage.
- Asking for recursive operations on large collections could attack processing time.
- Making multiple pipelined requests on multiple connections could attack network connections.

**Security through Obscurity**

WebDAV’s **PROPFIND** method lists the member resources of a collection. This implies that security or privacy relying on the difficulty of discovering the names of network resources does not work. Instead, WebDAV servers should use access control techniques to prevent unwanted access.
Privacy Issues Connected to Locks

When `LOCK`ing a resource, the client may submit an owner field with information about himself. In many cases, this contact information can be private and the server should limit the read access to this property.

Privacy Issues Connected to Properties

Property values should typically be used to store values such as author's name of a document. Sometimes more sensitive properties such as email address and phone number could be considered necessary to assign a resource. In that case, not all users should have access to the resource's properties. It should be possible to have separate access control for the resource's body and its properties.

Implications of XML External Entities

XML supports a facility known as “external entities”. This instructs an XML processor to retrieve and perform an inline include of XML located at a particular URI. An external XML entity can be used to append or modify the document type declaration (DTD) associated with an XML document. It can also be used to include XML within the content of an XML document. The risks here are that WebDAV's DTD can be modified and the form of the incoming XML document changes.
4 Analysis of WebDAV

In this chapter, the results of the work are described. First, file transfers with WebDAV are examined. Second, mail handling, and finally, the behavior of the most common WebDAV applications.

4.1 File transfers

4.1.1 Testbed setup

To measure the performance difference between HTTP and FTP, a test environment was set up. The test environment encompassed two computers, connected with a cross-connected twisted pair cable. The bit rate of the connection was 10 Mbit. As a client computer, a 200 MHz Pentium Pro with 96 MB RAM was used, running Windows NT 4 with Service Pack 4. The server was using the same hardware, but the Linux operating system.

When FTP’s performance was tested, the client program “Leap FTP” was used [36]. The server was running the default FTP daemon, which was wu-2.5.0 [37]. To measure HTTP’s performance, Perl scripts were written to send the necessary HTTP commands. The scripts were written with ActivePerl for Windows [38]. The HTTP server was running Apache version 1.3.9 with mod_dav version 0.9.12 [39]. During the tests, the network traffic was monitored with tcpdump on the Linux server. The data was later examined with tracelook [40]. The data files, that was transferred during the tests was created with a Perl script, that wrote random numbers to a file.

4.1.2 Performance

The first test scenario measured performance during simple file transfers. Test files was created with the following sizes: 10kB, 25kB, 50kB, 75kB, 100kB, 250kB, 500kB, 750kB, 1MB, 2.5MB, 5MB, 7.5MB and 10MB. The test files were transferred from the server to the client five times. An average transfer time was calculated and plotted in graphs (see Figure 30 and Figure 31).

Figure 30 File Transfer with FTP.
An evaluation of the WebDAV extensions to the HTTP protocol

Analysis of WebDAV

Figure 31 File Transfer with HTTP.

To compare the two different transfers, a ratio between the transfer times was created (see Figure 32). According to the graph, there are some differences when transferring small files under 100kB. When the ratio is above one, FTP is better, otherwise HTTP is better. The explanation to this can be that the operating systems is busy with interrupts and different processes and cannot devote all its time to networking activities. With the larger files, the ratios are more stable. Despite that they are never exactly the same (ratio 1). An explanation to that is that for FTP, only the traffic in the data channel is measured. It means that for HTTP, more data is included in the measured amount of data (both protocol commands and the actual data), while for FTP, only the data is measured.

Figure 32 HTTP/FTP Ratio.

The second test scenario studied the performance, when more than one file was transferred in the same session. Theoretically, HTTP has an advantage, since with persistent connections and pipelining the performance should be better than FTP. To use these features, the client sends a couple of GET methods after each other (pipelining) and then close the connection. Then, the server should process the commands in the order they are received and send the results back in the same order.
When all the commands are processed, the TCP connection is finally closed. An example of this is shown in Figure 33.

```
GET test10000.dat HTTP/1.1
Host: dav.malmo.trab.se

GET test10000.dat HTTP/1.1
Host: dav.malmo.trab.se

GET test10000.dat HTTP/1.1
Host: dav.malmo.trab.se

GET test10000.dat HTTP/1.1
Host: dav.malmo.trab.se

GET test10000.dat HTTP/1.1
Host: dav.malmo.trab.se
Connection: close
```

Figure 33 Pipelining in HTTP.

The test files that was used had the following sizes: 10kB, 50kB, 100kB, 500kB, 1MB, 5MB and 10MB.

![Multiple File Transfer with FTP](image)

Figure 34 Multiple File Transfer with FTP.
An evaluation of the WebDAV extensions to the HTTP protocol
Analysis of WebDAV

Figure 35 File Transfer with Pipelined HTTP.

In the charts, the gradient of the transfer increase with larger files. To better analyze this, the average transfer rate is calculated. The results are shown in Figure 36 and Figure 37.

Figure 36 Average Transfer Rate - FTP.

Figure 37 Average Transfer Rate - HTTP.
For FTP, it seems like the Average Transfer Rate varies from 27 000 bytes per second for small files, to 122 000 bytes per second for large files. With HTTP, the Transfer Rate seems to be quite stable around 120 000 bytes per second, it only increases a bit for large files. To better see the differences, a ratio was calculated (see Figure 38).

![HTTP/FTP Ratio (Average Transfer Rate)](image)

**Figure 38 HTTP/FTP ATR Ratio.**

When studying the Ratio for Average Transfer Rate, it seems like HTTP is four times better than FTP for 10kB files, two and a half times better for 50kB files and about 60 percent better for 100kB files. For 500kB files, HTTP is about 17% better and for larger files less than 10% better. A ratio to calculate the differences in transfer time and transferred bytes was also done. It is shown in Figure 39 and Figure 40.

![HTTP/FTP Ratio (Transfer Time)](image)

**Figure 39 HTTP/FTP Ratio (TT).**

The ratio for Transfer Time shows that HTTP has great advantages with small files. With 10kB test files, HTTP takes 30% of the time that FTP needs. It makes it 3.6 times faster than FTP. When the file size is increased to about 1MB, HTTP takes approximately 90% of the FTP transfer time. Then, HTTP is about 10% faster.
Finally, the actual number of bytes transferred in the sessions was examined. The comparison is not totally fair, since for FTP, only the bytes transferred in the data channel are counted. The commands sent in the control channel are excluded. To get an idea about how many bytes that are sent there, it is possible to count the number or bytes sent when requesting a file. It depends on the filename and the file's size, but is should be something about 200 bytes per file. When looking at the charts, the overhead can be estimated to about 18% larger for 10kB files and then rapidly decreasing. For 100kB, the overhead is only about 1%.

<table>
<thead>
<tr>
<th>Transfer</th>
<th>HTTP Time</th>
<th>FTP Time</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 × 10kB</td>
<td>8,60s</td>
<td>30,96s</td>
<td>HTTP 260% faster</td>
</tr>
<tr>
<td>100 × 10MB</td>
<td>7209,84s</td>
<td>8153,34s</td>
<td>HTTP 13% faster</td>
</tr>
</tbody>
</table>

Table 5 Transfer Time Summary.

<table>
<thead>
<tr>
<th>Transfer</th>
<th>HTTP Rate</th>
<th>FTP Rate</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 × 10kB</td>
<td>120369 bytes/s</td>
<td>28299 bytes/s</td>
<td>HTTP 325% faster</td>
</tr>
<tr>
<td>100 × 10MB</td>
<td>138951 bytes/s</td>
<td>122644 bytes/s</td>
<td>HTTP 13% faster</td>
</tr>
</tbody>
</table>

Table 6 Average Transfer Rate Summary.

<table>
<thead>
<tr>
<th>Transfer</th>
<th>HTTP Size</th>
<th>FTP Size</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 × 10kB</td>
<td>1035689 bytes</td>
<td>876000 bytes</td>
<td>HTTP 18% more</td>
</tr>
<tr>
<td>100 × 10MB</td>
<td>1001816979 bytes</td>
<td>999954000 bytes</td>
<td>HTTP 0,2% more</td>
</tr>
</tbody>
</table>

Table 7 Transferred Bytes Summary.

A summary of the test is found in Table 5, Table 6 and Table 7. Another study about HTTP’s performance, states that persistent connections do not substantially affect the majority of users [41]. Most users use modems and at these rates, persistent connections reduce the overall transaction time by 11%. As in this study, bandwidths over 200kbps are required to provide user-noticeable performance improvements.
4.1.3 Functionality

To compare the functionality between FTP and HTTP with WebDAV extensions, first an examination of FTP’s features is done. In a second step, a description about how to do the same thing (if possible) with HTTP and WebDAV is presented. Finally a presentation about things that are possible to do with WebDAV, but not with FTP. A large difference between the protocols is that FTP has a state, while HTTP is stateless. Despite that, it does not affect the functionality.

In FTP it is possible to navigate the remote file system. Descriptions about the features are found in Chapter 2.2. To continue to download a partially downloaded file, the REST command can be used. In the Internet Standard about FTP, this command is only defined in Block and Compressed transfer mode, but it is widely implemented for Stream mode too. Finally, in FTP, a user can remotely transfer files between two FTP servers. This is done by logging in at one server and set it in passive mode (and listen to data transfer connections) with the PASV command. Then, the user logs in at the other server, sets the host’s data port to the same as the first server is listening to, and copy files with RETR.

When using HTTP, the server can allow the user to browse directories (this is a feature that the webmaster has to activate). With this feature, files and directories are returned as HTML documents with links to the resources. If the HTTP server is using WebDAV, the file listing of a directory is retrieved with the PROPFIND method. On this point FTP and HTTP are similar. Directories can be created with either WebDAV’s MKCOL method, or the PUT or POST method. To delete them, the DELETE method is used. Files can be renamed by moving them to a new filename with the MOVE method. As directories, files are also deleted with the DELETE method. Files are retrieved with the GET method. To upload files to the server, the PUT method is used. Partial downloads are defined in the standard as a GET with a Range header.

A couple of features in WebDAV do not exist in FTP. Neither properties on files and directories nor write locks can be set in FTP. It is also possible to move and copy resources on the server with WebDAV’s MOVE and COPY methods. FTP does not have anything corresponding.

A summary of the comparison is found in Table 8.

<table>
<thead>
<tr>
<th>Feature</th>
<th>FTP</th>
<th>HTTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Navigate in file system</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>List files at server</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Create directory on server</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Delete directory on server</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Rename files on server</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Delete files on server</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Receive files</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Store files on server</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Append files on server</td>
<td>☑</td>
<td>☞</td>
</tr>
<tr>
<td>Restart in file</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Copy files between servers</td>
<td>☑</td>
<td>☞</td>
</tr>
<tr>
<td>Properties on resources</td>
<td>☞</td>
<td>☑</td>
</tr>
<tr>
<td>Write locks on resources</td>
<td>☞</td>
<td>☑</td>
</tr>
<tr>
<td>Copy and move files on server</td>
<td>☞</td>
<td>☑</td>
</tr>
</tbody>
</table>

Table 8 Functionality comparison between FTP and HTTP with WebDAV extensions.
4.1.4 Security

To compare security, an evaluation of the authentication, integrity and confidentiality functions in FTP and HTTP was done.

In FTP, authentication is done by sending the username with USER and the password with PASS. Both username and password is sent in plain text. It is not possible to send the password encrypted in any way. Integrity and confidentiality is not supported with FTP according to the Internet Standard. If the FTP server has implemented the security extensions found in RFC 2228, then all the security issues can be handled. At present, none of the popular FTP servers handles the security extensions, and the export restriction on cryptographic products prevents spreading of the technology.

In HTTP on the other hand, authentication can be performed in plain text with Basic Authentication, and with encrypted passwords with Digest Authentication. If integrity and confidentiality is desired, the best current practice is to use Netscape's SSL. A summary of the security comparison is found in Table 9.

<table>
<thead>
<tr>
<th>Feature</th>
<th>FTP</th>
<th>HTTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication with password in clear text</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Authentication with encrypted password</td>
<td>☐</td>
<td>☑</td>
</tr>
<tr>
<td>Integrity</td>
<td>☐</td>
<td>☑</td>
</tr>
<tr>
<td>Confidentiality</td>
<td>☐</td>
<td>☑</td>
</tr>
</tbody>
</table>

Table 9 Security comparison between FTP and HTTP.

4.2 Mail reading

In this section a model that will make it possible to represent a mailbox as a collection on a web server is presented.

4.2.1 Overview

If one wants to create a system where mail is read with HTTP and WebDAV instead of POP3, the visual look of the client can be the same. Only the communication between the client and the server will change. Since the POP3 protocol only defines how to retrieve mails from the server, mail still have to be sent with SMTP.

First, a namespace has to be created for each user on the server. The URL can look like: http://www.trab.se/mail/xbjni/. To make sure that no one reads another person's mail, some kind of authentication mechanism have to be used. If only authentication is needed, HTTP's Digest authentication can be used (it is generally a bad idea to choose Basic authentication, since the password is sent in plain text). If also integrity and confidentiality is wanted, TLS (or SSL) is the appropriate choice.

4.2.2 Properties

In RFC 822, “Standard for the format of ARPA Internet Text Messages” [42], messages are viewed as having an envelope and contents. The envelope contains the information that is needed to deliver the message to the recipient, while the content is the actual message.

In a user's collection, each message is stored as a resource. To be able to get information about a message, without downloading it, properties are assigned to each resource. To limit the amount of properties on each resource, this model only use the ones shown in Figure 41. It encompasses the most common properties of a
message. A special namespace, called POP, is also used to separate them from the other properties.

<table>
<thead>
<tr>
<th>POP3:From</th>
<th>POP3:Sender</th>
</tr>
</thead>
<tbody>
<tr>
<td>POP3:Reply-To</td>
<td>POP3:Date</td>
</tr>
<tr>
<td>POP3:To</td>
<td>POP3:cc</td>
</tr>
<tr>
<td>POP3:bcc</td>
<td>POP3:Subject</td>
</tr>
</tbody>
</table>

**Figure 41 POP3 properties on WebDAV resources.**

### 4.2.3 Functionality of the client

As stated before, only the communication between the server and the client will change when HTTP with WebDAV is used instead of POP3. An important difference between HTTP and POP3 is that POP3 is a connection-oriented protocol, while HTTP is stateless. When operations are done in the mail collection, authentication information has to be included every time. In POP3 session, where the user checks how many mails there are in the mailbox, the client sends the following commands: **USER/PASS** (Authentication state) and **STAT/LIST** (Transaction state). In HTTP, all this is done with a single **PROPFIND** request. To retrieve a message, the client uses the **RETR** command. In HTTP, this is done with the **GET** method. Finally, deleting a mail is made with **DELE** in POP3. The HTTP correspondence is **DELETE**. To do the same thing with HTTP as in the POP3 session in Figure 16, the appropriate command sequence is shown in Figure 42.
An evaluation of the WebDAV extensions to the HTTP protocol

Analysis of WebDAV

Figure 42 Checking the mailbox with HTTP.

In the example, a PROPFIND request is sent to the user's collection. Authentication information for HTTP Digest Authentication is included in the request. It shows that one message is in the mailbox. The session is done with one round-trip. If the same process should be done with POP3, the commands: USER, PASS, STAT, and QUIT has to be sent. That session would encompass four round-trips, and then we still do not know who sent the messages and their subject.

4.3 Behavior of common WebDAV applications

In this section is a description of how a couple of common WebDAV clients use the protocol. The company that has put much effort in the development of WebDAV is the Microsoft Corporation. That is the reason why their products have large support for the protocol.

4.3.1 Web Folders

With Internet Explorer 5 or Office 2000 an application called Web Folders is available. It shows up under “My Computer” on Windows machines. A Web Folder provides a local view of a remote web server. It allows resources on the server to be viewed and manipulated like files in the local file system.
The functionality with Web Folders is implemented in a programming interface called the “OLE DB Provider for Internet Publishing” [43]. It maps requests made via the user interface into network protocol requests. The Interface supports two different extensions to HTTP. One of them is WebDAV, and the other is FrontPage Web Extender Client (WEC) [44]. The WEC protocol has similar functionality to WebDAV, but instead of extending HTTP, it uses a Remote Procedure Call protocol that is layered on top of HTTP and HTML. It works by sending **GET** and **POST** requests to the server.

```
>>Request
OPTIONS / HTTP/1.1
User-Agent: Microsoft Data Access Internet Publishing Provider Cache Manager
Host: draco
Content-Length: 0
Connection: Keep-Alive
Cookie: HTMLA=FONTSIZE=LARGE

>>Response
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Mon, 29 Nov 1999 10:30:07 GMT
MS-Author-Via: DAV
Content-Length: 0
DASL: <DAV:sql>
DAV: 1, 2
Public: OPTIONS, TRACE, GET, HEAD, DELETE, PUT, POST, COPY, MOVE, MKCOL, PROPFIND,
PROPPATCH, LOCK, UNLOCK, SEARCH
Allow: OPTIONS, TRACE, GET, HEAD, COPY, PROPFIND, SEARCH, LOCK
```

Figure 43 OLE DB Provider's OPTIONS.

The first time a Web Folder is accessed the OLE DB Provider determines which protocol it should use. This is done by sending an **OPTIONS** request where the request-URL is the parent of the URL specified by the client (See Figure 43). If the server’s response contains an **MS-Author-Via** header, the value of the string is used to determine what protocol to use. The valid values are **MS-FP/4.0** and **DAV**. If only one value is specified, that alternative is used. Otherwise, if both are given, the one that comes first is selected. Since there is a performance cost in checking which protocol to use, the OLE DB Provider maintains an in-memory cache of associations between URLs and the correct protocols.

In order to see how Web Folders handles the WebDAV protocol, a couple of tests were done. In these tests Microsoft Internet Information Service version 5 server was used. The network traffic was monitored with Microsoft Network Monitor. Both the server software and the network monitor are included in Windows 2000 Advanced Server (RC2). In the tests, all possible ways of file manipulation was examined. It includes copying, moving, renaming and deleting files and folders. Folder creation was also studied.
An evaluation of the WebDAV extensions to the HTTP protocol
Analysis of WebDAV

Figure 44 Web Folder's PROPFIND on a collection.

To view the contents of a collection, Web Folders use the PROPFIND method. The request is shown in Figure 44. It requests properties on the specified collection and all its immediate children. In the body entity is a listing of the properties that is used.

Figure 45 Uploading a file with Web Folders.

Figure 46 HEAD on a file to be uploaded.
First, files and directories was copied to the server and then deleted. To copy a single file, Web Folders sends five HTTP requests. The sequence is shown Figure 45. First, a HEAD request is sent to see if there already is a file with the same name (Figure 46). When the response’s status code is 404 (Resource Not Found), Web Folders continues to create the resource with a PUT request. The request body is empty and the status code from the server is 201 (Created). A new try is made with HEAD. This time the returned status is 200 (OK) and Web Folders does a PUT request with the file to be uploaded. This is accepted and finally a new HEAD request is sent. It seems like the server closes the persistent connection when the first HEAD request fails. This leads to worse performance, than if a persistent connection should be used through the whole transaction.

```
MKCOL /TeamWork/New%20Folder HTTP/1.1
Accept-Language: sv, en-us;q=0.5
Translate: f
User-Agent: Microsoft Data Access Internet Publishing Provider DAV
Host: draco
Content-Length: 0
Connection: Keep-Alive
Cookie: HTMLA=FONTSIZE=LARGE

HTTP/1.1 201 Created
Server: Microsoft-IIS/5.0
Date: Wed, 15 Dec 1999 15:01:07 GMT
Location: http://draco/TeamWork/New%20Folder/
Content-Length: 0
```

Figure 47 Creating a collection.

If a collection was uploaded instead of a resource, Web Folders first created the collection and then uploaded each resource as mentioned before. The creation was done with a single request as shown in Figure 47. Before resources were PUT into the collection, a PROPFIND was done on it. When a resource or collection was to be deleted, Web Folders first did a PROPFIND on the collection it was located in. Then, the DELETE request was sent.

```
>> PROPFIND /TeamWork/New%20Folder
<< 207 Multi-Status
>> COPY /TeamWork/Rebecka.jpg
<< 201 Created
```

Figure 48 COPY resources with Web Folders.

```
>> Request
COPY /TeamWork/Rebecka.jpg HTTP/1.1
Accept-Language: sv, en-us;q=0.5
Destination: http://draco/TeamWork/New%20Folder/Rebecka.jpg
Overwrite: F
Translate: f
User-Agent: Microsoft Data Access Internet Publishing Provider DAV
Host: draco
Content-Length: 0
Connection: Keep-Alive
Cookie: HTMLA=FONTSIZE=LARGE

>> Response
HTTP/1.1 201 Created
Server: Microsoft-IIS/5.0
Date: Fri, 17 Dec 1999 09:25:19 GMT
Location: http://draco/TeamWork/New%20Folder/Rebecka.jpg
Content-Type: text/xml
Content-Length: 0
```

Figure 49 The COPY request.

Web Folders copied resources and collections with WebDAV’s COPY method. First, a PROPFIND was sent, targeted to the collection where the resource was going to be copied to (Figure 48). Then the resource was copied with the COPY method.
An evaluation of the WebDAV extensions to the HTTP protocol

Analysis of WebDAV

(Figure 49). The Overwrite header specifies that if the resource already exists, the request should fail. Copying a collection resulted in the same sequence of methods.

Moving resources on the server worked the same way. A PROPFIND was first issued on the destination, followed by a MOVE request.

```
>> PROPFIND /TeamWork/
<< 207 Multi-Status
>> MOVE /TeamWork/MOVE.txt
<< 201 Created
```

**Figure 50 Renaming a resource.**

<table>
<thead>
<tr>
<th>MOVE /TeamWork/MOVE.txt HTTP/1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept-Language: sv, en-us;q=0.5</td>
</tr>
<tr>
<td>Overwrite: F</td>
</tr>
<tr>
<td>Destination: <a href="http://draco/TeamWork/Flytta.txt">http://draco/TeamWork/Flytta.txt</a></td>
</tr>
<tr>
<td>Translate: f</td>
</tr>
<tr>
<td>User-Agent: Microsoft Data Access Internet Publishing Provider DAV</td>
</tr>
<tr>
<td>Host: draco</td>
</tr>
<tr>
<td>Content-Length: 0</td>
</tr>
<tr>
<td>Connection: Keep-Alive</td>
</tr>
<tr>
<td>Cookie: HTMLA=FONTSIZE=LARGE</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>HTTP/1.1 201 Created</td>
</tr>
<tr>
<td>Server: Microsoft-IIS/5.0</td>
</tr>
<tr>
<td>Date: Fri, 17 Dec 1999 09:26:55 GMT</td>
</tr>
<tr>
<td>Location: <a href="http://draco/TeamWork/Flytta.txt">http://draco/TeamWork/Flytta.txt</a></td>
</tr>
<tr>
<td>Content-Type: text/xml</td>
</tr>
<tr>
<td>Content-Length: 0</td>
</tr>
</tbody>
</table>

**Figure 51 Rename with the MOVE method.**

If resources were renamed, this was done with the MOVE method. As before, the operation begun with a PROPFIND request on the collection with the resource to be renamed. The sequence is shown in Figure 50. The actual renaming was done by moving the resource to the same collection, but with a different name (Figure 51).

```
>> PROPFIND /TeamWork/
<< 207 Multi-Status
>> PROPFIND /TeamWork/
<< 207 Multi-Status
>> GET /TeamWork/Move.txt
<< 200 OK
>> GET /TeamWork/Copy.txt
<< 200 OK
>> PROPFIND /TeamWork/
<< 207 Multi-Status
```

**Figure 52 Retrieving a collection.**

Finally, it was examined how Web Folders retrieves resources. If only one resource was retrieved, it was performed with a single GET request. When a collection was downloaded, a larger sequence of requests was sent (Figure 52). First, two PROPFINDs were sent, followed by a GET for each resource. Finally another PROPFIND was sent.

After performing the behavioral test, it seems as one aspect of WebDAV is not supported - Properties. Even if a resource was assigned properties, it was not possible to view them with Web Folders. Only the same properties as ordinary files on the local computer had could be viewed on remote files.

4.3.2 Office 2000

The new Office 2000 applications use Web Folders to enable remote collaborative authoring of its documents. All applications use the same dialog box to open and save files. With the dialog box, it is possible to open Web Folders.
An evaluation of the WebDAV extensions to the HTTP protocol

Analysis of WebDAV

To load a file from a Web Folder in an Office 2000 application, he selects File-&gt;Open… from the menu and then chooses a file on a web server. The client first sends an OPTIONS request to the server, targeted at the collection that the resource is placed in (Figure 53). If the response shows that the server is WebDAV enabled and supports locking, the client locks the file and fetches it.

In the second request, the client tries to lock the resource (Figure 54). It asks for an exclusive write lock and set the owner to the login name. The Timeout field specifies that the lock only should be set in two minutes. If the resource is already locked when the request is sent, the applications give the user an opportunity to open the document as read-only, and receive a notification when the document is unlocked. If it is opened this way, Office checks the lock status by polling the resource with LOCK requests. When the lock finally succeeds, it notifies the user.

Finally, the resource is retrieved from the web server. HTTP caching is used, so if the contents of the document have not changed on the remote server since the last time it was requested, the cached copy is used. In Figure 55, this request is shown. The If-None-Match header is included, since a local copy exists. The argument to the header is the ETag of the resource. It the server’s ETag of the resource is the same it will respond with 304 (Not modified), otherwise 200 (OK) (and then the resource is also returned). Office makes a local, temporary copy of the downloaded resource and then allows editing to begin. When the application saves...
An evaluation of the WebDAV extensions to the HTTP protocol

Analysis of WebDAV

Automatic, it goes to the local copy, but when the user explicitly saves the document it is also PUT to the web server.

\begin{verbatim}
>>Request
LOCK /TeamWork/Tendens.doc HTTP/1.1
Accept-Language: sv, en-us;g=0.5
If: (<opaquelocktoken:E54FC83B-9BDC-4977-BC2D-10D2F1123163:214748364854>)
Content-Length: 0
Timeout: Second-120
Translate: f
User-Agent: Microsoft Data Access Internet Publishing Provider DAV
Host: draco
Connection: Keep-Alive
Cookie: HTMLA=FONTSIZE=LARGE

Figure 56 Updating a lock.

Since the write lock is set to expire in two minutes, it has to be refreshed regularly. Such a request is shown in Figure 56. The lock request looks almost as before, but now an If header is included. It contains the lock token, which must be used when operations on locked resources are performed. The strategy with two minutes locks ensures that the resources won’t be tied up if the locking application dies. The disadvantages is that it prevents working offline and it also adds to increased server load.

\begin{verbatim}
>>Request
PUT /TeamWork/Tendens.doc HTTP/1.1
Accept-Language: sv, en-us;g=0.5
If: (<opaquelocktoken:E54FC83B-9BDC-4977-BC2D-10D2F1123163:214748364854>)
Translate: f
Content-Length: 20480
User-Agent: Microsoft Data Access Internet Publishing Provider DAV
Host: draco
Connection: Keep-Alive
Cookie: HTMLA=FONTSIZE=LARGE

Figure 57 PUTting a document to the WebDAV server.

When the document is saved (with File -> Save or File -> Close), the resource is put back to the web server. This is done in three requests. The first request is a PUT (Figure 57). Since the resource still is locked, the If header must be included. The server responds with 100 (Continue) and the client sends the document. After the document is sent, the server responds with 200 (OK).

\begin{verbatim}
>>Request
HEAD /TeamWork/Tendens.doc HTTP/1.1
Accept-Language: sv, en-us;g=0.5
Content-Length: 0
User-Agent: Microsoft Data Access Internet Publishing Provider DAV
Host: draco
Connection: Keep-Alive
Cookie: HTMLA=FONTSIZE=LARGE

Figure 58 A HEAD request.

When the document is stored on the server, the client sends a HEAD request (Figure 58).
\end{verbatim}
Finally, the resource is unlocked (Figure 59). As before, the lock token is included. The server responses with 204 (No Content) if the operation is successful.
5 Conclusions

In this Master's Thesis I have tried to examine the WebDAV extensions to HTTP in a couple of different situations. As a platform for file services, HTTP with WebDAV can be an interesting solution. The performance is better than a file server based on FTP if HTTP’s functionality with persistent connections and pipelining is used. With SSL or TLS, a secure server can be built to handle sensitive information. The problem today is the lack of clients that support the WebDAV methods. A couple of freeware clients exist, but they are quite primitive. According to these tests, it seems like HTTP with WebDAV can work as a replacement for FTP. As a protocol for reading mail, WebDAV has the functions that are necessary to work as POP3. It can even add some value, by making it possible to view the subject and sender of a mail without downloading it from the server. The secure connection that is possible with SSL and TLS is also something that is not common with ordinary POP3 traffic. There are though work going on with POP3, and there exist Internet Standards about secure POP3 sessions. At present they are not implemented in the clients. The commercial applications that support WebDAV have currently a cautious attitude to its functions. It is neither possible to set properties on resources nor decide the timeout value of locks. The user cannot decide which author information that is sent to the server. The purpose is probably to use WebDAV without letting the user see any change to ordinary behavior.

The HTTP protocol with WebDAV extensions takes another great step into becoming a general-purpose protocol. The work that is done with protocols today such as FTP, POP3 and IMAP can be replaced with HTTP and WebDAV extensions. It could be replaced, but the most important question is: Should it be replaced? In many cases, HTTP servers become overloaded with requests for HTML pages. If the server show some kind of dynamic content, it is probably heavily loaded. In such case it is a bad idea to let the HTTP server deliver mail or files. If HTTP is wanted for this kind of purposes, HTTP should probably be run as deliverer of HTML pages on a different server.

One possible further work should be to replace a current FTP service with HTTP and WebDAV. The work should contain setting up the HTTP server and measure its performance. Another important part should be to create a client with a graphical user interface similar to some popular FTP client. On the Windows platform, also FTP servers are viewed as Web Folders, but it is still not the common way to access files with FTP. The work is probably more interesting to do when the ACL protocol to WebDAV have become an Internet Standard and is implemented in servers (it should be done in 2001).

Despite the fact that WebDAV adds increased complexity to web servers, it is no doubt that it will become an important standard. The major web server developers are currently announcing their support for the standard, and the software vendors will follow. The common users will probably not know about WebDAV, but only enjoy the new features it gives (such as update a web pages without having problems using FTP). Applications such as CVS can also use WebDAV, but mostly things “under the hood” changes. The applications will work as before, but the way it communicates with others will become standardized.
6 References

References


[37] WU-FTPD <http://www.wu-ftpd.org>

[38] ActivePerl <http://www.activestate.com/ActivePerl/> 


Appendix A – Glossary

Collection  A resource whose state consists of at least a list of internal member
            URIs and a set of properties, but which may have additional state such
            as entity bodies returned by GET. Similar to a directory in a file system.

CVS  Concurrent Version System. An application for versioning and
      configuration management.

IMAP  Internet Message Access Protocol. A more advanced mail protocol than
      POP3, used for reading and managing email on a mail server.

MIME  Multipurpose Internet Mail Extensions. Specifications for converting
       binary data (such as graphics, audio and video files) to ASCII text,
       which allows it to be sent via email.

Resource  An object found on a Web server. Similar to a file in a file system.

URI  Uniform Resource Identifier. The generic term for all types of names
     and addresses that refer to objects on the Internet. A URI can be
     further classified as a locator (URL), a name (URN), or both.

URL  Uniform Resource Locator. A text string used to identify the location
     of Internet resources. The first part of the address indicates what
     protocol to use, and the second part specifies the IP address or the
     domain name where the resource is located.

URN  Uniform Resource Name. A URN differs from a URL in that its
     primary purpose is persistent labeling of a resource with an identifier.