

(19) World Intellectual Property Organization
International Bureau



(43) International Publication Date
13 March 2008 (13.03.2008)

PCT

(10) International Publication Number
WO 2008/028508 A1

(51) International Patent Classification:

H04L 29/08 (2006.01) G06F 21/00 (2006.01)

H04L 29/06 (2006.01)

(21) International Application Number:

PCT/EP2006/008710

(22) International Filing Date:

7 September 2006 (07.09.2006)

(25) Filing Language:

English

(26) Publication Language:

English

(71) Applicant (for all designated States except US): **FUJITSU LABORATORIES OF EUROPE LIMITED** [GB/GB];
Hayes Park Central, Hayes End Road, Hayes, Middlesex
UB4 8FE (GB).

(72) Inventor; and

(75) Inventor/Applicant (for US only): **DRESCHER, Michel**
[DE/GB]; 8 Barnsley Close, Ash Vale, Aldershot, Hamp-
shire GU12 5RH (GB).

(74) Agents: **LEGG, Cyrus, James, Grahame et al.**; Abel &
Imray, 20 Red Lion Street, London WC1R 4PQ (GB).

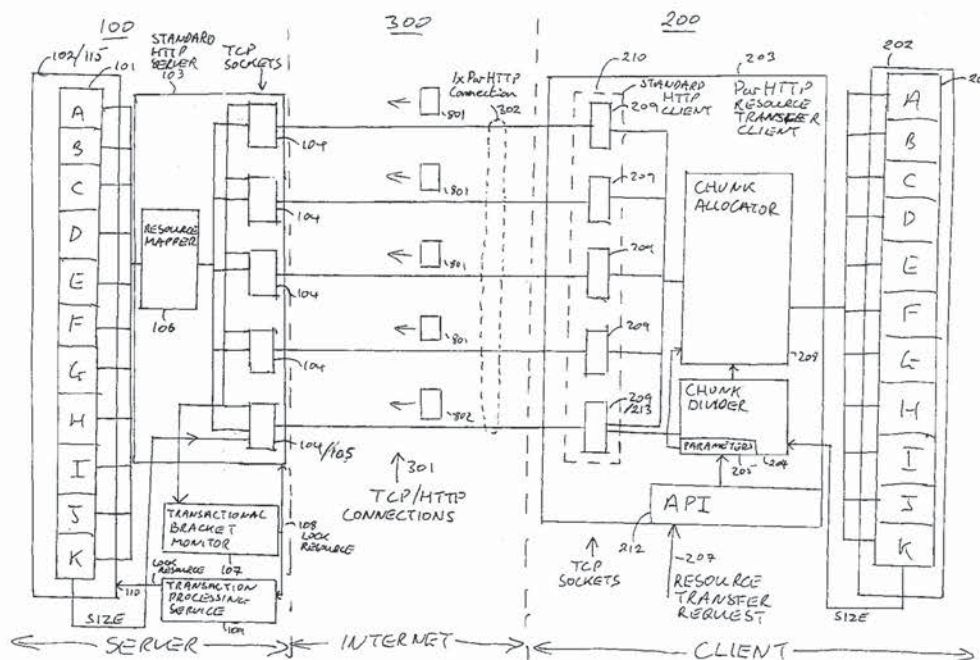
(81) Designated States (unless otherwise indicated, for every
kind of national protection available): AE, AG, AL, AM,
AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN,
CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI,
GB, GD, GE, GH, GM, HN, HR, HU, ID, IL, IN, IS, JP,
KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT,
LU, LV, LY, MA, MD, MG, MK, MN, MW, MX, MY, MZ,
NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RS, RU,
SC, SD, SE, SG, SK, SL, SM, SV, SY, TJ, TM, TN, TR,
TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States (unless otherwise indicated, for every
kind of regional protection available): ARIPO (BW, GH,
GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM,
ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM),
European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI,
FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT,
RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA,
GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— with international search report

(54) Title: **DISTRIBUTED COMPUTING AND COMMUNICATIONS PROTOCOL**



(57) Abstract: In a protocol for sending large files or resources over the internet chunks of the file are sent over parallel HTTP connections. Additional headers are provided in the HTTP request to identify the parties involved in the transfer, which are used by a security framework which checks the permissions based on the roles of the parties.

DISTRIBUTED COMPUTING AND COMMUNICATIONS PROTOCOL

The present invention relates to a communications protocol that is particularly useful in distributed computing and in
5 secure computing.

As is known, computing tasks can be distributed for execution to remote computers via a network (usually the Internet). Large tasks can be divided into smaller tasks which can be
10 distributed to different computers. Each task is distributed over the network, and once executed the results are sent back to one location and assembled into an overall results dataset. This process is often called grid computing.

15 The process of grid computing often requires the transfer of large data sets (e.g. 100's of megabytes or even several gigabytes or beyond) between computers.

Protocols for transferring datasets between computers are
20 known. One such is the file transfer protocol (FTP), which is used for moving files. Another is the hypertext transfer protocol (HTTP), which was developed to support the transfer of HTML document and other files for viewing dynamically in web browsers. These protocols are not suited to the needs of
25 grid computing.

One known improved protocol is called GridFTP. GridFTP provides extensions to the FTP protocol. These include an "extended block mode" in which data to be transferred is
30 divided into blocks which are distributed for transmission amongst several TCP connections. GridFTP, like FTP, employs a separate TCP connection as a control channel over which commands to control the process are sent. This means that the transfer process is particularly vulnerable to loss of the
35 control channel. Version 1 of the GridFTP protocol is

described in a document entitled "GWD-R (Recommendation) Grid FTP: Protocol Extensions to FTP for the Grid" of April 2003. Version 2 is described in a document entitled "GFD-R-P.047 GridFTP v2 Protocol Description" of May 2005.

5

For security GridFTP employs the "Grid Security Infrastructure" from the Globus Project. GSI sites allow remote users to have computing tasks performed at the site on the basis of an X.509 proxy certificate which is issued by the user thereof (i.e. not by a certificate authority) and contains the user's identity. A disadvantage of this system is that the private key of the certificate is not stored in encrypted form and so is vulnerable to access on the server performing the requested work and to subsequent misuse. For this reason such certificates are given a short lifetime of hours or days.

The problems of GSI and GridFTP are discussed in FUJITSU Sci. Tech. J. 40,2,p282-294 (December 2004). This document also describes the alternative UNICORE job management and security system for grid computing. UNICORE uses its own protocol for communication between the sites involved which runs over SSL. In the Explicit Trust Delegation (ETD) variant of UNICORE proposed in that document the security roles in the system include user, endorser and consigner, all of which have their identities established through the use of X.509 certificates issued by a certificate authority. Requests (known as Abstract Job Objects) for work to be executed on a server under the user's identity are transmitted to the server. These requests are signed either directly with the user's certificate or by an endorser, which is another entity in the UNICORE system, usually a server, that is permitted to make work requests on behalf of the user. (Such permissions are stored in the UNICORE system's authorisation

database.) Where the transfer of a data file is required various implementations of UNICORE use different protocols but GridFTP is one of the options available.

5 The present invention, in one aspect, provides an improved protocol for transferring large files, or other resources, over a network, improving, for example, the efficiency and speed with which they are transferred. The invention may be used to transfer resources securely and in a second aspect
10 provides features useful to a security framework, e.g. for job management and for the secure transfer of resources between sites. In both aspects the invention is useful in distributed computing.

15 According to the first aspect of the invention there is provided method of transferring a resource over a network between a first station and a second station, the method comprising:

dividing the resource into a plurality of chunks,
20 providing a plurality of connections over the network between the first and second stations, the connections being open at the same time, and

sending the chunks between the first and second station, at least one chunk being sent over each of the plurality of
25 connections,

wherein each chunk is sent over one of the connections of the plurality by making a hypertext transfer protocol (HTTP) request from the first station to the second station.

30

The first aspect of the invention also provides a resource transfer client software component for transferring a resource over a network between a first station and a second station, comprising:

a chunk divider arranged to divide the resource into a plurality of chunks, and

a chunk allocator arranged to make requests to the second station for the sending of the chunks between the first and second station over the network, at least one chunk being sent over each of a plurality of connections, the connections being open at the same time,

wherein the chunk allocator is arranged to make for the sending of each chunk a hypertext transfer protocol (HTTP) request to send the chunk over one of the connections of the plurality.

In one particular preferred form of the method of first aspect of the invention the method comprises monitoring at the second station to determine when the sending of the chunks of the plurality remains in progress, and preventing at least one type of concurrent access to the resource during a period that the sending of the chunks of the plurality remains in progress. Also the first aspect of the invention provides a resource transfer server software component comprising an HTTP server component, and a transactional bracket monitor arranged to monitor for HTTP requests occurring on a plurality of connections that are open at the same time for the sending of different chunks of the same resource.

According to the second aspect of the invention there is provided a method of making, from a first station to a second station, an HTTP request concerning a resource, comprising providing the HTTP request with one or more additional headers containing information identifying one or more entities, and checking at the second station whether that entity or combination of entities has permission to access the resource or to execute a requested action.

The second aspect of the invention also provides a resource transfer client software component for transferring a resource over a network between a first station and a second station, comprising:

an HTTP client software component arranged to make an HTTP request to the second station for the transferring of the resource, and

an identity provider arranged to provide the HTTP request with one or more additional headers containing information identifying one or more entities.

The second aspect of the invention further provides a resource transfer server component comprising:

an HTTP server component for receiving HTTP requests for resources,

an identity retriever arranged to retrieve from an HTTP request received by the HTTP server component information identifying one or more entities that is contained in one or more additional HTTP headers, and

a security system arranged to check whether that identity or combination of identities has permission to access the requested resource requested or to execute a requested action.

Further preferred features of the invention are given in the appended claims.

Examples of the invention will now be described, with reference to the accompanying drawings, of which:

- FIGURE 1 is a block diagram of showing the principal components of a preferred example of the data transfer system of the present invention.
- FIGURE 2 is a flow diagram showing the operation of the data transfer system for both upload and download operation.
- FIGURE 3 is a network diagram showing various possible ParHTTP connections.
- FIGURE 4 shows an example of the data transfer system of the invention being used to provide secure transfers.
- FIGURES 5A to 5E show various transfers and transactions possible in the UNICORE distributed computing system.
- FIGURE 6 shows an example of the data transfer system of the invention being used in a security framework.
- FIGURE 7 shows an example of the data transfer system using ordinary HTTP but with the additional headers of the secure aspect of the invention.
- FIGURE 8A and AB show an example of a ParHTTP request and the response thereto.

25

Figure 1 shows a preferred example of the data transfer system of the invention. The system comprises a server 100, a client 200 and a communications network 300, in this case the Internet, which allows data to be transmitted between the server and the client. Here the relative designations "client" and "server" have been given in relation to which initiates transfer of data, i.e. the client initiates the transfer. Nonetheless the system will transfer data in either direction as requested by the client. In this example the

server and clients are general purpose computers, however, the invention is also applicable to dedicated computing appliances.

5 The functions of the system are to transfer a resource either (i) from the server to the client, or (ii) from the client to to the server. In this preferred example both kinds of transfer are supported by the system, but there may be applications in which only one of those two transfers is
10 required and therefore in which only that one need be provided. A resource is a sequence of data values (e.g. bytes) to be transferred. The purpose of the data in the resource transferred may be anything and is of no relevance to the first aspect of the invention. The resource may be a
15 file in a file system, which file may be located on the server or client but equally may be somewhere else. A resource is however not limited to files but may be any other data sequence, for example the output of a sensor or the results output by some computation.

20

Note that Figure 1 shows both the source and target resources (101 and 201 or 201 and 101 depending on the direction of transfer). The target resource need not exist before the transfer is begun; the target resource is created by the
25 transfer process and if it exists before the transfer is begun then it is overwritten (either in its entirety, or partially if that is desired). Alternatively the system can be configured to fail if the target resource exists already. In further alternatives the behaviour as to whether the
30 system overwrites or fails if the target resource exists already is configured at the server and/or is specified in the request from the client.

In Figure 1 the resources are shown as being contained at locations 102 and 202. In the case of files, for example, these locations may be file stores. These file stores may be, for example, on disc or in the working RAM. In the preferred example they are on the server and client respectively, but may be in other places to which the server and client respectively have access. Note however that in some applications the resources may not exist permanently in the way that files do and that the resources may be created and used (at either end of the transfer) only temporarily and possibly they may only exist in part at any one time. Again the precise nature of the resources is of no consequence to the first aspect of the invention which is concerned only with their transfer.

15

The functions of the system are provided by software components on the server and the client. The client has a "ParHTTP" resource transfer client component 203, which is similar to a standard HTTP client component but has modified functions, and the server 100 has, in this example, a standard HTTP server component 103.

20

The standard HTTP server component 103 responds to HTTP requests, from clients, for files (or more generally resources) to be downloaded or uploaded in the usual known way. Known HTTP servers can respond to multiple requests from different client computers, or even from the same client computers, and the HTTP server can service these multiple requests simultaneously, as will be familiar from web servers (which are the usual use of the HTTP protocol). However each request serviced is independent and is generally for a different file (or occasionally is an independent request for the same file).

30

The ParHTTP resource transfer client component 203 functions similarly to a standard HTTP client in that it functions, on each occasion that it is used, to retrieve (or send) a single resource from (or to) a server. (Retrieving a resource is alternatively termed downloading or pulling, and sending a resource is alternatively termed uploading or pushing.) However the client component 203 transforms that request into plural requests for respective parts of that file. These requests are serviced by plural HTTP connections 301 to the server 100 that are set up for the purpose by the client component 203. The client component uses each HTTP connection to process one or more of those requests in turn.

Figure 2 is a flow diagram illustrating the operation of the system of Figure 1 in detail for the case where a resource is to be transferred from the client to the server (i.e. an "upload" or "push" transfer), which is described first. Figure 2 also serves to illustrate the transfer in the other direction which has a similar process.

In a first step 401 the ParHTTP resource transfer client component receives a request 207 (Figure 1) from an external source to upload a resource from the client to the server.

The request is generated, for example, by an application program or by the user entering a command at a command line interface. The request is made using a function call exposed by the resource transfer client component, e.g as part of an API 212 - "application programmer's interface" - or otherwise.

In a second step 402 a chunk divider 204 of the client component 203 determines the size of the resource requested and divides the file into a plurality of chunks or blocks (an

exemplary set of chunks is labelled A to K in Figure 1). In the case of a file, the chunk divider 204 interrogates the file system of the client computer 200 to determine the file size. For other kinds of resources it uses a method
5 appropriate to that resource. Alternatively the file size is included in the information provided in the external request and the chunk divider uses that information.

In the preferred example these chunks are all equal in size,
10 but they may be of different sizes. Also in the preferred example the division into chunks is done once at the beginning of the procedure, but in more complicated arrangements may be done dynamically.

15 The size of chunks depends on factors that are explained in greater detail below. As is also explained below the number of chunks is preferably above a minimum but thereafter is less important than the size. These parameters 205 may, for example, either be predetermined and stored in the chunk
20 divider, or be supplied in the external request 207.

Information on the number, size and position in the resource 201 of the chunks is then passed to a chunk allocator 208. Note that these variables are not independent, so depending
25 on the implementation not all of them need to be passed.

In the next step 404 the chunk allocator retrieves a certain number of the chunks from the resource 201 and initiates respective HTTP transfer requests for them. To do this it
30 uses, in the preferred example, a standard HTTP client 210, making one HTTP request 801 in respect of each chunk. The request is made, in this example, using the HTTP PUT Request. Each Request 801 specifies the same resource (which in HTTP is specified by a URL - "Uniform Resource Locator") for the

target resource 101 but a different range from within that resource, in particular in this example the same range that the chunk has in the source resource 201. The standard HTTP client 210 responds, as is known, to such a PUT Request by
5 creating a TCP socket 209 which in the process creates a corresponding TCP socket 104 at the HTTP server 103. The standard HTTP client 210 retrieves or receives the payload of the PUT Request, namely the one of the chunks specified by the range, from resource 201 and transfers that together with
10 the headers of the PUT Request to the HTTP server 103.

The ParHTTP resource transfer client (as with a standard HTTP client) accepts external requests 207 to transfer, from the client, data in RAM on the client machine or from the file
15 system e.g. on disc (which data may be located on the client machine or on another machine e.g. in a network file share). The ability to send data from RAM is of course useful for an application program that wishes to send the results of its calculations, or data captured from a sensor etc., to the
20 server without having to save it to disc first.

On receipt of such an HTTP PUT Request the HTTP server 103 responds by storing the chunk received therewith to its corresponding position in the target resource 101. (How the
25 transferred data is actually "stored" again depends on the nature of the resource - it may be stored to a file, it may depend on how the server has been configured to treat HTTP requests generally and it may depend on the URL specified in the external request (as is known some URL's cause
30 application programs to be executed which process the data uploaded)). One of the standard, but rarely used, Headers of HTTP is the Range Header and the chunk allocator 208 uses this to specify the position and size of the chunk in the target resource 101. In general the same position in the

source and target resources is used, but the system also allows external the request to specify that either the whole all part of a source resource be transferred to the whole or part of a target resource, and, other than where a whole
5 resource is transferred to a whole resource, the absolute position of the chunks within the source and target resources may be different (but the relative position of the chunks within the source and target resources is maintained).

10 The standard HTTP server comprises a unit for a standard function called here a resource mapper 106. The function of this, as is known, is to translate the URL specified in the PUT command received by the HTTP server 103 into a more directly usable form. Most frequently this will be either the
15 path of a file in the file system (which file may be on the server machine or may be located in another e.g. in a network file share, and which path is determined in many web server administration programs by establishing virtual directories), or that of an application program 115, which may be for
20 example an ASP or PHP program, but the invention is suitable for use with any other language. In the case of a file in the file system the HTTP server 103 saves the chunk at the appropriate position in the file. In the case of an application program the chunk is passed to the application
25 program, which program, of course, should support receiving its file of input data in chunks. Such application programs may save the data chunk to RAM for their own processing or even for passing to other programs which they call. Some gateway applications (which exist for example for SMTP email
30 FTP) are able to send the data received immediately onwards through an SMTP or FTP connection to a final destination and could be implemented as an application program 107.

Thus far the allocation of a single respective chunk to each of a plural number of HTTP connections between the client 200 and the server 100 has been described. In a next step 405 of the operation of ParHTTP resource transfer client 203 the
5 chunk allocator checks to see whether all the chunks in the source resource have now been sent (i.e. the whole resource has been sent). If so the method terminates 406. If not the chunk allocator 208 monitors 407 the progress of the HTTP PUT Requests and allocates additional chunks to the HTTP
10 connections 301. This monitoring is done by the standard HTTP client 210 issuing a call-back or event when the chunk has been sent, to which the chunk allocator 208 responds by providing the HTTP connection 301 that has just finished transmitting a chunk, assuming that that was successful, with
15 another chunk for transmission.

The HTTP connection is kept open for another chunk by specifying the value "keep alive" for the "Connection" Header in each HTTP Request (including both the initial and
20 subsequent ones for a particular HTTP connection), which prevents the connection from being closed automatically once a chunk has been sent and allows a subsequent PUT command for a subsequent chunk to use the same HTTP connection with the same TCP sockets at the client and server be used. This has
25 the advantage that an HTTP connection that sends data quickly is re-used increasing the overall speed of the transfer of the file. (Note that while in HTTP 1.1 "keep alive" is the default behaviour, automatic closure of the connection is the default in earlier versions of the HTTP standard, so the
30 "Connection" header is set to keep alive in order to ensure correct operation whatever version of HTTP is implemented by the standard HTTP client 210 and the standard HTTP server 103.) Each HTTP connection is closed after sending its final chunk. In the preferred example this is done by the chunk

allocator 208 determining which is the final chunk to be sent over an HTTP connection and setting the Connection Header of the HTTP PUT Request for that chunk to "Close" rather than "Keep Alive"; however alternatively it could also be done by
5 allowing the connection to time out, but that is wasteful of computer memory and processing power. (Note that if one of the final chunks should need to be resent once the HTTP connection has been closed in this manner the chunk allocator simply creates a new HTTP connection for that resend.)

10

Alternatively the chunk allocator 208 could be arranged not to use the keep alive command but this would result in each chunk having its own HTTP connection created (because the connection would close after sending its chunk) with the
15 result that the information concerning speedy HTTP connections is not utilised, which is not preferred because this would mean that each HTTP connection would send only a single chunk, which would mean that the advantage of sending further chunks down a connection that was proving to be fast
20 would be lost. Nonetheless some chunks would be sent faster than others, providing some speed gain.

If the standard HTTP client 210 indicates in its call-back or event that a chunk has not been sent successfully (provided
25 from the status code in the server's HTTP Response to the PUT Request) then the chunk allocator responds by rescheduling the chunk for resending and, at some point, allocates that to an available HTTP connection 301. To the extent that the standard HTTP client indicates that part or parts of a chunk
30 has been successfully sent the chunk allocator responds by defining the complementary unsent part or parts of the chunks as chunks themselves and resends those chunks.

The normal status code for a server Response to an HTTP PUT request does not include information on partial successful transfer of the payload of the HTTP PUT request. So the ParHTTP client 203 when making its PUT Requests via the standard HTTP client 210 specifies that the PUT Request includes an instruction to the server to supply in the body of its Response the server's log of progress of processing the PUT Request. The server then sends back items of the PUT Response indicating progress as that is made, i.e. the server sends back a single PUT Response but the lines of its body are not sent all at one but only as progress is made. The chunk allocator monitors these items arriving in the PUT response and if the HTTP connection is broken for some reason, or eventually the server returns a Response message indicating a failure, the chunk allocator then has the information needed to resend the relevant part of the chunk.

Some failure status codes indicate conditions that are such that resending a chunk or part thereof will never succeed and in those cases the ParHTTP returns a failure code to whatever issued the external request 207.

Note that standard HTTP servers and standard client components do not resend data in case of failure.

25

In summary, the method used in the preferred example by the chunk allocator to allocate the chunks is to supply a new chunk to each HTTP connection 301 as soon as it has sent the previous one. Other methods of allocating the chunks would nonetheless be possible but the method described above is quite efficient.

In the example of the system given above the chunk allocator creates the number of parallel HTTP connections given as one

of the arguments 205 of the external request transfer request. Experiments have determined that in some network conditions having only two HTTP connections operating to send chunks at the same time gave the quickest transfer of a
5 resource. This is thought to be because while one HTTP connection is stalled, for example, while a buffer somewhere in the network or in the client or server clears (e.g. while disc access is made), the other connection can utilise the bandwidth, which generally is most limited at the connections
10 of the client station and the server station to the Internet. Similar behaviour occurs for three or more connections working in parallel. In another set of network conditions fifteen HTTP connections operating was found to provide the fastest transfer.

15 Another factor may be that some HTTP connections can be faster or slower at transferring data over the network data. Although in principle each IP packet should be routed individually through the network some routers and traffic
20 management software may cause packets for a particular HTTP connection to follow the same path or have a bandwidth limitation.

As noted above the size of chunks is preset or is passed as
25 an argument in the external request. Preferably the chunk size is not so large that its transfer is likely to fail requiring it to be resent. For example a 5MB chunk is much less likely to fail than one of 500MB.

30 Also the chunks should be small enough to provide enough in number of them to allow the method of sending them via a plural number of parallel HTTP streams to provide transmission speed advantages. There is also the logical minimum that each connection transfers at least one chunk.

Sending exactly one chunk over each of two or more connections operating at the same time provides a basic advantage that any slow connection is not sending all the data. Having at least one more chunk than the number of
5 parallel connections means that those additional chunk(s) will generally be allocated to faster streams. There would not appear to be an advantage to having the limit of a very large number of tiny chunks since each chunk causes some overhead in sending it.

10

The system transfers a resource (or part thereof if desired) from the server to the client using the same method but with the following differences. The transfer is still initiated by a transfer request 207 being made to the ParHTTP resource
15 transfer client 203. In this case the chunk divider determines 402 the file size by sending an HTTP HEAD request to the HTTP server for the whole of the resource that has been requested in the transfer request. This provides the same function as the GET request but the server only returns
20 the header information, omitting the payload of the file itself. This returned header information includes the file size. (tcp sockets 105 and 213 for the HEAD request are shown, in this example they are efficiently reused for the chunk transfers - by using the "keep alive" header). If the
25 server indicates an error (e.g. an unknown resource) then that is passed back to whatever called the ParHTTP resource transfer component 203. The HEAD request is not performed in the case that the external request specifies part of the resource as the size of that will then be known.

30

The chunk allocator then proceeds in the same manner as before to allocate both initial and additional chunks to the plural HTTP connections. In this case, however, it instructs the standard HTTP client 209 to make HTTP GET requests, to

the server, that transfer the chunks from the source resource 101 at the server to the target resource 201 at the client.

During the sending of a particular chunk the chunk allocator
5 monitors the data received in the GET Response for progress
and if the Response fails at any point it notes the unsent
portions of the chunk and defines those as one or more
additional chunks which it later asks to be resent using one
or more additional GET Requests. (Compared to the uploading
10 function there is no need to request the server to send it
log of the transfer.)

Again the resource transferred may be a file from the file
system (whether on the server 100 or elsewhere - e.g. in a
15 network file share) or may be retrieved or generated by an
application program on the server identified by the URL given
in the external request given to the ParHTTP resource
transfer client 203. (Again that program might, for example
be written in ASP or PHP.)

20

The above described method of transferring files between a
client and server may used between any two stations on a
network. Indeed as shown in Figure 3, each server 100 may
establish ParHTTP connections 302 with multiple clients 200
25 (each connection being for, in general, the transfer of a
different file, which will generally be for the different
users or process etc.) and each client may establish plural
ParHTTP connections with multiple servers (again each
connection being for the transfer of a different file, either
30 for the same user or process etc. or for different ones).
Also possible is multiple ParHTTP connections between one
client server pair, which connections may be for the same
user/process (i.e. that of whatever makes the external
request to the ParHTTP resource transfer client) or different

users/processes and these may be for the same file or different files.

Further, each station may be, if desired, both a client and a
5 server (having both an HTTP server and the ParHTTP resource transfer client installed) and be a client for one ParHTTP transfer and server for another.

In the above examples of multiple concurrent ParHTTP
10 connections there may be problems if there are two concurrent requests for the same file. For example if the same file is being written by two different requests then some chunks provided by one request may well be overwritten by ones of the other, which is a result intended by neither of those
15 requests. In general two concurrent read requests are "idempotent" i.e. not dangerous, but one read and one write or two writes are. In principle those problems could be ignored or perhaps usage patterns for a particular application might be such that they would not occur. However,
20 in order to prevent those problems in other cases, the HTTP server 103 is provided with a transactional bracket monitor 107. (Such a monitor is not part of standard HTTP server.) This monitors the HTTP Requests and Responses on all the connections 301 made to the HTTP server 103. From those it
25 identifies those belonging to the same ParHTTP request. Preferably those having the same originating user, process, originating station and URL are identified as belonging to the same request; however other sets of information may be sufficient in some cases. A transactional bracket is defined
30 to exist over the period of when any of those connections belonging to the set identified as belonging to the same ParHTTP request is open and the monitor 109 records the existence of such a bracket for the period that it exists. (It may be still possible for indistinguishable ParHTTP

requests to be generated - e.g for the same user, process, originating station and URL. How this is handled is left to the originating process to handle; i.e. not to do it or to do it in a sensible manner.)

5

Other methods of identifying a transactional bracket are also possible. For example the ParHTTP resource transfer client 203 can be arranged to provide the first and last requests in respect of a particular ParHTTP request with special headers
10 identifying them as such. However such an arrangement is not preferred compared to the method described above of monitoring the open connections.

Having identified the transactional bracket the transactional
15 bracket monitor then prevents concurrent access to the same resource (i.e. preventing further access to a resource during the transactional bracket of a first access to a resource) by either or both of the following methods. (In the preferred example safe idempotent concurrent accesses to the same
20 resources are not so prevented.)

First it signals 108 the HTTP server 103 to refuse further requests for the same resource or it intercepts them and queues them for processing once the transactional bracket is
25 over. Which it does is a matter of configuration.

Second it signals 108 any other transaction processing service 109 relevant to the resource to prevent inconsistent access. For example if the resource is a file and there is a
30 file locking service then it locks 110 the file during the transactional bracket. This is useful against processes other than the HTTP server 103 providing inconsistent access. It may also in some circumstances duplicate the effects of the first method of refusing or delaying other ParHTTP requests,

since the HTTP server would encounter the effects of the other transaction processing service e.g. file locking when it tried to access the resource.

5 In the above example the HTTP connections 301 have been made over TCP/IP. The ordinary HTTP protocol is in fact not limited to using TCP/IP as its transport and since the present invention uses HTTP requests it is similarly not limited to using TCP/IP. Of course given the current state of
10 communications technology the use of TCP/IP is the most likely for most applications.

The ParHTTP communications protocol described above has advantages over the GridFTP protocol in that GridFTP being
15 based on FTP always uses a control channel. This makes GridFTP vulnerable to the loss of the control channel, which is significant in applications such as grid computing where very large files are transferred taking long periods and increasing the risk of failure. In contrast HTTP, which is
20 used in the present is stateless so the loss of any one HTTP connection does not cause the whole ParHTTP connection to fail; the ParHTTP client simply responds by sending the failed chunk again using one of the other HTTP connections in use. (The chunk allocator may also be arranged to open one or
25 more further HTTP connections when one, or perhaps more, of the HTTP connections has been lost.)

Another advantage is that of ease of implementation. HTTP, which of course is used in the invention to transfer the data
30 chunks, already provides the (otherwise seldom used) Range header which provides the invention with a simple mechanism for retrieving, transferring and storing chunks of data. The body portion of HTTP Requests are unstructured making it simple for them to carry the chunk payload in that. In

contrast GridFTP has to use the more complex block mode of GridFTP and FTP request have a complex single line syntax which has to be modified for GridFTP, whereas in ParHTTP the standard HTTP request can be used without modification. (The
5 easily extensible multi-line syntax of HTTP is useful in the second aspect of the invention as will become apparent from the description of that below.)

The above protocol may be used for any purpose desired;
10 nonetheless it is especially useful in grid computing in general, where large files are to be used. A second, secure, example of the invention is described below.

In the second example of the invention shown in Figure 4 the
15 components are the same as those of Figure 1 except that the ParHTTP resource transfer client 203 and server 103 use secure connections for the connections 301'. Preferably this is done using TLS (or its predecessor SSL). TLS is of course a known technology for providing secure connections and so is
20 only described briefly here. In TLS (or SSL), as is known, ordinary socket pairs at the client 200 and server 100 are established first and then secure connections over these are established resulting in the sockets becoming "secure sockets" 104' 209' that can be used for secure data
25 transmission. For TLS the secure connection is established using a public key infrastructure (PKI) and certificates identifying the parties between whom the connection is established. To establish the connection the parties provide respective certificates containing their distinguished names
30 (in particular X.509 DN names).

In this example the ParHTTP resource transfer client 203 establishes the secure connection between the client-server socket pairs using certificates for both the client and the

server ends. (Here client and server are used with their meanings for ParHTTP as was explained in respect of the first example of the invention above.) Once the TLS secure socket pair is established that establishes, as is known, the identities of the parties at the server and client ends. For the server the certificate used is generally a constant, it simply identifies the server. (However if required different certificates can be used on different occasions, e.g. different certificates can be used where one server is used to provide multiple virtual servers.) For the client it identifies an entity related to the making of the ParHTTP request and is, for many applications, preferably supplied by that user or process in the external request 207 made to the ParHTTP resource transfer client. Examples of the entity identified are that of a user or the client station 200 itself. However if the identity supplied is constant then it may be more convenient to supply the certificate as an application setting or on creation of an instance of the client component 203. The ParHTTP client 203 has an identity provider 220 to provide the relevant identity information for the securing of the sockets 209' in accordance with the external request or static identity settings.

The other benefit of using TLS is that the packets sent between client and server are encrypted. Although usual, encryption of the packets sent between the network stations is optional in both TLS and ParHTTP.

(This use of certificates identifying parties at both ends of the connection contrasts for example with secure HTTP - which is known as "HTTPS"- as that is commonly used, which is for the purpose of web browsing of public websites, where only the certificate of the server is used to secure the socket

pairs and encrypt the traffic - the encryption in that case being the main purpose.)

Figure 4 also shows a security system 113. As is known, the resource mapper makes 106 of an HTTP server makes its requests for access to resources at the server in the name of a particular entity. Also as is known, the function of the security system 113 is to monitor those access requests to determine if that entity has permission to do that and prevent access otherwise. The security system is, for example, a file access permission system. The HTTP server 103' is provided with an identity retriever 120 which obtains from the secured sockets 104' the identity used at the client end 200 to secure the connections. In a simple example of the operation of the example of Figure 4 the identity retriever makes that available to the resource mapper 106 which is arranged to make the accesses to resources at the server 100 (or some location to which it has access) in the name of that identity, which may be either directly in that name or by mapping the identity to a local account on the server machine. If the permission does not exist for that identity then the access to the resource is prevented and the HTTP server sends back a message to the client 203 which in turn informs whatever made the external request of the permission failure.

More complex security frameworks, however, allow various parties to be involved in allowing access to a particular resource, execution of a task etc.. (An example involving such a security framework, namely ETD, is given later.)

The secure version of ParHTTP provides, through the establishment of secure sockets as described above, a means of transferring one identity from the ParHTTP client to the

server. ParHTTP also provides a second method of communicating identities between the client 200 and the server 100. This is also implemented in the example of Figure 4 but may be used irrespective of whether the connection is secured. In this method the identity provider 220 adds to each HTTP request 802 sent to the server (i.e. for each chunk, and for PUT and GET) a header for each identity to be transferred. The header is called in the request "X-parhttp-<entity(i)>", where <entity(i)> is a name particularising the role of the i^{th} entity, i.e. party, involved in the request. (In the later ETD example those roles are named "user", "endorser" etc., thus making the name of the header "X-parhttp-user" etc.. The information contained in the additional header for a party identifies the party but its form depends on the security framework being implemented. If the security framework employs PKI the header may well contain the X.509 signature and distinguished name (DN) of the party. In other security frameworks it could be a username and password pair, Kerberos ticket, etc.. The header content is binary data encoded in Base64 to meet the HTTP/1.1 requirement that headers are end encoded in US ASCII.

In some cases it will be convenient to use more than one header for each party, each header containing a portion of the identifying information of a party. For example a pair of headers "X-parhttp-user-sig" and "X-parhttp-user-dn" could contain respectively the X.509 signature and distinguished name for a particular entity if it was convenient for the security system to use the information in that form to check the permissions of that entity.

If secure sockets are used and if one of the identities is transferred using the secure sockets method it is not

essential to repeat that identity in the additional http headers, but may be convenient nonetheless.

On receipt of each HTTP request 802 at the HTTP server 103',
5 the HTTP server submits the chunk to be stored, or requests that it is retrieved, generally as in the first example but before that access is made the identity retriever 120 retrieves the identities from the HTTP request headers and that, if any, transferred to the server as the sockets are
10 secured and provides those identities to the resource mapper 106 make the access. The security system 113 then checks whether that combination of identities has permission to do that (or if only a single identity is received whether that has permission). Again if the permission does not exist the
15 HTTP server 103' sends back a corresponding message to the ParHTTP client, which responds in the same way as noted before.

As noted above the ParHTTP protocol is useful in distributed
20 computing in general. It is further proposed that it should be used in the UNICORE distributed job management and security system. Figures 5A and 5B show some common transfers that are used in the UNICORE distributed job management and security system in its ETD security framework variant.

25

In Figure 5A a user 502 (having the name "Dave") sends 503 data over a network from his workstation C 509 to a server A 504; this transfer is known as a "direct push". Data from server A is retrieved 505 by the user 502; this transfer is
30 known as a "direct pull".

Figure 5B shows another example. The user 502 submits 506 a request for a job to be done on server A, (which transfer is a direct push). Although the data is to be processed on

server A the raw data exists initially in this example on a server E 507. Server A therefore makes a request to server E for the raw data to be transferred 508 to it. The request is made by server A on behalf of the user "Dave". This transfer
5 is known as a "3rd party pull".

In this example the job also specifies that the results are to be returned 510 to server E, which server A does on behalf of user "Dave", which transfer is called a "3rd party push".

10

In the existing versions of UNICORE these transfers are made using, depending on the implementation, the UNICORE Protocol layer (UPL), GridFTP, or some other protocol. In the present invention the transfers are made using ParHTTP.

15

ParHTTP can be used in the UNICORE system (in either the ETD or other variants) to transfer any resource (e.g. data or results, or job requests) between stations. Figure 6 shows a third example of the invention in which ParHTTP is used in
20 the ETD variant of UNICORE. This is generally as described with reference to Figure 4. (The secure version of ParHTTP is used as is preferred, but use of the unsecured version is also possible.)

25 The UNICORE system needs only some modifications to use ParHTTP. In the example of Figure 6 the ParHTTP mechanism for transferring resources is used for both the transfer of data and job requests, etc., i.e. it replaces both GridFTP, the UNICORE Protocol Layer (UPL) etc..

30

Generally in UNICORE the communications between servers uses secure sockets so this aspect requires no change.

In the example of Figure 6 one UNICORE station 100 has a UNICORE component 111, which provides the security and job administration of the UNICORE system, and this is provided with an HTTP server 103'. Another UNICORE station 200 is
5 provided with a UNICORE component 211. This station may also be one providing security and job administration but here we are only concerned with its sending of job requests and issuing resource transfer requests to station 100. Station 200 may be for example, a UNICORE server (e.g. A and E in
10 Figures 5A and 5B) or a workstation where a user is working (e.g. workstation C in Figures 5A and 5B).

When station 200 wishes to transfer a resource to or from, or send ("consign") a job request to, another UNICORE station it
15 makes a request 207 to its ParHTTP resource transfer client 203. UNICORE station 100 has an identity in the UNICORE system and its certificate is used to secure the sockets 104' at its end. If the station 200 has an identity in the UNICORE system (e.g. because it is a UNICORE server) then that is
20 used to secure the sockets at its end. If it is a workstation then the certificate of the user working at it is used to secure the sockets 209'.

In the case of a UNICORE server being the station 200 the
25 identity used to secure the connections 209' is a constant and the at the ParHTTP client is preferably arranged to obtain the station's certificate from an application setting or on creation of an instance of the Par HTTP client 203 by the UNICORE component 211. In the case of a workstation it is
30 preferable for the ParHTTP client to be arranged to accept the user certificate as an argument in the API calls 207 for resource transfers. In ETD terminology the UNICORE server station or user at the ParHTTP client end has the "consigner" role i.e. the party who sends the request and in this example

of the invention the consigner identity is passed through the mechanism of securing the connections.

ETD also provides "user" and "endorser" roles. The user role
5 is the entity (usually a person) for whom the work or transfer is being done. The endorser is the entity asking for it to be carried out, which may be a person (i.e. the user) or a UNICORE server station.

10 UNICORE has its own permission, or "authorisation", database 112 in which the UNICORE component 111 checks for permissions based these identities (as provided by the identity retriever) and their roles before executing the job requested or allowing the resource access requested. In the ETD variant
15 of UNICORE the permission to access a particular resource is dependent on the "user" and "endorser" identities and not generally on the "consigner" identity, rather the UNICORE station generally trusts particular UNICORE server stations and particular users to send it requests. In the example of
20 Figure 6 the UNICORE component 111 is arranged to check the consigner identity to see whether the station 200, or the user, that sent the request (as identified by the establishment of the secure sockets) is allowed in general to consign resource transfer and job requests to it.

25

In the example of Figure 6 the API 212 at the ParHTTP client end provides calls for push and pull requests, which calls have parameters for both the identities of the user and the endorser of a request. In this example the X.509 certificates
30 are used as the identifying information. The chunk allocator 208' is arranged to send both of those in the additional headers of the HTTP requests 802. The two additional headers are therefore called "X-parhttp-user" and "X-parhttp-endorser". In the case of direct pushes and pulls the user

and endorser are the same and whatever is making the request (which is UNICORE component 211 in the case of station 200 being a UNICORE server) simply supplies the same certificate for both roles in the request 207 to the ParHTTP client API.

5 (Such additional headers are termed, in the official IETF HTTP standard, "experimental". The standard allows for their inclusion and provides that they may exist and, for example, that they should be relayed by proxy servers. Their particular names and content are not controlled by the IETF.

10

Having checked the permissions the UNICORE component allows requests for general resources to proceed via the resource mapper 106 to access the resource 102.

15 The UNICORE component 211 in the terminology of Figure 4 a security system 113 but the server may also have other security systems controlling access to resources, for example that of a file system. Again the resource mapper 106 is arranged to make to make requests for the resource (e.g. to

20 file store 102) using the appropriate identities which then are checked by that security system, which is shown in Figure 6 as item 113.

On the other hand, job requests arriving from another station

25 are intercepted by the UNICORE component 111, which checks the permission for permission to execute the job, again using the identities (i) passed as additional header to the HTTP PUT request transmitting the job request and (ii) by the establishment of secure sockets. If there is permission the

30 UNICORE component 111 schedules, queues, etc. the job for execution. When a job is executed at the server 100 that is done in the name of the "user" identity sent in an additional header of the HTTP request that sent the job request. This is

done either using that identity directly or by mapping it to a local user account.

The execution of the job may, as we have seen, cause the station 100 to issue requests to other stations and for that purpose station 100 is provided with its own ParHTTP resource transfer client (not shown in Figure 6). Thus as discussed above in relation to Figure 3 a station may from time to time act as a ParHTTP server or a ParHTTP client.

10

In the UNICORE system job requests are in the form of "Abstract Job Objects" (AJO). These are transferred to a UNICORE server (the station 100 in the example of Figure 6) using a ParHTTP request with the AJO as the payload in the body of the HTTP PUT request generated. In the example of Figure 6 these job requests are signed for the various roles of ETD in the manner described above. In distributed computing the data to be processed is large and ParHTTP of course provides an efficient method of transferring large resources. Job requests are, however, normally much smaller. Therefore for convenience of implementation the UNICORE components are arranged in this example to transfer job requests as a single chunk.

25 The UNICORE component recognises HTTP PUT requests containing job requests as such, for example, by the use of a particular URL in the HTTP PUT request that transfers them or by the content type specified in the "Content-Type" header.

30 Figures 5C to 5E show various kinds of job request transaction in the ETD variant of UNICORE proposed in the cited paper from the Fujitsu Scientific and Technical Journal. (Figures 5C to 5E are taken from that paper.) Again

these transactions are supported by the Figure 6 example of ParHTTP as noted below.

The Figures show stations on a network and connections
5 between them. The connections between them are secured using Secure Sockets Layer (SSL) security using X.509 certificates at either end verified by a certification authority. The certificate used at the initiating end is identified with a "T" symbol attached to the connection, which is either that
10 of the server at that end or that of the user) and the certificate used at the other end is that of the server at the other end. All the transactions concern the transfer of requests for computing work to be done the stations labelled "Server", or "Server A" and "Server B". These requests are
15 labelled "Job" and may include requests for component parts of the work labelled "Sub Job". The requests identify the "user" under whose personality the work is to be executed at the server. An exemplary user identity of is labelled as "Dave" in the diagram. Each job is also "endorsed" either
20 with the user's X.509 certificate or with that of one of the stations as is explained further below. The signature is shown as the starburst symbol and the broad arrow pointing to some of these indicate the act of signing the work request. The user and endorser signatures are passed between stations
25 in the additional HTTP headers of ParHTTP.

In Figure 5C the user, "Dave" again, working at a workstation, signs his own job request and passes, or "consigns" in ETD terminology, this to the server over the
30 secure connection to UNICORE software component running on the server. This checks in the UNICORE permission, or "authorisation", database that the user, Dave, has permission to run the job on the server and if so causes it to be executed.

In Figure 5D the user, Dave, again working at the workstation signs a job request as both the "user" and "endorser" and similarly also a component sub-job request and passes these to Server A using ParHTTP again passing those signatures as additional HTTP headers. Server A forwards, or "consigns", the sub-job to Server B via a secure connection. Servers A and B then check in the UNICORE authorisation database that the user, Dave, as identified in the additional headers has permission to run their respective jobs and cause them to be executed.

In Figure 5E the user, Dave, working at a client station contacts a job submission website at a portal station, using an ordinary secure HTTP connection except that the user's certificate is provided to secure the connection in order to inform the portal of the user's identity. The portal trusting the identity of Dave compiles a job request (again comprising a main job and a sub job) using the user's input to the web pages of the website and then signs the job on behalf of the user. The portal station then "consigns" the job requests to Server A which then in turn "consigns" the sub job to Server B. Servers A and B then both, before causing their respective jobs to be executed, check in their respective UNICORE authorisation databases both that the user has permission to execute their respective jobs and that the Portal station has permission to endorse the job requests on behalf of the user, Dave. These checks are made using the identities transferred in the additional HTTP headers. They also respectively check that the Portal station and server A have permission to consign job requests to them using the identities transferred by securing the connections.

The above examples have shown that ParHTTP can transfer identities using two methods, namely through the establishment of secure connections and through the addition of extra headers to HTTP requests. Although this has been in the context of ETD and specifically ETD for UNICORE, ParHTTP can be used to transfer identities in other security systems or other applications generally. Further as the example of sending a job request in only one chunk illustrates, the parallel transmission of chunks over multiple connections aspect of ParHTTP is not essential to the ParHTTP methods of transferring the identities. These methods can be used in ordinary HTTP requests and a security system at the server can still use the identity information transferred to check permissions for access to the resource etc. requested. Figure 7 shows such an arrangement where a standard HTTP client is used to make a single HTTP request with the additional headers identifying one or more entities, with either one or no party being identified by the securing of the connection.

To make the request the standard HTTP client generates a single TCP/IP connection, optionally securing it using the connection. Optionally one entity is identified by using its identity at the client end in the process of securing the connection.

The additional header(s) identifying one or more entities are compiled by whatever requests the HTTP client to do that and that then uses the standard functions of the HTTP client to add them to the request. Alternatively a modified form of standard HTTP client is provided having an identity provider to assist with the compilation of the additional headers and their addition to the HTTP request. This is responsive to external requests made to an API of the modified client. As in previous examples the identities

may be provided in the arguments of the API calls or as static values, for example retrieved from application settings. The identity provider also, if desired, causes an identity of an entity to be used at the client end in
5 securing the connection.

In this example multiple connections are not being used the client 210 or 203' is arranged to provide a single connection and a single HTTP request.

10

As with the earlier examples on reception of the HTTP request 803 the HTTP server 103' examines the request and makes an access to the requested resource 101 (e.g. a file 101 in a file store 102 etc.) The server 103' is provided with a
15 resource mapper 106 and an identity retriever 120 to provide it with the identities from the additional headers in the HTTP request (and optionally that used at the client end to secure the connection) to the resource mapper so it can make its access in the names of those. A security system 103 at
20 the server allows those accesses if those identities have the relevant permissions.

Figures 8A shows the text of the headers of a typical ParHTTP request, which shows both headers for transferring the
25 certificates and the range header. As noted above in some embodiments of the invention either of those may be omitted. The text of the Figure has some annotations in square brackets; "[...]" indicates more similar data.

30 Figure 8B shows the response to the request. The data after "Content-Type: application/octet-stream" is binary and has been shown as random characters for the purpose of illustration.

CLAIMS:

1. A method of transferring a resource over a network between a first station and a second station, the method comprising:
dividing the resource into a plurality of chunks,
providing a plurality of connections over the network between the first and second stations, the connections being open at the same time, and
sending the chunks between the first and second station, at least one chunk being sent over each of the plurality of connections,
wherein each chunk is sent over one of the connections of the plurality by making a hypertext transfer protocol (HTTP) request from the first station to the second station.
2. A method according to claim 1 wherein one or more of the plurality of connections are each used to send two or more of the chunks of the resource, each connection used to send two or more chunks being kept open between those chunks.
3. A method according to claim 2 wherein, in response to a connection having completed sending a chunk, another chunk, if any remain, is sent via that connection.
4. A method according to any preceding claim wherein if the sending of one of the chunks fails it is resent over one of the connections.
5. A method according to any preceding claim wherein if the sending of one of the chunks fails in part, the unsent parts are resent over one or more of the connections.

6. A method according to any preceding claim wherein the HTTP request used to send a chunk utilises the HTTP Range header to specify a range for the chunk within the resource to be transferred.

7. A method according to any preceding claim wherein the protocol used to send chunks is the secure version (HTTPS) of the hypertext transfer protocol.

8. A method according to any preceding claim wherein the resource is transferred from the first station to the second station.

9. A method according to any preceding claim wherein the resource is transferred from the second station to the first station.

10. A method according to any preceding claim comprising monitoring at the second station to determine when the sending of the chunks of the plurality remains in progress.

11. A method according to claim 10 comprising preventing at least one type of concurrent access to the resource during a period that the sending of the chunks of the plurality remains in progress.

12. A method according to claim 10 or claim 11 wherein the monitoring comprises monitoring for when at least one of the connections being used to send the chunks of the resource remains open.

13. A method according to claim 10 or claim 11 wherein the first station sends via one or more of the connections information indicative of when the chunks have all been sent and the monitoring comprises monitoring for that information.

14. A method according to any preceding claim wherein the resource is a file.

15. A method according to any preceding claim wherein the resource is an ordered stream of data.

16. A method according to any preceding claim wherein the resource is of a predetermined size.

17. A method according to any preceding claim wherein the connections are secured in a manner identifying to the second station an entity determined by the first station.

18. A method according to claim 17 wherein the connections are secured using certificates.

19. A method according to claim 17 or claim 18 comprising checking at the second station that the entity identified to the first station has permission to access the resource or to execute a requested action.

20. A method according to any one of claims 1 to 16 comprising providing the HTTP requests for transferring the chunks with one or more additional headers containing information identifying one or more entities, and checking at the second station whether that entity or combination of entities has permission to access the resource or to execute a requested action.

21. A method according to any one of claims 17 to 19 comprising providing the HTTP requests for transferring the chunks with one or more additional headers containing information identifying one or more entities, and checking at the second station whether that or those entities and that identified to the second station in the securing of the connections have in combination permission to access the resource or to execute a requested action.

22. The use of the method of transferring a resource as claimed in any preceding claim to transfer job requests between UNICORE stations.

23. The use of the method of transferring a resource as claimed in any one of claims 1 to 21 to transfer resources to be used in, or that are the results of, a distributed computing task.

24. A resource transfer client software component for transferring a resource over a network between a first station and a second station, comprising:

- a chunk divider arranged to divide the resource into a plurality of chunks, and

- a chunk allocator arranged to make requests to the second station for the sending of the chunks between the first and second station over the network, at least one chunk being sent over each of a plurality of connections, the connections being open at the same time,

- wherein the chunk allocator is arranged to make for the sending of each chunk a hypertext transfer protocol (HTTP) request to send the chunk over one of the connections of the plurality.

25. A resource transfer client software component according to claim 24 wherein the chunk allocator is arranged to allocate the sending of two or more chunks over each of one or more of the connections of the plurality, and to keep open between those chunks each connection used to send two or more chunks.

26. A resource transfer client software component according to claim 25 wherein the chunk allocator is responsive to a connection having completed sending a chunk to request that another, if any remain, is sent over that connection.

27. A resource transfer client software component according to any one of claims 24 to 26 wherein the chunk allocator is responsive to the sending of one of the chunks failing to request its resending over one of the connections.

28. A resource transfer client software component according to any one of claims 24 to 27 wherein the chunk allocator is responsive to the sending of one of the chunks failing in part to request the resending of the unsent parts over one or more of the connections.

29. A resource transfer client software component according to any one of claims 24 to 28 wherein chunk allocator is arranged to make each HTTP request used to send a chunk utilising the HTTP Range header to specify a range for the chunk within the resource to be transferred.

30. A resource transfer client software component according to any one of claims 24 to 29 arranged to request the sending of chunks using the secure version (HTTPS) of the hypertext transfer protocol.

31. A resource transfer client software component according to any one of claims 24 to 30 wherein the chunk allocator is arranged to make the HTTP requests to transfer the resource from the first station to the second station.

32. A resource transfer client software component according to any one of claims 24 to 28 wherein the chunk allocator is arranged to make the HTTP requests to transfer the resource from the second station to the first station.

33. A resource transfer client software component according to any one of claims 24 to 32 arranged to secure the connections used for the HTTP requests in a manner identifying to the second station an entity determined by the first station.

34. A resource transfer client software component according to claim 33 arranged to secure the connections using certificates.

35. A resource transfer client software component according to any one of claims 24 to 34 comprising an identity provider arranged to provide the HTTP requests for transferring the chunks with one or more additional headers containing information identifying one or more entities.

36. The use of resource transfer client software component according to any one of claims 24 to 35 to transfer job requests between UNICORE stations.

37. The use of resource transfer client software component according to any one of claims 24 to 35 to transfer resources to be used in, or that are the results of, a distributed computing task.

38. A resource transfer server software component comprising
an HTTP server component, and
a transactional bracket monitor arranged to monitor for
HTTP requests occurring on a plurality of connections that
are open at the same time for the sending of different chunks
of the same resource.

39. A resource transfer server software component according
to claim 38 wherein the transactional bracket monitor is
arranged to prevent at least one type of concurrent access to
the resource during a period that the sending of the chunks
of the plurality remains in progress.

40. A resource transfer server software component according
to claim 38 or claim 39 wherein the transactional bracket
monitor is arranged to monitor for when at least one of the
connections being used to send the chunks of the resource
remains open.

41. The use of resource transfer server software component
according to any one of claims 39 to 41 to transfer job
requests between UNICORE stations.

42. The use of resource transfer server software component
according to any one of claims 39 to 41 to transfer resources
to be used in, or that are the results of, a distributed
computing task.

43. A method of making, from a first station to a second station, an HTTP request concerning a resource, comprising providing the HTTP request with one or more additional headers containing information identifying one or more entities, and checking at the second station whether that entity or combination of entities has permission to access the resource or to execute a requested action.

44. A method of making an HTTP request according to claim 43 comprising securing the connection over which the HTTP request is made in a manner identifying to the second station an entity determined by the first station, and wherein the checking of permission at the second station is of whether that entity in combination with that or those entities identified to the second station in the additional header or headers of the HTTP request have permission to access the resource or to execute the requested action.

45. A method according to claim 44 wherein the connections are secured using certificates.

46. A method according to claim 44 or claim 45 wherein the entity identified in the securing of the connection is the first station itself.

47. A method according to any one of claims 43 to 46 wherein the, or a one of, the entities identified in the additional headers of the HTTP request is a user for whom the resource is being transferred or the action is being carried out.

48. A method according to any one of claims 43 to 47 wherein the, or a one of the, entities identified in the additional headers of the HTTP request is an entity who is making the request on behalf of a user for whom the resource is being transferred or the action is being carried out.

49. The use of method of making an HTTP request according to any one of claims 43 to 49 to transfer job requests between UNICORE stations.

50. The use of method of making an HTTP request according to any one of claims 43 to 49 to transfer resources to be used in, or that are the results of, a distributed computing task.

51. A resource transfer client software component for transferring a resource over a network between a first station and a second station, comprising:

- an HTTP client software component arranged to make an HTTP request to the second station for the transferring of the resource, and

- an identity provider arranged to provide the HTTP request with one or more additional headers containing information identifying one or more entities.

52. A resource transfer client software component according to claim 51 wherein the identity provider is arranged to provide an identity of an entity to secure the connection over which the HTTP request is made in a manner identifying that entity to the second station.

53. A resource transfer client software component according to claim 52 wherein identity provider is arranged to secure the connection using certificates.

54. A resource transfer client software component according to claim 52 or claim 53 wherein identity provider is arranged to secure the connection with the identity of the first station itself.

55. A resource transfer client software component according to any one of claims 51 to 54 wherein identity provider is arranged to provide the HTTP request with one or more additional headers identifying a user for whom the resource is being transferred or an action is being carried out.

56. A resource transfer client software component according to any one of claims 51 to 55 wherein identity provider is arranged to provide the HTTP request with one or more additional headers identifying an entity who is making the request on behalf of a user for whom the resource is being transferred or an action is being carried out.

57. The use of resource transfer client software component according to any one of claims 51 to 56 to transfer job requests between UNICORE stations.

58. The use of resource transfer client software component according to any one of claims 51 to 56 to transfer resources to be used in, or that are the results of, a distributed computing task.

59. A resource transfer server component comprising:

an HTTP server component for receiving HTTP requests for resources,

an identity retriever arranged to retrieve from an HTTP request received by the HTTP server component information identifying one or more entities that is contained in one or more additional HTTP headers, and

a security system arranged to check whether that identity or combination of identities has permission to access the requested resource requested or to execute a requested action.

60. A resource transfer server component according to claim 59 wherein for an HTTP request made over a secure connection the identity retriever is arranged to retrieve an identity of an entity used to secure the connection at the client end, and

wherein the security system is arranged to check whether that entity in combination with that or those entities identified to the second station in the additional header or headers of the HTTP request have permission to access the resource or to execute the requested action.

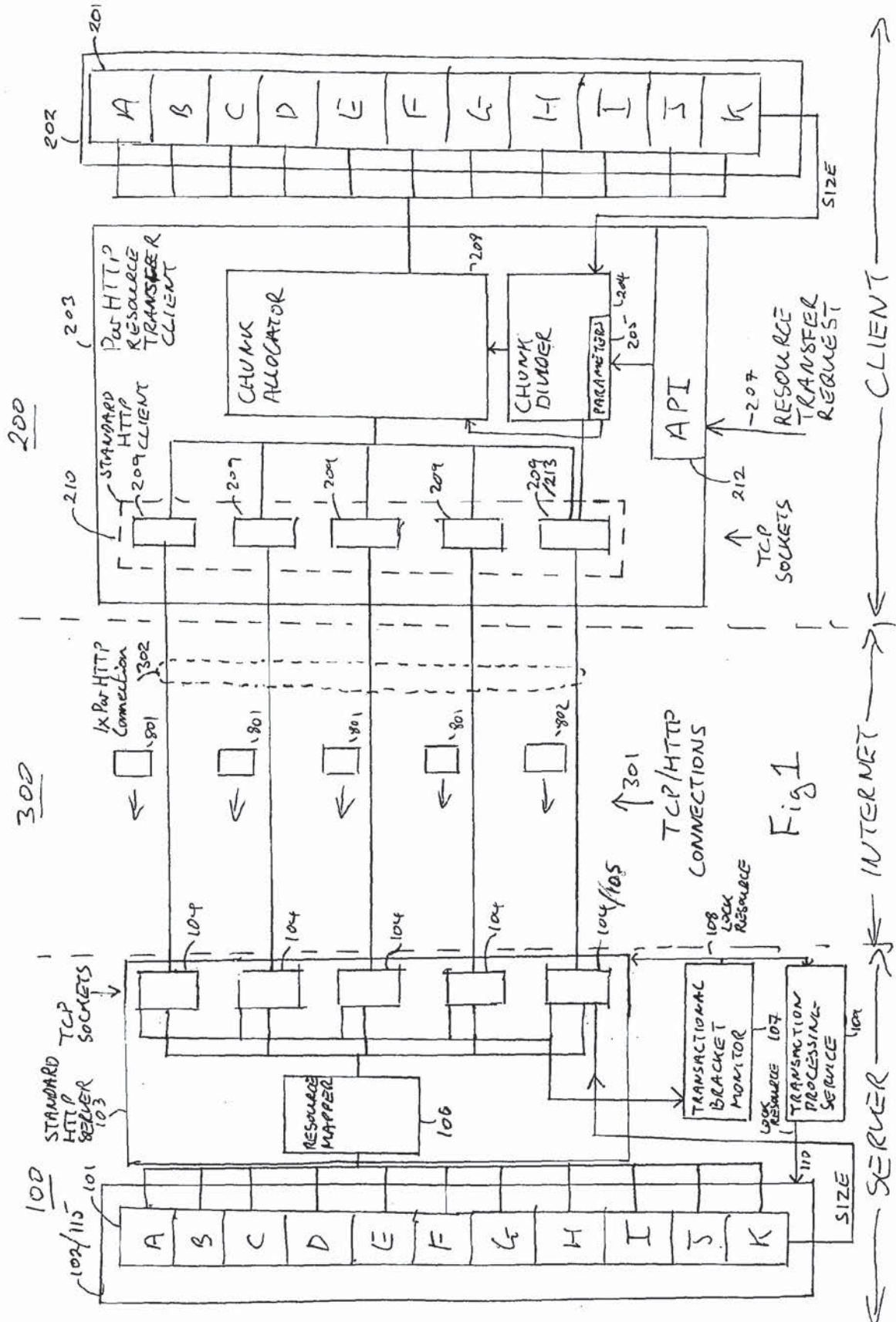
61. The use of resource transfer server software component according to claim 59 or claim 60 to transfer job requests between UNICORE stations.

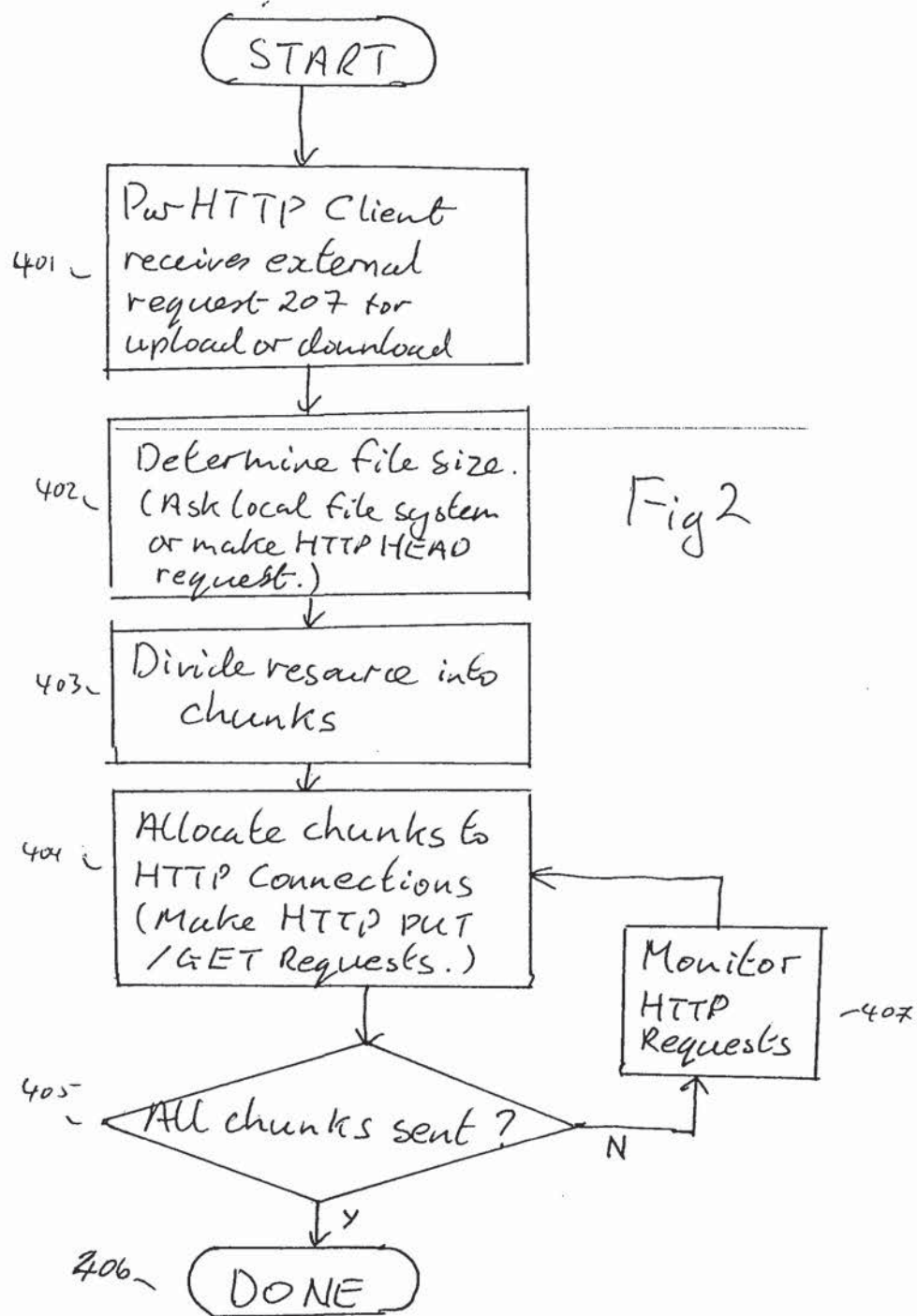
62. The use of resource transfer server software component according to claim 59 or claim 60 to transfer resources to be used in, or that are the results of, a distributed computing task.

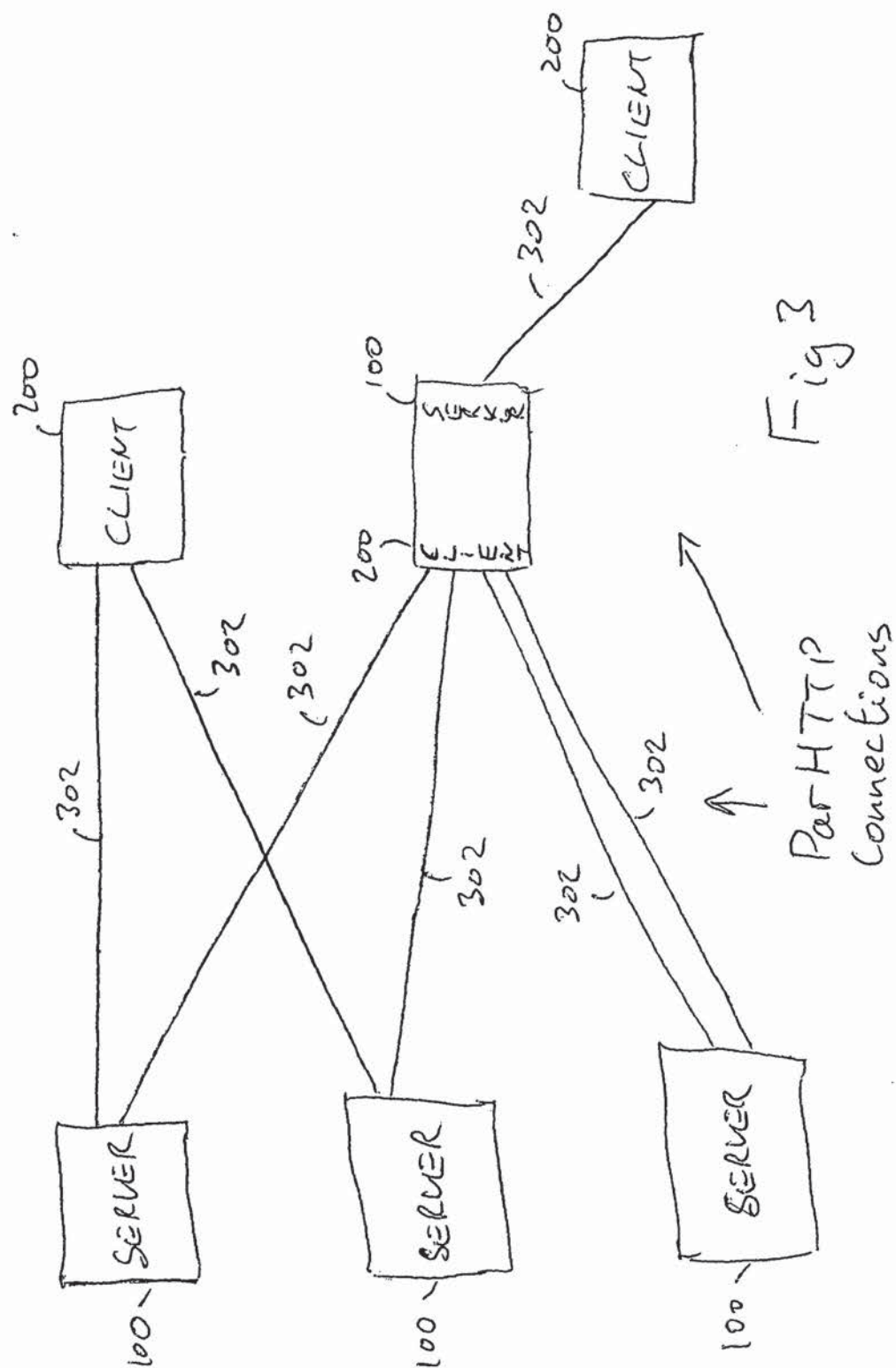
63. A method according to any one of claims 1 to 21 and according to any one of claims 43 to 48.

64. A resource transfer client software component according to any one of claims 24 to 35 and according to any one of claims 51 to 56.

65. A resource transfer server software component according to any one of claims 38 to 40 and according to claim 59 or claim 60.







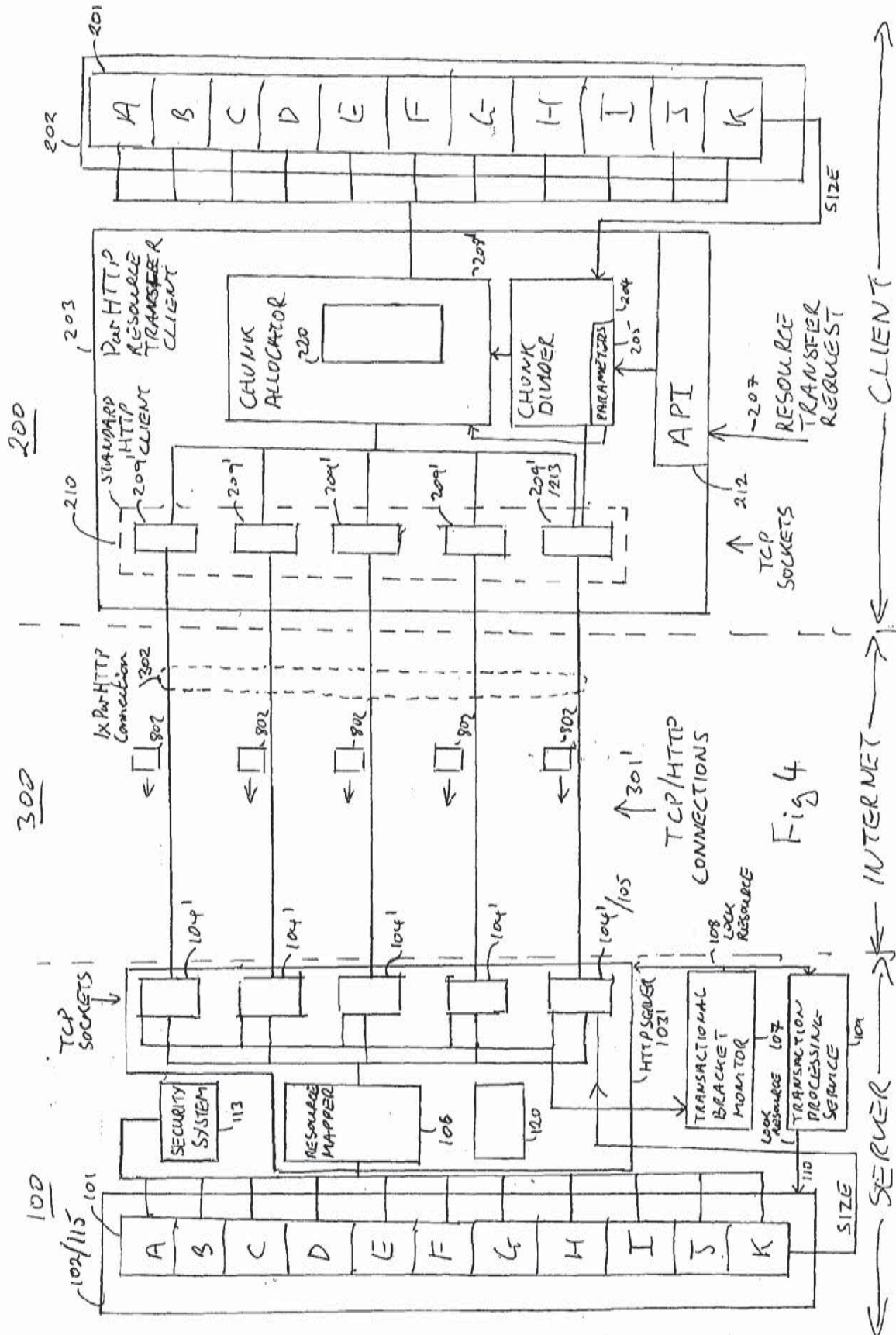


Fig 4

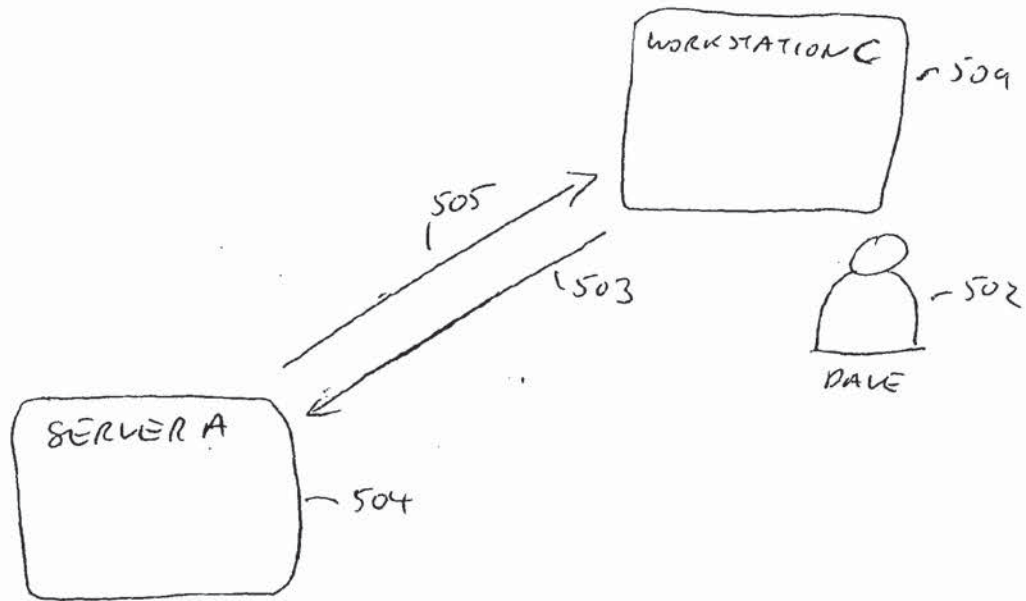


Fig 5A

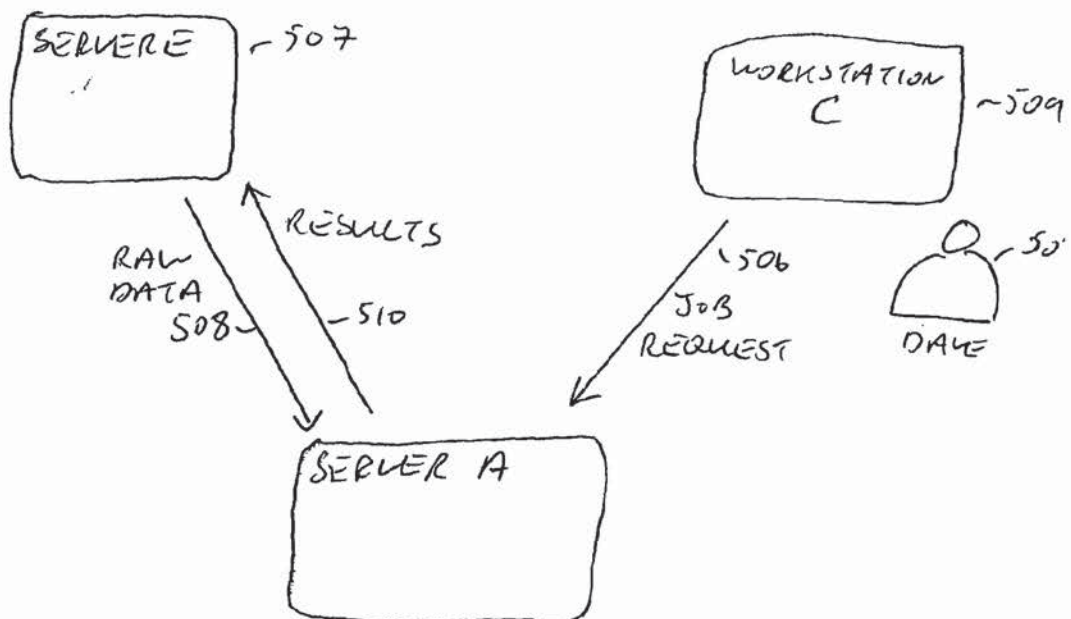
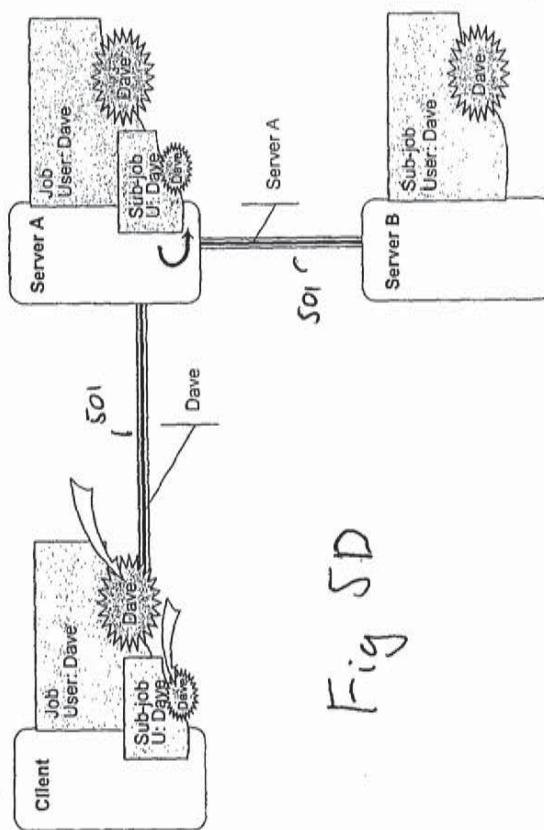
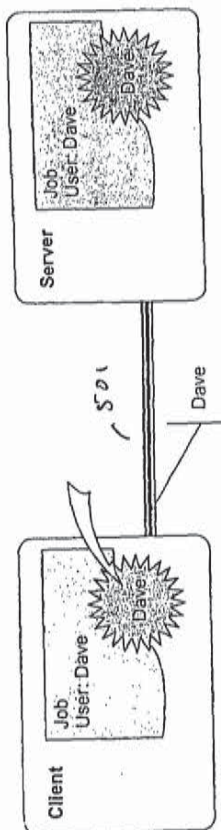
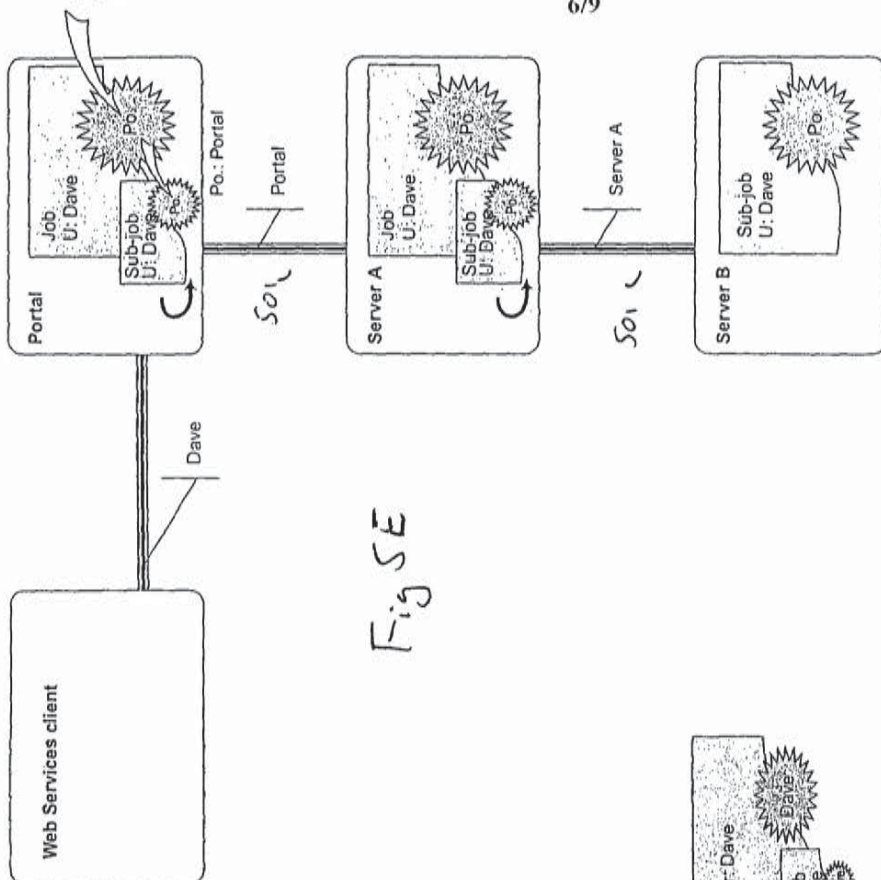


Fig 5B



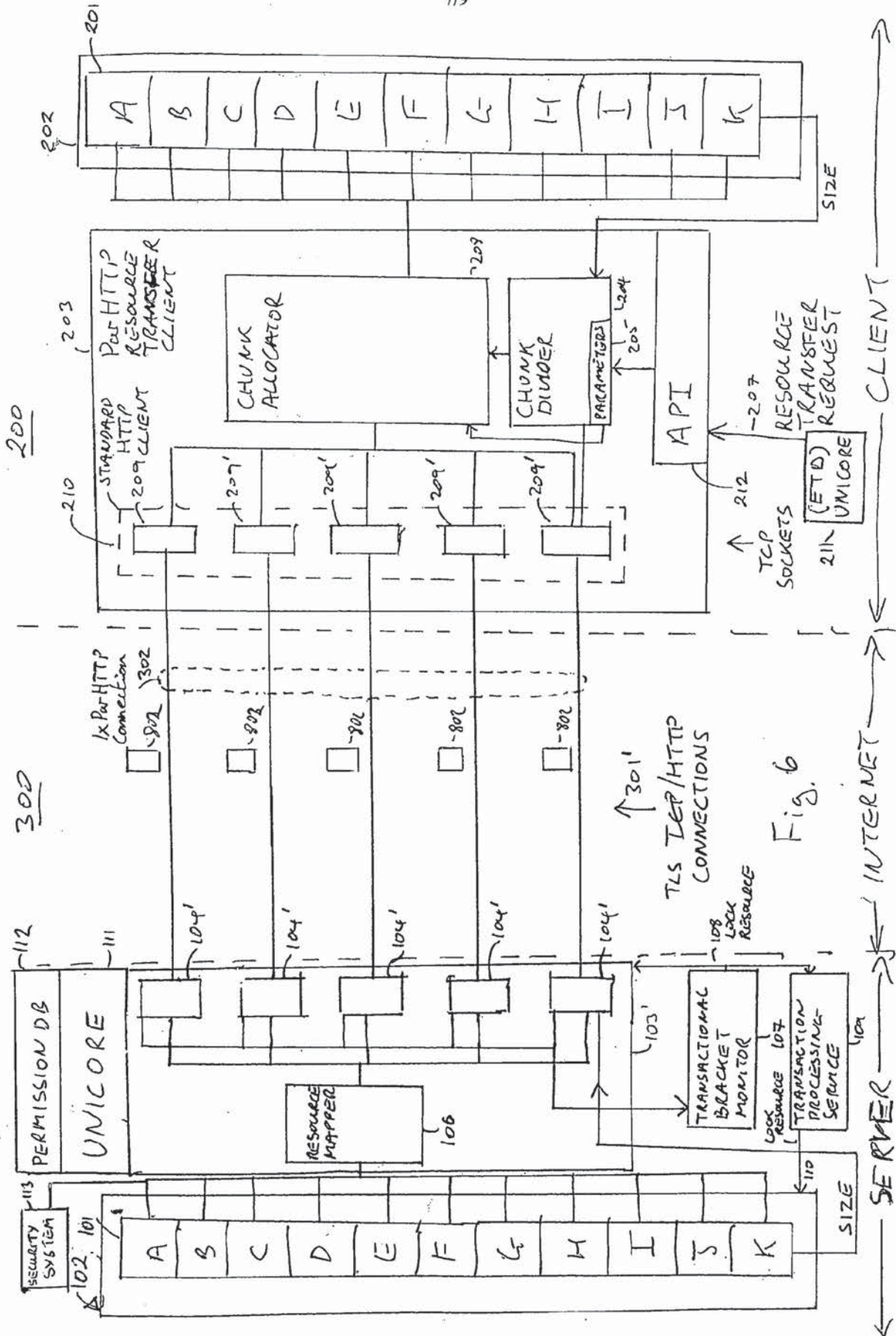
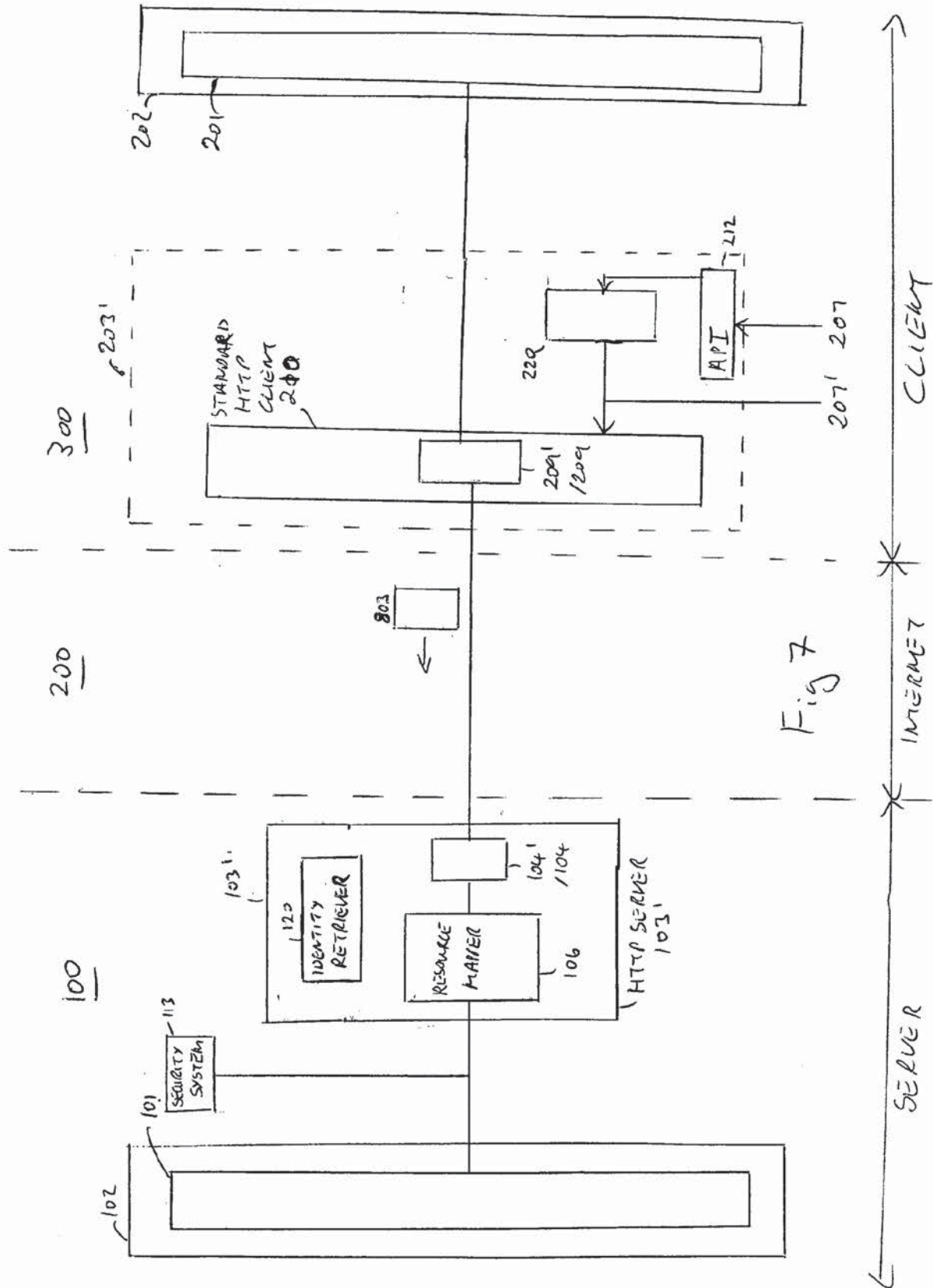


Fig. 6



GET /foo/huge-picture.gif HTTP/1.1

Host: www.example.org
 Range: bytes=0-1048575
 Date: Tue, 8 Aug 2006 16:25:31 GMT
 User-Agent: Mozilla/5.0
 Connection: Keep-Alive
 X-parHTTP-User: MIIDmzCCA0OgAwIBAgIBZTANBgkqhkiG9w0BAQQFADCBpZELMAGAlUEBh[...]
 X-parHTTP-Endorser: MIIEVzCCA6egAwIBAgIBbZANBgkqhkiG9w0BAQQFADCBpZELMAGAlUEBh[...]
 X-parHTTP-Consignor: FyJIIETCCA6mgAwIBAgIBcTANBgkqhkiG9w0BAQQFADCBpZELMAGAlUEBh[...]
 [more HTTP headers]
 [empty request body]

Fig 8A

HTTP/1.1 206 Partial Content
 Date: Tue, 8 Aug 2006 16:25:33 GMT
 Server: Apache/2.0.54 (Debian GNU/Linux) ...
 Accept-Ranges: bytes
 Connection: Keep-Alive
 Keep-Alive: timeout=15, max=100
 Content-Range: bytes 0-1048575/10485760
 Content-Length: 1048576
 Content-Type: application/octet-stream
 GIFS9a\T\$.<..4 J+ O... *D+Q..&\$T.%....Qa E'b.' \$Z.6.
 N...8. .R..1,jT.&F. .) (5* .T.; .V.f*.7((q.Dr. .'.p.O Aj.\$6.-
 .UUU. .g."p.d.\$ <.#FH4 O)5...4. .WR. .'.PuZ.n..B. .F-I. . .!Er.*p
 .R.COXP. .Q\$.T.i>1 u. <Ly. . .Jn. .7. .Y54%,i.\$#. (#1..
 [...]

Fig 8B

INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2006/008710

A. CLASSIFICATION OF SUBJECT MATTER
INV. H04L29/08 H04L29/06 G06F21/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
H04L G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

EPO-Internal, WPI Data, INSPEC, COMPENDEX

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2005/273592 A1 (PRYOR ROBERT F [US] ET AL) 8 December 2005 (2005-12-08)	1-16, 22-32, 36-42
Y	paragraph [0005] paragraph [0008] - paragraph [0010] paragraph [0017] - paragraph [0018] paragraph [0020] paragraph [0032]	17-21, 33-35
X	WO 2005/109224 A (MOVE NETWORKS [US]) 17 November 2005 (2005-11-17) page 11, line 17 - page 12, line 19 ----- -/--	1-16, 22-32, 36-42

☒ Further documents are listed in the continuation of Box C.

☒ See patent family annex.

* Special categories of cited documents:

- *A* document defining the general state of the art which is not considered to be of particular relevance
- *E* earlier document but published on or after the international filing date
- *L* document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
- *O* document referring to an oral disclosure, use, exhibition or other means
- *P* document published prior to the international filing date but later than the priority date claimed

- *T* later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
- *X* document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
- *Y* document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
- * & * document member of the same patent family

Date of the actual completion of the international search

19 July 2007

Date of mailing of the international search report

26/07/2007

Name and mailing address of the ISA/
European Patent Office, P.B. 5818 Patentlaan 2
NL - 2280 HV Rijswijk
Tel. (+31-70) 340-2040, Tx. 31 651 epo nl,
Fax: (+31-70) 340-3016

Authorized officer

Huber, Oliver

INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2006/008710

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2002/162002 A1 (GUNTER CARL A [US] ET AL) 31 October 2002 (2002-10-31)	43-65
Y	paragraph [0045] paragraph [0054] paragraph [0085]	17-21, 33-35
X	----- EP 1 443 792 A1 (NTT DOCOMO INC [JP]) 4 August 2004 (2004-08-04) paragraph [0051] paragraph [0064]	43-65
A	----- US 2006/074894 A1 (REMAHL THOMAS [US] ET AL) 6 April 2006 (2006-04-06) figures 1,2	43-65
A	----- US 2005/287990 A1 (MONONEN RISTO [FI] ET AL) 29 December 2005 (2005-12-29) paragraph [0036] - paragraph [0049] -----	43-65

INTERNATIONAL SEARCH REPORT

International application No.
PCT/EP2006/008710

Box II Observations where certain claims were found unsearchable (Continuation of item 2 of first sheet)

This International Search Report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:
2. ☐ Claims Nos.:
because they relate to parts of the International Application that do not comply with the prescribed requirements to such an extent that no meaningful International Search can be carried out, specifically:
3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box III Observations where unity of invention is lacking (Continuation of item 3 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

see additional sheet

1. ☒ As all required additional search fees were timely paid by the applicant, this International Search Report covers all searchable claims.
2. ☐ As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.
3. ☐ As only some of the required additional search fees were timely paid by the applicant, this International Search Report covers only those claims for which fees were paid, specifically claims Nos.:
4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this International Search Report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest

- ☐ The additional search fees were accompanied by the applicant's protest.
- ☒ No protest accompanied the payment of additional search fees.

FURTHER INFORMATION CONTINUED FROM PCT/ISA/ 210

This International Searching Authority found multiple (groups of) inventions in this international application, as follows:

1. claims: 1-16,22-32,36-42

Transfer of a resource over a plurality of connections.

2. claims: 17-21,33-35,43-65

Providing identity information in a HTTP header.

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No

PCT/EP2006/008710

Patent document cited in search report		Publication date	Patent family member(s)		Publication date
US 2005273592	A1	08-12-2005	CN	1700634 A	23-11-2005
WO 2005109224	A	17-11-2005	AU	2005241481 A1	17-11-2005
			CA	2564861 A1	17-11-2005
			EP	1743249 A2	17-01-2007
			US	2005262257 A1	24-11-2005
US 2002162002	A1	31-10-2002	NONE		
EP 1443792	A1	04-08-2004	CN	1520226 A	11-08-2004
			CN	1953616 A	25-04-2007
			JP	2004236182 A	19-08-2004
			TW	242350 B	21-10-2005
			US	2004192195 A1	30-09-2004
US 2006074894	A1	06-04-2006	NONE		
US 2005287990	A1	29-12-2005	CN	1977514 A	06-06-2007
			EP	1763947 A1	21-03-2007
			WO	2006003499 A1	12-01-2006