PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F	A2	(11) International Publication Number: WO 99/46660 (43) International Publication Date: 16 September 1999 (16.09.99)

(21) International Application Number: PCT/US99/05248

(22) International Filing Date: 11 March 1999 (11.03.99)

(30) Priority Data:

09/041,149	12 March 1998 (12.03.98)	US
09/100,914	19 June 1998 (19.06.98)	US
09/239,475	28 January 1999 (28.01.99)	US

(71) Applicant *(for all designated States except US)*: FAIRBANKS SYSTEMS GROUP [US/US]; 3550 General Atomics Court, San Diego, CA 92121-1194 (US).

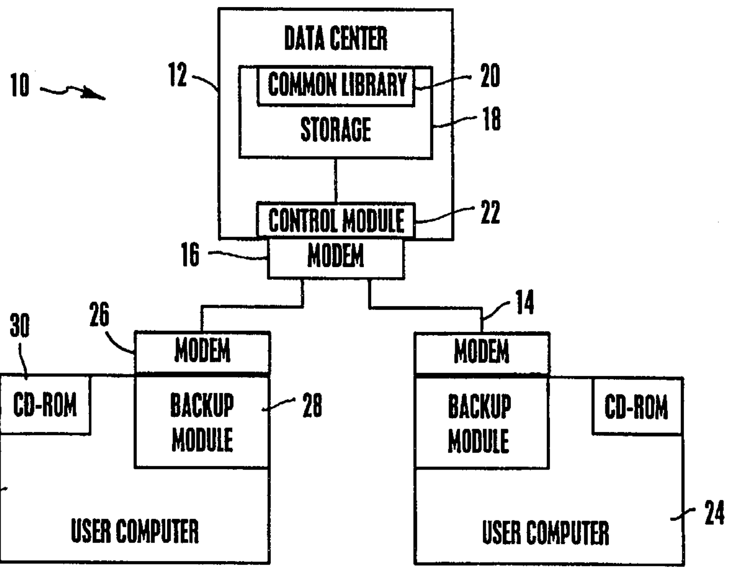
(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: MICKELSEN, Stephen, Peter [CA/US]; 543 Gardendale Road, Encinitas, CA 92024 (US). BOLT, Thomas, B. [US/US]; 279 Horizon Drive, Encinitas, CA 92024-4147 (US). MCCLAIN, Fred, W. [US/US]; 6886 Circo Diegueno, Del Mar, CA 92014 (US).

(74) Agent: ROGITZ, John, L.; Rogitz & Associates, Suite 3120, 750 B Street, San Diego, CA 92101 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, HR, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: SYSTEM AND METHOD FOR BACKING UP COMPUTER FILES OVER A WIDE AREA COMPUTER NETWORK



(57) Abstract

A laptop computer back up logic system (300) copies user selected files or automatically selected files for back up as the laptop computer (24) is being used. Changed files are backed up and stored in a hold area of the laptop computer (24). When the laptop computer (24) senses a network connection and the network (14) is idle, the blocks of the changed files are transmitted from the hold area to a remote facility (12) via the network (14). The backing up process can be triggered within the laptop computer (24) when the screen saver of the laptop computer (24) is invoked.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania
AM	Armenia
AT	Austria
AU	Australia
AZ	Azerbaijan
BA	Bosnia and Herzegovina
BB	Barbados
BE	Belgium
BF	Burkina Faso
BG	Bulgaria
BJ	Benin
BR	Brazil
BY	Belarus
CA	Canada
CF	Central African Republic
CG	Congo
CH	Switzerland
CI	Côte d'Ivoire
CM	Cameroon
CN	China
CU	Cuba
CZ	Czech Republic
DE	Germany
DK	Denmark
EE	Estonia
ES	Spain
FI	Finland
FR	France
GA	Gabon
GB	United Kingdom
GE	Georgia
GH	Ghana
GN	Guinea
GR	Greece
HU	Hungary
IE	Ireland
IL	Israel
IS	Iceland
IT	Italy
JP	Japan
KE	Kenya
KG	Kyrgyzstan
KP	Democratic People's Republic of Korea
KR	Republic of Korea
KZ	Kazakstan
LC	Saint Lucia
LI	Liechtenstein
LK	Sri Lanka
LR	Liberia
LS	Lesotho
LT	Lithuania
LU	Luxembourg
LV	Latvia
MC	Monaco
MD	Republic of Moldova
MG	Madagascar
MK	The former Yugoslav Republic of Macedonia
ML	Mali
MN	Mongolia
MR	Mauritania
MW	Malawi
MX	Mexico
NE	Niger
NL	Netherlands
NO	Norway
NZ	New Zealand
PL	Poland
PT	Portugal
RO	Romania
RU	Russian Federation
SD	Sudan
SE	Sweden
SG	Singapore
SI	Slovenia
SK	Slovakia
SN	Senegal
SZ	Swaziland
TD	Chad
TG	Togo
TJ	Tajikistan
TM	Turkmenistan
TR	Turkey
TT	Trinidad and Tobago
UA	Ukraine
UG	Uganda
US	United States of America
UZ	Uzbekistan
VN	Viet Nam
YU	Yugoslavia
ZW	Zimbabwe

SYSTEM AND METHOD FOR BACKING UP COMPUTER FILES OVER A WIDE AREA COMPUTER NETWORK

RELATED APPLICATIONS

The present application is a continuation-in-part of U.S. patent application serial no. 09/100,914, filed June 19, 1998, from which priority is claimed, which in turn is a continuation-in-part of U.S. patent application serial no. 09/041,149, filed March 12, 1998, from which priority is claimed, which in turn is a continuation-in-part of U.S. patent application serial no. 08/757,134, filed December 3, 1996, now U.S. Patent No. 5,794,254, from which priority is claimed.

FIELD OF THE INVENTION

The present invention relates generally to computer file back up systems, and more particularly to automated computer file back up systems.

BACKGROUND

Extraordinarily large amounts of information are stored on electronic media, such as personal computer hard disk drives and other well-known data storage media. Increasingly, the information is exclusively stored on such media; no "hard" (i.e., paper) copies exist for much information. It can readily be appreciated that such so-called "paperless" information storage systems consume less space and virtually no natural resources, in contrast to systems that require information to be stored on paper. Unfortunately, it is not uncommon for electronic data storage media to "crash", i.e., to become damaged or otherwise lose its information. Such information loss can be devastating to the information owner and highly costly.

Accordingly, many computer system managers routinely make back up copies of computer files. Typically, once each day (or at some other specified periodicity) a system manager will cause the computer system to copy files that are resident on the hard disk drives of the system onto storage media such as other hard disk drives or magnetic tape. Should a file or entire hard disk drive in the system be damaged, lost, or otherwise rendered inaccessible, the back up copy of the file that is stored on, e.g., the storage tape can be copied back into the system.

Such back up systems suffer from several drawbacks. First, they usually require user interaction to initiate the back up process, and consequently occupy the time of a person who must undertake the tedious chore. Further, systems which back up computer files and store the files in the same premises as the system being backed up are of little value in the event of a catastrophe such as a fire or flood. This is because the backed up file copies can be expected to perish along with the files that are resident in the system, thereby rendering the back up system a failure.

An example of an alternate back up system is disclosed in U.S. Pat. No. 5,479,654, which teaches sending changed portions of computer files via modem to an off-site electronic storage facility. As taught in the '654 patent, segments of computer files are checked for changes that have been made since the previous back up. This check first consists of comparing a so-called "exclusive-OR" ("XOR") product of the file against an XOR value that was calculated previously for the stored back up version of the file. If the XORs do not agree, a file change is indicated, and the changed portion of the file is sent via modem, LAN, WAN, or other network device to the off-site facility.

On the other hand, if the XORs do agree, a second, more rigorous check is made to determine whether the segment has been changed since the last back up. The second check consists of comparing a cyclic redundancy check (CRC) product of the segment against a previously recorded CRC value. If the CRCs do not match, a file change is indicated, and the changed portion of the file is sent via modem, LAN, WAN, or other network device to the off-site facility. Otherwise, the segment is assumed to have not been changed since the last back up.

Unfortunately, the system disclosed in the '654 patent can require two calculations per check. Each calculation consumes computing time, and as recognized by the present invention, it is consequently desirable to minimize the number of calculations undertaken to determine whether a change has been made to a computer file. As further recognized herein, it is nevertheless desirable to undertake an initial "quick and dirty" check to reduce the number of computationally rigorous checks that must be made.

Moreover, as recognized by the present invention it is necessary to manage the transmission of data to the off-site storage facility, to avoid one user interfering with the back up operations of another user. Still further, the present invention recognizes that some files need not initially be backed up at all. And, the present invention recognizes that file restoration can be provided for in more than one way to reduce the time required to restore lost files to a computer system.

In addition to the above considerations, the present invention recognizes that data backup undertaken at predetermined intervals might risk failing to backup data that is generated and lost between the intervals. Fortunately, the present invention recognizes that data backup via wide area networks can be undertaken as a user is actively using a computer, thereby providing real time and, hence, almost completely up to date data backup.

Moreover, when data is to be transmitted via wide area networks, the data is preferably first encrypted for security reasons. The present invention recognizes that the longer a user employs a single key for encrypting data, the greater the risk that the key will be broken. For this reason, it is preferably to use more than one key to encrypt data. In the context of data backup over time, the present invention understands that the use of more than one key over a period of time unfortunately is rendered somewhat difficult because it is difficult during subsequent decryption to associate the correct encryption key with the blocks that were originally encrypted with the key. As recognized herein, however, it is possible to facilitate the use of more than one encryption key to thereby reduce the risk of encryption system compromise. The present invention still further recognizes that in addition, it is desirable and possible to provide a session-unique key for each transmission session that cannot be broken unless every bit of data to be transmitted is known a priori.

A further consideration of the present invention is the growing use of laptop or other portable computers, which many people carry with them to and from work and as they travel. Frequently, a user of a laptop computer will seek to connect the laptop computer to a computer network. The network might be a simple home network having, e.g., a single printer, or the network might be a local area network (LAN) at a place of business, or wide area network (WAN) having many devices connected thereto, including printers, server computers, and so on. In any case, the present invention understands that the user of the laptop computer typically must reconfigure the computer each time the laptop is connected to a network, for example, to reconfigure the laptop computer to print to the particular printer or communicate with the particular server in the network to which the laptop computer happens to be connected. The present invention recognizes that requiring a user of a portable computer to reconfigure the computer each time it is connected to a network is annoyingly time consuming, and that such reconfiguration need not require user intervention to accomplish.

With further respect to laptop computers, laptop computers typically are not connected to a network continuously, or indeed at any regular interval, but usually are networked only on an ad hoc basis as the user's schedule permits. The present invention accordingly recognizes that prescheduled backups over the Internet, for laptop computers, is less than an optimum solution. Rather, as recognized herein, the transmission of backed up computer files in a laptop computer must be undertaken in a sufficiently robust manner to account for the episodic connecting of a laptop computer to the Internet. It is an object of the present invention to address one or more of the above issues.

SUMMARY OF THE INVENTION

A computer program product includes a computer program storage device that is readable by a digital processing system such as a portable laptop computer. Logic means are on the program storage device, and the logic means include instructions that can be executed by the digital processing system for performing method steps for backing at least some blocks in at least one local computer file from at least one portable computer that can be connected to a computer network. The method steps that are performed by the computer program product include receiving at least one screen saver signal, and in response undertaking at least one of: preparing the blocks for back up, and transmitting the blocks via the network. In a preferred embodiment, the method steps also include generating a user warning when one or more of the following conditions is present: a predetermined amount of data has been backed up, and the amount of unprotected data exceeds a predetermined threshold.

In another aspect, a computer program product includes logic for determining at least blocks of computer files in a portable computer to be backed up. Also, the program product includes logic to access a file system in the portable computer to access the blocks to be backed up. Moreover, logic means copy blocks to a hold area of the portable computer, in response to the means for determining blocks to be backed up, and logic is provided for determining when the portable computer is connected to a computer network. As set forth in greater detail below, the present logic determines when a data path of the network is idle. When the data path is idle, blocks in the hold area are transmitted to a remote data storage device via the network.

In the preferred logic, a network interface is invoked when it is determined that blocks of computer files in the portable computer are to be backed up. A file change signal can be received to cause blocks to be backed up to the hold area. Also, a data ready signal can be received to cause the blocks to be transmitted from the hold area to the network. The computer program product is disclosed in combination with the portable computer, and in combination with the network.

In another aspect, a method is disclosed for backing up blocks of computer files in a portable computer. The method includes receiving a computer idle signal from the portable computer, and in response to the computer idle signal, backing up blocks in a file system of the computer in accordance with a backup program.

In another embodiment, a computer program product includes a computer program storage device that is readable by a digital processing system and a program means on the program storage device. As intended by the present invention, the program device is realized in a critical machine component that causes the digital processing system to perform method steps to back up one or more blocks in at least one local computer file from at least one portable computer connectable to a computer network. The method steps performed by the computer program product include preparing the blocks for back up, and determining whether the portable computer has been connected to a network. Also, the method steps include, if the portable computer has been connected to a network, permitting transmission of one or more of the blocks. Per the present invention, the blocks are interleaved with other data for transmission thereof.

As described in the above-referenced prior applications, for at least some of the blocks in a local file to be backed up, two respective characters thereof defining respective first comparison values are copied, and respective digital signature codes defining second comparison values are generated. For at least some of the blocks, a test digital signature code is periodically determined compared to the respective second comparison value. When the test digital signature code of a block does not equal the respective second comparison value, it is determined whether the first two characters of the block equal one of the first comparison values. In contrast, when the first two characters equal one of the first comparison values, a block digital signature code is determined using a test $byte_i$ as the first byte of a test block. Then, it is determined whether the block digital signature code equals one of the second comparison values, and when the block digital signature code equals one of the second comparison values, a "resynchronized" signal is returned. Otherwise, when either the first two characters of the test $byte_i$ do not equal one of the first comparison values, or when the block digital signature code does not equal one of the second comparison values, "i" is incremented by unity. The computer program product is disclosed in combination with the portable computer.

If desired, the preferred computer program product can determine a network configuration based on the presence and type of one or more devices connected to the network, and then correlate the network configuration to at least one portable computer setting. The setting is established on the computer in response to the correlating step.

In yet another aspect, a portable computer back up system for backing up one or more blocks in one or more files in the portable computer and transmitting the blocks to a remote computer storage location via a network includes logic means for determining when the portable computer is connected to the network. The system further includes logic means for determining whether an availability of the network is at least equal to a predetermined availability. Moreover, logic means transmit over the network blocks to be backed up when one or more transmission criteria are satisfied. These transmission criteria can include the portable computer being connected to the network.

In still another embodiment, a computer-implemented method for backing up computer file blocks includes, at a user computer, encrypting at least one block using one or more encryption keys. The method further includes storing the one or more encryption keys in a key file, and transmitting the block to a remote data center. Next, the method contemplates, at a restore time, transmitting the block from the data center to the user computer, and selecting one of the keys from the key file. The block is decrypted with the key, and the method then determines whether the decryption was successful. When the decryption is not successful, another key is selected from the key list and the data is decrypted, and the above steps repeated until the decryption is successful.

In another aspect, a method for providing a session key for a transmission session for transmitting data between a transmitting computer and a receiving computer includes generating a digital signature unique to the data, and using the digital signature to encrypt the data.

In another aspect, a method for backing up data in a system including user computers and at least one data storage facility includes, at a user computer, determining whether data associated with the user computer is resident in a common software library associated with the data storage facility. The data is not backed up when the data is resident in the common software library. Otherwise, the data is automatically added to the common software library when the data is present in at least a predetermined number of user computers in the system.

The details of the present invention, both as to its structure and operation, can best be understood in reference to the accompanying drawings, in which like reference numerals refer to like parts, and in which:

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a schematic diagram of the computer file back up system of the present invention;

Figure 2 is a flow chart of the initialization process;

Figure 3 is a flow chart of the subfile incremental back up process;

Figure 4 is a flow chart of the restore process;

Figure 5 is a flow chart of the process for automatically updating the common software library;

Figure 6A is a flow chart of the process for encrypting data to be backed up;

Figure 6B is a flow chart of the process for decrypting backed up data received from the data center;

Figure 7A is a flow chart of the process for transferring data from a user computer to a data center using the message digest (MD)-5 comparison value of the data being transferred as an encryption random number;

Figure 7B is a flow chart of the process for validating the data transfer shown in Figure 7A;

Figure 8 is a flow chart of the process for continuously backing up the computer files of a user computer while a user is using the computer and the computer is online with a remote data center via a wide area network;

Figure 9 is a flow chart of the details of the process for backing up computer files on a portable computer that might from time to time be connected to a computer network;

Figure 10 is a flow chart of the process for transmitting the data generated by the logic in Figure 9;

Figure 11 is a flow chart of the process for automatically reconfiguring a portable computer when it is connected to a computer network;

Figure 12 is a flow chart of another logical process for backing up files on a laptop computer;

Figure 13 continues the logic shown in Figure 12; and

Figure 14 is a flow chart showing the steps of a simplified laptop backup system.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Referring initially to Figure 1, a system is shown, generally designated 10, which includes a server computer, referred to herein as a data center 12, which is part of a computer network. In the preferred embodiment, the data center 12 is part of the computer network 14 referred to as the Internet, and the data center 12 can communicate with other computers on the network 14 via a modem, LAN, WAN, or other network device 16.

As shown in Figure 1, the data center 12 includes an electronic data storage repository 18 with an associated common library 20. Preferably, to permit scaling of the system 10 to support a large number of users, the data repository 18 has a multi-terabyte or indeed a petabyte capacity. Preferably, the capacity of the repository 18 is over one hundred terabytes, and more preferably is five hundred terabytes or more. A control module 22 controls the flow of data into and out of the repository 18 in consonance with the novel logic described below.

Per the present invention, the common library 20 stores the blocks of computer files that are widely distributed and used by many users and, hence, that are common to many users. For example, the common library 20 can store copies of well-known off-the-shelf personal computer operating systems, network browsers, and so on. The common library 20 can be automatically updated as discussed further below in reference to Figure 5. As intended by the present invention, the data storage repository 18 with common library 20 can be physically implemented in a suitable medium, such as magnetic tape, hard disk arrays, compact disk read-only memory (CD-ROM), DASD arrays, or even floppy diskettes.

Plural user computers 24, e.g., personal computers or portable laptop or palm top computers, communicate with the network 14 via respective user modems 26 or other appropriate network connection in accordance with well-known principles. In the preferred embodiment, each user computer 24 includes a respective software-implemented back up module 28 that undertakes the inventive steps of the present invention. It is to be understood that in another embodiment, the back up modules 28 can be remote from, but accessible to, the user computers 24. Additionally, each user computer 24 preferably includes a respective CD-ROM drive 30 for purposes to be made clear shortly. As those skilled in the art will recognize, the user computers 24 can also be associated with respective video monitors, printers, and other output devices, and keyboards, keypads, mice, and other input devices.

Figures 2-8 illustrate the structure of the back up module 30 of the present invention as embodied in computer program software. Those skilled in the art will appreciate that the Figures illustrate the structures of logic elements, such as computer program code elements or electronic logic circuits, that function according to this invention. Manifestly, the invention is practiced in its essential embodiment by a machine component that renders the logic elements in a form that instructs a digital processing apparatus (that is, a computer) to perform a sequence of function steps corresponding to those shown in the Figures.

These instructions may reside on a program storage device including a data storage medium to establish a computer program product, such as a programmed computer diskette. Alternatively, such media can also be found in semiconductor devices, on magnetic tape, on optical disks, on a DASD array, on a conventional hard disk drive, on electronic read-only memory or on electronic random access memory, or other appropriate data storage device. In an illustrative embodiment of the invention, the logic means are computer-executable instructions that are written in C^{++} language code.

Referring now to Figure 2, the initialization process can be understood. The process begins at start state 32 for each user computer 24 and moves to block 34, wherein a file index counter "j" is set equal to unity. Moving to decision diamond 36, it is determined whether a copy of the j^{th} file, using the so-called "message digest five" ("MD5") code discussed further below, is already stored in the common library 20. This can be done on a file-by-file basis by comparing the MD5 of each j^{th} file to the MD5s of the files in the common library 20. For each file that is stored in the common library 20, as indicated by a match of a file's MD5 with one of the MD5s of the files in the library 20, the process moves to block 38 to flag the file as being in the common library and, hence, not a candidate for wholesale transfer to the data center 12.

Alternatively, greater granularity can be achieved by comparing the MD5 of each block of each j^{th} file to a list of common block MD5s, and flagging only those blocks having MD5s that match an MD5 on the common library 20 list of MD5s as being in the common library and, hence, not candidates for wholesale transfer to the data center 12. Thus, it is to be understood that in the event that the j^{th} file as it exists in the user computer 24 has been modified from the common, off-the-shelf version in the common library 20, the modified portions will be detected and transferred to the data center 12 during the subfile incremental procedure discussed below in reference to Figure 3.

In any case, it can now be appreciated that the use of the common library 20 avoids the necessity of transmitting to the data center 12 very large, off-the-shelf computer files that are resident on many if not most personal computer systems and that would otherwise require backing up. In other words, the common library 20 reduces the time and cost of making initial copies of the files in the user computers 24.

Additionally, at block 38 a block-by-block comparison value listing is recorded that has two entries for every block_i (wherein "i" is a block index counter) of every file_j flagged as being in the common library 20. The first entry is the first two characters ("XX_i") of the first byte of each block_i. In contrast, the second entry is a digital signature ("MD5$_i$"), the value of which is uniquely defined by the contents of the block$_i$. The first entry establishes a first comparison value, and the second entry establishes a second comparison value.

As mentioned above, in the presently preferred embodiment, the digital signature used is the 128 bit-long "message digest five" ("MD5") code known in the cryptology art. As recognized by the present invention, the MD5 code of a block changes a great deal with even very small changes of the block. Thus, the digital signature of the present invention advantageously is more sensitive to changes in a block than are check sums, cyclic redundancy codes, and so-called "exclusive or" codes.

From block 38, the logic proceeds to decision diamond 40 to determine whether the last file in the user computer 24 has been tested. If so, the initialization process terminates at state 42. Otherwise, the process moves to block 44 to increment by one the file index counter "j", and then to loop back to decision diamond 36.

In contrast, if it is determined at decision diamond 36 that a copy of the ith block of the jth file is not in the common library 20, the logic proceeds to block 46 to trickle copy each non-resident block$_i$ of the file$_j$ to the storage repository 18 of the data center 12. By "trickle copy" is meant that the non-resident blocks of the file are transmitted to the data center 12 during a predetermined transmission period, also referred to herein as a "window", until the entire non-resident portions file has been transmitted, or until the transmission period elapses. Alternatively, as discussed below in reference to Figure 8, the data to be backed up can be transmitted real time as the user inputs data to the user computer 24, provided the user computer 24 is online with the network 14.

If the transmission period elapses before the non-resident portion of the file has been transmitted in its entirety, the respective back up module 28 completes the transmission of the chunk, and then terminates. If the transmission is interrupted during the transmission period, the module 28 notes the value of the block index counter "i", such that transmission can be resumed from there during the next transmission period. Then, at the start of the next transmission period, the back up module 28 increments the block index counter "i" by unity and commences transmitting the file$_j$ where it previously had left off. As indicated in Figure 2, the block-by-block listing of XX_i and $MD5_i$ is generated at block 46 as described previously in reference to block 38. From block 46, the logic proceeds to decision diamond 40 to function as previously described.

Turning now to Figure 3, the subfile incremental back up logic can be seen. It is to be understood that as intended by the present invention, periodically (e.g., on a daily basis) the back up module 28 of each user computer 24 will undertake the process discussed below.

Commencing at decision diamond 48, it is determined whether a user-selected predetermined back up period has commenced. When it has, the process moves to block 50 to enter a do loop for each file$_j$ that has been designated by the user to be backed up. Then, the process moves to block 52 to enter a do loop for each block$_i$ of the file$_j$ that was selected at block 50.

Moving to block 54, the back up module 28 computes the digital signature $MD5_i^{new}$ of the $block_i$ under test. Next, the digital signature $MD5_i^{new}$ of the $block_i$ under test is compared, at decision diamond 56, with the digital signature $MD5_i^{old}$ that is recorded in the listing that was generated in Figure 2, as described above.

It is to be understood that the listing of digital signatures $MD5^{old}$ is an ordered list from $i = 0$ to n, wherein n = number of digital signatures in the list. It is to be further understood that the digital signature $MD5_i^{new}$ of the $block_i$ under test is first compared to the i^{th} $MD5^{old}$ in the list, and if it doesn't match, the digital signature $MD5_i^{new}$ of the $block_i$ under test is next compared to the next $MD5^{old}$ in the ordered list (i.e., the $i^{th} + 1$ $MD5^{old}$), and so on, until a match is found, or until the end of the ordered list has been reached. When a match is found anywhere in the list, the test at decision diamond 56 is positive, and the process moves to decision diamond 58. Also, when a match is found at the l^{th} position in the list, wherein $l > i$, the comparison of the $MD5_{i+1}^{new}$ of the $i^{th} + 1$ block to be tested is first compared to $MD5_{l+1}^{old}$, i.e., to the $l^{th}+1$ element in the ordered list of old digital signatures. If no match is found, the $MD5_{i+1}^{new}$ of the $i^{th} + 1$ block is then serially tested against the digital signatures in the list after the $l^{th}+1$ element in the ordered list of old digital signatures.

Returning to the discussion of the i^{th} block, when a digital signature match is found, indicating that no change was made to the $block_i$ under test since the last back up, the process moves to decision diamond 58. At decision diamond 58, the logic determines whether the last block of the $file_j$ has been tested, and if not, the process loops back to retrieve the next $block_i$ and compute its digital signature as shown and described. In other words, if further blocks exist for the file$_j$ under test, the index counter "i" is incremented by one, and the process loops back to block 54.

If, on the other hand, the last block of the file$_j$ has been tested, the logic determines, at decision diamond 60, whether the last file in the user computer 24 has been tested. If not, the process loops back to retrieve the next file and proceed as before to test the blocks in the file. Otherwise, the process moves to decision diamond 62.

At decision diamond 62, it is determined whether the back up period has not elapsed, i.e., whether the allocated back up window is still open. If the period has elapsed, the logic loops back to decision diamond 48 to await the commencement of the next allocated back up period. In contrast, if the back up window is still open, the logic attempts to establish communication with the data center 12 via the modems 26, 16 (or other network devices, such as LANs, WANs, etc.).

At decision diamond 64, it is determined whether the communication connection was successful. If it wasn't, the logic moves to block 66 to wait a predetermined time-out period (preferably five minutes) before looping back to decision diamond 62 to retry the connection if the window is still open. In any case, when it is determined that communication has been established between the user computer 24 and the data center 12 subsequent to testing all files$_j$ in the user computer 12, the logic encrypts and compresses any remaining updated blocks (described further below) and transmits them to the data center 12, at block 68. From block 68, the process loops back to decision diamond 48.

Returning to the negative loop originating at decision diamond 56, when the digital signature of the block$_i$ does not match one of the signatures stored in the listing for the block, a change to the block$_i$ is indicated, and the block$_i$ therefore becomes a candidate for back up. Additionally, the process resynchronizes itself with the block-by-block comparison value listing discussed above.

Accordingly, after a negative test at decision diamond 56 (i.e., when the logic essentially returns a "changed block" signal), the logic moves to block 70, wherein a do loop is entered for the bytes$_k$ of the block$_i$. "k" is an index counter initially equal to unity. Moving to decision diamond 72, the logic determines whether the first two characters ("XX_k^{new}") of the block starting with the byte$_k$ under test equal the first comparison value ("XX^{old}") that corresponds to the block$_i$ in the comparison value listing. It is to be understood that the comparison value listing parallels the ordered list of digital signatures $MD5^{old}$. Consequently, if the first comparison does not yield a match, similar to the testing at decision diamond 56 the two characters XX_k^{new} are compared to the next two characters XX^{old} in the ordered list until a match is found or until the end of the list is reached.

When the end of the list of comparison values is reached without a match, the logic moves to decision diamond 73 to determine whether the current byte$_k$ is the last byte of the candidate block under test. Stated differently, at decision diamond 73 the logic determines whether the byte index counter "k" equals about two thousand forty eight (2048). If it does, the logic moves to block 74 to increment the block index counter "i" by unity, and then the logic loops proceeds to decision diamond 58. In this instance, the entire block under test is flagged for copying. If it is determined that the byte index counter "k" does not equal one thousand at decision diamond 73, the logic loops back to increment "k" by unity and test the next byte in the block_i at decision diamond 72.

If, however, it is determined at decision diamond 72 that "XX_k^{new}" equals an "XX^{old}", the logic proceeds to decision diamond 75. At decision diamond 75, the digital signature MD5 of the block having as its first byte the byte_k under test is determined and compared to the second comparison value, i.e., the digital signature $MD5^{old}$ that corresponds to the matched characters XX^{old} in the comparison value listing. As was the case with the test at decision diamond 56, the digital signature MD5 being tested is tested against subsequent digital signatures $MD5^{old}$ in the ordered list, if necessary, to find a match. If no match is found, the logic moves to decision diamond 73.

If, however, the digital signature MD5 of the block having as its first byte the byte_k under test is determined to be equal to one of the digital signatures $MD5^{old}$ in the ordered list at decision diamond 75, the logic returns "resynchronized" and moves to block 76. In other words, a positive test at decision diamond 75 indicates that the logic has found an old, unchanged block that previously has been backed up, and, hence, that the logic is resynchronized with the comparison value listing.

At block 76, the changed block(s) (also referred to herein as "transmission blocks") are moved to a "next chunk" file. Additionally, at block 76 the comparison value listing is updated to include the first two characters and digital signatures of the changed block(s), for use as the first and second comparison values, respectively, during the test of the blocks during the next back up cycle. Moving to decision diamond 78, it is determined whether the chunk file is full. In the presently preferred embodiment, the chunk file is full when its size is five megabytes (5 MB).

If the chunk file is not full, the logic returns to decision diamond 58. In contrast, if the chunk file is full, the process moves to decision diamond 80 to determine whether the back up period has not elapsed, i.e., whether the allocated back up window is still open. If the period has elapsed, the logic loops back to decision diamond 48 to await the commencement of the next allocated back up period. In contrast, if the back up window is still open, the logic transmits the chunk when a successful connection has been established with the data center 12, using the procedure of steps 64 and 66 discussed above. From block 82, the logic returns to decision diamond 58.

Figure 4 shows the logic by which lost files may be restored to a user computer 24. It is to be understood that as envisioned herein, backed up file blocks at the data center 12 are periodically (e.g., weekly, monthly, quarterly, etc.) copied from the storage repository 18 to CD-ROM disks, and the disks then transported to the same physical location as the associated user computer 24. Hereinafter, these CD-ROM disks are referred to as "local back up disks".

At block 84, a directory of blocks in the data center 12 that correspond to the user computer 24 is downloaded from the data center 12. Next, at block 86 a do loop is entered for each block$_i$ of the lost file(s) as follows. It is determined at decision diamond 88 whether the requested version of the block of the file (ordinarily the latest version) is on the local back up disks. If so, the block is preferentially restored from the local back up disks at block 90. Otherwise, it is restored via the modems 26, 16 from the data center 12 at block 92. From block 90 or block 92, the logic loops back to retrieve the next block$_i$, and continues this process until the entire requested files have been restored.

Further inventive features of the present invention can be appreciated in reference to Figures 5-8. With particular regard to Figure 5, the common software library 20 shown in Figure 1 may be automatically updated by commencing at block 100 in Figure 5. At block 100, the server (i.e., at the data center 12 shown in Figure 1) undertakes, for each digital signature MD5 code with its attendant block received for storage, a loop by moving to decision diamond 102, wherein it is determined whether the digital signature MD5 code under test has been recorded as a candidate for the library 20. The first time the particular digital signature MD5 code is received from any user computer, the test at decision diamond 102 is negative, and the process consequently moves to block 104 to record the digital signature MD5 code on a candidate list. Also, a counter J_{MD5} for the particular digital signature MD5 code under test is set equal to unity at block 104. Moving from block 104 to block 106, the next received digital signature MD5 code is retrieved and the above process repeated as indicated by the dashed loop back line in Figure 5.

On the other hand, when the digital signature MD5 code previously has been received from a user computer, the test at decision diamond 102 is positive, and the process moves to block 108 to increment the counter J_{MD5} that is associated with the digital signature MD5 code under test by one. It is to be understood that the counter J_{MD5} that is associated with the digital signature MD5 code under test can be incremented whenever the digital signature MD5 code under test is received from any user computer, or only when it received from a user computer that has not previously sent the digital signature MD5 code under test.

Then, at decision diamond 110 it is determined whether the value of the counter J_{MD5} that is associated with the digital signature MD5 code under test equals a predetermined value "k". If not, the process loops back to block 106, but otherwise the process moves to block 112 to add the digital signature MD5 code under test, along with its associated block(s), to the common software library 20. The process then loops back to block 106 to retrieve the next digital signature MD5 code.

Figures 6A and 6B show the present inventive steps for allowing a user to use multiple encryption keys to transmit data for back up, thereby increasing security. Commencing at block 114, at a user computer, the digital signature MD5 code for each block for which backup has been selected by the steps disclosed above in Figures 2-4 is determined and attached to its respective data block, and then the digital signature MD5 code with data block is compressed using compression principles known in the art. Moving to block 116, the data block, but not the MD5 code, is encrypted with a user-selected key on a user key list. From block 116, the process ends at state 120, it being understood that further transmission steps as previously disclosed can be undertaken. Subsequently, when the user computer, for example, loses the data packet processed by the steps shown in Figure 6A and accordingly requests and receives a copy of the packet from the data center 12, the user computer invokes the steps shown in Figure 6B. Commencing at block 124, the user computer receives the blocks to be restored from the data center and then undertakes the following steps for each block. At block 126, the computer selects the first encryption key in the list, and then at block 128 the computer decrypts and decompresses the block. Moving to decision diamond 130, the user computer determines whether decompression failed as indicated by, e.g., data overflow. If decompression failed, the process moves to block 132 to select the next key in the list, and then loops back to block 128 to again attempt decompression.

If, on the other hand, decompression is determined at decision diamond 130 to have been executed satisfactorily, the process moves to block 136 to further validate the key under test by stripping the digital signature MD5 code from the packet and then calculating anew the digital signature MD5 code of the data that is associated with the decrypted block. In undertaking the process at block 136, the present invention understands that the digital signature MD5 code is a 128 bit-long string that is appended to the data with which it is associated in a known location, and that consequently the digital signature MD5 code easily can be stripped.

At decision diamond 138 the calculated digital signature MD5 code of the decrypted data is compared to the decrypted digital signature MD5 code that had been stripped from the data at block 136. If the two digital signature MD5 codes match each other at decision diamond 138, the key under test has been validated and the process returns "correct key" at block 140. Then the process retrieves the next data portion at block 124. Otherwise, the process loops back to block 132 to retrieve the next key in the user computer's key list.

Figure 7A shows a method for providing a unique, session-specific session key for encryption purposes. Commencing at block 142, the digital signature MD5 code for the entire set of data to be transmitted is calculated by the computer that is to transmit the data. Moving to block 144, the digital signature MD5 code is used as the encryption key for the data transmission session. In the event that only 56 bits are required for the key, the two 64 bit halves of the 128 bit-long digital signature MD5 code are combined using an exclusive-OR (XOR) operation and then 8 bits, e.g., the last 8 bits, of the resulting 64 bit string are discarded.

From block 144, the present logic follows two branches, denoted in the Figures as "A" and "B". More specifically, at block 146A the data is encrypted using the digital signature MD5 code (or portion thereof) as a random number, and then at block 148A the data is transmitted to the receiving computer. On the other hand, at block 146B the portion of the digital signature MD5 code that is to serve as the session key (i.e., the portion of the digital signature MD5 code that is used at block 146A to encrypt the data) is transferred from the transmitting computer to the receiving computer using conventional private key/public key encryption principles, or more preferably using Diffie-Helman encryption principles. That is, at block 146B the digital signature code can be encrypted by the transmitting computer using the receiving computer's public key, and then at block 148B the receiving computer can decrypt the digital signature code using its private key. Then, as shown in Figure 7A, block 150 receives the results of blocks 148A and 148B to decrypt the data using the digital signature code session key.

Figure 7B additionally shows that the present logic can validate the data received at block 152 by undertaking the validation steps shown at blocks 152-158. More particularly, at block 152 the receiving computer calculates the digital signature code of the data that is associated with the decrypted data from block 150. Next, at decision diamond 154 the calculated digital signature code of the decrypted data is compared to the digital signature code that had been received and decrypted at block 148B. It is to be understood that the in the event that only 56 bits of an XOR'd digital signature MD5 code are used as the session key, the comparison at decision diamond 154 can be between the session key and an XOR'd version of the digital signature MD5 code calculated at block 152.

In any case, if the two digital signature codes match each other at decision diamond 154, the session key under test has been validated and the process returns "transmission validated" at block 156. Otherwise, the process returns "transmission error" at block 158.

Now referring to Figure 8, a method is shown for continuous data backup via a wide area computer network while a user computer is online with the network and while a user is using the user computer. Commencing at block 160, upon determination by a user computer that a changed data block is to be backed up in accordance with the principles discussed above (with the exception that a backup start time is not waited for but rather that the process perpetually determines, real-time, data blocks to be backed up), the process moves to decision diamond 162. At decision diamond 162, the process determines whether the user computer is online with the network, e.g., the network 14 shown in Figure 1. If not, the process moves to block 164 to store the data for transmission thereof when the computer is online.

On the other hand, if, at decision diamond 162, the process determines that the user computer is online with the network, the present logic moves to decision diamond 166 to determine whether the central processing unit (CPU) of the user computer is busy. Specifically, the determination at decision diamond 166 is whether the CPU is busy with a task that has a higher priority assigned to it than the priority of the backup task shown in Figure 8. If the CPU is not occupied with another task of higher priority than the present data backup task, the process moves to block 168 to transmit the block until the entire block has been transmitted, or until the computer becomes reoccupied with a task of higher priority than backup. It is to be understood that when transmission of a block is interrupted by a higher priority task, the user computer undertakes the higher priority task, and when finished returns to the present data backup task to complete transmission of the data block. The logic then moves to block 169 to determine further blocks for transmission, and to transmit the blocks virtually real-time as the user uses the computer for other tasks.

The loop between block 170 and decision diamond 166, and the "busy" loop at block 168, illustrate the principles of the above discussion. When the CPU is busy at decision diamond 166 with, e.g., the requirement to execute an input event such as a key stroke, the logic can proceed to block 170 to wait a predetermined period Δt, and then retry the determination at decision diamond 166. It will be appreciated that the length of Δt is very short, on the order of milliseconds or microseconds. Alternatively, the logic can essentially define "Δt" to be "whenever the CPU is idle", and when the CPU is idle, proceed to block 168 to transmit data.

For example, during the test at decision diamond 166, the CPU might receive a data input via a key stroke by the user of the computer, and if so, the process idles at block 170 until the key stroke is completed, before transmitting data at block 168. Thus, in one embodiment the present process transmits backup data between CPU tasks that support a user's needs, e.g., between key strokes, with the data that is changed by the key strokes being determined as backup data and transmitted offsite via the Internet for storage virtually real time as the data is being created by the user of the computer.

Figures 9 and 10 illustrate the principles discussed above in a preferred configuration when, e.g., a user computer 24 is a laptop or palm top computer that hosts the present logic. As disclosed below, in a portable computer environment, the present invention copies file blocks when the host portable computer is not connected to a network, and when the present invention senses a network connection, the copied blocks are transmitted via the network to a storage facility.

Commencing at block 180, when the host portable computer is energized, the logic determines whether all files on the user's computer are to be considered for back up, or whether only a user-defined set of files is to be considered. If the user has defined a set of files (by, e.g., directory) for back up, using, for example, file inclusion/exclusion lists with wildcards, this set is received at block 182. From block 182, or from decision diamond 180 if no user-defined set is received and all files (or a default set of files) are to be candidates for back up, the logic moves to block 184 to prepare the blocks in, e.g., chunks as described above, for back up preferably in accordance with the logic set forth in Figure 3 above. As mentioned above, the back up process is undertaken in the "background", transparently to the user as the user employs the host computer for other tasks such as, e.g., word processing or presentation slide generation.

Decision diamond 186 represents that the logic of the present invention monitors for whether a predetermined storage space limit has been reached on the host portable computer. As recognized by the present invention, a portable computer such as a laptop computer that is energized but disconnected from a network might encounter storage space limitations due to the generation of duplicate blocks for back up, and the step at decision diamond 186 is to ensure that the back up process set forth herein, which is intended to be a "background" process, does not fill up the user's storage. In the preferred network embodiment, no more than 5% of the host computer's storage capacity is used for back up storage. Alternatively, the user can define the predetermined storage space limit. If a storage space limit has been reached, the logic moves to block 188 to suspend the back up process until, e.g., subsequent transmission of the blocks to be backed up followed by deletion of the back up copies on the host computer frees additional storage space.

From block 188 or decision diamond 186 when the test there is negative the logic moves to decision diamond 190. As represented by decision diamond 190, the present logic monitors whether the host portable computer has been connected to a network. When no network connection is sensed, the logic moves to block 192 to continue back up processing as described above (storage space permitting), and then logically loops back to decision diamond 186.

In contrast, when the logic senses a network connection, it determines whether the host portable computer is actively transmitting data, as represented at decision diamond 194. If so, the present logic suspends back up processing at block 196 so as not to interfere with the transmission, and then logically loops back to decision diamond 194 to await the end of the transmission. When the host portable computer is not transmitting, the logic returns to block 192.

It is to be understood that while the logic flow shown in Figure 9 represents the principles of the present invention for disclosure purposes, the actual logic code might be event-driven, i.e., it might continuously monitor for the conditions represented at decision diamonds 186, 190, and 194. Likewise, the logic shown in flow format in Figures 10 and 11 can be implemented as event-driven code.

Now referring to Figure 10, the logic for transmitting the blocks to be backed up that are generated by the logic shown in Figure 9 is shown. Commencing at block 198, the transmission logic commences when a network connection is sensed. In the preferred embodiment, priority is given to transmitting back up blocks when the blocks are available and the network bandwidth permits, with the back up processing shown in Figure 9 being undertaken while the host computer is connected to a network either when no blocks are ready for transmission or when network usage precludes transmission.

Moving to decision diamond 200, the logic determines whether a block of data is ready for transmission, and if so, the logic moves to block 202 to interleave the block with, e.g., the associated meta data or with other data to be transmitted. The purpose of this is to allow for the use of all forward progress made during a back up session up to the last file processed in its entirety, thus taking advantage of work completed up to the point that a transmission might be aborted. In this regard, the process can be considered an "inchworm" data transmission, in that the transmission stream can be severed (by, e.g., an unintended abort) at an arbitrary point and still maintain a viable entity.

From block 200 the process moves to block 202 to define the "last block" to have the identification of the last actually interleaved block. Then, the process moves to decision diamond 206.

Decision diamond 206 represents the logic determining whether adequate network availability exists for back up data transmission. In one embodiment, the network availability must be less than a predetermined setpoint, e.g., 70% of full availability, for the logic to transmit the back up data, and if it is, the logic moves to block 208 to transmit the data. Otherwise, the logic moves to block 210 to wait until the network availability is below the predetermined setpoint, restarting the process with the resynchronizing logic shown above in Figure 3.

In one preferred embodiment, when the network is a local area network (LAN), the decision at step 206 is undertaken by using operating system performance counters. For example, when the operating system is a Microsoft operating system, the number of dynamic keys in the performance registry can be used to determine the number of transmission packets sent per unit time, and this ratio indicates how busy the network is (and, hence, network availability). Or, when the network is a wide area network (such as the Internet), the round trip time from when the host computer transmits a packet and when it receives back an acknowledgement from the recipient that the packet has been received is used as an indication of network availability, with a lower time indicating higher availability. When the host computer is connected to a LAN that in turn is connected to a WAN, either one or both of the network availabilities can be used in the determination at decision diamond 206.

Figure 11 shows that apart from the back up logic discussed above, the present invention advantageously provides for automatically configuring a portable computer for a network to which the portable computer is connected. More specifically, the present invention automatically reconfigures the host portable computer for the particular printer(s), server(s), internet connection(s), and storage resource(s) of the network to which the host computer happens to be connected, so that the user does not have to undertake such time-consuming reconfigurations manually, each time the host portable computer is connected to a network.

Commencing at block 202, the invention compiles (automatically or based on a one-time user manual input) a list of network identifications, along with a list of devices in each network. For example, a first network might be established by the user's workplace LAN that includes a workplace printer of a particular designation, a workplace server of a particular designation, a workplace internet connection of a particular designation, and a workplace storage resource of a particular designation. Also, a second network might be established by the user's home LAN that includes a home printer of a particular designation, a home server of a particular designation, a home internet connection of a particular designation, and a home storage resource of a particular designation.

As intended by the present invention, a "network configuration" refers to what devices of what type are on the network, i.e., what printers, etc. are connected to the network. For each network configuration, a set of host computer settings is stored or implied. In other words, for each network configuration, a set of host computer settings is correlated to enable the host computer to, e.g., print to the network printer, etc. At block 212, the network configurations are stored.

Moving to block 214, the invention senses when the host portable computer is connected to a network. At block 216, the invention detects what devices are connected to the network. Proceeding to decision diamond 218, the logic compares the devices actually connected to the network to the stored network configurations. When a match is found within, e.g., 90% (that is, 9 of 10 devices on the network are found in a single entry in the network configuration table), the logic moves to block 220 to reconfigure the host portable computer in accordance with the network settings that are correlated to the matched network configuration. If the sensed network configuration does not match any of the stored network configurations, the process ends from decision diamond 218 by establishing a set of default settings in the host portable computer.

Now considering Figures 12 and 13, another embodiment of the logic for backing up blocks of computer files on the computer 24 (Figure 1) when the computer 24 is a laptop computer is shown, generally designated as system 300. The system 300 includes a sequence of logical instructions that can be executed by the computer 24, with the sequence beginning at block 302, wherein the operating system of the computer is started.

Moving to decision diamond 304, it is determined whether the computer is properly configured for back up, i.e., whether the present logic as embodied in software has been properly installed in the computer 24. If it is, the process continues to decision diamond 306 to determine whether any files have been selected, i.e., designated, as backup candidates, either by the user or automatically. By "backup candidates" is meant that some or all of the files in the portable computer can be selected for consideration by the above-described backup algorithm, while making it possible to omit some (less important) files from consideration for backup and thereby speed processing.

If no files have been selected, the process moves from decision diamond 306 to block 308 to run a user interface which permits the user to select some or all of the files in the computer to be candidates for backup. Or, some or all of the files in the computer can be automatically selected at block 308 to be candidates for backup. For example, it could be automatically determined at block 308 to consider all files in the computer to be candidates for backup, with the exception of temporary Internet files.

When it is determined at decision diamond 306 that at least some files have been selected as candidates for backup, the process moves to block 310 to start a network interface, as set forth more fully below in reference to Figure 13. Moving then to block 312, once the network interface is started, the file system of the computer is "hooked". By "hooked" is meant that the present logic is inserted into the native file system code of the computer 24. In this way, the present logic is activated whenever any user accesses the file system, such that modifications to blocks in the file system can be detected for use as set forth above in reference to Figures 2 and 3.

At decision diamond 314, it is determined whether the file system was successfully hooked by the system 300, i.e., whether the file system hooks are correctly and safely in place. If so, the logic proceeds to state 316, wherein the system 300 enters a wait state for one of two computer system signals. First, a file change signal can be received at block 318, in which case the logic flows to block 320 to process new file data in accordance with the above-disclosed backup algorithm (e.g., the algorithm shown in Figures 2 and 3). Blocks that have been so processed as backup blocks in accordance with principles above are copied into an internal data storage area, referred to herein as a "hold" area, within the portable computer. From block 320, the logic loops back to wait state 316.

If, on the other hand, a computer shutdown signal is received at block 322, or it is determined at decision diamond 314 that the file system "hook" was not successful, or it is determined at decision diamond 304 that the computer is not properly configured, the logic moves to block 324. At block 324, file system hooks are closed. Moving to block 326, an exit signal is sent to the network interface logic shown in Figure 13, and then at block 328 an "exit" message is displayed on the monitor of the computer. The process then exits to the operating system of the computer at block 330.

Now referring to Figure 13, the network interface logic can be seen. Commencing at start state 332, the logic moves to decision diamond 334 to determine whether, at the computer 24, network software is present. If it is, the logic moves to a first wait state 336 to await one of two system signals. If an exit signal is received at block 338 from the logic shown in Figure 12, the process moves to block

340 to end any existing network and to return to the logic of Figure 12. In contrast, if a signal indicating that network communication has started is received at block 342, the logic proceeds to a second wait state 344.

The logic awaits one of three system signals in the second wait state 334. An exit signal can be received at block 346, in which case the logic moves to block 340 to function as described above. Or, a signal indicating that the network has stopped can be received at block 348, in which case the logic loops back to the first wait state 336. Still further, a signal indicating that the network is idle can be received at block 350, in which case the logic proceeds to a third wait state 352. By "network idle" is meant that a data path in the network is idle, preferably the data path from the computer 24 to the network or less preferably some other a data path within the network, as measured in accordance with principles discussed above with respect to Figure 10.

The logic awaits one of four system signals in the third wait state 352. An exit signal can be received at block 354, in which case the logic moves to block 340 to function as described above. Or, a signal indicating that the network has stopped can be received at block 356, in which case the logic loops back to the first wait state 336. Still further, a signal indicating that the network is active can be received at block 358, in which case the logic loops back to the second wait state 344, to await a network idle signal.

As shown in Figure 13, a signal indicating that data is ready in the hold area can be received at block 360. When such a signal is received prior to the other three signals that can be received while in the third wait state 352 (i.e., while the "network idle" signal remains present or in force), the data is transmitted to an off-site data storage facility via the network. The logic remains at block 360 as long as the data ready signal is present and none of the other three signals of the third wait state 352 are received.

Now referring to Figure 14, a "Spartan" version of the logic for invoking the backup of laptop computer files can be seen. Commencing at block 362, an idle signal is received from, e.g., the operating system of the portable computer whose files are to be backed up. For example, the commencement of a screen saver program can establish the idle signal received at block 362.

When, in the above example, the screen saver comes on, the logic would accordingly move to block 364, to copy file blocks for backup purposes in accordance with the algorithm shown in Figures 2 and 3. Additionally, two tests can be undertaken by the logic shown in Figure 14. First, at decision diamond 366, it can be determined whether the amount of unprotected data in the laptop computer is greater than a predetermined amount of data. By "unprotected" is meant blocks of data that have been selected as candidates for backup either by the user or automatically, and which have changed since the blocks were last backed up (or which have not yet been backed up at all). Also, at decision diamond 368 it is determined whether the amount of data in the hold area (i.e., the amount of data backed up but not yet transmitted) exceeds a predetermined amount. The predetermined amount can be based on, e.g., a network capacity. When either of the tests at decision diamonds 366, 368 are positive, the logic can present a warning on the monitor of the portable computer at block 370, suggesting, as an example, that an immediate network connection be established to transmit the blocks in the hold area.

While the particular SYSTEM AND METHOD FOR BACKING UP COMPUTER FILES OVER A WIDE AREA COMPUTER NETWORK as herein shown and described in detail is fully capable of attaining the above-described objects of the invention, it is to be understood that it is the presently preferred embodiment of the present invention and is thus representative of the subject matter which is broadly contemplated by the present invention, that the scope of the present invention fully encompasses other embodiments which may become obvious to those skilled in the art, and that the scope of the present invention is accordingly to be limited by nothing other than the appended claims, in which reference to an element in the singular is not intended to mean "one and only one" unless explicitly so stated, but rather "one or more".

WHAT IS CLAIMED IS:

1. A computer program product comprising:

a computer program storage device readable by a digital processing system; and

logic means on the program storage device and including instructions executable by the digital processing system for performing method steps for backing at least some blocks in at least one local computer file from at least one portable computer connectable to a computer network, the method steps performed by the computer program product comprising:

receiving at least one screen saver signal; and

in response to the screen saver signal, undertaking at least one of: preparing the blocks for back up, and transmitting the blocks via the network.

2. The computer program product of Claim 1, wherein the method steps further comprise generating a user warning when one or more of the following conditions is present: a predetermined amount of data has been backed up, and the amount of unprotected data exceeds a predetermined threshold.

3. The computer program product of Claim 1, wherein the method steps further comprise:

for at least some of the blocks in a local file to be backed up, copying two respective characters thereof defining respective first comparison values and generating respective digital signature codes defining second comparison values;

for at least some of the blocks, periodically determining a test digital signature code and comparing it to the respective second comparison value; and

when the test digital signature code of a block does not equal the respective second comparison value, determining whether the first two characters of the block equal one of the first comparison values.

4. The computer program product of Claim 3, wherein the method steps further comprise:

when the first two characters equal one of the first comparison values, determining a block digital signature code using a test byte$_i$ as the first byte of a test block;

determining whether the block digital signature code equals one of the second comparison values;

when the block digital signature code equals one of the second comparison values, returning "resynchronized"; otherwise

when either the first two characters of the test byte$_i$ do not equal one of the first comparison values, or when the block digital signature code does not equal one of the second comparison values, setting $i = i+1$ and repeating select steps.

5. The computer program product of Claim 1, in combination with the portable computer.

6. A computer program product, comprising:

logic means for accessing a file system in the portable computer to access blocks to be backed up;

logic means for copying blocks to a hold area of the portable computer;

logic means for determining when the portable computer is connected to a computer network;

logic means for determining when a data path of the network is idle; and

logic means for transmitting blocks in the hold area to a remote data storage device via the network when the path is idle.

7. The computer program product of Claim 6, further comprising:

logic means for determining at least blocks of computer files in a portable computer to be backed up; and

logic means for invoking a network interface in response to the logic means for determining blocks of computer files to be backed up.

8. The computer program product of Claim 6, further comprising logic means for determining when the logic means for accessing has successfully accessed the file system.

9. The computer program product of Claim 6, further comprising logic means for receiving a file change signal to cause the logic means for backing up the blocks to back up blocks to the hold area.

10. The computer program product of Claim 6, further comprising logic means for receiving a network communication started signal.

11. The computer program product of Claim 6, further comprising logic means for receiving a data ready signal to cause the logic means for transmitting blocks to transmit blocks from the hold area to the network.

12. The computer program product of Claim 6, in combination with the portable computer.

13. The computer program product of Claim 6, in combination with the network.

14. A method for backing up blocks of computer files in a portable computer, comprising:

receiving a computer idle signal from the portable computer;

in response to the computer idle signal, backing up blocks in a file system of the computer in accordance with a backup program.

15. The method of Claim 14, wherein the computer idle signal is established by a screen saver start signal.

16. The method of Claim 15, further comprising generating a user warning when one or more of the following conditions is present: a predetermined amount of data has been backed up, and the amount of unprotected data exceeds a predetermined threshold.

17. The method of Claim 14, wherein the backup program undertakes method steps comprising:

for at least some of the blocks in a local file to be backed up, copying two respective characters thereof defining respective first comparison values and generating respective digital signature codes defining second comparison values;

for at least some of the blocks, periodically determining a test digital signature code and comparing it to the respective second comparison value; and when the test digital signature code of a block does not equal the respective second comparison value, determining whether the first two characters of the block equal one of the first comparison values.

18. The method of Claim 17, wherein the backup program further undertakes method steps comprising:

when the first two characters equal one of the first comparison values, determining a block digital signature code using a test $byte_i$ as the first byte of a test block;

determining whether the block digital signature code equals one of the second comparison values;

when the block digital signature code equals one of the second comparison values, returning "resynchronized"; otherwise

when either the first two characters of the test $byte_i$ do not equal one of the first comparison values, or when the block digital signature code does not equal one of the second comparison values, setting $i = i+1$ and repeating select steps.

19. A method for backing up blocks of files in a laptop computer, comprising:

determining blocks to be backed up;

determining when the laptop computer is connected to a computer network;

determining when a data path of the network is idle; and

transmitting the blocks determined to be backed up to a remote data storage device via the network when the path is idle.

20. The method of Claim 19, further comprising invoking a network interface.

21. The method of Claim 19, further comprising receiving a file change signal to cause blocks to be copied to a hold area in the laptop computer.

22. The computer program product of Claim 21, further comprising receiving a data ready signal to cause blocks in the hold area to be transmitted from the hold area to the network.

23. A system for configuring a portable computer for a network to which the portable computer is connected, comprising:

logic means for determining at least one other device on the network to determine a network configuration;

logic means for comparing the network configuration to one or more predetermined configurations; and

logic means for configuring the portable computer for the network when the network configuration matches at least one of the predetermined configurations.

24. The system of Claim 23, wherein sets of predetermined portable computer settings are correlated with respective predetermined configurations, and the means for configuring establishes at least one of the sets in the portable computer when the network configuration matches the respective predetermined configuration.

25. A portable computer back up system for backing up one or more blocks in one or more files in the portable computer and transmitting the blocks to a remote computer storage location via a network, comprising:

logic means for determining when the portable computer is connected to the network;

logic means for determining whether an availability of the network is at least equal to a predetermined availability;

logic means for transmitting over the network blocks to be backed up when one or more transmission criteria are satisfied, the transmission criteria including:

the means for determining determines that the portable computer is connected to the network.

26. The system of Claim 25, wherein the transmission criteria further include:

the means for determining determines that the availability of the network is less than the predetermined availability.

27. The system of Claim 26, wherein the blocks are transmitted by interleaving the blocks with meta data.

28. A computer program product comprising:

a computer program storage device readable by a digital processing system; and

logic means on the program storage device and including instructions executable by the digital processing system for performing method steps for backing up one or more blocks in at least one local computer file from at least one portable computer connectable to a computer network, the method steps performed by the computer program product comprising:

preparing the blocks for back up;

determining whether the portable computer has been connected to a network;

if the portable computer has been connected to a network, permitting transmission of one or more of the blocks; and

interleaving the blocks with other data for transmission thereof.

29. The computer program product of Claim 28, wherein the method steps further comprise:

suspending backing up blocks when a predetermined computer storage space limit of the portable computer is reached.

30. The computer program product of Claim 28, wherein the method steps further comprise:

if the portable computer has been connected to a network and is transmitting data to the network, suspending the preparing step, and otherwise executing the preparing step until all blocks designated for back up have been prepared for back up.

32. The computer program product of Claim 28, wherein the method steps further comprise:

transmitting the block only if the portable computer has been connected to the network and an availability of the network is less than a threshold availability.

33. The computer program product of Claim 28, wherein the method steps further comprise:

for at least some of the blocks in a local file to be backed up, copying two respective characters thereof defining respective first comparison values and generating respective digital signature codes defining second comparison values;

for at least some of the blocks, periodically determining a test digital signature code and comparing it to the respective second comparison value; and when the test digital signature code of a block does not equal the respective second comparison value, determining whether the first two characters of the block equal one of the first comparison values.

34. The computer program product of Claim 33, wherein the method steps further comprise:

when the first two characters equal one of the first comparison values, determining a block digital signature code using a test byte_i as the first byte of a test block;

determining whether the block digital signature code equals one of the second comparison values;

when the block digital signature code equals one of the second comparison values, returning "resynchronized"; otherwise

when either the first two characters of the test byte_i do not equal one of the first comparison values, or when the block digital signature code does not equal one of the second comparison values, setting $i = i+1$ and repeating select steps.

35. A computer program product comprising:

a computer program storage device readable by a digital processing system; and

logic means on the program storage device and including instructions executable by the digital processing system for performing method steps for backing up at least some blocks in at least one local computer file from at least one user computer in a computer network, the method steps performed by the computer program product comprising:

(a) transmitting the local file to a data center remote from the user computer to render a remote version, the data center being accessible via the computer network, the data center having an electronic storage capacity of at least one hundred terabytes, the local file being transmitted only when the data center does not have an exact copy of the local file.

36. The computer program product of Claim 35, wherein the method steps further comprise:

(b) for each of the blocks in the local file, copying two respective characters thereof defining respective first comparison values and generating respective digital signature codes defining second comparison values;

(c) for each block, periodically determining a test digital signature code and comparing it to the respective second comparison value; and

(d) designating a block of the local file as a transmission block to be transmitted to the data center to thereby update the remote version when the test digital signature code of the block does not equal the second comparison value of the block;

(e) when the test digital signature code of a block does not equal the respective second comparison value, determining whether the first two characters of a block, starting with a test byte_i of the block, equal one of the first comparison values, wherein i = a natural number;

(f) when the first two characters equal one of the first comparison values, determining a block digital signature code using the test byte_i as the first byte of a test block;

(g) determining whether the block digital signature code equals one of the second comparison values;

(h) when the block digital signature code equals one of the second comparison values, returning "resynchronized"; otherwise

(i) when either the first two characters do not equal one of the first comparison values, or when the block digital signature code does not equal one of the second comparison values, setting i = i+1 and repeating steps (e)-(i).

37. The computer program product of Claim 35, wherein the method steps further comprise:

gathering transmission blocks in a transfer chunk; and

when the size of the transfer chunk equals a predetermined size, transmitting the transfer chunk to the data center, wherein transfer chunks are transmitted to the data center only during a predetermined period.

38. The computer program product of Claim 35, wherein the method steps further comprise:

for each local file block, determining whether a duplicate of the local file block is stored in a common file block library at the data center; and

undertaking step (a) only when a duplicate of the local file block is not stored in a common file block library.

39. The computer program product of Claim 35, wherein the method steps further comprise:

periodically copying remote versions at the data center onto a portable data storage medium; and

restoring local files by copying remote versions from the portable data storage medium when the remote versions stored on the portable data storage medium are at least as current as the remote versions stored at the data center, and otherwise restoring local files by copying remote versions from the data center.

40. A system for backing up files in user computers, comprising:

a data center including a common library of computer data;

a plurality of user computers remote from the data center and in communication with the data center;

common library logic means for determining which if any of data blocks of the user computers are common computer file blocks by virtue of being contained in the common library, computer file blocks that are determined not to be common computer file blocks being non-common file blocks; and

common library addition means for adding to the common library a non-common file block when the non-common file block appears at least a predetermined number of times in the user computers.

41. The system of Claim 40, further comprising:

initialization logic means associated with each user computer for causing file blocks except common file blocks to be transmitted to the data center;

subfile incremental back up logic means associated with each user computer for determining changed blocks that have been changed since a predetermined back up time and generating changed block signals in response thereto;

chunk transmission logic means responsive to the subfile incremental logic means for sending only changed blocks to the data center in chunks having a predetermined size;

synchronizing logic means responsive to the changed block signals for synchronizing the subfile incremental logic means; and

logic means for generating a listing of the blocks of the files stored at the data center and, associated with each block, the first two characters of the block and a digital signature of the block, the subfile incremental logic means using the listing to undertake the determining function.

42. The system of Claim 40, wherein a non-common file block is added to the common library when the non-common file block appears in at least a predetermined number of user computers.

43. A computer-implemented method for backing up computer file blocks, comprising the steps of:

(a) at a user computer, encrypting at least one block using one or more encryption keys;

(b) transmitting at least the block to a remote data center;

(c) at a restore time, transmitting at least the block from the data center to the user computer;

(d) selecting one of the keys from the key file;

(e) decrypting at least the block with the key;

(f) determining whether the decryption was successful; and

(g) when the decryption is not successful, selecting another key from the key list and decrypting the block, and repeating steps (f) and (g) until the decryption is successful.

44. The method of Claim 43, wherein the block defines a block digital signature, and the block is encrypted and decrypted to render a decrypted block, and the determining step includes:

determining the digital signature for the decrypted block;

comparing the digital signature for the decrypted block to the block digital signature; and

determining that the decryption was successful when the digital signature for the decrypted block equals the block digital signature.

45. The method of Claim 44, wherein the block is compressed prior to encryption, and wherein the method further includes the steps of:

decompressing the block; and

determining that the key is not an appropriate decryption key when the decompressing step is not successful.

46. A method for providing a session key for a transmission session for transmitting data between a transmitting computer and a receiving computer, comprising the steps of:

generating a data digital signature based on the data; and

using the data digital signature to encrypt the data.

47. The method of Claim 46, wherein the data digital signature is generated by the transmitting computer, and the method further comprises the steps of:

encrypting the data digital signature using a public key of the receiving computer; and

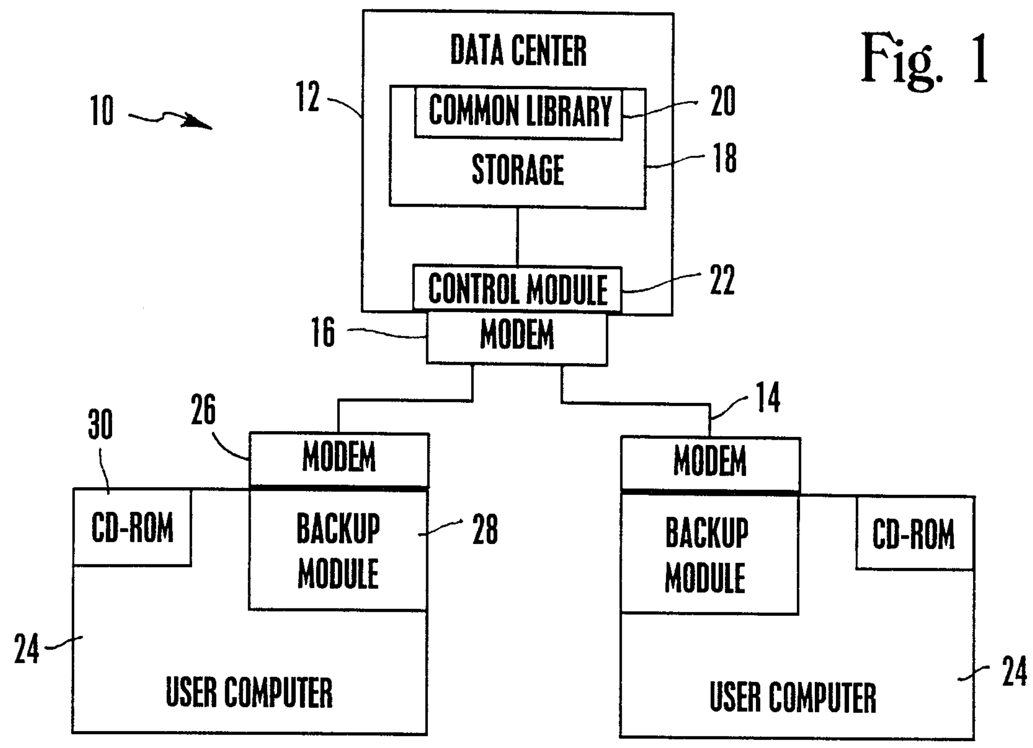
transmitting the data digital signature to the receiving computer.

48. The method of Claim 47, further comprising the steps of:

decrypting the data to render decrypted data;

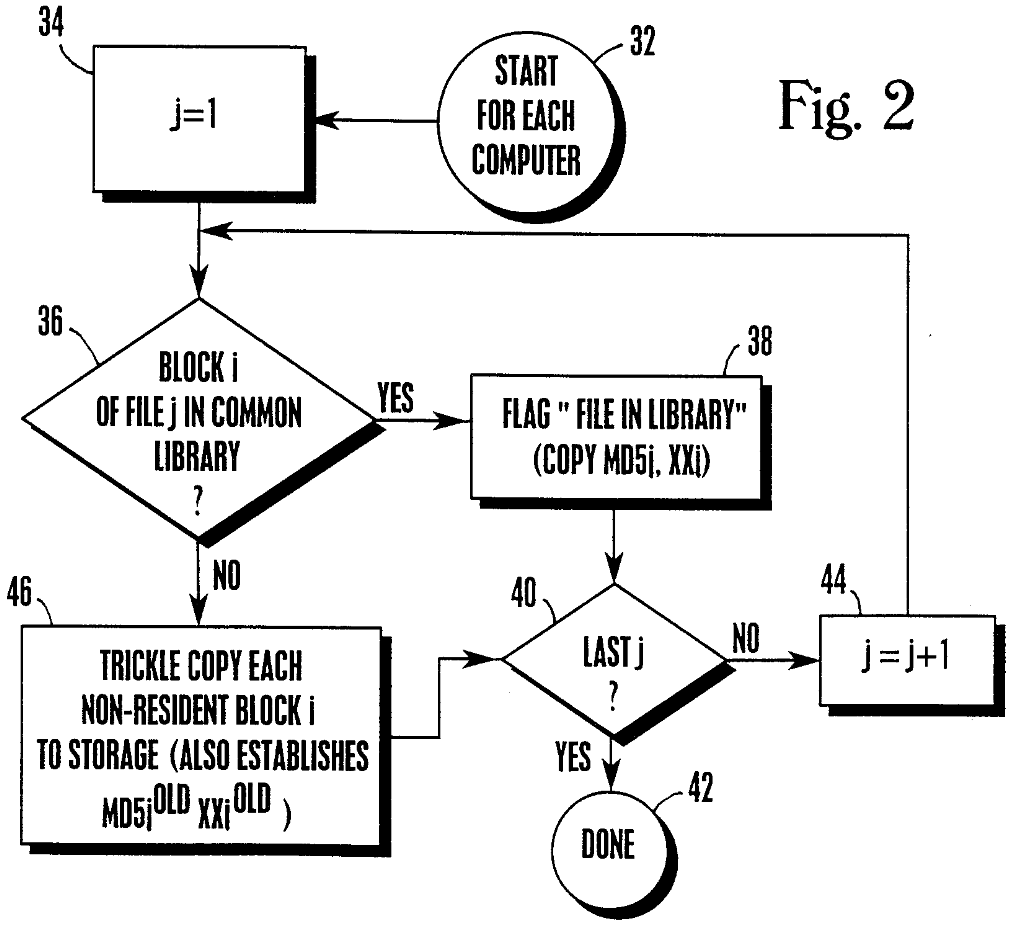
determining the digital signature of the decrypted data; and

comparing the digital signature of the decrypted data to the data digital signature to determine whether the data was transmitted satisfactorily.



DATA CENTER
COMMON LIBRARY
STORAGE
CONTROL MODULE
MODEM
MODEM
BACKUP MODULE
CD-ROM
USER COMPUTER
MODEM
BACKUP MODULE
CD-ROM
USER COMPUTER
10
12
20
18
22
16
26
30
28
14
24
24

Fig. 1



START FOR EACH COMPUTER
j=1
BLOCK i OF FILE j IN COMMON LIBRARY ?
YES
FLAG " FILE IN LIBRARY" (COPY MD5i, XXi)
NO
TRICKLE COPY EACH NON-RESIDENT BLOCK i TO STORAGE (ALSO ESTABLISHES MD5i^OLD XXi^OLD)
LAST j ?
NO
j = j+1
YES
DONE
32
34
36
38
40
42
44
46

Fig. 2

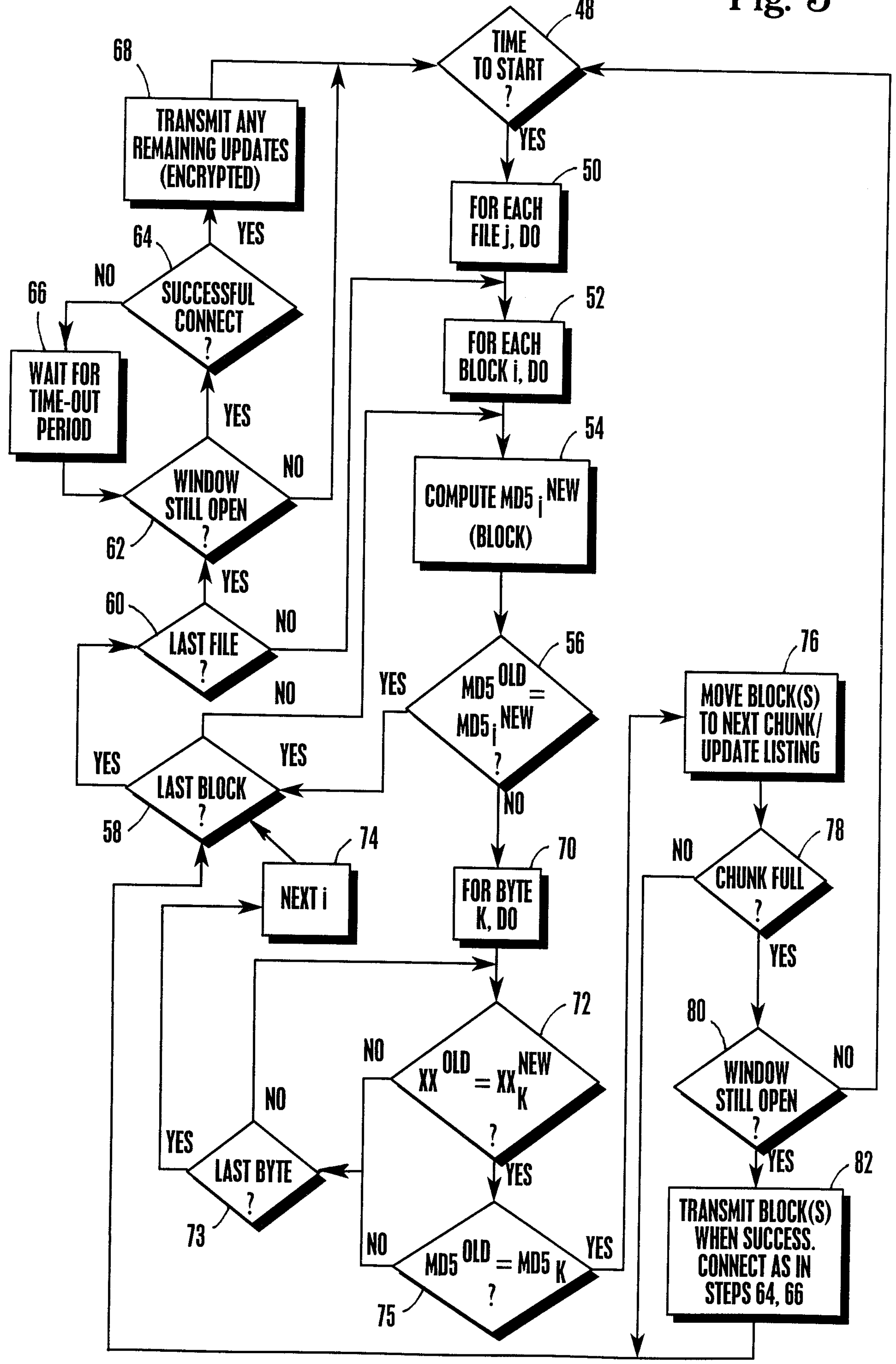
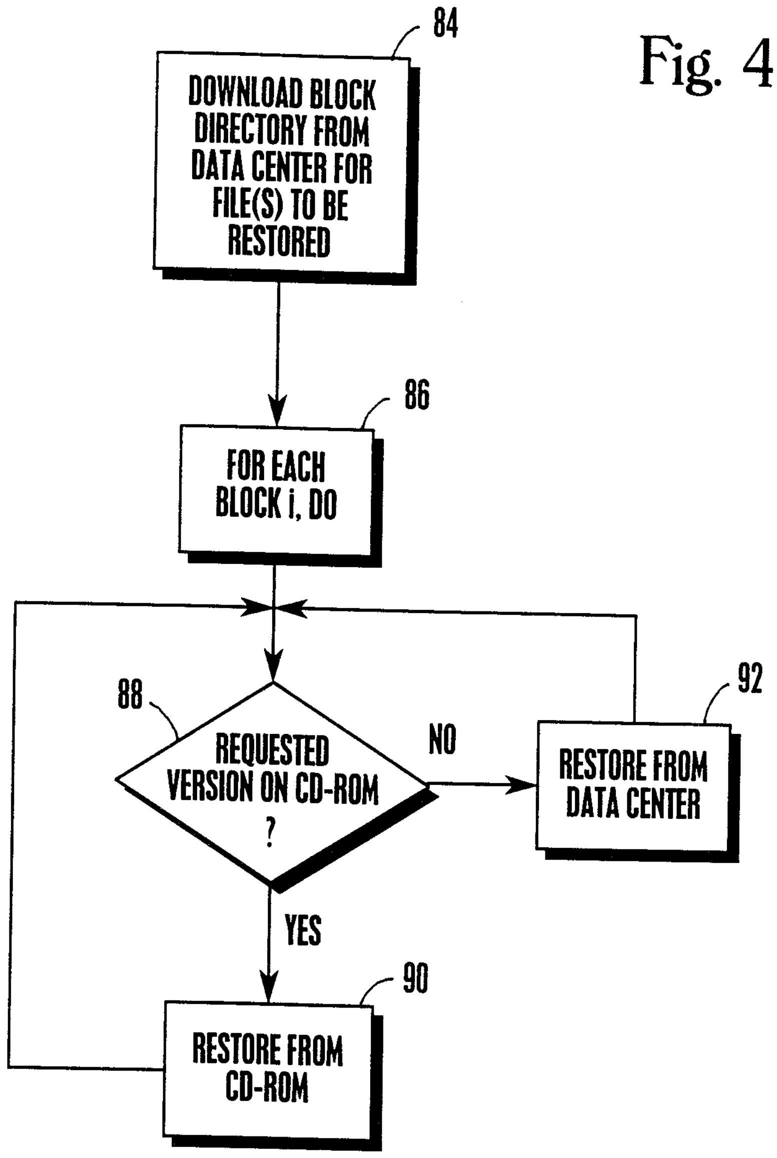

TIME TO START ?
48
YES
FOR EACH FILE j, DO
50
FOR EACH BLOCK i, DO
52
COMPUTE MD5 i NEW (BLOCK)
54
MD5 OLD = MD5 i NEW ?
56
YES
NO
FOR BYTE K, DO
70
XX OLD = XX K NEW ?
72
NO
YES
MD5 OLD = MD5 K ?
75
NO
YES
LAST BYTE ?
73
NO
YES
NEXT i
74
LAST BLOCK ?
58
YES
NO
LAST FILE ?
60
NO
YES
WINDOW STILL OPEN ?
62
NO
YES
SUCCESSFUL CONNECT ?
64
NO
YES
WAIT FOR TIME-OUT PERIOD
66
TRANSMIT ANY REMAINING UPDATES (ENCRYPTED)
68
MOVE BLOCK(S) TO NEXT CHUNK/ UPDATE LISTING
76
CHUNK FULL ?
78
NO
YES
WINDOW STILL OPEN ?
80
NO
YES
TRANSMIT BLOCK(S) WHEN SUCCESS. CONNECT AS IN STEPS 64, 66
82

Fig. 3

Fig. 4


84
DOWNLOAD BLOCK DIRECTORY FROM DATA CENTER FOR FILE(S) TO BE RESTORED
86
FOR EACH BLOCK i, DO
88
REQUESTED VERSION ON CD-ROM ?
NO
92
RESTORE FROM DATA CENTER
YES
90
RESTORE FROM CD-ROM

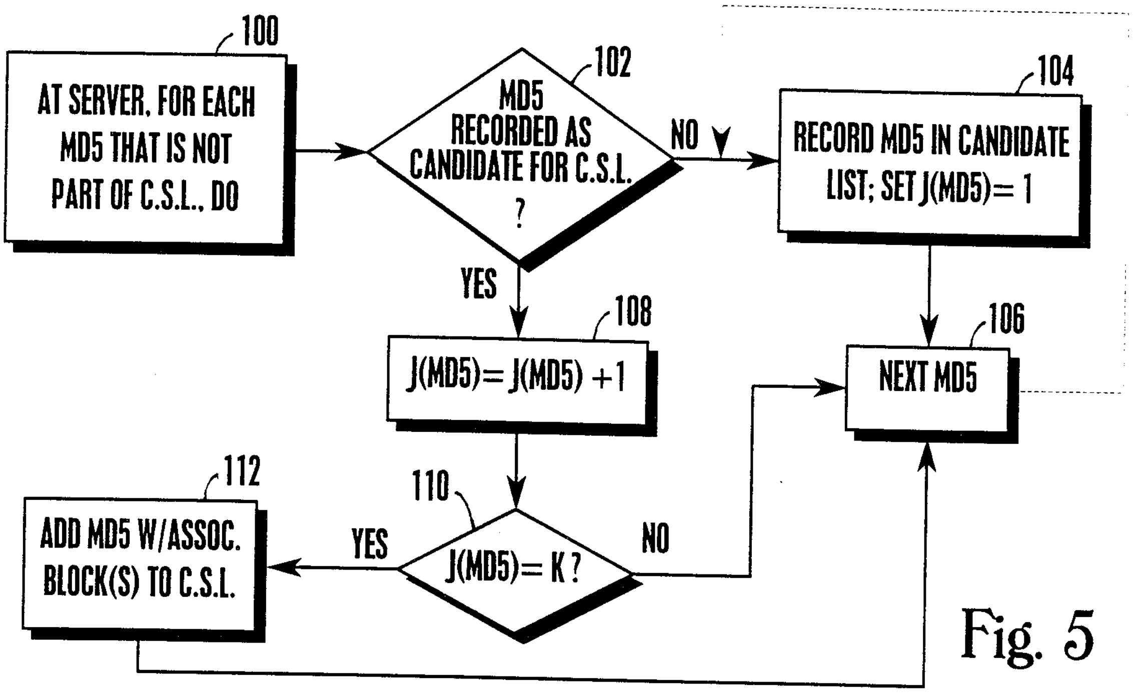

100
AT SERVER, FOR EACH MD5 THAT IS NOT PART OF C.S.L., DO
102
MD5 RECORDED AS CANDIDATE FOR C.S.L. ?
NO
104
RECORD MD5 IN CANDIDATE LIST; SET J(MD5)= 1
YES
108
J(MD5)= J(MD5) +1
106
NEXT MD5
112
ADD MD5 W/ASSOC. BLOCK(S) TO C.S.L.
110
YES
J(MD5)= K ?
NO

Fig. 5

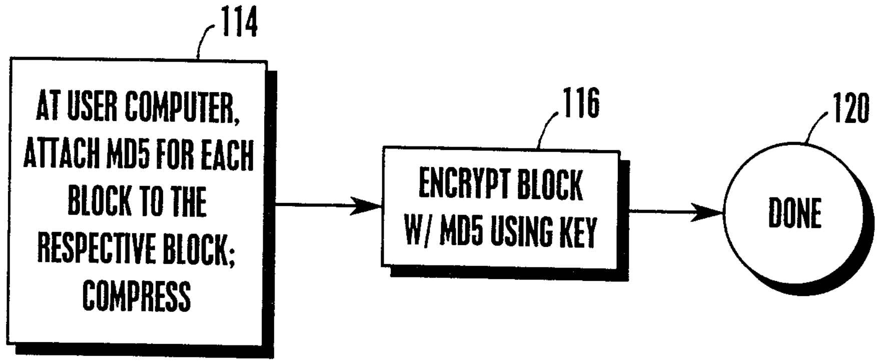
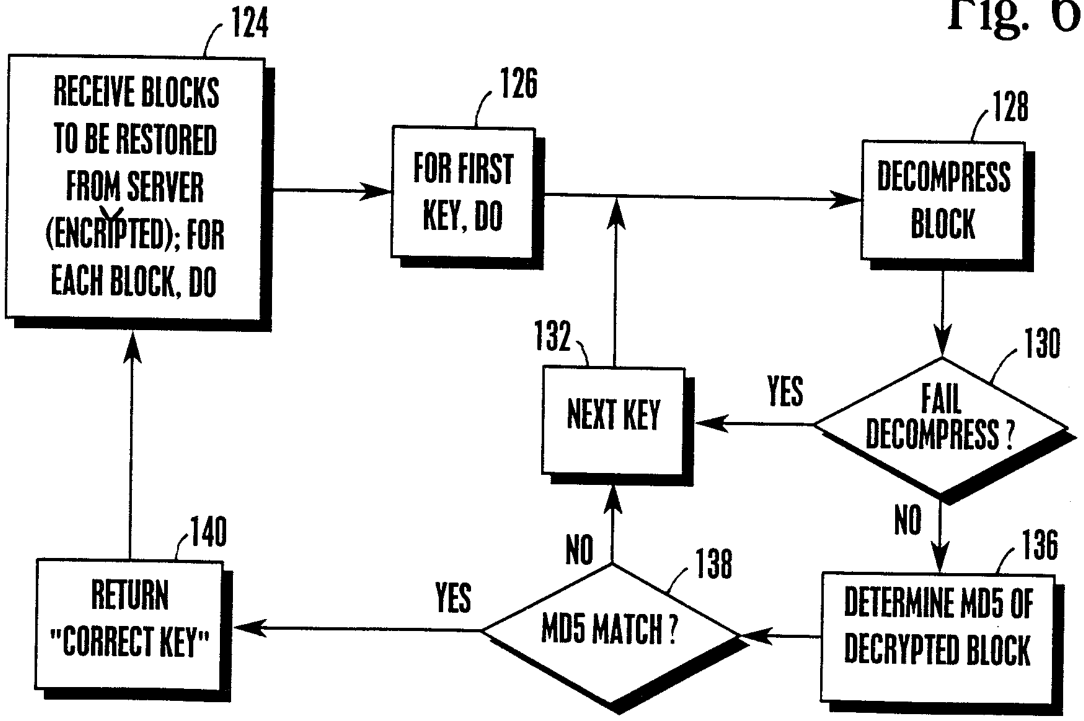

114
AT USER COMPUTER, ATTACH MD5 FOR EACH BLOCK TO THE RESPECTIVE BLOCK; COMPRESS
116
ENCRYPT BLOCK W/ MD5 USING KEY
120
DONE

Fig. 6A

Fig. 6B



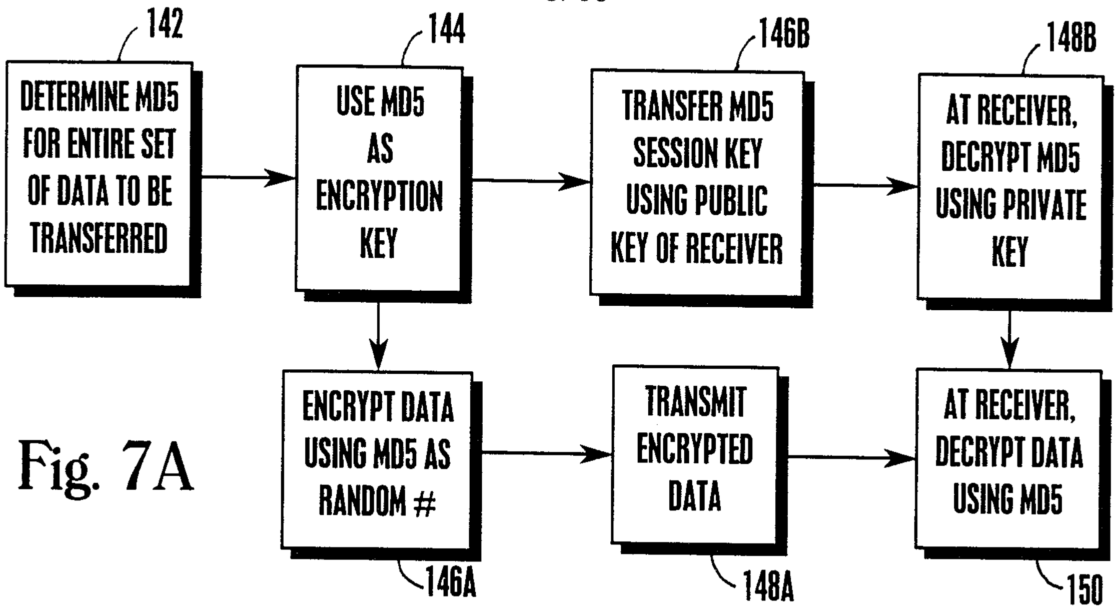

142
DETERMINE MD5 FOR ENTIRE SET OF DATA TO BE TRANSFERRED
144
USE MD5 AS ENCRYPTION KEY
146B
TRANSFER MD5 SESSION KEY USING PUBLIC KEY OF RECEIVER
148B
AT RECEIVER, DECRYPT MD5 USING PRIVATE KEY
ENCRYPT DATA USING MD5 AS RANDOM #
146A
TRANSMIT ENCRYPTED DATA
148A
AT RECEIVER, DECRYPT DATA USING MD5
150

Fig. 7A

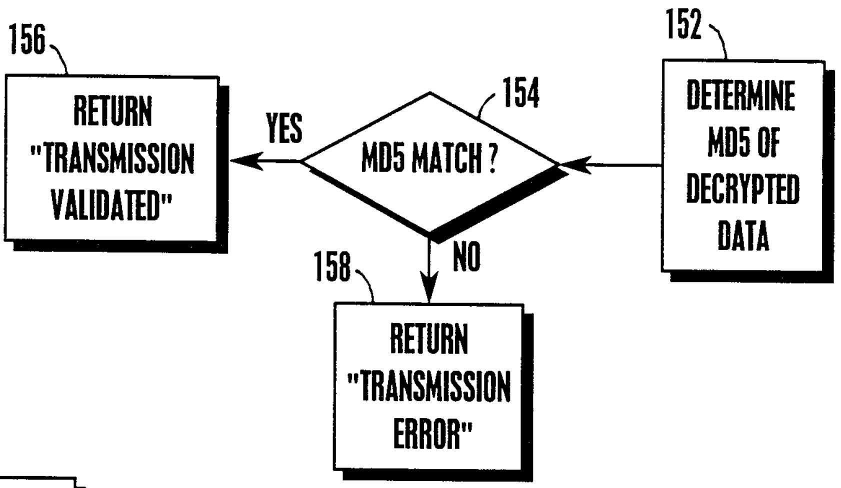

156
RETURN "TRANSMISSION VALIDATED"
YES
154
MD5 MATCH ?
NO
152
DETERMINE MD5 OF DECRYPTED DATA
158
RETURN "TRANSMISSION ERROR"

Fig. 7B

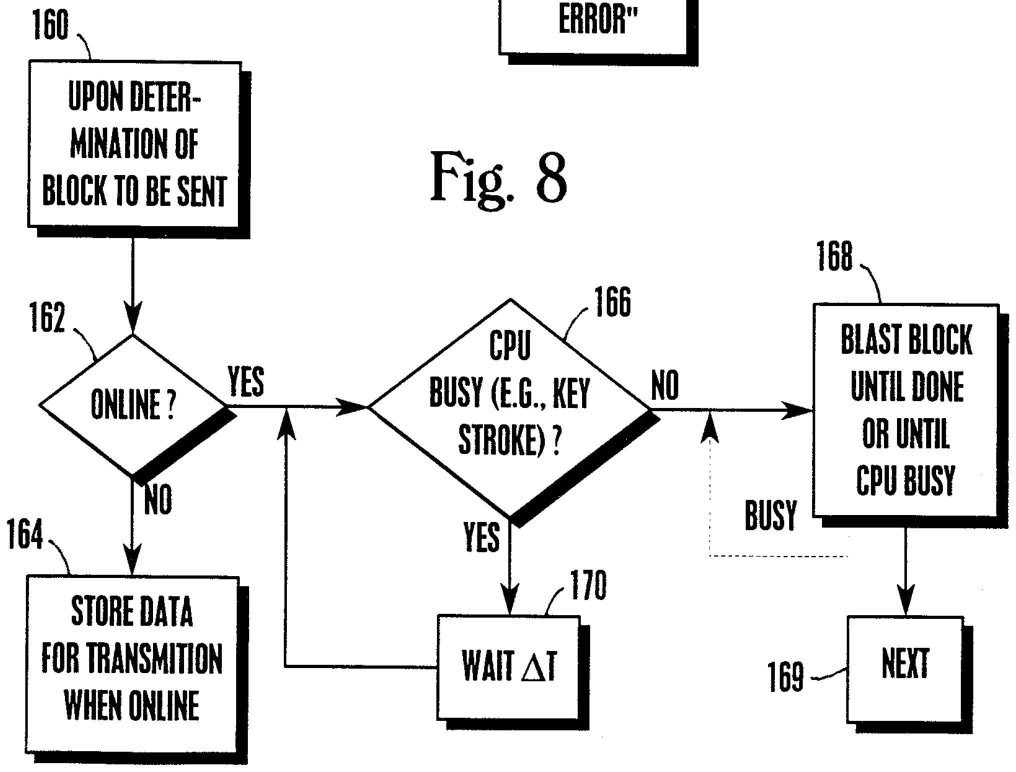
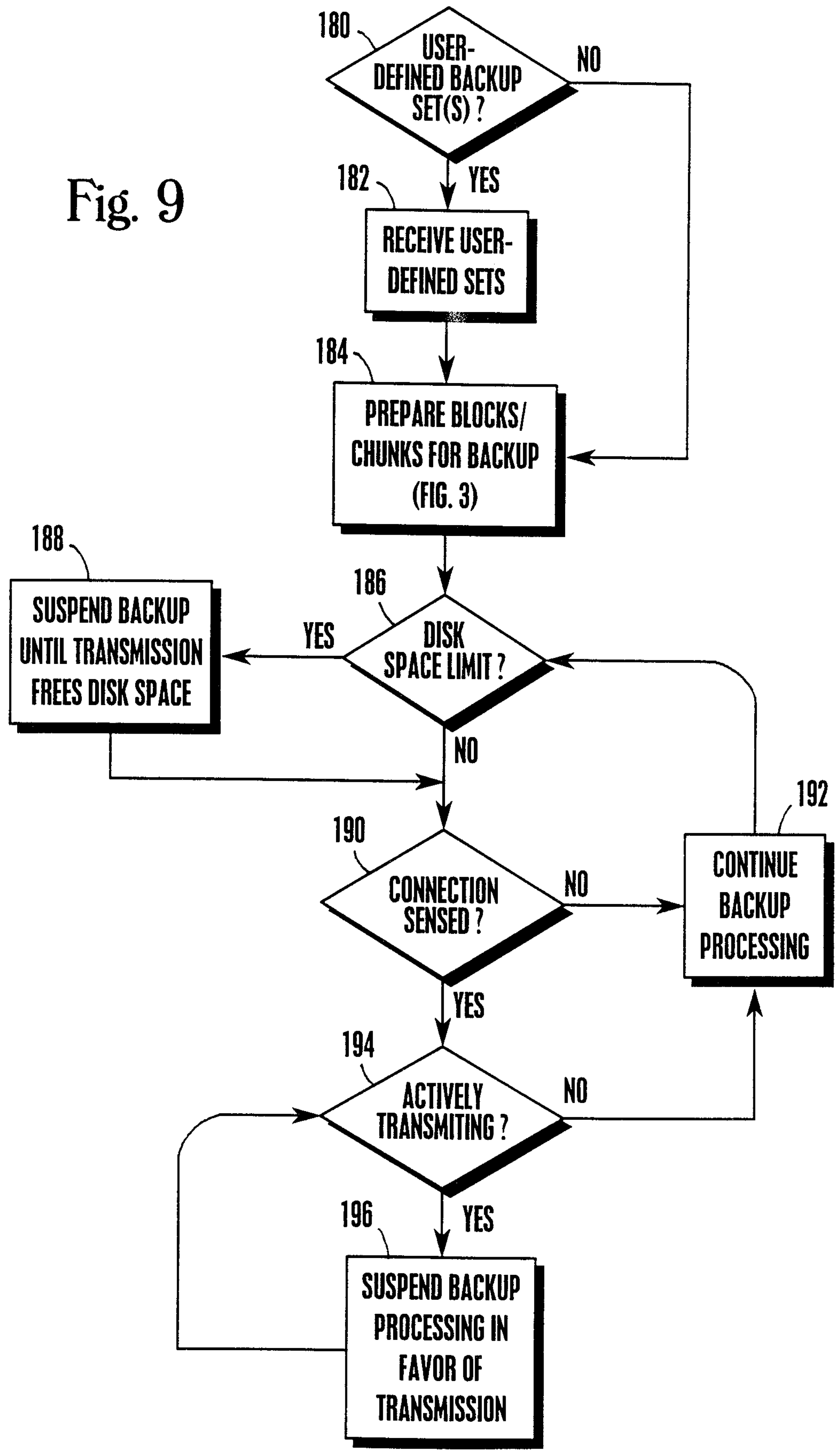

160
UPON DETERMINATION OF BLOCK TO BE SENT
162
ONLINE ?
YES
NO
164
STORE DATA FOR TRANSMITION WHEN ONLINE
166
CPU BUSY (E.G., KEY STROKE) ?
NO
YES
170
WAIT ΔT
168
BLAST BLOCK UNTIL DONE OR UNTIL CPU BUSY
BUSY
169
NEXT

Fig. 8

Fig. 9



180
USER-DEFINED BACKUP SET(S) ?
NO
YES
182
RECEIVE USER-DEFINED SETS
184
PREPARE BLOCKS/ CHUNKS FOR BACKUP (FIG. 3)
188
SUSPEND BACKUP UNTIL TRANSMISSION FREES DISK SPACE
186
YES
DISK SPACE LIMIT ?
NO
190
CONNECTION SENSED ?
NO
192
CONTINUE BACKUP PROCESSING
YES
194
ACTIVELY TRANSMITING ?
NO
196
YES
SUSPEND BACKUP PROCESSING IN FAVOR OF TRANSMISSION

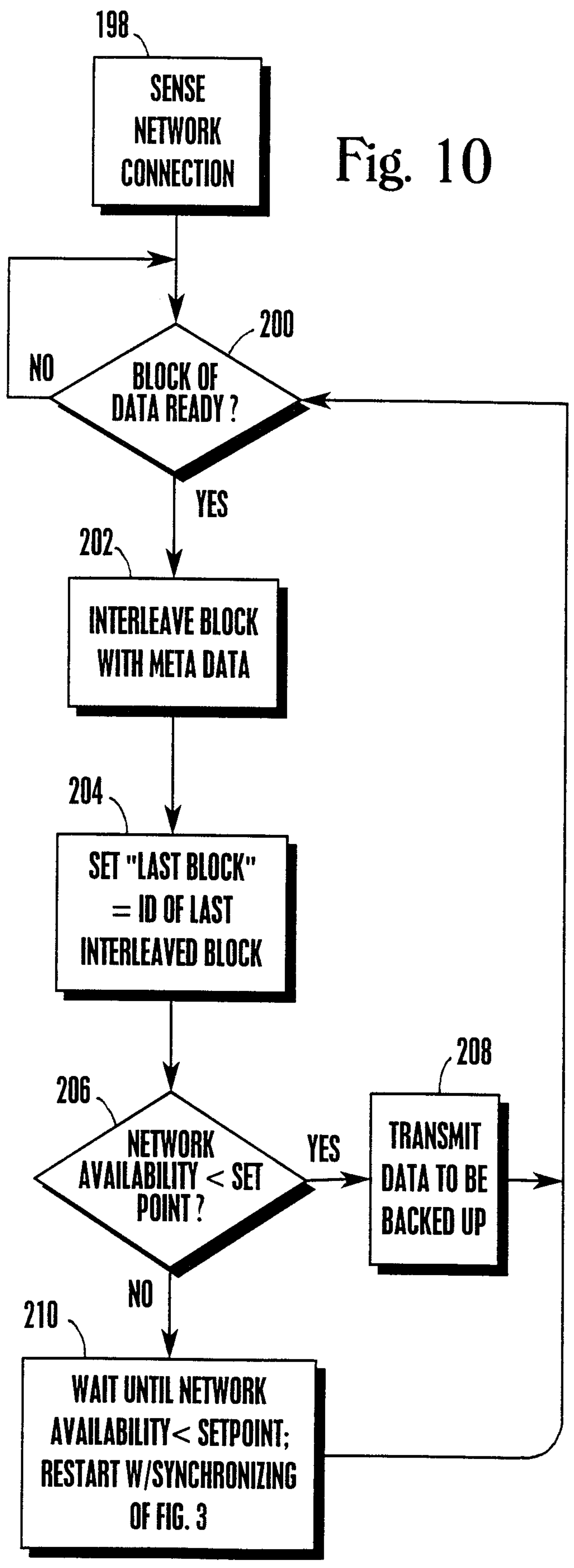

198
SENSE NETWORK CONNECTION
200
BLOCK OF DATA READY ?
NO
YES
202
INTERLEAVE BLOCK WITH META DATA
204
SET "LAST BLOCK" = ID OF LAST INTERLEAVED BLOCK
206
NETWORK AVAILABILITY < SET POINT ?
YES
208
TRANSMIT DATA TO BE BACKED UP
NO
210
WAIT UNTIL NETWORK AVAILABILITY< SETPOINT; RESTART W/SYNCHRONIZING OF FIG. 3

Fig. 10

Fig. 11

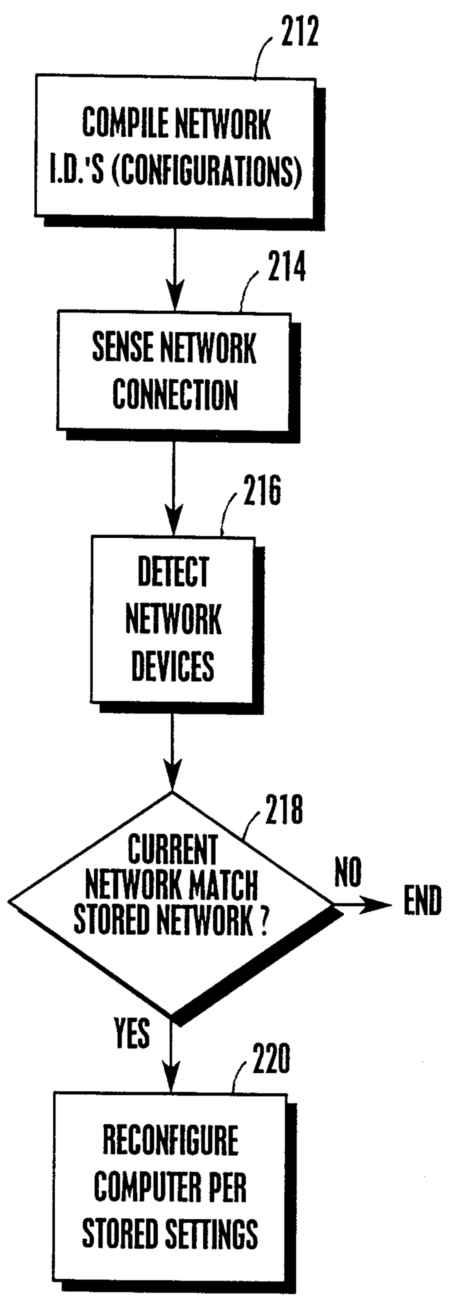
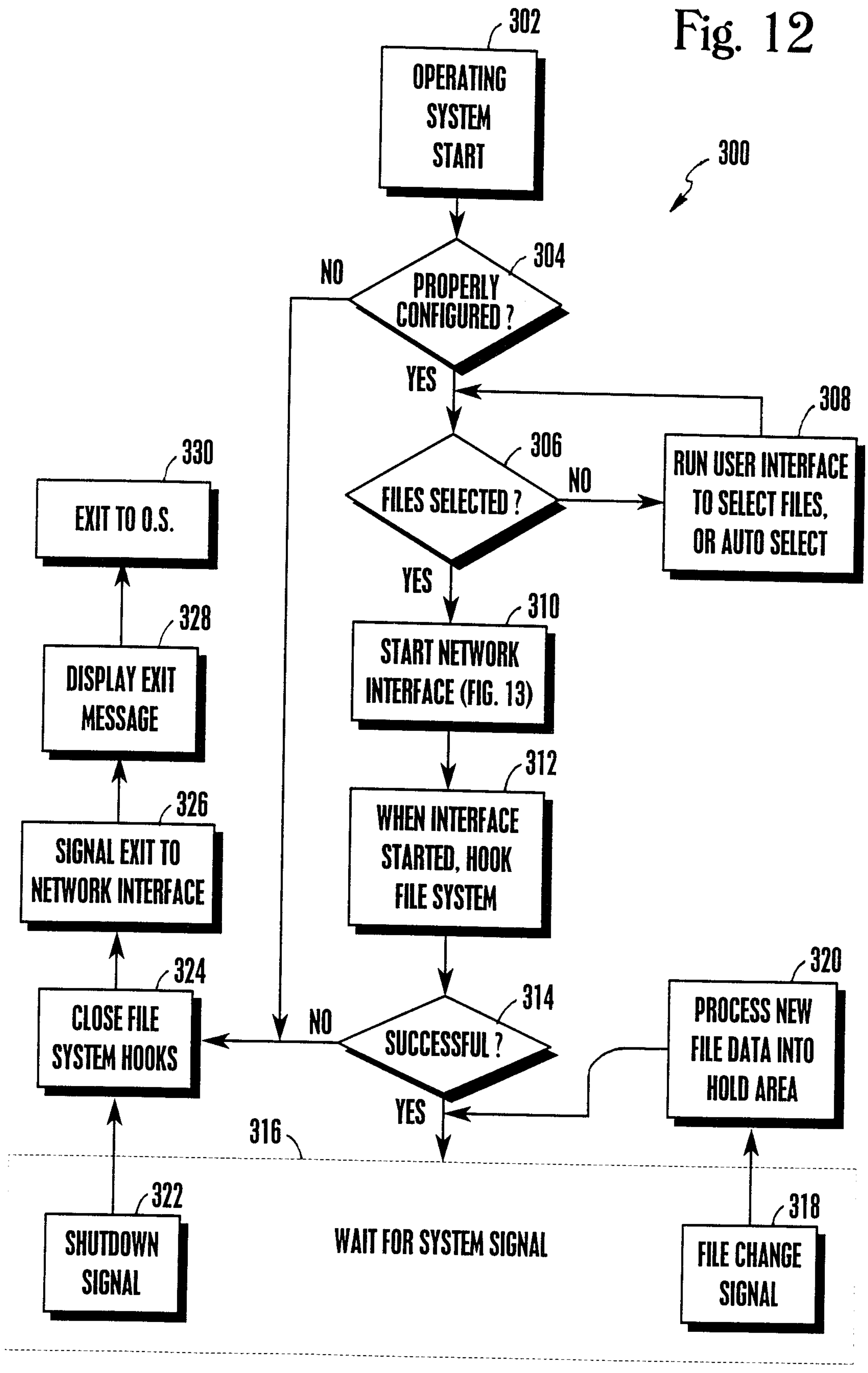

212
COMPILE NETWORK I.D.'S (CONFIGURATIONS)
214
SENSE NETWORK CONNECTION
216
DETECT NETWORK DEVICES
218
CURRENT NETWORK MATCH STORED NETWORK ?
NO
END
YES
220
RECONFIGURE COMPUTER PER STORED SETTINGS

Fig. 12


302
OPERATING SYSTEM START
300
304
PROPERLY CONFIGURED ?
NO
YES
306
FILES SELECTED ?
NO
308
RUN USER INTERFACE TO SELECT FILES, OR AUTO SELECT
YES
310
START NETWORK INTERFACE (FIG. 13)
312
WHEN INTERFACE STARTED, HOOK FILE SYSTEM
314
SUCCESSFUL ?
NO
YES
320
PROCESS NEW FILE DATA INTO HOLD AREA
330
EXIT TO O.S.
328
DISPLAY EXIT MESSAGE
326
SIGNAL EXIT TO NETWORK INTERFACE
324
CLOSE FILE SYSTEM HOOKS
316
322
SHUTDOWN SIGNAL
WAIT FOR SYSTEM SIGNAL
318
FILE CHANGE SIGNAL

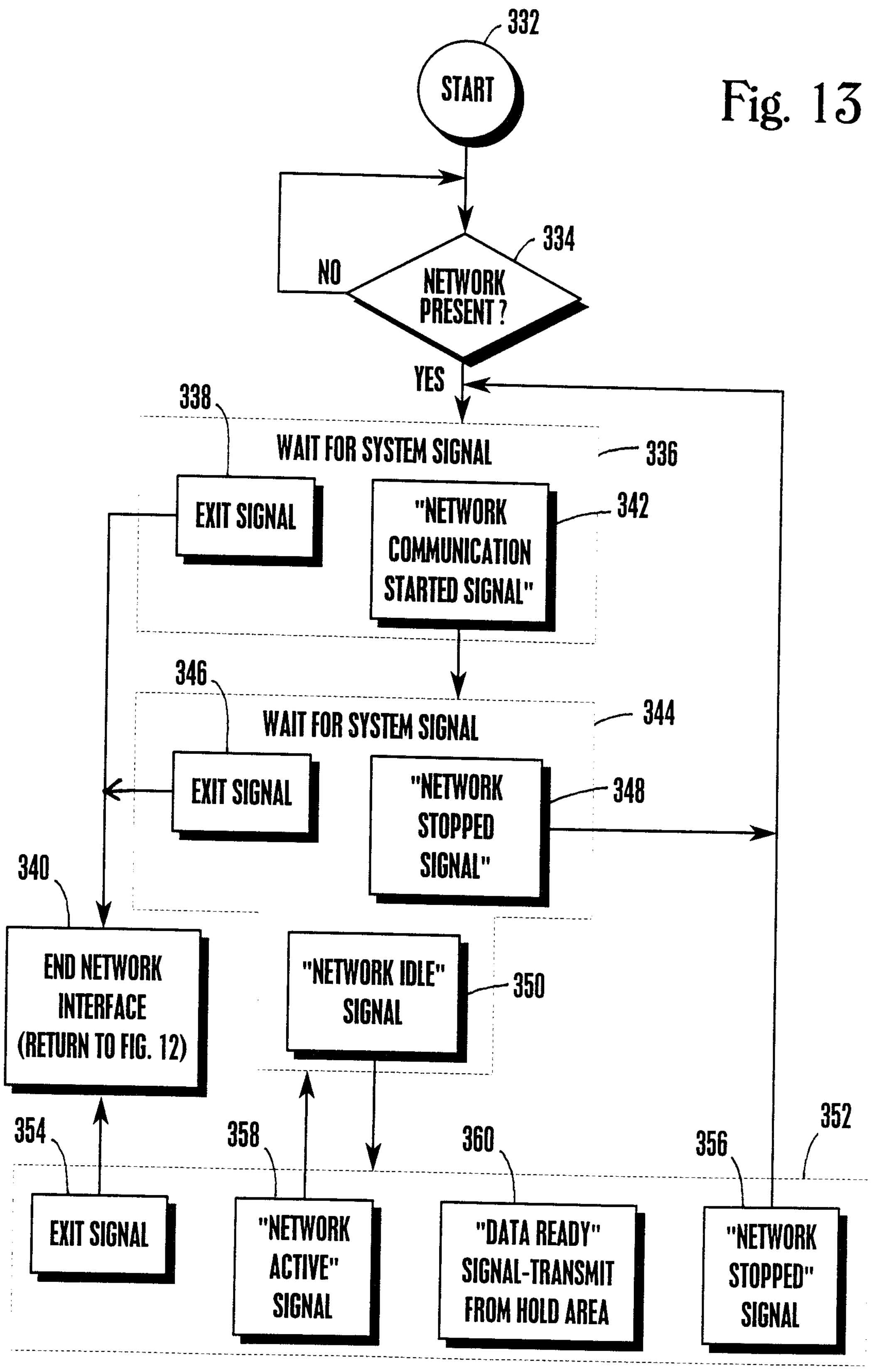

332
START
NO
NETWORK PRESENT ?
334
YES
338
WAIT FOR SYSTEM SIGNAL
336
EXIT SIGNAL
"NETWORK COMMUNICATION STARTED SIGNAL"
342
346
WAIT FOR SYSTEM SIGNAL
344
EXIT SIGNAL
"NETWORK STOPPED SIGNAL"
348
340
END NETWORK INTERFACE (RETURN TO FIG. 12)
"NETWORK IDLE" SIGNAL
350
354
358
360
356
352
EXIT SIGNAL
"NETWORK ACTIVE" SIGNAL
"DATA READY" SIGNAL-TRANSMIT FROM HOLD AREA
"NETWORK STOPPED" SIGNAL

Fig. 13

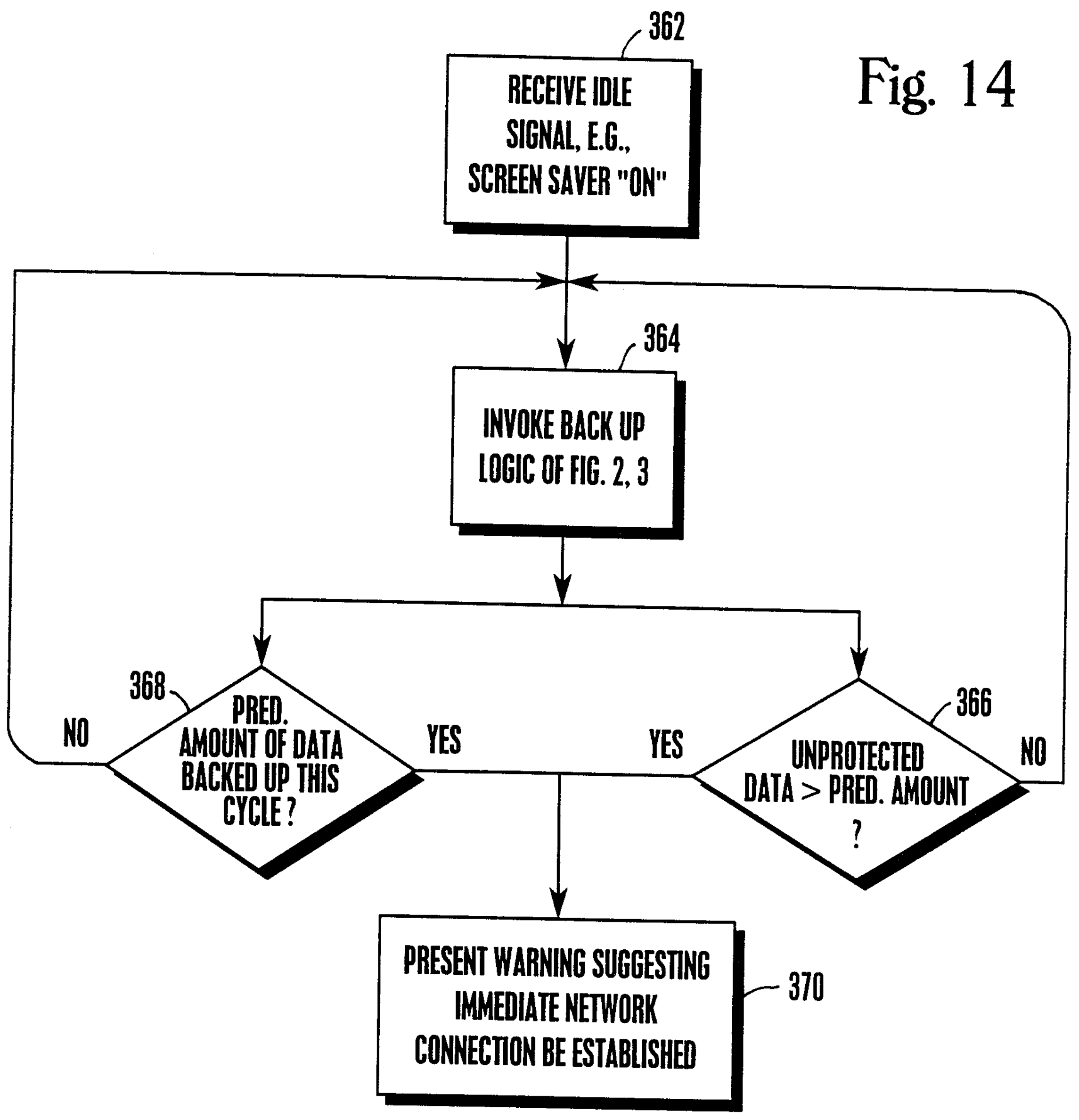

362
RECEIVE IDLE SIGNAL, E.G., SCREEN SAVER "ON"
364
INVOKE BACK UP LOGIC OF FIG. 2, 3
368
PRED. AMOUNT OF DATA BACKED UP THIS CYCLE ?
NO
YES
366
UNPROTECTED DATA > PRED. AMOUNT ?
YES
NO
PRESENT WARNING SUGGESTING IMMEDIATE NETWORK CONNECTION BE ESTABLISHED
370

Fig. 14