(12) UK Patent Application (19) GB (11) 2 394 382 (13) A

(43) Date of A Publication 21.04.2004

(21) Application No: 0224396.2

(22) Date of Filing: 19.10.2002

(71) Applicant(s):
Hewlett-Packard Company
(Incorporated in USA - Delaware)
3000 Hanover Street, Palo Alto,
California 94304, United States of America

(72) Inventor(s):
Matthew Murray Williamson
Andrew Patrick Norman

(74) Agent and/or Address for Service:
Hewlett-Packard Limited
Intellectual Property Section, Building 3,
Filton Road, Stoke Gifford, BRISTOL,
BS34 8QZ, United Kingdom

(51) INT CL[7]:
H04L 12/26 // H04L 29/06

(52) UK CL (Edition W):
H4K KFMA KTKX KTN

(56) Documents Cited:
GB 2373130 A **GB 2362076 A**

(58) Field of Search:
UK CL (Edition V) **G4A, H4K**
INT CL[7] **G06F, H04L, H04Q**
Other: **Online: WPI, EPODOC, JAPIO, INSPEC**

(54) Abstract Title: **Monitoring the propagation of viruses through an Information Technology network**

(57) Requests to send data from a first host within a network of hosts are monitored against a record of destination hosts who have been sent data in accordance with a predetermined policy. Destination host identities are stored in a buffer. The buffer size is monitored to establish whether requests from the first host are pursuant to viral activity therein. Virus related traffic is often characterised by a relatively high frequency of transmission, this is monitored by Viral Propagation Monitoring Software (500).

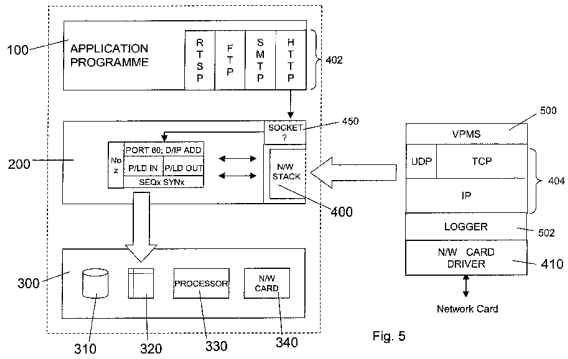

Fig. 5

GB 2 394 382 A

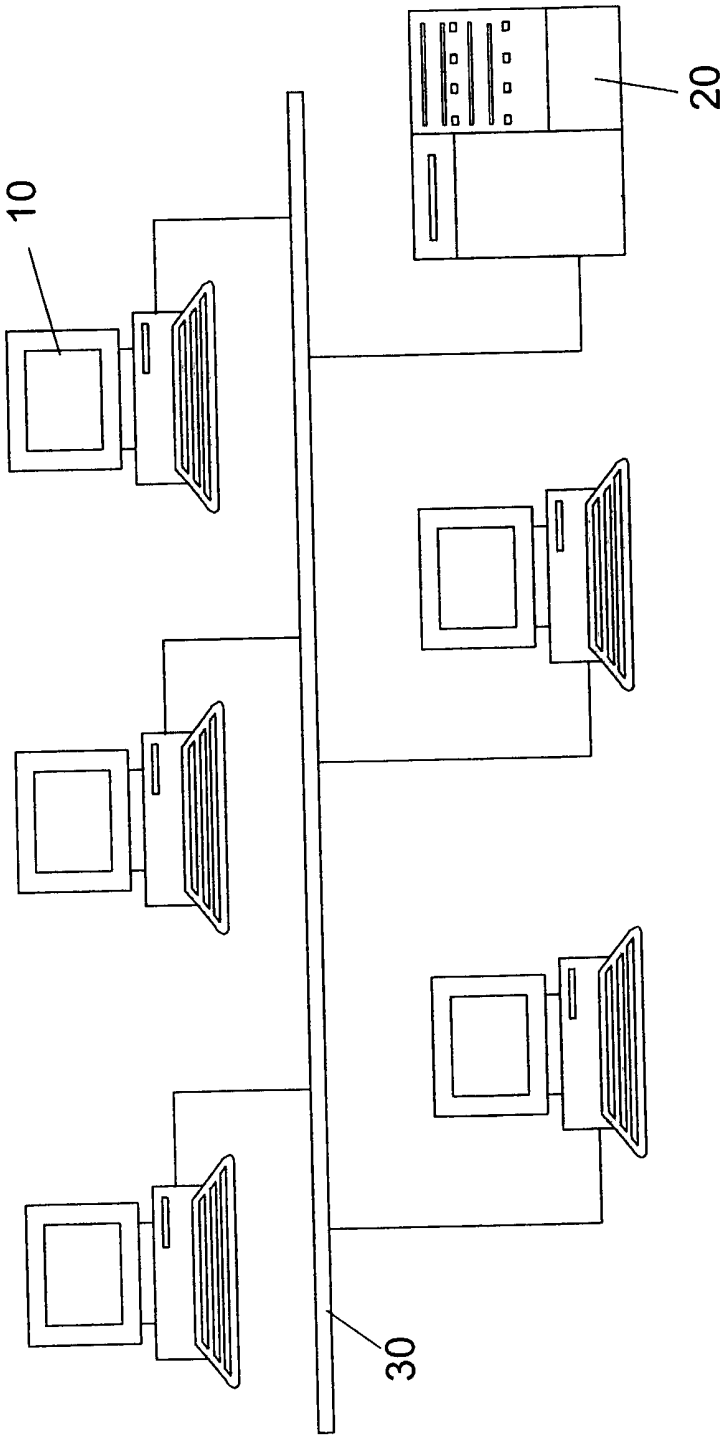

Fig. 1

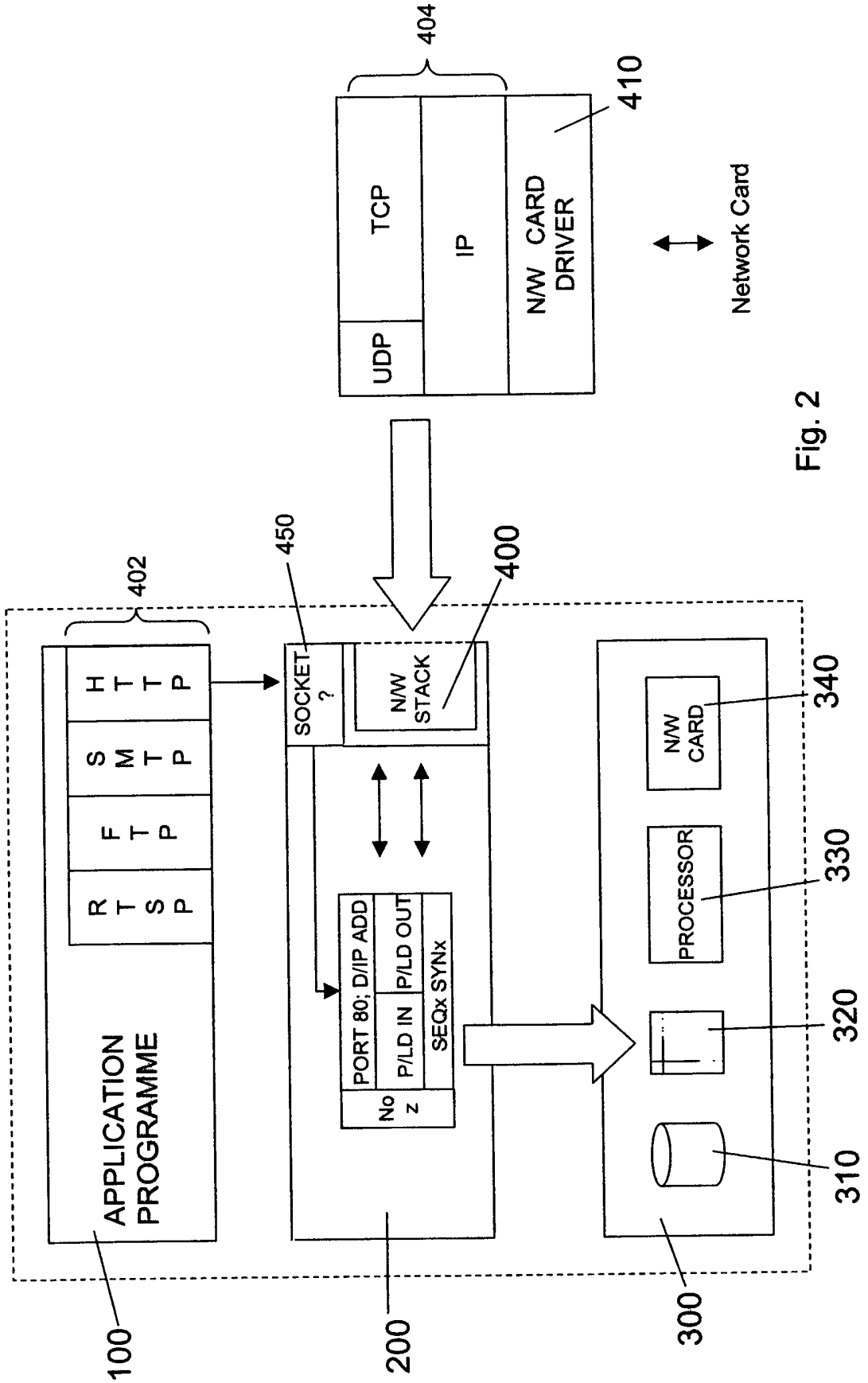
404
410
TCP
UDP
IP
N/W CARD DRIVER
Network Card
402
450
400
APPLICATION PROGRAMME
HTTP
SMTP
FTP
RTSP
SOCKET ?
N/W STACK
PORT 80; D/IP ADD
P/LD IN
P/LD OUT
SEQx SYNx
No z
N/W CARD
PROCESSOR
340
330
320
310
100
200
300

Fig. 2

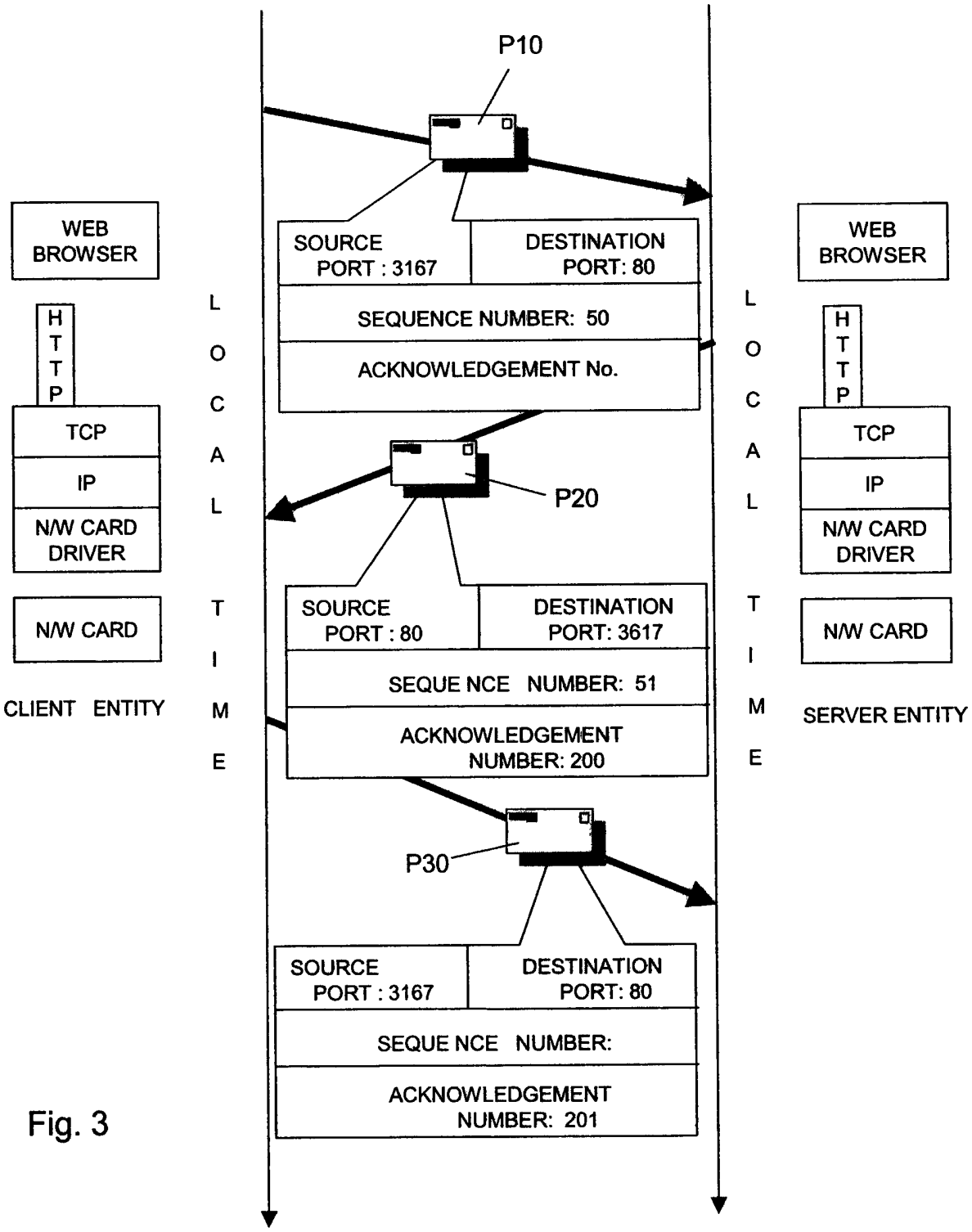
WEB BROWSER
HTTP
TCP
IP
N/W CARD DRIVER
N/W CARD
CLIENT ENTITY
LOCAL TIME
P10
SOURCE PORT : 3167
DESTINATION PORT: 80
SEQUENCE NUMBER: 50
ACKNOWLEDGEMENT No.
P20
SOURCE PORT : 80
DESTINATION PORT: 3617
SEQUE NCE NUMBER: 51
ACKNOWLEDGEMENT NUMBER: 200
P30
SOURCE PORT : 3167
DESTINATION PORT: 80
SEQUE NCE NUMBER:
ACKNOWLEDGEMENT NUMBER: 201
WEB BROWSER
HTTP
TCP
IP
N/W CARD DRIVER
N/W CARD
SERVER ENTITY

Fig. 3

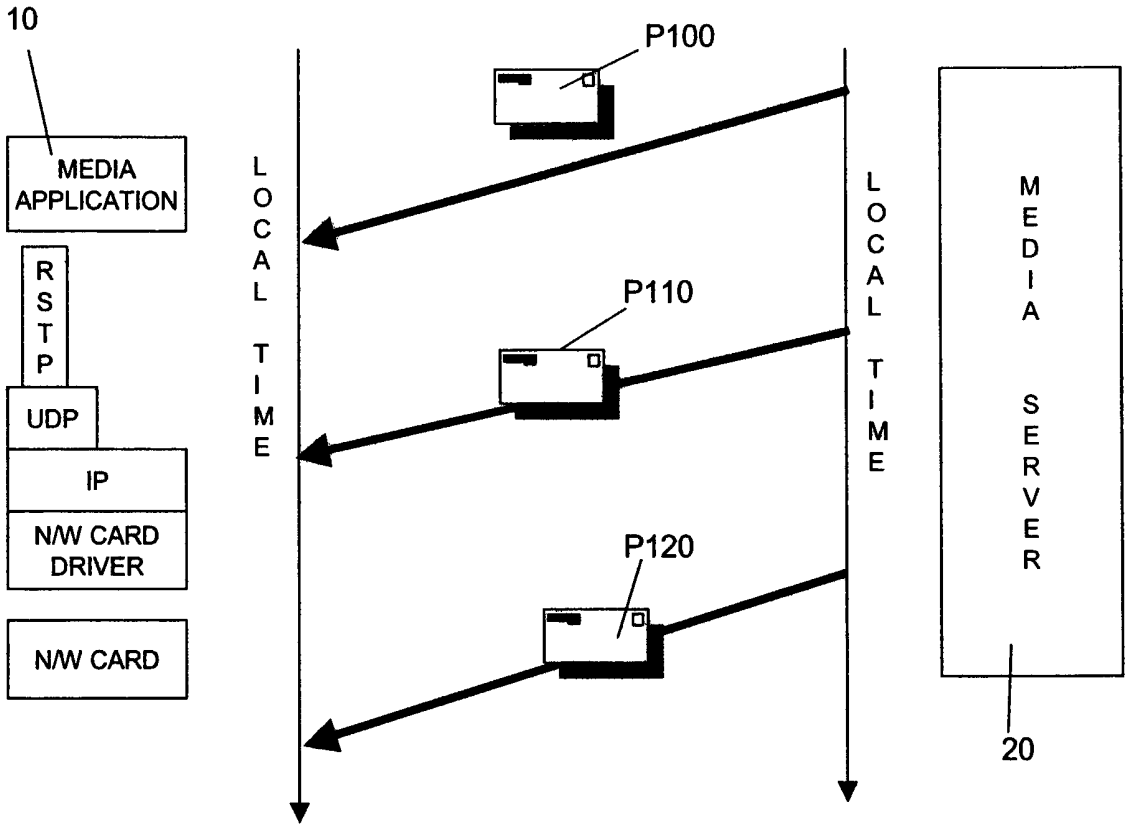
10
MEDIA APPLICATION
RSTP
UDP
IP
N/W CARD DRIVER
N/W CARD
LOCAL TIME
LOCAL TIME
P100
P110
P120
MEDIA SERVER
20

Fig. 4

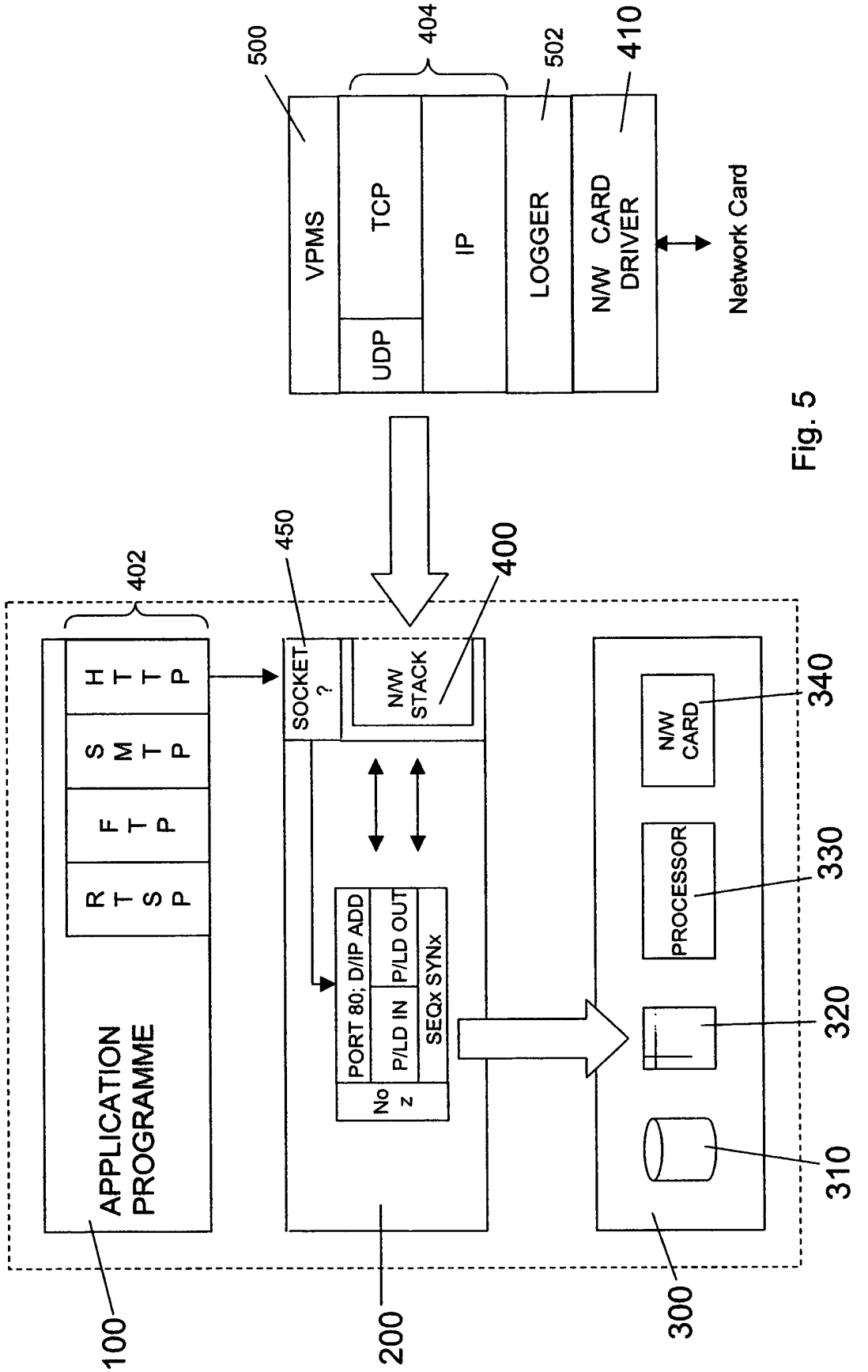
500
404
502
410
VPMS
TCP
UDP
IP
LOGGER
N/W CARD DRIVER
Network Card
402
450
400
APPLICATION PROGRAMME
RTSP
FTTP
SMTP
HTTP
SOCKET ?
N/W STACK
PORT 80; D/IP ADD
P/LD IN
P/LD OUT
SEQx SYNx
No z
N/W CARD
PROCESSOR
340
330
320
310
100
200
300

Fig. 5

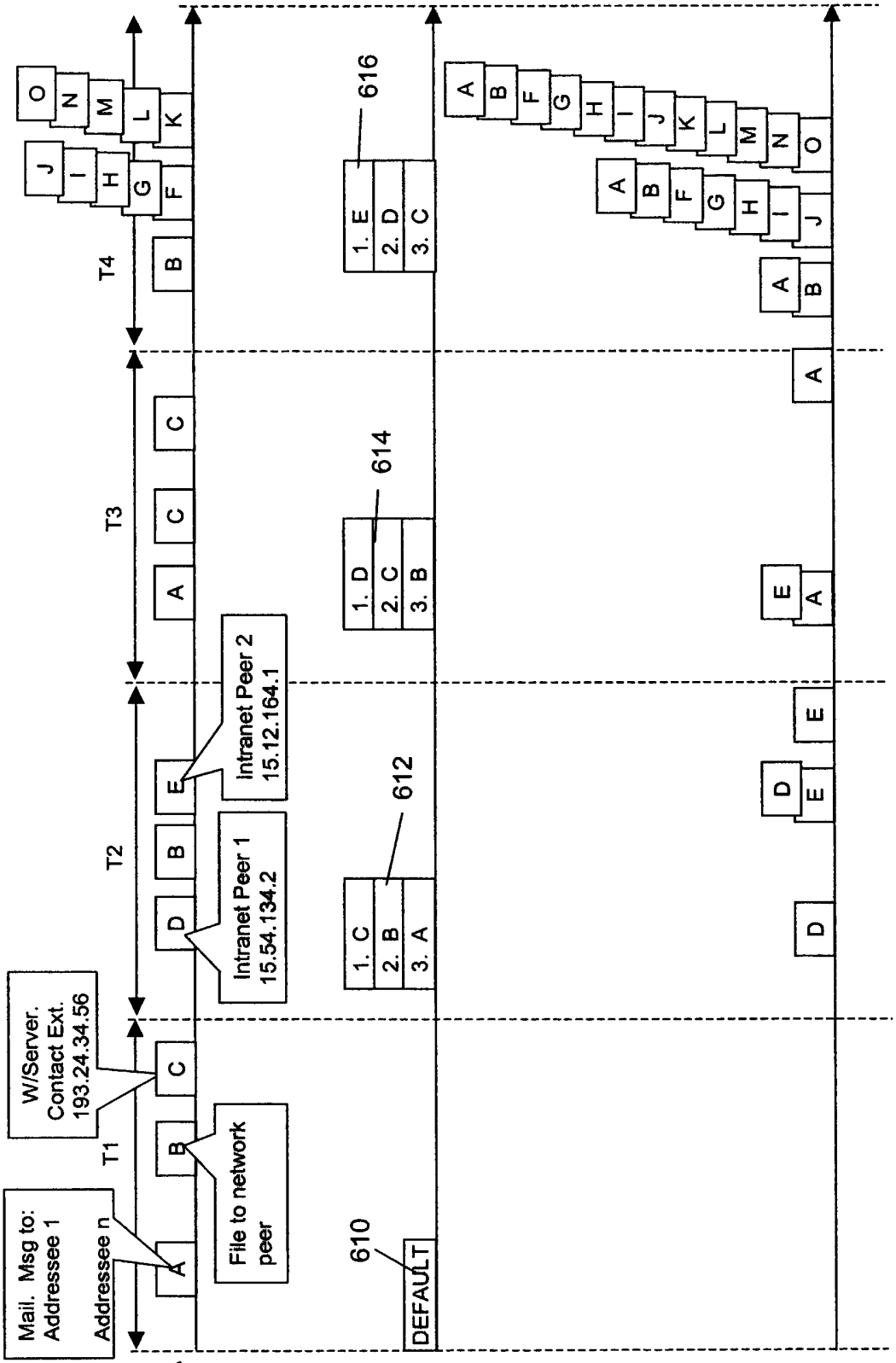
Mail. Msg to:
Addressee 1
Addressee n
W/Server.
Contact Ext.
193.24.34.56
File to network
peer
Intranet Peer 1
15.54.134.2
Intranet Peer 2
15.12.164.1
DEFAULT
610
612
614
616
T1
T2
T3
T4
1. C
2. B
3. A
1. D
2. C
3. B
1. E
2. D
3. C
6B
6C

Fig. 6A

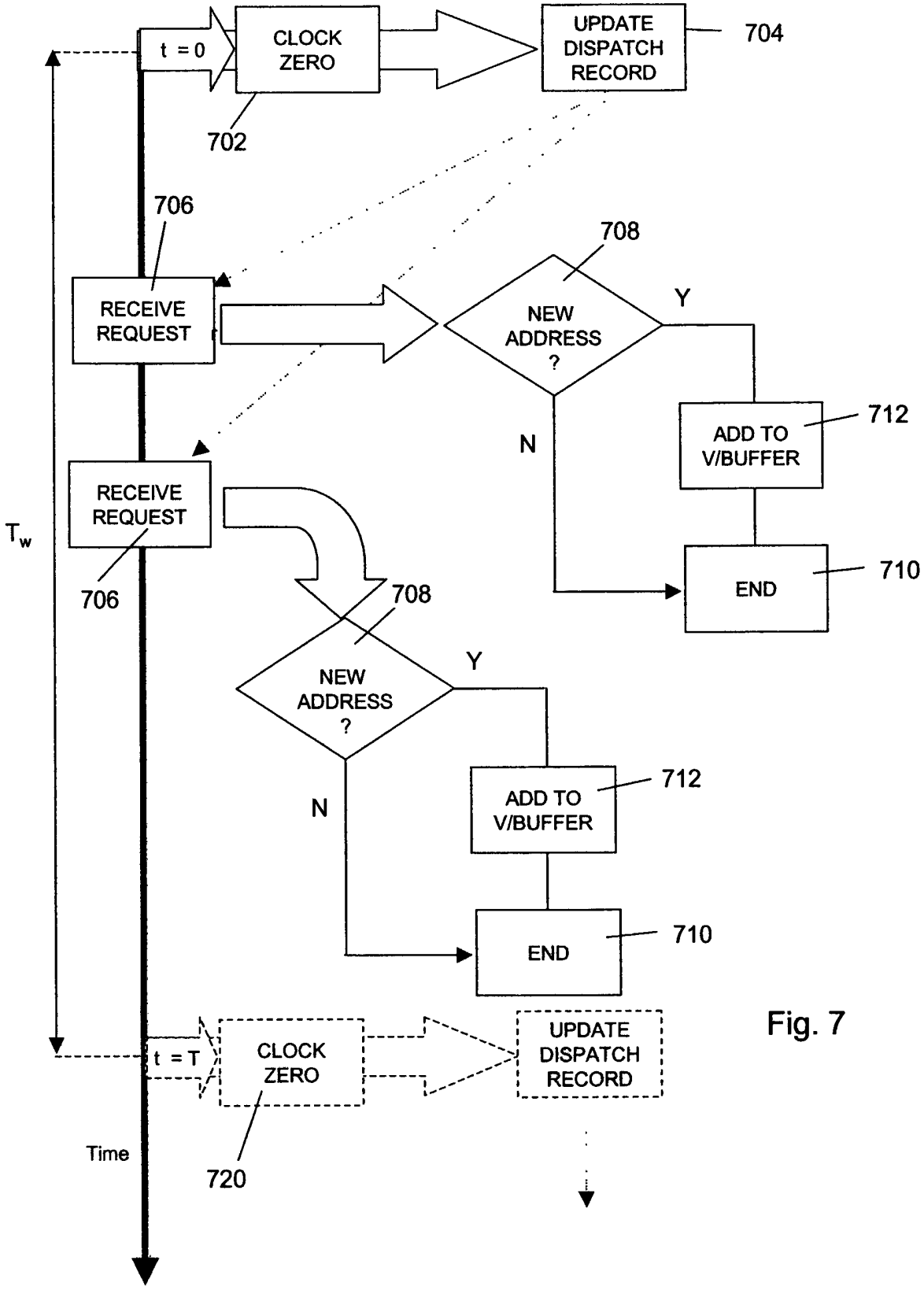
t = 0
CLOCK ZERO
UPDATE DISPATCH RECORD
704
702
706
RECEIVE REQUEST
708
NEW ADDRESS ?
Y
N
ADD TO V/BUFFER
712
END
710
RECEIVE REQUEST
706
708
NEW ADDRESS ?
Y
N
ADD TO V/BUFFER
712
END
710
T_w
t = T
CLOCK ZERO
UPDATE DISPATCH RECORD
720
Time

Fig. 7

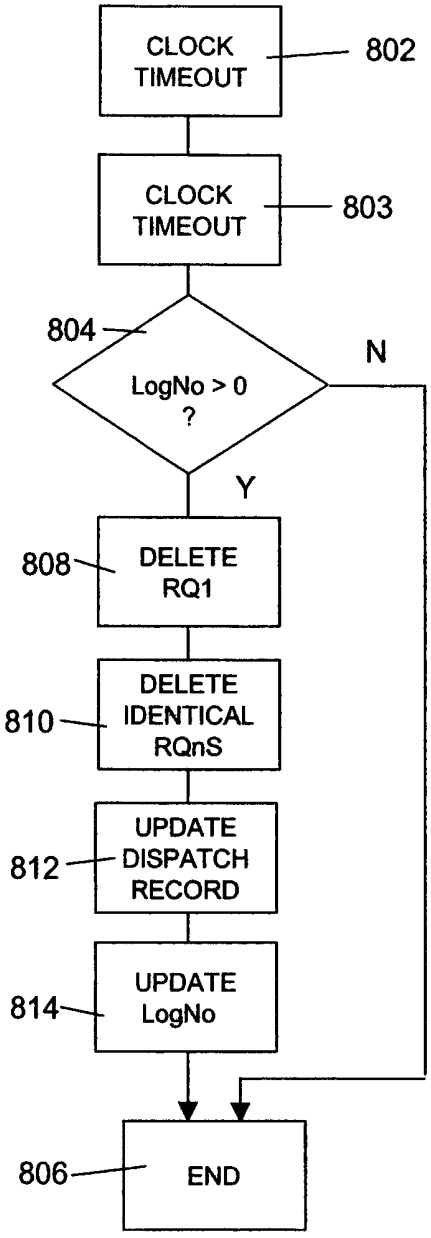
CLOCK TIMEOUT
802
CLOCK TIMEOUT
803
804
LogNo > 0 ?
N
Y
808
DELETE RQ1
DELETE IDENTICAL RQnS
810
UPDATE DISPATCH RECORD
812
UPDATE LogNo
814
806
END

Fig. 8A

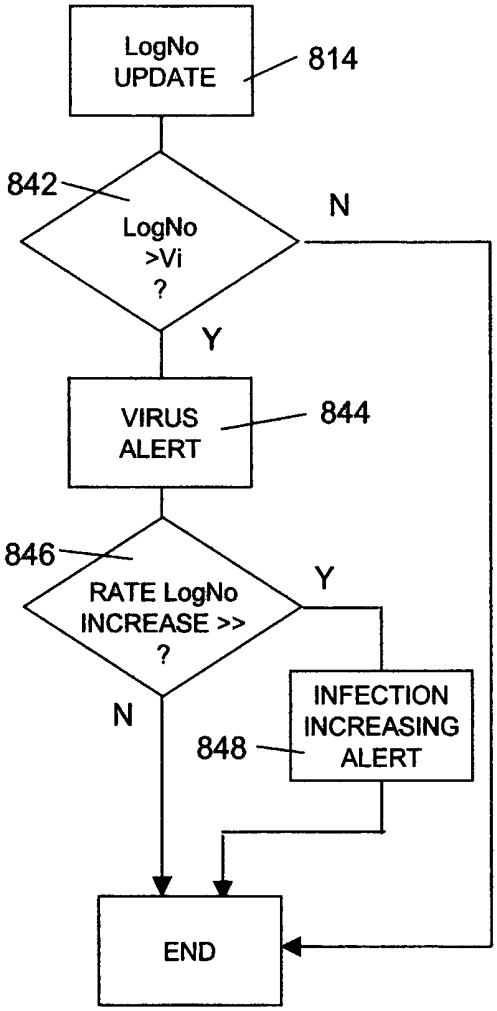
LogNo UPDATE
814
842
LogNo >Vi ?
N
Y
VIRUS ALERT
844
846
RATE LogNo INCREASE >> ?
Y
N
INFECTION INCREASING ALERT
848
END

Fig. 8B

PROPAGATION OF VIRUSES THROUGH AN INFORMATION TECHNOLOGY NETWORK

The present invention relates to the propagation of viruses through a network of interconnected processing entities.

In current network environments virtually any processing entity (or "host") is at one time or another connected to one or more other hosts. Thus for example in the case of an IT environment, a host in the form of a computer (such as a client, a server, a router, or even a printer for example) is frequently connected to one or more other computers, whether within an intranet of a commercial organisation, or as part of the Internet. Alternatively, in the case of a communications technology environment, a host in the form of a mobile telephone is, merely by virtue of its intrinsic purpose, going to be connected to one or more other hosts from time to time, and an inevitable result is that the opportunities for the propagation of viruses are enhanced as a result. For example in the case of a computer virus known as the "Code Red" virus, once assimilated within a host the virus operates to generate Internet Protocol ("IP") addresses of other potential hosts at random, and then instructs the host to send a copy of the virus to each of these randomly-generated IP addresses. Although not all of the potential hosts are genuine (since the IP addresses are randomly generated), sufficient of the randomly generated addresses are real addresses of further hosts to enable the virus to self propagate rapidly through the Internet, and as a result to cause a substantial drop in performance of many commercial enterprise's computing infrastructure.

Within the context of this specification a virus is data which is assimilable by a host to cause a deleterious effect upon the performance of either: the aforesaid host; one or more other hosts; or a network of which any of the above-mentioned hosts are a part. Thus for example, a virus may act by becoming assimilated within a first host, and subsequent to its assimilation may then cause deleterious effects within that first host, such as corruption and/or deletion of files. In addition the virus may cause self-propagation to one or more further hosts at which it will then cause similar corruption/deletion and further self-propagation. Alternatively the virus may merely be assimilated within the first host and cause no deleterious effects whatsoever, until

it is propagated to one or more further hosts where it may then cause such deleterious effects, such as, for example, corruption and/or deletion of files. In yet a further alternative scenario, a virus may for example become assimilated within a first host, and then cause itself to be propagated to multiple other hosts within the network. The virus may have no deleterious effect upon any of the hosts by whom it is assimilated, however the self-propagation through the network *per se* may be of a sufficient magnitude to have a negative effect on the speed of "genuine" network traffic, so that the performance of the network is nonetheless affected in a deleterious manner. The three examples given above are intended for illustration of the breadth of the term virus, and are not intended to be regarded in any way as exclusively definitive.

It has been established that in situations where viruses are likely to cause deleterious effects upon either one or more hosts, or the network infrastructure as a whole, one of the most important parameters in attempting to limit and then to reverse such effects is the speed of propagation of a virus. Human responses to events are typically one or more orders of magnitude slower than the propagation speeds of viruses, and so substantial difficulties are frequently apt to arise within a network before any human network administrator is either aware of the problem, or capable of doing anything to remedy it. Therefore any reduction in the initial rate of propagation of a virus through a network is likely to be of benefit to attempts to limit any negative effects, and/or to remedy them.

One existing and relatively popular approach to tackling the problems of virus propagation within a network may be thought of as an absolutist approach. Viral infection is prevented using virus-checking software, attempts to check all incoming data, for example email attachments. If subsequently a virus is discovered within a host, that host is typically removed from the network immediately, and disinfected once the nature of the virus has been established. In accordance with this philosophy each host may be thought of as contributing to protecting the network against widespread infection firstly by avoiding incidence of infection, and secondly in the event of infection, by its sacrificial removal from the network.

The present invention provides an alternative approach to infection and propagation of viruses in a network of hosts. According to one aspect of the present invention,

communications established by one or more hosts with other hosts within the network is monitored to determine whether the or each monitored host is infected, and upon detection of an infection, data by which communications from the or each monitored host are established with, and possibly additional data besides, are recorded. Depending upon the precise nature of the data recorded, it may then be used to establish *inter alia*: the nature of the virus, the manner in which it propagates.

Embodiments of the invention will now be described, by way of example, and with reference to the accompanying drawings, in which:

Fig. 1 is a schematic representation of one form of network architecture;

Fig. 2 is a schematic illustration of the conventional operational architecture of a computing entity forming a part of, for example, the network of Fig. 1;

Fig. 3 is a schematic illustration of establishment of a connection in accordance with an application protocol from Fig. 2;

Fig. 4 is a schematic illustration of data transmission in accordance with a further application protocol from Fig. 2

Fig. 5 is a schematic illustration of an operational architecture according to an embodiment of the present invention of a computing entity forming a part of a network;

Fig. 6 is a graphical representation of the operation of an embodiment of method according to the present invention;

Fig. 7 is a flowchart illustrating the operation of the method of Figs. 6

Figs. 8A and B are flowcharts illustrating further aspects of embodiments of method according to the present invention.

Referring now to Fig. 1, one typical form of network includes a plurality of client computing entities 10, and a server computing entity 20 each of which is connected to a network backbone, usually referred to as a bus 30. In the present example, each of the computing entities has a similar architecture enabling dispatch and receipt of data from other entities connected to the network. Referring now to Fig. 2, each of the entities includes what may be thought of as three functional parts: one or more application programmes 100, which in general terms may be thought of as enabling implementation of a particular task that a user of the entity may wish to perform, such as browsing the Internet, word processing and so on; hardware 300 (such as a hard drive 310, memory 320, a processor 330, and a network card 340); and an operating system 200. The operating system 200 may be thought of, in part, as an interface between the applications programmes and the hardware, and performs scheduling of tasks required by applications programmes, allocates memory and storage space amongst other things. The operating system 200 may, in accordance with this way of describing the architecture of a computing entity, also include a hierarchy, or stack 400 of programmes which provide the entity in question with the ability to dispatch and receive data to and from other entities in the network, in accordance with a number of different sets of formal rules governing the transmission of data across a network, known as protocols. The network stack 400 may be thought of as being inserted into the operating system so that the two operate in conjunction with each other. The stack 400 includes a strata of low level programmes which provide for the implementation of low level protocols 404, concerned for example with the formation of bundles of data known as "packets" (which will be discussed in more detail later), the order in which bytes of data are to be sent and, where appropriate, error detection and correction. A further, high level strata of protocols usually implemented within applications programmes ("application protocols"), apply in conjunction with the low level protocols to provide for the dispatch and receipt of data at the behest of applications programmes In the present example the application programme uses four different high level protocols 402; RTSP (real time streaming protocol), FTP (file transfer protocol), SMTP (simple mail transfer protocol – used for email), and HTTP (hyper text transfer protocol – used primarily in internet related applications), and the operating system implements two low level protocols 404: UDP (User Datagram Protocol for use with RTSP), and TCP (Transfer Control Protocol for use with the remaining three application protocols), both low level protocols being implemented

above, and in conjunction with Internet Protocol (IP). Finally, the network stack 400 includes a system programme known as a driver 410 for the network card, which in essence is low level software that controls the network card.

In the present illustrated examples, the process of establishing a connection in accordance with HTTP will be considered. Usually a request for such a connection is made by the web browser application programme, and this in turn is most likely to be at the behest of a user operating the web browser. Where this is the case, the request will identify the address or "URL" within the network of the computing entity with which a connection is sought, initially using alphanumeric characters entered at the address bar of the browser application programme (for example http://www.hp.com). Ultimately however these are "resolved" into a numerical "IP address" of the form: xxx.xxx.xxx.xxx, where x is either an integer or no number at all and each set of three X's is an integer no greater than 255. An example of an IP address is 192.168.2.2. The IP address is subsequently further resolved into what is known as a physical, or Media Access Control ("MAC") address of the network card of the destination computing entity. Resolution of the URL into an IP address, and the IP address to a MAC address usually takes place at dedicated computing entities within the network, in a manner which is well known *per se*, and will not be described further herein. This description of the connection process in accordance with HTTP, well known *per se*, has described connections legitimately requested by a user, and by means of a URL. However it should be appreciated that it is possible for example to request a connection from the web browser application programme using an IP address, rather than the alphanumeric characters of the URL. This is an aspect of the system behaviour which has been exploited by viruses, some of which randomly generate IP addresses in accordance with the rules governing their allowable format, and then seek connection to those randomly generated addresses.

In the context of the present application it should be appreciated that the term "connection" is a term of art, and is used to refer to a manner of transmitting messages in which acknowledgement of receipt of data is required, so that in the absence of an acknowledgement the connection is deemed either not to have been established, or to have failed, and the transmitted message deemed not to have arrived. On application protocol which operates using connections is HTTP, and an

example of the establishment of a connection in accordance with HTTP will now be described with reference to Figs. 2 and 3. A connection in accordance with HTTP is typically established at the behest of a web browser application programme (i.e. a programme in the applications layer 100 in Fig. 2) within the client entity, which requests a connection with a server entity, for example. When an application programme such as a web browser seeks to establish a connection with another computing entity, it initially requests what is known as a socket 450 from the operating system. A socket is effectively an allocated memory space in which data relating to the communication sought by the web browser (in this instance) is stored. Upon receiving a request for a socket, the operating system duly creates or "opens" one (which in effect means that memory is allocated), and returns a socket number, which is the identifier for that particular socket. In Fig. 2 the particular socket is indicated by reference numeral 450, and the number of the socket is "z", while the part of the operating system which allocates the socket is shown as a "layer" above the network stack, by which it is sought to indicate that, from a methodical perspective, use of the socket (further uses of which will subsequently be described) in the case of outgoing data, precedes the passage of data from the application programme through the network stack. Once a socket has been opened, the web browser then requests that the socket z is "bound" to the IP address with which a connection is sought, and a parameter known as the "port" number (which is essentially a label identifying the application protocol used), by writing these parameters in the socket (which in due course will additionally contain further data). The port number for connections via HTTP is usually port 80. Once a socket has been created and bound the browser then requests that a connection be established, and this causes the emission of what is known as a data packet P10 to the destination computing entity. The requesting packet P10 contains: an identification of the destination port, i.e. an identification of the suitable application protocol for handling messages transmitted over the requested connection (here, because the connection is established in accordance with HTTP, port 80); a source port (here 3167) which is an arbitrary number (but one which is not: (i) already in use at that time, and (ii) not already allocated as a standard number to define a port identified in accordance with established standards) whose purpose is to provide, to the client requesting the connection, an identification of the connection in acknowledgement messages (e.g., since it is entirely possible that there may simultaneously be two are more

connections using the same protocol this may be used to distinguish one such connection from the other; a flag indicating that the synchronisation status of the requesting entity is set to "on" (meaning that sequence numbers - which indicate the order of the packet in a total number of packets sent - between the requesting and destination computing entity are to be synchronised), and an initial sequence number 50 (this could be any number). Upon receipt of this packet, the destination machine sends back a packet P20 identifying the source port as 80, the destination port as 3167, a flag indicating that the acknowledgement status is "on", an acknowledgement number 51which augments the sequence number by one, and its own synchronisation flag number 200. When the requesting entity receives this packet it returns a further packet P30 once again identifying the source and destination ports, and a flag indicating that its acknowledgement status is on, with an acknowledgement number 201 (i.e. which augments the sequence number by one). Once this exchange is complete, a connection between the client and server entities is defined as being open, and both the client and server entities send messages up through their respective network stacks to the relevant application programmes indicating that a connection is open between them. In connection with the socket, it should also be noted that the socket comprises an area 460 allocated to store the actual body of the message which it is desired to transmit (sometimes known as the outbound message content, or the outgoing payload), and similarly a further area 470 allocated to store the body of messages which are received (inbound message content, or incoming payload).

When the outgoing payload is to be transmitted, the TCP layer breaks it up into packets (i.e. data structures such as those illustrated above in Fig. 3, but further including at least part of the payload), and the IP layer attaches an IP address header. When an incoming message arrives, it passes up through the network stack, i.e. from the network card 340, up through the Internet Protocol software, etc., and is written in to the relevant socket (as identified, *inter alia* from the port number), from which the application programme retrieves the incoming payload.

Data may alternatively be transmitted using the protocols RSTP/UDP/IP (indicating the hierarchy of protocols in the network stack adopted in conjunction with each other to transmit the data) which do not require a connection; the dispatching entity sends a packet to the destination entity, and does not require an acknowledgement of receipt.

Referring now to Fig. 4, when transmitting data in accordance with RTSP/UDP, media for example is streamed to a client entity 10 from a media server 20 in a series of packets P100, P120, P120....., and the client does not acknowledge receipt of any of them. Streaming in accordance with this protocol typically follows an initial request to establish a connection between the client and the server by some other connection based protocol, for the purpose of identifying a destination port on the client, amongst other things.

Thus far all that has been described is entirely conventional. Referring now to Fig. 5, in accordance with an embodiment of the present invention, a layer of viral propagation monitoring software 500, runs within the network stack of one or more machines within the network. The VPMS acts as a gateway for all outbound data from the computing entity on which it is running, and operates to monitor the propagation of viruses within the network by observing what is, in accordance with a predetermined policy, defined as "unusual" behaviour in contacting other entities (also known as "hosts", since they may acts as hosts for viral infection) within the network. In accordance with one aspect of the present invention, it has been established that in many networks, normal network traffic (i.e. non-virally related) is characterised by a relatively low frequency of events in which data is sent to destination hosts (i.e. hosts which are the intended destination for data) within the network which have previously not been contacted. In contrast, virally-related traffic is often characterised by a relatively high frequency events in which data is dispatched (or attempts are made to dispatch data) to previously uncontacted destination hosts. Broadly speaking, the function of the VPMS is to monitor abnormal and therefore possibly virally-related traffic, as defined in accordance with a predetermined policy, and to record such abnormal traffic.

In the present example the VPMS operates upon the basis of a series of time intervals or time windows, which in the present illustrated example are of predetermined and constant length T_n In any given time window T_n the VPMS monitors requests to send data to "new" destination hosts, i.e. destination hosts whose identities differ from those specified in a record of identities of destination hosts most recently contacted. The record only holds a predetermined number N of destination host identities, so that a destination host is classified as new if it is not one of the N most recently contacted

destination hosts. The number of new hosts allowed per time window, and the value of N are determined on the basis of the policy, typically defined by a system administrator, and the policy is preferably formulated to take account of the nature of non virally-related network traffic. In this way, the VPMS operates to monitor the speed at which a virus resident on the host may propagate from that host to other hosts within the network.

Referring to Fig. 6A, over the course of a time window T1, various applications programmes running on the workstation send requests via the VPMS to send data (whether by connection or otherwise) to other hosts within the network ("outbound requests"): the email application programme, which requests dispatch of an email message (having multiple addressees) to a mail server, Mail (Request A) using SMTP, the file management application programme requesting dispatch of a file recording a text document to another user (Request B) via FTP, and the web browser programme which requests connection, (typically via a Web Proxy server), W/Server in order to connect to a site using HTTP (Request C). In the present example, outbound requests to the VPMS from each of these hosts are requests to send data to an identified destination host, and are ultimately manifested by the dispatch of one or more data packets in accordance with the relevant application protocol. The term "request" is intended to be interpreted broadly to encompass any indication (usually from an application programme, although by no means necessarily) that contact with a destination host is sought, and for ease of terminology, the transmission of a request is to be interpreted as indicating that data is transmitted pursuant to a request to transmit such data.

The VPMS operates in accordance with a routine illustrated in Fig. 7, whose features will now be described in more detail in conjunction with Figs. 6A-C, although Fig. 7 should be regarded as a generic illustration of the operation of the VPMS routine, rather than a specific illustration of individual event depicted in Figs. 6. As explained above, the VPMS operates with reference to a series of time intervals, or windows, which in the present example are of constant length. The routine is initiated at step 702 by a clock (typically the clock which defines the time windows) indicating that a time window has commenced. At step 704 the routine then updates a dispatch record, which is a record of the identities of a predetermined number N (which in this

example is 3) of destination hosts most recently contacted (in accordance with the policy – see later) in the previous time window are stored (and which are shown for each time window in Fig. 6B). At this point the routine is effectively in a waiting mode until a request to send data is received at step 706. This is a step whose occurrence is entirely outside the control of the VPMS since it usually is initiated at the behest of an application programme, as is the case with Requests A, B and C. Each of these requests passes through the relevant application protocol layer in the network stack from the respective application programme by which they were generated, to the VPMS, and this event is labelled in Fig. 7 as step 706. Step 706 may be thought of as a triggering event, so that when a request passes into the VPMS, the identity of the requested destination host specified in each the request is matched with the dispatch record. This matching process therefore determines whether the requested destination host is a new host, and is represented at step 708. In the present example, somewhat artificially, but nonetheless serving to illustrate the principles underlying the present invention, the time interval T1 is the first time interval after start-up of the computing entity. The VPMS therefore matches the destination host identities for each of the Requests A-C against identities held in a "default" dispatch record 610 for the time period T1, which may be (and in the illustrated example, is) simply a record of the three hosts most frequently contacted during the lifetime of the host on which the VPMS is running. In the present example the three most frequently contacted hosts, and therefore the three identities retained in the default dispatch record are those of the mail server (Request A), the file server (Request B) and the web proxy server (Request C). Since each of the three outbound requests from the workstation during the time period T1 identify a host destination matching one of the three host identities in the default dispatch record, and therefore none of the Requests is seeking to establish contact with a new destination host, the VPMS therefore takes no action and simply ends at step 710.

During the course of the second time interval T2, three further outbound requests are received, identifying host destinations "Intranet Peer 1" (Request D), Request B (described above) and "Intranet Peer 2" (Request E) are received. As in the previous time window, as each request triggers an individual VPMS routine for that request, i.e. a step 706 as it passes through the VPMS, and is followed by the step 708 of matching the identity of the host destination in the request with the identities present

in the dispatch record 612 for this time window T2 is performed in order to establish whether the request is new. The dispatch record however is now a genuine record of the identities of the three hosts contacted most recently during the previous time window T1 (although coincidentally this is identical to the default dispatch record). Upon receipt of Request D, the VPMS routine for that request establishes at step 708 that the identity of this host is not in the dispatch record 612, i.e. that it is a new destination host. It therefore proceeds to step 712, where it adds a copy of the Request D to a virtual buffer whose contents are shown in Fig. 6C, and then ends at 710. In one preferred embodiment, the entire contents of the socket relating to Request D are duplicated in the virtual buffer. However in an alternative embodiment, where for example the payload is large, this is omitted. On receipt of Request B, the VPMS establishes at a step 708 that B is present in the dispatch record, and so the VPMS routine ends at step 710. Request E is also a new request within the time window T2 and so at a step 712 the identity of host E is added to the virtual buffer.

Because receipt of requests are the trigger for the commencement of the routine illustrated in Fig. 7, neither the number of occasions in a given time window in which the VPMS routine is run, nor the timing of their commencement can be known in advance. Additionally, as illustrated in Fig. 7, it is possible for two (or indeed more, although only two are illustrated in Fig. 7) routines to be running in temporal overlap, since one may still be running when another is triggered by a further request. Similarly, a request may trigger the execution of the routine of Fig. 7 just prior to the end of a time window (a situation also illustrated in Fig. 7, with steps which occur at the end of a time window/the beginning of a subsequent time window being shown in dashed lines), so that the execution of the routine may overlap temporally with a part of the next time window. The approach taken by this embodiment of the present invention to this issue of overlap is relatively simple: if at the commencement of time window T_{n+1}, the update of the dispatch record for a previous time window T_n has been completed during the simultaneous running of a VPMS routine commenced in the previous time window T_n, but prior to execution the step 712 (adding a request to the virtual buffer) for that routine, the subsequent update of the virtual buffer in that step 712 will be treated as if performed for a request received in the current time window T_{n+1}. This approach has the benefit of being simple, although it may on

occasions yield minor inaccuracies, with a request being recorded as being outside of the policy simply because processing of the request received and initially processed during one time window extended into the next time window, but this is not significant overall.

At the end of the time window T2, the virtual buffer contains two new requests. At this juncture (i.e. at end of time period T2), the policy which the VPMS is designed to monitor comes into play. In the present example, the policy provides that a single new host may be contacted per time interval. This element of the policy is monitored by a first buffer management routine, which is illustrated schematically in flowchart form in Fig. 8A, and begins at step 802 with the advent of a clock timeout, that is to say that the clock (not shown) which defines the time intervals T_n has completed another time period, following which, at step 803 the routine counts the number of requests in the virtual buffer to update the variable known as LogNo, this being the number of requests in the virtual buffer at any moment. At step 804 the routine determines whether there are any requests in the virtual buffer, and it does this by examining the value of LogNo, to determine whether it's greater than 0. If there are no requests in the virtual buffer the routine ends at step 806. In the present illustrated example however it can be seen that over the course of the time interval T2 two requests, D and E have accumulated in the virtual buffer, and so the routine proceeds to step 808, at which the first request RQ1 (i.e. the one which has been in the buffer for the longest time) is deleted from the buffer. At step 810, the routine then searches the buffer for other requests specifying the same destination host and deletes any such requests, since they are effectively regarded as one request identity. This is followed at step 812 by updating the dispatch record so that it accurately reflects the identity of the three hosts most recently contacted in accordance with policy. It should be noted that the dispatch record does not therefore necessarily reflect the identities of hosts which have most recently actually been contacted, if requests to these hosts are outside of the policy. For example in this case the destination host of Request E, which although contacted, was not contacted in accordance with the policy of one new destination host per time interval. This updating of the dispatch record can be seen reflected in Fig. 6B, where the dispatch record contains the identities of Requests D, C, B. The final step in the first buffer management routine is the updating of the value of the variable LogNo denoting the size of the virtual buffer, which in this

example, following the transmission of the Request D, is one (i.e. the single Request E). Thus, in the same way that the dispatch record is a record of all requests which have been transmitted in accordance with policy, the virtual buffer is effectively a record of all requests which have been transmitted outside that policy.

The role of the virtual buffer is to enable a determination to be made with regard to whether the host upon which the VPMS is running is virally infected. One way in which this can be manifested is the size of the virtual buffer. A state of viral infection may therefore be defined in terms of the size of the buffer, and the stage of any such viral infection by the rate of change of the buffer size. This follows from the generally different behaviour of virally-related and non virally-related network traffic, in that non virally-related or "legitimate" network traffic usually involves contacting only a relatively small number of new destination hosts, whereas, because viruses tend to propagate by transmission to as many disparate destination hosts as possible, an instance of a large number of requests to contact a new destination host will typically be indicative of viral infection. The virtual buffer may be thought of as a queue of virtual new requests waiting for opportunities to be virtually transmitted in accordance with policy (since their "counterpart" real requests are simply transmitted without hindrance). The size of the virtual buffer is therefore one indication of whether there is viral infection, since a large buffer size is indicative of a large number of requests to contact a new host within a short space of time. An alternative indication of viral infection may be the existence of an increasing buffer size. Conversely, generally speaking a buffer size which is steadily declining from a relatively high value may be indicative of a temporary increase in legitimate traffic levels. It can be seen therefore that buffer size may be used to interpret the existence of viral infection with varying levels of complexity, the interpretation typically being something which is defined in the policy.

A second buffer management routine, illustrated in Fig. 8B monitors the virtual buffer, and is triggered by performance of step 814 from the routine of Fig. 8A, i.e. an update in the value of the variable LogNo, following which, at decision step 842, the routine determines whether the size of the buffer is greater than a quantity V_1, which the policy has determined represents viral infection, whereupon at step 844 it generates a virus alert. This may simply be a visual alert to a user of the workstation

10, or a message to the network administrator, or both, or even a trigger for automated action to shut the network down, as desired. At step 846, the routine determines whether the variable V_i is increasing above a given rate, and if it is, issues a further warning indicating the onset of viral infection at step 848, following which the routine ends.

A situation in which the second buffer management routine generates a viral infection warning can be seen in Figs. 6A-C. As mentioned previously, during time interval T3, a single Request A (which it will be recalled from the time interval T1 is to contact the mail server), and two Requests C are received. Because the dispatch record 614 for this time interval does not contain Request A, it adds the identity of host A to the virtual buffer, but not the identify of host C. At the end of the time interval T3 the virtual buffer therefore contains Request E (stored in the virtual buffer since time interval T2) and Request A. Since only one new request is transmitted per time window in accordance with policy, and since Request E has been in the virtual buffer since time interval T2, whereas Request A has just been added, Request E is deleted from the virtual buffer (a process with may be thought of as "virtual transmission"), so that at the start of time interval T4 the virtual buffer contains only Request A. This indicates that at this point in time, since startup of the entity on which the VPMS is running, only one more request has been transmitted than the policy allows. The first Request for connection in time interval T4 is Request B, which illustrates that over the course of three time intervals, during which only normal network traffic has been transmitted, connection has only been requested to five different destination hosts. However, Request B is nonetheless defined as new because it's not in the dispatch record 616 for time interval T4, and so the identity of host B is stored in the virtual buffer (this action being illustrated at the same point in the timeline in Fig. 6C). After receipt of request B, two groups of five virtually simultaneous requests are received: F-J, and K-O, and since these are also new, their identities are also added to the virtual buffer. Referring specifically to Fig. 6C during time interval T4, it can readily be seen that the virtual buffer has increased from a size of one, to 12, and in accordance with the policy, this is defined as viral infection, since in the present example a buffer size of greater than five generates this alert. Moreover, size the rate of change is positive and rapid (from 1 to 12 in a single time interval), this is indicative of the onset of infection. Thus the likelihood is that a

substantial number of the requests transmitted during the course of time interval T4 have been virally related.

In the event that a viral warning is generated, various further actions may then be taken, the majority of which are directed toward finding out more about the nature of any possible virus. Specifically the type of information sought may typically include: the destinations to which a virus has been propagated, where applicable the application programme or programmes which it uses to propagate itself, and the action and behaviour of the virus. The nature of the information which may obtained directly from the virtual buffer, or which may be deduced therefrom depends to an extent upon the nature of the data stored in the virtual buffer, and the operating system of the host concerned. For example in the case of one preferred embodiment in which the virtual buffer simply copies the socket, including payload, the destination host will be recorded in the buffer, and possibly, in the case where the virus copies itself to the socket as the outgoing payload, also the virus. Additionally, where the operating system records an identifier in the socket denoting the application programme requesting the socket, and an ability to map this process identifier to the requesting application programme after the socket has been closed (remembering that the virtual buffer contains a copy of the socket, while the actual socket is transient since it is used to implement the request to send data and is then deleted), then the application programme responsible for requesting data transmission can be identified. The use of the data in a socket is only one way in which to collect data relating to possible viral infection, and when using sockets, depending upon the extent of the data collected, the of copying of the sockets is likely to vary. For example, if, as referenced above, the fullest data (including e.g. copies of the payload) is to be retained, further copies of the sockets in the virtual buffer (stored for example in a manner which tags them to the copy of the socket in the virtual buffer) are preferably made over time as the contents of the socket changes over time. However, because two functional elements within the host may cause a change in the data in a socket (e.g. the writing of outgoing data to a socket by an application programme, and removal from the socket of outgoing data by the network stack), maintaining a complete record may nevertheless still be difficult simply from observing the contents of sockets.

In an alternative embodiment, the network stack additionally includes a layer 502, known as a packet logger, known *per se*. According to one embodiment, when a viral warning is generated as a result of the virtual buffer size (the virtual buffer this embodiment still being made of a single copy of a socket), the logger 502 is switched on, and makes copies of outgoing packets. These may be all outgoing packets, or packets identified by one or more particular destination IP address, the identity of which may for example be established from the copies of the sockets in the virtual buffer. By logging packets complete information may be stored relatively easily, since, for example even in the case of large payloads, the individual packets carrying various parts of the payload may easily be aggregated using the SEQ and ACK numbers. Further, if desired, the use of the logger enables incoming packets from designated IP addresses to be logged, which may provide valuable information in circumstances for example where a virus has a "hand-shake" action with another host (i.e. sends back a packet to its originating host from a destination host) as part of its propagation process (as is the case, for example with the Nimda worm).

The relatively early provision of warning of viral infection is potentially extremely beneficial, since in the case of many viruses the rate at which they can establish infection accelerates over time. For example, in the case of the code red virus, it has been established that over the course of the first 16 hours, 10,000 hosts were infected, but that in the subsequent 8 hours the virus infected a further 340,000 hosts. The early collection of data on viral infection can thus enable action to be taken, either within the hosts within which infection has been detected, and/or within other hosts, which can substantially reduce the extent of subsequent infection.

In the scenario illustrated in connection with Fig. 6, a single outbound request (Request A) to the VPMS, specifying a single destination host, namely the mail server, actually contains a plurality of email messages to different specified addressees. This outbound request may therefore be thought of as a carrier request for a plurality of sub-requests, here having the form of putative email messages intended for dispatch from the mail server to a list of addressees specified within the outbound carrier request (similarly, the mail server may be thought of as acting as a proxy destination host for the ultimate addressees specified in the outbound carrier request). In this situation, allowing transmission of the data packet constituting the message to

the mail server will in fact effectively allow the workstation to contact multiple other hosts within the network (i.e. the specified addressees) all of which may be new, even though, in accordance with the routine described in connection with Fig. 7, the outbound carrier request will only count as a single request which may not even be recognised as new if, as may be likely, the mail server is identified in the current dispatch record. In such a situation therefore, if the VPMS operates simply to record in the virtual buffer those new destination hosts to be contacted per time window on the basis only of those destination hosts which are ostensibly identified in the outbound request, the desired monitoring of viral propagation may be circumvented or reduced, because a single outbound request specifying the mail server does not necessarily represent only a single email subsequently propagating through the network after processing and forwarding by the mail server.

In a modification of the embodiment thus far described therefore, the VPMS includes within its routine a step of identifying the application programme by which an outbound request has been generated. Because certain applications programmes are more likely than others to use outbound carrier requests which invoke the use of a proxy (for example the above-mentioned instance of email, or the case of a web browser programme) it is possible in advance to specify criteria, based on the provenance of an outbound request, identifying those outbound requests likely to be carrier requests. If the packet is generated by one such specified application programme, then the VPMS invokes the use of the application protocol concerned to reveal the identities of the destination hosts specified in the sub-requests; here the eventual addressees for whom the email message is intended. Once the identities of the genuine or ultimate addressees have been obtained, there are several options for processing the request. In accordance with one alternative the identities of the destination hosts specified in the sub-request can be regulated in accordance with the same policy which applies to all other requests, and they can be matched against the host identities within the dispatch record in the manner previously described in the embodiment of Figs.

Since in the case for example of email, the use of outbound carrier requests to a host acting as a proxy for the ultimate addressees of the email messages is the norm, it is, in a modification, possible for different versions of VPMS to run simultaneously,

effectively operating in parallel with each other: one which applies to hosts specified in the outbound request (including carrier requests), and another which applies to hosts specified in any sub-requests identified by the email application programme. In such a situation, each VPMS will operate independently, using its own dispatch record, and implementing a policy for outbound requests tailored to the traffic it is set up to control, for example in the manner previously described and illustrated in connection with Figs. 6 and 7. The two policies may be the same (e.g. a dispatch record of 3 identities, a time window of constant duration T_n, and one new host per outbound request/sub-request), or different as desired.

The choice of the length of the time window, the number of identities retained in a dispatch record, and the number of new hosts to be allowed per time window are all dependent upon the likely "normal" performance of the network within which the VPMS is operating, and more particularly, the nature of the network traffic the VPMS is intended to control. Therefore, while a policy such as that illustrated in connection with Figs. 6 and 7 may be effective in monitoring the propagation of viruses through the network to a rate of infection of one new host per time interval, it may also be susceptible to false warnings caused by non virally-related, or "legitimate" network traffic whose characteristic behaviour differs substantially from the policy the VPMS is implementing. To ameliorate this difficulty, it is possible to provide a version of VMPS for each application programme from which network traffic emanates, with each VMPS implementing a policy tailored specifically to minimise the level of impediment to legitimate network traffic. Alternatively, in accordance with a further preferred embodiment of the present invention, an individual VPMS is provided in respect of each application protocol which the hosting entity supports, and requests are routed to appropriate VPMS on the basis of the port identified in outgoing requests from application software.

CLAIMS

1. A method of monitoring propagation of viruses within a network of hosts comprising the steps of:

establishing a record which is at least indicative of identities of hosts within the network to whom data has been sent by a first host ("destination hosts");

during a first time interval, comparing (a) identities of destination hosts identified in requests to send data from the first host and (b) identities of destination hosts identified in the record;

transmitting all requests to send data;

storing in a buffer data relating to requests which identify a destination host not in the record.

2. A method according to claim 1 wherein the record is established by monitoring identities of destination hosts to whom requests have been transmitted during a second time interval, which precedes the first time interval.

3. A method according to claim 2, wherein the record contains a predetermined maximum number of destination host identities, the maximum number being defined in accordance with a policy.

4. A method according to claim 3, wherein the policy additionally defines a maximum number of destination host identities not in the record, to whom requests may be legitimately transmitted in accordance with policy.

5. A method according to claim 4 further comprising the step, at the end of any given time interval, of deleting from the buffer data relating to requests transmitted during the given time interval in accordance with policy.

6. A method according to claim 5 further comprising the step, at the end of the given time interval, of updating the record to reflect identities of hosts identified in requests which are transmitted in accordance with policy during the given time interval.

7. A method according to claim 6 further comprising the step of updating the record to reflect the identity of the predetermined maximum number of destination host identities to whom data has most recently been sent in accordance with policy.

8. A method according to any one of the preceding claims, wherein the data stored in the buffer is a copy of a socket created to send data in accordance with a request.

9. A method according to claim 8 wherein the socket enables identification of at least one application programme at whose behest the socket is created.

10. A method of operating a first host within a network of a plurality of hosts comprising the steps of:

over the course of a first time interval, monitoring creation of sockets within the first host to identify destination hosts identified therein;

comparing identities of destination hosts monitored during the first time interval with destination host identities in a record; and

storing, in a buffer, data from all sockets which identify destination hosts not in the record.

11. A method according to claim 10 wherein the socket data stored in the buffer at least enables identification of the destination host identified therein.

12. A method according to claim 10 wherein the record identifies a maximum number of destination hosts, the maximum number being determined in accordance with a policy.

13. A method according to claim 12 wherein the record is established by monitoring creation of sockets during a time interval preceding the first time interval.

14. A method according to claim 12 wherein the policy additionally specifies a maximum number of sockets identifying a destination host not in the record to be legitimately created in any given time interval.

15. A method according to claim 14 wherein at the end of a time interval, socket data containing identities of destination hosts in respect of whom sockets have legitimately been created is deleted from the buffer.

16. A method according to claim 10 further comprising the step, in the event that the number of socket data items stored in the buffer exceeds a predetermined value, of storing outgoing packets from the first host.

17. A method according to claim 16 wherein packets having a designated destination IP address are stored.

18. A method according to claim 17 further comprising the step of establishing the predetermined IP address from the socket data stored in the buffer.

19. A method according to claim 10 further comprising the step, in the event that the number of socket data items stored in the buffer exceeds a predetermined value, of storing incoming packets to the first host.

20. A method according to claim 19 wherein packets having a designated source IP address are stored.

21. A method according to claim 20, further comprising the step of establishing the predetermined IP address from the socket data stored in the buffer.

Application No:	GB 0224396.2	**Examiner:**	Daniel Voisey
Claims searched:	1 to 21	**Date of search:**	12 March 2003

Patents Act 1977 : Search Report under Section 17

Documents considered to be relevant:

Category	Relevant to claims	Identity of document and passage or figure of particular relevance	
X	1	GB 2362076 A	(3COM) see particularly page 2 lines 11 to 29, page 4 lines 14 to 30, page 5 line 29 to page 6 line 17 and figure 2.
A		GB 2373130 A	(MESSAGELABS) see particularly page 1 line 28 to page 2 line 30.

Categories:

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention
&	Member of the same patent family	E	Patent document published on or after, but with priority date earlier than, the filing date of this application

Field of Search:

Search of GB, EP, WO & US patent documents classified in the following areas of the UKC[V]:

H4K, G4A

Worldwide search of patent documents classified in the following areas of the IPC[7]:

H04L, H04Q, G06F

The following online and other databases have been used in the preparation of this search report:

WPI, EPODOC, JAPIO, INSPEC