(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
3 January 2013 (03.01.2013)

WIPO | PCT

(10) International Publication Number
WO 2013/002980 A2

(51) International Patent Classification: Not classified

(21) International Application Number: PCT/US2012/041049

(22) International Filing Date: 6 June 2012 (06.06.2012)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
13/172,757 29 June 2011 (29.06.2011) US

(71) Applicant *(for all designated States except US)*: **MICROSOFT CORPORATION** [US/US]; One Microsoft Way, Redmond, WA 98052-6399 (US).

(72) Inventors: **TALPEY, Thomas, M.**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, WA 98052-6399 (US). **KRAMER, Gregory, Robert**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, WA 98052-6399 (US). **KRUSE, David, Matthew**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, WA 98052-6399 (US).

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

(84) **Designated States** *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

[Continued on next page]

(54) Title: TRANSPORTING OPERATIONS OF ARBITRARY SIZE OVER REMOTE DIRECT MEMORY ACCESS

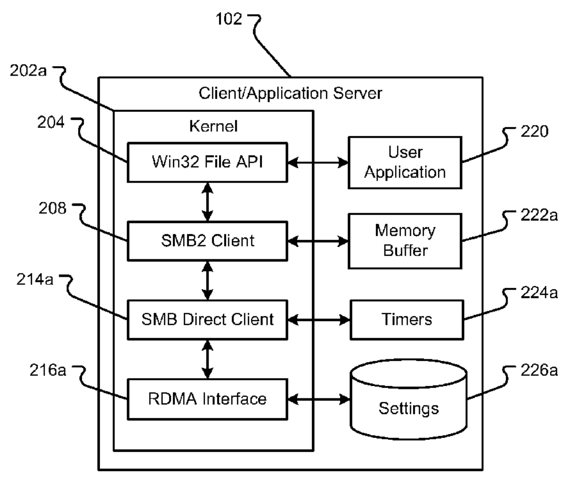


Fig. 2A

(57) Abstract: The embodiments described herein generally relate to a protocol for implementing data operations, e.g., a version of SMB, atop RDMA transports. In embodiments, systems and methods use the protocol definition, which specifies new messages for negotiating an RDMA connection and for transferring SMB2 data using the negotiated communication. A new protocol message may include new header information to determine message size, number of messages, and other information for sending the SMB2 data over RDMA. The header information is used to accommodate differences in message size requirements between RDMA and SMB2. The SMB Direct protocol allows SMB2 data to be fragmented into multiple individual RDMA messages that a receiver may then logically concatenate into a single SMB2 request or SMB2 response. The SMB Direct protocol also may allow SMB2 to transfer application data via efficient RDMA direct placement and to signal the application data's availability when the transfer is complete.

Published:

— *without international search report and to be republished upon receipt of that report (Rule 48.2(g))*

TRANSPORTING OPERATIONS OF ARBITRARY SIZE OVER REMOTE DIRECT MEMORY ACCESS

BACKGROUND

[0001] File access protocols, such as Server Message Block (SMB) or versions thereof, e.g., SMB2, may operate as application-layer network protocols mainly used to provide shared access to files and miscellaneous communications between nodes on a network. Historically, SMB or SMB2 operated atop transmission control protocol (TCP) transports and traditional network infrastructure. While SMB2 has been very successful as a protocol for general purpose remote file access, SMB2 has not been widely adopted for remote file access where high throughput and low latency file input/output is required.

[0002] Remote Direct Memory Access (RDMA) is a direct memory access from the memory of one computer into that of another computer without involving the operating system of the other computer. This direct transfer permits high-throughput, low-latency data transfers over a network, which is especially useful in performance-critical deployments. When an application performs an RDMA Read or Write request, the application data are delivered directly from a source memory buffer to a destination memory buffer using RDMA-capable network adapters, which do not involve the central processing unit (CPU) (also referred to simply as a processor) or operating system in the transfer. These RDMA transfers reduce latency and enable fast message transfer. Unfortunately, the benefits of RDMA have not been exploited by systems using SMB2 because SMB2 has not operated with RDMA.

[0003] Although specific problems have been addressed in this Background, this disclosure is not intended in any way to be limited to solving those specific problems.

SUMMARY

[0004] Embodiments generally relate to a protocol and processing to implement data operations, such as SMB2 operations (or other versions of SMB operations or file access protocol operations, for example), atop RDMA transports. In embodiments, the protocol definition specifies new messages for negotiation of an RDMA connection and for transferring SMB2 data, for example, using the negotiated connection. In an embodiment, a protocol for implementing SMB2 operations atop RDMA transports is the SMB Direct protocol. However, other embodiments provide for other SMB protocols, SMB protocol versions, or other data operation protocols without departing from the spirit and scope of the present disclosure. According to an embodiment, a new SMB Direct message may include new header information, which may include, but is not limited to, one or more of the following: CreditsRequested, CreditsGranted, Flags, Reserved, RemainingDataLength, DataOffset, and DataLength. The header information is used because RDMA transports support receiving messages of a size fixed only by the receiver, and SMB2 message size can vary widely from about a hundred bytes to very large messages over a million bytes. The SMB2 protocol is modified to recognize the existence of the RDMA capability, while the SMB Direct protocol adds a new layer to the networking stack to allow multiple individual RDMA messages to be logically concatenated into a single request or response to accommodate both the fixed size restrictions on RDMA messages and the indeterminate size requirements inherent to SMB2 messaging. The changes to the SMB2 protocol and the addition of the SMB Direct protocol allow for the direct transfer of data between memories of the peers. In embodiments, the SMB2 server may read from or write to a client's memory using RDMA to perform direct placement of data. The server performs an RDMA Write to the client to complete an SMB2 read and performs an RDMA Read to complete an SMB2 write. While the SMB Direct protocol allows for the direct transfer of SMB/SMB2 data between memories of peers, the SMB Direct protocol may be adapted to other protocols, according to embodiments. According to embodiments, the bi-directional, peer-to-peer nature of the SMB Direct protocol lends itself to numerous types of data transfer operations.

[0005] RDMA transports also restrict the number of messages which may be processed at any time, again with a value fixed only by the receiver. To conform to this requirement of RDMA, embodiments provide for the peers to exchange or assign "credits," which are numeric values, requested by and granted to each mutual peer, in the protocol header that specify the number of RDMA messages the sender may send to the receiver. Credits are dynamic and are managed independently by each peer. Rules for managing and making sufficient credits available to perform SMB2 exchanges are defined by the protocol in embodiments disclosed herein.

[0006] In embodiments, the provision of independent, bidirectional credits may permit each peer to send requests and responses without explicit negotiation or prior knowledge and agreement by the receiving peer. Sequenced sends, associated with RDMA, may permit the exchange of unexpectedly large messages without invoking errors in the RDMA processing and without resorting to less efficient negotiated transfers when such conditions arise.

[0007] Additional messages may be used for negotiation of the protocol version and other parameters, according to embodiments. A Negotiate Request message may include, for example, fields for CreditsRequested, Reserved, MinVersion, MaxVersion, OutboundSendSize, MaxInboundSendSize, etc. In turn, a Negotiate Response message sent in response to the request message may include, for example, fields for CreditsRequested, CreditsGranted, Version, Reserved, Status, OutboundSendSize, InboundSendSize, etc. These parameters support the negotiation of capabilities, end-to-end optimization of resources, and compatibility with future enhanced versions of the protocol.

[0008] This Summary is provided to introduce a selection of concepts in a simplified form that is further described below in the Detailed Description. This Summary is not intended to identify key or essential features of the claimed subject matter, nor is it intended to be used in any way as to limit the scope of the claimed subject matter.

BRIEF DESCRIPTION OF THE DRAWINGS

[0009] Embodiments of the present disclosure may be more readily described by reference to the accompanying drawings in which like numerals refer to like items.

[0010] FIG. 1 illustrates an example logical representation of an environment or system for exchanging SMB2 messages over RDMA, in accordance with embodiments of the present disclosure.

[0011] FIG. 2A depicts an example logical representation of a client system for sending SMB2 messages over RDMA, in accordance with embodiments of the present disclosure.

[0012] FIG. 2B illustrates a logical representation of a server system for receiving SMB2 messages over RDMA, in accordance with embodiments of the present disclosure.

[0013] FIGS. 3A-3E show logical representations of messages sent or received when exchanging data using SMB2 messages over RDMA, in accordance with embodiments of the present disclosure.

[0014] FIGS. 4A-4C depict a flow diagram illustrating the operational characteristics of a process for negotiating a communication using SMB2 over RDMA, in accordance with embodiments of the present disclosure.

[0015] FIGS. 5A-5D show a flow diagram illustrating the operational characteristics of a process for exchanging data using SMB2 over RDMA, in accordance with embodiments of the present disclosure.

[0016] FIG. 6 illustrates a flow diagram depicting the operational characteristics of a process for exchanging data using an RDMA direct data transfer, in accordance with embodiments of the present disclosure.

[0017] FIG. 7 depicts an example computing system upon which embodiments of the present disclosure may be implemented.

DETAILED DESCRIPTION

[0018] This disclosure will now more fully describe example embodiments with reference to the accompanying drawings, in which specific embodiments are shown. Other aspects may, however, be embodied in many different forms, and the inclusion of specific embodiments in this disclosure should not be construed as limiting such aspects to the embodiments set forth herein. Rather, the embodiments depicted in the drawings are included to provide a disclosure that is thorough and complete and which fully conveys the intended scope to those skilled in the art. Dashed lines may be used to show optional components or operations.

[0019] Embodiments generally relate to systems, methods, and protocols for exchanging SMB2 data over an RDMA connection. RDMA provides advantages in transporting data. For example, RDMA makes transfers from one memory to another device's or system's memory. These transfers do not involve the processor and, thus, reduce the overhead involved in transporting data. Further, without the involvement of a processor in transport management, RDMA transmits data with fewer clock cycles. Thus, RDMA provides a low-latency, high bandwidth connection.

[0020] In general, a version of SMB, such as SMB2, is an application-layer network protocol used to provide shared access to files and miscellaneous communications between nodes on a network. SMB and SMB2 are thus examples of file access operation protocols. SMB2 allows for the transfer of data in the exchange of messages between a client(s) and a server(s). The description of some embodiments herein refers to SMB2. However, in other embodiments, any version of SMB or other file access operation protocol may be used without departing from the spirit and scope of the present disclosure.

[0021] Due, generally, to differences in operation, SMB2 (or SMB) does not currently operate over RDMA but, instead, uses TCP to transport data. The present embodiments create a system and protocol for exchanging SMB2 messages over an RDMA-capable network protocol. First, a client and/or a server, which may be collectively referred to as peers, discover the capabilities or abilities of each other using a component of SMB2. In an embodiment, a request is sent by the client asking for information about the server.

The responding server may reply with information about the number of network interface cards (NICs) the server has, the internet protocol (IP) address for the NICs, the speed of the NICs, whether the NICs are RDMA capable, and/or possibly an IP address. In embodiments, the asking client may use this information to determine how to interface with the server.

[0022] If a server is RDMA capable, an RDMA-capable client may then negotiate an RDMA connection with that server, according to embodiments of the present disclosure. A new interface, referred to as the RDMA interface in embodiments, is an interface between a RDMA-capable network adapter and the other system components (including an SMB client). Further, a new module is added to the stack called an SMB Direct client/server. First, the SMB Direct client, possibly in response to the capabilities request, pre-posts a receive by reserving at least a portion of a memory buffer to receive a message from the other peer. The SMB Direct client may then send an SMB Direct Negotiate Request packet. An SMB Direct Negotiate Request begins a process of establishing an SMB Direct connection between the peers. According to embodiments, the SMB Direct Negotiate Request includes one or more fields that define how the RDMA connection will function.

[0023] In embodiments, part of the SMB Direct Negotiate Request is a request for "credits." Credits are an allotment of space in the receiving peer's memory buffer. As RDMA transfers directly from one memory to another memory, the receiving peer reserves space in a memory buffer where the sending peer may place transported data. The memory buffers are allotted in blocks (e.g., 1Kbyte blocks). In embodiments, the allotments are set in that the block size does not change after the SMB Direct connection has been established. In other embodiments, the allotments are not set, in which the block size may change. Where the allotments are set, any transfer to the memory buffer may not exceed the block size. To transfer SMB2 data having a size greater than the block size, the sender sends two or more packets that are stored into two or more allotments of the memory buffer. To reserve the memory allocations, the sending peer requests credits, where the number of credits requested may be governed by a local policy and is not necessarily affected by the composition of the message. Each credit represents a block of the memory buffer and, thus, represents a message the requester may send to the other peer.

[0024] In embodiments of the present disclosure, the server may send a response, e.g., an SMB Direct Negotiate Response, to the requester. This SMB Direct Negotiate

Response also includes various fields that define the RDMA connection. In response to the request, the SMB Direct Negotiate Response provides a number of credits to the client. Further, the SMB Direct Negotiate Response may also ask for credits that reserve an allocation in the client's memory buffer. The exchange of the SMB Direct Negotiate Request and SMB Direct Negotiate Response, in embodiments, establish the RDMA connection. Thereafter, SMB2 data may be exchanged over the connection.

[0025] According to embodiments, the exchange of SMB2 data over the RDMA connection includes the transport of at least one packet. If the SMB2 data being sent is smaller than the block size, only one packet, called an SMB Direct Data Transfer packet, may be sent. However, if SMB2 data to be sent is larger than the block size, two or more SMB Direct Data Transfer packets may be sent, according an embodiment. RDMA allows for the sequenced reception of packets. Thus, a second packet received in a communication will be placed directly after a first received packet. To utilize this advantage of RDMA, the SMB Direct Data Transfer packets include fields to announce the total amount of SMB2 data to be sent and how much data are left to be sent after the present packet. In this way, the SMB Direct peers may determine when an SMB transfer is completed, and the SMB2 data may be reassembled. The SMB Direct Protocol, therefore, provides in embodiments for quick transfers of SMB2 data with low latency and low overhead, while overcoming the issues associated with transferring SMB2 data over RDMA.

[0026] An example logical environment or system 100 for exchanging SMB2 data over an RDMA connection is shown in FIG. 1, according to embodiments disclosed herein. Connection peers (referred to also as client 102 and server 106) 102 and 106 may move SMB2 data across an RDMA connection between RDMA NICs (RNICs) 108a and 108b over network 104. A connection peer may be any computer system as described with respect to FIG. 7, for example. The connection peers are shown as a client/application server 102 and a file server 106 in FIG. 1. However, these peers are offered by way of example only. Any type of client(s) or server(s) may function as a connection peer in the embodiments. Thus, RDMA connections may be between multiple clients, multiple servers, a server farm(s), a server cluster(s), a message server(s), or between a client(s) and a server(s), for example. The client/application server 102 and file server 106 are offered as examples only for purposes of understanding the teachings of the embodiments disclosed herein.

[0027] While SMB2 data are moved over network 104, the network 104 may be as described with respect to FIG. 7, for example. Network 104, although shown as an individual single network, may be any type of network conventionally understood by those of ordinary skill in the art. In accordance with an embodiment, the network may be the global network (e.g., the Internet or World Wide Web, i.e., "Web" for short). It may also be a local area network, e.g., intranet, or a wide area network. In accordance with embodiments, communications over network 104 occur according to one or more standard packet-based formats, e.g., H.323, IP, Ethernet, and/or ATM.

[0028] Further, in embodiments, an RNIC 108(a and/or b) may be any network interface card, network adapter, and/or network interface controller that supports RDMA. There are several vendors that offer RNICs. iWARP or InfiniBand, for example, are network protocols that support RDMA. The RNICs may support RDMA, which allows direct transfers of data from memory 110a to memory 110b and vice versa. These transfers of data do not require or include the oversight of the processor 112a or 112b. Thus, the RDMA transfers are high-bandwidth, low latency, and low overhead, according to embodiments of the present disclosure. The processors 112 and memories 110 may be as described with respect to FIG. 7, for example.

[0029] While FIG. 1 illustrates a general environment for SMB2 data exchanges over RDMA, FIG. 2A depicts an example peer 102 for sending or receiving the SMB2 data over the RDMA connection, according to embodiments disclosed herein. In this example, the peer 102 is a client and/or application server. The various components of the client 102 may include software and/or hardware, according to embodiments of the present disclosure. However, for purposes of illustration, the components hereinafter will be described as software modules. In embodiments, the client 102 includes one or more of, but is not limited to, a kernel 202a, at least one user application 220, a memory buffer 222a, one or more timers 224a, and/or one or more settings 226a. In embodiments, a "kernel" is the core of an operating system, in which memory, files, and peripheral devices are managed, applications are triggered and launched, and system resources are allocated.

[0030] In embodiments, the kernel 202a may include one or more of, but is not limited to, a WIN32® File application programming interface (API) or equivalent 204, an SMB2 module (shown as an SMB2 Client 208) , an SMB Direct module (shown as an SMB Direct Client 214a), and a RDMA Interface 216a. The SMB Direct module and the RDMA Interface 216a are introduced to perform the methods and processes described herein. The modules, components, and/or interfaces will be described as follows.

[0031] In embodiments, the WIN32® File API 204 may be an interface between the kernel 202a and one or more user applications 220. In an example embodiment, the WIN32® File API 204 is a set of APIs available in the MICROSOFT WINDOWS® operating systems. Almost all WINDOWS® programs interact with the WINDOWS® API to perform functions. Embodiments of the WIN32® File API 204 provide access to the resources available to a WINDOWS® system, such as, for example, file systems, devices, and/or error handling. WIN32® File API 204 may provide access to functionality additional to the kernel, according to embodiments. In embodiments, WIN32® File API 204 additionally allows a system to perform actions on remote files that use underlying file access functions that, in turn, use various networking capabilities and allow for remote access.

[0032] In embodiments, the SMB2 Client 208 manages the communication between applications and the interfaces provided by the SMB2 Server 212b. Because devices may operate at speeds that may not match the operating system, the communication between the operating system and device drivers is primarily done through I/O request packets (IRPs). These packets are similar to network packets or WINDOWS® message packets, for example. The packets are passed from an operating system(s) to specific drivers and from one driver to another. In embodiments, the SMB2 Client 208 may redirect the I/O requests to network resources and compose SMB messages to conduct the communications over the network. The SMB2 Client 208 communicates the SMB packets to the SMB Direct Client 214a to exchange the SMB packets or data over RDMA. Similarly, the SMB2 Server 212b can also use IRPs to send file requests from the SMB2 Client 208 to the server's storage, according to an embodiment.

[0033] In embodiments, the SMB Direct Module 214 is an instance created from the SMB Direct network provider interface (NPI) in the kernel 202. The SMB Direct Module 214 exposes an API (referred to as the SMB Direct NPI) to the SMB2 client and SMB2 server modules. The SMB2 client/server modules use this SMB Direct NPI to make requests to send or receive data over an SMB Direct connection. The SMB Direct module 214 implements the SMB Direct Protocol and sits between the SMB2 client/server modules and the underlying RDMA interface, according to embodiments. The SMB Direct NPI enables the SMB Direct protocol. The SMB Direct NPI may create and destroy SMB Direct connections, send and receive data over SMB Direct connections, register/unregister memory, perform RDMA Read/Write data operations from/to a peer over a SMB Direct connection, receive notifications when an SMB Direct connection is disconnected by the peer, marshal/unmarshal SMB2 packets for transmission across a SMB Direct connection, among other operations. To accomplish these tasks, an SMB Direct Module 214 is created that may manage the sending and retrieving of the SMB2 data from the memory buffers 222 as stored by the RDMA protocol. Thus, the SMB Direct Module 214 converts data from simply SMB2 data into RDMA and back from RDMA to SMB2. The SMB Direct Module 214 communicates with another new module, the RDMA Interface 216 to execute the operations in embodiments of the present disclosure.

[0034] According to embodiments, the SMB Direct Module 214 performs several functions. An SMB DirectReceiveEvent callback function notifies SMB2 Client 208 or SMB2 Server 212b that a message has been received on a SMB Direct endpoint. An SMB DirectDisconnectEvent event callback function notifies SMB2 Client 208 or SMB2 Server 212b that a connection on an endpoint has been disconnected. An SMB DirectAcceptEvent event callback function notifies the SMB2 Server 212b that an incoming connection on a listening endpoint has been accepted. An SMB DirectListen function creates a listener endpoint that listens for incoming connections on a given local address. An SMB DirectCloseEndpoint function closes an endpoint and frees any associated resources. An SMB DirectConnect function connects an endpoint to a remote SMB Direct transport address. An SMB DirectDisconnect function disconnects an endpoint from a remote SMB Direct transport address. An SMB DirectSend function sends a buffer of data to a remote SMB Direct peer. An SMB DirectRegisterMemory function allows the SMB2 Client 208 to register memory buffers for use in RDMA Read/Write operations. An SMB DirectUnregisterMemory function unregisters memory buffers that were previously registered via the SMB DirectRegisterMemory API. An SMB DirectRdmaRead function causes RDMA to read data directly from the memory of the remote peer to which the endpoint is connected. An SMB DirectRdmaWrite function causes RDMA to write data directly into the memory of the remote peer to which the endpoint is connected. These functions and the operation thereof will be explained in conjunction with FIG. 6.

[0035] In an embodiment, the RDMA Interface 216 is a new interface to interface with the vendor-specific RDMA functionality of the RNICs. The RDMA Interface 216 can give access to the functions of the RDMA device. The functions of the RDMA device can include listening to a port to receive SMB Direct packets and provide the SMB2 data to the SMB Direct Module 214. In an embodiment, the RDMA device may include a kernel mode RDMA module to manage communications over the RDMA connection. Further, the RDMA device may include an RDMA access layer and Extensions to access and listen on a port sending RDMA messages. A proxy driver may interface with the hardware driver of the RNIC, according to embodiments.

[0036] In embodiments, the user application 220 may be any software executed by a processor for a user or other software. Examples of user applications 220 include web browsers, email, etc. These user applications 220 interface with the kernel 202a to send and receive data, especially from remote storage locations, such as the file server 106.

[0037] In an embodiment, a memory buffer 222 may be any type of memory, as described with respect to FIG. 7. The memory buffer 222 may be used to receive SMB Direct messages and/or SMB2 data carried in the messages and may be used to stage outgoing SMB Direct messages before transmitting those messages. Thus, the memory buffer 222 may be partitioned into blocks as described hereinafter, according to embodiments.

[0038] The timers 224, in embodiments, are a set of clocks that may count down from a predetermined time to zero. Thus, the timers 224 represent stored data and a clock function executed by a processor. Expiration of a timer 224 may trigger one or more functions in the RDMA Interface or with the SMB Direct Module 214. In other embodiments, the timers may count from zero to a threshold or perform some other type of counting. Some of the timers may include a SendCreditGrantTimer, which is the timer that is started when SendCreditCount reaches zero and operates in the timers section 224. The remotely connected peer has until this timer expires to grant additional Send Credits. The SendCreditGrantTimer can also regulate the amount of time that the client/server waits for the peer to grant it additional Send Credits. When the client/server finds that it cannot send a packet to the peer because the value of SendCreditsCount is zero, then the client/server sets a timer that will expire in a predetermined number of seconds, according to embodiments. If the timer expires before a Send Credit becomes available, the client/server disconnects the connection. In an embodiment, an Idle Connection Timer regulates the amount of time that the client/server waits for the peer to send a packet. If no packets have been received from the peer in the last predetermined number of seconds, the client/server sends a keep alive request to the peer and sets a KeepAliveResponse timer. If no response is received before the expiration of the KeepAliveResponse timer, then the connection is disconnected, according to embodiments of the present disclosure.

[0039] A KEEPALIVE_REQUESTED flag may also be set for any SMB Direct Data transfers over the SMB Direct Connection, according to embodiments of the present disclosure. In an embodiment, the KEEPALIVE_REQUESTED flag is a request for the receiving peer to respond to the sender as soon as possible so that the sender knows that the receiver is still connected and responsive. One or more systems may try to time out the connection without a message exchange. Thus, a message with the KEEPALIVE_REQUESTED flag can maintain the connection. In alternative embodiments, a message with the KEEPALIVE_REQUESTED flag set can be used to request or receive credits.

[0040] In an embodiment, a settings 226 store stores and retrieves data relevant to exchanging SMB messages over RDMA. The settings 226 may be stored in any type of memory or storage devices as described with respect to FIG. 7. As an example, the settings may include how many seconds after being set does the Credit Replenishment Timer expire, what is the maximum sized SMB Direct Data Transfer packet that the peer is willing to receive from another peer, or what is the limit on the number of send credits the peer will grant to another peer, etc.

[0041] The SMB_DIRECT_ENDPOINT Structure may be an opaque structure that represents a SMB Direct endpoint, in embodiments. An SMB Direct endpoint is analogous in function to a network socket, for example. SMB2 Client 208 or SMB2 Server 212b may not access the members of this structure directly but through the SMB Direct Module 214, according to an embodiment. The SMB_DIRECT_ENDPOINT Structure may include data for several operations. MwReleaseList is a list of memory windows that may be released back to the remotely connected peer. These memory windows are associated with RDMA Read/Write operations that have completed. ReceiveCreditCount is the number of Receive Credits that are to be granted to the remotely connected peer. The endpoint's host may have at least this number of receives pending on the endpoint. SendCreditCount is the number of Send Credits that the endpoint's host currently has. The remotely connected peer may have at least this number of receives pending on their endpoint. PendingRdmaReadCount is the number of RDMA Read operations that have been initiated but not yet completed on this endpoint. PendingRdmaReadLimit is the maximum number of RDMA Read operations that may be simultaneously pending on the endpoint. DeferredInitiatorOpQueue is a queue of initiator operations that are deferred because the endpoint resources to issue them are not currently available, in embodiments. NdkQp is the queue pair object that represents the receive and initiator request queues. NdkReceiveCq is the NDK receive completion queue. Receive request completions are queued to this queue. NdkReceiveQueueCapacity is the maximum number of Receive requests that may be simultaneously pending on the endpoint. NdkInitiatorCq is the NDK initiator completion queue. Send, Bind, Fast-Register, Read, Write, and Invalidate request completions are queued to this queue, for example. In embodiments, NdkInitiatorQueueCapacity is the maximum number of Send, Bind, Fast-Register, Read, Write, and Invalidate requests that may be simultaneously pending on the endpoint.

[0042] FIG. 2B depicts an example peer 106 for sending or receiving the SMB2 data over the RDMA connection, in accordance with embodiments disclosed herein. In this example, the peer 106 is a file server. The various components of the file server 106 may include software and/or hardware. While embodiments describe the components as software modules, other embodiments provide for other types of modules. In embodiments, the file server 106 includes one or more of, but is not limited to, a kernel 202b, an NTFS 232, a memory buffer 222b, one or more timers 224b, and/or one or more settings 226b. Some of these functions are the same or similar to those described in FIG. 2A.

[0043] According to embodiments, the kernel 202b (and/or as described with respect to FIG. 2A) may include one or more of, but is not limited to, an input/output (I/O) manager 206b, an SMB2 Server 212b, a SMB Direct Server 214b, and an RDMA Interface 216b. Some of these functions are the same or similar to those described in FIG. 2A.

[0044] In embodiments, SMB2 Server 212b is a driver that implements the server side of the SMB2 protocol for MICROSOFT® servers. Other embodiments provide for other types of servers. SMB2 Server 212b may start and exchange data using SMB and supply the data or receive the data from the I/O Manager 206b. In embodiments, SMB2 Server 212b sends or receives the SMB2 data over a network connection. SMB2 Server 212b functions to communicate over networks for the server. Thus, SMB2 Server 212b provides the SMB2 data for transmission or receives the SMB2 data from a network transmission, according to embodiments.

[0045] In embodiments, the New Technology File System (NTFS) 232 may be a standard file system. NTFS includes support for metadata and the use of advanced data structures to improve performance, reliability, and disk space utilization, plus additional extensions such as security access control lists (ACL) and file system journaling. NTFS functions to organize and store file data for one or more clients. This file data may be provided to the client over communications with the client, such as SMB2 data transfers over RDMA.

[0046] According to embodiments, SMB Direct creates an RDMA connection to exchange SMB2 data. The protocol creates the RDMA connection through a negotiation process. After the peers 102 and 106 negotiate the RDMA connection, either peer 102 or 106 may send SMB2 data over the RDMA connection.

[0047] According to embodiments, a first SMB Direct Negotiate Request packet 300 is shown in FIG. 3A. In turn, the associated SMB Direct Negotiate Response packet 336 is shown in FIG. 3B, in embodiments. Further, the SMB Direct Data Transfer packet 338 used to transfer data is shown in FIG. 3C, according to embodiments. Each message or packet may be created, transmitted, stored, and/or received, for example. The packets or messages may each include portions or fields that store different data, in embodiments.

[0048] Turning to FIG. 3A, an SMB Direct Negotiate Request packet 300 is shown, according to embodiments of the present disclosure. The SMB Direct Negotiate Request packet 300 may include one or more of, but is not limited to, the following fields, for example: MinVersion 302, MaxVersion 304, Reserved 306, CreditsRequested 308, PreferredSendSize 310, MaxReceiveSize 312, and/or MaxSMB2MessageSize 314. The SMB Direct Negotiate Request packet 300 may include additional or fewer fields than those shown in FIG. 3A, as represented by ellipses 316.

[0049] In an embodiment, the MinVersion field 302 may include a value for the lowest SMB Direct Protocol version that the client/requester 102 supports. The MaxVersion field 304 may store the highest SMB Direct Protocol version that the client/requester 102 supports. This value may be equal to or greater than the value of the MinVersion field 302. In embodiments, the client/requester supports all of the protocol versions (inclusive) in the range between the value in the MinVersion field 302 and the value in the MaxVersion field 304. The Reserved field 306 is simply a reserved field for future unknown requirements and is not used by the client, according to an embodiment.

[0050] The CreditsRequested field 308, according to embodiments, comprises a value for the number of send credits that the client/requester 102 is requesting from the server/receiver 106. In embodiments, the value in the CreditsRequested field 308 is greater than zero (0) to ensure that the SMB2 data may be sent in a subsequent message. However, the value in the CreditsRequested field 308 may be any number and may be set based on an average usage, according to embodiments. In alternative embodiments, the value in the CreditsRequested field 308 may be based at least on the size of the SMB message to be sent. In still further embodiments, other or additional factors may be considered in requesting credits. Also, large messages may be transferred using RDMA, in embodiments.

[0051] The PreferredSendSize field 310 may include the size (possibly in bytes) of the largest SMB Direct Data Transfer packet 338 that the client/requester 102 wishes to be able to transmit to the server/receiver 106. The MaxReceiveSize field 312 includes a size (possibly in bytes) of the largest SMB Direct Data Transfer message that the client/requester 102 will accept from the server/receiver 106. This value may be equal to the block or predetermined allocation of the memory buffer 222a that the client 102 set in embodiments. Thus, no single SMB2 data transfer will exceed the memory allocation. In embodiments, this value is greater than or equal to threshold 128, which is at least the size of the SMB Direct header in the data transfer packet and a small SMB2 message. The MaxSmb2MessageSize field 314 may include a size (possibly in bytes) of the largest SMB2 Protocol message that the client/requester will accept from the server/receiver 106, according to an embodiment. This value is predetermined and set by the client 102. In embodiments, the value in the MaxSmb2MessageSize field 314 may not be greater than the total size of the memory buffer 222a. In this way, no SMB message may overflow the memory buffer 222a. However, the value of the MaxSmb2MessageSize field 314 may be less than the amount of memory in the memory buffer 222a, as determined by a user, according to an embodiment.

[0052] Turning to FIG. 3B, an SMB Direct Negotiate Response packet 336 is shown in accordance with embodiments. The SMB Direct Negotiate Response packet 336 may be sent by a peer 106 to complete the negotiation for the RDMA connection. The SMB Direct Negotiate Response packet 336 may include one or more of, but is not limited to, the following fields, for example: MinVersion 302b, MaxVersion 304b, PreferredVersion 318, Reserved 306b, CreditsRequested 308b, CreditsGranted 320a, Status 322, PreferredSendSize 310b, MaxReceiveSize 312b, and/or MaxSMB2MessageSize 314b. The SMB Direct Negotiate Response packet 336 may include additional or fewer fields than those shown in FIG. 3B, as represented by ellipses 324.

[0053] According to embodiments, the MinVersion field 302b may include a value for the lowest SMB Direct Protocol version that the server/receiver 106 supports. The MaxVersion field 304b may store the highest SMB Direct Protocol version that the server/receiver 106 supports. This value may be equal to or greater than the value of the MinVersion field 302b. In embodiments, the server/receiver 106 supports all of the protocol versions (inclusive) in the range between the value in the MinVersion field 302b and the value in the MaxVersion field 304b. The PreferredVersion field 318 stores a value for the common SMB Direct Protocol version. The value of the PreferredVersion field 318, in embodiments, is within the range specified by MinVersion field 302a and MaxVersion field 304a of the client's SMB Direct Negotiate Request packet 300. In further embodiments, the value of the PreferredVersion field 318 is between the range specified by MinVersion field 302b and the MaxVersion field 304b of the server's SMB Direct Negotiate Response packet 336. The Reserved field 306b is simply a reserved field for future unknown requirements and is not used by the server, according to embodiments.

[0054] The CreditsRequested field 308b, according to embodiments, comprises a value for the number of send credits that the server/receiver 106 is requesting from the client/requester 102. In embodiments, the value in the CreditsRequested field 308b is greater than zero (0) to ensure that the SMB2 data may be sent in a subsequent message. However, the value in the CreditsRequested field 308b may be any number and may be set based on an average usage. In alternative embodiments, the value in the CreditsRequested field 308b is based on the size of the SMB message to be sent or may be based on other factors. Large messages may be sent when one credit is requested, according to embodiments. The CreditsGranted field 320a includes the number of credits granted from the server/receiver 106 to the client/requester 102. In embodiments, the value of the CreditsGranted field 320a is greater than zero (0) to allow the client/requester 102 to send the next SMB Direct message.

[0055] In embodiments, a status field 322 includes at least one flag or value. In an embodiment, the status field 322 includes one or two values, either status success or status not supported. The success flag designates that the server/receiver 106 has accepted the client/requester's SMB Direct Negotiate Request packet 300. The not supported flag designates that the server/receiver 106 has rejected the client/requester's 102 SMB Direct Negotiate Request packet 300.

[0056] According to embodiments, the PreferredSendSize field 310b may include the size (possibly in bytes) of the largest SMB Direct Data Transfer packet 338 that the server/receiver 106 wishes to be able to transmit to the client/requester 102. The PreferredSendSize field 310b is, in embodiments, smaller than or equal to the MaxReceiveSize field 312a in the SMB Direct Negotiate Request packet 300. Thus, the packet size sent by the server/receiver 106 may not be larger than the memory buffer 222a allotment of the client/requester 102.

[0057] In embodiments, the MaxReceiveSize field 312b includes a size (possibly in bytes) of the largest SMB Direct Data Transfer message that the server/receiver 106 will accept from the client/requester 102. This value may be equal to the block or predetermined allocation of the memory buffer 222b that the server/receiver 106 set. Thus, in embodiments, no SMB2 data transfer will exceed the memory allocation. In embodiments, this value is greater than or equal to the threshold 128, which is at least the size of the SMB Direct header in the data transfer packet. The MaxSmb2MessageSize field 314b may include a size (possibly in bytes) of the largest SMB2 Protocol message that the server/receiver 106 will accept from the client/requester 102. This value is predetermined and set by the server/receiver 106. In embodiments, the value in the MaxSmb2MessageSize field 314b may not be greater than the total size of the memory buffer 222b. In this way, no SMB message may overflow the memory buffer 222b. However, the value of the MaxSmb2MessageSize field 314b may be less than the amount of memory in the memory buffer 222b, as determined by a user, according to an embodiment.

[0058] An SMB Direct Data Transfer packet 338 is shown in FIG. 3C, in accordance with embodiments disclosed herein. The SMB Direct Data Transfer packet 338 may be sent to transfer SMB2 data across the RDMA connection established during the negotiation. Either the client/requester 102 or the server/receiver 106 may send or receive the SMB2 data. As such, both the client/requester 102 and the server/receiver 106 are referred to generally as the receiver and sender. The SMB Direct Data Transfer packet 338 may include one or more of, but is not limited to, the following fields, for example: CreditsRequested 308c, CreditsGranted 320b, Reserved 306c, RemainingDataLength 326, DataOffset 328, DataLength 330, and/or SMB2 Data 332. The SMB Direct Data Transfer packet 338 may include additional or fewer fields than those shown in FIG. 3C, as represented by ellipses 334, according to embodiments.

[0059] In embodiments, the CreditsRequested field 308c comprises a value for the number of send credits that the sender is requesting from the receiver. In embodiments, the value in the CreditsRequested field 308c is greater than zero (0) to ensure that the SMB2 data may be sent in a subsequent message. However, the value in the CreditsRequested field 308c may be any number and may be set based on predicted future usage, i.e., how many more packets are to complete the transfer of the SMB2 data. The CreditsGranted field 320b includes the number of credits granted from the sender to the receiver. In embodiments, the value of the CreditsGranted field 320b can be zero because the peer is not obligated to honor a peer's request for credits. However, the peer can provide credits in the CreditsGranted field 320b to allow the client/requester 102 to send the next SMB Direct message. The Reserved field 306c is simply a reserved field for future unknown requirements and is not used by the client, according to embodiments disclosed herein.

[0060] In embodiments, the RemainingDataLength field 326 may include a number of bytes of a fragmented SMB2 message that the receiver has yet to receive. Thus, any value other than zero in this field indicates to the receiver that another SMB Direct Data Transfer packet 338 will be sent with more data. If the SMB Direct Data Transfer packet 338 carries a complete SMB2 message or the last of two or more SMB Direct Data Transfer packets 338 that carry fragmented SMB2 data, then the value in the RemainingDataLength field 326 is zero (0), according to embodiments. RDMA is capable of sending messages sequentially. As such, reassembling messages is simplified because sequential messages are reassembled in strict order of receipt. Thus, the embodiments forgo the need for a message identifier in the header or the use of other more complicated reassembly techniques, for example.

[0061] In an embodiment, the DataOffset field 328 includes a value for the offset, in bytes, from the beginning of the SMB Direct Data Transfer packet 338 to the first 8-byte aligned byte of the encapsulated SMB2 Protocol message. In embodiments, if the value of the DataLength field 330 is zero, then the DataOffset field 328 is also set to zero. If the value of DataLength is not zero, then the DataOffset field 328 is some value and may be greater than or equal to 24 bytes, which is the size of the other fields in the header, according to an embodiment. The DataLength field 330 may include the size, in bytes, of the encapsulated SMB2 Protocol message in the SMB2 Data field 332. In embodiments, if the SMB Direct Data Transfer packet 338 does not encapsulate an SMB2 Protocol message, then the value of the DataLength field 330 is set to zero. The SMB2 Data field includes any SMB2 Protocol message, according to embodiments.

[0062] An SMB2 Read/Write Request 340 is shown in FIG. 3D, in accordance with embodiments disclosed herein. The SMB2 Read/Write Request 340 may be encapsulated in the SMB Direct Data Transfer packet 338 to perform a direct RDMA read/write. A direct RDMA read/write sends only unencoded application data. The SMB2 Read/Write Request 340 may be a SMB2 Read Request if the server 106 is to perform an RDMA write to the client 102 and a SMB2 Write Request if the server 106 is to perform an RDMA read from the client 102, according to embodiments of the present disclosure. Regardless, the fields in the SMB2 Read/Write Request 340 are similar. It should be noted that the server 106 performs the RDMA transfer that the client 102 requests. The SMB2 Read/Write Request 340 may include one or more of, but is not limited to, the following fields, for example: Channel 342, ChannelInfoOffset 346, and ChannelInfoLength 348. In an embodiment, the fields may include the steering information for completing the RDMA data transfer. The Reserved field 344 is simply a reserved field for future unknown requirements and is not used, according to an embodiment. The SMB2 Read/Write Request 340 may include additional or fewer fields than those shown in FIG. 3D, as represented by ellipses 351. The fields shown are offered for purposes of illustration and are not intended to be limiting.

[0063] In embodiments, the Channel field 342 comprises a value for the channel where the data are to be found. The RDMA connection may include several channels. The channel information may include information that the peer provides to the RDMA device to accomplish the transfer, according to embodiments. For example, the channel information may contain one or more tokens, offsets, and lengths of memory segments, along with other RDMA-specific information, as shown in FIG. 3E.

[0064] In an embodiment, the ChannelInfoOffset field 346 and the ChannelInfoLength field 348 are pointers to the offset, token, and length information in the data packet. The pointers give the location in the SMB2 read or write request where the information may be located. In embodiments, the Flags 350 include any information to control or change the behavior of the RDMA direct data transfer.

[0065] Turning to FIG. 3E, an RDMA channel descriptor 352 is shown, in accordance with embodiments disclosed herein. The RDMA channel descriptor 352 may be encapsulated in the SMB Direct request, typically in the channel field 342. The RDMA channel descriptor 352 may include one or more of, but is not limited to, the following fields, for example: Offset 354, Token 356, and Length 358. In an embodiment, the fields are the steering information for completing the RDMA data transfer. The RDMA channel descriptor 352 may include additional or fewer fields than those shown in FIG. 3E, as represented by ellipses 360. The fields shown are offered for purposes of illustration. To complete a direct data transfer, RDMA is directed to the memory with the data that is to be read or written and the length of the data based on the information in the channel field 342. As the direct data transfer sends only application data, this information helps ensure a proper transfer.

[0066] According to an embodiment, the Offset 354 is the value in bytes or bits where the data begins. The Offset 354 may be measured from a memory block starting address, which may be located using information provided in the Token 356. The Length 358 is the value in bits or bytes of how long the data segment is. This steering information guides the direct data transfer, for example.

[0067] The interactions of the various software functional modules depicted in FIGS. 2A and 2B are further illustrated in the operational steps 400 depicted in FIGS. 4A-4C for negotiating an RDMA connection, in accordance with an embodiment disclosed herein. FIG. 4A shows the representation 400A of the transfers of packets 408 and 424 between a client 102 and a server 106, according to embodiments of the present disclosure. FIG. 4B is in the perspective of the client 102, while FIG. 4C is in the perspective of the server 106, according to embodiments of the present disclosure. It should be noted that the process 400 is described as the client requesting the RDMA connection and the server 106 responding. However, the reverse could be true where the server 106 requests and the client responds, according to embodiments. Further, the method may be conducted between multiple clients and/or between multiple servers, for example.

[0068] Turning to FIG. 4B, example operational steps 400B for negotiating a connection using SMB2 over RDMA are shown in accordance with an embodiment. Process 400B is initiated at START operation 402B, and process 400B proceeds to the client pre-posting a receive 404. In embodiments, the RDMA Interface 216a automatically detects and leverages the RNICs 108a/108b to see that they are available and properly configured. Next, in an embodiment, the SMB Direct NPI is exposed by the SMB Direct module 214. The SMB Direct Client 214a may open a handle to the RDMA Adapter to form an RDMA connection for the SMB2 Client 208. In an embodiment, the SMB Direct Client 214a then requests an RDMA Connection through the RDMA Interface 216a. The SMB Direct Client 214a may also create the allocations in the memory buffer 222a and determine what to request for initial send credits and what will be allowed for receive credits. The allocations in the memory buffer 222a may be of a predetermined size, in embodiments. The total amount of memory in the memory buffer 222a may also be determined, according to embodiments.

[0069] Returning to FIG. 4B, the SMB Direct Client 214a may then construct 406 the SMB Direct Negotiate Request packet 300 to send to the RDMA Interface 216a. The SMB Direct Negotiate Request packet 300 is as described in FIG. 3A, for example. As such, the SMB Direct Client 214a sets the fields in the SMB Direct Negotiate Request packet 300. Thus, the SMB Direct Client 214a determines the minimum and maximum versions of SMB Direct that the client 102 may support. The SMB Direct Client 214a determines the number of credits to request. The number of credits to request may be based on known future SMB messages that are to be sent, on predicted SMB traffic that may occur in the future, on historical use of SMB, or by some other method, in embodiments. The send size is determined by internal functions and speed considerations. The maximum receive size may be pegged to the size of the memory buffer allotment. Finally, a maximum SMB message size is determined and set (generally, the maximum SMB message size is large enough to allow a peer to send large SMB2 packets but not so large as to use all the memory). This collected information is entered into the SMB Direct Negotiate Request packet 300, according to embodiments of the present disclosure.

[0070] Next, the SMB Direct Client 214a sends 408B the SMB Direct Negotiate Request packet 300. In an embodiment, the SMB Direct Client 214a requests the RDMA Interface 216a to send the SMB Direct Negotiate Request packet 300 by a send operation. The RDMA Interface 216a communicates with the RDMA Interface 216b and sends the SMB Direct Negotiate Request packet 300 to the server 106. With the sending of the SMB Direct Negotiate Request packet 300, the SMB Direct Client 214a begins a negotiation request expiration timer 410. The negotiation request expiration timer may have a predetermined value and counts down from the value to zero. The SMB Direct Client 214a then waits for a response to the SMB Direct Negotiate Request packet 300. If it is determined 411 that the negotiation request expiration timer expires before receiving a response to the SMB Direct Negotiate Request packet 300, process 400B proceeds NO to END operation 432B where the connection is dropped.

[0071] Turning to FIG. 4C at start operation 402C, the server 106 anticipates the SMB Direct Negotiate Request packet 300 and pre-posts a receive 412, in embodiments. The SMB2 Server 212b may listen for activity and automatically discover that RDMA is available and bound on a port of the RNIC 108b, according to embodiments. SMB2 Server 212b opens a listener endpoint and determines that RDMA is available on the RNIC 108b. Next, SMB2 Server 212b then begins communications with the SMB Direct Server 214b. The SMB Direct Server 214b may open an RDMA connection and receive an RDMA Adapter Handle. The SMB Direct Server 214b then requests an RDMA Connection through the RDMA Interface 216b. The SMB Direct Server 214b may then accept the RDMA Connection, thereby informing the client 102 that the connection request succeeded. This success indication from the RDMA interface 216a triggers the client to send the negotiate request.

[0072] The SMB Direct Server 214b may also create the allocations in the memory buffer 222b and determine what to request for initial send credits and what will be allowed for receive credits, according to embodiments. The allocations in the memory buffer 222b may be of a predetermined size. The total amount of memory in the memory buffer 222b may also be determined, according to embodiments of the present disclosure.

[0073] Returning to FIG. 4C, the SMB Direct Server 214b may then start a negotiation request expiration timer 414. The negotiation request expiration timer at the server 106 may have a predetermined value and counts down from the value to zero, in embodiments. The SMB Direct Server 214b then waits to receive the SMB Direct Negotiate Request packet 300. While waiting, the SMB Direct Server 214b monitors the RDMA connection and determines 416 if the negotiation request expiration timer expires before receiving the SMB Direct Negotiate Request packet 300. If the negotiation request timer expires, process 400C proceeds NO to END operation 432C where the connection is dropped.

[0074] However, if the SMB Direct Negotiate Request packet 300 arrives at the RNIC 108b and the RDMA Interface 216b before the expiration of the negotiation request expiration timer, process 400C proceeds YES to receiving and validating 408C, by the RDMA Interface 216b, the SMB Direct Negotiate Request packet 300 and placing the packet in the memory buffer 222b. Next, the SMB Direct Server 214b is informed of the data and determines if the SMB Direct Negotiate Request packet 300 is valid 418.

[0075] In embodiments, to validate the SMB Direct Negotiate Request packet 300, the SMB Direct Server 214b reads the data from the SMB Direct Negotiate Request packet 300 and determines the following, for example: if the value of the MaxVersion field 304 is less than the value of the MinVersion field 302 or if no value in the range is supported; if the value of the CreditsRequested field 308 is zero; if the value of the MaxReceiveSize field 312 is less than a predetermined threshold (e.g., 128 bytes); or, if the value of the MaxSmb2MessageSize field 314 is less than a predetermined threshold. If any of the above are true, then the SMB Direct Negotiate Request packet 300 is not valid, and process 400C proceeds NO to END operation 432C, where SMB Direct Server 214b disconnects. If all of the checks are not true, then process 400C proceeds YES to process SMB Direct Response 420, in which the SMB Direct Negotiate Request packet 300 is sent to the SMB Direct Client 214a by the SMB Direct Server 214b.

[0076] In processing the SMB Direct Negotiate Request packet 300, the SMB Direct Server 214b sets the setting ProtocolVersion to equal the maximum protocol version shared by the client 102 and the server 106, in embodiments. The SMB Direct Server 214b also determines how many credits to grant. The number of credits to grant depends, in embodiments, on the available space in the memory buffer 222b or other factors. After making these determinations, the SMB Direct Server 214b generates 422 the SMB Direct Negotiate Response packet 336. In an embodiment, the SMB Direct Server 214b sets the fields in the SMB Direct Negotiate Response packet 336. For example, the PreferredVersion field is set to the value in the setting ProtocolVersion. The CreditsGranted320a is set to the number of credits determined by the SMB Direct Server 214b, according to an embodiment. The other fields are filled in a similar fashion to the SMB Direct Negotiate Request packet 300. The SMB Direct Negotiate Response packet 336 is then sent 424C to the client. In embodiments, the SMB Direct Server 214b sends the SMB Direct Negotiate Response packet 336 to the RDMA Interface 216b to send.

[0077] Returning to FIG. 4B, the SMB Direct client 214a may, according to embodiments, determine 411 if the response was received before expiration of the negotiate expiration timer. If the response was received before expiration of the timer, process 400B proceeds YES to validate 424B the SMB Direct Negotiate Response packet 336. The RDMA Interface 216a receives the message into the memory buffer 222a. The SMB Direct Client 214a reads the SMB Direct Negotiate Response packet 336 and then validates the message. To validate the SMB Direct Negotiate Response packet 336, the SMB Direct Client 214a determines 428 the following, for example: if the Status field 322 is not STATUS_SUCCESS; if the PreferredVersion field 318 does not contain a value that is within the range specified by the MinVersion field 302 and MaxVersion field 304 of the client's SMB Direct Negotiate Request packet 300; if the CreditsRequested field 308b is zero; if the CreditsGranted field 320a is zero; if the PreferredSendSize field 310b is greater than the value specified by the MaxReceiveSize field 312 of the client's SMB Direct Negotiate Request packet 330; or if the MaxReceiveSize field 312b is less than a predetermined threshold (e.g., 128 bytes), according to embodiments of the present disclosure. If any of the above are true, then the SMB Direct Negotiate Response packet 336 is not valid, and process 400B proceeds NO to END operation 432B, in which the SMB Direct Client 214a disconnects. If all of the checks are not true, then process 400B proceeds YES to process SMB Direct Response 430, in which the SMB Direct Negotiate

Response packet 336 is processed by the SMB Direct Client 214a, according to embodiments disclosed herein.

[0078] In processing the SMB Direct Negotiate Response packet 336, the SMB Direct Client 214a completes the following, in embodiments, for example: sets the connection's PeerTargetSendCreditsCount setting in the settings 226 equal to the CreditsRequested field 308b; sets the connection's SendCreditsCount setting in the SMB Direct connection properties equal to the CreditsGranted field 320a; sets the connection's MaxSendSize setting in the settings 226 equal to the MaxReceiveSize field 312b; sets the connection's ReceiveSize setting in the settings 226 equal to the PreferredSendSize field 310b; sets the connection's MaxOutboundFragmentedMessageSize setting in the settings 226 equal to the MaxReceiveSize field 312b; and sets the connection's Idle Connection timer to expire in a predetermined amount of time (e.g., a few hours, a few minutes, etc.) and starts the timer. Once the above steps have been performed, the RDMA connection negotiation has completed, and the client 102 and server 106 may begin exchanging SMB Direct Data Transfer packets. Process 400B then terminates at END operation 432B.

[0079] The interactions of the various software functional modules depicted in FIGS. 2A and 2B are further illustrated in the operational steps 500 depicted in FIGS. 5A-5D for exchanging SMB2 data over the established RDMA connection, in accordance with an embodiment disclosed herein. In request and response communications, SMB2 data are transmitted between the client 102 and the server 106 over the established SMB Direct connection by encapsulating the SMB2 data as the data payload of a SMB Direct Data Transfer packet 338, according to embodiments of the present disclosure. Request and response communications may be used for control channel communications, the exchange of file metadata, or for other processes, for example. FIG. 5A shows a representation 500A of the transfer of packets 516/518, 530, 532, and 546 between a client(s) 102 and a server(s) 106, according to embodiments of the present disclosure. FIGS. 5B and 5C are in the perspective of the client 102, while FIG. 5D is in the perspective of the server 106. It should be noted that the process 500 is described as the client sending SMB2 data to the server 106 either independently or in response to a request from the server. However, the reverse could be true where the server 106 responds to a request and sends data back to the client 102. Thus, the processes described herein apply where the server sends requests and requires credits, in embodiments. Further, the method may be conducted between multiple clients or between multiple servers, for example.

[0080] Turning to FIG. 5B, example operational steps 500B for exchanging data using SMB2 over RDMA are illustrated in accordance with embodiments disclosed herein. Start operation 502 is initiated, and process 500B proceeds to the client 102 establishing an RDMA connection 504. Establishing an RDMA connection may be as described with respect to FIGS. 4A-4C, according to embodiments of the present disclosure. In an embodiment, the SMB Direct Client 214a sets several connection properties including setting the MaxSendSize setting equal to the MaxReceiveSize field 312b, setting the connection's ReceiveSize setting equal to the PreferredSendSize field 310b, and setting the connection's MaxOutboundFragmentedMessageSize setting in the settings 226 equal to the MaxReceiveSize field 312b. In this way, the client 102 has established the size of packets that may be sent to the server 106, according to embodiments.

[0081] Returning to FIG. 5B, the SMB Direct Client 214a may determine the number of SMB Direct Data Transfer packets that are needed to transport an SMB2 message to the server 106, according to embodiments. To make the determination, the SMB Direct Client 214a may first retrieve the MaxSendSize (designated as R) as set during the negotiation. The SMB Direct Client 214a may then determine the size of the SMB2 protocol message(s) that is to be sent; this value is set as "S" in embodiments. The SMB Direct Client 214a also determines the number of bytes (designated as P, in embodiments) consumed by the SMB Direct Data Transfer packet 338 header and the padding (e.g., the DataOffset 328, etc.) for the SMB Direct Data Transfer packet 338 payload to begin on an 8-byte-aligned boundary. In embodiments, the SMB Direct Client 214a then determines if S is less than or equal to (R – P), and, if so, may determine the number of packets by dividing S by (R-P), according to embodiments.

[0082] Process 500B then proceeds to determining, by the SMB Direct Client 214a, if fragmentation is used 510. In an embodiment, fragmentation is used when the SMB2 protocol message does not "fit" into a single SMB Direct Data Transfer packet 338. In other words, is S greater than (R – P)? If fragmentation is not to be used, process 500B proceeds NO to send one SMB Direct Data Transfer Packet 518B. On the other hand, if fragmentation is used, process 500B proceeds YES to initialize transmission buffer bytes 512.

[0083] Returning to step 518, an SMB2 protocol message may be transmitted without fragmentation, according to embodiments. A portion of the memory buffer 222a, used for transmitting packets, is initialized. In embodiments, the first P bytes are initialized to set the header information for the SMB Direct Data Transfer packet 338 and to add zeroed padding bytes to ensure that the SMB2 payload is 8-byte aligned. Thus, in the buffer, the DataOffset field 328 is set to P, the DataLength field 330 is set to S, and RemainingDataLength field 326 is set to zero (0), according to embodiments. Information about, or requests for, credits are stored in the CreditsRequested field 308c or the CreditsGranted field 320b. Thus, the client 102 may request or grant more credits in the SMB Direct Data Transfer packet 338. In embodiments, the sender of an SMB Direct Data Transfer packet must set the CreditsRequested field to at least one (1). The next S bytes are initialized in the memory buffer 222a, and the SMB2 protocol message is stored in the SMB2 data field 332. After the SMB Direct Data Transfer packet 338 is assembled, the SMB Direct Client 214a sends 518B the packet to the RDMA Interface 216a to send through the RDMA connection to the server 106. Process 500B then proceeds through page connector B 522 to END operation 548C, and process 500B terminates.

[0084] Returning to step 512, where S is greater than (R – P), the SMB2 protocol message is divided into two or more portions, according to embodiments. The portions are then sequenced to be sent in a series of RDMA Send operations, each of which carries a portion (called a fragment) of the SMB2 protocol message. To divide the SMB2 protocol message into fragments, the size of the first payload is set to X, which is equal to or less than (R – P). X represents the number of bytes of the SMB2 protocol message that will be transmitted by at least the first RDMA Send operation, according to an embodiment. As with step 518B, a portion of the memory buffer 222a, used for transmitting packets, is initialized. In embodiments, the first P bytes are initialized to set the header information for the SMB Direct Data Transfer packet 338 plus the padding bytes. Next, the DataOffset field 328 is set to P, the DataLength field 330 is set to X, and the RemainingDataLength field 326 is set to (S – X) 514. The RemainingDataLength field 326 indicates, in embodiments, how many bytes of the SMB2 protocol message remain to be transmitted. In embodiments, when the last fragment of the SMB2 protocol message is transmitted, (S – X) will be zero (0) In the last packet, the RemainingDataLength field 326 is then set to zero, which indicates that the SMB Direct Data Transfer packet 338 carries the last fragment of the SMB2 protocol message. Information about, or requests for, credits are stored in the CreditsRequested field 308c or the CreditsGranted field 320b. Thus, in embodiments, the client 102 requests or grants more credits in the SMB Direct Data Transfer packet 338. The next X bytes of the memory buffer 222a are then initialized. The fragment of the SMB2 protocol message may then be stored in the SMB2 data field 332, according to embodiments disclosed herein.

[0085] After assembling the SMB Direct Data Transfer packet 338 with the first untransmitted X bytes of the SMB2 protocol message, the SMB Direct Data Transfer packet 338 is sent 516B to the server. The SMB Direct Client 214a sends 516B the SMB Direct Data Transfer packet 338 to the RDMA Interface 216a to send through the RDMA connection to the server 106, in embodiments. Process 500B then proceeds through page connector A 520 to optional step 526, in which the SMB Direct Client 214a determines 526 if another SMB Direct Data Transfer packet 338 containing another fragment is to be sent. Thus, in embodiments, the SMB Direct Client 214a determines if, after sending the last packet, (S – X) is 0. If (S – X) is 0 or less than zero, meaning the last fragment was sent in the last SMB Direct Data Transfer packet 338, process 500B proceeds NO to END operation 548C, and process 500B terminates. However, if (S – X) is greater than 0, then process 500B proceeds YES to determine if enough send credits exist 528. In embodiments, step 526 may be optional (as shown) because the SMB Direct Data Transfer packets 338 transporting the fragments may be pre-staged in the memory buffer 222a and sent sequentially. When the buffer 222a is empty, process 500B terminates. There is then no need to determine if a next packet is to be sent. It should be noted that when an SMB2 protocol message is fragmented, the RDMA Send operations that carry the fragments are sent sequentially and monotonically, and may not be interrupted by other RDMA Send operations that are unrelated to the fragmented SMB2 protocol message, according to embodiments. The RDMA transport ordering, at the receiver, will preserve the sequencing of the fragments, such that the receiving peer may reconstruct the original message, according to an embodiment.

[0086] Returning to step 528, the SMB Direct Client 214a determines 528 if there are enough send credits to send the next fragment. Essentially, the SMB Direct Client 214a determines if there is one send credit left. If there is a send credit, process 500B proceeds YES through page connector C 524 to step 512. In embodiments, RDMA transports have a receive buffer pre-posted by the receiver before the sender may send a packet. This rule requires coordination between the sender and receiver to ensure that senders do not attempt to send a packet before the receiver has pre-posted a receive. SMB Direct uses a system of Send Credits to achieve the desired coordination, according to embodiments of the present disclosure.

[0087] In an embodiment, a send credit granted from a server 106 or client 102 represents a single pre-posted receive on the server 106 or client 102. The peer is entitled to perform one RDMA Send operation with one send credit. The client 102 may also provide information to the server 106 about how many send credits the client 102 needs to efficiently support its workload by setting the CreditsRequested field 308c or in the form of a separate Send Credit Request (which is not to be sent in the middle of a set of SMB Direct Data Transfer packets 338 transporting fragmented SMB2 data, according to an embodiment). Send credits are granted in the CreditsGranted field 320b of the SMB Direct Data Transfer packet 338. Thus, the SMB Direct Client 214a may request more send credits by setting the value in the CreditsRequest field 308c to a higher number (possibly covering all the future fragmented packets). The SMB Direct Client 214a may then wait 530C to receive send credits by awaiting a SMB Direct Data Transfer packet 338, sent from the server 106, which has no data but includes send credits in the CreditsGranted field 320b of the SMB Direct Data Transfer packet 338, according to an embodiment. The SMB Direct Client 214a may determine at query 532C if credits are received by setting a SendCreditsGrantedTimer in the timers 224a. If the SendCreditsGrantedTimer expires 547 before credits are granted, process 500B proceeds YES to fail, or terminate, the transmission operation, and process 500B then terminates at END operation 548C. However, if credits are granted 532C, process 500B proceeds YES through page connector C 524 to step 512. According to an embodiment, if no timer is set and/or the timer has not expired 547, process 500B proceeds NO and does not fail, or terminate, the transmission operation but, instead, continues to wait 530C to receive send credits.

[0088] Turning to FIG. 5D and start operation 502D for process 500D, at the server 106, the SMB Direct Data Transfer packet 338 is received 516D. In an embodiment, the RDMA Interface 216b receives the SMB Direct Data Transfer packet 338 and sends it to the SMB Direct Server 214b. First, the SMB Direct Server 214b reads the header information. The SMB Direct Server 214b proceeds to query 536 to determine if the RemainingDataLength field 326 is something other than zero. If the RemainingDataLength field 326 is zero, then the SMB Direct Data Transfer packet 338 is the only packet, and process 500D proceeds NO to process SMB packet data 544 where the SMB Direct Server 214b processes the packet data and sends the SMB2 protocol message to SMB2 Server 212b, in an embodiment. If the RemainingDataLength field 326 is something other than zero, process 500D proceeds YES to allocate reassembly buffer 538.

[0089] In step 538, the SMB Direct Server 214b allocates a portion of the memory buffer 222b to reassemble the SMB2 protocol message in the buffer 222b, in an embodiment. Thus, the SMB Direct Server 214b may allocate enough blocks in the memory buffer 222b to accept the data in the current SMB Direct Data Transfer packet 338 and in the upcoming SMB Direct Data Transfer packets 338, which is based on the DataLength field 330 value plus the RemainingDataLength field 326 value. The SMB Direct Server 214b may then read the SMB2 data from the SMB2 data field 332 and copy the data into the allocated buffer 540, according to embodiments of the present disclosure.

[0090] Next, an embodiment provides for the SMB Direct Server 214b to wait and receive 546D the next SMB Direct Data Transfer packets 338. The data in the SMB2 data field 332 is also copied into the next portion of the memory buffer 222b, in an embodiment. In an alternative embodiment, the SMB Direct Server 214b may optionally determine if this next SMB Direct Data Transfer packet 338 is the last packet containing fragmented data 542. To make the determination, the SMB Direct Server 214b checks if the RemainingDataLength field 326 is zero. If the RemainingDataLength field 326 is zero, the SMB Direct Server 214b understands that no more packets will be received, and process 500D proceeds YES to process the SMB2 protocol message 544 from the memory buffer 222b. Process 500D then terminates at END operation 548D. On the other hand, if the RemainingDataLength field 326 is something other than zero, the SMB Direct Server 214b understands that another packet(s) will be received, and process 500D proceeds NO to receive 546D the next SMB Direct Data Transfer packet 338, according to an embodiment.

[0091] The interactions of the various software functional modules depicted in FIGS. 2A and 2B are further illustrated in the operational steps 600 depicted in FIG. 6 for performing an RDMA Direct Data Transfer in accordance with an embodiment disclosed herein. FIG. 6 shows the representation of the data transfer between a client 102 and a server 106, according to embodiments of the present disclosure.

[0092] As shown in FIG. 6, process 600 is initiated at start operation 602, and the SMB2 Client 208 creates 604 an SMB2 read/write request 340. An application 220 can request data to be transferred to the server 106 or read from the server 106. The data transfer may be directed by the SMB2 Client 208 to employ an RDMA direct data transfer, such that the SMB2 server 106 uniquely performs the actual RDMA request, in accordance with embodiments of the present disclosure. To begin the transfer, the SMB2 Client 208 registers the target memory buffers to provide or receive unencoded application data, and then generates an SMB read/write command that is to be sent and conducted over RDMA. The command may be included in the SMB2 read/write request 340 and sent to the SMB Direct Client 214a, which encapsulates 606 the SMB2 read/write request 340 in a SMB Direct Data Transfer packet. In embodiments, the SMB2 read/write request 340 is stored in the SMB2 Data field 332 in the SMB Direct Data Transfer packet. The SMB Direct Data Transfer packet may then be sent 608 over the RDMA interface 216 to the server 106. In an embodiment, the server 106 receives the SMB Direct Data Transfer packet and reads the SMB2 read/write request 340 from the SMB2 Data field 332. The SMB Direct Server 214b may then read the data from the SMB2 read/write request 340, including the channel 342, ChannelInfoOffset 346, and the ChannelInfoLength 348. This information may direct the SMB Direct Server 214b to the steering information 610 in the RDMA Channel Descriptor 352. From the steering information, the SMB Direct Server 214b may start the direct data transfer by sending or retrieving 612 unencoded application data to or from a memory location in a buffer of the client 102. Process 600 then terminates at END operation 614.

[0093] FIGS. 4A-4C, 5A-5D, and 6 illustrate example operational characteristics for negotiating a communication using a file access protocol over RDMA and exchanging data using a file access protocol over RDMA, respectively, in accordance with embodiments disclosed herein. In embodiments, operational steps depicted may be combined into other steps and/or rearranged. Further, fewer or additional steps may be used, for example.

[0094] In general, credits are an advantage to the present embodiments. A send credit represents a buffer that has been pre-posted to receive incoming data on a peer. As such, a send credit represents limited receiver resources (memory, memory regions, etc.) that have been committed to the peer so that the peer may use them to transmit data. Due to the nature of RDMA, a receive, once-posted, may not be canceled, in embodiments. For the resources that are associated with the receive to be released, the resources are used to service an incoming send. This poses a possible problem in embodiments if the receiving peer starts to run low on resources and wants to reclaim some of the resources it has dedicated to outstanding receives. Since receives may not be cancelled according to embodiments disclosed herein, reclaiming these resources relies on the cooperation of the peer.

[0095] A solution to this issue in SMB Direct is referred to as the Send Credit Revocation, according to an embodiment. An SMB Direct Data Transfer packet that is transmitted to the peer has a CreditsGranted field that specifies how many additional send credits have been granted to the peer. In alternative embodiments, by setting a

CreditRevocation flag (not shown) in the SMB Direct Data Transfer packet 338, the meaning of the CreditsGranted field 320b changes to be the number of Send Credits that the peer may keep. For example, if the CreditRevocation flag were set and the value of CreditsGranted field 320b is ten (10) then ten (10) is the number of credits the receiver may retain. If the receiver currently holds more than ten (10) Send Credits, then the receiver performs a series of RDMA Send operations to use up the revoked Send Credits so that the receiver ends up with ten (10) or fewer Send Credits. The send operations that are performed to use up revoked Send Credits may include sending an empty SMB Direct Data Transfer packet 338 (a packet with no data payload). Upon receiving the incoming empty SMB Direct Data Transfer packets 338, the peer that revoked the Send Credits may release these resources or reuse them as it sees fit, according to embodiments.

[0096] While Send Credits allow two peers to synchronize their Send and Receive operations, the peers, in embodiments, attempt to avoid send credit deadlocks. Imagine a scenario in which Peer Y has a single Send Credit from peer X and where peer X has a single Send Credit from peer Y. In this scenario, each peer is entitled to perform a single send to its peer at any time. Imagine that both X and Y simultaneously perform a Send operation, using their single Send Credit in the process, but the SMB Direct Data Transfer packet 338 that is transmitted by both peers grants no additional Send Credits to the peer. The resulting state is known as a Send Credit deadlock. Both X and Y have used their last Send Credit. To be able to send additional packets to the peer, each peer needs additional Send Credits. However, Send Credits are granted via SMB Direct Data Transfer packets 338, and neither peer may perform any further send operations. The result is that neither peer may transmit any additional packets, and there is no mechanism by which they may acquire further send credits. A deadlock has therefore occurred.

[0097] A solution to this issue is simply a rule, according to embodiments of the present disclosure. When an SMB Direct peer uses its last Send Credit, the SMB Direct Data Transfer packet 338 that is being sent grants at least one (1) additional send credit to the peer, according to an embodiment. If both peers follow this rule, the deadlock does not occur because each peer will always be able to respond to the other with a grant of additional Send Credits.

[0098] Finally, FIG. 7 illustrates an example computing system 700 upon which embodiments disclosed herein may be implemented. A computer system 700, such as client/application server 102 or file server 106, for example, which has at least one processor 702 for exchanging message data as shown herein, is depicted in accordance with embodiments disclosed herein. The system 700 has a memory 704 comprising, for example, system memory, volatile memory, and non-volatile memory. In its most basic configuration, computing system 700 is illustrated in FIG. 7 by dashed line 706. Additionally, system 700 may also include additional storage (removable and/or non-removable) including, but not limited to, magnetic or optical disks or tape. Such additional storage is illustrated in FIG. 7 by removable storage 708 and non-removable storage 710.

[0099] The term computer readable media as used herein may include computer storage media. Computer storage media may include volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information, such as computer readable instructions, data structures, program modules, or other data. System memory 704, removable storage 708, and non-removable storage 710 are all computer storage media examples (i.e., memory storage.) Computer storage media may include, but is not limited to, RAM, ROM, electrically erasable read-only memory (EEPROM), flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which may be used to store information and which may be accessed by computing system 700. Any such computer storage media may be part of system 700. The illustration in FIG. 7 is intended in no way to limit the scope of the present disclosure.

[00100] The term computer readable media as used herein may also include communication media. Communication media may be embodied by computer readable instructions, data structures, program modules, or other data in a modulated data signal, such as a carrier wave or other transport mechanism, and includes any information delivery media. The term "modulated data signal" may describe a signal that has one or more characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media may include wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, radio frequency (RF), infrared, and other wireless media.

[00101] System 700 may also contain communications connection(s) 716 that allow the device to communicate with other devices. Additionally, system 700 may have input device(s) 714 such as a keyboard, mouse, pen, voice input device, touch input device, etc. Output device(s) 712 such as a display, speakers, printer, etc. may also be included. All of these devices are well known in the art and need not be discussed at length here. The aforementioned devices are examples and others may be used.

[00102] Having described embodiments of the present disclosure with reference to the figures above, it should be appreciated that numerous modifications may be made to the embodiments that will readily suggest themselves to those skilled in the art and which are encompassed within the scope and spirit of the present disclosure and as defined in the appended claims. Indeed, while embodiments have been described for purposes of this disclosure, various changes and modifications may be made which are well within the scope of the present disclosure.

[00103] Similarly, although this disclosure has used language specific to structural features, methodological acts, and computer-readable media containing such acts, it is to be understood that the subject matter defined in the appended claims is not necessarily limited to the specific structure, acts, features, or media described herein. For example, while specific names or naming conventions have been used in describing aspects of the embodiments, such as names for APIs, routines, etc., numerous modifications may be made to such names and/or naming conventions which are encompassed within the spirit and scope of the present disclosure. The specific structures, features, acts, names, naming conventions, and/or media described above are disclosed as example forms of implementing the claims. Aspects of embodiments allow for multiple client/application servers, multiple file servers, multiple networks, multiple connection peers, etc. Or, in other embodiments, a single client computer with a single server and a single network are used. One skilled in the art will recognize other embodiments or improvements that are within the scope and spirit of the present disclosure. Therefore, the specific structure, acts, or media are disclosed as example embodiments of implementing the present disclosure. The disclosure is defined by the appended claims.

CLAIMS

1. A computer readable storage medium storing computer-executable instructions that when executed by a processor perform a method for exchanging data using data operations over remote direct memory access (RDMA), the method comprising:

establishing a connection with a server;

negotiating a connection with the server, wherein the negotiation establishes a maximum number of bytes the server will receive in the connection;

determining a number of packets to send data associated with the connection to the server;

determining if fragmentation of the data is to be used;

if fragmentation of the data is not to be used, sending a first protocol packet to the server with the data;

if fragmentation of the data is to be used, initializing first bytes within a transmission buffer to send as data in the first protocol packet;

setting a DataLength field and a RemainingDataLength field in the first protocol packet;

sending the first protocol packet to the server;

for at least a second protocol packet, instructions to repeat, comprising:

initializing bytes within the transmission buffer to send as data in the at least a second protocol packet;

setting the RemainingDataLength field in the at least a second protocol packet; and

sending the at least a second protocol packet to the server.

2. The computer readable storage medium of claim 1, wherein the determining if fragmentation is to be used comprises determining if the number of packets to send the data is greater than one.

3. The computer readable storage medium of claim 2, wherein the determining a number of packets to send data associated with the connection to the server comprises determining if the maximum number of bytes the server will receive in the connection is less than a number of bytes of data to be sent.

4. The computer readable storage medium of claim 1, wherein negotiating the connection with the server comprises:

pre-posting a receive;

constructing a protocol negotiate request;

sending the protocol negotiate request to the server;

setting a negotiate expiration timer;

determining if a response to the protocol negotiate request is received before the negotiate expiration timer expires;

if a response to the protocol negotiate request is not received before the negotiate expiration timer expires, resending the protocol negotiate response;

if a response to the protocol negotiate request is received before the negotiate expiration timer expires, receiving a protocol negotiate response;

validating the protocol negotiate response; and

processing the protocol negotiate response, wherein the protocol negotiate response provides at least one credit to the sender to send the protocol packet.

5. The computer readable storage medium of claim 4, further comprising:

if enough credits exist for sending the at least a second protocol packet, sending the at least a second protocol packet;

if enough credits do not exist to send the at least a second protocol packet, requesting at least one more credit from the server;

determining if enough credits are received;

if enough credits are not received, terminating the connection; and

if enough credits are received, sending the at least a second protocol packet.

6. A system configured to exchange data using server message block (SMB/SMB2) over remote direct memory access (RDMA), the system comprising:

a RDMA network interface card (RNIC) operable to transfer data by a RDMA message;

memory operable to store computer program instructions executable by a processor; and

the processor, in communication with the RNIC and the memory, operable to execute a kernel, the kernel comprising:

a first protocol client operable to:

communicate SMB data in one or more RDMA messages; and

create a receive buffer to exchange credits, wherein the credits determine a number of messages that a sender may send to a receiver.

7. The system of claim 6, further comprising an RDMA Interface in communication with the first protocol client, the RDMA Interface operable to open an RDMA connection through the RNIC.

8. The system of claim 7, further comprising:

a second protocol client operable to:

create an SMB2 read/write request to instruct a server to perform an RDMA direct data transfer;

send the SMB2 read/write request to the first protocol client; and

wherein the first protocol client is further operable to:

encapsulate the SMB2 read/write request in a first protocol Data Transfer packet; and

send the first protocol Data Transfer packet to the server.

9. The system of claim 8, wherein the SMB2 read/write request includes fields that identify RDMA channel descriptor information, and wherein the first protocol is the SMB Direct protocol.

10. The system of claim 9, wherein the RDMA channel descriptor information includes steering information, and wherein the server determines the steering information and performs an RDMA write or an RDMA read of unencoded application data to or from memory addresses identified by the steering information.

11. A computer-implemented method for establishing a connection that exchanges data using server message block (SMB/SMB2) over remote direct memory access (RDMA), the method comprising:

receiving a first SMB Direct Data Transfer packet, wherein the first SMB Direct Data Transfer packet comprises a RemainingDataLength field and SMB2 data;

determining if the RemainingDataLength field is non-zero;

if the RemainingDataLength field is zero, processing the SMB2 data in the first SMB Direct Data Transfer packet;

if the RemainingDataLength field is non-zero, allocating a reassembly buffer to the connection;

copying the SMB2 data from the first SMB Direct Data Transfer packet into the reassembly buffer;

receiving at least one other SMB Direct Data Transfer packet; and

copying the SMB2 data from the at least one other SMB Direct Data Transfer packet into the reassembly buffer.

12. The method of claim 11, wherein the SMB Direct Data Transfer packet comprises one or more fields from the group consisting of: CreditsRequested, CreditsGranted, DataOffset, and DataLength.

13. The method of claim 12, wherein the CreditsGranted field delineates how many credits were provided to a sender of the SMB Direct Data Transfer packet.

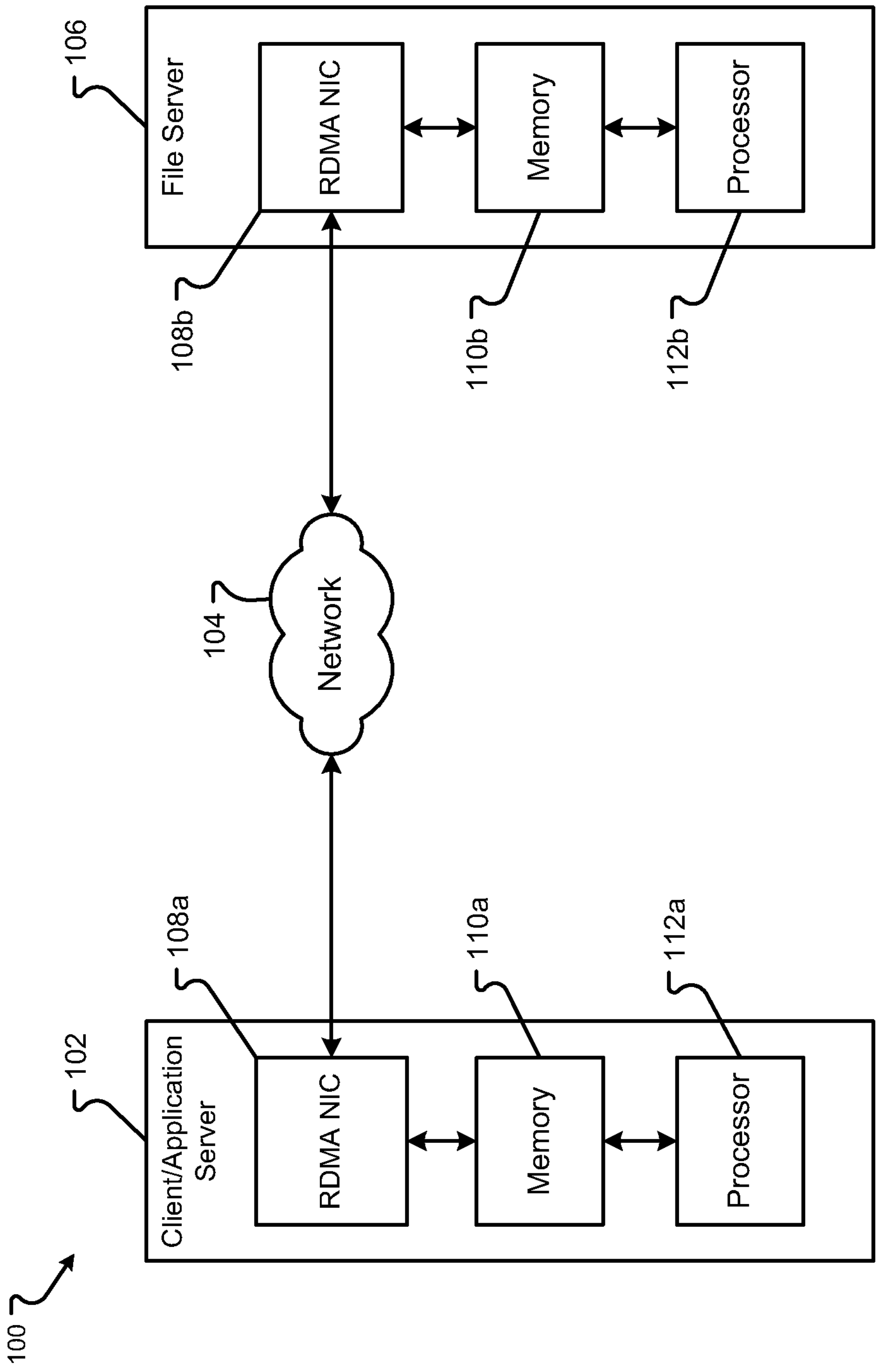
14. The method of claim 11, wherein a credit is a value for the number of RDMA messages that may be sent by the sender.

15. The method of claim 11, further comprising:

determining if the at least one other SMB Direct Data Transfer packet is a last packet;

if the at least one other SMB Direct Data Transfer packet is the last packet, processing the SMB2 data in the reassembly buffer;

if the at least one other SMB Direct Data Transfer packet is not the last packet, repeating receiving at least one other SMB Direct Data Transfer packet and copying the SMB2 data from the at least one other SMB Direct Data Transfer packet into the reassembly buffer.



100
102
Client/Application Server
RDMA NIC
Memory
Processor
108a
110a
112a
104
Network
106
File Server
RDMA NIC
Memory
Processor
108b
110b
112b

Fig. 1

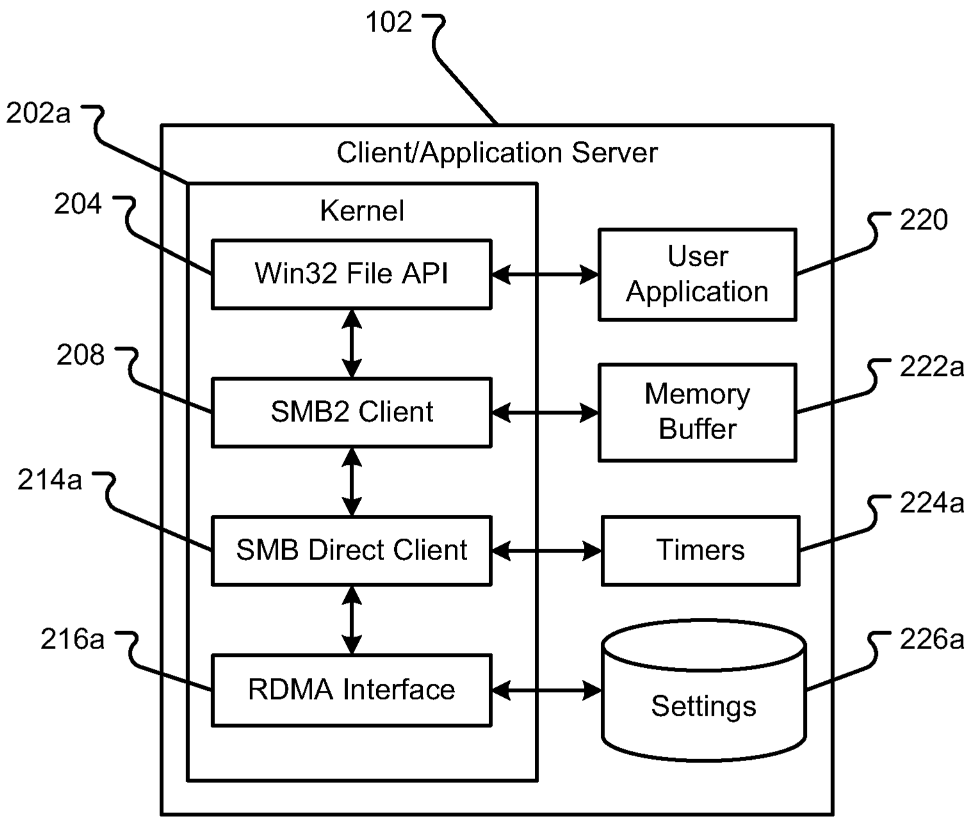

102
Client/Application Server
202a
Kernel
204
Win32 File API
208
SMB2 Client
214a
SMB Direct Client
216a
RDMA Interface
220
User Application
222a
Memory Buffer
224a
Timers
226a
Settings

Fig. 2A

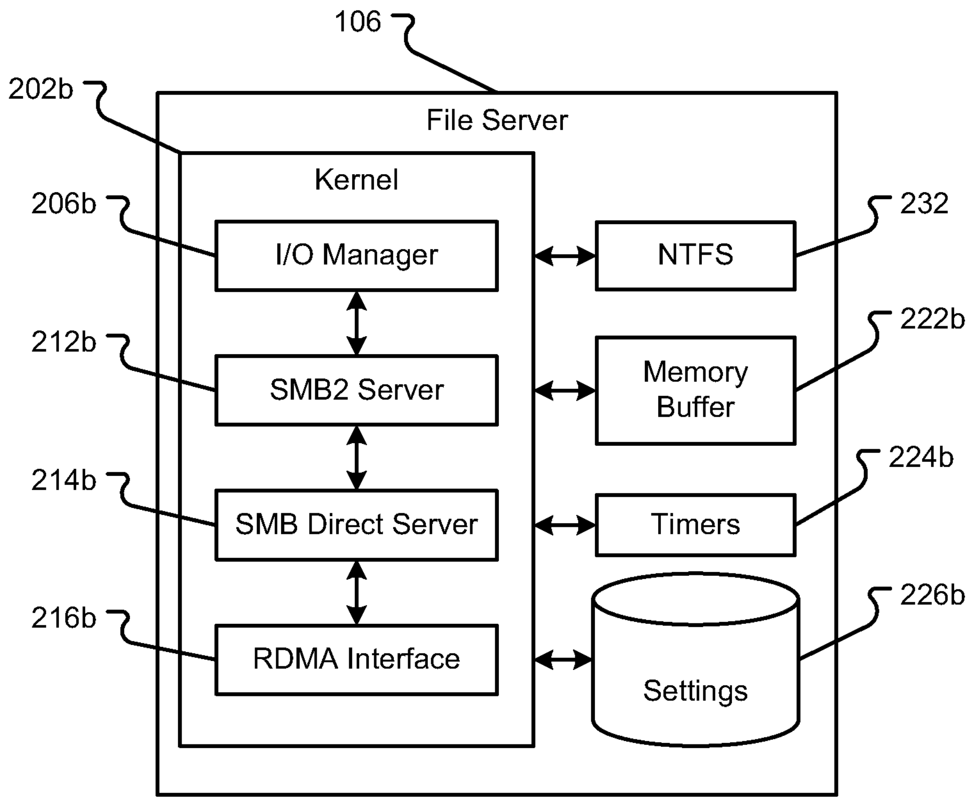

106
202b
File Server
Kernel
206b
I/O Manager
212b
SMB2 Server
214b
SMB Direct Server
216b
RDMA Interface
NTFS
232
Memory Buffer
222b
Timers
224b
Settings
226b

Fig. 2B

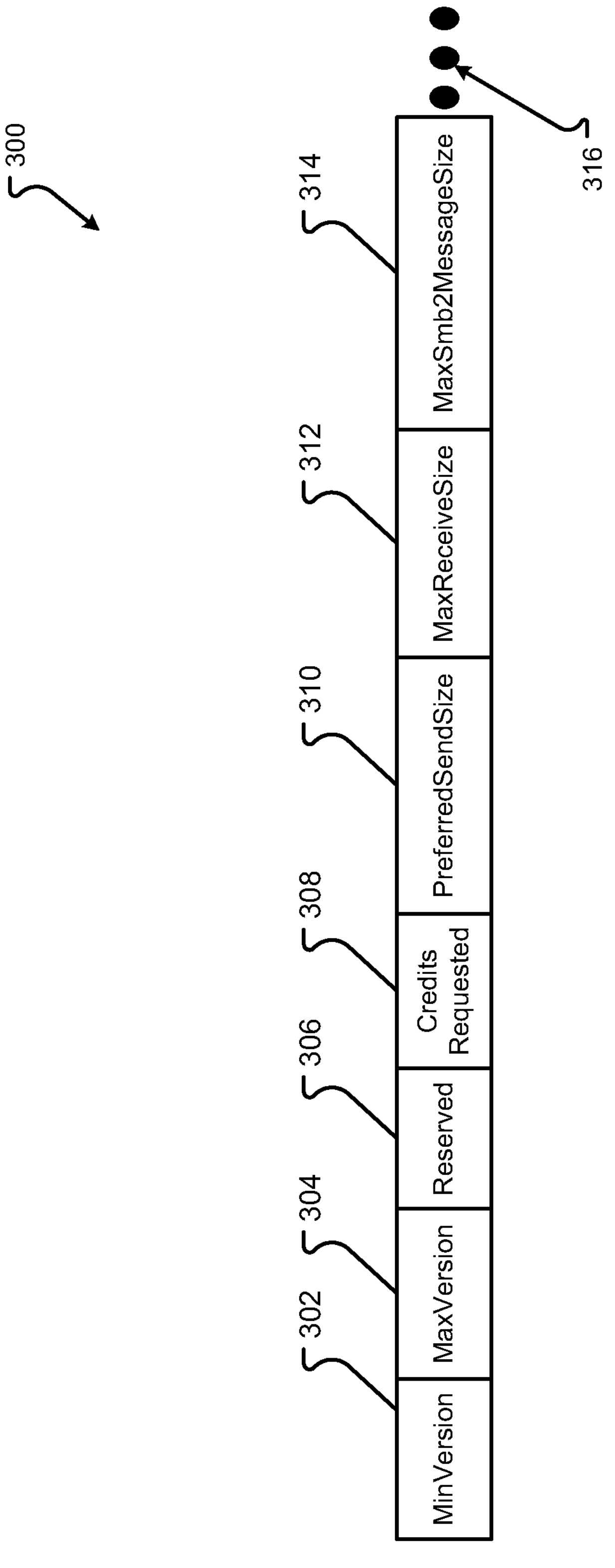

300
MinVersion
MaxVersion
Reserved
Credits Requested
PreferredSendSize
MaxReceiveSize
MaxSmb2MessageSize
302
304
306
308
310
312
314
316

Fig. 3A

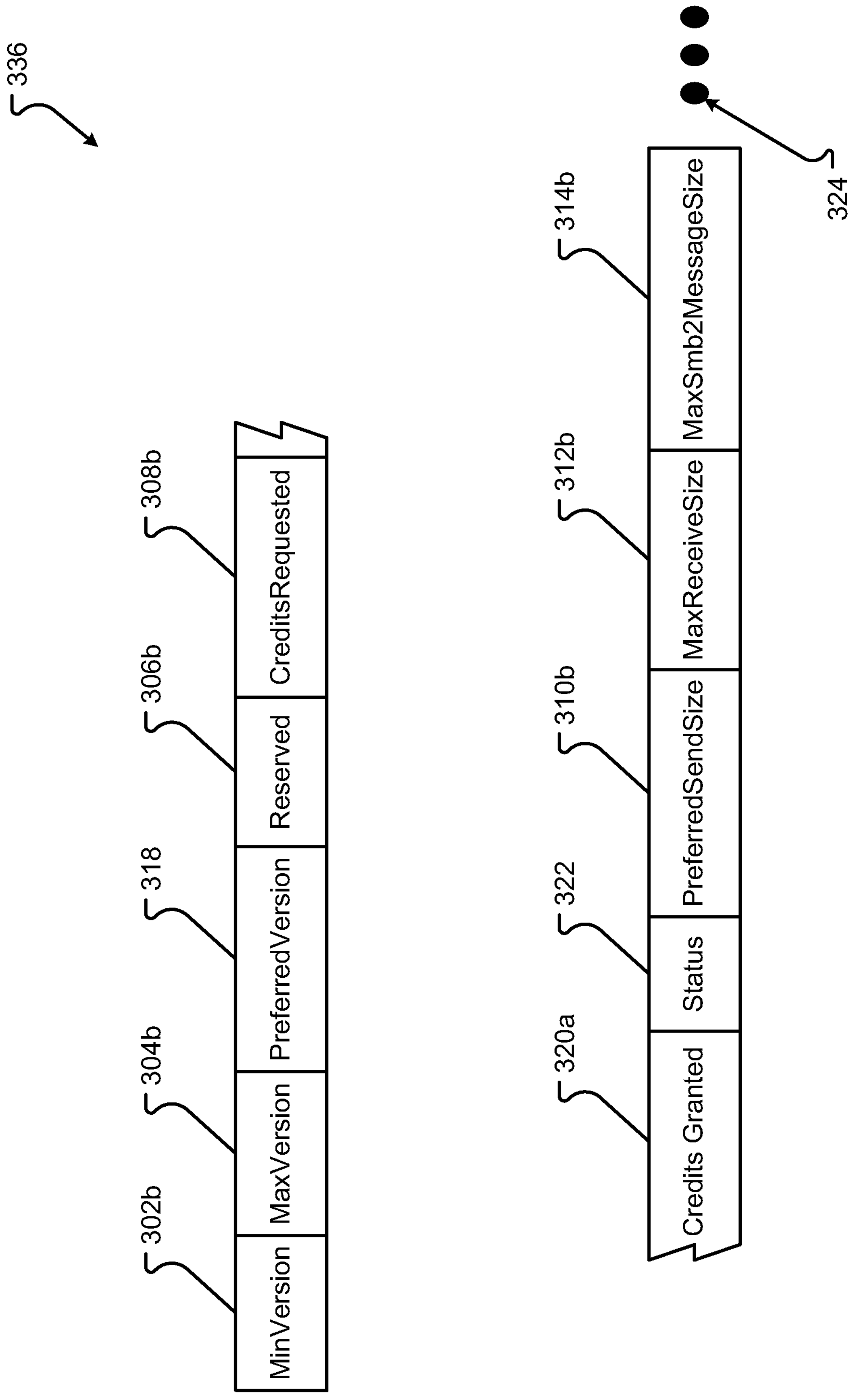

336
MinVersion
MaxVersion
PreferredVersion
Reserved
CreditsRequested
302b
304b
318
306b
308b
Credits Granted
Status
PreferredSendSize
MaxReceiveSize
MaxSmb2MessageSize
320a
322
310b
312b
314b
324

Fig. 3B

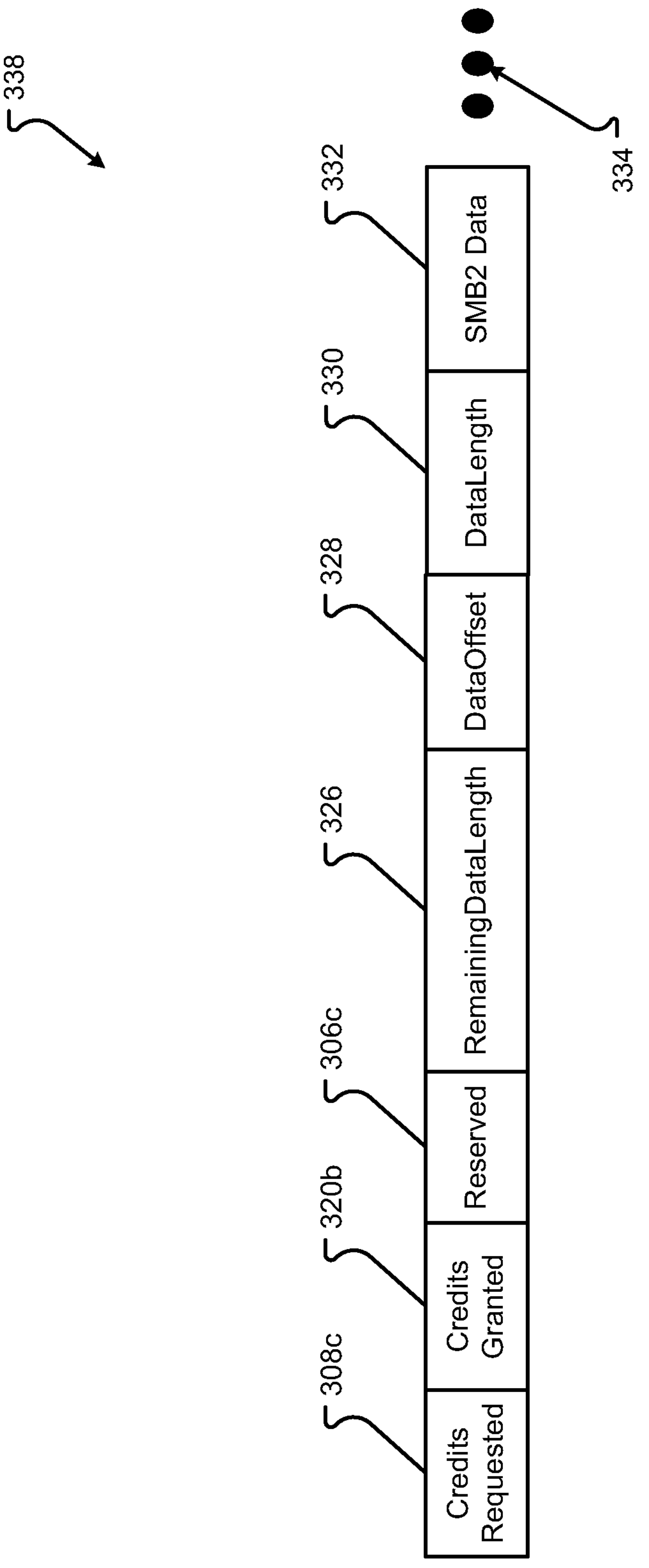

338
308c
Credits Requested
320b
Credits Granted
306c
Reserved
326
RemainingDataLength
328
DataOffset
330
DataLength
332
SMB2 Data
334

Fig. 3C

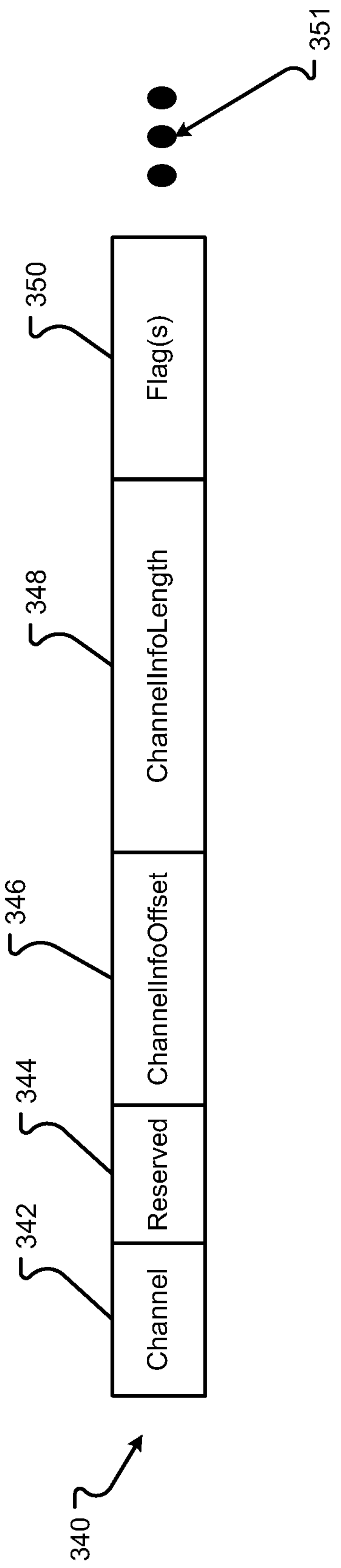

340
Channel
342
Reserved
344
ChannelInfoOffset
346
ChannelInfoLength
348
Flag(s)
350
351

Fig. 3D

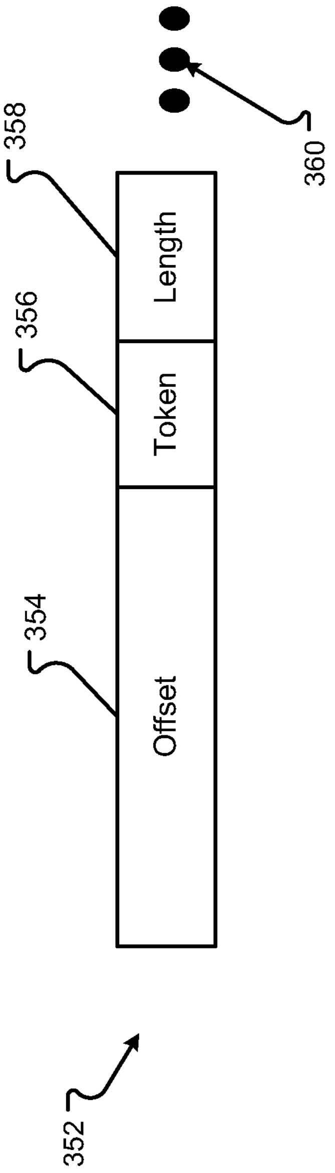

352
Offset
354
Token
356
Length
358
360

Fig. 3E

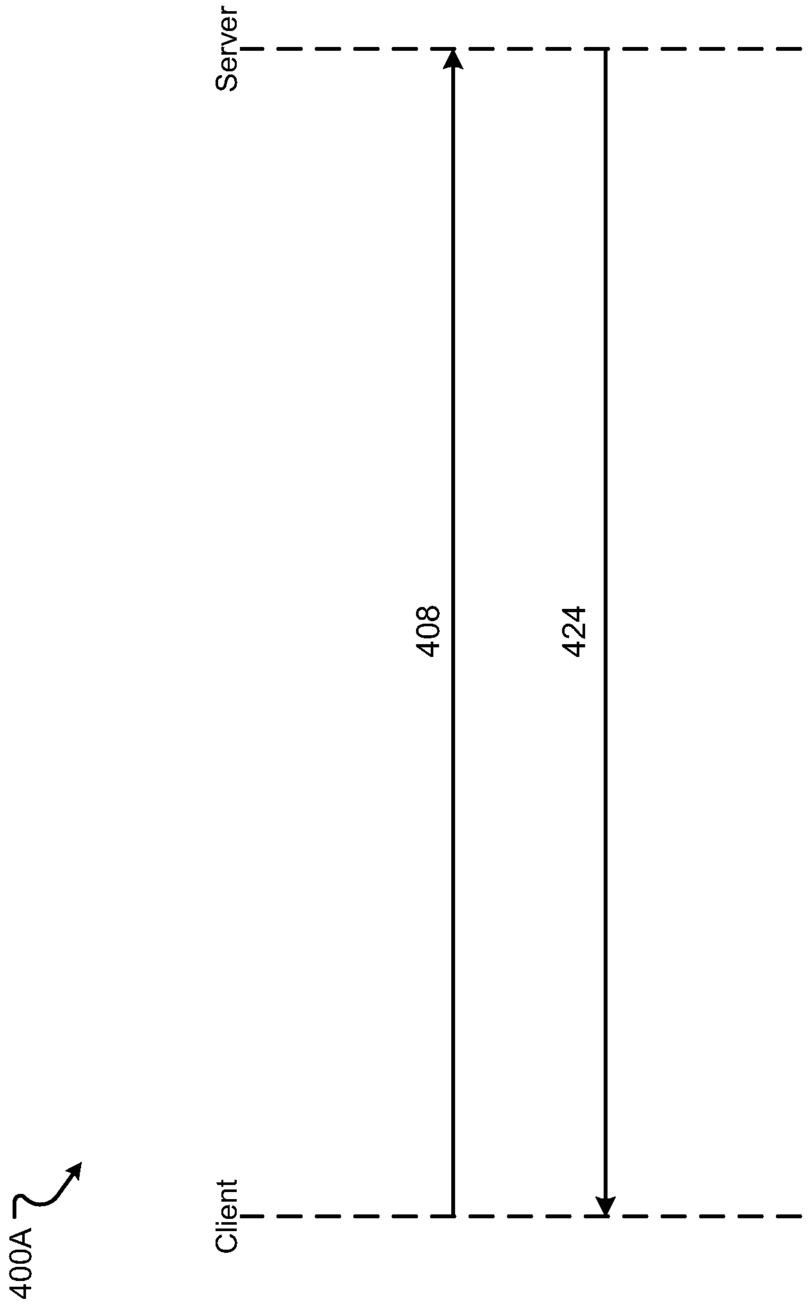


Fig. 4A

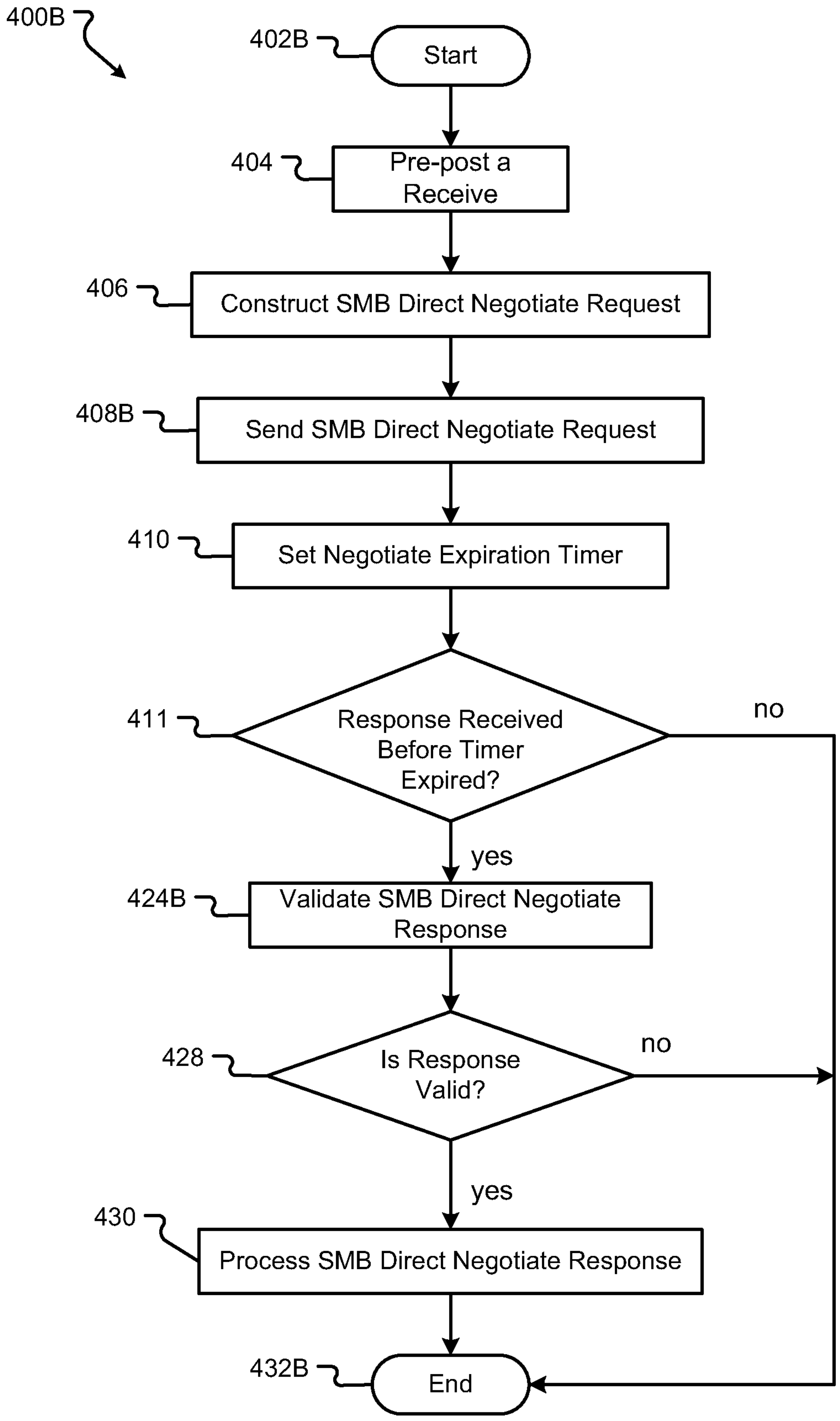

400B
402B
Start
404
Pre-post a Receive
406
Construct SMB Direct Negotiate Request
408B
Send SMB Direct Negotiate Request
410
Set Negotiate Expiration Timer
411
Response Received Before Timer Expired?
no
yes
424B
Validate SMB Direct Negotiate Response
428
Is Response Valid?
no
yes
430
Process SMB Direct Negotiate Response
432B
End

Fig. 4B

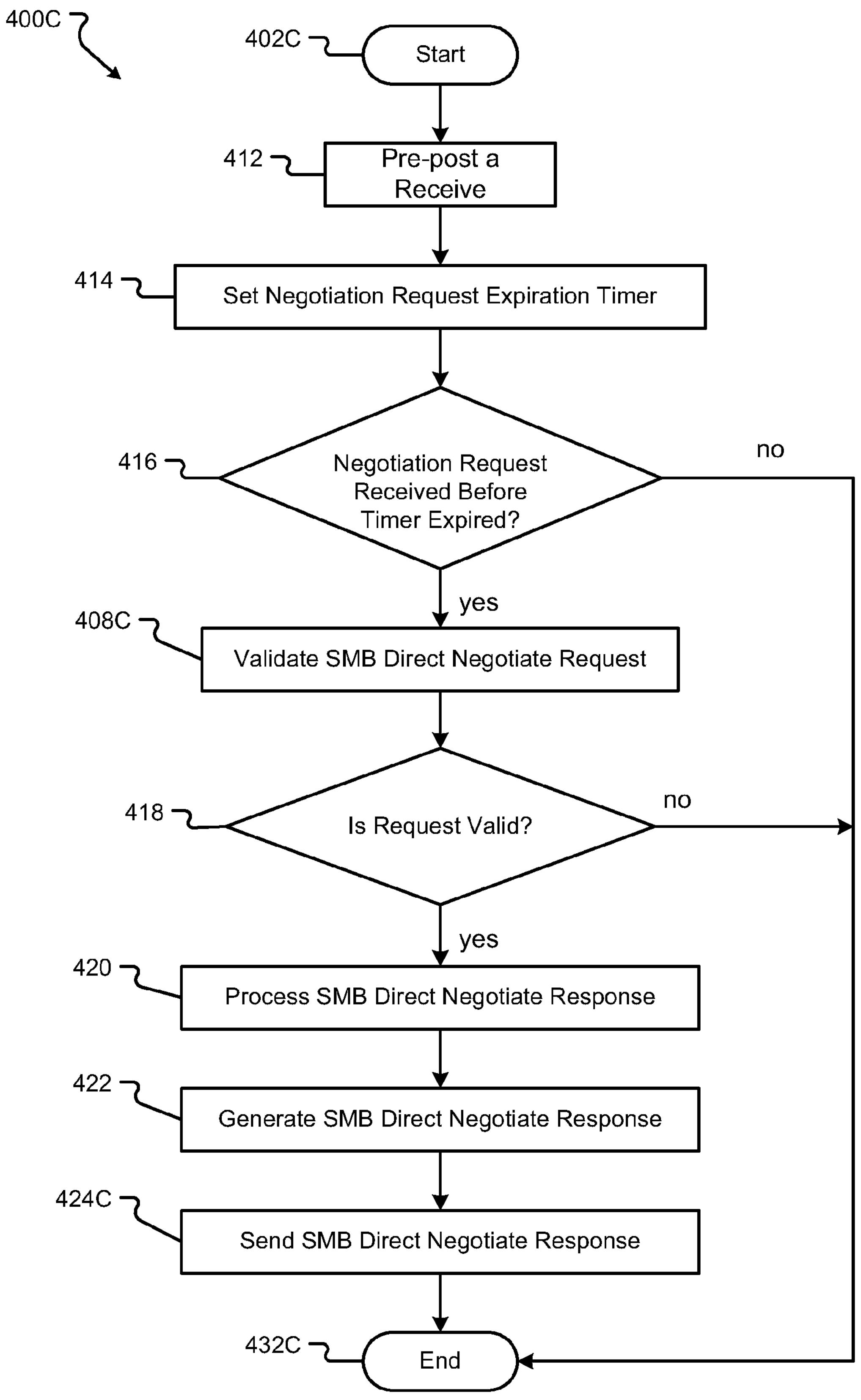

400C
402C
Start
412
Pre-post a Receive
414
Set Negotiation Request Expiration Timer
416
Negotiation Request Received Before Timer Expired?
no
yes
408C
Validate SMB Direct Negotiate Request
418
Is Request Valid?
no
yes
420
Process SMB Direct Negotiate Response
422
Generate SMB Direct Negotiate Response
424C
Send SMB Direct Negotiate Response
432C
End

Fig. 4C

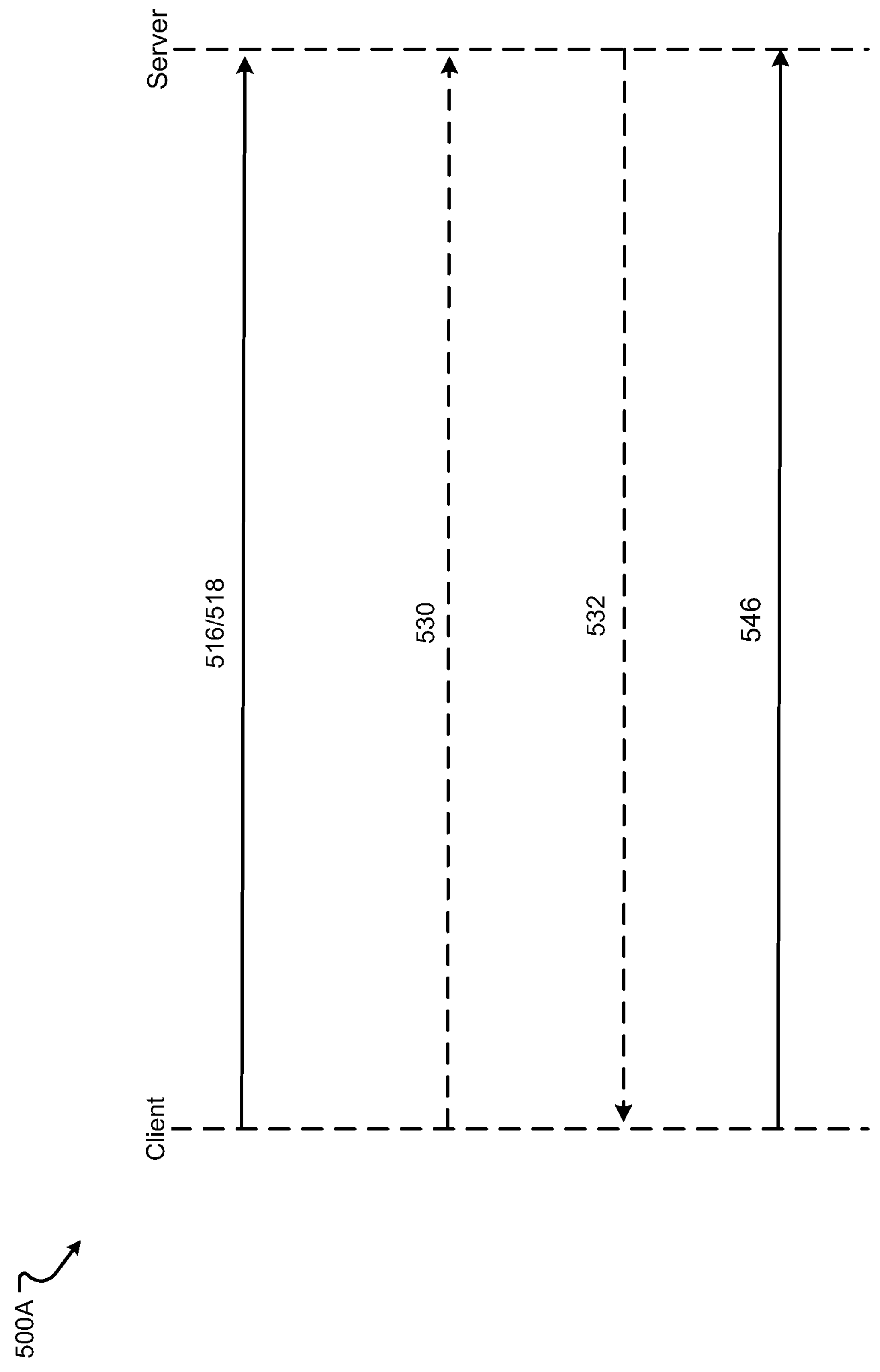


Fig. 5A

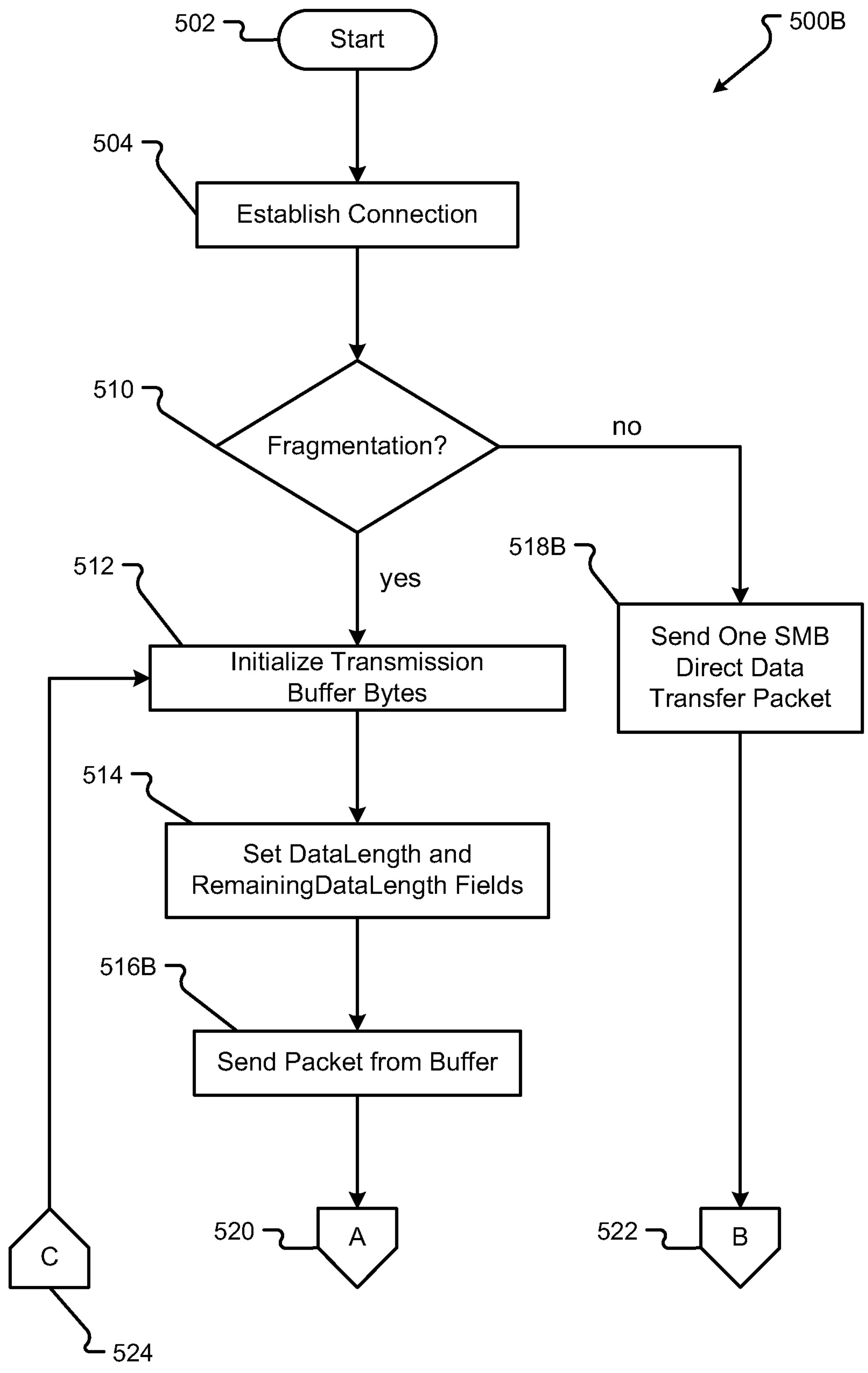

500B
502
Start
504
Establish Connection
510
Fragmentation?
no
yes
518B
Send One SMB Direct Data Transfer Packet
512
Initialize Transmission Buffer Bytes
514
Set DataLength and RemainingDataLength Fields
516B
Send Packet from Buffer
520
A
522
B
C
524

Fig. 5B

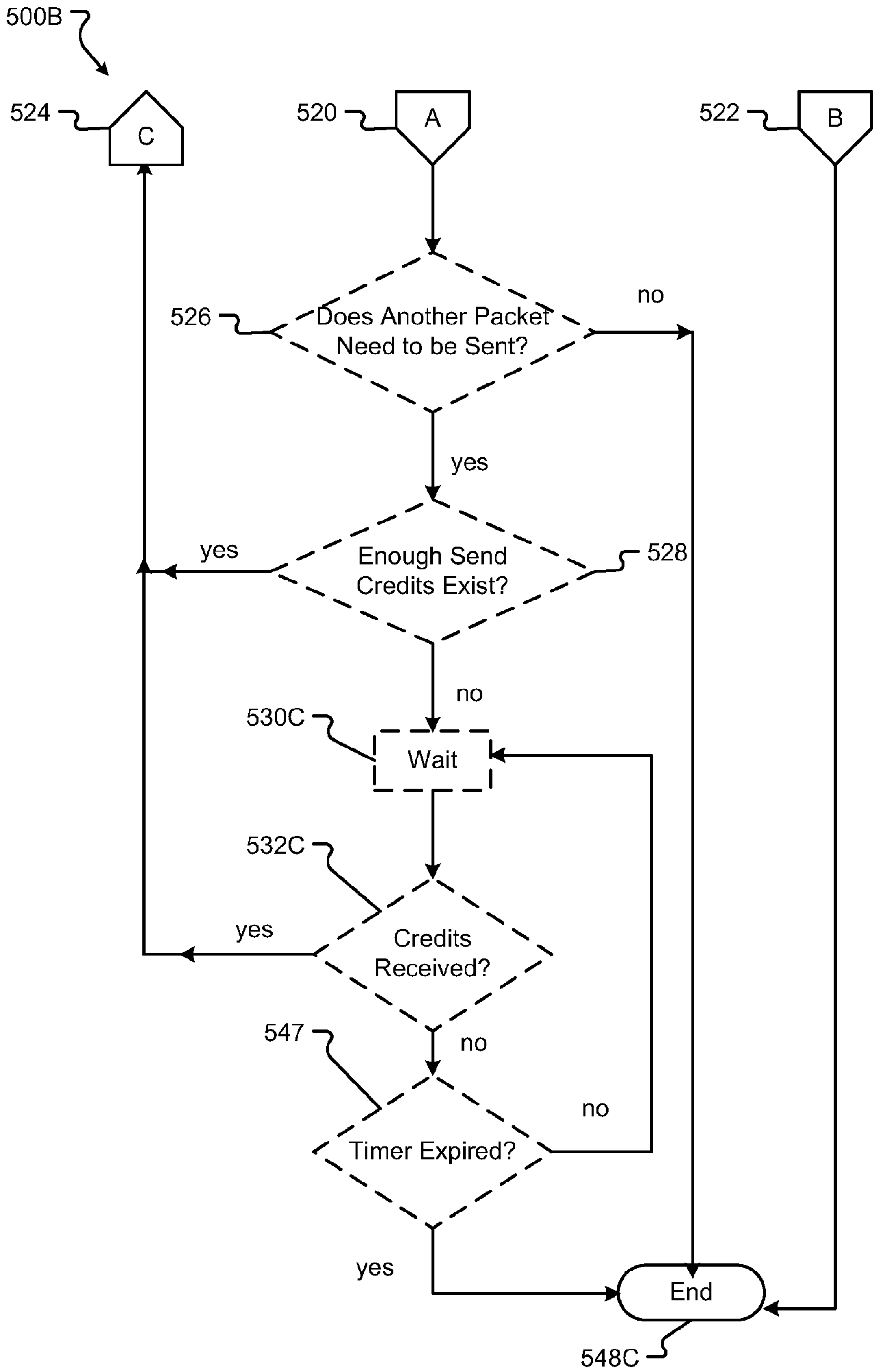

500B
524
C
520
A
522
B
526
Does Another Packet Need to be Sent?
no
yes
yes
Enough Send Credits Exist?
528
no
530C
Wait
532C
yes
Credits Received?
547
no
no
Timer Expired?
yes
End
548C

Fig. 5C

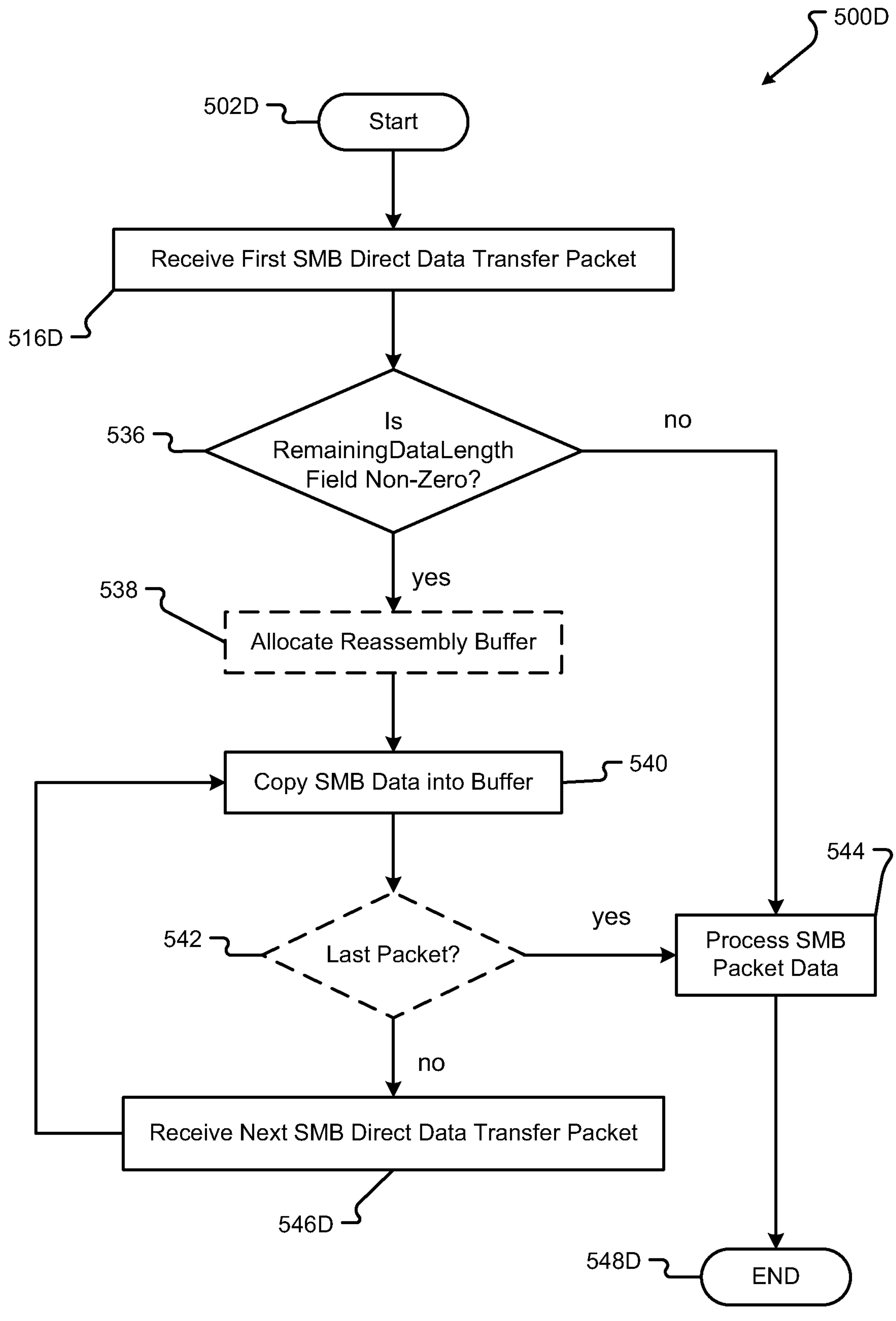

500D
502D
Start
Receive First SMB Direct Data Transfer Packet
516D
536
Is RemainingDataLength Field Non-Zero?
no
yes
538
Allocate Reassembly Buffer
Copy SMB Data into Buffer
540
544
542
Last Packet?
yes
Process SMB Packet Data
no
Receive Next SMB Direct Data Transfer Packet
546D
548D
END

Fig. 5D

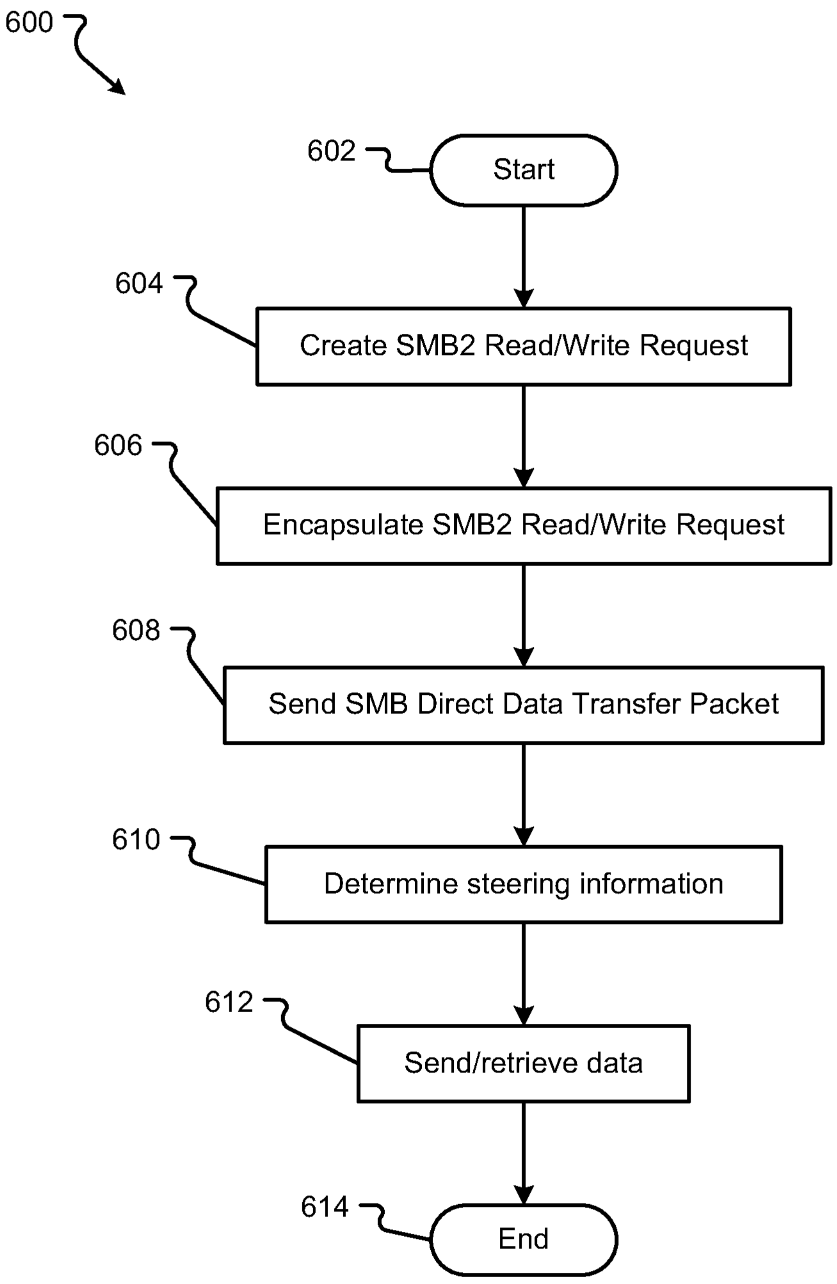

600
602
Start
604
Create SMB2 Read/Write Request
606
Encapsulate SMB2 Read/Write Request
608
Send SMB Direct Data Transfer Packet
610
Determine steering information
612
Send/retrieve data
614
End

Fig. 6

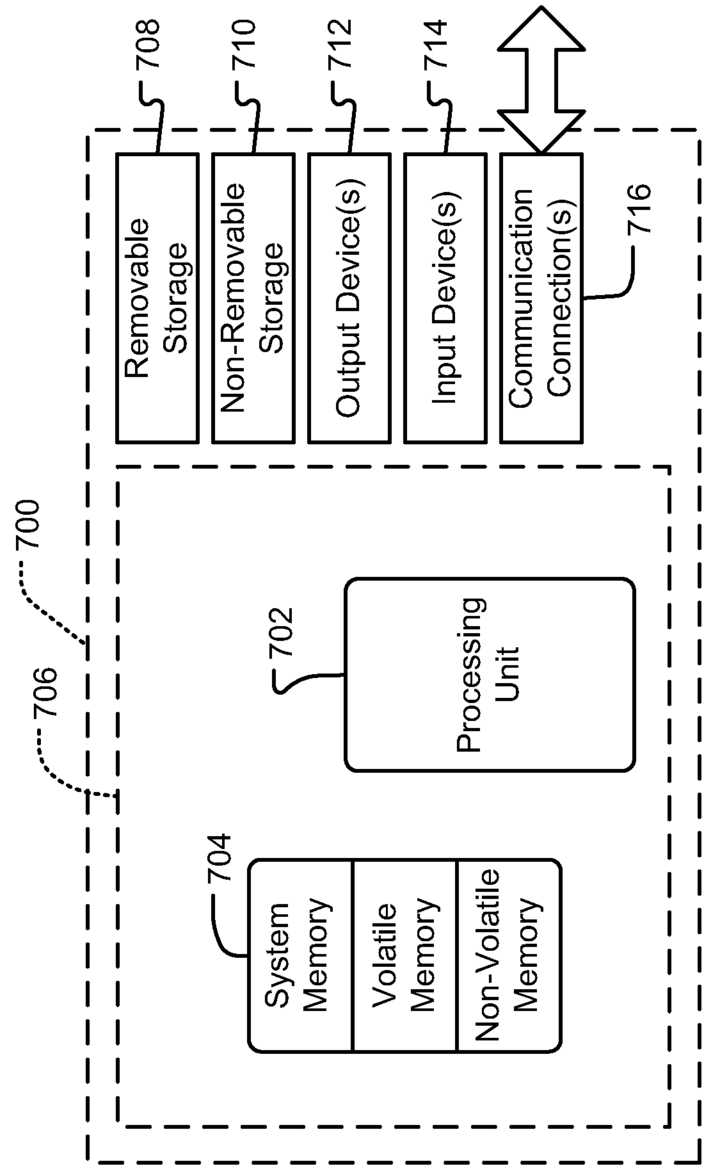

700
706
708
710
712
714
716
702
704
Removable Storage
Non-Removable Storage
Output Device(s)
Input Device(s)
Communication Connection(s)
Processing Unit
System Memory
Volatile Memory
Non-Volatile Memory

Fig. 7