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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
9 March 2017 (09.03.2017)

WIPO | PCT

(10) International Publication Number
WO 2017/040940 A1

(51) International Patent Classification:
H04L 1/18 (2006.01) *H04W 4/00* (2009.01)
H04L 29/08 (2006.01) *H04L 29/06* (2006.01)

(21) International Application Number:
PCT/US2016/050123

(22) International Filing Date:
2 September 2016 (02.09.2016)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
62/213,335 2 September 2015 (02.09.2015) US
62/213,388 2 September 2015 (02.09.2015) US

(71) Applicant: **CONVIDA WIRELESS, LLC** [US/US]; 200 Bellevue Parkway, Suite 300, Wilmington, DE 19809-3727 (US).

(72) Inventors: **LI, Xu**; 28 Ashford Drive, Plainsboro, NJ 08536 (US). **DONG, Lijun**; 6085 African Holly Trail, San Diego, CA 92130 (US). **LY, Quang**; 115 Sterling Drive, North Wales, PA 19454 (US). **CHEN, Zhuo**; 1397 Parish Avenue, Claymont, DE 19703 (US). **RAHMAN, Shamim, Akbar**; 6704 Conklin Road, Cote St. Luc, QC H4V 1B8 (CA). **LU, Guang**; 1 Mendel Crescent, Thornhill, ON L4J 9B6 (CA). **WANG, Chonggang**; 9 Carlyle Court, Princeton, NJ 08540 (US).

(74) Agents: **SAMUELS, Steven, B.** et al.; Baker & Hostetler LLP, Cira Centre, 12th Floor, 2929 Arch Street, Philadelphia, PA 19104-2891 (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, BN, 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, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, 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,

[Continued on next page]

(54) Title: IMPROVED BLOCK TRANSFER OPERATION IN COAP PROTOCOL

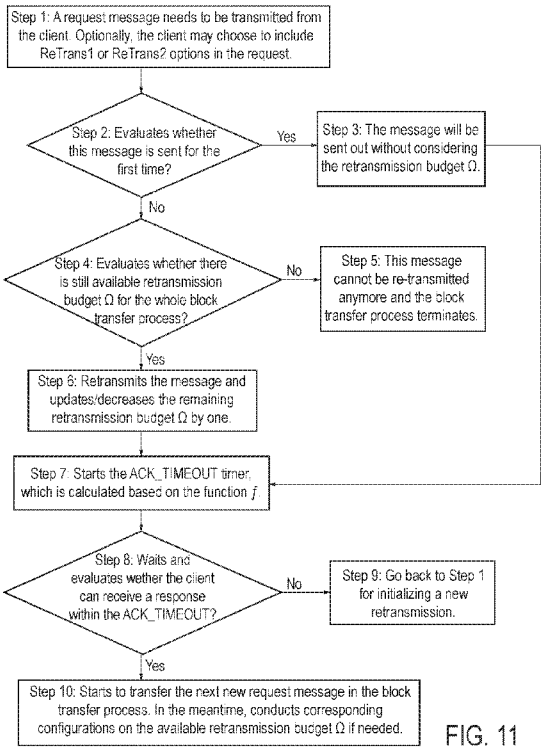


FIG. 11

(57) Abstract: Modifications are made to the existing CoAP block transfer mechanism to minimize waste associated with excess load and congestion and excess message and block transfers during transfers of resources that are dynamically changing. In one aspect, a retransmission budget is introduced to limit the number of attempted message retransmissions. In another aspect, a retransmission time function is introduced to determine when to retransmit a message. In yet another aspect, two new Block Options are introduced to coordinate retransmission operations between the client and the server. Additionally, in an aspect, two new Block Options are introduced to indicate which range of blocks of a requested resource have changed since last retrieval. In another aspect, each block of a resource will have an associated ETag, and three new Block Options are introduced to indicate which specific blocks have changed since last retrieval.

WO 2017/040940 A1

GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, 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, KM, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

IMPROVED BLOCK TRANSFER OPERATION IN COAP PROTOCOL

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims benefit under 35 U.S.C. §119(e) of U.S. Provisional Patent Application No. 62/213,335, filed on September 2, 2015, and titled "Integrated Message Retransmission for Block Transfer Operation in CoAP Protocol," and Provisional U.S. Patent Application No. 62/213,388, filed on September 2, 2015, and titled "Enhanced CoAP Block Transfer Operation on Dynamically-Changing Resources," the content of which is incorporated by reference herein in its entirety.

BACKGROUND

[0002] The Internet Engineering Task Force (IETF) Constrained RESTful Environments (CoRE) Working Group (IETF CoRE) has developed Constrained Application Protocol (CoAP) (RFC7252). CoAP is an application protocol for use with constrained nodes and constrained networks, such as wireless-sensor networks and other Internet of Things (IoT) systems.

[0003] The interaction model of CoAP is similar to the client/server model of Hypertext Transfer Protocol (HTTP). CoAP is a single protocol that uses a two-layer approach, as shown in FIG. 1: a CoAP Messages Layer for handling UDP, though with different reliability requirements than typical UDP; and a Request/Response Layer that uses Methods – GET, PUT, POST, DELETE – and Request/Response codes for handling the asynchronous nature of the network interactions. In implementation, both the messages layer and the request/response layer are features of the CoAP header.

[0004] CoAP defines four types of messages: CON, NON, ACK, and RST. A Confirmable message (CON) is retransmitted using a default timeout and exponential back-off between retransmissions until the recipient sends back an Acknowledgement message (ACK) with the same Message ID. FIG. 2 shows an example of a CON with ACK of the same Message ID 0x7d34. A Non-confirmable message (NON) does not require reliable transmission. For example, each single measurement out of a stream of sensor data can be sent as a NON. NONs are not acknowledged, but still have a Message ID for duplicate detection. An example of a NON with Message ID 0x01a0 is shown in FIG. 3. An ACK is used to acknowledge receipt of a CON. A Reset message (RST) may be used when a recipient is not able to process a NON.

[0005] CoAP defines four methods: GET, POST, PUT, and DELETE.

[0006] The GET method retrieves a representation for the information that currently corresponds to the resource identified by the request Uniform Resource Identifier (URI). Upon success, a 2.05 (Content) or 2.03 (Valid) response code should be present in the response.

[0007] The POST method requests that the representation enclosed in the request be processed. The actual function performed by the POST method is determined by the origin server and dependent on the target resource. POST usually results in a new resource being created or the target resource being updated.

[0008] The PUT method requests that the resource identified by the request URI be updated or created with the enclosed representation. If a resource exists at the request URI, the enclosed representation should be considered a modified version of that resource, and a 2.04 (Changed) response code should be returned. If no resource exists, then the server may create a new resource with that URI, resulting in a 2.01 (Created) response code.

[0009] The DELETE method requests that the resource identified by the request URI be deleted.

[0010] Regarding the CoAP Request/Response Layer shown in FIG. 1, CoAP request and response semantics are carried in CoAP messages. A Token is used to match responses to requests independently from the underlying messages. Typically, there are two cases: a synchronous case, and an asynchronous case.

[0011] FIG. 4 shows an example of the synchronous case. In the synchronous case, when a request is carried in a CON, the response to this request will be directly carried in the resulting ACK. This is called a "piggy-backed response." If the CoAP Client that issued the request in a CON does not receive an ACK within an ACK timer, it will retransmit the request using an exponential back-off algorithm and double the ACK timer to wait for the CoAP Server's Response. If a request is sent in a NON, then the response is sent using a new NON.

[0012] FIG. 5 shows an example of the asynchronous case. In the asynchronous case, when a request is carried in a CON, the client will receive an ACK without a payload. Later, when the requested content becomes available, the server will issue a new CON that includes the response to the client's initial CON request.

[0013] CoAP defines a number of options that may be included in a message. Each Option instance in a message specifies the Option Number of the defined CoAP Option, the length of the Option Value, and the Option Value itself. Option Value can be empty, opaque, an unsigned integer, or a string. Both requests and responses may include a list of one or

more Options. CoAP defines a single set of Options that are used in both requests and responses.

[0014] Options are either "critical" or "elective". An Option is identified by an Option Number, which also provides some additional semantics information. For example, odd numbers indicate a critical Option, while even numbers indicate an elective Option. The difference between these is how an Option unrecognized by an endpoint is handled. Upon reception, unrecognized Options of class "elective" must be silently ignored. Unrecognized Options of class "critical" that occur in a CON request must cause the return of a 4.02 (Bad Option) response. Unrecognized Options of class "critical" that occur in a CON response, or piggy-backed in an ACK, must cause the response to be rejected. Unrecognized Options of class "critical" that occur in a NON must cause the message to be rejected.

[0015] Table 1 displays the properties of CoAP Options using two examples defined in the current CoAP block-transfer specification: Block1 and Block2.

Table 1. Example--Block Option Numbers

No.	C (Critical)	U (Unsafe)	N (NoCacheKey)	R (Repeatable)	Name	Format	Length	Default
23	X	x	-	-	Block2	Unit	0-3 B	(none)
27	X	x	-	-	Block1	Unit	0-3 B	(none)

[0016] The CoAP Options are maintained by an Internet Assigned Numbers Authority (IANA) registry. The IANA policy for future additions to this sub-registry is split into three tiers: range [0-255] is reserved for Options defined by the IETF; range [256-2047] is reserved for commonly used Options with public specifications; and range [2048-64999] is for all other Options, including private or vendor-specific Options.

[0017] CoAP is a Representational State Transfer (RESTful) transfer protocol for use with constrained nodes and networks. Basic CoAP messages work well for small data transfers, such as temperature sensors, light switches, and similar building-automation devices. Occasionally, however, applications need to transfer larger payloads. For example, in an e-health application, it may be desirable to update the firmware of a watch from time to time so that the watch can work properly with newly added features.

[0018] Whereas protocols like HTTP may rely on transport protocols like TCP to perform the low-level work, CoAP is based on simpler datagram transports like UDP. HTTP relies on TCP to divide large payloads into multiple packets and ensure that all packets arrive and are handled in the correct order. In contrast, UDP has a size limit on resource

representations that may be transferred without excessive fragmentation, and although UDP may support larger payloads through IP fragmentation, IP fragmentation is limited to a size of 64 KiB and is not well-suited for constrained applications and networks.

[0019] To avoid problems with IP fragmentation, existing CoAP defines "Block" Options for transferring multiple blocks of information from a resource representation, and the transferring occurs using multiple request-response pairs. For example, if a CON from a client to a server includes a GET request, the server will immediately send back an ACK with the requested payload piggy-backed in the message. In such a manner, this existing block transfer mechanism can be thought of as part of the Messages Layer of CoAP where the Messages Layer is exactly aligned with the Request/Response Layer.

[0020] Accordingly, the concept of "request-response" pairs as defined in block transfers refers to the messaging paradigm of the CoAP Messages Layer: a "request message" typically refers to a CON with an included GET, PUT, or POST request; and a "response message" typically refers to a response piggy-backed in the corresponding ACK. Further, the term "payload" refers to the actual content of a single CoAP message, while the term "body" refers to the entire resource representation being transferred in block-wise fashion.

[0021] As mentioned above, CoAP has two Block Options defined to enable block-wise transfers: "Block1" and "Block2". Generally both Block1 and Block2 Options may be present in both request and response messages, and their usages may vary depending on context. The Block1 Option pertains to the request payload, and the Block2 Option pertains to the response payload. For example, if a Block2 is included in the response of a GET request, it describes the current payload in this response and is referred to as "descriptive usage." However, when a Block2 is included in a GET request, it provides additional control over how the payload will be formed in the response to the GET request and is referred to as "control usage."

[0022] Each Block Option may need to include three items of information: block size ("SZX"); whether more blocks will follow the current message's block ("M"); and the relative number of the block within a sequence of blocks with the given size ("NUM"). The structure of a Block Option is shown in FIG. 6.

[0023] The block size is represented as a three-bit unsigned integer, and the allowed values of SZX are [0-6]. The block size is found by calculating $2^{(SZX+4)}$, making the minimum block size $2^{(0+4)} = 16$ and the maximum block size $2^{(6+4)} = 1024$.

[0024] The "more blocks" flag, M, is represented as a single bit, meaning it may only have a value of 0 or 1. For "descriptive usage," when unset (M = 0), M indicates that

the payload of this message is the last block in the body; when set (M=1), M indicates that there are one or more additional blocks available for transfer. When a Block1 or Block2 is used for "control usage," then M must be set to 0.

[0025] The block number indicates the block number being requested or provided. A message having NUM = 0 indicates the first block of a resource body.

[0026] FIG. 7 shows an example illustrating how to use Block2 in GET requests and responses. In this example, the whole resource body has been retrieved from Server by using three request-response pairs in a block transfer process. FIG. 7 shows Block2 usage in both the descriptive usage context and the control usage context.

[0027] In the descriptive usage context with Block2 Option in a response message, NUM = 0 indicates that the ACK payload contains the first block, M = 1 indicates that more blocks need to be transferred, and SZX = 128 indicates that the block size is 128 bytes. In implementation, SZX = 3 because $2^{(3+4)} = 128$, but this has been simplified for illustration in FIG. 7. Further, because M is set, SZX must be equal to the size of the payload for compilation by the Client.

[0028] In the control usage context with Block2 Option in a request message, NUM = 1 indicates that Client is requesting the second block of the body, M has no function and is set to 0, and SZX = 128 indicates that the Client expects the block size of the next block to be 128. In this context generally, SZX will likely be the size of the previous block, or SZX will be a suggested block size if the Client has not yet received any blocks. As shown in FIG. 7, the Server first proposed a block size of 128 in Message (MID) 1234; the Client then followed the suggestion by setting SZX = 128 in the subsequent GET requests. Similarly, FIG. 8 shows an example illustrating how to use the Block1 Option in PUT requests and responses.

[0029] In CoAP, an ETag is intended for use as an identifier for differentiating between varying representations of the same resource over time. An ETag is typically generated by the server providing the resource and may be generated using any feasible manner and with respect to any feasible variable, including but not limited to version number, checksum, hash value, or time. Block transfers attempting to retrieve resources can benefit by using an ETag Option to ensure retrieval of the correct resource representation. For example, when retrieving a resource representation using GET, the Block2 Option could be used in conjunction with the ETag Option to ensure that the blocks being reassembled at the client side are from the same version of the representation.

SUMMARY

[0030] CoAP's block transfer mechanism is needed when M2M/IoT applications intend to transfer large payloads using CoAP. However, the existing solution may not operate in an optimal way. In particular, the retransmission process of the existing block transfer mechanism does not consider the integrity of all blocks belonging to the same resource body. In accordance with one aspect disclosed herein, a new solution is provided for message retransmission in a CoAP block transfer. The disclosed solution considers the integrity of blocks belonging to the same resource body.

[0031] In particular, embodiments disclosed herein provide modifications to the existing block transfer mechanism to minimize waste associated with excess load and congestion. In one aspect, a retransmission budget is introduced to limit the number of attempted message retransmissions. In another aspect, a retransmission time function is introduced to determine when to retransmit a message. In yet another aspect, two new Block Options are introduced to coordinate retransmission operations between the client and the server.

[0032] Additionally, with the existing block transfer mechanism in CoAP, block transfers can be used to transfer resources that are changing over time by including an ETag Option in the message to indicate the version of the blocks being transferred. Presently, an ETag is defined for the entire resource body, not for individual blocks. Therefore, if any part of the resource body changes, the entire resource body will be assigned a new ETag. If the ETags are not consistent during reassembly, the client may attempt to retrieve the updated values for the blocks it retrieved first. Thus, any blocks having outdated ETags will need to be retransferred – even though the content of those particular blocks may not actually have been modified or changed from one version of the resource to the next. Thus, the existing block transfer process using ETags may not operate efficiently when resources are dynamically changing. According to another solution disclosed herein, a new solution is provided for resource transmission in a CoAP block transfer which operates more efficiently with dynamically changing resources.

[0033] In particular, embodiments disclosed herein provide modifications to the existing block transfer mechanism to minimize waste associated with excess message and block transfers. In an aspect, two new Block Options are introduced to indicate which range of blocks of a requested resource has changed since last retrieval. In another aspect, each block of a resource will have an associated ETag, and three new Block Options are introduced to indicate which specific blocks have changed since last retrieval.

[0034] This Summary is provided to introduce a selection of concepts in a simplified form that are further described below in the Detailed Description. This Summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used to limit the scope of the claimed subject matter. Furthermore, the claimed subject matter is not limited to limitations that solve any or all disadvantages noted in any part of this disclosure.

BRIEF DESCRIPTION OF THE DRAWINGS

[0035] A more detailed understanding may be had from the following description, given by way of example in conjunction with the accompanying drawings wherein:

[0036] FIG. 1 illustrates the interaction model of the CoAP protocol;

[0037] FIG. 2 shows an example message flow in accordance with the CoAP protocol;

[0038] FIG. 3 shows another example message flow in accordance with the CoAP protocol;

[0039] FIG. 4 shows another example message flow in accordance with the CoAP protocol;

[0040] FIG. 5 shows another example message flow in accordance with the CoAP protocol;

[0041] FIG. 6 illustrates the format of CoAP Block Options, Block1 and Block2;

[0042] FIG. 7 shows an example message flow in accordance with using the CoAP Block2 Option;

[0043] FIG. 8 shows an example message flow in accordance with using the CoAP Block1 Option;

[0044] FIG. 9 shows an example message flow in accordance with the CoAP block transfer process;

[0045] FIG. 10 illustrates the basic interaction model of the CoAP block transfer process;

[0046] FIG. 11 is a flow diagram of an example CoAP block transfer process in which one or more embodiments may be implemented;

[0047] FIG. 12 shows an example graphical user interface (GUI) in accordance with an example embodiment;

[0048] FIGs. 13A and 13B illustrate a call flow that shows an example embodiment using a retransmission budget, CoAP Options ReTrans1 and ReTrans2, and an ACK_TIMEOUT calculated by a function *f*;

[0049] FIGs. 14A and 14B show an example message flow in accordance with the CoAP block transfer process;

[0050] FIG. 15 illustrates one embodiment of a method for a CoAP block transfer process using new CoAP Options FirstChange and LastChange;

[0051] FIGs. 16A and 16B illustrate one embodiment of a method for a CoAP block transfer process using new CoAP Options Validation, TagsForAllBlocks, and ReFetch;

[0052] FIG. 17 shows an example graphical user interface (GUI) in accordance with an example embodiment;

[0053] FIG. 18A is a system diagram of an example machine-to-machine (M2M), Internet of Things (IoT), or Web of Things (WoT) communication system in which one or more disclosed embodiments may be implemented;

[0054] FIG. 18B is a system diagram of an example architecture that may be used within the M2M/IoT/WoT communications system illustrated in FIG. 18A;

[0055] FIG. 18C is a system diagram of an example communication network apparatus, such as an M2M/IoT/WoT device, gateway, or server that may be used within the communications system illustrated in FIGs. 18A and 18B; and

[0056] FIG. 18D is a block diagram of an example computing system in which a node of the communication system of FIGs. 18A and 18B may be embodied.

DETAILED DESCRIPTION

Integrated Message Retransmission for Block Transfer Operation in the CoAP Protocol

[0057] CoAP's block transfer mechanism is needed when M2M/IoT applications intend to transfer large payloads using CoAP. Current CoAP retransmission mechanisms treat each message individually without consideration of relationships between different messages, a consideration that is important during a block transfer process where all blocks are part of the same resource body. For example, during a block transfer process, it is possible for a message, which could carry either a request or a response/ACK, to get lost due to the lossy nature of transmissions in M2M/IoT systems. For example, as shown in FIG. 9, when a GET request is sent in a CON message, as shown in MID=1235, it is possible that Client does not receive an ACK within an ACK time period (presently, CoAP specifies an

ACK_TIMEOUT parameter, of which the default initial value is 2 seconds). In such a case, Client will retransmit the message if it has not reached the maximum permitted retransmission times (presently, CoAP specifies a MAX_RETRANSMIT parameter, of which the default value is 4 times) and will use a default exponential back-off algorithm of doubling the ACK timer to wait for Server's response. An issue is that such a retransmission mechanism treats each message individually and does not consider the relationships between different messages, which in fact belong to the same block transfer process and the payloads being transferred belong to the same resource body.

[0058] To further illustrate the deficiencies of the current retransmission mechanisms, consider a block that is supposed to be transferred using a pair of request-response messages. If this block consistently fails to be transferred, meaning the request message or response message gets lost each time an attempted transfer is made, and the corresponding retransmissions of the request message or response message reach the maximum retransmission time defined by the current CoAP specification, there are two possible outcomes. The first outcome ("Case 1") is that the block transfer process will halt. The second outcome ("Case 2") relies on external logic on the client side that is separate from CoAP and issues a new request message to attempt to complete the block transfer operation. The current CoAP specification does not define which outcome is to occur after failure to transmit, and each might cause potential issues.

[0059] For Case 1, consider a use case in which a new firmware update needs to be pushed to a watch, and the firmware will be transferred in 100 blocks through 100 request-response message pairs. The block transfer process, however, could be completely halted due to an unsuccessful transfer for a given block. This halting is especially undesirable in the case that transfer failure of the No. 98 block may void all previous efforts for transferring the Nos. 1-97 blocks.

[0060] For Case 2, assume the client is able to continue the block transfer process by proactively issuing a new request message. To retransfer a previously-undelivered block, the client side can proceed at will issuing new request messages again and again until the previously-undelivered block gets transferred. However, this process makes the current CoAP retransmission mechanism useless. Even if each of the new request messages follows the default CoAP retransmission mechanism, all messaging operation involved for transferring a given block is still at the client's will, becoming uncontrollable.

[0061] Because the existing options for block retransmission of a failed transfer either stop the entire process or behave uncontrollably, the current CoAP block transfer process may not function in an optimal manner, wasting time and resources.

[0062] As further background, the basic interaction model of a CoAP block transfer process is shown in FIG. 10. Typically, there are two situations: (1) when the request is a GET message, the blocks making up a resource move from Server to Client, meaning Client is retrieving the resource from Server; and (2) when the request is a PUT message or POST message, the blocks move from Client to Server. As shown in FIG. 10, "Client" is the party that initializes the GET/PUT/POST requests, and "Server" is the party that responds to those requests. It is beneficial to handle retransmissions of the blocks in an integrated way when transferring blocks belonging to the same block transfer process. Otherwise, a failed transfer might stop the entire process or cause uncontrolled behavior. The solutions disclosed herein handle the retransmission of blocks in an integrated way.

[0063] There are three main aspects to the solution disclosed herein. In a first aspect, a retransmission budget is introduced to limit the number of attempted message retransmissions. In a second aspect, a retransmission time function is introduced to determine when to retransmit a message. In a third aspect, two new Block Options are introduced to coordinate retransmission operations between the client and the server. Each of these aspects is described more fully below.

1. Retransmission Budget

[0064] In a first aspect, a retransmission budget is introduced and integrated into the block transfer process so that the client cannot issue new request messages uncontrollably. The messages used for transferring a given block use the same message ID, meaning that the same message ID will not be reused in other messages for transferring other blocks. Request messages belonging to the same block transfer process will share a retransmission budget, which may be accumulated and consumed during the block transfer process. This retransmission budget would take the place of CoAP's MAX_RETRANSMIT parameter, which is used for individual messages and not specially designed for block transfer processes. When deciding whether or not to send another request message, a critical factor will now be whether there is available retransmission budget. The retransmission budget prohibits a case where a client can uncontrollably issue messages for transferring a given block, and the retransmission budget allows for handling retransmission of a message in a way integrated into the block transfer process.

[0065] In an embodiment, a retransmission budget, defined as Ω, attaches to the whole block transfer process, and whether or not a request message may be retransmitted depends on the current value of Ω. The value of budget Ω may be configured by the following rules, which may be extended or modified based on specific need. The following rules are not intended to be limiting and are described for purposes of example.

[0066] The initial value of Ω may be zero or any customized value. This is the value that Ω will have at the beginning of a block transfer process before any blocks have been transferred between client and server.

[0067] When first attempting to transfer a block, the client may directly issue a request message to start the transfer of the block without considering the retransmission budget Ω.

[0068] For a given block, the request message for transferring the given block should use the same message ID. Thus, the client cannot issue a new message having a different message ID to transfer the given block. Further, all retransmissions of the message attempting to transfer the given block will be decided by the retransmission budget Ω and preclude the client from initializing a retransmission at will.

[0069] Retransmission budget Ω may be accumulated during the transfer process. If a block is successfully transferred during its first attempted transfer, the client may contribute a certain amount of new budget to Ω so that the retransmission opportunities for transferring the current block can by shared by subsequent blocks queued for transfer. Typically, the amount of contribution depends on implementation, and an example could be adding four to the current budget: $\Omega = \Omega + 4$. The four is obtained from CoAP's default maximum retransmission of four for a given message.

[0070] Retransmission budget Ω may be consumed during retransmission. Depending on the value of Ω, a client may retransmit the request message of a block that fails to be transferred. If $\Omega>0$, the request message will be retransmitted. In an embodiment, when the message is retransmitted is dependent on function *f*, discussed below. Once a request message is retransmitted, the budget Ω will decrease by one: $\Omega = \Omega - 1$. If a retransmission fails again, further retransmissions will be performed as long as $\Omega>0$. In addition, once a block is successfully transferred using one or more retransmissions, delivery of the block will not contribute any new budget to Ω.

[0071] Using the above example rules, the maximum total number of retransmissions for a successful block transfer process using the retransmission budget method (every block except the last block is transferred without retransmission, and the last

block consumes all retransmission budget Ω accumulated from successfully transferring the previous blocks) is equal to that of CoAP's default retransmission mechanism (each block is transferred on the fourth retransmission). In an alternative embodiment, in order to enable some flexibility on budget Ω and also to be general, a factor α ($1 \geq \alpha > 0$) can be defined for budget Ω, so that the available budget will be based on the value of $\alpha \times \Omega$, instead of Ω. In another embodiment, and using the previous example where firmware needs to be transferred in 100 blocks through 100 request-response message pairs, if Ω is accumulated to a large value due to the efficient transferring of No. 1-97 blocks but the No. 98 block cannot be transmitted due to bad network conditions, a loss factor α may be associated with Ω so that the client will not naively make too many retransmissions for block No. 98 based on the large value of Ω. In yet another embodiment, a maximum value for Ω may be defined so that the retransmission budget can always be limited to a certain value.

2. Retransmission Time Function

[0072] In a second aspect, a retransmission time function may be introduced to allow for a dynamic response message timeout. The standard CoAP ACK_TIMEOUT parameter is still employed; however, the value of ACK_TIMEOUT will now be calculated by a new retransmission function *f* instead of the default exponential back-off mechanism. Function *f* may be customized based on various needs.

[0073] When a request message is sent, the timer defined by ACK_TIMEOUT will be triggered such that a new retransmission may need to be initialized if the client does not receive an ACK within the ACK_TIMOUT timer. Instead of CoAP's default exponential back-off mechanism, function *f* can be customized based on different needs or scenarios, and any available and appropriate function may be used with or in *f*. When designing function *f*, it is important to avoid the situation that a given request message cannot be retransmitted anyway if the next allowed retransmission time is far in the future (*i.e.*, the ACK_TIMEOUT is too large if it undergoes exponential increases as defined in the current CoAP specification), even if there is still enough retransmission budget Ω. The design of function *f* may be governed by the following rules, which may be extended or modified based on specific need. The following rules are not intended to be limiting and are described for purposes of example.

[0074] In one embodiment, if a request message has already been retransmitted several times, ACK_TIMEOUT could be assigned a larger value than the time interval between the previous retransmissions. Formally, for a given resource R to be transferred through a block transfer, the following mathematical notations are in place: *K* is the total

number of blocks of resource R, and accordingly those K blocks will be transferred through K request-response pairs. In particular, for the i^{th} request message (where $K\text{-}1 \geq i \geq 0$), retransmission time is denoted as j, where $j \geq 0$. ACK_TIMEOUT should be proportional to the value of j. In order to normalize j to be a 0-1 value, and assuming X_1 is system-configured large constant, the following variable Θ_1 ranging from 0-1 is defined:

$$\Theta_1 = \frac{j}{X_1} \tag{1}$$

[0075] In an alternative embodiment, if there is a sufficient retransmission budget (Ω) available, ACK_TIMEOUT could be set to a relatively small value so that the budget can be consumed more aggressively. However, if there is only a limited retransmission budget left, ACK_TIMEOUT could be set to a larger value so that the budget will be consumed more conservatively. To reflect this idea, ACK_TIMEOUT should be inversely proportional to the value of Ω. Assuming X_2 is also a system-configured large constant, ACK_TIMEOUT will be proportional to X_2-Ω. Similarly, in order to normalize X_2-Ω to be a 0-1 value, the following variable Θ_2 ranging from 0-1 is defined:

$$\Theta_2 = \frac{X_2 - \Omega}{X_2} \tag{2}$$

[0076] In another embodiment, if the block currently being transferred is one of the first blocks being transferred in the block transfer process, ACK_TIMEOUT could be set to a larger value because sufficient retransmission budget might not be available, especially if it is gradually accumulated. The ACK_TIMEOUT parameter could then shrink as the block transfer process continued to later blocks. To reflect this idea, ACK_TIMEOUT should be inversely proportional to the value of i. Assuming X_3 is a system-configured large constant (in fact, X_3 could be set equal to K), ACK_TIMEOUT will be proportional to X_3-i. Similarly, in order to normalize X_3-i to be a 0-1 value, the following variable Θ_3 ranging from 0-1 is defined:

$$\Theta_3 = \frac{X_3 - i}{X_3} \tag{3}$$

[0077] Other embodiments might combine any or all of formulas (1), (2), and (3) to contribute weight to function f. One embodiment defines f with respect to retransmission budget, block number, and retransmission attempt. Given the current value of Ω, for the j^{th}

retransmission of the i^{th} block of Resource R, ACK_TIMEOUT for this retransmission can be calculated by $f(i, j, \Omega)$, which is given by:

$$f(i,j,\Omega) = (\alpha \times \Theta_1 + \beta \times \Theta_2 + \gamma \times \Theta_3) \times T_B$$
$$= (\alpha \times \frac{j}{X_1} + \beta \times \frac{X_2 - \Omega}{X_2} + \gamma \times \frac{X_2 - i}{X_3}) \times T_B \quad (4)$$

where T_B is the pre-configured baseline value for ACK_TIMEOUT. Each of factors (Θ_1-Θ_3) will have a weight affecting the final value of $f(i, j, \Omega)$, and one embodiment sets $\alpha=\beta=\gamma=1/3$, for example.

3. New CoAP Block Options

[0078] In a third aspect of the disclosed solution, new CoAP block options may be introduced to allow the client and server to coordinate retransmission instead of allowing only the client to control retransmission. These new block options may be referred to as ReTrans1 and ReTrans2. Such coordination will enable more flexibility for block transfers. The ReTrans1 option is used for the client's retransmission budget, while ReTrans2 is used for the server's expected or desired value of the client's retransmission budget.

[0079] When the ReTrans1 option is in a request message from a client, it indicates the total available retransmission budget Ω for the current block transfer process. The client may proactively indicate ReTrans1 to the server, or the client may provide ReTrans1 at the server's request. When ReTrains1 is in included in a response message from a server, it indicates the server's request for the client's available retransmission budget Ω; in this case, ReTrans1 must be set to zero. After the client receives the ReTrans1=0 option from the server, the client will be stimulated to include a ReTrans1 option in the next request message. Optionally, the client may ignore the server's request for its available retransmission budget Ω.

[0080] When the ReTrans2 option is in a response message from a server, it indicates the expected or desired value of the retransmission budget Ω for the server. This option allows the server to suggest a way for the client to configure retransmission budget Ω for the block transfer process; however, client may decide not to consider server's opinion. When ReTrans2 is included in a request message from a client, it indicates the client's request for the server's expected or desired value of the retransmission budget Ω; in this case, ReTrans2 must be set to zero.

[0081] Using ReTrans1 and ReTrans2, the client and server may better coordinate their resources for the current block transfer process without affecting CoAP's current block

transfer process. Using ReTrans1, the server may know how many potential retransmissions could be initialized by the client during the current block transfer process and may then use this information to better facilitate related operations on the server side. For example, the server can use this information to estimate the success probability of the entire block transfer process and then decide how to store previously-delivered blocks. Similarly, by using ReTrans2, the retransmissions executed at the client side can take the server's opinion into consideration, reflecting the coordination between the two parties. For example, the client may reduce retransmissions if the server indicates a low expected value of Ω. A low expected value of Ω indicates that the server is more willing to serve other clients that have more-favorable network conditions for successful, more-efficient data delivery. Additionally, apart from conveying retransmission budget-related information, ReTrans1 and ReTrans2 have no effect on the current CoAP Block1 and Block2 options, making backward compatibility feasible.

[0082] Table 2 shows the implementation details of one embodiment of a definition for the new ReTrans1 and ReTrans2 CoAP Options. The proposed options may be included in the No. 29 and No. 31 CoAP block options within the headers of the request and response messages, respectively. Note that assigning Nos. "29" and "31" to these new Options is presented solely for purposes of example and should not be construed to limit the Options to these specific numbers.

Table 2. Implementation Details for ReTrans1 and ReTrans2 Options

No.	C (Critical)	U (Unsafe)	N (NoCacheKey)	R (Repeatable)	Name	Format	Length	Default
29	x	x	-	-	ReTrans1	Unit	0-1 B	(none)
31	x	x	-	-	ReTrans2	Unit	0-1 B	(none)

4. Example Workflow

[0083] FIG. 11 illustrates one embodiment of a client workflow using the various aspects described in this specification. In this example, it is assumed that the client has already chosen to conduct a block transfer process on a Resource R via a number of request-response pairs. In particular, for each of the blocks to be transferred, the client needs to transition through the workflow in FIG. 11 to decide how to handle transmissions and retransmissions of the request message for that block.

[0084] In step 1, the client generates a CON request message with message ID 'X' for sending to the server. The request message may be a GET, PUT, or POST message. If the message is a GET message, the request will indicate which block, Y, is to be retrieved, while if the message is a PUT or POST message, Block Y is already in the payload of Message X. Messaging related to Block Y will be assigned the same message ID, X. In addition, Message X might be a new request message or a previously-failed message that requires retransmission because the client did not receive an ACK within the ACK_TIMEOUT. Regardless, the client may optionally include ReTrans1 and/or ReTrans2 options to attempt to coordinate retransmission budget Ω with the server. The client then moves to step 2.

[0085] In step 2, the client determines if Message X is being sent for the first time or is being retransmitted. If Message X is being sent for the first time, the client may go to step 3. Otherwise, the client goes to step 4 for retransmission.

[0086] In step 3, the client sends Message X for the first time and does not need to consider the retransmission budget Ω. The client then moves to step 7.

[0087] In step 4, the client evaluates whether there is any available retransmission budget Ω for resending Message X. If there is available Ω, then the client moves to step 6. Otherwise, the client moves to step 5.

[0088] In step 5, the block transfer process terminates because Message X could not be retransmitted. Accordingly, the client may need to perform internal housekeeping procedures such as deleting already-retrieved blocks, clearing caches, etc. At this point, the client has exited the workflow. However, it is possible that the entire block transfer process may be restarted by the application layer at another time, which may be handled outside of the CoAP layer.

[0089] In step 6, the client retransmits Message X and consumes some available retransmission budget Ω. Using the aforementioned embodiment, Ω would be reduced by 1. The client then moves to step 7.

[0090] In step 7, the client starts the ACK_TIMEOUT timer for the current transmission, and, using the aforementioned embodiment, the value of ACK_TIMEOUT is calculated by using the retransmission time function *f*. The client then moves to step 8.

[0091] In step 8, after sending Message X, the client waits to receive an ACK during the ACK_TIMEOUT timer. If the client does receive an ACK, then the client moves to step 10. Otherwise, the client moves to step 9.

[0092] In step 9, at this point, the (re-)transmission of message X has not been successful. Then, the client needs to go back to Step 1 for evaluating whether to initialize an additional retransmission for message X.

[0093] In step 10, the client knows that transmission of Message X was successful. If Message X was successfully transferred on its first attempt, then the client will add additional retransmission budget to Ω. Further, if the client has coordinated with the server using ReTrans1 and/or ReTrans2, the client may update and configure Ω accordingly. Then the client will restart this process at step 1 to request the next block in the block transfer process if one exists.

[0094] It is understood that the steps of FIG. 11 may be performed by a client or server, such as the clients and servers illustrated in FIGs. 2-5 and 7-10, which may represent logical entities in a communication network and may be implemented in the form of software (i.e., computer-executable instructions) stored in a memory of, and executing on a processor of, an apparatus of such network, which may comprise one of the general architectures illustrated in FIGs. 18C or 18D as described more fully below. That is, the operations illustrated in FIG. 11 may be implemented in the form of software (i.e., computer-executable instructions) stored in a memory of a network apparatus, such as for example the apparatus or computer system illustrated in FIGs. 18C or 18D, which computer executable instructions, when executed by a processor of the apparatus, perform the steps illustrated in the figure. It is also understood that any transmitting and receiving steps illustrated in these figures may be performed by communication circuitry (e.g., circuitry 34 or 97 of FIGs. 18C and 18D, respectively) of the apparatus under control of the processor of the apparatus and the computer-executable instructions (e.g., software) that it executes.

[0095] The above aspects and embodiments might be associated with a certain amount of node overhead, but the benefits introduced outweigh the overhead costs. As previously mentioned, the newly introduced aspects with retransmission mechanisms involving Ω behave in equivalent worst-case scenarios to those in existing CoAP. The example Ω calculations described above may be implemented using a variable and counter in source code, requiring only a negligible amount of computing storage resources to maintain the value of Ω. Additionally, the aforementioned function *f* for calculating the value of ACK_TIMEOUT incurs minimal time overhead. Finally, from a communication overhead perspective, the new aforementioned block options are lightweight options that can be carried in message headers like other CoAP options. Therefore, included embodiments add great benefit at minimal cost to the current block transfer process.

5. Example Graphical User Interface

[0096] From a protocol stack perspective, the above embodiments run at the application protocol layer below the application layer, and the upper-layer applications and services do not directly interact with the application protocol layer unless cross-layer interaction is implemented. To provide a convenient avenue for configuring any or all of the aforementioned parameters, an embodiment includes a user control panel. In particular, the interface may have a graphical user interface (GUI) that allows a user to input desired values for the parameters during protocol setup and deployment. Such an example GUI can be seen in FIG. 12. For example, depending on the specific context or application scenario, users may use such a GUI to preconfigure the initial value of Ω or the specific form of function *f*. As can be seen in FIG. 12, input boxes may be used to enter the values, and a submit button may be used to configure the client with the entered values.

6. Example Scenario

[0097] FIGs. 13A and 13B illustrate an example of how the disclosed aspects may work in an example scenario.

[0098] In stage 1, Client needs to retrieve Resource A from Server using a block transfer operation. Accordingly, Client uses a message having MID=1 to retrieve the first block, Block 0 (zero), which is indicated in a standard Block2 Option. In addition, Client sets the retransmission budget Ω to an initial value of 0 (zero). Setting Ω may be performed using the above-described GUI.

[0099] In stage 2, because Block 0 was retrieved without any retransmission, Client increases its retransmission budget Ω by 4. Client then attempts to retrieve Block 1 with a new request message having MID=2.

[0100] In stage 3, the request message was lost and so Client cannot receive an ACK within the ACK_TIMEOUT timer. The ACK_TIMEOUT timer is calculated by the retransmission time function *f* as defined earlier in the specification and input using the GUI. Because an ACK was not received, Client checks for available Ω to retransmit the request message MID=2.

[0101] In stage 4, Client determines that there is available Ω and retransmits the request message having MID=2 and reduces Ω by 1 because retransmissions cost 1 unit of budget from Ω, leaving Ω=3.

[0102] In stage 5, Server is experience high access load, so it wishes to know how much retransmission budget Ω is remaining. Accordingly, Server includes a ReTrans1 option in the response message to MID=2.

[0103] In stage 6, Server's response message was lost, so Client did not receive an ACK within the ACK_TIMEOUT timer.

[0104] In stage 7, Client retransmits the message having MID=2 and reduces Ω by 1, leaving Ω=2.

[0105] In stage 8, Block 1 is delivered to Client due to the retransmission. Client then starts to retrieve Block 2 using request message with MID=3 and includes a ReTrans1option in message with MID=3 to answer Server's query, and because Ω=2, ReTrans1=2.

[0106] The block transfer process continues in such a fashion until either the last block is retrieved by Client or the block transfer process is halted.

[0107] As with the steps of FIG. 11, it is understood that the steps of FIGs. 13A and 13B may be performed by a client and server, such as the clients and servers illustrated in FIGs. 2-5 and 7-10, which may represent logical entities in a communication network and may be implemented in the form of software (i.e., computer-executable instructions) stored in a memory of, and executing on a processor of, an apparatus of such network, which may comprise one of the general architectures illustrated in FIGs. 18C or 18D as described more fully below. That is, the operations illustrated in FIGs. 13A and 13B may be implemented in the form of software (i.e., computer-executable instructions) stored in a memory of a network apparatus, such as for example the apparatus or computer system illustrated in FIGs. 18C or 18D, which computer executable instructions, when executed by a processor of the apparatus, perform the steps illustrated in the figure. It is also understood that any transmitting and receiving steps illustrated in FIGs. 13A and 13B may be performed by communication circuitry (e.g., circuitry 34 or 97 of FIGs. 18C and 18D, respectively) of the apparatus under control of the processor of the apparatus and the computer-executable instructions (e.g., software) that it executes.

Enhanced CoAP Block Transfer Operation on Dynamically-Changing Resources

[0108] In the current CoAP protocol, block-wise transfers may be used to transfer resources that are changing over time by including an ETag Option in the message to indicate the version of the blocks being transferred. For this to work correctly, the server should include the ETag Option in each response carrying a block. If an ETag Option is being used, the client should compare the ETags of those blocks it received in order to reassemble the resource. If the ETags are not consistent during reassembly, the client may attempt to retrieve the updated values for the blocks it retrieved first.

[0109] Currently, in the CoAP protocol, an ETag is defined for an entire resource body being retrieved, not for individual blocks. Therefore, if any part of the resource body changes during the block transfer process, the entire resource body will be assigned a new ETag. An example of this process is shown in FIGs. 14A and 14B, in which the ETag for a resource changes from 001 to 002 because of a change to Block 4 during resource transfer. As mentioned, all retrieved blocks must have identical ETags for reassembly. Thus, after all blocks have been transferred, any blocks having outdated ETags will need to be retransferred. In FIGs. 14A and 14B, this can be seen by the refetching of Block 0 and Block 1, which have ETag 001 on the Client side instead of the latest ETag, ETag 002. But in this example, Block 0 and Block 1 were not modified after transfer. However, with the existing ETag mechanism, those blocks were still required to be refetched for assembly purposes.

[0110] As a result, it can be seen that the existing block transfer mechanism cannot transfer resources in an efficient way if resources are dynamically changing overtime. For example, as shown in FIGs. 14A and 14B, due to the fact that an ETag is associated with the whole resource body, Blocks 0 and 1 need to be refetched even if the modified or updated part of the resource does not involve those blocks, but involves Block 4 instead. Such refetching leads to unnecessary communication costs, the effect of which will further deteriorate if the number of blocks to be retransferred is large.

[0111] An example use case for the improved methods disclosed herein involves a client attempting to retrieve a resource, Resource A, from a server. In the example, Resource A is comprised of blocks of data, numbered sequentially, and is transferred using a block transfer operation. During the transfer, Resource A is dynamically changing, meaning that at least some of its blocks have been modified on the server. To avoid CoAP's current unnecessary refetching of blocks, embodiments are proposed that can support a more efficient block transfer process of dynamically changing resources.

[0112] There are multiple aspects to the solution disclosed herein. In an aspect, two new Block Options are introduced to indicate which range of blocks of the requested resource has changed since last retrieval. In another aspect, each block of a resource will have an associated ETag, and three new Block Options are introduced to indicate which specific blocks have changed since last retrieval. Each of these aspects is described more fully below.

[0113] An aspect works with the existing CoAP block transfer method of assigning a single ETag for an entire resource body with the client being able to assemble all of the transferred blocks of the resource body only if all blocks have the same ETag. To combat the inefficiencies of the existing block transfer process, two new block options are introduced to

indicate the number range of blocks that have been updated during an ongoing block transfer process. These options are carried in the response messages from the server to the client to inform the client which blocks of the resource have changed on the server. Accordingly, the client may refetch only the blocks that fall into the indicated range instead of refetching all previously-delivered blocks due to the application of a new ETag to the entire resource body on the server side (any update on the server side leads to all blocks of the resource being assigned a new ETag). Further, on the client side, the client may also override the old ETag with the new ETag for the blocks that have already been fetched by the client and have not been modified during a resource update on the server (these blocks have not changed, so the client just needs to assign them the new ETag).

[0114] In one embodiment, Block options FirstChange and LastChange are used to indicate the number range of blocks that have been modified on the server. When Resource A is modified on the server during an ongoing block transfer process, the server may determine a block number range indicating the tight range covering all blocks that have been modified during the transfer process. In particular, the FirstChange option may indicate the block number of the first changed block, and the LastChange option may indicate the block number of the last changed block. The "tight" range is the lower-bound range, which is just wide enough to cover all blocks that have been modified during the update on the server. Because all blocks falling into this range will probably be refetched by the client, a tight range may incur the additional transmission overhead in a minimal way.

[0115] In an embodiment, if there is an update on Resource A, the server will automatically include the FirstChange and LastChange options in a response message to the client during the ongoing block transfer process of Resource A. Further, depending on the level of completeness of the entire block transfer process, the client may have different actions after receiving the FirstChange and LastChange options.

[0116] In one embodiment for example, if the client is retrieving block N, where N<FirstChange, then the client does not need to refetch any blocks because the client has not yet started to retrieve the changed blocks in range [FirstChange, LastChange]. If the client is retrieving block N, where FirstChange<N<LastChange, then the client will need to refetch the blocks between FirstChange and N because those have just been modified on the server. If the client is retrieving block N, where LastChange<N, then the client will need to refetch all blocks between [FirstChange, LastChange] because those blocks have already been retrieved by the client but have recently been modified on the server.

[0117] Regardless of the situation, the client must make sure that all fetched blocks have the most recent ETag. For example, even if the client does not need to refetch any blocks from the server after a resource update, the client still needs to update the client-side ETags to the newest server-side ETag for all fetched blocks. In addition, to allow the block transfer process to operate in its existing manner, updating the ETags should be done at the latest stage of the entire transfer process, i.e., when the last block has been retrieved.

[0118] Table 2 shows the implementation details of one embodiment of a definition for the new FirstChange and LastChange CoAP Options. The proposed options may be included in the No. 41 and No. 43 CoAP block options within the headers of the response messages, respectively. Note that assigning Nos. "41" and "43" to these new Options is presented solely for purposes of example and should not be construed to limit the Options to these specific numbers.

Table 2. Implementation Details for FirstChange and LastChange Options

No.	C (Critical)	U (Unsafe)	N (NoCacheKey)	R (Repeatable)	Name	Format	Length	Default
41	x	x	-	-	FirstChange	Unit	0-1 B	(none)
43	x	x	-	-	LastChange	Unit	0-1 B	(none)

[0119] FIG. 15 illustrates one embodiment of a method using the FirstChange and LastChange options.

[0120] In step 1, Client needs to retrieve Resource A from Server using a block transfer operation. Accordingly, Client will start to retrieve blocks of Resource A, beginning from Block 0 (zero).

[0121] In step 2, the block transfer process continues and Client successfully retrieves Blocks 0 through 13, with all retrieved blocks having ETag=001.

[0122] In step 3, Block 7, Block 9, and Block 12 are updated on Server, and Resource A is assigned a new ETag: ETag=002 (all blocks of Resource A now have ETag 002). The server then determines that FirstChange=7 and LastChange=12 and sends this information in a response message to Client.

[0123] In step 4, Client receives the range of changed blocks and determines that it already fetched the blocks covered by [FirstChange, LastChange] and so must refetch those blocks. Note that the refetch process is conducted after the current ongoing block transfer

process is complete. Finishing the current transfer may be indicated with a 0 (zero) in the M bit of the Block2 Option carried in a response message, as can be seen in the ACK of the message having MID=15.

[0124] In step 5, Client successfully refetches Blocks 7 through 12 and overwrites the ETags of all other retrieved blocks from ETag=001 to ETag=002 to match the ETag=002 of the refetched blocks.

[0125] In step 6, Client completes the block transfer process by reassembling the resource body.

[0126] In another aspect, each block of a resource has an individual ETag instead of an ETag being assigned to the entire resource body. As a result, it is possible that each block may have a different ETag when the client attempts to reassemble the resource. Therefore, the client relies on the server to evaluate whether the blocks can be reassembled because the server is the party making updates to the resource. Otherwise, the client would not know whether all retrieved blocks belong to the same version of the resource body

[0127] In an embodiment, the resource may be divided into uniform block sizes for simpler ETag implementation. For example, Resource A may be divided into blocks of 16 bytes, or any other customized size. Each block having the defined uniform/minimum size will be assigned an individual ETag, and if a block is modified, a new ETag will be assigned to it. Regardless of the block size chosen for ETag assignment, the client and server may still agree upon different block sizes for transferring the resource, leaving Block1 and Block2 CoAP Options unchanged while assigning ETags to 16-byte blocks.

[0128] In an embodiment, while any suitable ETag update method may be used, it is desirable to use a method utilizing a time-stamp, allowing the client and server to more easily identify when blocks have last changed.

[0129] In an embodiment, it may be desirable to design and implement a common middleware between the CoAP layer and application layer. Such a middleware layer may help the CoAP layer identify any partial resource updates made by an application and assign new ETags to blocks, which in fact also makes the CoAP layer more light-weight. Further, a middleware layer may alleviate the need for the application layer to be involved in ETag assignment or resource transfer, which is desirable from a protocol stack perspective. The application layer should not know that a given resource is in fact transferred through a block transfer process on the CoAP layer.

[0130] An example embodiment of a block transfer process using this new aspect is as follows. After the client completes retrieval of the last block of Resource A from the

server, it may perform a process called "Resource Integrity Validation" (RIV). If the client performs RIV, it will send to the server an additional GET request carrying the ETags of all 16-byte blocks that have been retrieved, which is then used by the server to validate whether the blocks can be assembled by the client. When the server receives the ETags, it will compare them with the 16-byte blocks of Resource A currently held by the server. Through this comparison, the server will determine which 16-byte blocks need to be redelivered to the client based on ETag discrepancies. Then the server will give the client this list of blocks so that the client can retrieve only those needed 16-byte blocks and assemble Resource A. The RIV process may be executed multiple times until the client holds all of the latest blocks. Multiple RIV executions may be required if Resource A is changing at a rapid pace.

[0131] In an embodiment, the RIV process may be implemented via three new message options: Validation, TagsForAllBlocks, and ReFetch. The client may set Validation equal to 1 to indicate that it is asking the server to validate the ETags of the client's received blocks. The TagsForAllBlocks option may be used to carry the ETags of all retrieved blocks. Alternatively, the ETags may be sent as the payload of a request message, which might be desirable if there are many blocks and ETags involved. After receiving the validation request, the server will determine which blocks have been modified during the current block transfer process and will send to the client all block numbers that should be refetched. To facilitate this, a ReFetch option may be used to carry the modified block numbers. Alternatively, the block numbers may be sent as the payload of a response message.

[0132] In addition, the ReFetch option may also be leveraged in the first aspect of the solution, as disclosed above, so that instead of returning the range numbers indicated by FirstChange and LastChange options, the server may use the ReFetch option to send back the list of block numbers to the client in a more fine-grained manner.

[0133] Table 3 shows the implementation details of one embodiment of a definition for the new Validation, TagsForAllBlocks, and ReFetch CoAP Options. The proposed options may be included in the No. 45, No. 47, and No. 49 CoAP block options within the headers of the response messages, respectively. Note that assigning Nos. "45", "47", and "49" to these new Options is presented solely for purposes of example and should not be construed to limit the Options to these specific numbers.

Table 3. Implementation Details for Validation, TagsForAllBlocks, and Refetch Options

No.	C (Critical)	U (Unsafe)	N (NoCacheKey)	R (Repeatable)	Name	Format	Length	Default

45	x	x	-	-	Validation	Unit	0-1 B	(none)
47	x	x	-	-	TagsForAllBlocks	String	0-8 B	(none)
49	x	x	-	-	ReFetch	String	0-8 B	(none)

[0134] FIGs. 16A and 16B illustrate one embodiment of a method using the Validation, TagsForAllBlocks, and ReFetch options.

[0135] In step 1, Client needs to retrieve Resource A from Server using a block transfer operation. Accordingly, Client will start to retrieve blocks of Resource A, beginning from Block 0 (zero).

[0136] In step 2, the block transfer process continues and Client successfully retrieves Blocks 0 through 13.

[0137] In step 3, the server does a partial update on Resource A, and Block 7 has been updated.

[0138] In step 4, after obtaining the last block of Resource A, Block 14, Client begins the RIV process by sending an additional GET request with Validation=1 and TagsForAllBlocks holding the ETags of all the blocks of Resource A that Client is holding. Alternatively, Client may send the ETags in the payload part of the message.

[0139] In step 5, because Server updated Resource A, Server determines that Block 7 needs to be redelivered to Client and sends a response message with ReFetch holding "7" to indicate Block 7 to Client. In general, it is possible that multiple blocks have been modified by the server, and all may be included in the ReFetch option sent back to Client.

[0140] In step 6, Client receives the ReFetch message and begins to retrieve Block 7 again. Block 7 is calculated based on a 16-byte block size in this embodiment, so Client will indicate the transferring size as 16 in the standard Block2 CoAP Option. If more than two blocks need to be refetched, Client will retrieve them in ascending order based on block number.

[0141] In step 7, after the refetch is complete, Client, similar to step 4, needs to repeat the RIV process to check if Server has made any further updates on Resource A because Server may still make further updates on Resource A during the refetch. Client sends a GET request with Validation=1 and TagsForAllBlocks holding the ETags of all the blocks of Resource A that Client is holding.

[0142] In step 8, Server receives the GET request and determines that Client is holding the latest version of all blocks. Server then sends a response message with an empty ReFetch option to indicate that no further refetching is required for Resource A's assembly by Client.

[0143] In step 9, Client receives the response message indicating that no further refetching is required and assembles Resource A, completing the block transfer process.

[0144] It is understood that the stages of FIGs. 15 and 16A-B may be performed by a client or server, such as the clients and servers illustrated in FIGs. 2-5, 7-10, and 13A-16B, which may represent logical entities in a communication network and may be implemented in the form of software (i.e., computer-executable instructions) stored in a memory of, and executing on a processor of, an apparatus of such network, which may comprise one of the general architectures illustrated in FIGs. 18C or 18D as described more fully below. That is, the operations illustrated in FIGs. 15 and 16A-B may be implemented in the form of software (i.e., computer-executable instructions) stored in a memory of a network apparatus, such as for example the apparatus or computer system illustrated in FIGs. 18C or 18D, which computer executable instructions, when executed by a processor of the apparatus, perform the steps illustrated in the figure. It is also understood that any transmitting and receiving steps illustrated in these figures may be performed by communication circuitry (e.g., circuitry 34 or 97 of FIGs. 18C and 18D, respectively) of the apparatus under control of the processor of the apparatus and the computer-executable instructions (e.g., software) that it executes.

[0145] FIG. 17 illustrates one embodiment of a Graphical User Interface (GUI), which may be used to select a method of performing CoAP block transfers on dynamically changing resources. As discussed above, aspects are discussed herein for more efficient CoAP block transfer operations when operating on dynamically-changing resources. An aspect uses the FirstChange and LastChange options, and another aspect uses the Validation, TagsForAllBlocks and ReFetch options. Accordingly, a GUI may be provided for the disclosed solutions to allow a user to select which approach is going to be adopted in a given scenario. As shown in the example GUI of FIG. 17, a user may select between three options: (1) using the default CoAP block transfer operation; (2) using the disclosed FirstChange and LastChange options; or (3) using the disclosed Validation, TagsForAllBlocks, and ReFetch options. Once a specific approach is selected, the remaining operations may be automatically executed without further user intervention.

[0146] Embodiments contemplated herein include the following:

1. An apparatus comprising a processor, a memory, and communication circuitry, the apparatus being connected to a communications network via its communication circuitry, the apparatus further comprising computer-executable instructions stored in the memory of the apparatus which, when executed by the processor of the apparatus, cause the apparatus to:

send, via the communication network to an other apparatus on the network, a CoAP message indicating a request to retrieve a first block of a resource from the other apparatus;

receive, via the communication network from the other apparatus on the network, a CoAP message including the first block of the resource and a first ETag associated with the resource indicating a first version of the resource;

send, via the communication network to the other apparatus on the network, a CoAP message indicating a request to retrieve a second block of the resource from the other apparatus; and

receive, via the communication network from the other apparatus on the network, a CoAP message including the second block of the resource and a second ETag associated with the resource indicating a second version of the resource, the message indicating a range of blocks of the resource that have changed from the first version to the second version.

2. The apparatus of embodiment 1, wherein the range of blocks of the resource that have changed from the first version to the second version is indicated in a CoAP Option field of the message.

3. The apparatus of any of embodiments 1 or 2, wherein the range of blocks of the resource that have changed from the first version to the second version is indicated in a payload of the message.

4. An apparatus comprising a processor, a memory, and communication circuitry, the apparatus being connected to a communications network via its communication circuitry, the apparatus further comprising computer-executable instructions stored in the memory of the apparatus which, when executed by the processor of the apparatus, cause the apparatus to:

receive, via the communication network from an other apparatus on the network, a CoAP message indicating a request to retrieve a first block of a resource;

send, via the communication network to the other apparatus on the network, a CoAP message including the first block of the resource and a first ETag associated with the resource indicating a first version of the resource;

receive, via the communication network from the other apparatus on the network, a CoAP message indicating a request to retrieve a second block of the resource; and

send, via the communication network to the other apparatus on the network, a CoAP message including the second block of the resource and a second ETag associated with the resource indicating a second version of the resource, the message indicating a range of blocks of the resource that have changed from the first version to the second version.

5. The apparatus of embodiment 4, wherein the range of blocks of the resource that have changed from the first version to the second version is indicated in at least one CoAP Option field of the message.

6. The apparatus of any of embodiments 4 or 5, wherein the range of blocks of the resource that have changed from the first version to the second version is indicated in a payload of the message.

7. An apparatus comprising a processor, a memory, and communication circuitry, the apparatus being connected to a communications network via its communication circuitry, the apparatus further comprising computer-executable instructions stored in the memory of the apparatus which, when executed by the processor of the apparatus, cause the apparatus to:

send, via the communication network to an other apparatus on the network, a CoAP message indicating a request to retrieve a first block of a resource from the other apparatus;

receive, via the communication network from the other apparatus on the network, a CoAP message including the first block of the resource and a first ETag associated with the first block indicating a first version of the first block;

send, via the communication network to the other apparatus on the network, a CoAP message indicating a request to retrieve a second block of the resource from the other apparatus;

receive, via the communication network from the other apparatus on the network, a CoAP message including the second block of the resource and a second ETag associated with the second block indicating a first version of the second block;

send, via the communication network to the other apparatus on the network, a CoAP message indicating a request to check the ETags associated with the received blocks against the corresponding blocks of the resource on the other apparatus and including the ETags associated with the received blocks; and

receive, via the communication network from the other apparatus on the network, a CoAP message indicating which blocks of the resource have changed.

8. The apparatus of embodiment 7, wherein the computer-executable instructions further cause the apparatus to retrieve, via the communication network from the other apparatus on the network, the changed blocks of the resource.

9. The apparatus of embodiment 8, wherein the computer-executable instructions further cause the apparatus to assemble the received blocks into the resource.

10. The apparatus of any of embodiments 7 to 9, wherein the request to check the ETags is provided in at least one CoAP Option field of a message.

11. The apparatus of any of embodiments 7 to 10, wherein the ETags associated with the received blocks for which the request to check the ETags is made are provided in at least one CoAP Option field of a message.

12. An apparatus comprising a processor, a memory, and communication circuitry, the apparatus being connected to a communications network via its communication circuitry, the apparatus further comprising computer-executable instructions stored in the memory of the apparatus which, when executed by the processor of the apparatus, cause the apparatus to:

send, via the communication network to an other apparatus on the network, a plurality of CoAP messages indicating requests to retrieve a plurality of blocks of a resource from the other apparatus;

receive, via the communication network from the other apparatus on the network, a plurality of CoAP messages including the plurality of blocks of the resource, each block of the plurality of blocks having a respective ETag indicating a first version of each of the plurality of blocks;

send, via the communication network to the other apparatus on the network, a CoAP message indicating a request to check the ETags associated with the received blocks against the corresponding blocks of the resource on the other apparatus and including the ETags associated with the received blocks; and

receive, via the communication network from the other apparatus on the network, a CoAP message indicating which blocks of the resource have changed.

Example Environment

[0147] FIG. 18A is a diagram of an example machine-to machine (M2M), Internet of Things (IoT), or Web of Things (WoT) communication system 10 in which one or more disclosed embodiments may be implemented. Generally, M2M technologies provide building blocks for the IoT/WoT, and any M2M device, M2M gateway, M2M server, M2M service platform or other network apparatus may be a component or node of the IoT/WoT as well as an IoT/WoT service layer, etc. Any of the client or server devices illustrated in any of FIGs. 2-5, 7-10, and 13A-16B may comprise a node (*i.e.* network apparatus) of a communication system such as the one illustrated in FIGs. 18A-D.

[0148] As shown in FIG. 18A, the M2M/ IoT/WoT communication system 10 includes a communication network 12. The communication network 12 may be a fixed network (e.g., Ethernet, Fiber, ISDN, PLC, or the like) or a wireless network (e.g., WLAN, cellular, or the like) or a network of heterogeneous networks. For example, the communication network 12 may be comprised of multiple access networks that provide content such as voice, data, video, messaging, broadcast, or the like to multiple users. For example, the communication network 12 may employ one or more channel access methods, such as code division multiple access (CDMA), time division multiple access (TDMA), frequency division multiple access (FDMA), orthogonal FDMA (OFDMA), single-carrier FDMA (SC-FDMA), and the like. Further, the communication network 12 may comprise other networks such as a core network, the Internet, a sensor network, an industrial control network, a personal area network, a fused personal network, a satellite network, a home network, or an enterprise network for example.

[0149] As shown in FIG. 18A, the M2M/IoT/WoT communication system 10 may include the Infrastructure Domain and the Field Domain. The Infrastructure Domain refers to the network side of the end-to-end M2M deployment, and the Field Domain refers to the area networks, usually behind an M2M gateway. The Field Domain and Infrastructure Domain may both comprise a variety of different network apparatuses (e.g., servers, gateways,

devices, and like apparatuses) of the network. For example, the Field Domain may include M2M gateways 14 and devices 18. It will be appreciated that any number of M2M gateway devices 14 and M2M devices 18 may be included in the M2M/ IoT/WoT communication system 10 as desired. Each of the M2M gateway devices 14 and M2M devices 18 are configured to transmit and receive signals, using communications circuitry, via the communication network 12 or direct radio link. A M2M gateway 14 allows wireless M2M devices (e.g. cellular and non-cellular) as well as fixed network M2M devices (e.g., PLC) to communicate either through operator networks, such as the communication network 12 or direct radio link. For example, the M2M devices 18 may collect data and send the data, via the communication network 12 or direct radio link, to an M2M application 20 or other M2M devices 18. The M2M devices 18 may also receive data from the M2M application 20 or an M2M device 18. Further, data and signals may be sent to and received from the M2M application 20 via an M2M service layer 22, as described below. M2M devices 18 and gateways 14 may communicate via various networks including, cellular, WLAN, WPAN (e.g., Zigbee, 6LoWPAN, Bluetooth), direct radio link, and wireline for example. Exemplary M2M devices include, but are not limited to, tablets, smart phones, medical devices, temperature and weather monitors, connected cars, smart meters, game consoles, personal digital assistants, health and fitness monitors, lights, thermostats, appliances, garage doors and other actuator-based devices, security devices, and smart outlets.

[0150] Referring to FIG. 18B, the illustrated M2M service layer 22 in the field domain provides services for the M2M application 20, M2M gateways 14, and M2M devices 18 and the communication network 12. It will be understood that the M2M service layer 22 may communicate with any number of M2M applications, M2M gateways 14, M2M devices 18, and communication networks 12 as desired. The M2M service layer 22 may be implemented by one or more nodes of the network, which may comprises servers, computers, devices, or like apparatuses. The M2M service layer 22 provides service capabilities that apply to M2M devices 18, M2M gateways 14, and M2M applications 20. The functions of the M2M service layer 22 may be implemented in a variety of ways, for example as a web server, in the cellular core network, in the cloud, etc.

[0151] Similar to the illustrated M2M service layer 22, there is the M2M service layer 22' in the Infrastructure Domain. M2M service layer 22' provides services for the M2M application 20' and the underlying communication network 12' in the infrastructure domain. M2M service layer 22' also provides services for the M2M gateways 14 and M2M devices 18 in the field domain. It will be understood that the M2M service layer 22' may

communicate with any number of M2M applications, M2M gateways and M2M devices. The M2M service layer 22' may interact with a service layer by a different service provider. The M2M service layer 22' may be implemented by one or more network apparatuses (*i.e.*, nodes) of the network, which may comprises servers, computers, devices, virtual machines (e.g., cloud computing/storage farms, etc.) or like apparatuses.

[0152] Referring also to FIG. 18B, the M2M service layers 22 and 22' provide a core set of service delivery capabilities that diverse applications and verticals can leverage. These service capabilities enable M2M applications 20 and 20' to interact with devices and perform functions such as data collection, data analysis, device management, security, billing, service/device discovery etc. Essentially, these service capabilities free the applications of the burden of implementing these functionalities, thus simplifying application development and reducing cost and time to market. The service layers 22 and 22' also enable M2M applications 20 and 20' to communicate through various networks 12 and 12' in connection with the services that the service layers 22 and 22' provide.

[0153] The M2M applications 20 and 20' may include applications in various industries such as, without limitation, transportation, health and wellness, connected home, energy management, asset tracking, and security and surveillance. As mentioned above, the M2M service layer, running across the devices, gateways, servers and other apparatuses of the system, supports functions such as, for example, data collection, device management, security, billing, location tracking/geofencing, device/service discovery, and legacy systems integration, and provides these functions as services to the M2M applications 20 and 20'.

[0154] Generally, a service layer, such as the service layers 22 and 22' illustrated in FIGs. 18A and 18B, may be a functional layer within a network service architecture. Service layers are typically situated above the application protocol layer such as HTTP, CoAP or MQTT and provide value added services to client applications. The service layer also provides an interface to core networks at a lower resource layer, such as for example, a control layer and transport/access layer. The service layer supports multiple categories of (service) capabilities or functionalities including a service definition, service runtime enablement, policy management, access control, and service clustering. Recently, several industry standards bodies, e.g., oneM2M, have been developing M2M service layers to address the challenges associated with the integration of M2M types of devices and applications into deployments such as the Internet/Web, cellular, enterprise, and home networks. A M2M service layer can provide applications and/or various devices with access to a collection of or a set of the above mentioned capabilities or functionalities, supported by

the service layer, which can be referred to as a CSE or SCL. A few examples include but are not limited to security, charging, data management, device management, discovery, provisioning, and connectivity management which can be commonly used by various applications. These capabilities or functionalities are made available to such various applications via APIs which make use of message formats, resource structures and resource representations defined by the M2M service layer. The CSE or SCL is a functional entity that may be implemented by hardware and/or software and that provides (service) capabilities or functionalities exposed to various applications and/or devices (i.e., functional interfaces between such functional entities) in order for them to use such capabilities or functionalities. The Third Generation Partnership Project (3GPP) has also defined an architecture for machine-type communications (MTC). In that architecture, the service layer and the service capabilities it provides are implemented as part of a Service Capability Server (SCS). Whether embodied in a DSCL, GSCL, or NSCL of the ETSI M2M architecture, in a Service Capability Server (SCS) of the 3GPP MTC architecture, in a CSF or CSE of the oneM2M architecture, or in some other apparatus of a network, an instance of the service layer may be implemented as a logical entity (e.g., software, computer-executable instructions, and the like) executing either on one or more standalone apparatuses in the network, including servers, computers, and other computing devices or apparatuses, or as part of one or more existing apparatuses. As an example, an instance of a service layer or component thereof may be implemented in the form of software running on a network apparatus (e.g., server, computer, gateway, device or the like) having the general architecture illustrated in FIG. 18C or FIG. 18D described below.

[0155] Further, the methods and functionalities described herein may be implemented as part of an M2M network that uses a Service Oriented Architecture (SOA) and/or a Resource-Oriented Architecture (ROA) to access services.

[0156] FIG. 18C is a block diagram of an example hardware/software architecture of a network apparatus, such as one of the clients or servers illustrated in FIGs. 2-5, 7-10, and 13A-16B which may operate as an M2M server, gateway, device, or other apparatus or node in an M2M network such as that illustrated in FIGs. 18A and 18B. As shown in FIG. 18C, the apparatus 30 may include a processor 32, non-removable memory 44, removable memory 46, a speaker/microphone 38, a keypad 40, a display, touchpad, and/or indicators 42, a power source 48, a global positioning system (GPS) chipset 50, and other peripherals 52. The apparatus 30 may also include communication circuitry, such as a transceiver 34 and a transmit/receive element 36. It will be appreciated that the apparatus 30 may include any sub-

combination of the foregoing elements while remaining consistent with an embodiment. This apparatus may be a node that implements the integrated message retransmission and/or enhanced block transfer operation on dynamically-changing resources functionalities described herein.

[0157] The processor 32 may be a general purpose processor, a special purpose processor, a conventional processor, a digital signal processor (DSP), a plurality of microprocessors, one or more microprocessors in association with a DSP core, a controller, a microcontroller, Application Specific Integrated Circuits (ASICs), Field Programmable Gate Array (FPGAs) circuits, any other type of integrated circuit (IC), a state machine, and the like. In general, the processor 32 may execute computer-executable instructions stored in the memory (e.g., memory 44 and/or memory 46) of the apparatus in order to perform the various required functions of the apparatus. For example, the processor 32 may perform signal coding, data processing, power control, input/output processing, and/or any other functionality that enables the apparatus 30 to operate in a wireless or wired environment. The computer-executable instructions stored in the memory of the apparatus, and executed by the processor, may further cause the apparatus to perform the operations illustrated in FIGs. 11, 13A, 13B, 15, 16A, and 16B described above. The processor 32 may run application-layer programs (e.g., browsers) and/or radio access-layer (RAN) programs and/or other communications programs. The processor 32 may also perform security operations such as authentication, security key agreement, and/or cryptographic operations, such as at the access-layer and/or application layer for example.

[0158] As shown in FIG. 18C, the processor 32 is coupled to its communication circuitry (e.g., transceiver 34 and transmit/receive element 36). The processor 32, through the execution of computer executable instructions, may control the communication circuitry in order to cause the apparatus 30 to communicate with other apparatuses via the network to which it is connected. In particular, the processor 32 may control the communication circuitry in order to perform the transmitting and receiving steps described and illustrated herein (e.g., in FIGs. 2-5, 7-10, and 13A-16B) and in the claims. While FIG. 18C depicts the processor 32 and the transceiver 34 as separate components, it will be appreciated that the processor 32 and the transceiver 34 may be integrated together in an electronic package or chip.

[0159] The transmit/receive element 36 may be configured to transmit signals to, or receive signals from, other apparatuses, including M2M servers, gateways, device, and the like. For example, in an embodiment, the transmit/receive element 36 may be an antenna

configured to transmit and/or receive RF signals. The transmit/receive element 36 may support various networks and air interfaces, such as WLAN, WPAN, cellular, and the like. In an embodiment, the transmit/receive element 36 may be an emitter/detector configured to transmit and/or receive IR, UV, or visible light signals, for example. In yet another embodiment, the transmit/receive element 36 may be configured to transmit and receive both RF and light signals. It will be appreciated that the transmit/receive element 36 may be configured to transmit and/or receive any combination of wireless or wired signals.

[0160] In addition, although the transmit/receive element 36 is depicted in FIG. 18C as a single element, the apparatus 30 may include any number of transmit/receive elements 36. More specifically, the apparatus 30 may employ MIMO technology. Thus, in an embodiment, the apparatus 30 may include two or more transmit/receive elements 36 (e.g., multiple antennas) for transmitting and receiving wireless signals.

[0161] The transceiver 34 may be configured to modulate the signals that are to be transmitted by the transmit/receive element 36 and to demodulate the signals that are received by the transmit/receive element 36. As noted above, the apparatus 30 may have multi-mode capabilities. Thus, the transceiver 34 may include multiple transceivers for enabling the apparatus 30 to communicate via multiple RATs, such as UTRA and IEEE 802.11, for example.

[0162] The processor 32 may access information from, and store data in, any type of suitable memory, such as the non-removable memory 44 and/or the removable memory 46. For example, the processor 32 may store session context in its memory, as described above. The non-removable memory 44 may include random-access memory (RAM), read-only memory (ROM), a hard disk, or any other type of memory storage device. The removable memory 46 may include a subscriber identity module (SIM) card, a memory stick, a secure digital (SD) memory card, and the like. In other embodiments, the processor 32 may access information from, and store data in, memory that is not physically located on the apparatus 30, such as on a server or a home computer. The processor 32 may be configured to control lighting patterns, images, or colors on the display or indicators 42 to reflect the status of communications. In one embodiment, the display/indicators 42 may present the graphical user interface illustrated in FIGs. 12 and 17 described herein.

[0163] The processor 32 may receive power from the power source 48, and may be configured to distribute and/or control the power to the other components in the apparatus 30. The power source 48 may be any suitable device for powering the apparatus 30. For example, the power source 48 may include one or more dry cell batteries (e.g., nickel-

cadmium (NiCd), nickel-zinc (NiZn), nickel metal hydride (NiMH), lithium-ion (Li-ion), etc.), solar cells, fuel cells, and the like.

[0164] The processor 32 may also be coupled to the GPS chipset 50, which is configured to provide location information (e.g., longitude and latitude) regarding the current location of the apparatus 30. It will be appreciated that the apparatus 30 may acquire location information by way of any suitable location-determination method while remaining consistent with an embodiment.

[0165] The processor 32 may further be coupled to other peripherals 52, which may include one or more software and/or hardware modules that provide additional features, functionality and/or wired or wireless connectivity. For example, the peripherals 52 may include various sensors such as an accelerometer, biometrics (e.g., fingerprint) sensors, an e-compass, a satellite transceiver, a digital camera (for photographs or video), a universal serial bus (USB) port or other interconnect interfaces, a vibration device, a television transceiver, a hands free headset, a Bluetooth® module, a frequency modulated (FM) radio unit, a digital music player, a media player, a video game player module, an Internet browser, and the like.

[0166] The apparatus 30 may be embodied in other apparatuses or devices, such as a sensor, consumer electronics, a wearable device such as a smart watch or smart clothing, a medical or eHealth device, a robot, industrial equipment, a drone, a vehicle such as a car, truck, train, or airplane. The apparatus 30 may connect to other components, modules, or systems of such apparatuses or devices via one or more interconnect interfaces, such as an interconnect interface that may comprise one of the peripherals 52.

[0167] FIG. 18D is a block diagram of an exemplary computing system 90 which may also be used to implement one or more nodes of a network, such as the clients or servers illustrated in FIGs. 2-5, 7-10, and 13A-16B and described herein, which may operate as an M2M server, gateway, device, or other network apparatus in an M2M network such as that illustrated in FIGs. 18A and 18B. Computing system 90 may comprise a computer or server and may be controlled primarily by computer readable instructions, which may be in the form of software, wherever, or by whatever means such software is stored or accessed. Such computer readable instructions may be executed within a processor, such as central processing unit (CPU) 91, to cause computing system 90 to do work, such as, for example, performing the operations illustrated and described in FIGs. 11, 13A, 13B, 15, 16A, 16B and the accompanying descriptions. In many known workstations, servers, and personal computers, central processing unit 91 is implemented by a single-chip CPU called a microprocessor. In other machines, the central processing unit 91 may comprise multiple

processors. Coprocessor 81 is an optional processor, distinct from main CPU 91, that performs additional functions or assists CPU 91. .

[0168] In operation, CPU 91 fetches, decodes, and executes instructions, and transfers information to and from other resources via the computer's main data-transfer path, system bus 80. Such a system bus connects the components in computing system 90 and defines the medium for data exchange. System bus 80 typically includes data lines for sending data, address lines for sending addresses, and control lines for sending interrupts and for operating the system bus. An example of such a system bus 80 is the PCI (Peripheral Component Interconnect) bus.

[0169] Memories coupled to system bus 80 include random access memory (RAM) 82 and read only memory (ROM) 93. Such memories include circuitry that allows information to be stored and retrieved. ROMs 93 generally contain stored data that cannot easily be modified. Data stored in RAM 82 can be read or changed by CPU 91 or other hardware devices. Access to RAM 82 and/or ROM 93 may be controlled by memory controller 92. Memory controller 92 may provide an address translation function that translates virtual addresses into physical addresses as instructions are executed. Memory controller 92 may also provide a memory protection function that isolates processes within the system and isolates system processes from user processes. Thus, a program running in a first mode can access only memory mapped by its own process virtual address space; it cannot access memory within another process's virtual address space unless memory sharing between the processes has been set up.

[0170] In addition, computing system 90 may contain peripherals controller 83 responsible for communicating instructions from CPU 91 to peripherals, such as printer 94, keyboard 84, mouse 95, and disk drive 85.

[0171] Display 86, which is controlled by display controller 96, is used to display visual output generated by computing system 90. Such visual output may include text, graphics, animated graphics, and video. Display 86 may be implemented with a CRT-based video display, an LCD-based flat-panel display, gas plasma-based flat-panel display, or a touch-panel. Display controller 96 includes electronic components required to generate a video signal that is sent to display 86. Display 86, in combination with the computer-executable instructions executed by CPU 91, may generate and operate the graphical user interface illustrated and described in FIGs. 12 and 17 and their accompanying descriptions.

[0172] Further, computing system 90 may contain communication circuitry, such as for example a network adaptor 97, that may be used to connect computing system 90 to an

external communications network, such as network 12 of FIG. 18A and FIG. 18B, to enable the computing system 90 to communicate with other nodes of the network. The communication circuitry, alone or in combination with the CPU 91, may be used to perform the transmitting and receiving steps described herein (e.g., in FIGs. 2-5, 7-10, and 13A-16B) and in the claims.

[0173] It is understood that any or all of the systems, methods and processes described herein may be embodied in the form of computer executable instructions (i.e., program code) stored on a computer-readable storage medium which instructions, when executed by a machine, such as an apparatus of an M2M network, including for example an M2M server, gateway, device or the like, perform and/or implement the systems, methods and processes described herein. Specifically, any of the steps, operations or functions described above may be implemented in the form of such computer executable instructions. Computer readable storage media include both volatile and nonvolatile, removable and non-removable media implemented in any non-transitory (i.e., tangible or physical) method or technology for storage of information, but such computer readable storage media do not includes signals. Computer readable storage media include, but are not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical disk storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other tangible or physical medium which can be used to store the desired information and which can be accessed by a computer.

[0174] This written description uses examples to disclose the invention, including the best mode, and also to enable any person skilled in the art to practice the invention, including making and using any devices or systems and performing any incorporated methods. The patentable scope of the invention is defined by the claims, and may include other examples that occur to those skilled in the art. Such other examples are intended to be within the scope of the claims if they have elements that do not differ from the literal language of the claims, or if they include equivalent elements with insubstantial differences from the literal language of the claims.

What is Claimed:

1. An apparatus comprising a processor, a memory, and communication circuitry, the apparatus being connected to a communications network via its communication circuitry, the apparatus further comprising computer-executable instructions stored in the memory of the apparatus which, when executed by the processor of the apparatus, cause the apparatus to:

send, via the communication network to an other apparatus on the network, a first CoAP message indicating a request to retrieve a first portion of a resource from the other apparatus; and

upon determining available retransmission budget:

resend, via the communication network to the other apparatus on the network, the first CoAP message indicating a request to retrieve the first portion of the resource from the other apparatus.

2. The apparatus recited in claim 1, wherein the computer-executable instructions, when executed by the processor of the apparatus, further cause the apparatus to:

include, in the first CoAP message, a CoAP Option field set to indicate a request to retrieve a retransmission budget of the other apparatus.

3. The apparatus recited in claim 1, wherein the computer-executable instructions, when executed by the processor of the apparatus, further cause the apparatus to:

include, in the first CoAP message, a CoAP Option field set to indicate a retransmission budget of the apparatus.

4. The apparatus recited in claim 1, wherein the computer-executable instructions, when executed by the processor of the apparatus, further cause the apparatus to:

accumulate retransmission budget upon successful transmission of a second CoAP message.

5. The apparatus recited in claim 1, wherein the computer-executable instructions, when executed by the processor of the apparatus, further cause the apparatus to:

consume retransmission budget upon resending the first CoAP message.

6. The apparatus recited in claim 1, wherein the computer-executable instructions, when executed by the processor of the apparatus, further cause the apparatus to:

upon determining no available retransmission budget, terminate communication with the other apparatus.

7. An apparatus comprising a processor, a memory, and communication circuitry, the apparatus being connected to a communications network via its communication circuitry, the apparatus further comprising computer-executable instructions stored in the memory of the apparatus which, when executed by the processor of the apparatus, cause the apparatus to:

send, via the communication network to an other apparatus on the network, a first CoAP message indicating a request to retrieve a first portion of a resource from the other apparatus;

wait for an acknowledgement from the other apparatus on the network, via the communication network, for an amount of time calculated using a retransmission time function;

resend, via the communication network to the other apparatus on the network, the first CoAP message indicating a request to retrieve the first portion of the resource from the other apparatus.

8. The apparatus recited in claim 7, wherein the computer-executable instructions, when executed by the processor of the apparatus, further cause the apparatus to:

include, in the first CoAP message, a CoAP Option field set to indicate a request to retrieve a retransmission budget of the other apparatus.

9. The apparatus recited in claim 7, wherein the computer-executable instructions, when executed by the processor of the apparatus, further cause the apparatus to:

include, in the first CoAP message, a CoAP Option field is set to indicate a retransmission budget of the apparatus.

10. The apparatus recited in claim 7, wherein the computer-executable instructions, when executed by the processor of the apparatus, further cause the apparatus to:

calculate the amount of time, using the retransmission time function, in a manner inversely proportional to a remaining retransmission budget.

11. The apparatus recited in claim 7, wherein the computer-executable instructions, when executed by the processor of the apparatus, further cause the apparatus to:

assign a first message identifier to the first CoAP message, the first message identifier being associated with the first portion of the resource;

send, via the communication network to the other apparatus on the network, a second CoAP message indicating a request to retrieve a second portion of a resource from the other apparatus;

assign a second message identifier to the second CoAP message, the second message identifier being associated with the second portion of the resource; and

calculate the amount of time, using the retransmission time function, in a manner inversely proportional to the number of assigned message identifiers.

12. The apparatus recited in claim 7, wherein the computer-executable instructions, when executed by the processor of the apparatus, further cause the apparatus to:

repeat the waiting and resending;

calculate the amount of time, using the retransmission time function, in a manner proportional to the number of times the waiting and resending is repeated.

13. The apparatus recited in claim 7, wherein the computer-executable instructions, when executed by the processor of the apparatus, further cause the apparatus to:

consume retransmission budget upon resending the first CoAP message.

14. A apparatus comprising a processor, a memory, and communication circuitry, the apparatus being connected to a communications network via its communication circuitry, the apparatus further comprising computer-executable instructions stored in the memory of the apparatus which, when executed by the processor of the apparatus, cause the apparatus to:

send, via the communication network to an other apparatus on the network, a first CoAP message indicating a request to retrieve a first portion of a resource from the other apparatus;

wait for an acknowledgement from the other apparatus on the network, via the communication network, for an amount of time designated by a retransmission time function; and

upon determining the amount of time designated by a retransmission time function has passed:

check available retransmission budget; and

upon determining available retransmission budget:

resend, via the communication network to the other apparatus on the network, the first CoAP message indicating a request to retrieve the first portion of the resource from the other apparatus.

15. The apparatus recited in claim 14, wherein the computer-executable instructions, when executed by the processor of the apparatus, further cause the apparatus to:

include, in the first CoAP message, a CoAP Option field set to indicate a request to retrieve a retransmission budget of the other apparatus.

16. The apparatus recited in claim 14, wherein the computer-executable instructions, when executed by the processor of the apparatus, further cause the apparatus to:

include, in the first CoAP message, a CoAP Option field is set to indicate a retransmission budget of the apparatus.

17. The apparatus recited in claim 14, wherein the computer-executable instructions, when executed by the processor of the apparatus, further cause the apparatus to:

consume retransmission budget upon resending the first CoAP message.

18. The apparatus recited in claim 14, wherein the computer-executable instructions, when executed by the processor of the apparatus, further cause the apparatus to:

calculate the amount of time, using the retransmission time function, in a manner inversely proportional to a remaining retransmission budget.

19. The apparatus recited in claim 14, wherein the computer-executable instructions, when executed by the processor of the apparatus, further cause the apparatus to:

repeat the waiting and resending;

calculate the amount of time, using the retransmission time function, in a manner proportional to the number of times the waiting and resending is repeated.

20. The apparatus recited in claim 14, wherein the computer-executable instructions, when executed by the processor of the apparatus, further cause the apparatus to:

assign a first message identifier to the first CoAP message, the first message identifier being associated with the first portion of the resource;

send, via the communication network to the other apparatus on the network, a second CoAP message indicating a request to retrieve a second portion of a resource from the other apparatus;

assign a second message identifier to the second CoAP message, the second message identifier being associated with the second portion of the resource; and

calculate the amount of time, using the retransmission time function, in a manner inversely proportional to the number of assigned message identifiers.

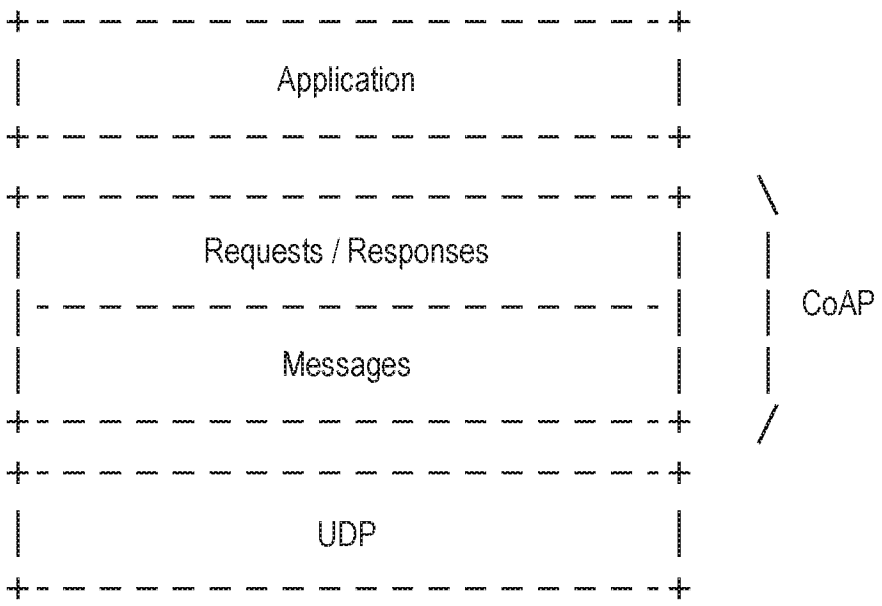


FIG. 1

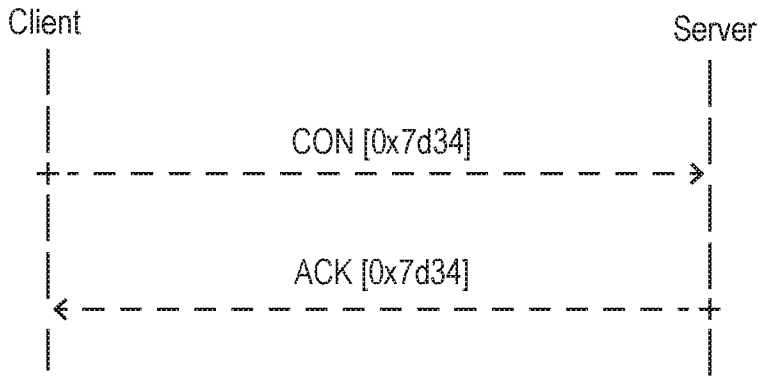


FIG. 2

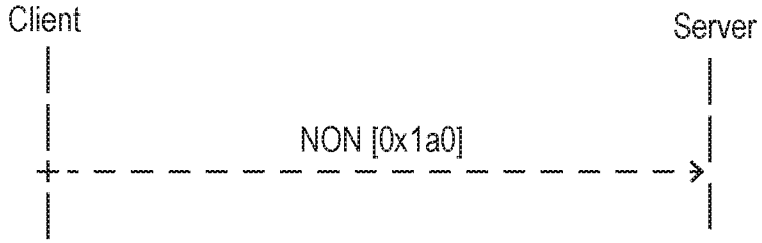


FIG. 3

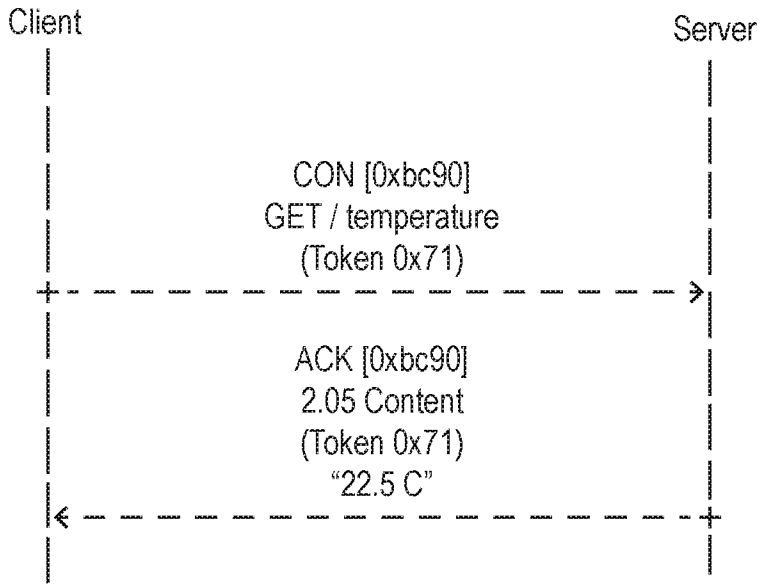


FIG. 4

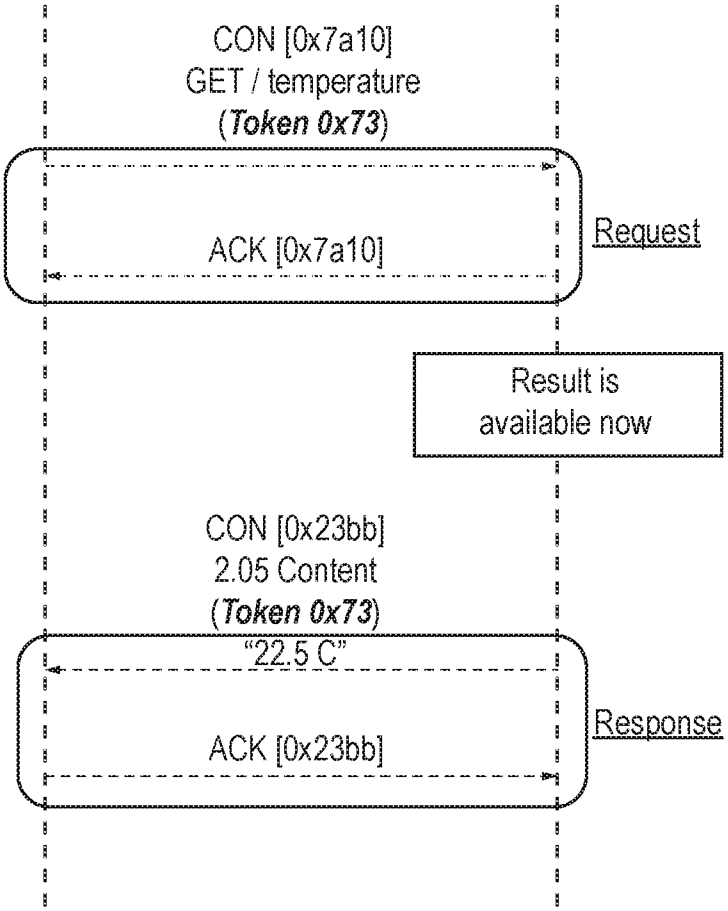

CON [0x7a10]
GET / temperature
(Token 0x73)
ACK [0x7a10]
Request
Result is
available now
CON [0x23bb]
2.05 Content
(Token 0x73)
"22.5 C"
ACK [0x23bb]
Response

FIG. 5

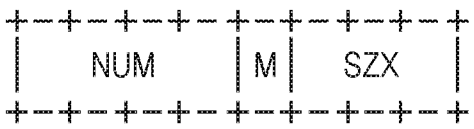

NUM
M
SZX

FIG. 6

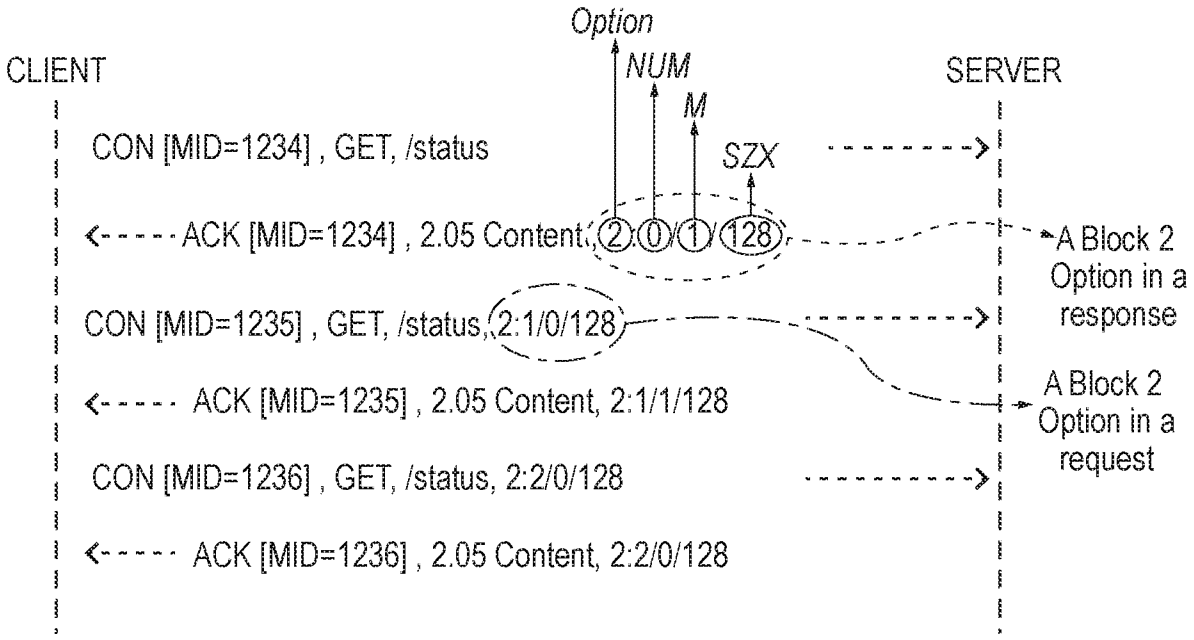

Option
NUM
M
SZX
CLIENT
SERVER
CON [MID=1234] , GET, /status
ACK [MID=1234] , 2.05 Content, 2 0 1 128
A Block 2 Option in a response
CON [MID=1235] , GET, /status, 2:1/0/128
ACK [MID=1235] , 2.05 Content, 2:1/1/128
A Block 2 Option in a request
CON [MID=1236] , GET, /status, 2:2/0/128
ACK [MID=1236] , 2.05 Content, 2:2/0/128

FIG. 7

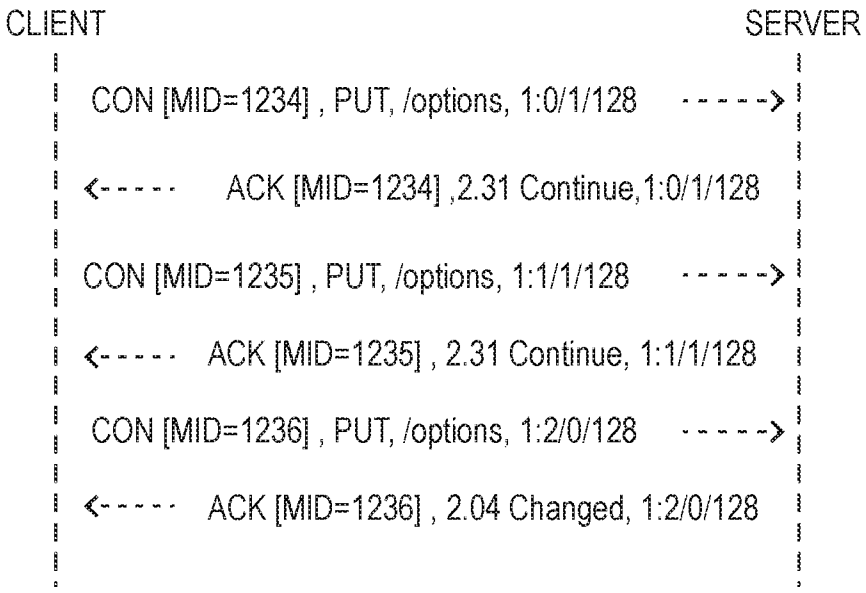

CLIENT
SERVER
CON [MID=1234] , PUT, /options, 1:0/1/128
ACK [MID=1234] ,2.31 Continue,1:0/1/128
CON [MID=1235] , PUT, /options, 1:1/1/128
ACK [MID=1235] , 2.31 Continue, 1:1/1/128
CON [MID=1236] , PUT, /options, 1:2/0/128
ACK [MID=1236] , 2.04 Changed, 1:2/0/128

FIG. 8

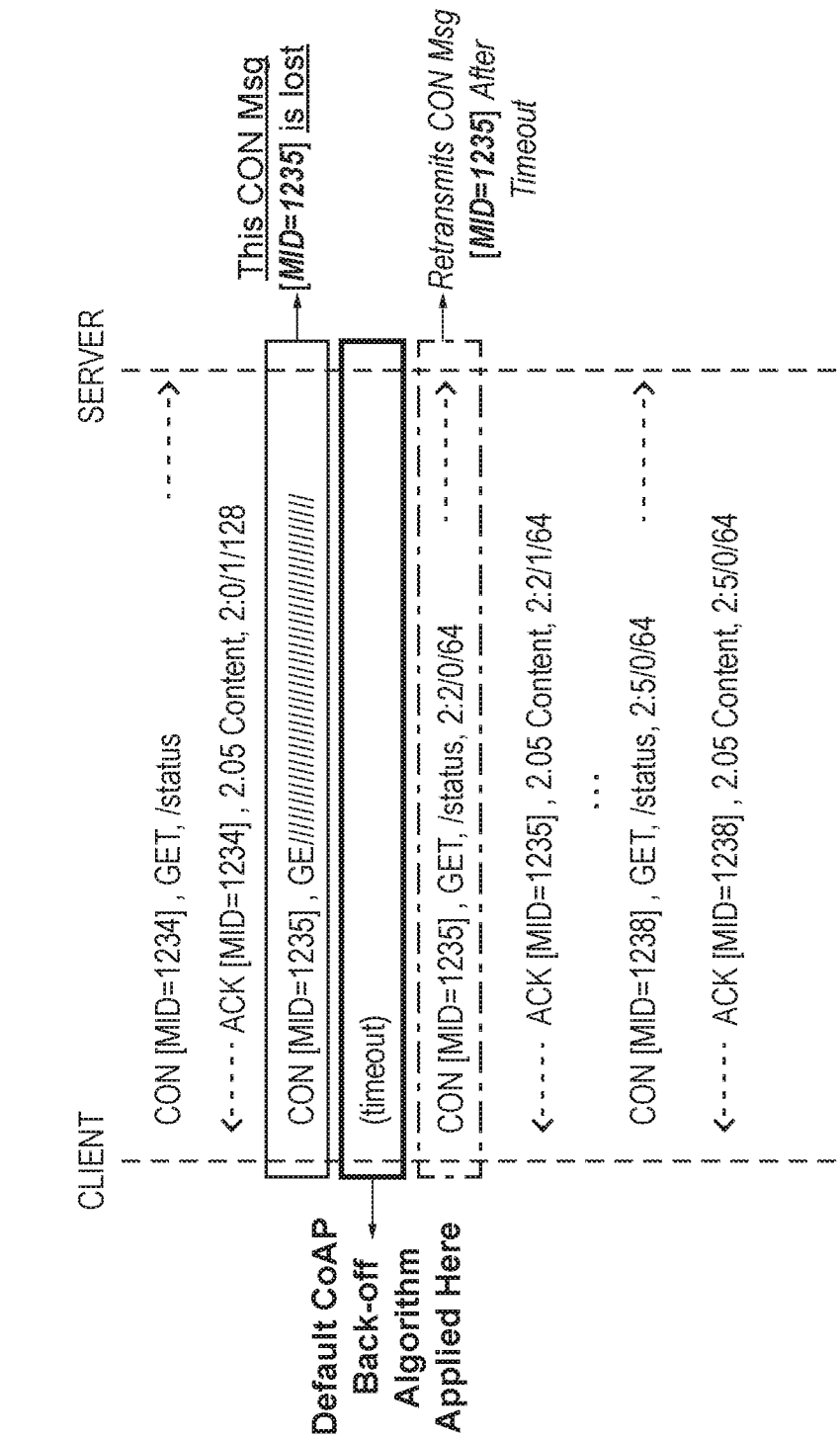

CLIENT
SERVER
CON [MID=1234], GET, /status
<- - - - ACK [MID=1234], 2.05 Content, 2:0/1/128
This CON Msg
[MID=1235] is lost
CON [MID=1235], GE////////////////////////////////
(timeout)
Default CoAP
Back-off
Algorithm
Applied Here
Retransmits CON Msg
[MID=1235] After
Timeout
CON [MID=1235], GET, /status, 2:2/0/64
<- - - - ACK [MID=1235], 2.05 Content, 2:2/1/64
...
CON [MID=1238], GET, /status, 2:5/0/64
<- - - - ACK [MID=1238], 2.05 Content, 2:5/0/64

FIG. 9

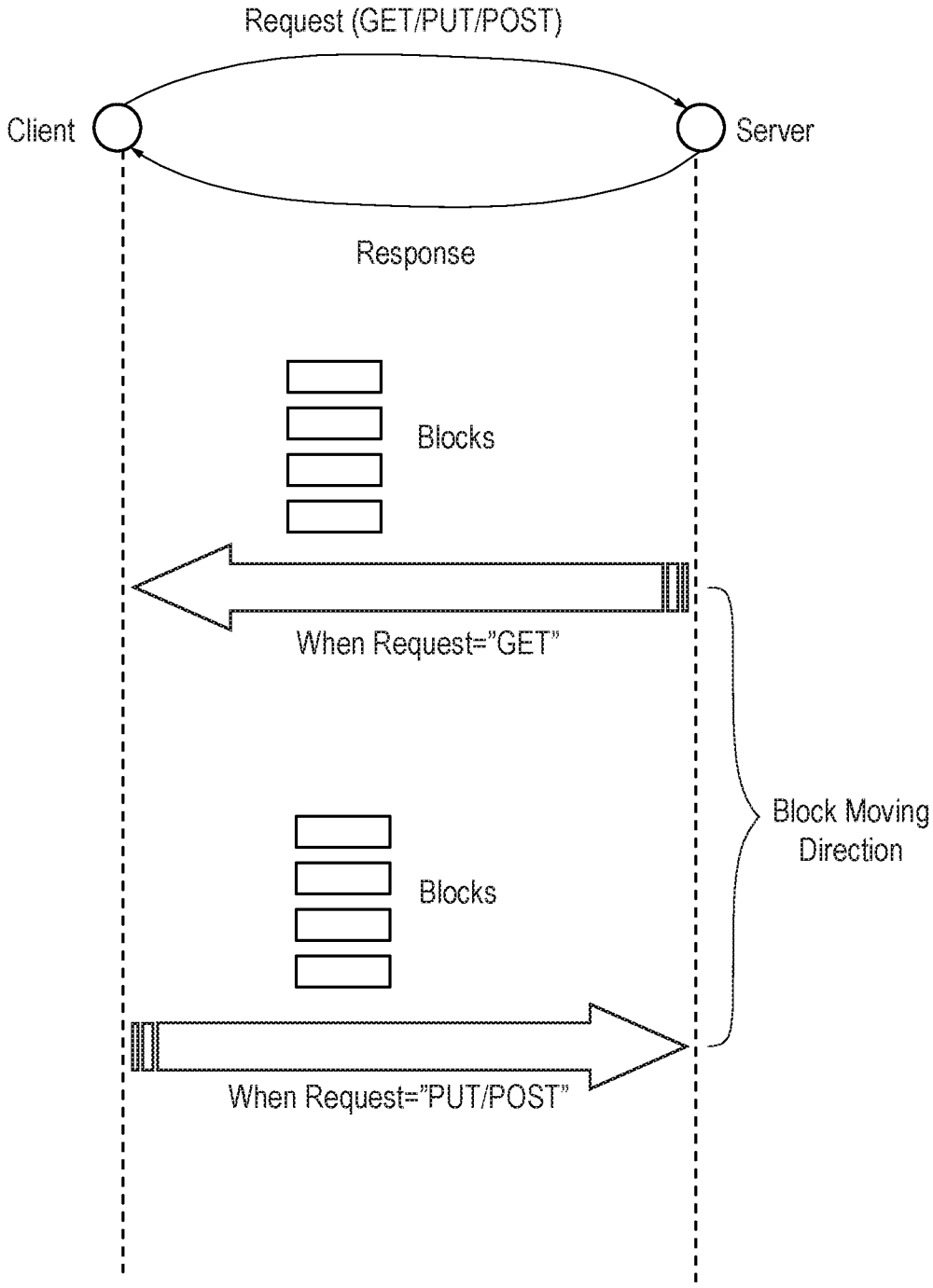

Request (GET/PUT/POST)
Client
Server
Response
Blocks
When Request="GET"
Block Moving Direction
Blocks
When Request="PUT/POST"

FIG. 10

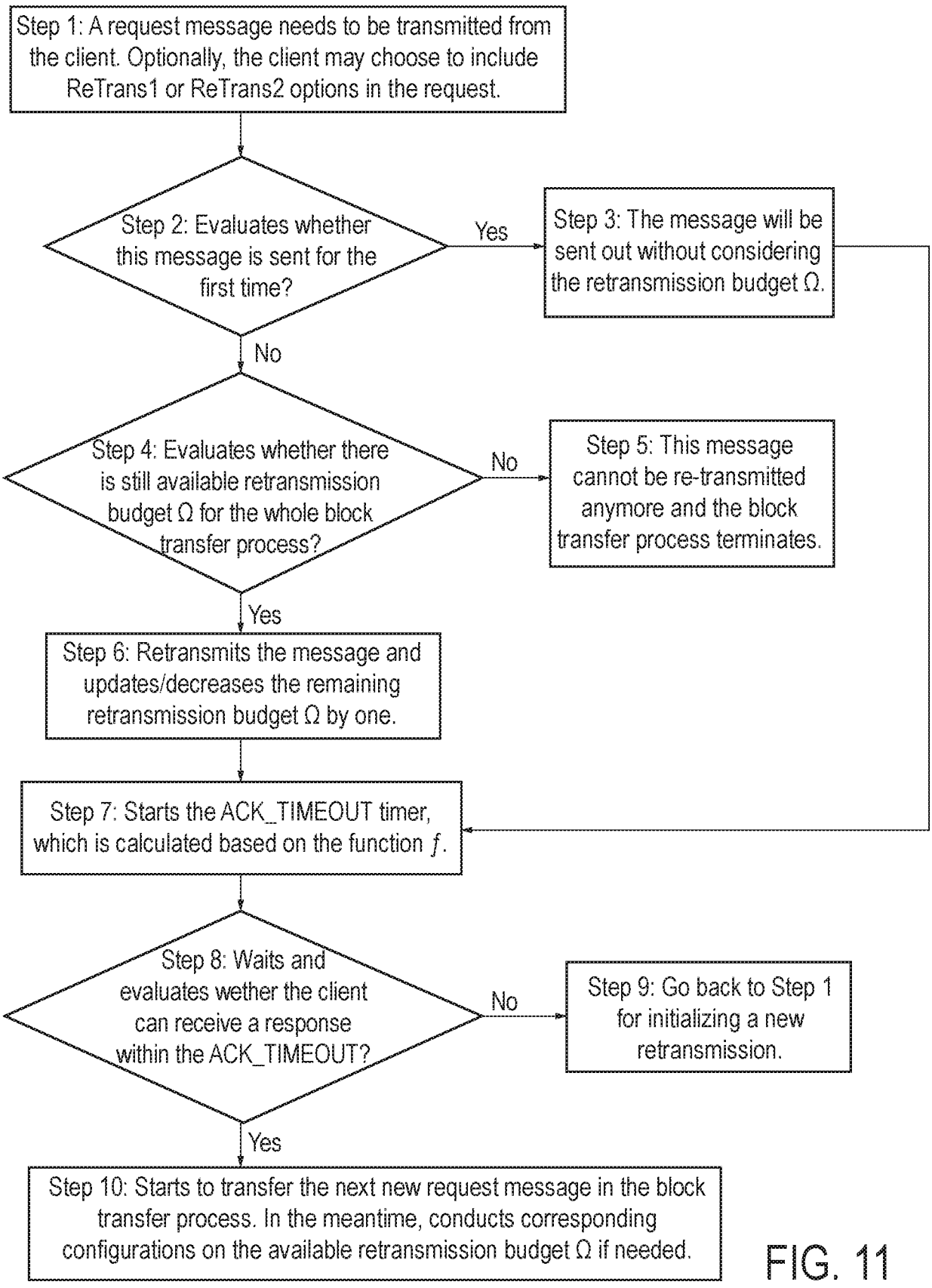

Step 1: A request message needs to be transmitted from the client. Optionally, the client may choose to include ReTrans1 or ReTrans2 options in the request.
Step 2: Evaluates whether this message is sent for the first time?
Yes
Step 3: The message will be sent out without considering the retransmission budget Ω.
No
Step 4: Evaluates whether there is still available retransmission budget Ω for the whole block transfer process?
No
Step 5: This message cannot be re-transmitted anymore and the block transfer process terminates.
Yes
Step 6: Retransmits the message and updates/decreases the remaining retransmission budget Ω by one.
Step 7: Starts the ACK_TIMEOUT timer, which is calculated based on the function f.
Step 8: Waits and evaluates wether the client can receive a response within the ACK_TIMEOUT?
No
Step 9: Go back to Step 1 for initializing a new retransmission.
Yes
Step 10: Starts to transfer the next new request message in the block transfer process. In the meantime, conducts corresponding configurations on the available retransmission budget Ω if needed.

FIG. 11

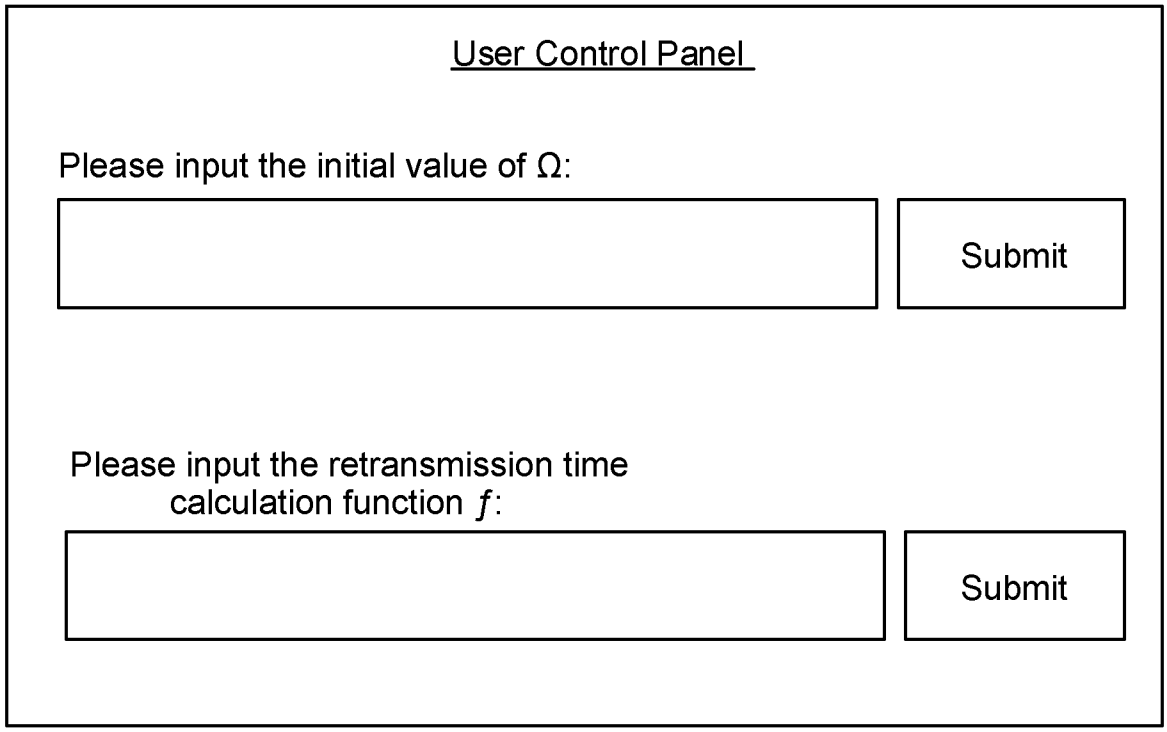

User Control Panel
Please input the initial value of Ω:
Submit
Please input the retransmission time calculation function f:
Submit

FIG. 12

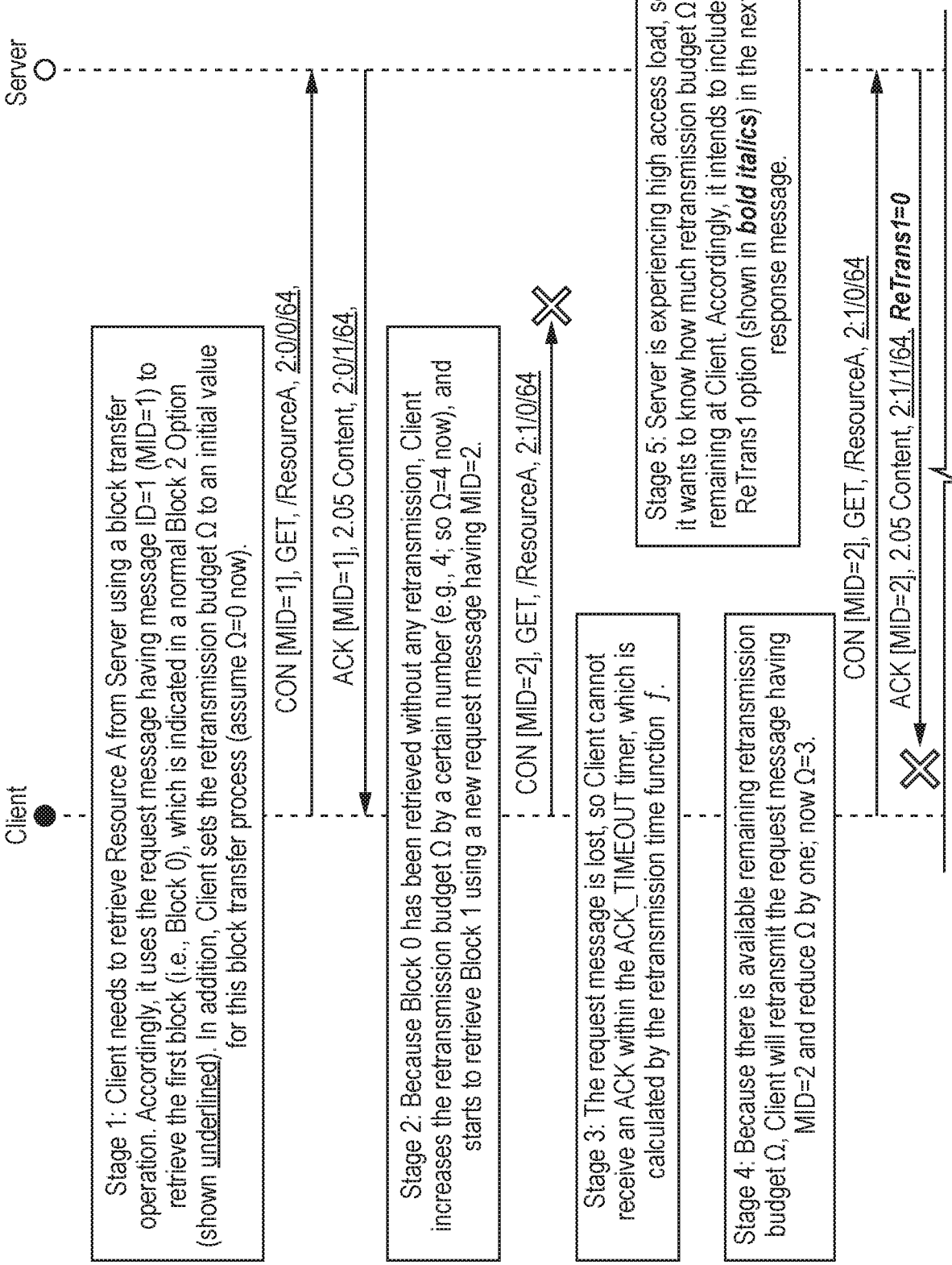

Client
Server
Stage 1: Client needs to retrieve Resource A from Server using a block transfer operation. Accordingly, it uses the request message having message ID=1 (MID=1) to retrieve the first block (i.e., Block 0), which is indicated in a normal Block 2 Option (shown underlined). In addition, Client sets the retransmission budget Ω to an initial value for this block transfer process (assume Ω=0 now).
CON [MID=1], GET, /ResourceA, 2:0/0/64
ACK [MID=1], 2.05 Content, 2:0/1/64
Stage 2: Because Block 0 has been retrieved without any retransmission, Client increases the retransmission budget Ω by a certain number (e.g., 4; so Ω=4 now), and starts to retrieve Block 1 using a new request message having MID=2.
CON [MID=2], GET, /ResourceA, 2:1/0/64
Stage 3: The request message is lost, so Client cannot receive an ACK within the ACK_TIMEOUT timer, which is calculated by the retransmission time function f.
Stage 4: Because there is available remaining retransmission budget Ω, Client will retransmit the request message having MID=2 and reduce Ω by one; now Ω=3.
CON [MID=2], GET, /ResourceA, 2:1/0/64
ACK [MID=2], 2.05 Content, 2:1/1/64 ReTrans1=0
Stage 5: Server is experiencing high access load, so it wants to know how much retransmission budget Ω is remaining at Client. Accordingly, it intends to include a ReTrans1 option (shown in bold italics) in the next response message.

FIG. 13A

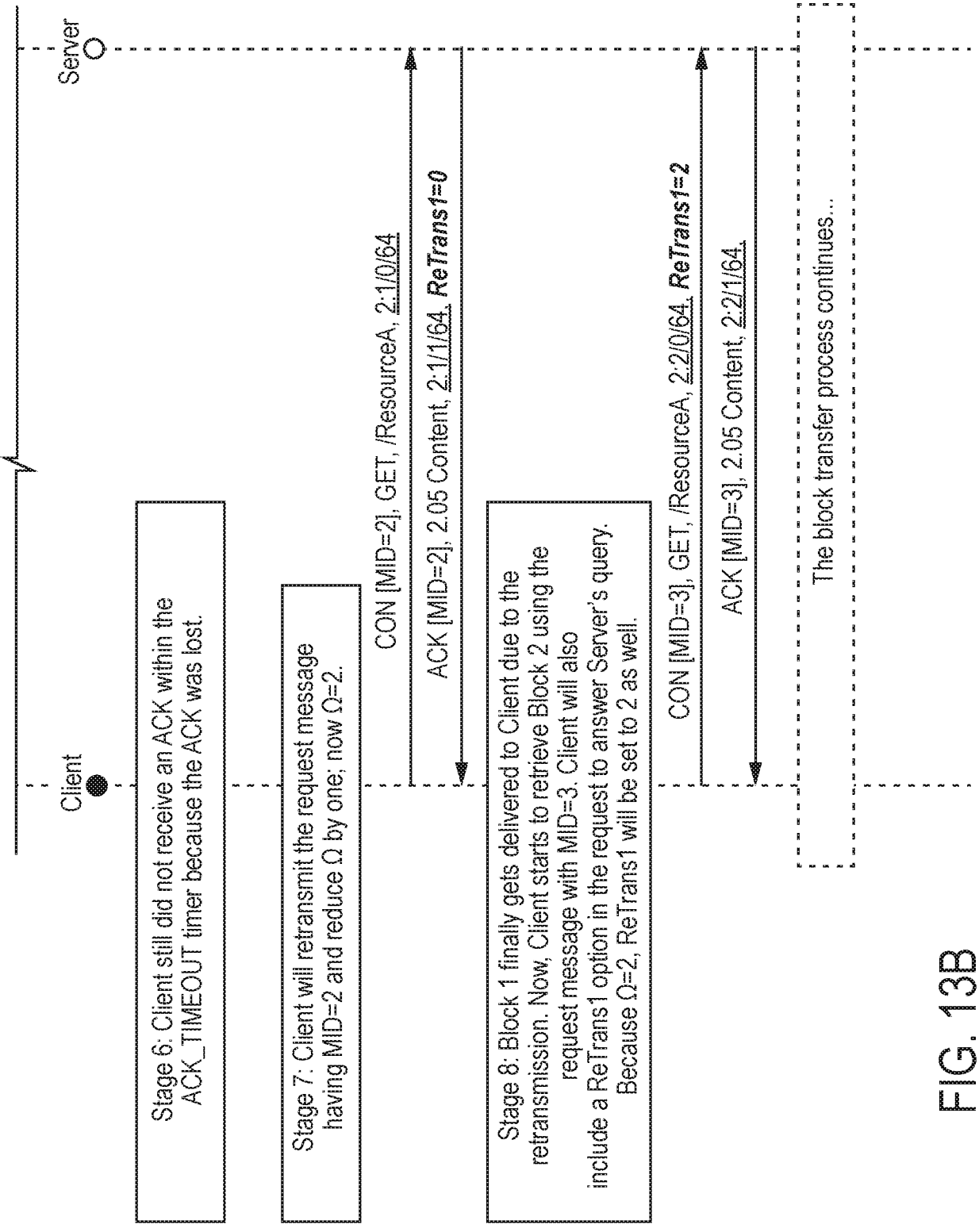

Client
Server
Stage 6: Client still did not receive an ACK within the ACK_TIMEOUT timer because the ACK was lost.
Stage 7: Client will retransmit the request message having MID=2 and reduce Ω by one; now Ω=2.
CON [MID=2], GET, /ResourceA, 2:1/0/64
ACK [MID=2], 2.05 Content, 2:1/1/64, ReTrans1=0
Stage 8: Block 1 finally gets delivered to Client due to the retransmission. Now, Client starts to retrieve Block 2 using the request message with MID=3. Client will also include a ReTrans1 option in the request to answer Server's query. Because Ω=2, ReTrans1 will be set to 2 as well.
CON [MID=3], GET, /ResourceA, 2:2/0/64, ReTrans1=2
ACK [MID=3], 2.05 Content, 2:2/1/64,
The block transfer process continues...

FIG. 13B

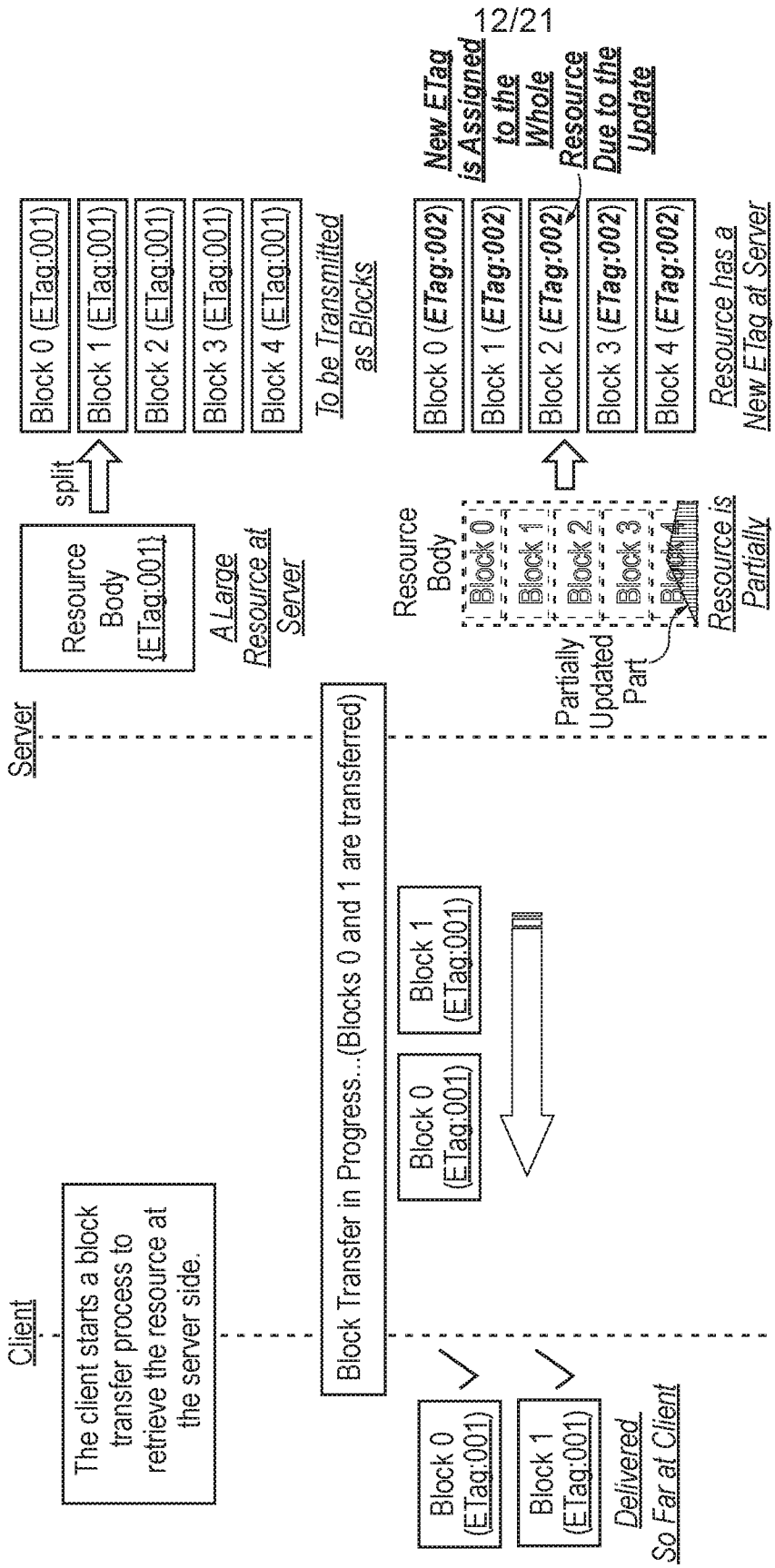

Client
Server
The client starts a block transfer process to retrieve the resource at the server side.
Block Transfer in Progress...(Blocks 0 and 1 are transferred)
Block 0 (ETag:001)
Block 1 (ETag:001)
Delivered So Far at Client
Resource Body (ETag:001)
A Large Resource at Server
split
Block 0 (ETag:001)
Block 1 (ETag:001)
Block 2 (ETag:001)
Block 3 (ETag:001)
Block 4 (ETag:001)
To be Transmitted as Blocks
Resource Body
Block 0
Block 1
Block 2
Block 3
Block 4
Partially Updated Part
Resource is Partially Updated at Server
Block 0 (ETag:002)
Block 1 (ETag:002)
Block 2 (ETag:002)
Block 3 (ETag:002)
Block 4 (ETag:002)
Resource has a New ETag at Server
New ETag is Assigned to the Whole Resource Due to the Update

FIG. 14A

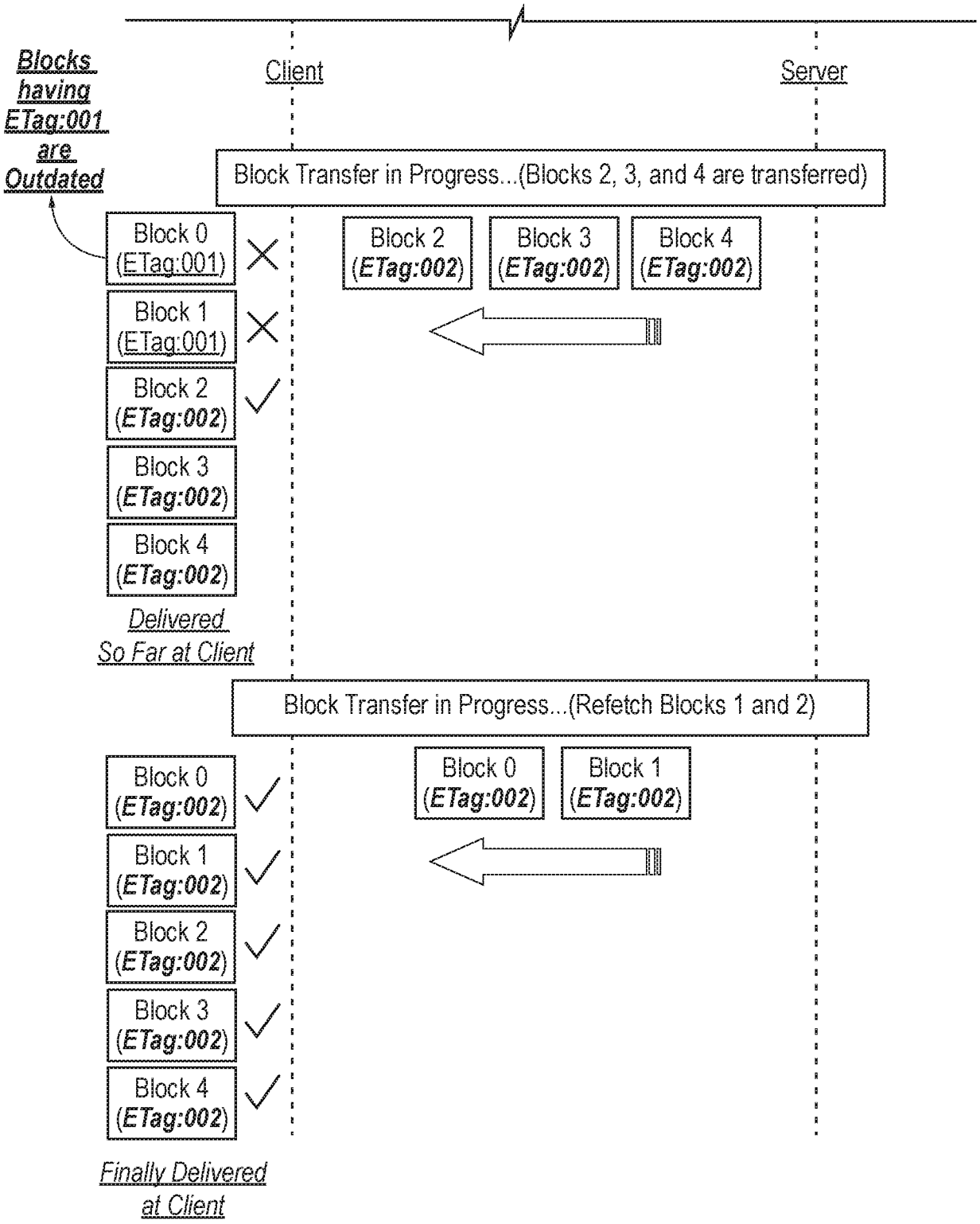

Blocks having ETag:001 are Outdated
Client
Server
Block Transfer in Progress...(Blocks 2, 3, and 4 are transferred)
Block 0 (ETag:001)
Block 1 (ETag:001)
Block 2 (ETag:002)
Block 3 (ETag:002)
Block 4 (ETag:002)
Block 2 (ETag:002)
Block 3 (ETag:002)
Block 4 (ETag:002)
Delivered So Far at Client
Block Transfer in Progress...(Refetch Blocks 1 and 2)
Block 0 (ETag:002)
Block 1 (ETag:002)
Block 0 (ETag:002)
Block 1 (ETag:002)
Block 2 (ETag:002)
Block 3 (ETag:002)
Block 4 (ETag:002)
Finally Delivered at Client

FIG. 14B

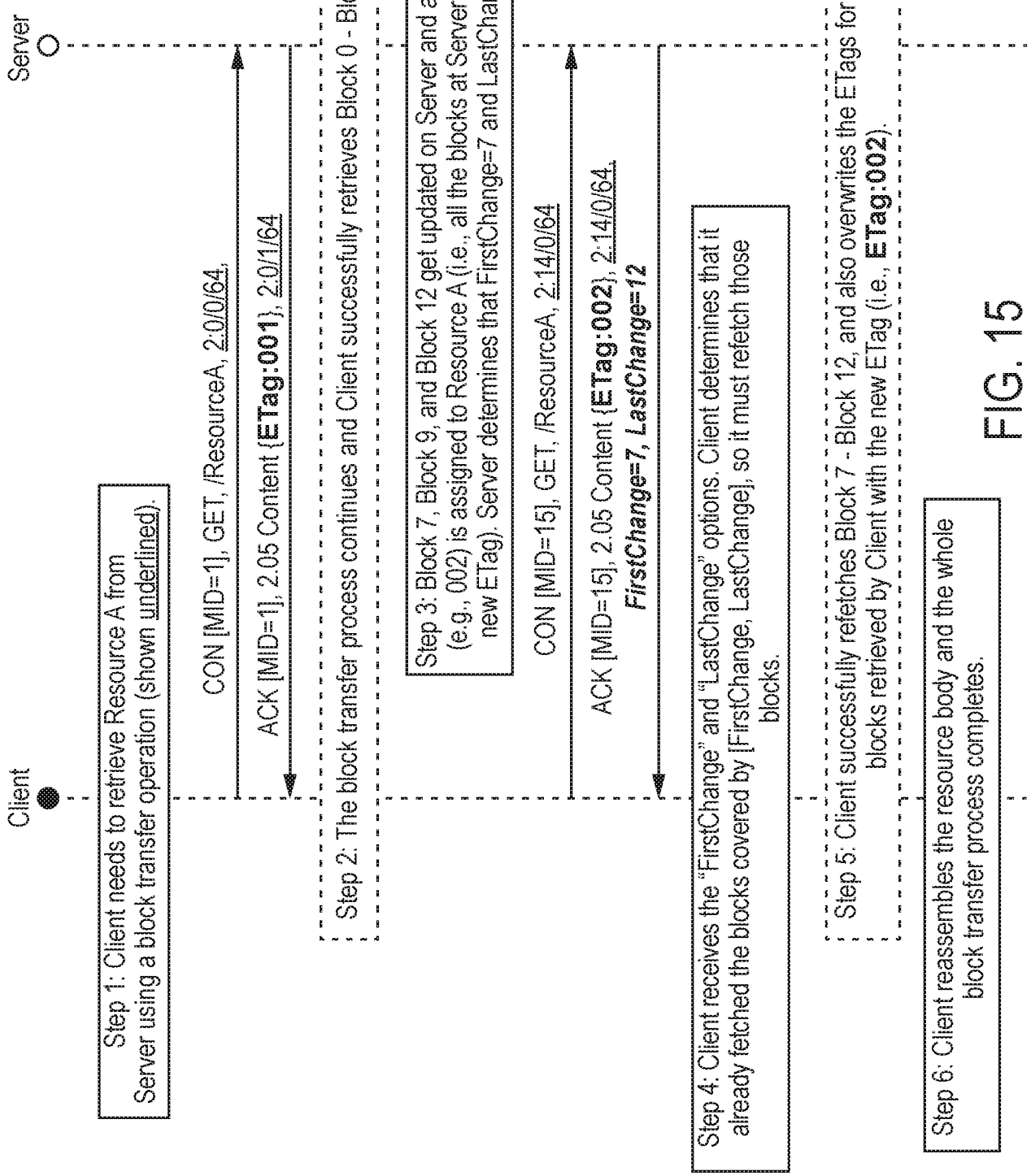

Client
Server
Step 1: Client needs to retrieve Resource A from Server using a block transfer operation (shown underlined).
CON [MID=1], GET, /ResourceA, 2:0/0/64,
ACK [MID=1], 2.05 Content {ETag:001}, 2:0/1/64
Step 2: The block transfer process continues and Client successfully retrieves Block 0 - Block 13.
Step 3: Block 7, Block 9, and Block 12 get updated on Server and a new ETag (e.g., 002) is assigned to Resource A (i.e., all the blocks at Server have the new ETag). Server determines that FirstChange=7 and LastChange=12.
CON [MID=15], GET, /ResourceA, 2:14/0/64
ACK [MID=15], 2.05 Content {ETag:002}, 2:14/0/64,
FirstChange=7, LastChange=12
Step 4: Client receives the "FirstChange" and "LastChange" options. Client determines that it already fetched the blocks covered by [FirstChange, LastChange], so it must refetch those blocks.
Step 5: Client successfully refetches Block 7 - Block 12, and also overwrites the ETags for all blocks retrieved by Client with the new ETag (i.e., ETag:002).
Step 6: Client reassembles the resource body and the whole block transfer process completes.

FIG. 15

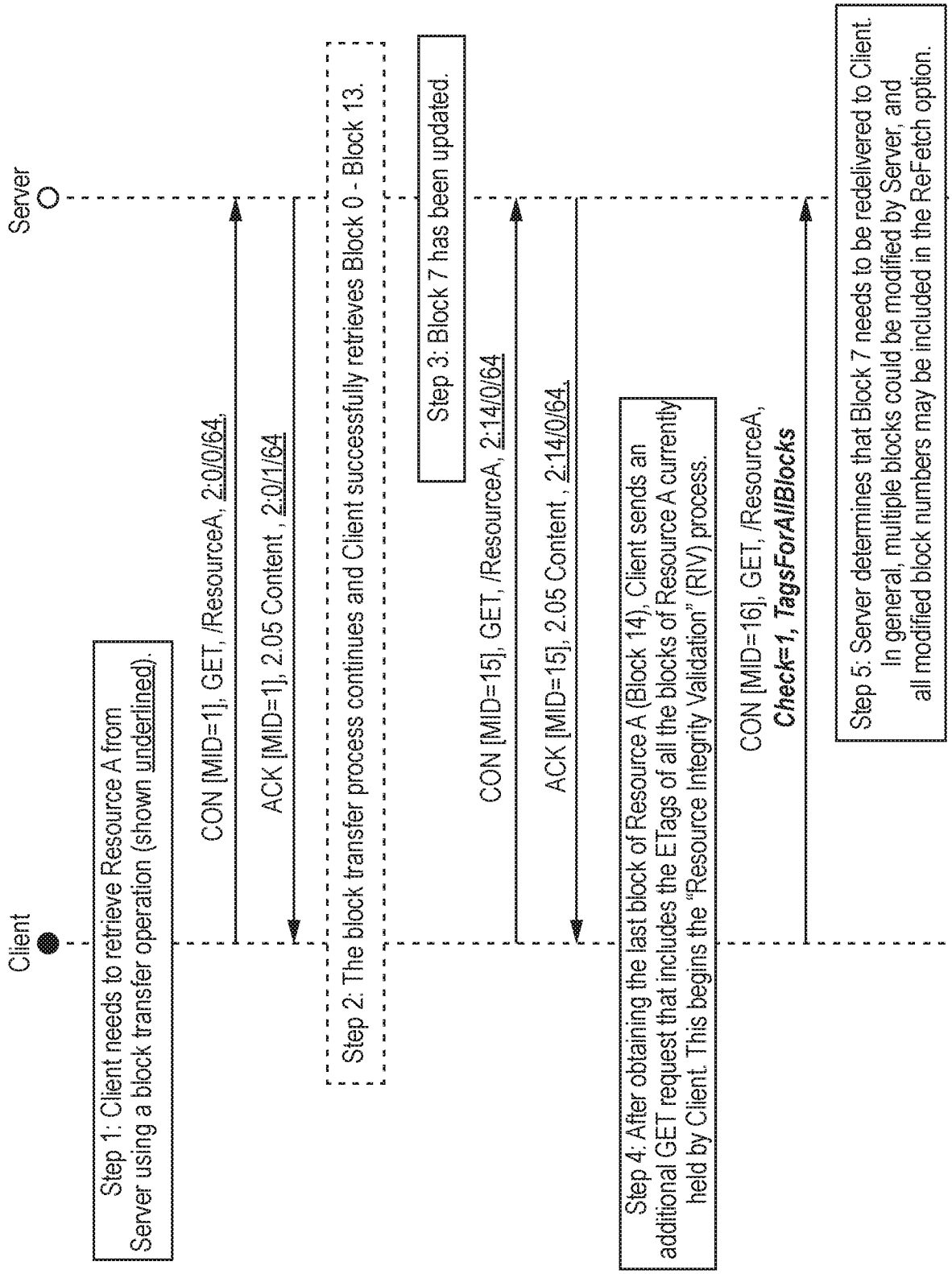

Client
Server
Step 1: Client needs to retrieve Resource A from Server using a block transfer operation (shown underlined).
CON [MID=1], GET, /ResourceA, 2:0/0/64,
ACK [MID=1], 2.05 Content , 2:0/1/64
Step 2: The block transfer process continues and Client successfully retrieves Block 0 - Block 13.
Step 3: Block 7 has been updated.
CON [MID=15], GET, /ResourceA, 2:14/0/64
ACK [MID=15], 2.05 Content , 2:14/0/64
Step 4: After obtaining the last block of Resource A (Block 14), Client sends an additional GET request that includes the ETags of all the blocks of Resource A currently held by Client. This begins the "Resource Integrity Validation" (RIV) process.
CON [MID=16], GET, /ResourceA, Check=1, TagsForAllBlocks
Step 5: Server determines that Block 7 needs to be redelivered to Client. In general, multiple blocks could be modified by Server, and all modified block numbers may be included in the ReFetch option.

FIG. 16A

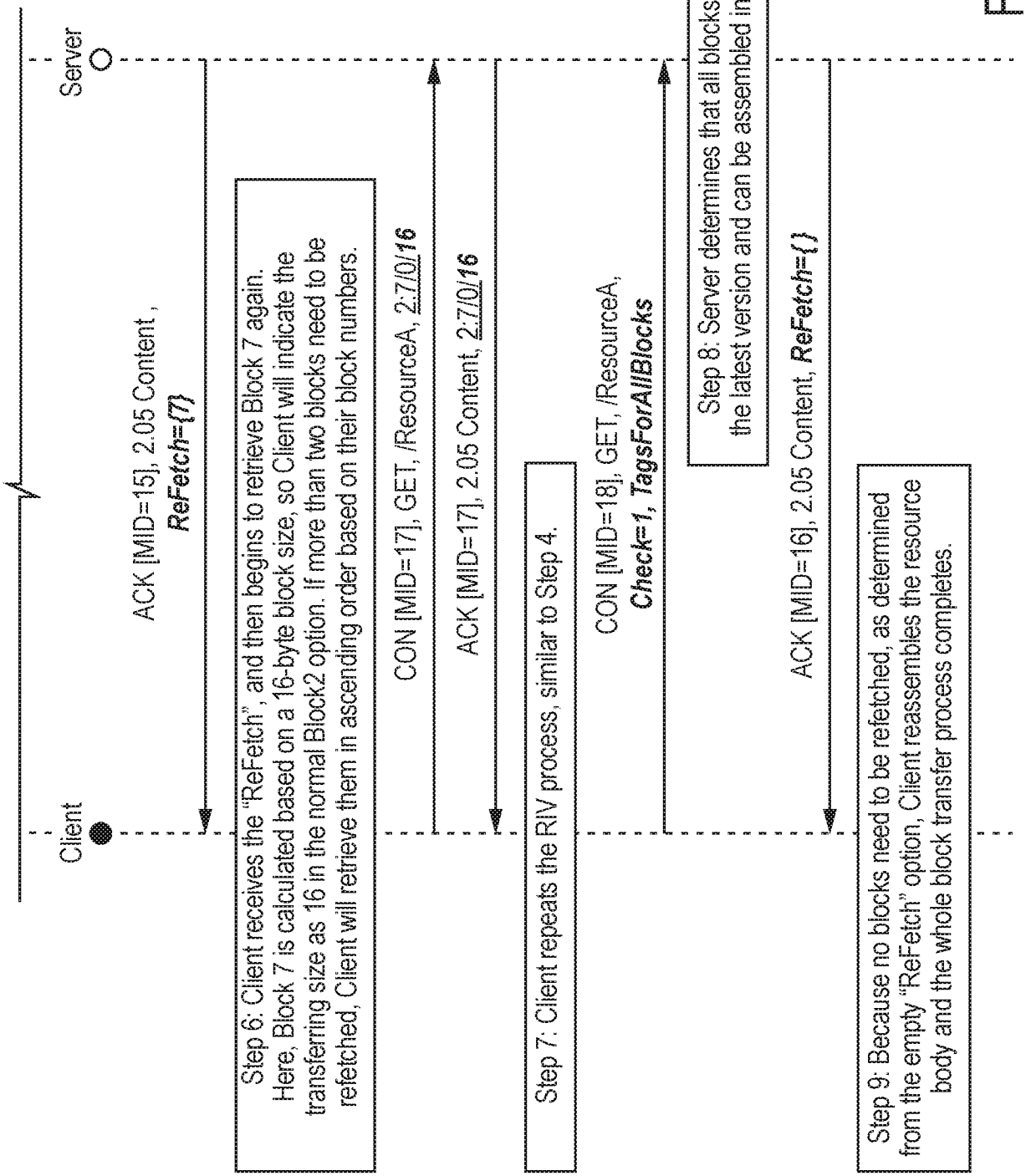

Client
Server
ACK [MID=15], 2.05 Content ,
ReFetch=(7)
Step 6: Client receives the "ReFetch", and then begins to retrieve Block 7 again. Here, Block 7 is calculated based on a 16-byte block size, so Client will indicate the transferring size as 16 in the normal Block2 option. If more than two blocks need to be refetched, Client will retrieve them in ascending order based on their block numbers.
CON [MID=17], GET, /ResourceA, 2:7/0/16
ACK [MID=17], 2.05 Content, 2:7/0/16
Step 7: Client repeats the RIV process, similar to Step 4.
CON [MID=18], GET, /ResourceA,
Check=1, TagsForAllBlocks
Step 8: Server determines that all blocks at Client are the latest version and can be assembled into Resource A.
ACK [MID=16], 2.05 Content, ReFetch=(1)
Step 9: Because no blocks need to be refetched, as determined from the empty "ReFetch" option, Client reassembles the resource body and the whole block transfer process completes.

FIG. 16B

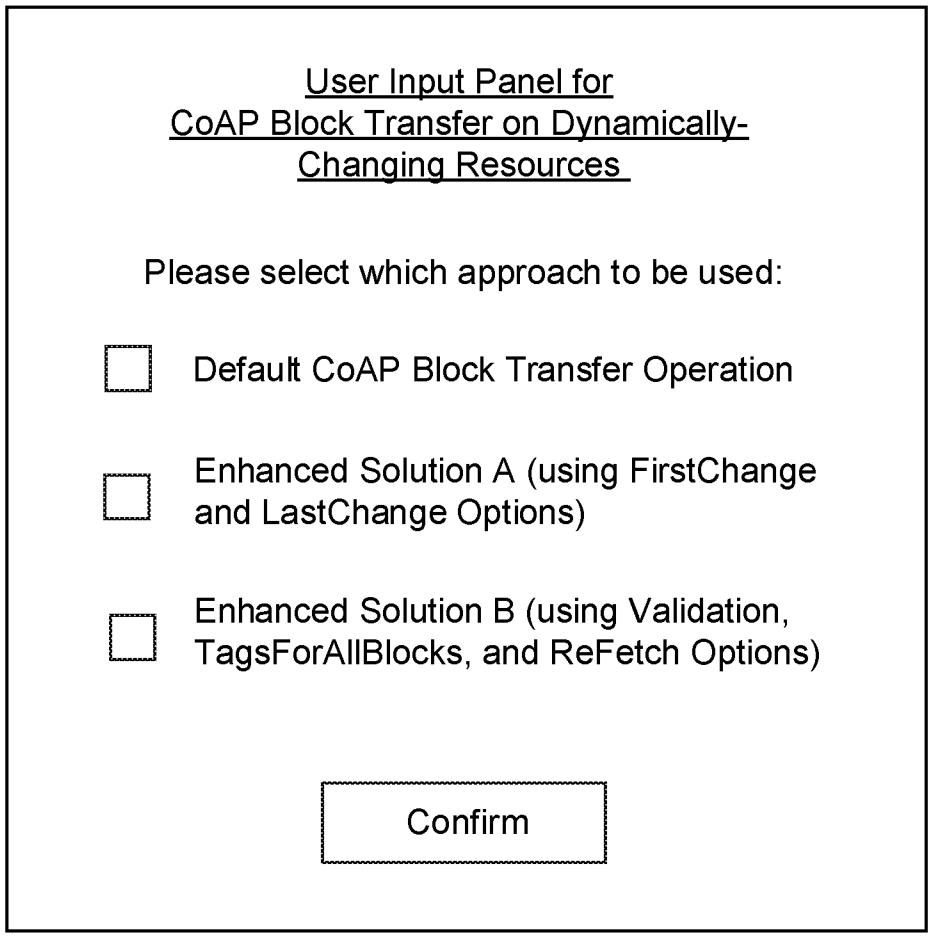

User Input Panel for
CoAP Block Transfer on Dynamically-Changing Resources
Please select which approach to be used:
Default CoAP Block Transfer Operation
Enhanced Solution A (using FirstChange and LastChange Options)
Enhanced Solution B (using Validation, TagsForAllBlocks, and ReFetch Options)
Confirm

FIG. 17

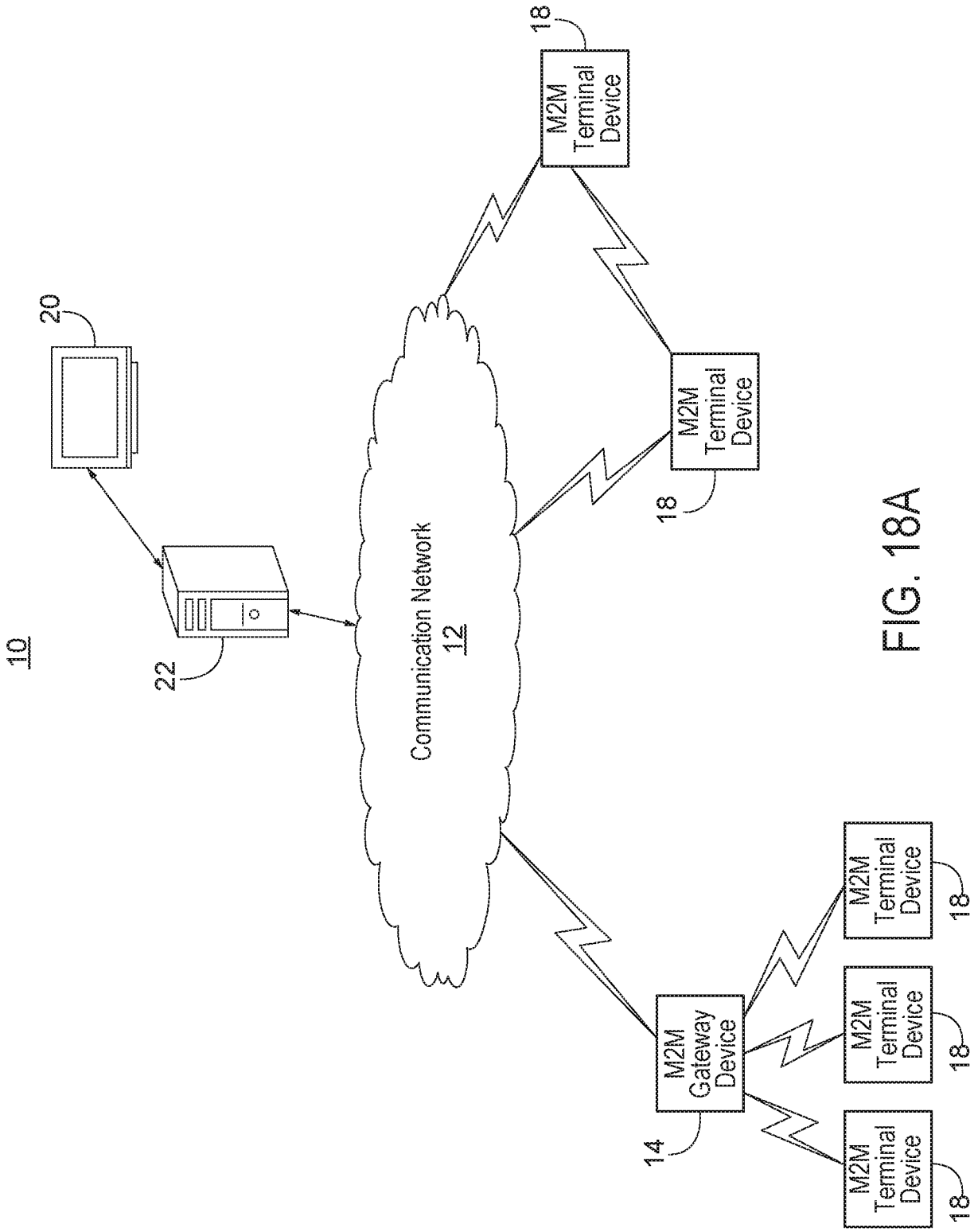

10
20
22
Communication Network
12
M2M Terminal Device
18
M2M Terminal Device
18
M2M Gateway Device
14
M2M Terminal Device
18
M2M Terminal Device
18
M2M Terminal Device
18

FIG. 18A

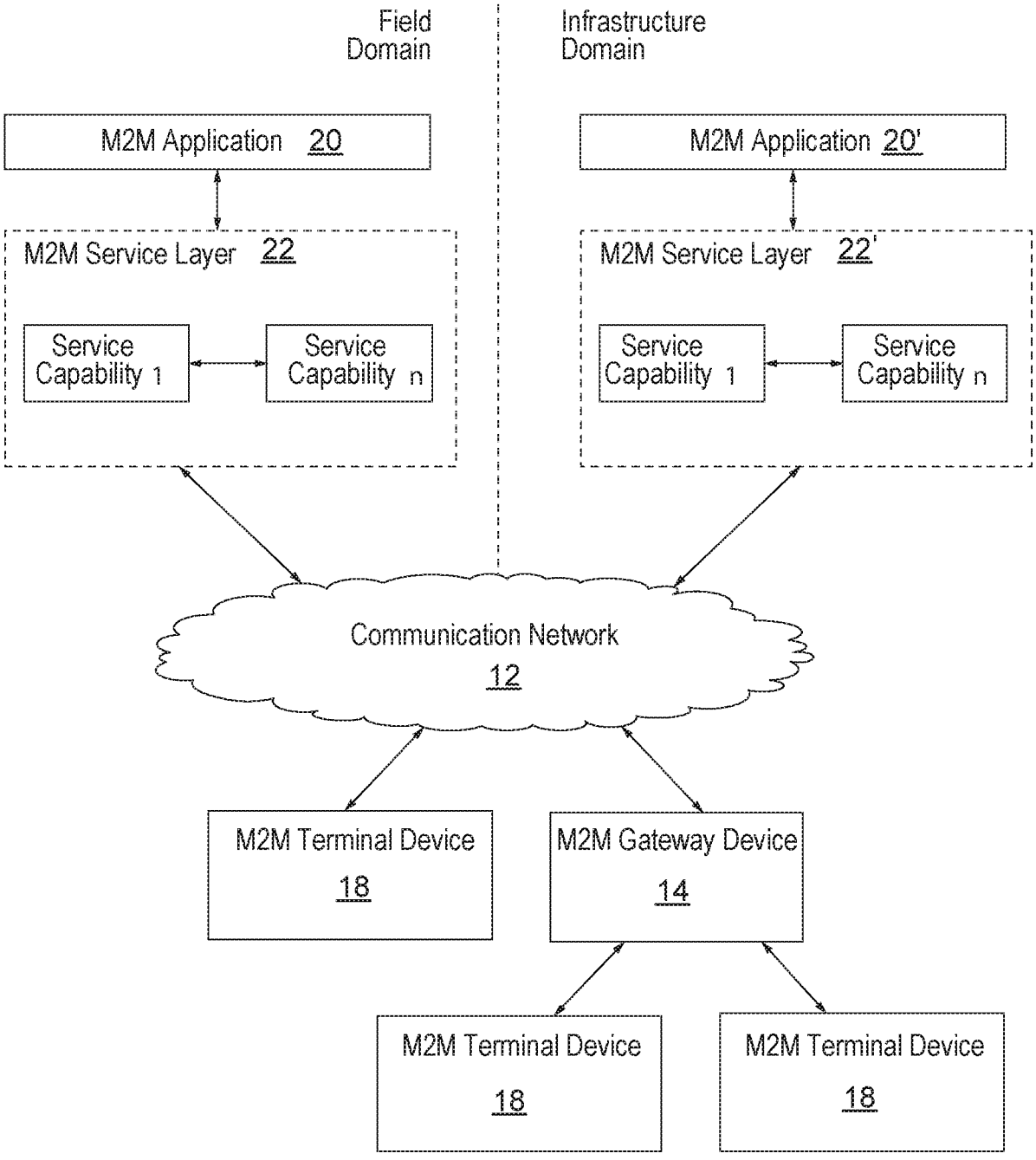

Field Domain
Infrastructure Domain
M2M Application 20
M2M Application 20'
M2M Service Layer 22
M2M Service Layer 22'
Service Capability 1
Service Capability n
Service Capability 1
Service Capability n
Communication Network 12
M2M Terminal Device 18
M2M Gateway Device 14
M2M Terminal Device 18
M2M Terminal Device 18

FIG. 18B

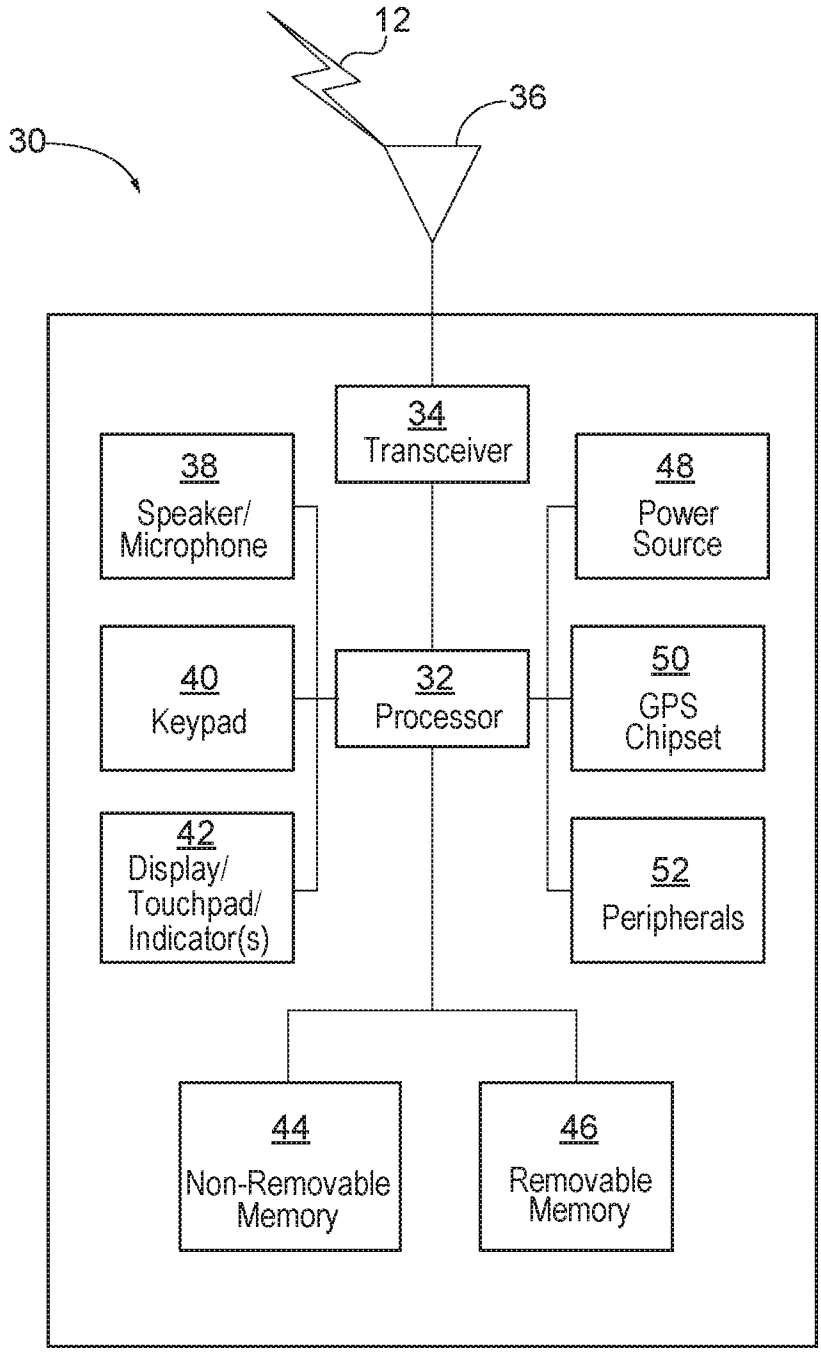

12
36
30
34
Transceiver
38
Speaker/
Microphone
48
Power
Source
40
Keypad
32
Processor
50
GPS
Chipset
42
Display/
Touchpad/
Indicator(s)
52
Peripherals
44
Non-Removable
Memory
46
Removable
Memory

FIG. 18C

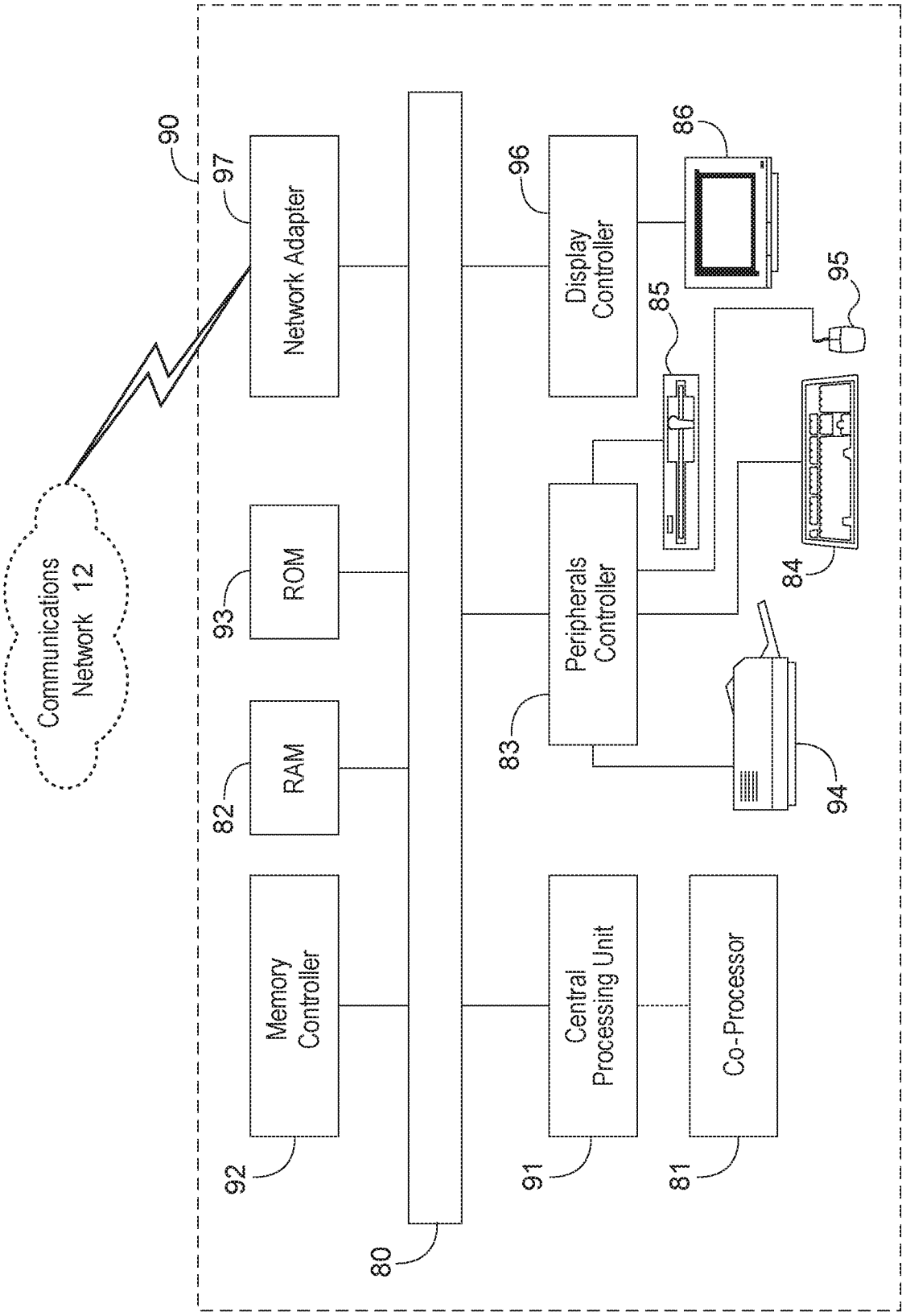

90
Communications Network 12
97
Network Adapter
93
ROM
82
RAM
92
Memory Controller
80
96
Display Controller
86
85
95
Peripherals Controller
84
83
94
91
Central Processing Unit
81
Co-Processor

FIG. 18D

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2016/050123

A. CLASSIFICATION OF SUBJECT MATTER
INV. H04L1/18 H04L29/08 H04W4/00 H04L29/06
ADD.

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

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
H04L H04W

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

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	SHELBY K HARTKE C BORMANN UNIVERSITAET BREMEN TZI Z: "The Constrained Application Protocol (CoAP); rfc7252.txt", THE CONSTRAINED APPLICATION PROTOCOL (COAP); RFC7252.TXT, INTERNET ENGINEERING TASK FORCE, IETF; STANDARD, INTERNET SOCIETY (ISOC) 4, RUE DES FALAISES CH-1205 GENEVA, SWITZERLAND, 27 June 2014 (2014-06-27), pages 1-112, XP015100722, [retrieved on 2014-06-27]	1,5-7, 11,13, 14,17,20
Y	page 6, lines 19-25 page 21, lines 14-16 page 21, line 41 - page 22, line 6 page 22, lines 38-41	2-4, 8-10,12, 15,16, 18,19

-/--

[X] Further documents are listed in the continuation of Box C.

[X] See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
18 November 2016	25/11/2016

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

2

Form PCT/ISA/210 (second sheet) (April 2005)

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2016/050123

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	STEWART R: "RFC 4690 - Stream Control Transmission Protocol", INTERNET CITATION, September 2007 (2007-09), pages 1-152, XP002648548, Retrieved from the Internet: URL:http://tools.ietf.org/html/rfc4960 [retrieved on 2011-07-07] page 25, line 23 page 31, line 9 page 96, lines 23-35	2-4,8,9, 15,16
Y	EP 1 558 000 A2 (NTT DOCOMO INC [JP]) 27 July 2005 (2005-07-27) paragraph [0012]	12,19
Y	EP 2 140 627 A1 (BRITISH TELECOMM [GB]) 6 January 2010 (2010-01-06) paragraph [0005]	10,18

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/US2016/050123

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 1558000 A2	27-07-2005	CN 1645866 A	27-07-2005
		EP 1558000 A2	27-07-2005
		JP 4594110 B2	08-12-2010
		JP 2005236961 A	02-09-2005
		US 2005190720 A1	01-09-2005
EP 2140627 A1	06-01-2010	AT 547872 T	15-03-2012
		EP 2140627 A1	06-01-2010
		US 2010128653 A1	27-05-2010
		WO 2008119948 A1	09-10-2008