FIG. 50 – ACCELERATION SYSTEM LAYOUT

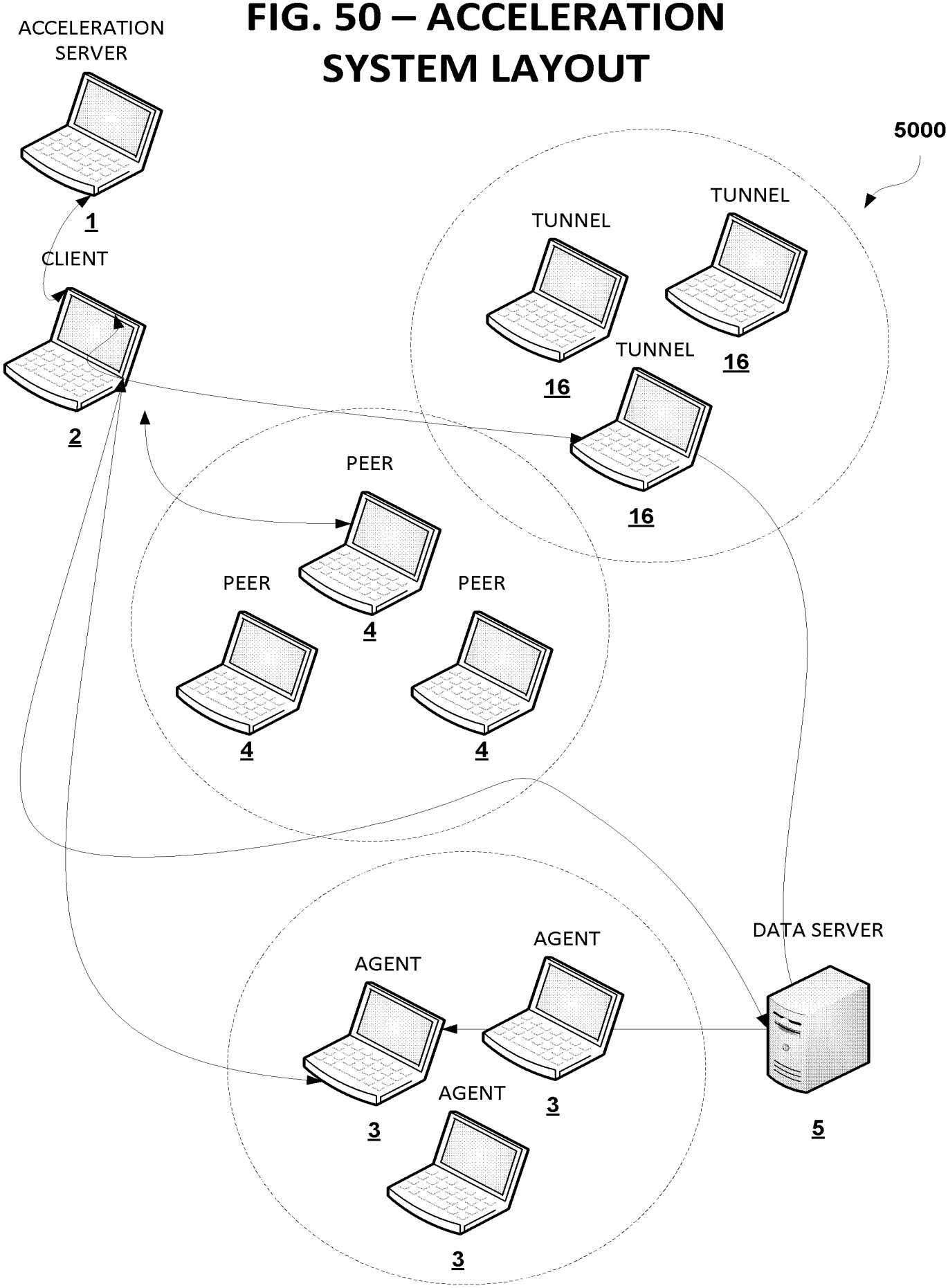


FIG. 55 – ACCELERATION MODULE

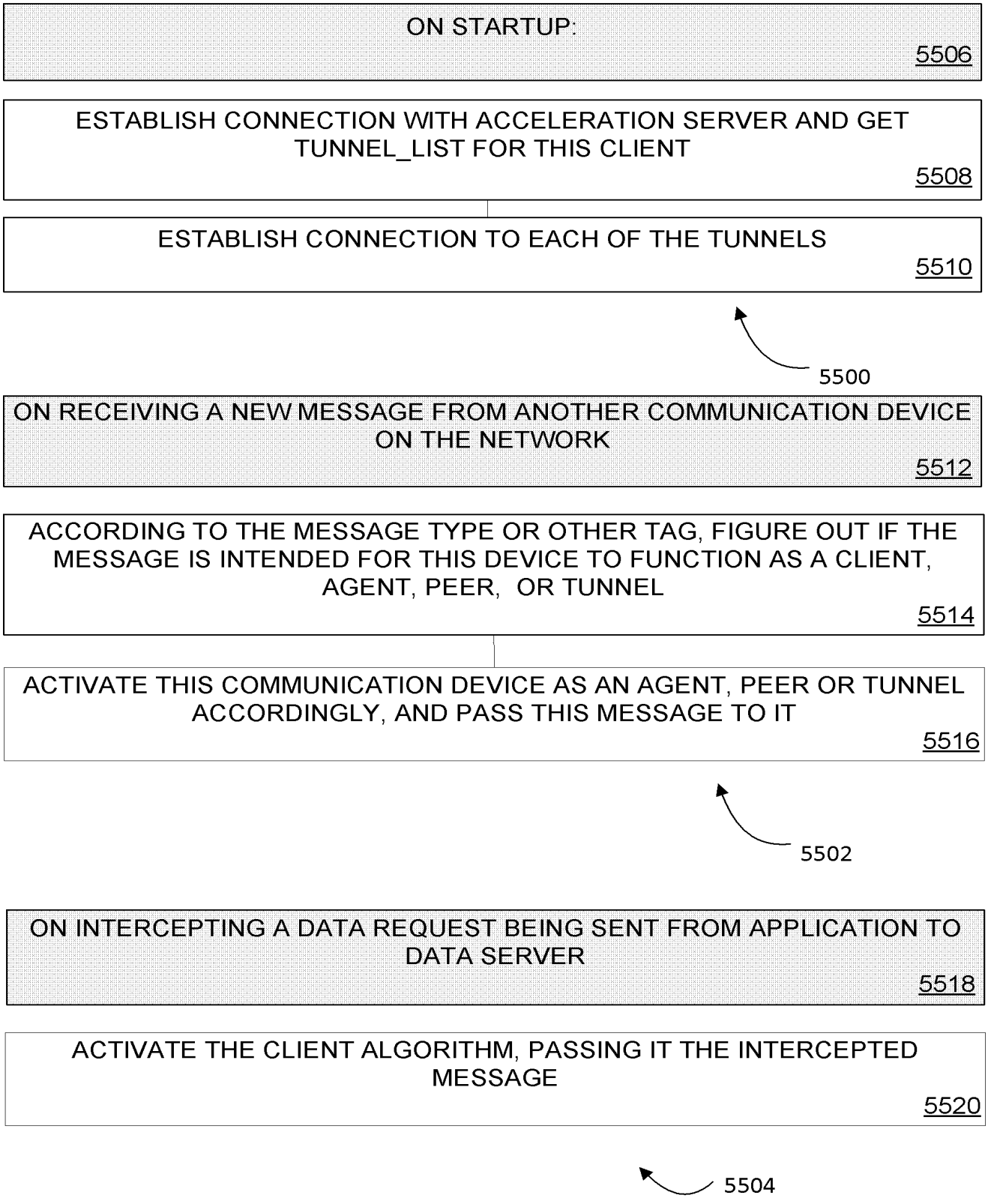


FIG. 56 – DATA_REQUEST STRUCTURE

Entry Fields	Description
DATA_SERVER	THE SERVER THAT IS BEING QUERIED
REQUEST	THE REQUEST POSTED TO THE DATA_SERVER
REQUEST_META_DATA	ADDITIONAL INFORMATION FOR THE REQUEST. FOR EXAMPLE, FOR HTTP THIS WILL INCLUDE THE HTTP HEADERS AND VALIDITY
RESPONSE	THE RESPONSE DATA
RESPONSE_META_DATA	ADDITIONAL INFORMATION FOR THE REQUEST. FOR EXAMPLE, FOR HTTP THIS WILL INCLUDE THE HTTP HEADERS AND VALIDITY
CHECKSUM	THE CHECKSUM OF THE DATA OF THE RESPONSE TO THIS DATA_REQUEST
MAP_LIST	A LIST OF SEGMENTS OF DATA REPRESENTING THE FULL RESPONSE, WHERE FOR EACH SEGMENT, THERE IS A LIST OF PEERS THAT THIS CLIENT KNOWS ABOUT THAT HAVE THAT SEGMENT IN THEIR STORAGE. THIS INCLUDES WHETHER THIS CLIENT DEVICE HAS THESE SEGMENTS STORED
PEER_LIST	LIST OF COMMUNICATION DEVICES THAT MAY BE USED AS PEERS FOR THIS CLIENT. EACH OF THE PEERS IN THE LIST HAS DATA ASSOCIATED WITH IT THAT TELLS THE CLIENT HOW TO COMMUNICATE WITH THAT PEER, AND OTHER META INFORMATION ABOUT IT
TUNNEL_LIST	LIST OF COMMUNICATION DEVICES THAT MAY BE USED TO ASSIST IN LOADING THIS DATA_REQUEST FROM THE DATA_SERVER
AGENT_LIST	LIST OF COMMUNICATION DEVICES THAT MAY BE USED AS AGENTS FOR THIS REQUEST

FIG. 58 – USE OF TUNNELS

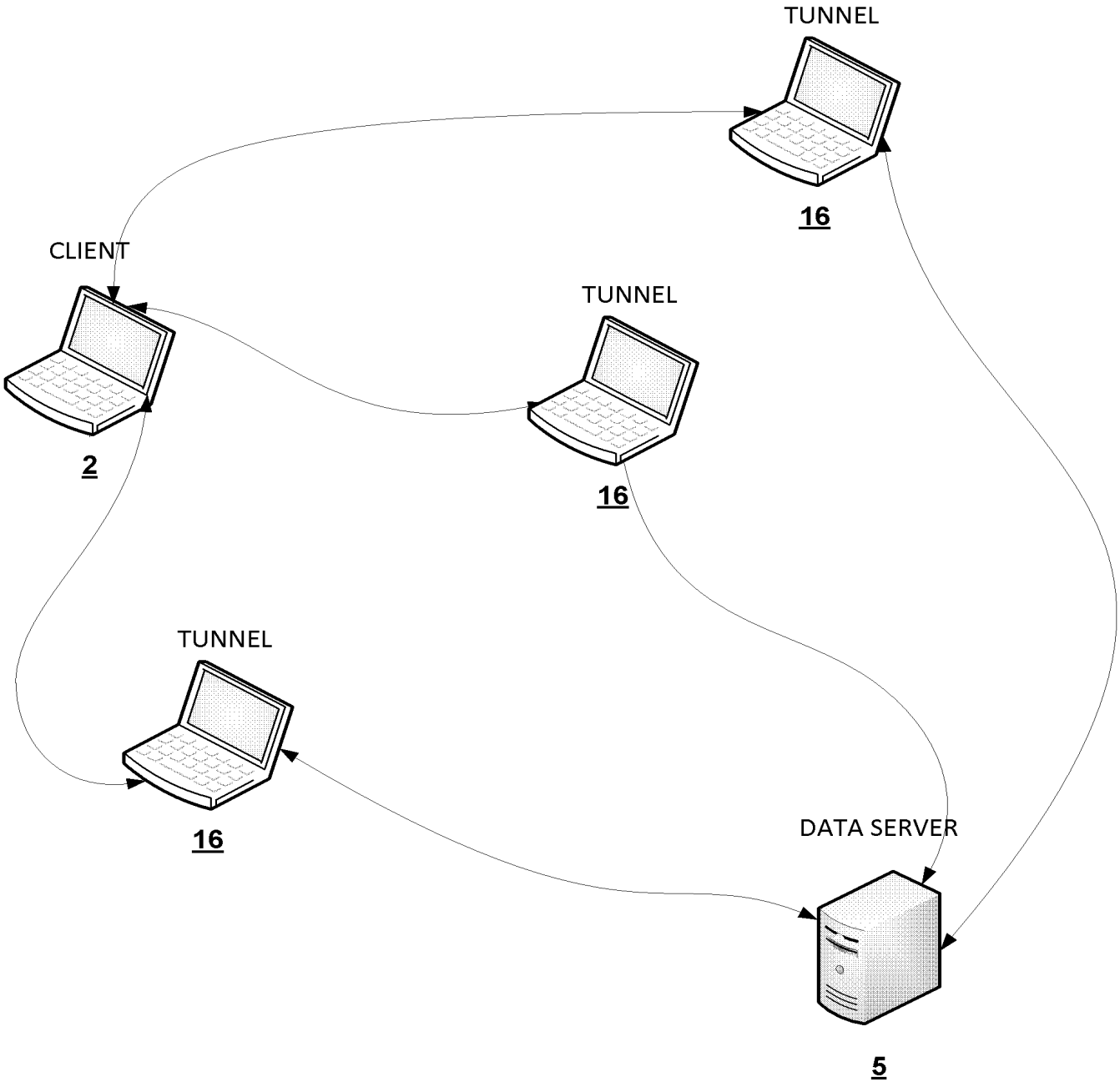


FIG. 61 – LOADING DATA THROUGH PEERS OR THROUGH TUNNELS - FLOWCHART

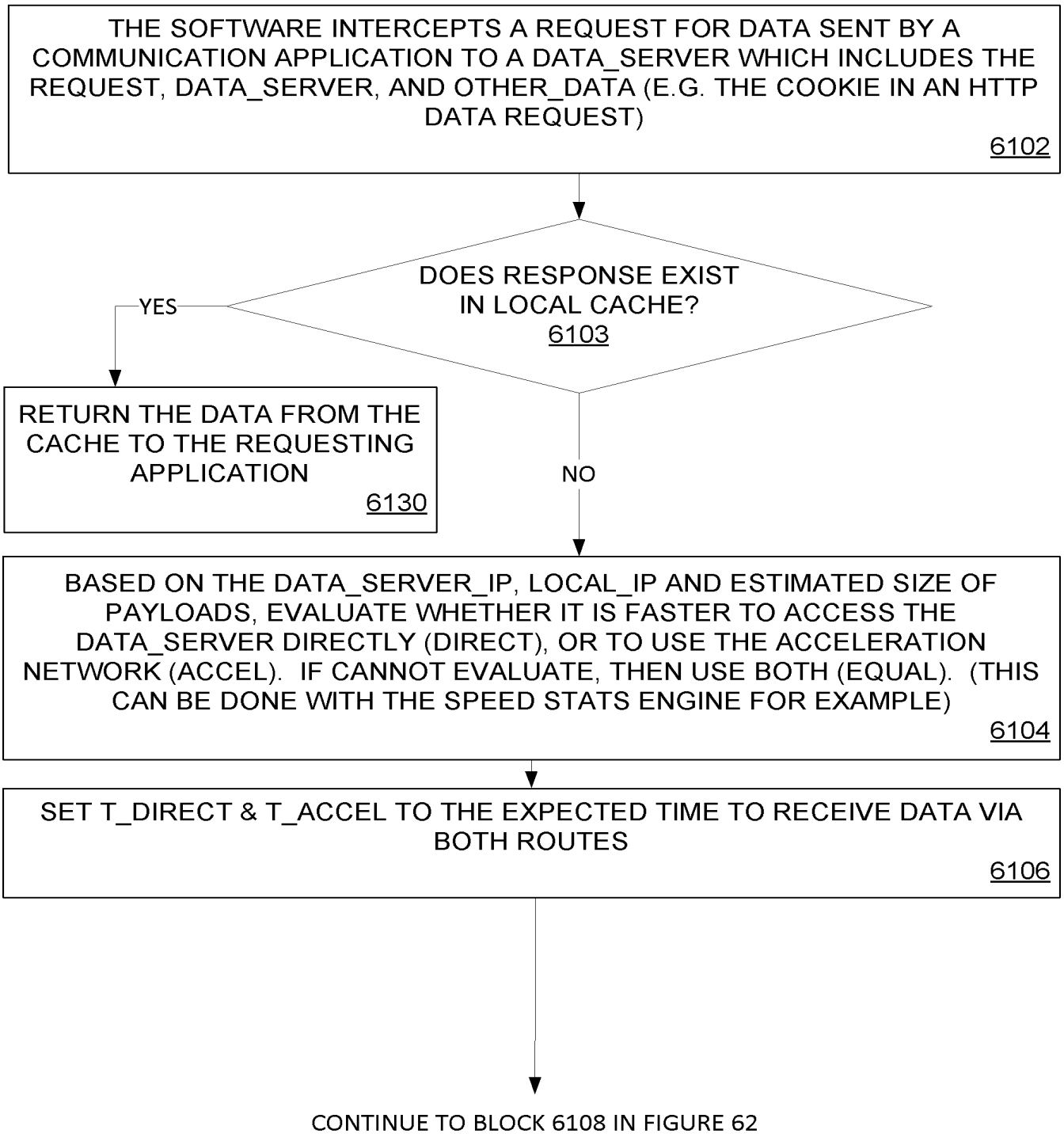


FIG. 62 – TUNNELS VS. PEERS FLOWCHART (CONT.)

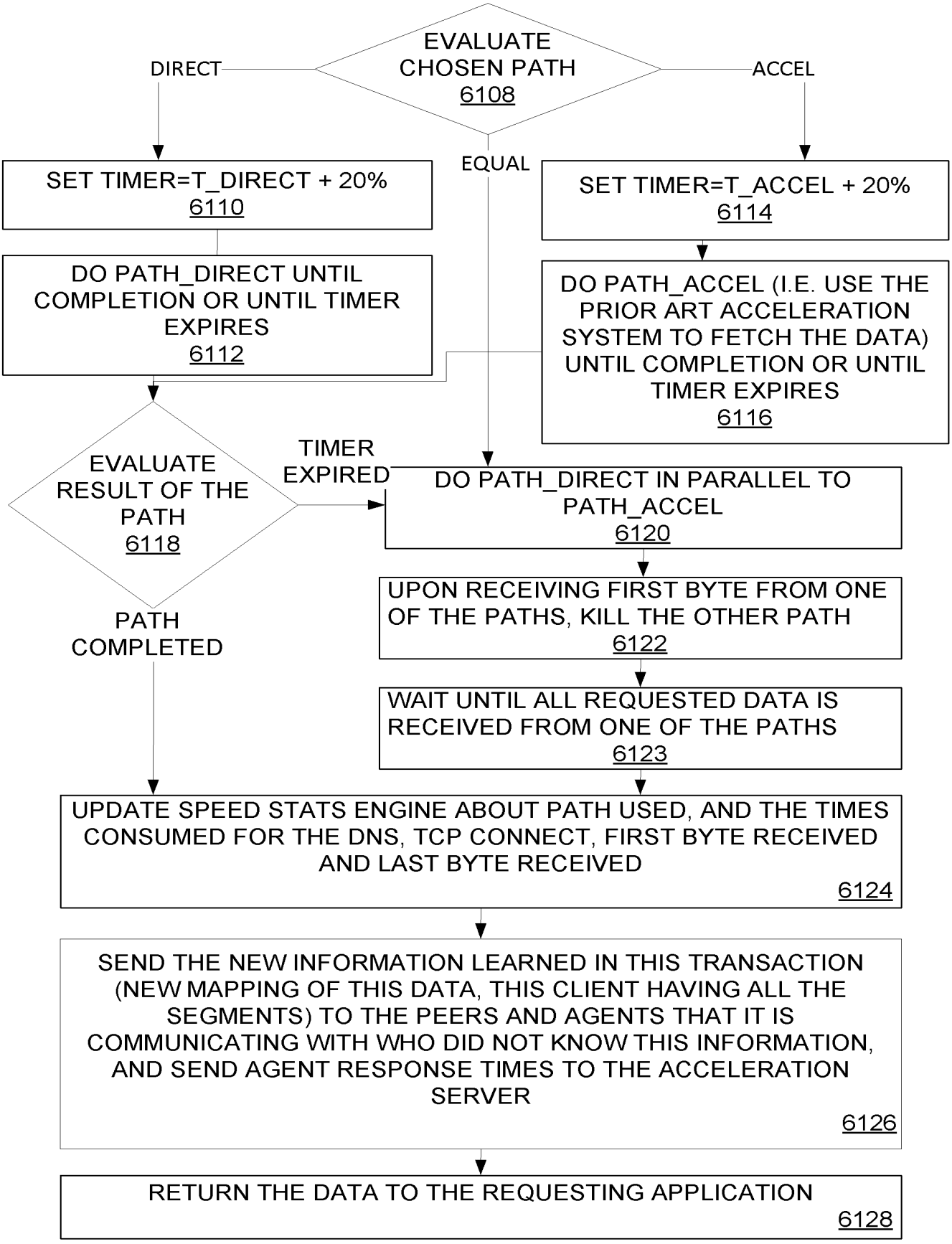


FIG. 63 – ALGORITHMS FOR SELECTING THE TUNNELS

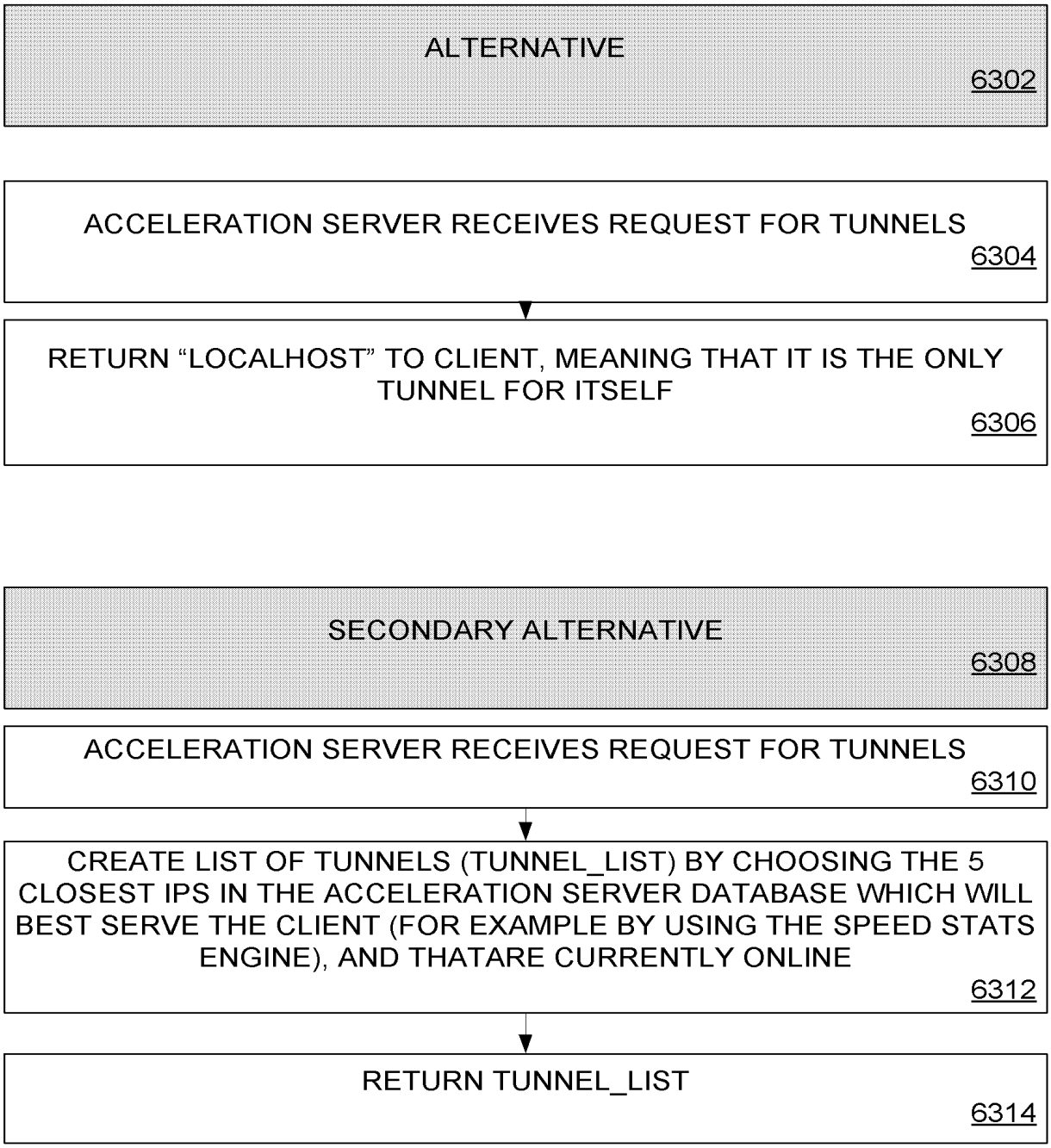


FIG. 64 – SPEED STATS ENGINE

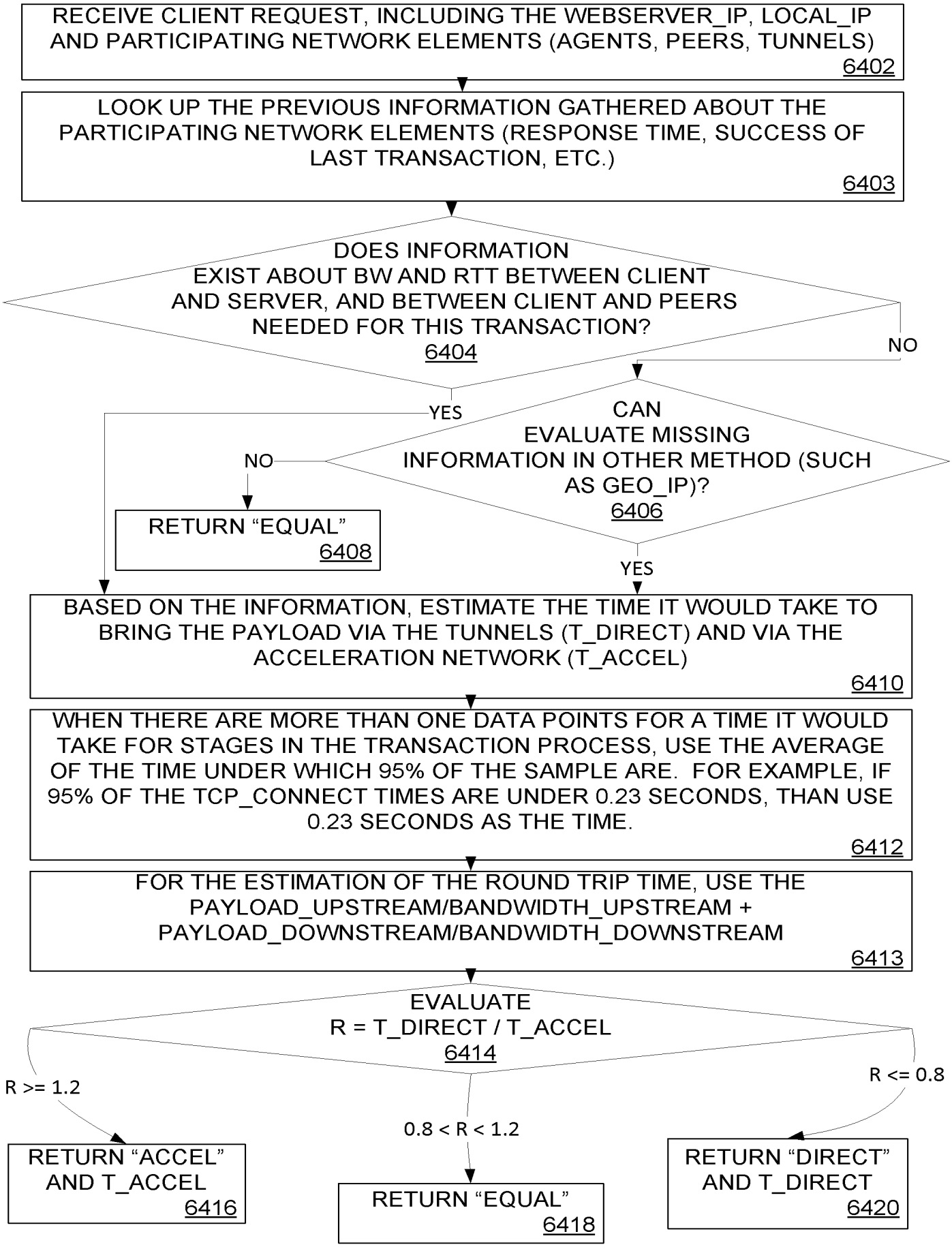


FIG. 69 – PATH_DIRECT

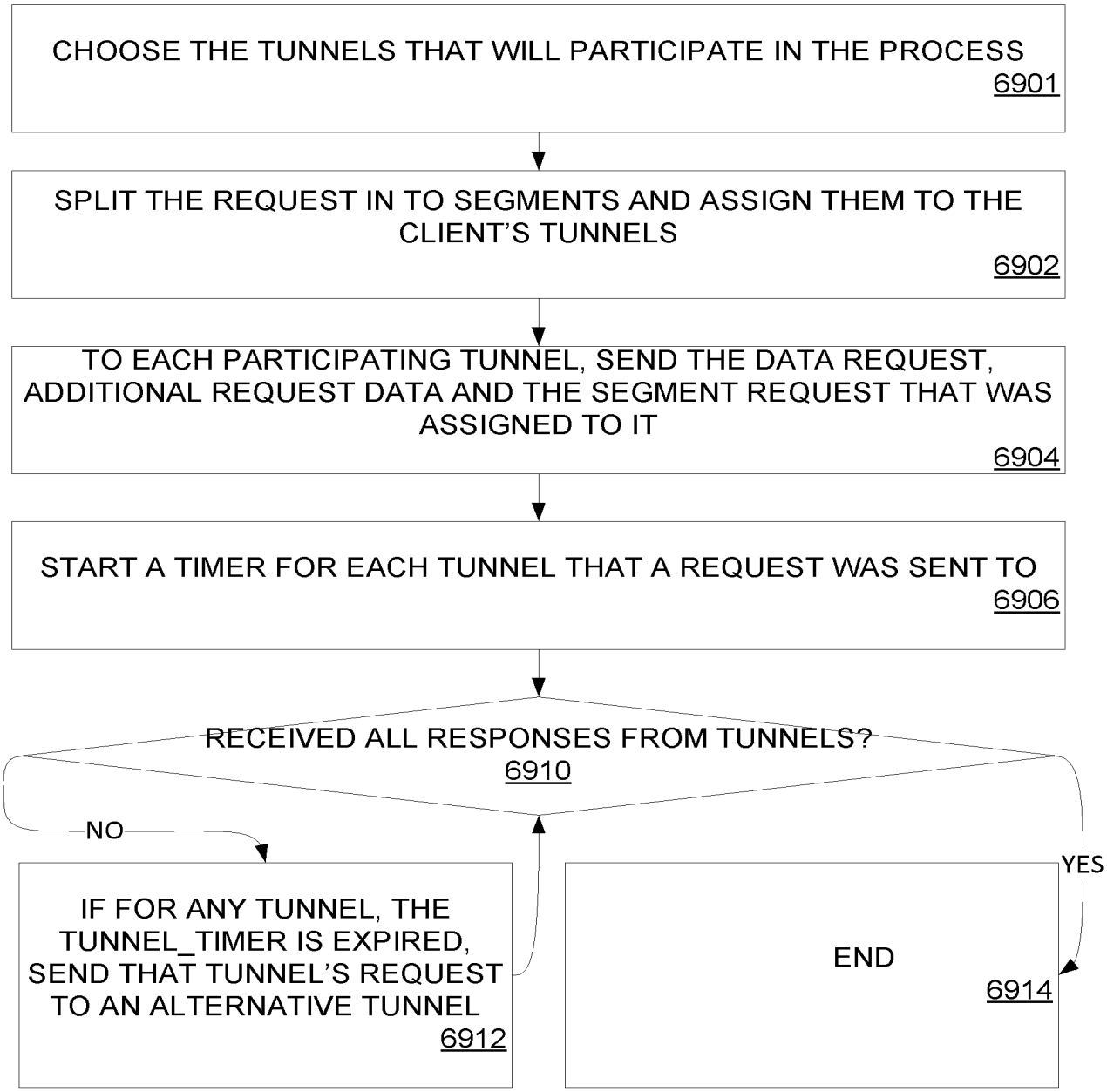


FIG. 70 – ACCELERATION TUNNEL

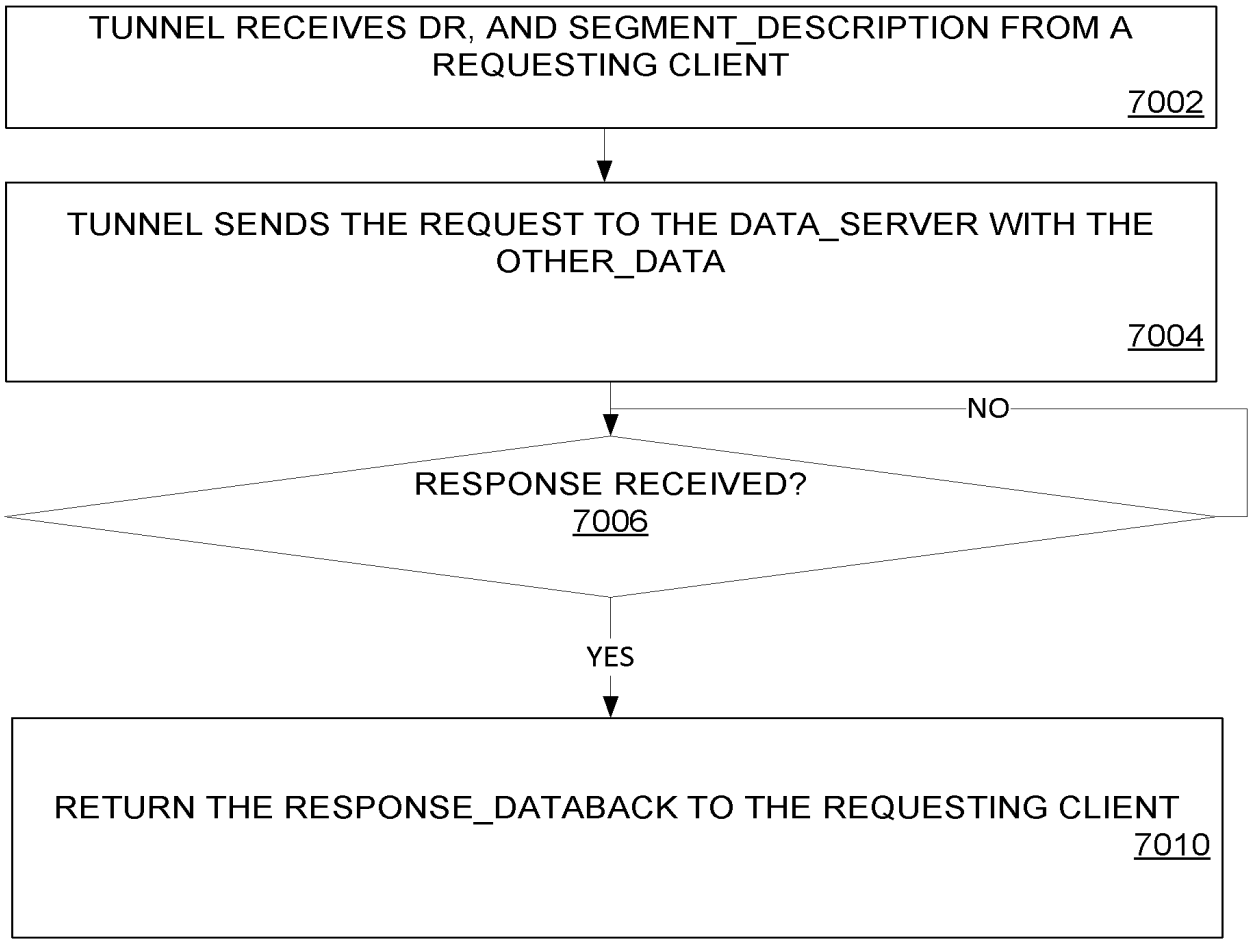
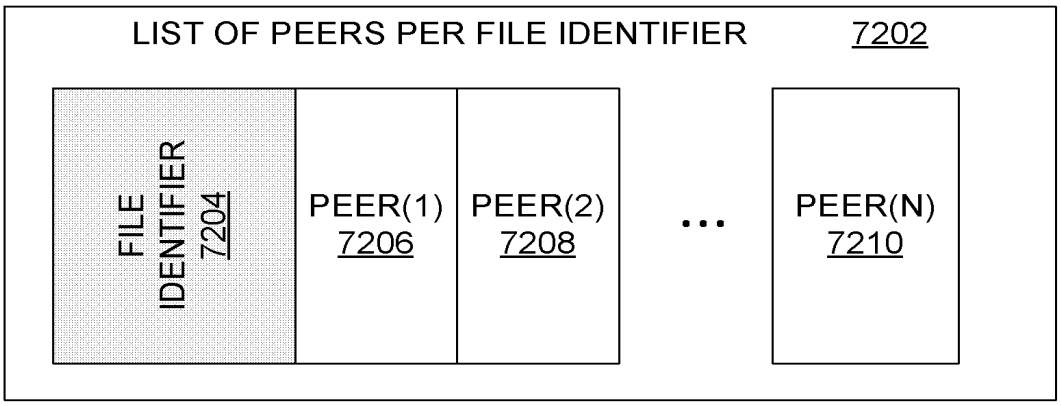


FIG. 72 – PEER MAPS – PRIOR ART - DIAGRAMS

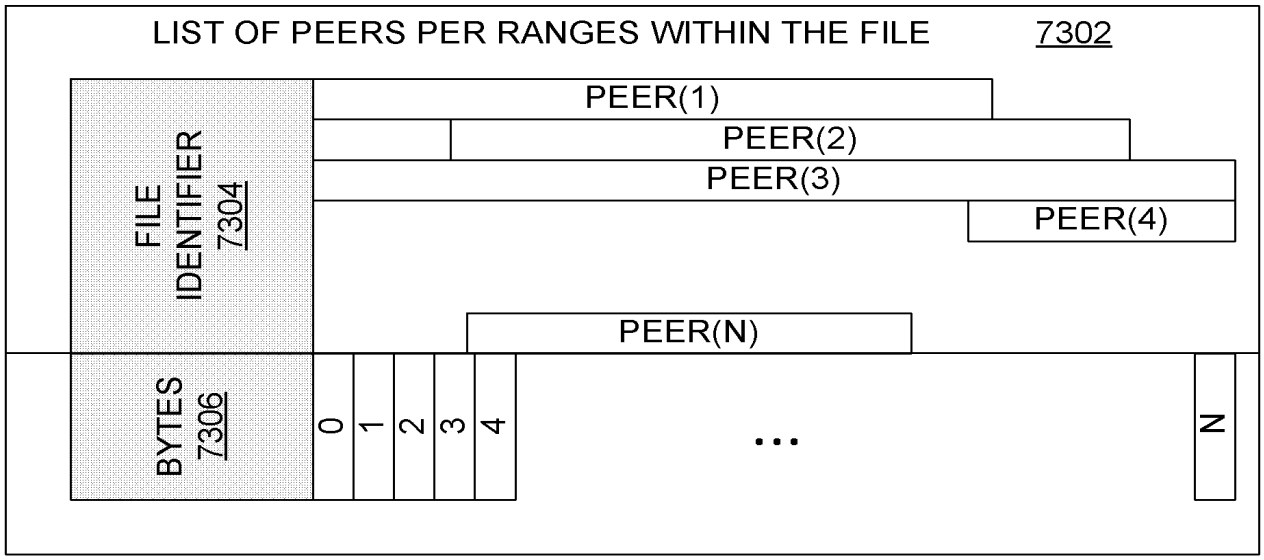
PRIOR ART – FOR EACH FILE, A LIST OF PEERS THAT HAVE CACHED ALL OF THE FILE



PROBLEM WITH PRIOR ART: ONLY PEERS THAT HAVE CACHED EVERY BYTE OF THE FILE CAN SERVE AS PEERS FOR A CLIENT REQUEST FOR THE FILE

FIG. 73 – PEER MAPS – DIAGRAMS

PEER MAP – FOR EACH FILE, A LIST OF PEERS THAT HAVE PARTIAL OR COMPLETE RANGES OF THE DATA FOR THAT FILE



BENEFIT: PEERS THAT HAVE CACHED EVEN PART OF THE FILE CAN SERVE AS PEERS FOR A CLIENT REQUEST FOR THIS FILE, THUS PROVIDING MORE PEERS ON AVERAGE PER REQUEST

FIG. 74 – PEER MAPS - FLOWCHART

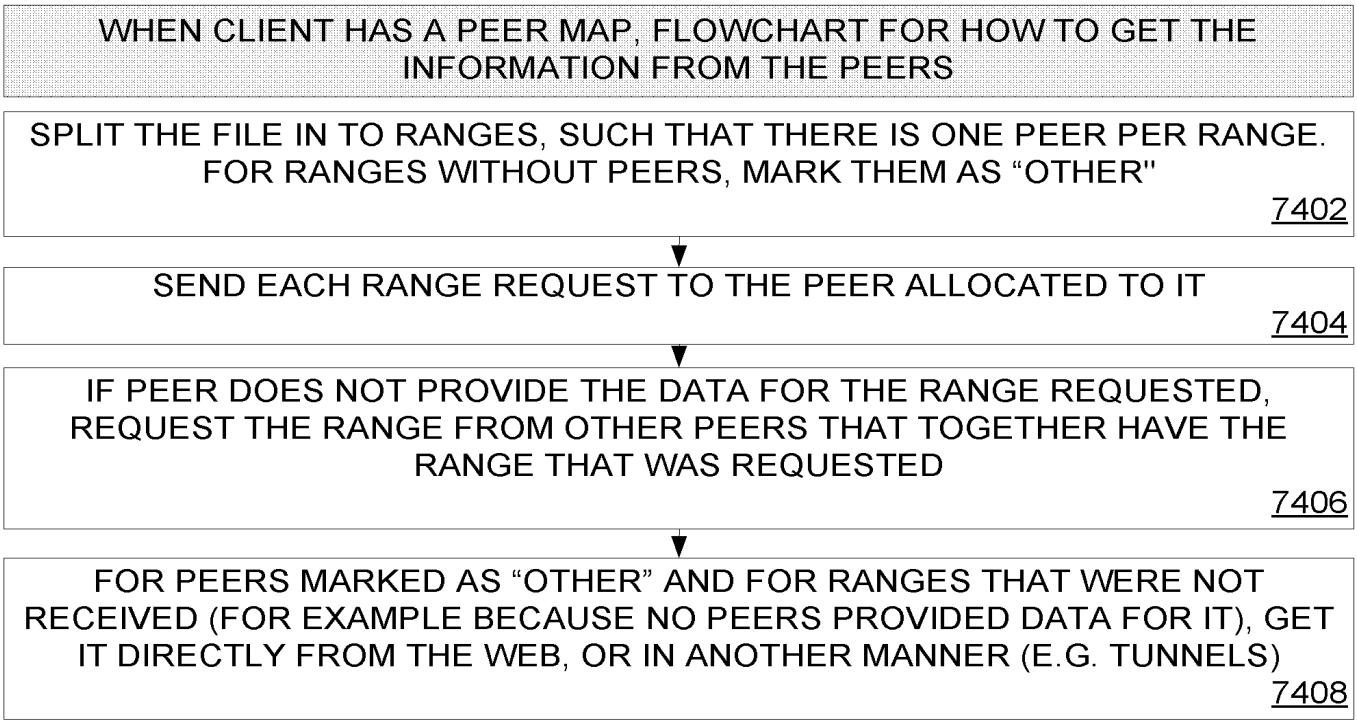


FIG. 76 – PEER MAPS – ALLOCATING RANGES TO PEERS BY BW & RTT - FLOWCHART

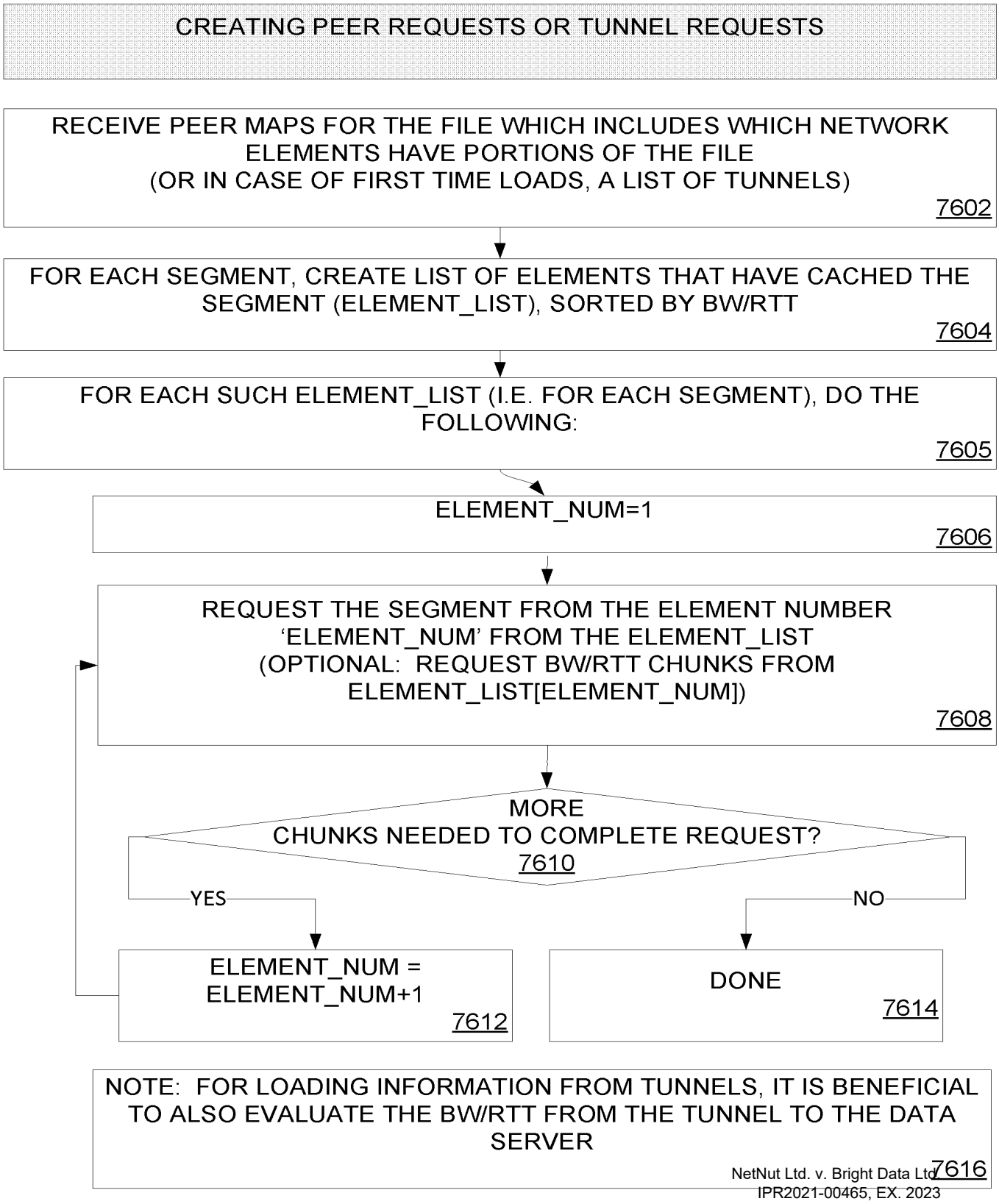


FIG. 78 – PEER SELECTION BY AGENT

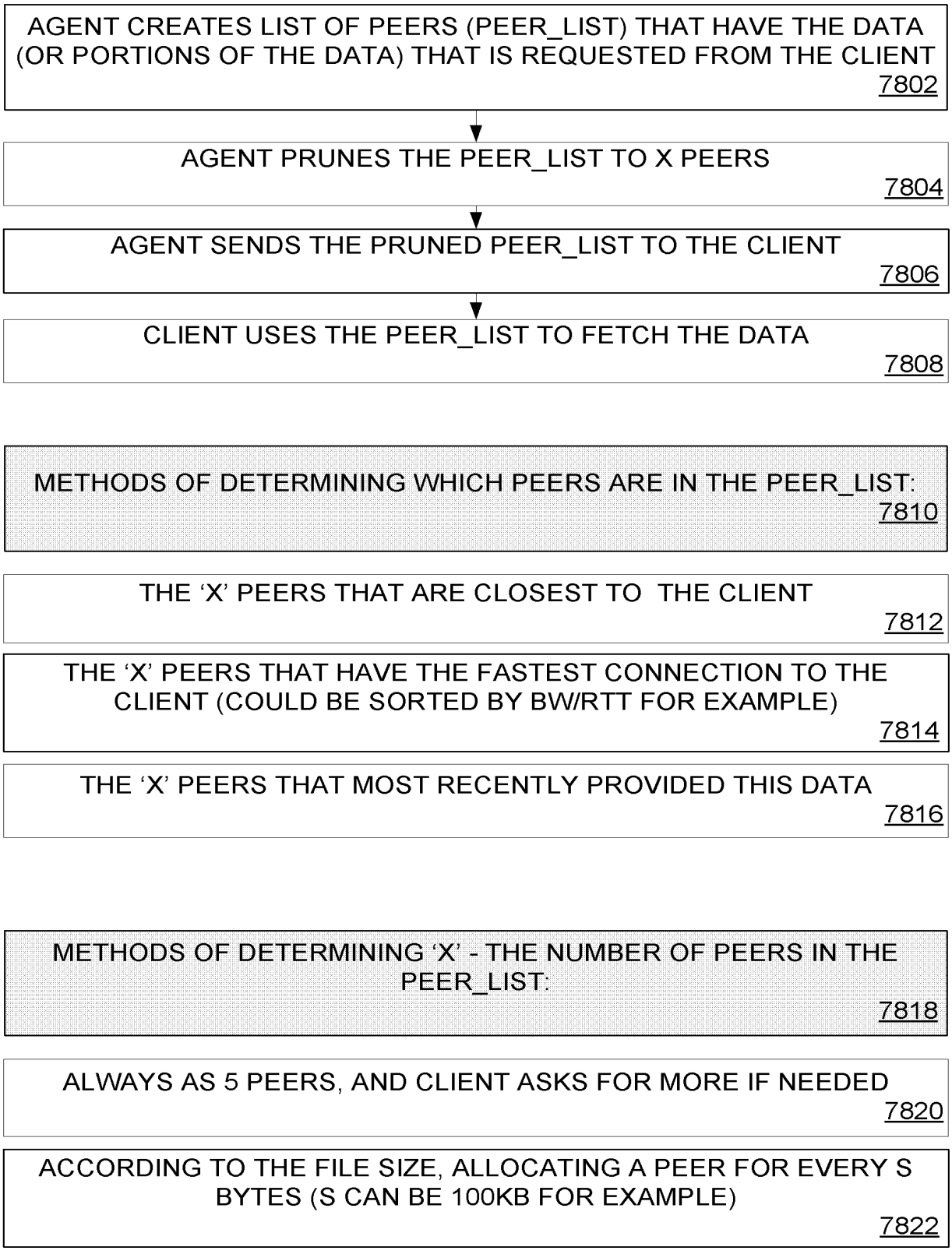


FIG. 80 – PEER ALLOCATION ACCORDING TO BW/RTT

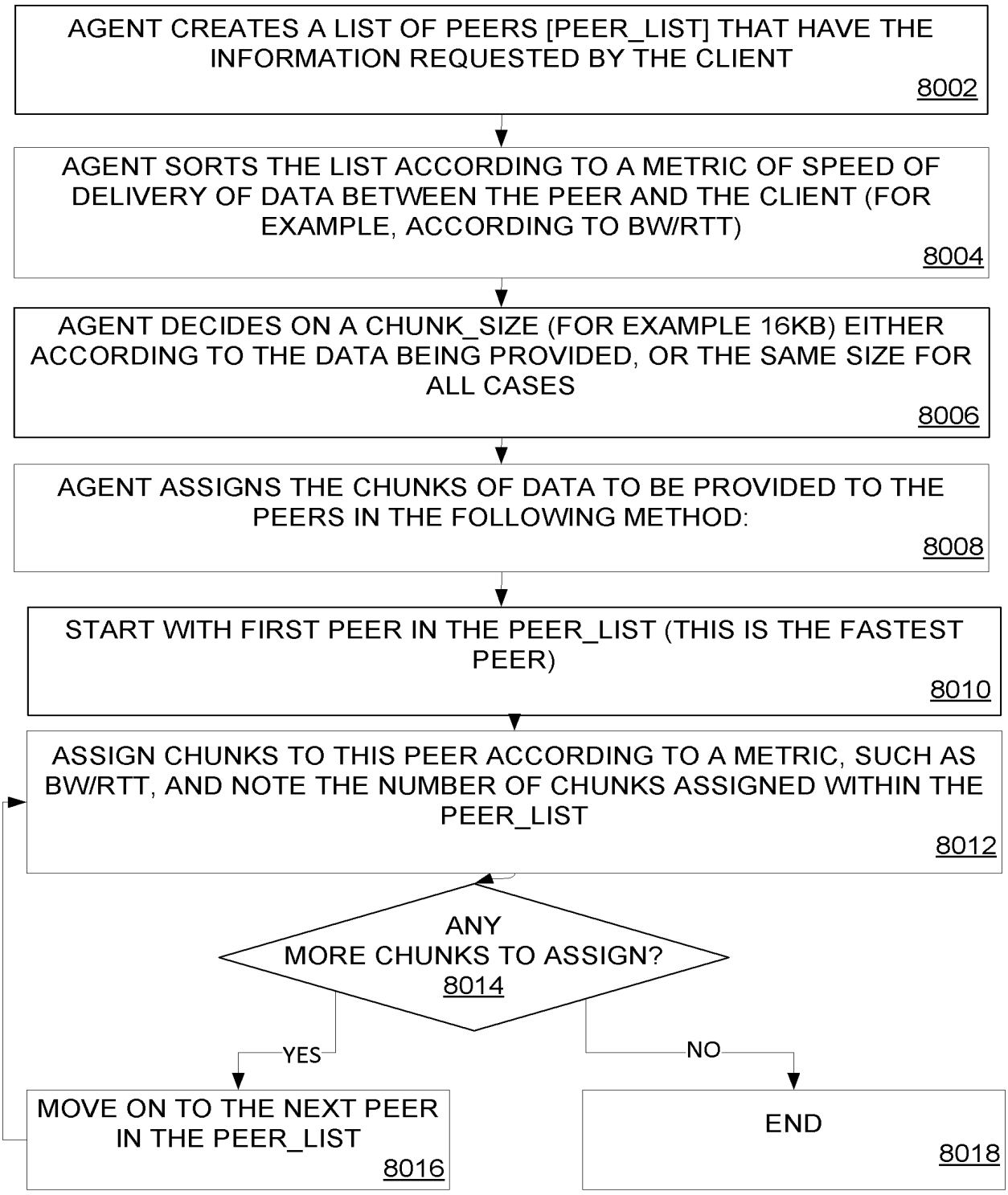


FIG. 90 – CACHING PREVIOUSLY USED AGENTS

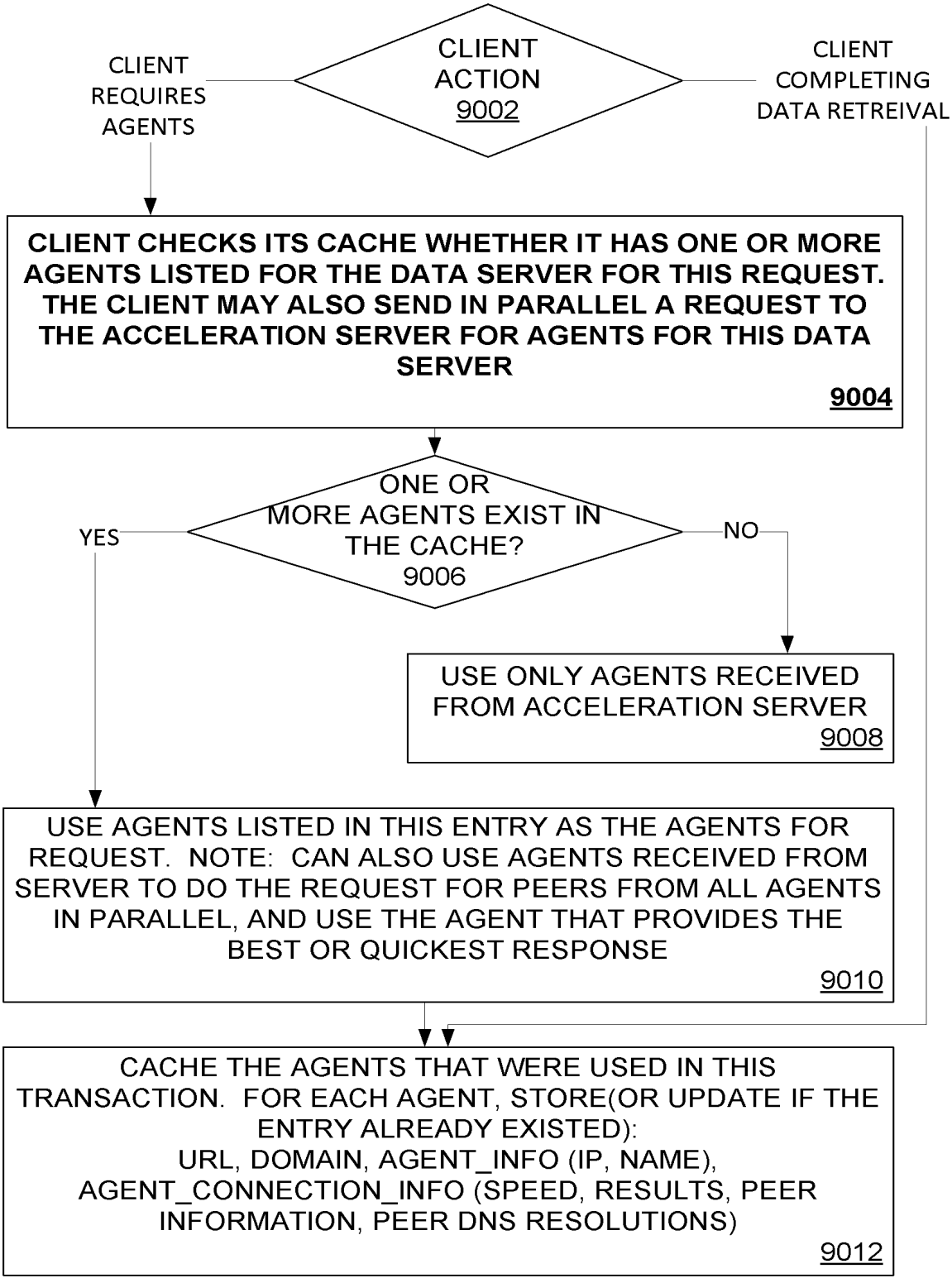


FIG. 92 – PASSIVE AGENTS – USING KNOWLEDGEABLE PEERS AS AGENTS

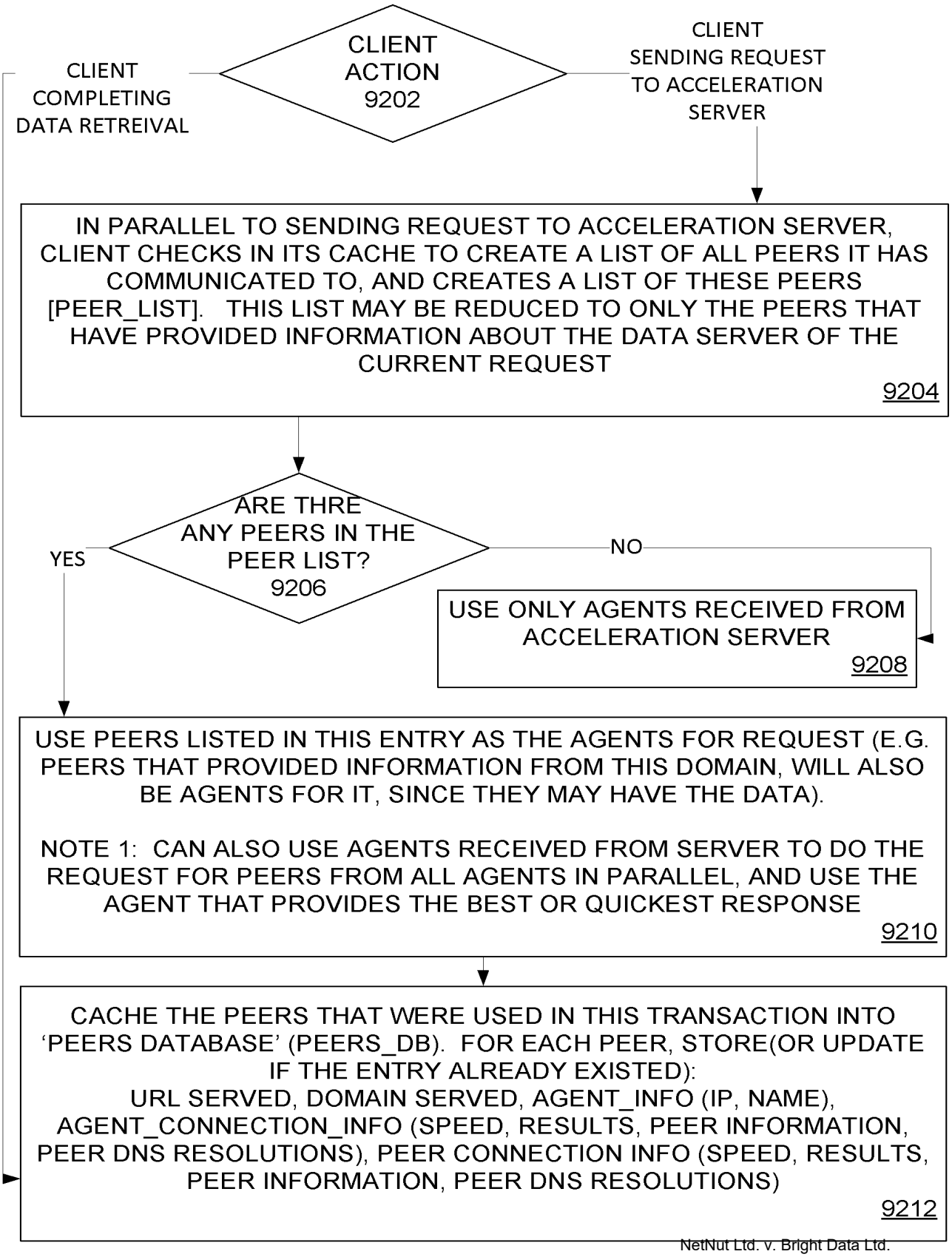


FIG. 94 – INLINE DATA – CLIENT SIDE

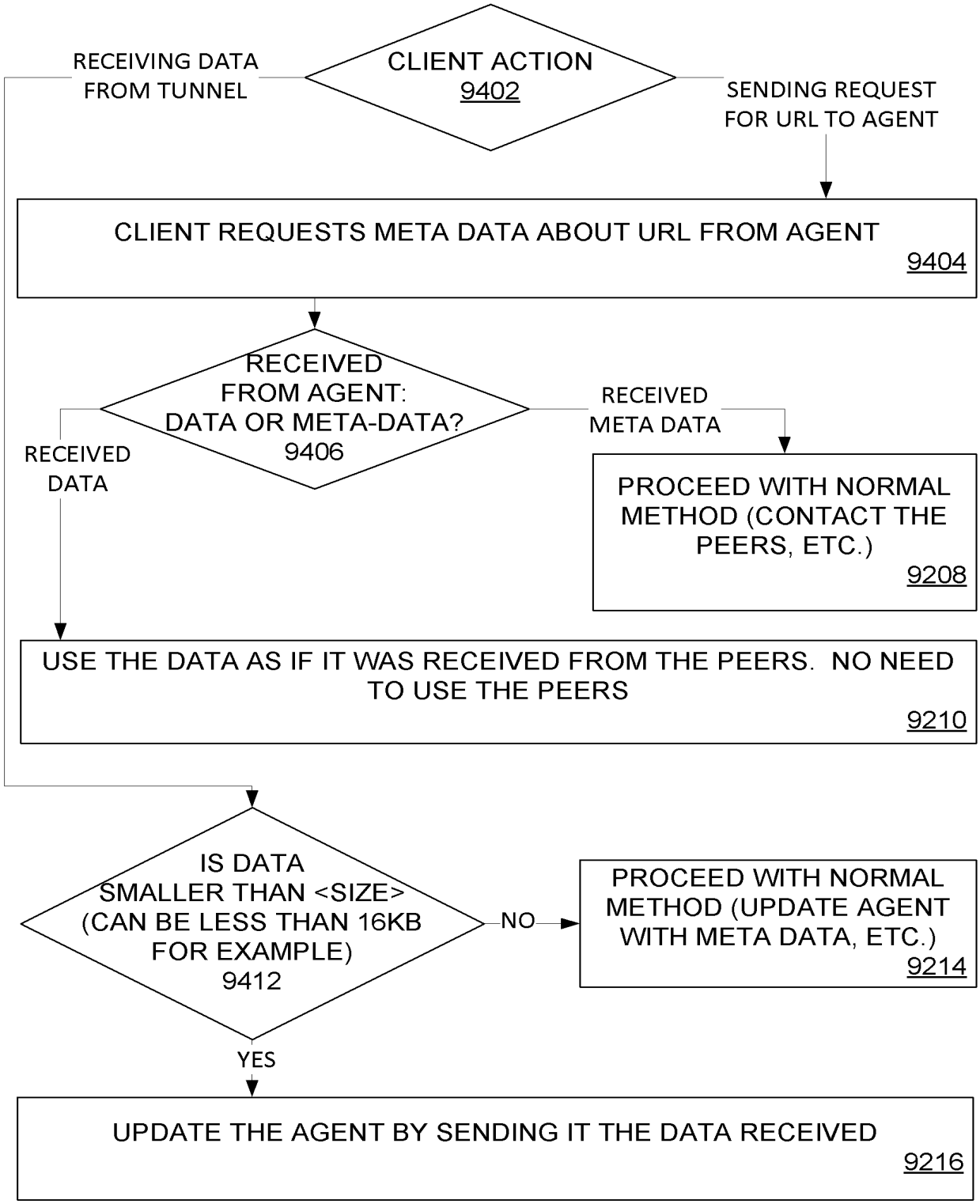


FIG. 100 – INLINE DATA – AGENT SIDE

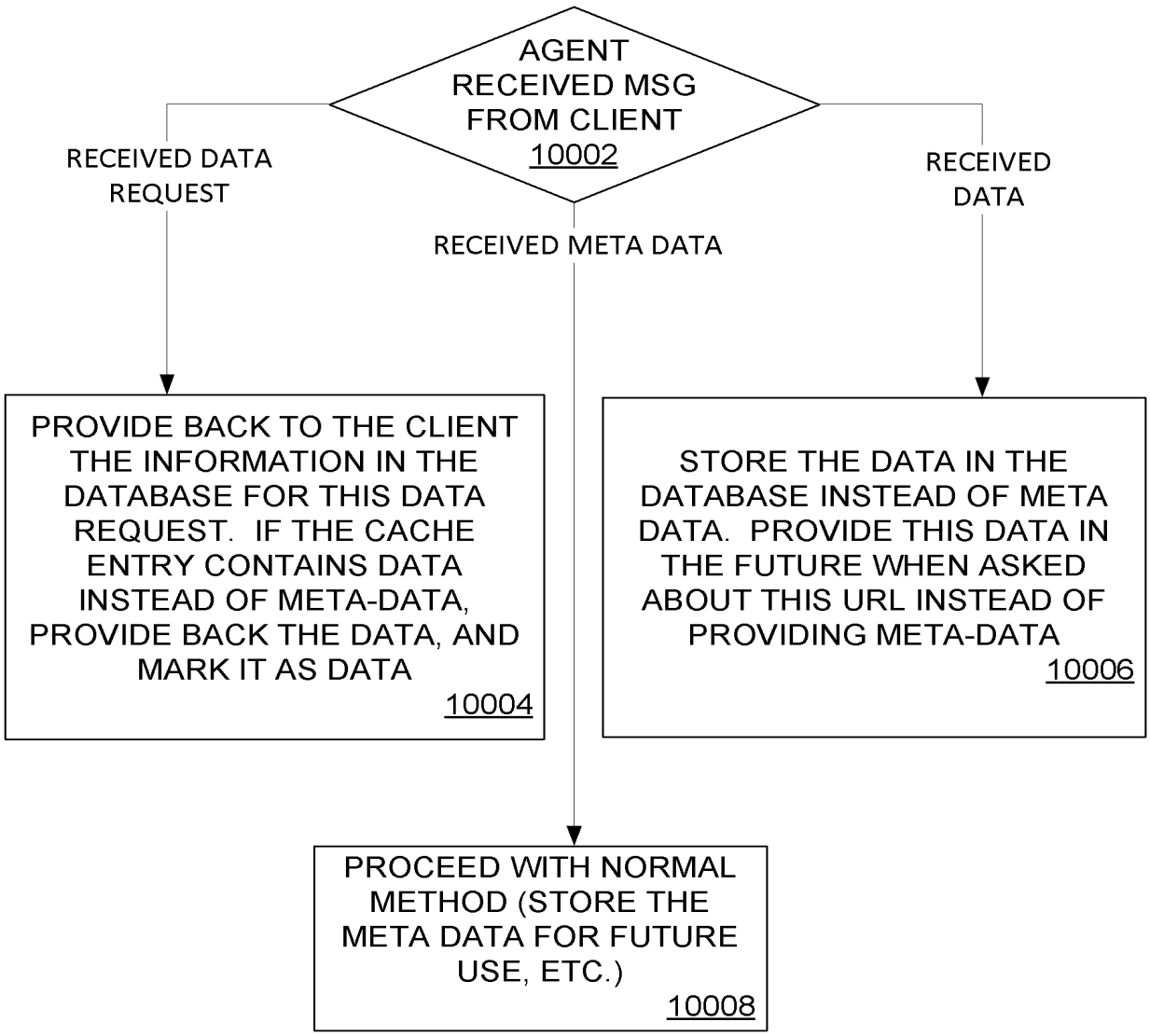


FIG. 102 – TUNNEL CACHING – NETWORK SETUP

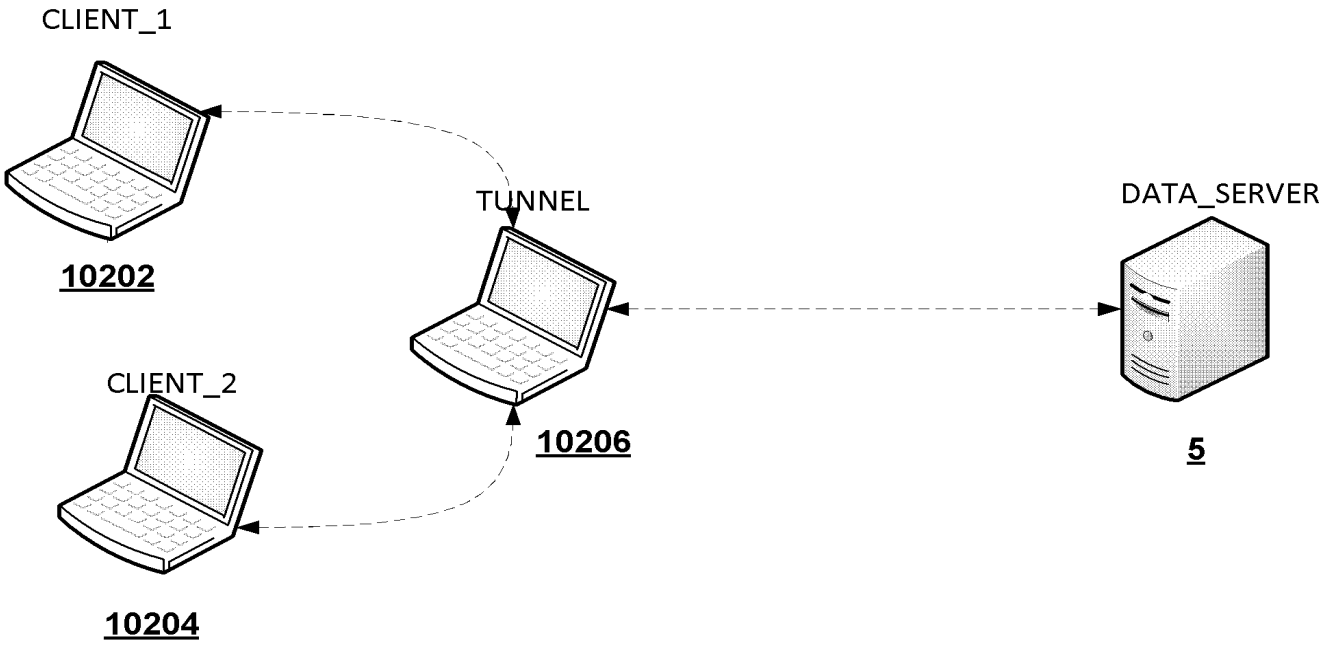


FIG. 104 – TUNNEL CACHING – FLOWCHART

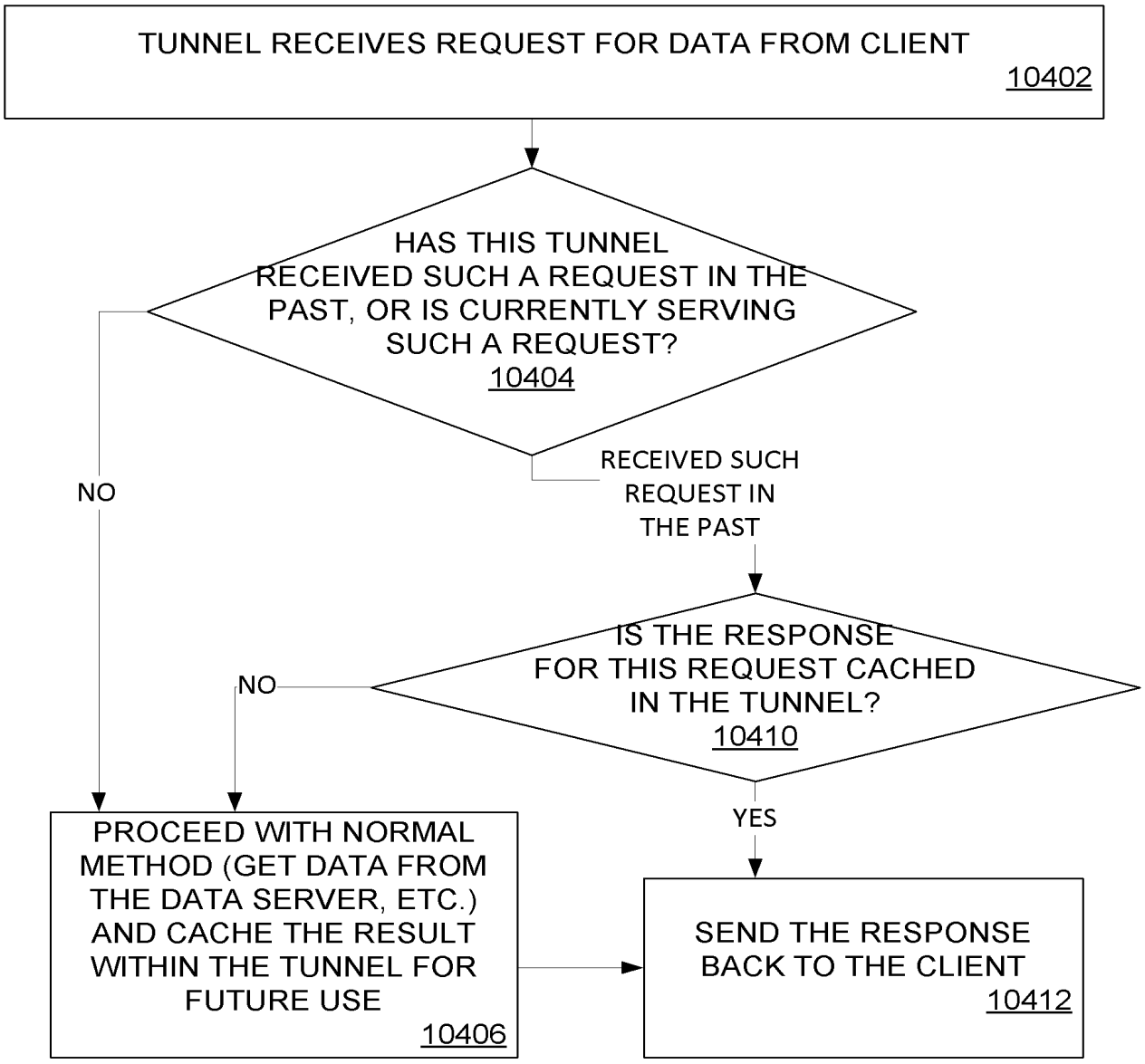
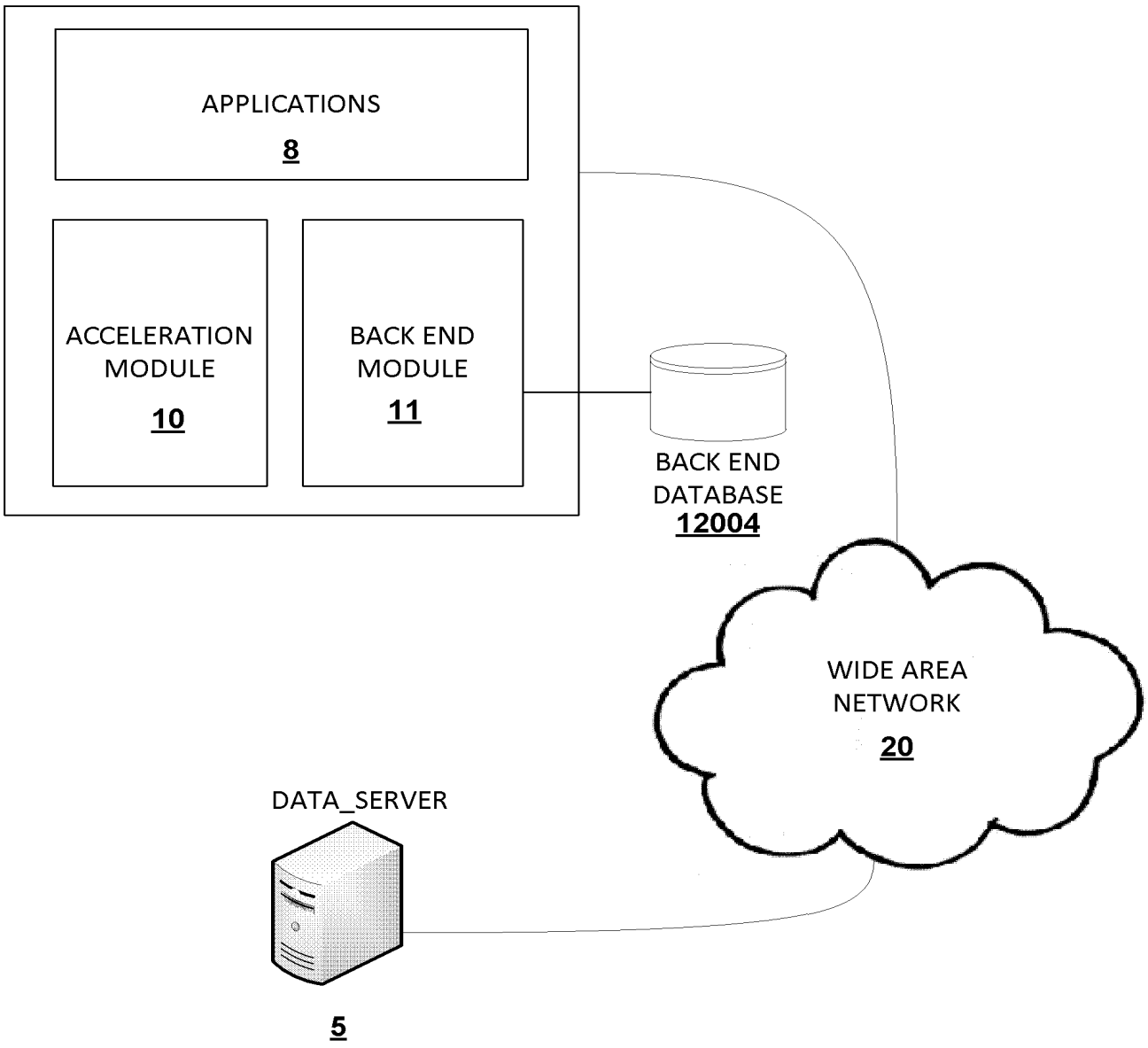


FIG. 120 – USING BACK ENDS TO CUSTOMIZE SPECIFIC DATA REQUESTS - DIAGRAM



CUSTOMIZE BY:

- URL
- DOMAIN
- DATA SERVER IP
- TYPE OF FILE (.JPG, .FLV)
- TIME OF DAY
- GEOGRAPHY OF CLIENT OR SERVER OR BOTH

12002

FIG. 122 – USING BACK ENDS TO CUSTOMIZE SPECIFIC DATA REQUESTS - FLOWCHART

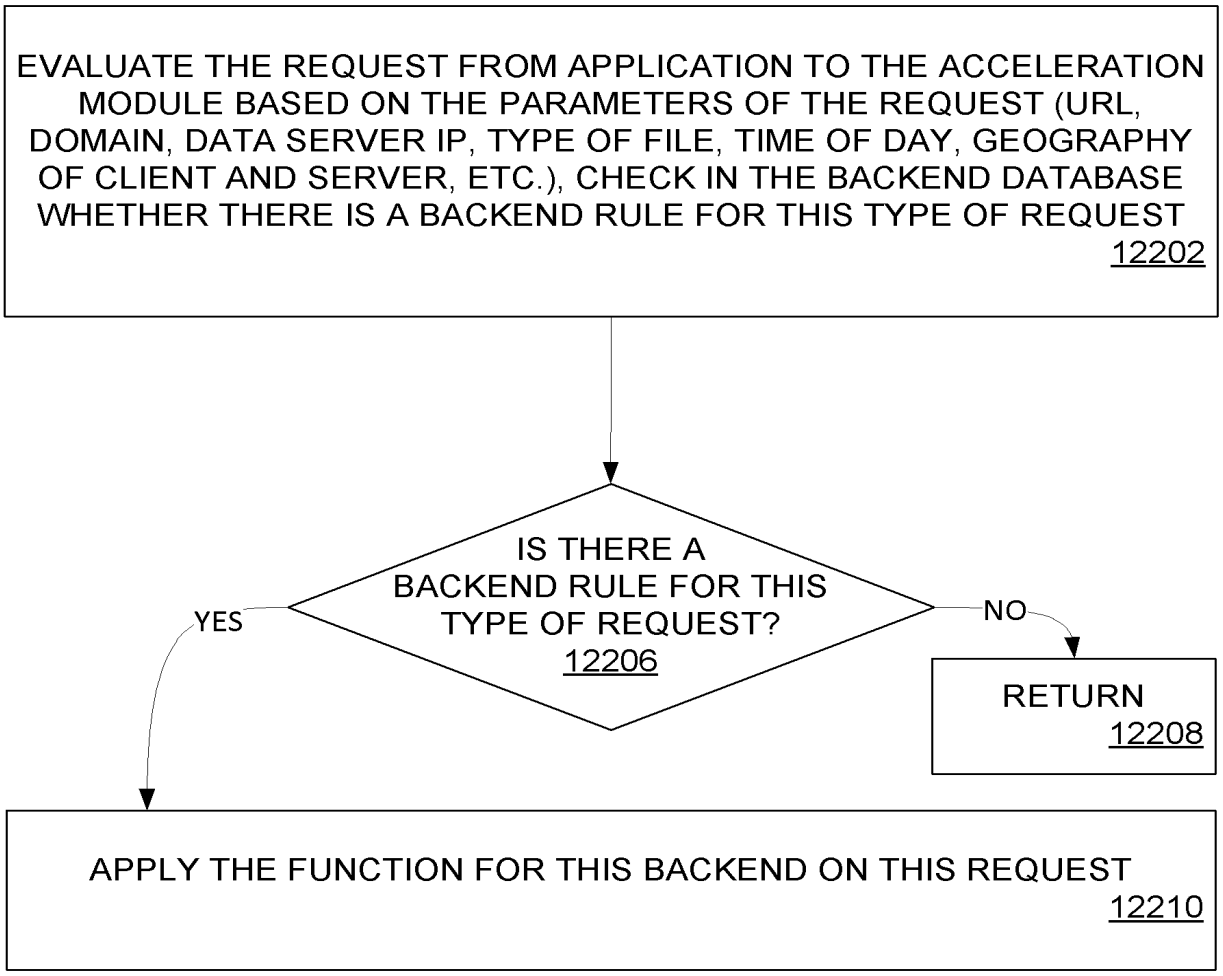


FIG. 124 – SPECIFIC BACK END – USAGE STATS BACKEND

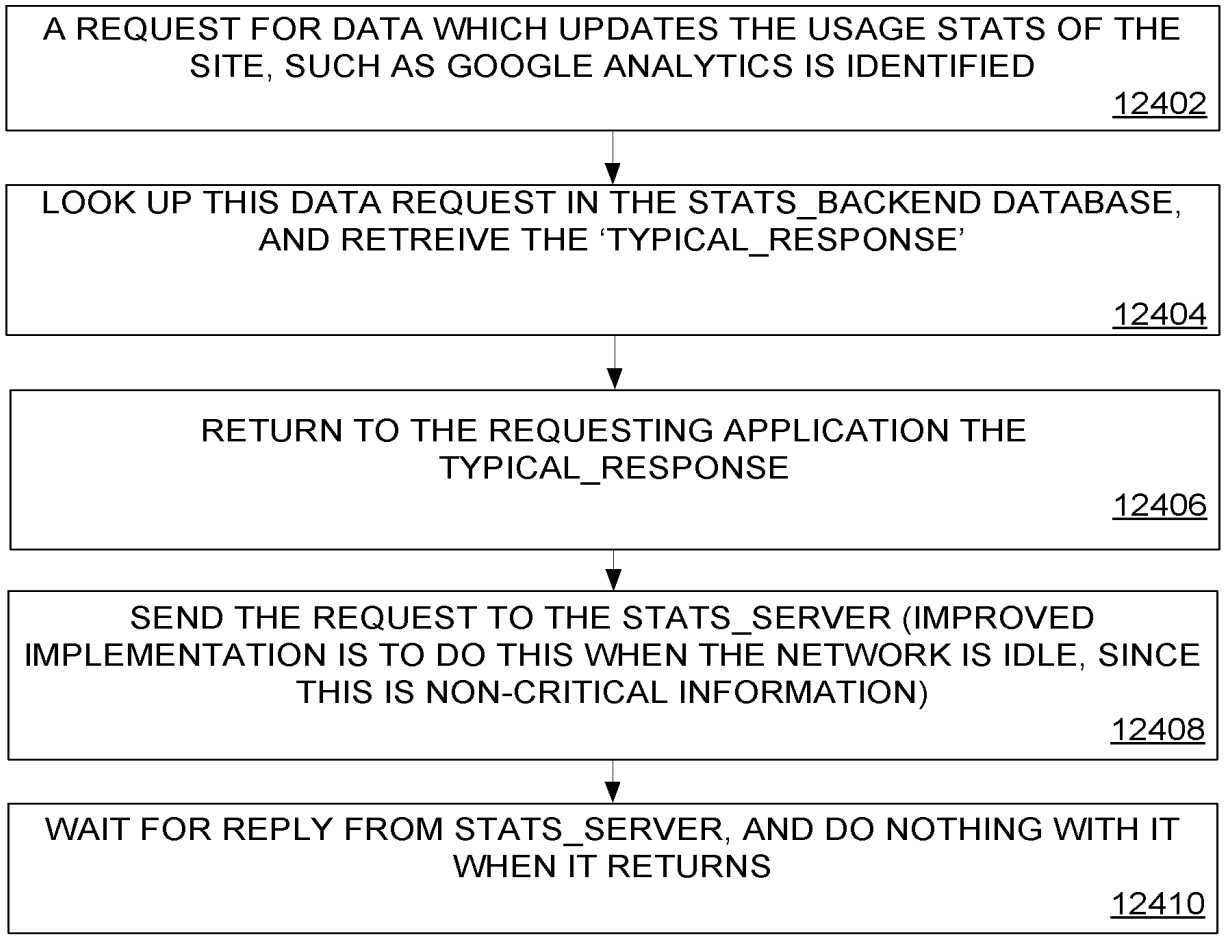
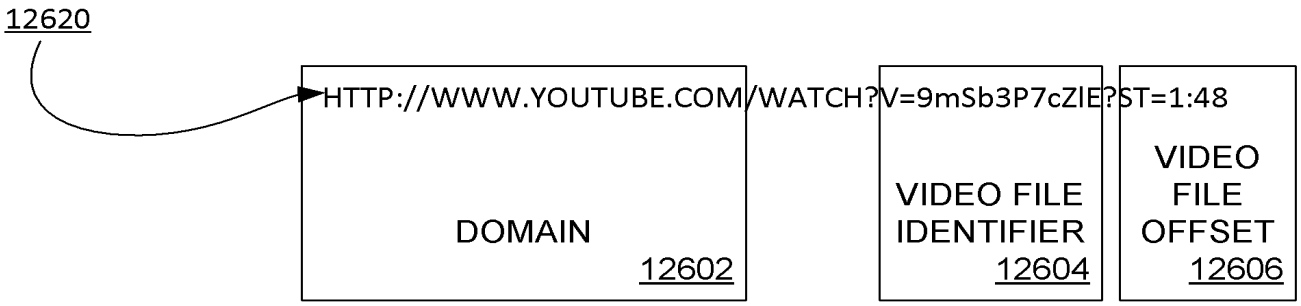


FIG. 126 – SPECIFIC BACK END – VIDEO OFFSET BACKEND – PRIOR ART



PROBLEM #1: IF A VIDEO FILE IS REQUESTED WITH AN OFFSET, IT CANNOT BE CACHED AS A URL SINCE EVERY DIFFERENT OFFSET IS A DIFFERENT URL.

12608

PROBLEM #2: OFFSETS CAN BE BY TIME OFFSET, BYTE OFFSET OR RELATIVE OFFSET. FOR CACHING AND RETREIVAL, HOW IS THE NORMALIZATION BETWEEN THESE DONE?

12610

FIG. 128 – SPECIFIC BACK END – VIDEO OFFSET BACKEND – CACHING BY URL

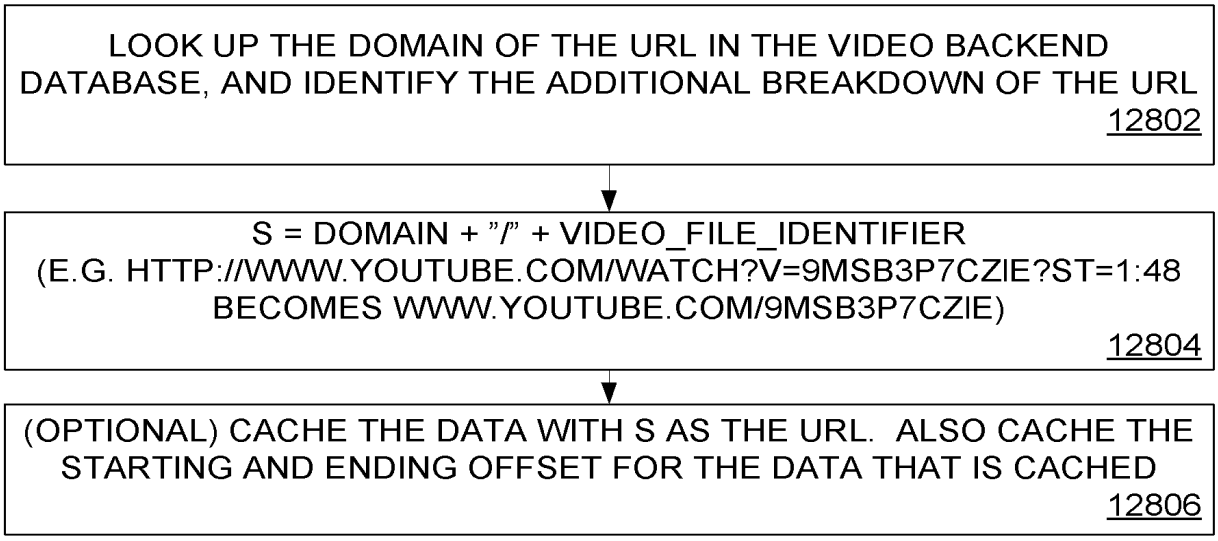
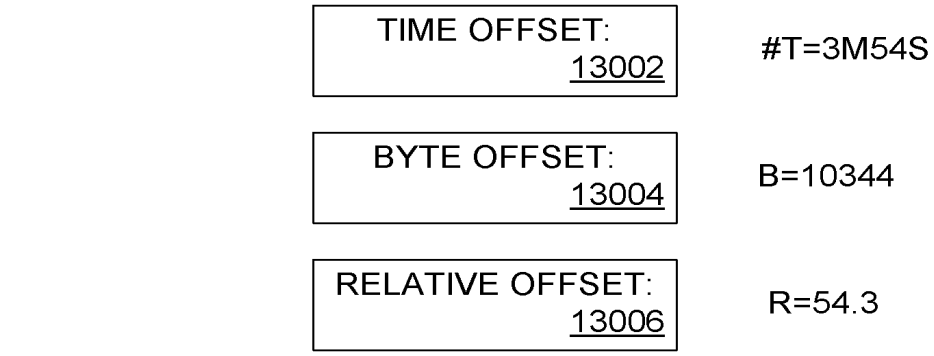


FIG. 130 – SPECIFIC BACK END – VIDEO OFFSET BACKEND – NORMALIZING OFFSET



URL WITH OFFSET IS REQUESTED OR RECEIVED
(E.G. HTTP://WWW.YOUTUBE.COM/WATCH?V=9MSB3P7CZIE?ST=1:48)
13008

↓

NORMALIZE TO URL TO NEW_URL ACCORDING TO THE ABOVE 'CACHING BY URL' ALGORITHM
(E.G. HTTP://WWW.YOUTUBE.COM/WATCH?V=9MSB3P7CZIE?ST=1:48 BECOMES WWW.YOUTUBE.COM/9MSB3P7CZIE)
13010

↓

SEND REQUEST (NEW_URL) TO DATA SERVER, AND IMMEDIATELY CLOSE IT, TO GET FULL FILE INFO, INCLUDING CONTENT LENGTH (LENGTH_SOURCE_UNITS) IN THE UNITS OF THE OFFSET RECEIVED AND THE CONTENT LENGTH (LENGTH_DEST_UNITS) IN THE UNITS OF THE OFFSET DESIRED
13012

↓

RELATIVE_OFFSET = TIME_OFFSET / TOTAL TIME
OR BYTE_OFFSET / TOTAL BYTES
OR RELATIVE OFFSET RECEIVED
13014

↓

NEW_ OFFSET = RELATIVE_OFFSET * CONTENT_LENGTH
13016

↓

USE NEW_URL AND NEW_OFFSET FOR CACHING OR RETREIVING FILE DATA FROM THE CACHE (IF CACHING THE DATA, CACHE THE DATA ONLY, WITHOUT THE HEADERS)
13018

FIG. 136 – INCREASING MAXIMUM NUMBER OF CONNECTIONS TO DATA SERVER VIA TUNNELS - DIAGRAM

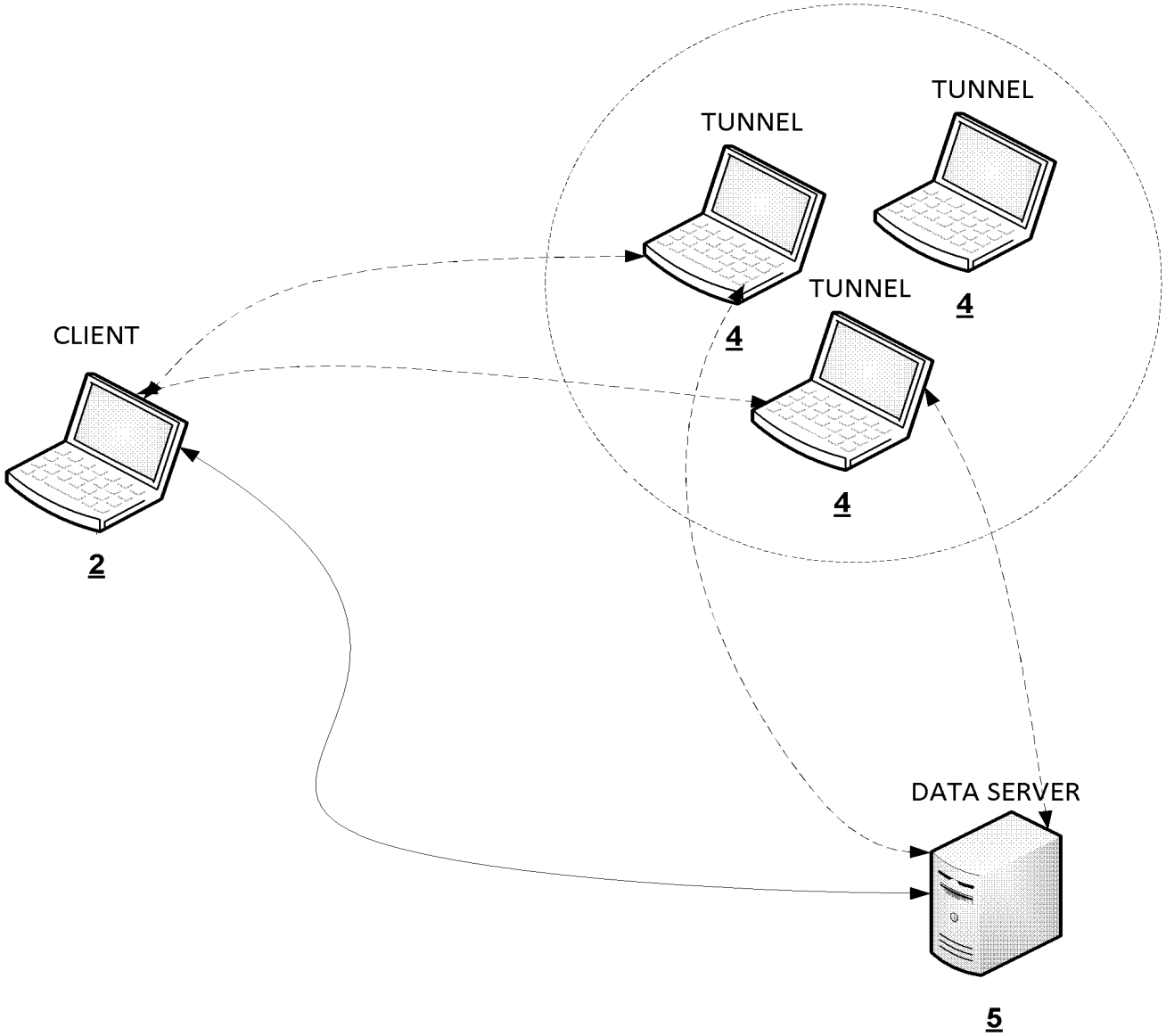


FIG. 138 – INCREASING MAXIMUM NUMBER OF CONNECTIONS TO DATA SERVER VIA TUNNELS - FLOWCHART

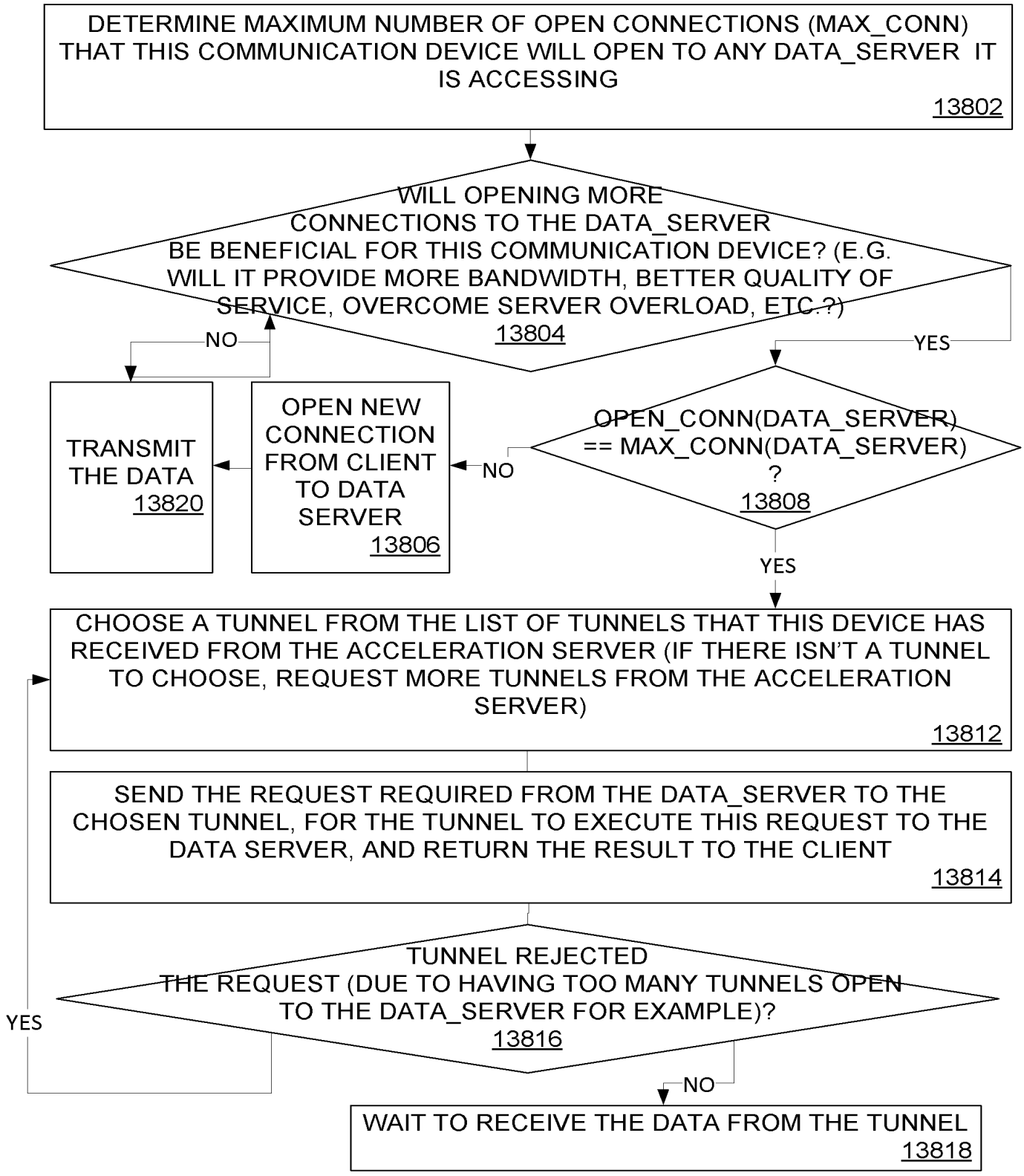


FIG. 142 – AGENT SELECTION – FEEDBACK TO SERVER ABOUT AGENTS - DIAGRAM

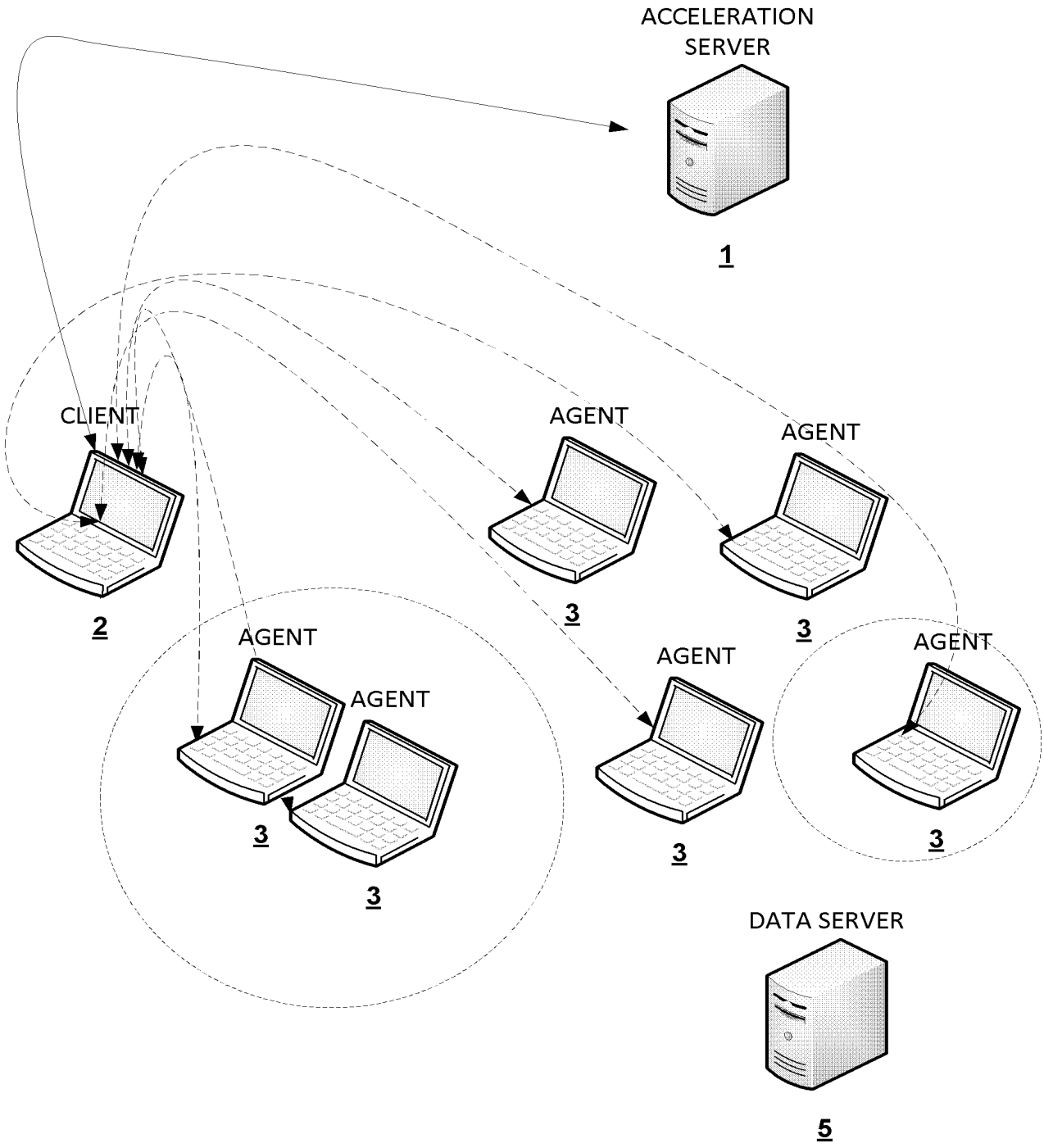


FIG. 144 – AGENT SELECTION – FEEDBACK TO SERVER ABOUT AGENTS – FLOWCHART FOR SERVER SIDE

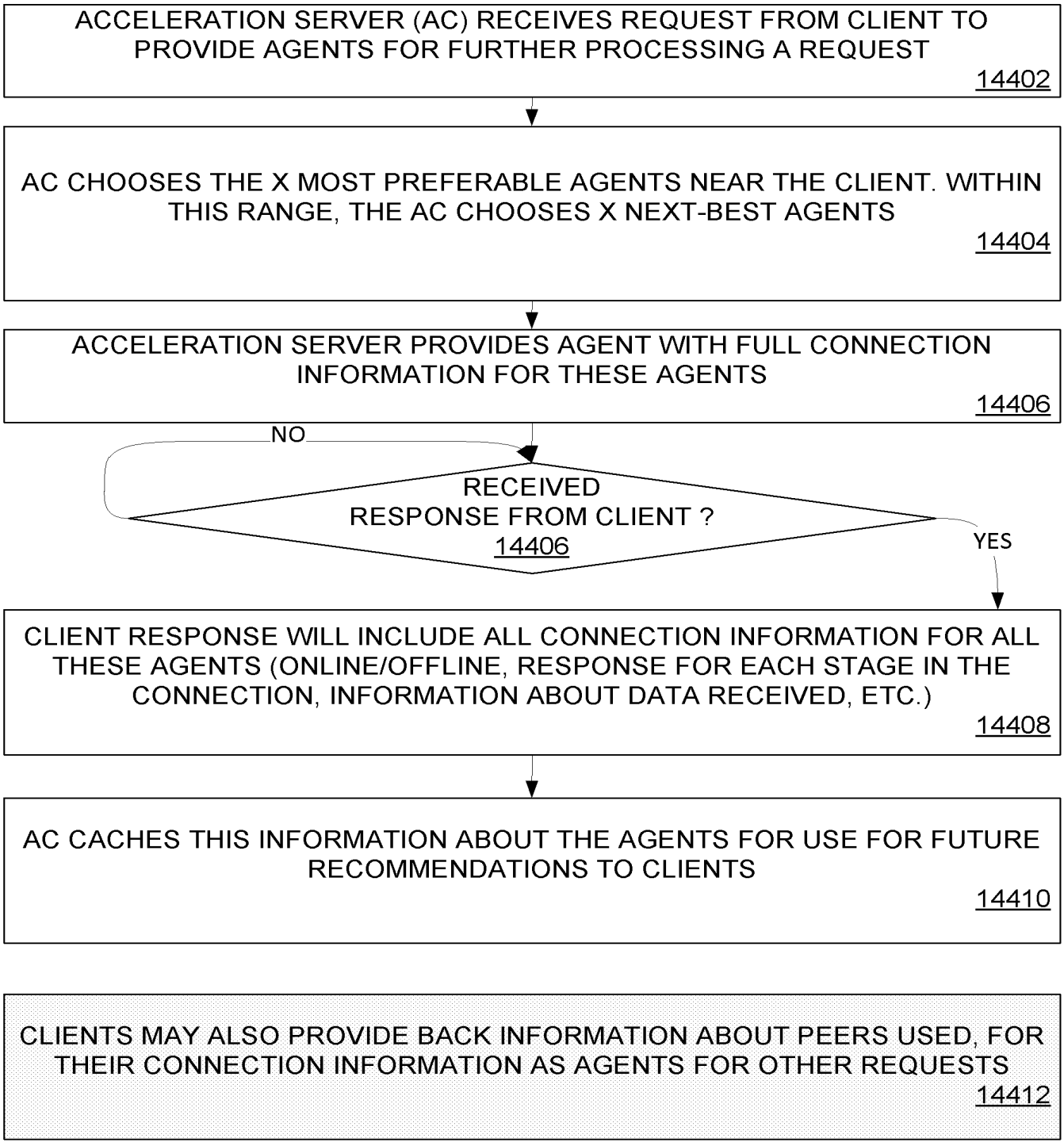


FIG. 146 – AGENT SELECTION – FEEDBACK TO SERVER ABOUT AGENTS – FLOWCHART FOR CLIENT SIDE

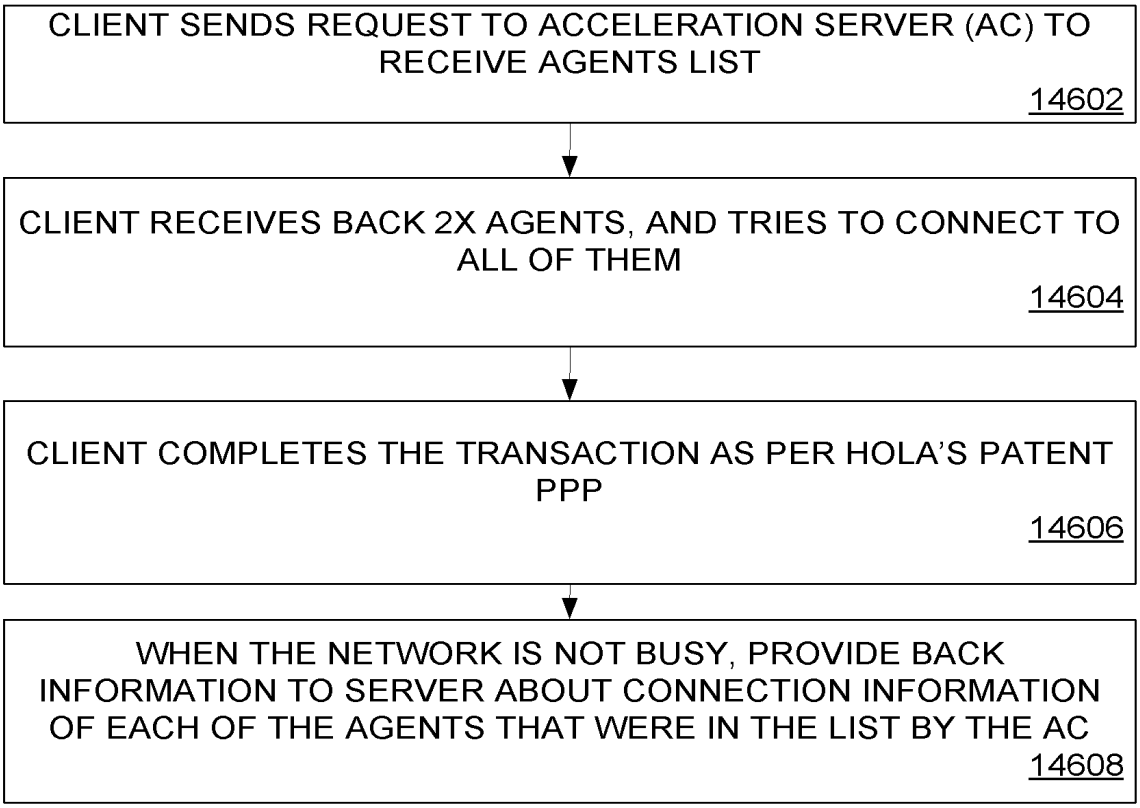


FIG. 148 – PEER SELECTION – FEEDBACK TO AGENT ABOUT PEERS - DIAGRAM

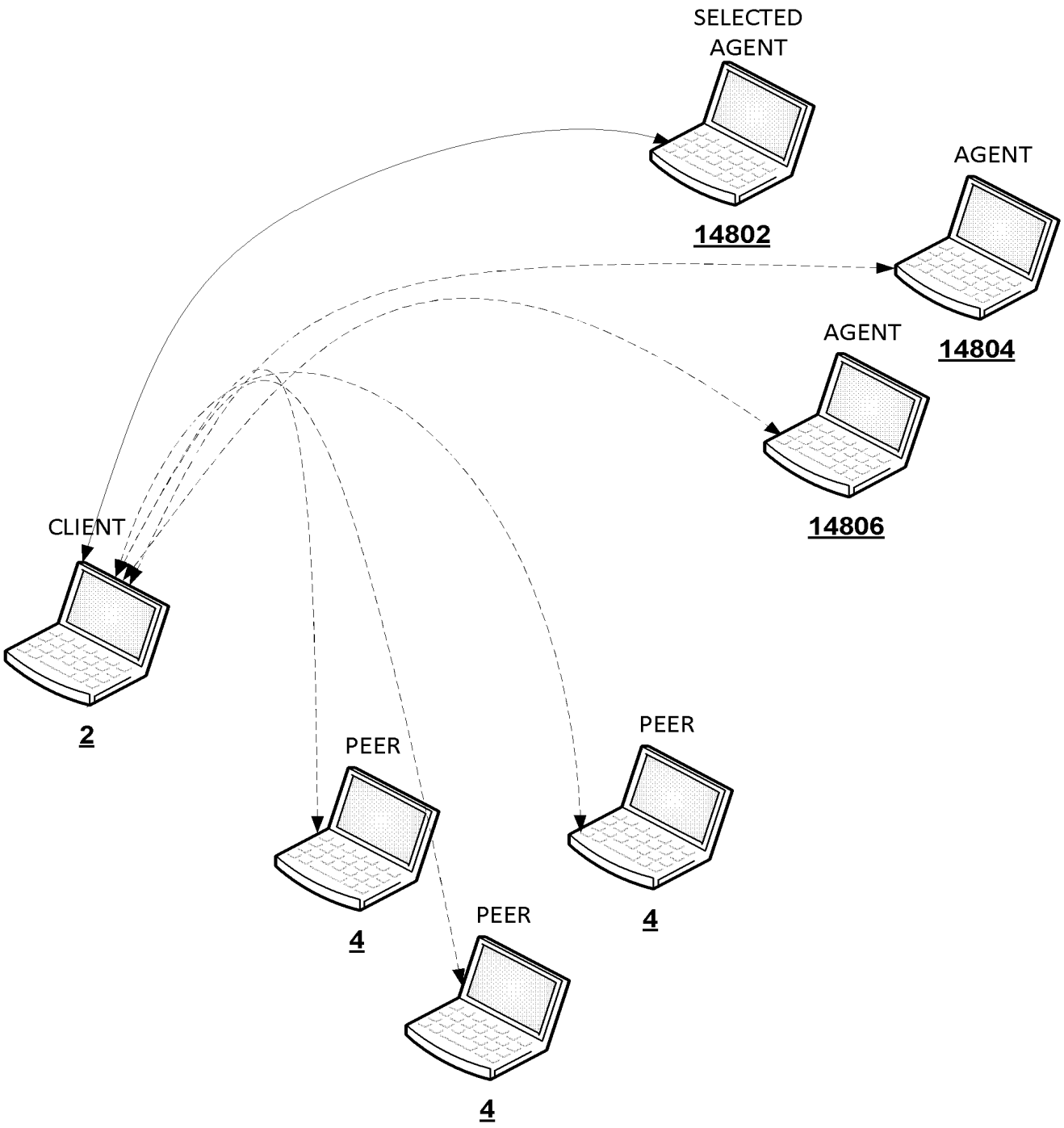


FIG. 152 – AGENT SELECTION – FEEDBACK TO AGENT ABOUT PEERS – FLOWCHART FOR CLIENT SIDE

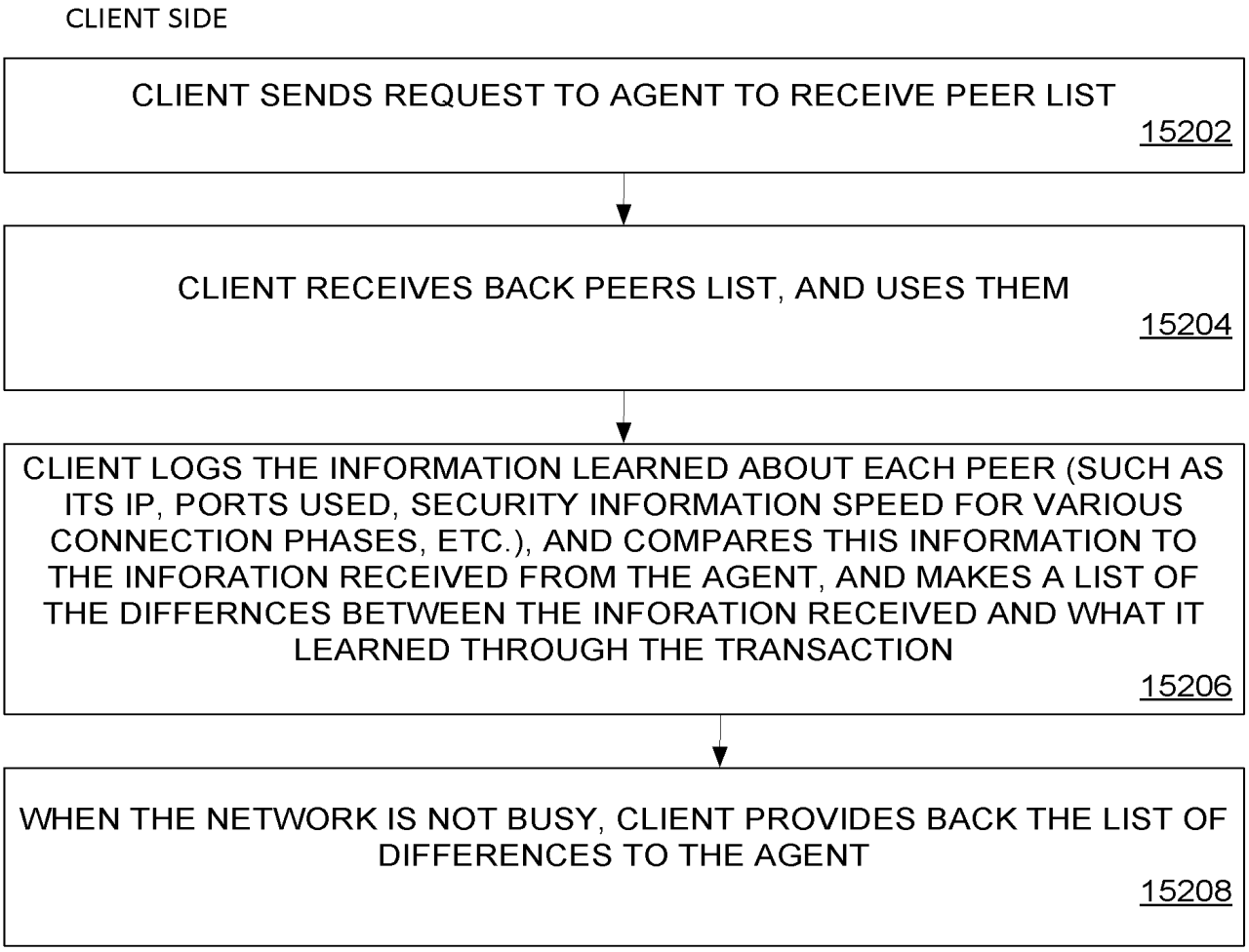


FIG. 154 – PEER SELECTION – FEEDBACK TO AGENT ABOUT PEERS - FLOWCHART

AGENT SIDE

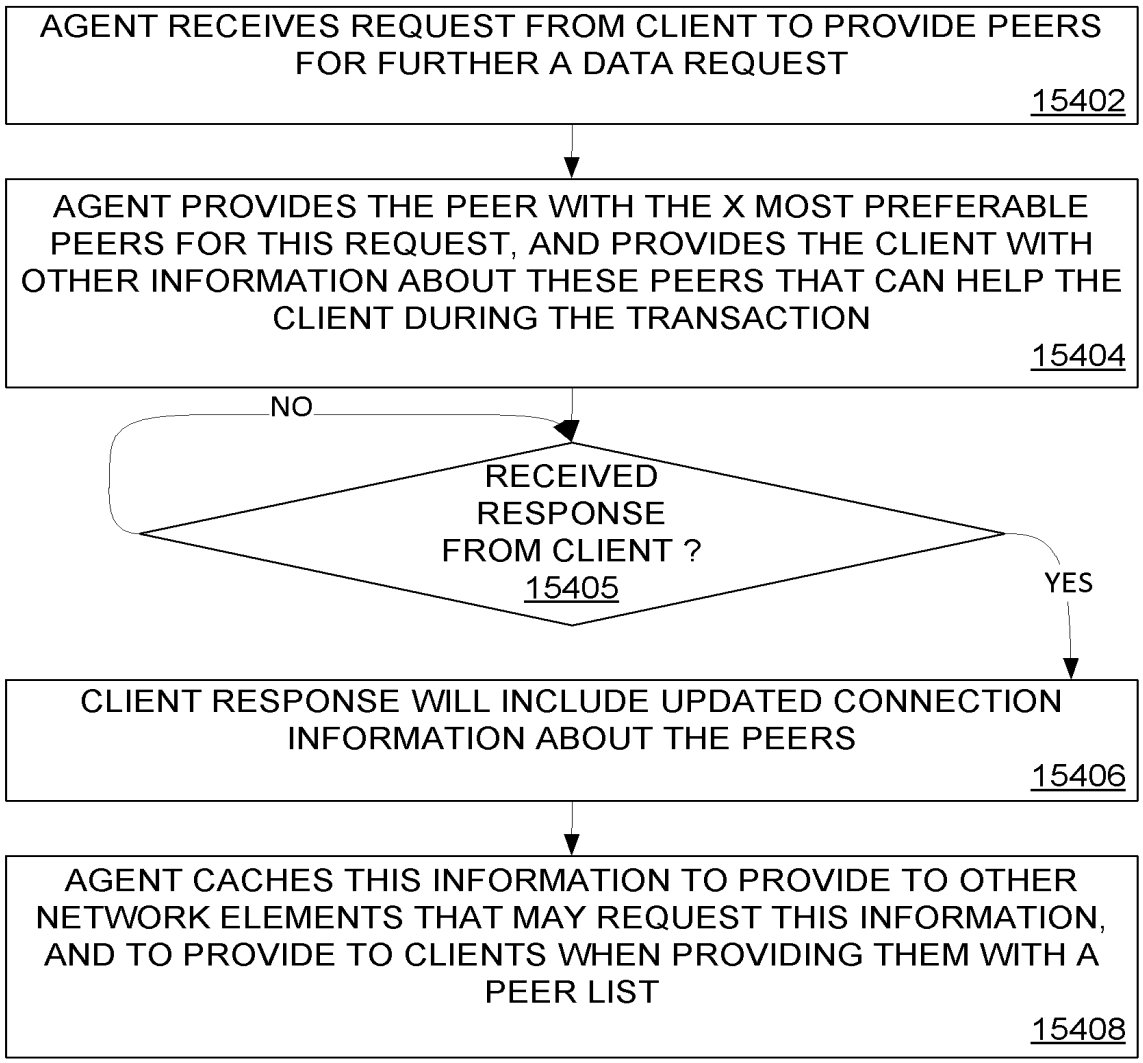
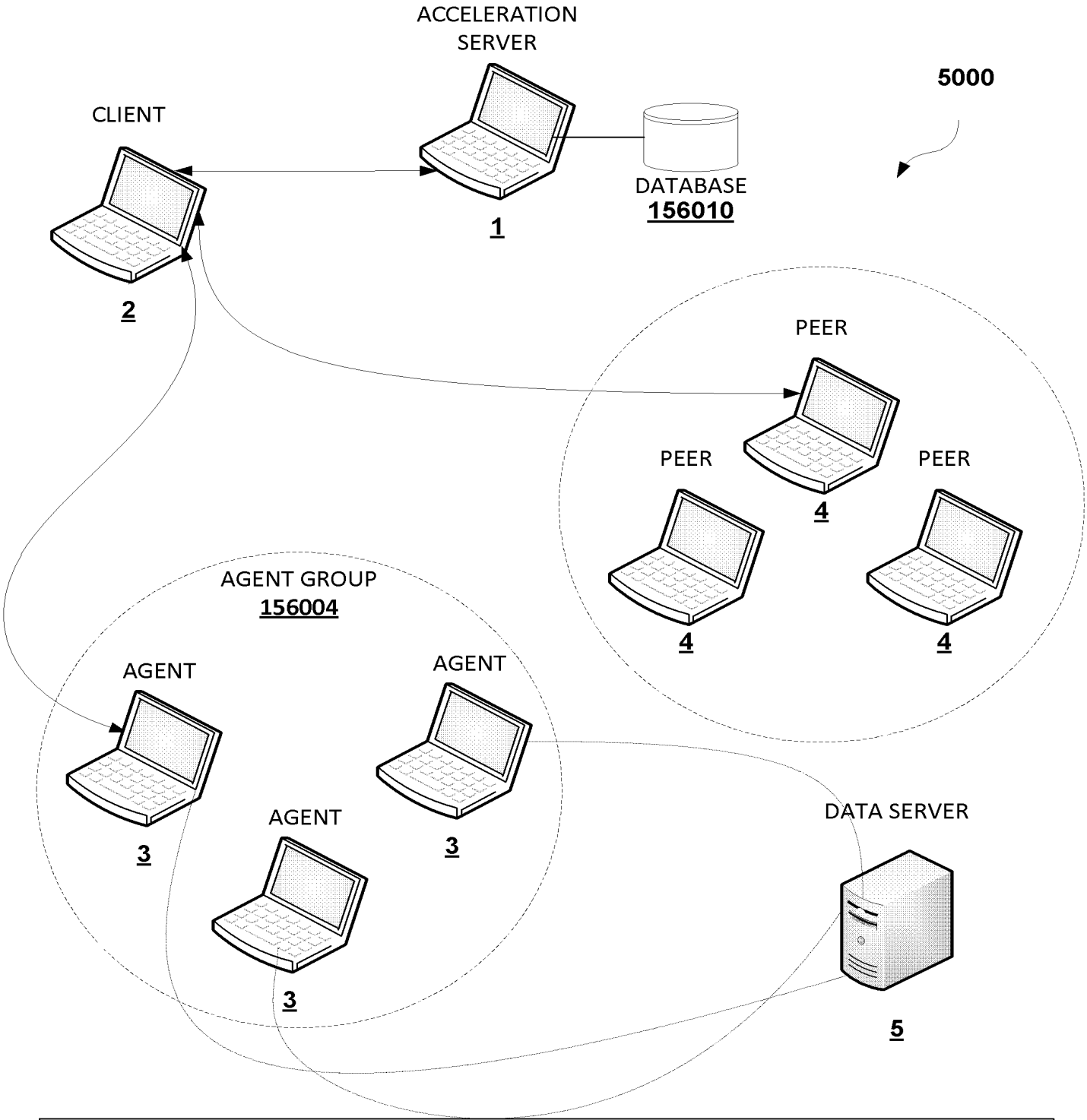


FIG. 156 – LOAD BALANCING OF AGENTS - DIAGRAM



NOTE: THE AGENT GROUP IS A GROUP OF AGENTS ASSIGNED TO A SPECIFIC DATA SERVER. AS THE VARIOUS AGENTS IN THE GROUP BECOME MORE CONGESTED, MORE NETWORK ELEMENTS ARE JOINED IN TO THAT GROUP BY THE ACCELERATION SERVER. AS AGENTS FREE UP, SOME ARE RELEASED BY THE ACCELERATION SERVER AND THE GROUP BECOMES SMALLER

156006

FIG. 158 – LOAD BALANCING OF AGENTS - FLOWCHART

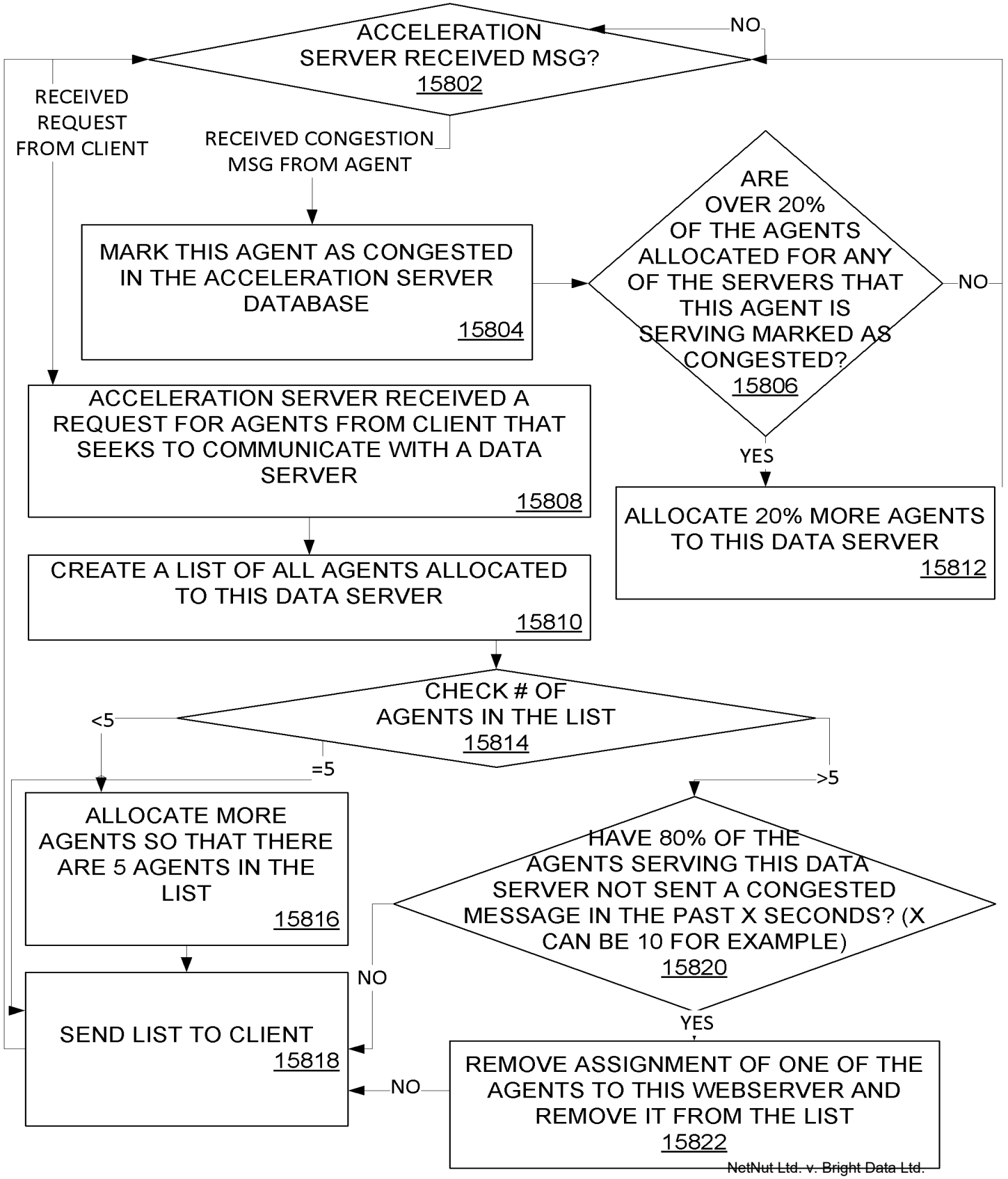


FIG. 160 – LOAD BALANCING OF AGENTS – ADDITIONAL METHODS OF LEARNING CONGESTION

BY CLIENT

IF AGENTS IN THE LIST DO NOT RESPOND TO THE CLIENT'S REQUESTS, CLIENT NOTIFIES SERVER THAT THESE ARE CONGESTED (ACCELERATION SERVER TREATS IT AS IF THE AGENT SENT A 'CONGESTED' SIGNAL)

16002

BY AGENT

IF AGENT'S RESOURCES (E.G. STORAGE, I/O, CPU, BANDWIDTH, ETC.) ARE BEING USED FOR THE BENEFIT OF THE NETWORK (I.E. NOT FOR OWN BENEFIT) OVER A CERTAIN THRESHHOLD (E.G. ABOVE 10% OF THE CPU PROCESSING), THEN AGENT SENDS A 'CONGESTED' SIGNAL TO THE ACCELERATION SERVER

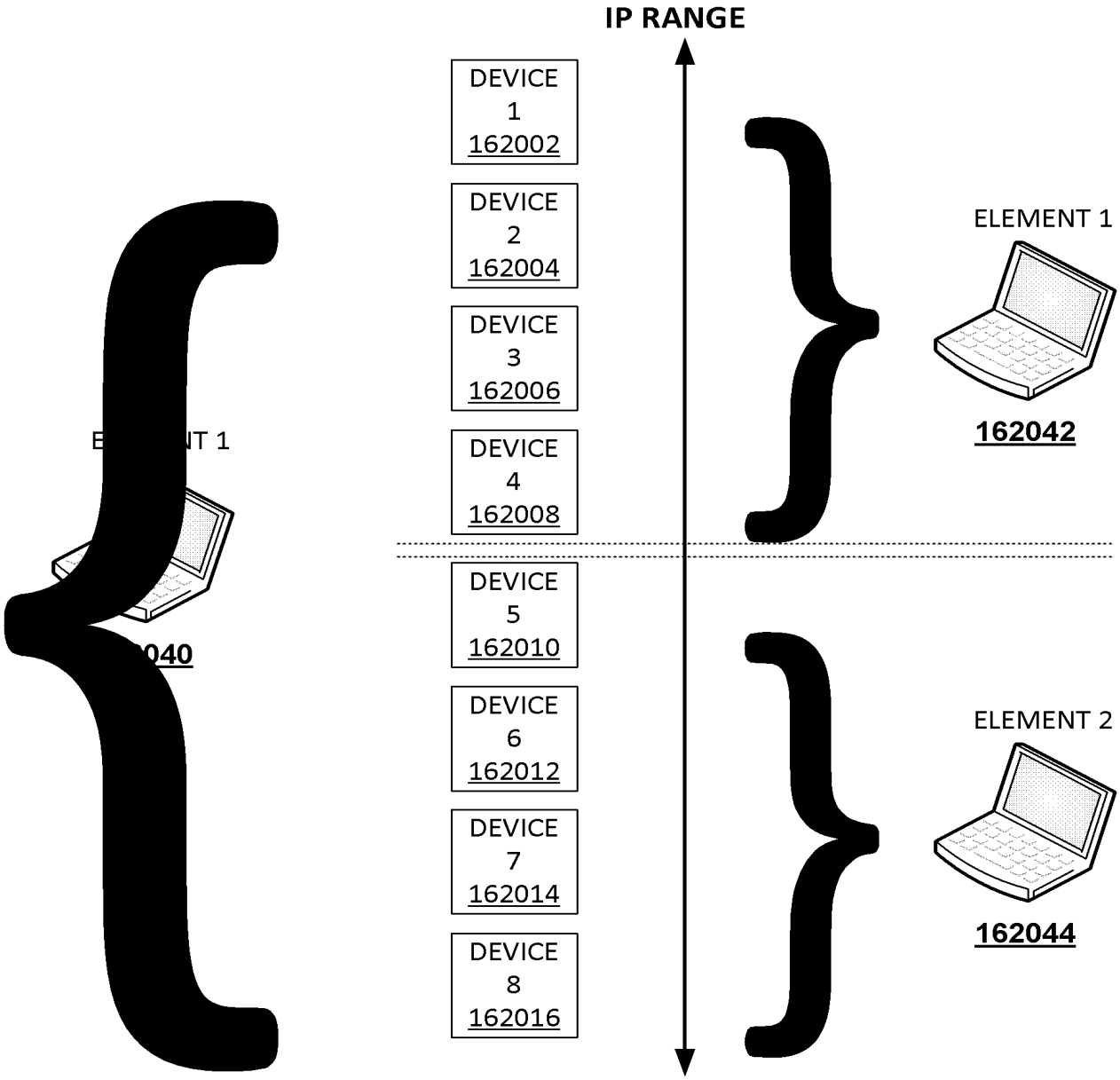
16004

BY ACCELERATION SERVER

ACCELERATION SERVER CAN PING THE AGENTS AND MARK THEM AS CONGESTED IF THEY DO NOT REPLY ABOVE A CERTAIN TIME (E.G. ABOVE 300MS)

16006

FIG. 162 – AGENT LOAD REDUCTION: REDUCING NUMBER OF IPS PER AGENT - DIAGRAM



NOTES:

1. AGENT 1 IS BEING ACCESSED BY A RANGE OF IPS [IP(1) TO IP(8)] AND IS OVERLOADED
2. THERE ARE OTHER AGENTS THAT ARE FREE AND CAN SHARE THE LOAD [AGENT 2]
3. AGENT 1 DIVIDES THE IP RANGE IN TO 2 EQUAL RANGES [IP(1)...IP(4) AND IP(5)...IP(8)] AND ASSIGNS THE FIRST RANGE TO HIMSELF AND THE SECOND TO AGENT 2
4. ALL SECOND RANGE IPS WILL BE REFERRED TO AGENT2 BY THE AGENTS AND ACCELERATION SERVER, AND WILL BE REDIRECTED TO AGENT2 BY AGENT1 AND AGENT1 WILL NOTIFY THE REFERRER TO REDIRECT TO AGENT2 NEXT TIME

FIG. 164 – AGENT LOAD REDUCTION: REDUCING NUMBER OF IPS PER AGENT - FLOWCHART

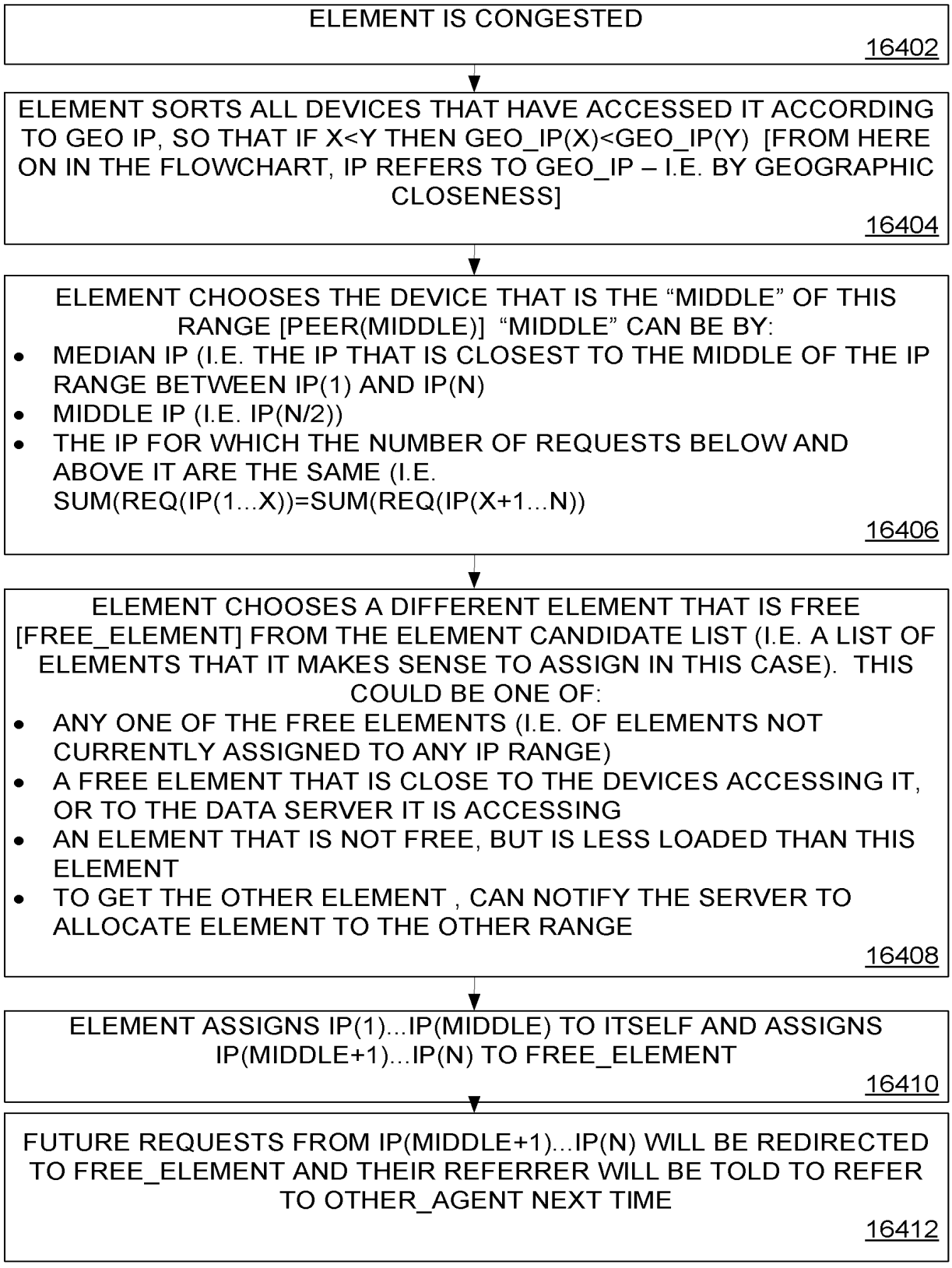


FIG. 166 – AGENT LOAD REDUCTION: REDUCING NUMBER OF IPS PER AGENT - IMPROVEMENTS

IMPROVEMENT 1: FINDING FREE ELEMENTS –- ALSO POSSIBLE FOR THE SERVER TO PERIODICALLY LOOK FOR "UNDERLOADED" ELEMENTS (FOR EXAMPLE BY LOOKING FOR ELEMENTS THAT HAVE NOT BEEN OVERLOADED IN A WHILE), AND TO MERGE THEIR RANGES TO FREE SOME OF THEM UP FOR ASSIGNING TO OVERLOADED ELEMENTS

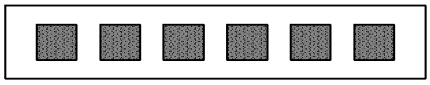
16602

IMPROVEMENT 2: INSTEAD OF LOGGING ALL THE IPS THAT ARE ACCESSING AN ELEMENT, THE ELEMENT CAN LOG ONLY THE X RECENT REQUESTS (X COULD BE 1,000 FOR EXAMPLE), OR KEEP A LOGARITHMIC RANDOM LIST OF X FROM THE PAST 24 HOURS, X/2 FROM PAST WEEK, X/4 FROM PAST MONTH, ETC.)

16604

FIG. 170 – MICRO CHUNKS – PRIOR ART – DIAGRAM

17001 FILE_TO_LOAD: THE FILE TO RETREIVE, COMPRISED OF 6 EQUALLY SIZED CHUNKS:



THE CLIENT LOADS THE FILE'S 6 CHUNKS FROM 3 SOURCES, TAKING MORE CHUNKS FROM THE FASTER SOURCES (3 FROM SOURCE_1, 1 FROM SOURCE_2, AND 2 FROM SOURCE_3):

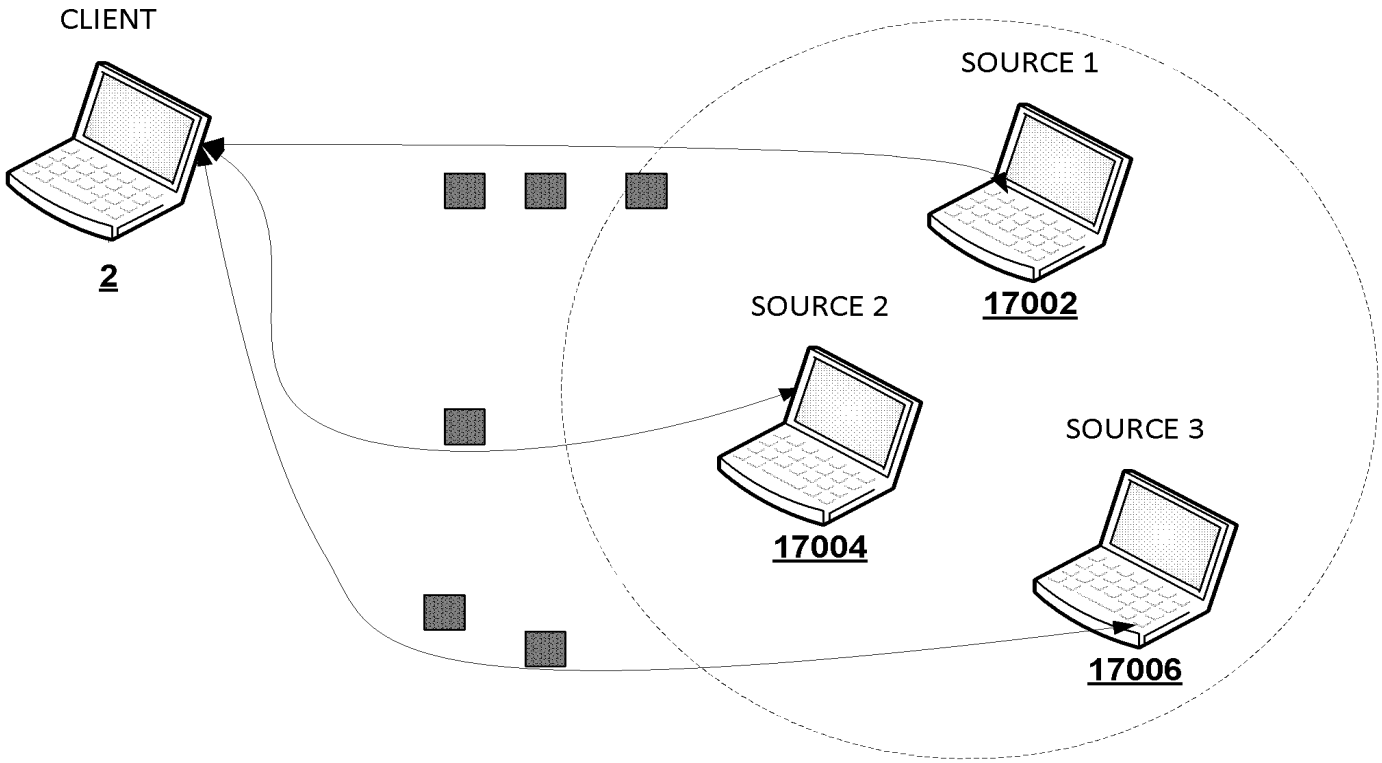


FIG. 172 – MICRO CHUNKS – DIAGRAM

THE ONE CHUNK FROM SOURCE_2 IS SPLIT IN TO TWO SMALLER CHUNKS, IN ORDER NOT TO DELAY THE FULL LOAD:

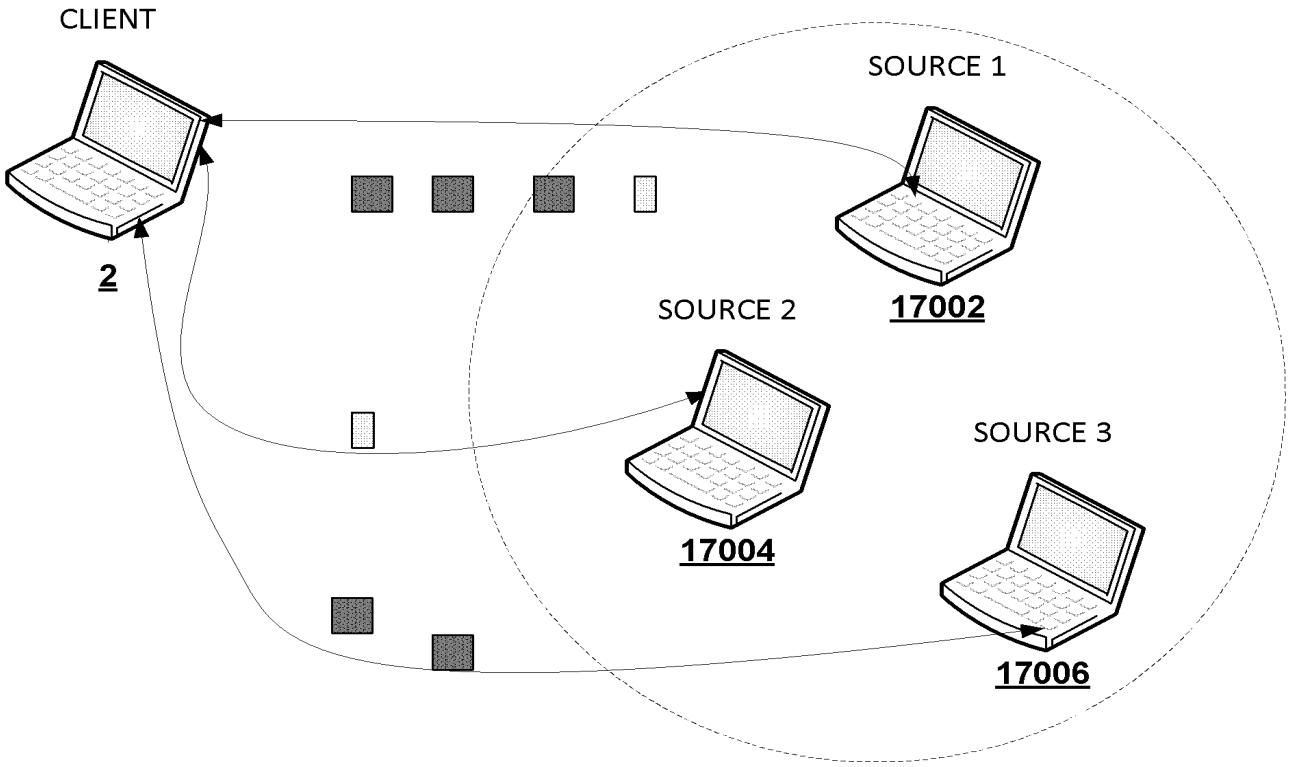


FIG. 174 – MICRO CHUNKS – FLOWCHART

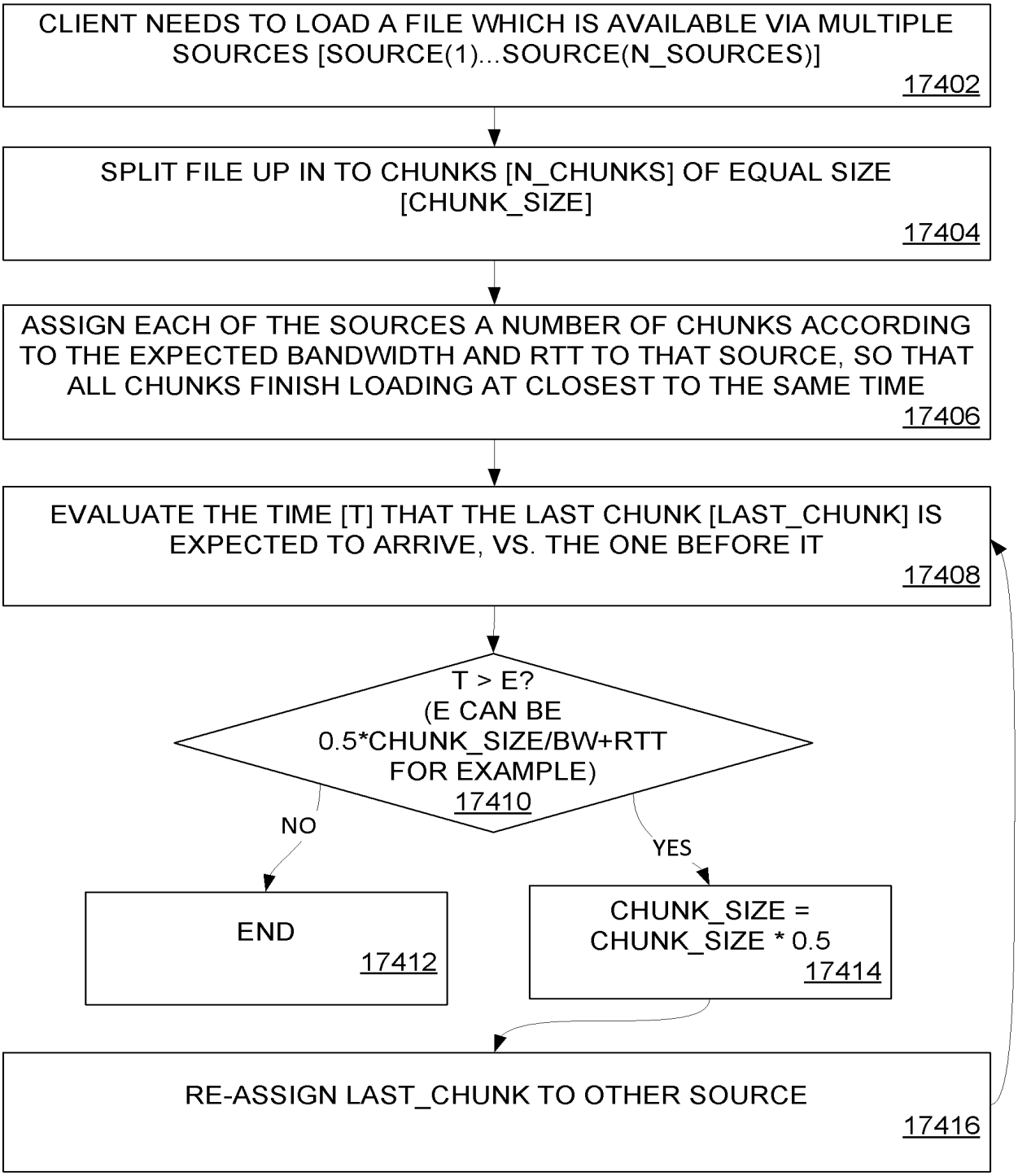


FIG. 200 – LOWERING AVERAGE CONNECTION TIME IN MULTI-SOURCE SCENARIO - DIAGRAM

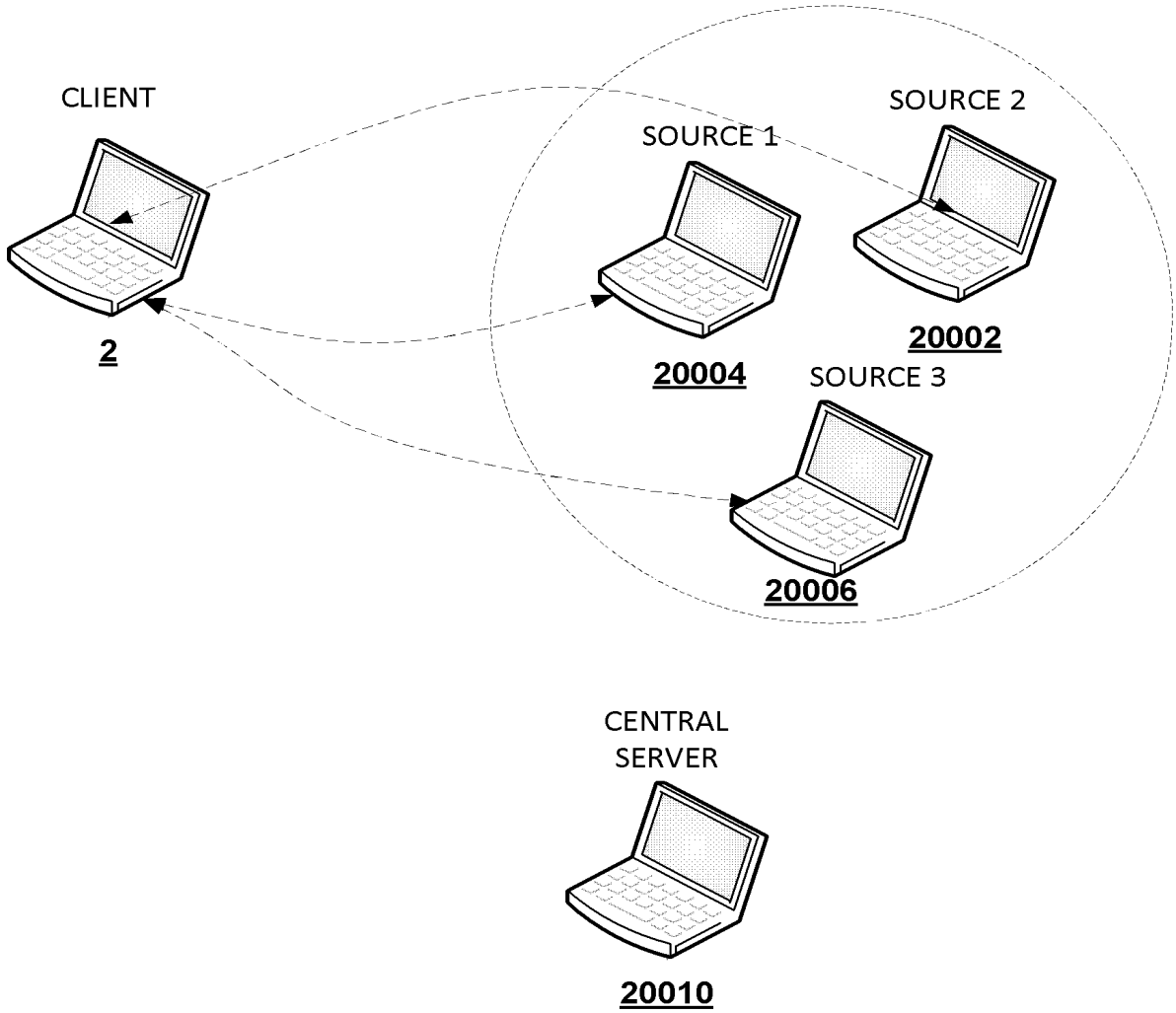
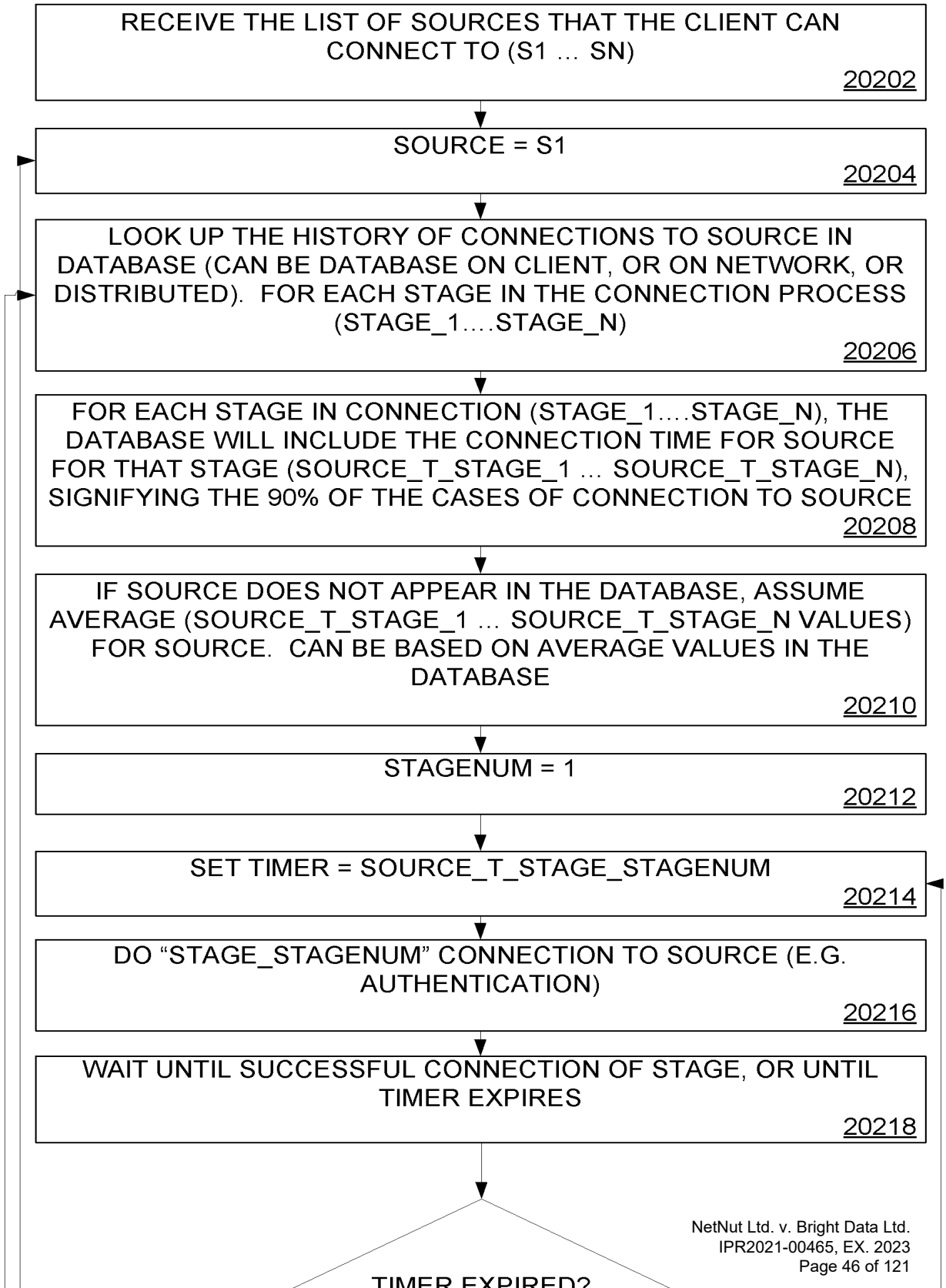


FIG. 202 – LOWERING AVERAGE CONNECTION TIME IN MULTI-SOURCE SCENARIO - FLOWCHART



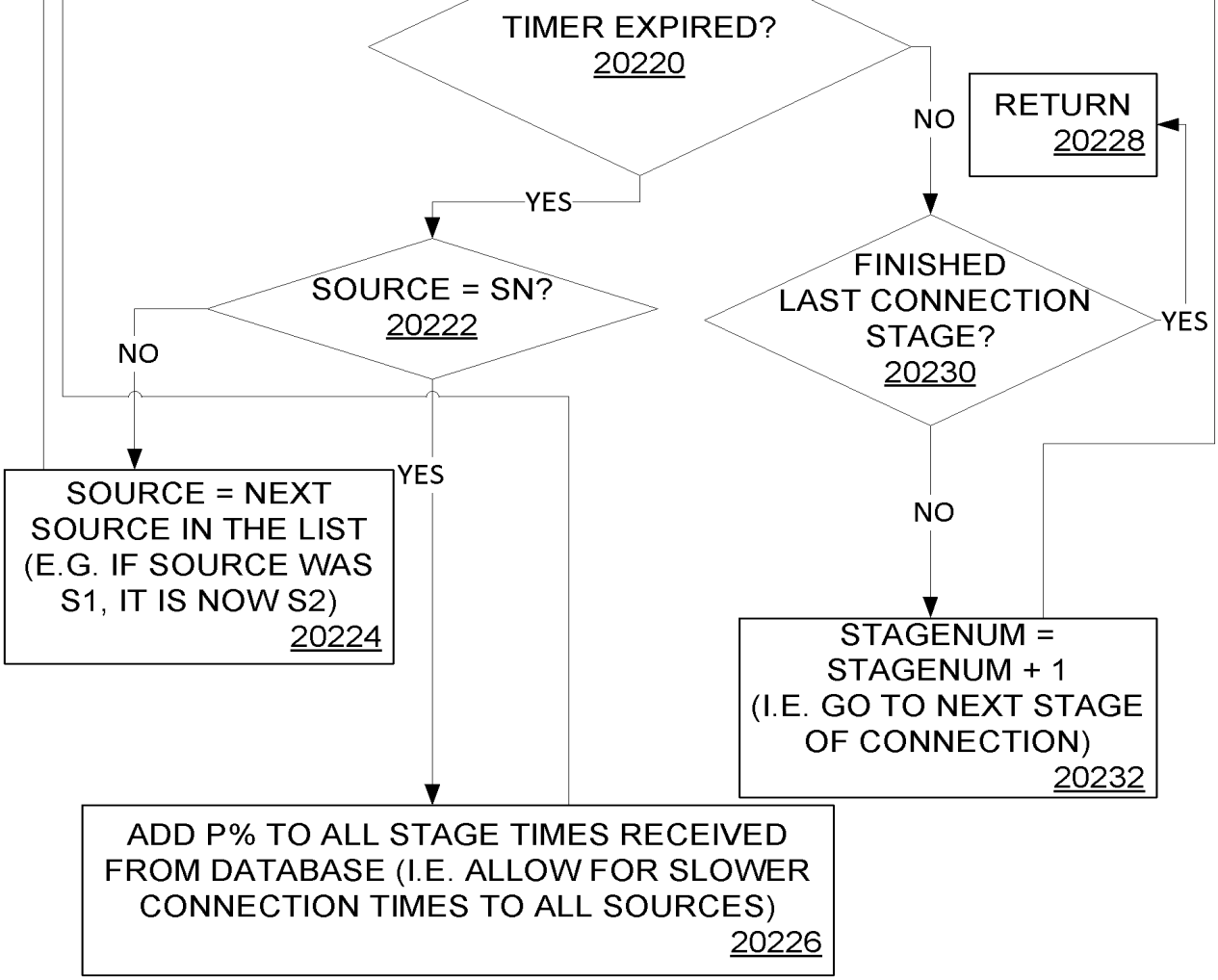
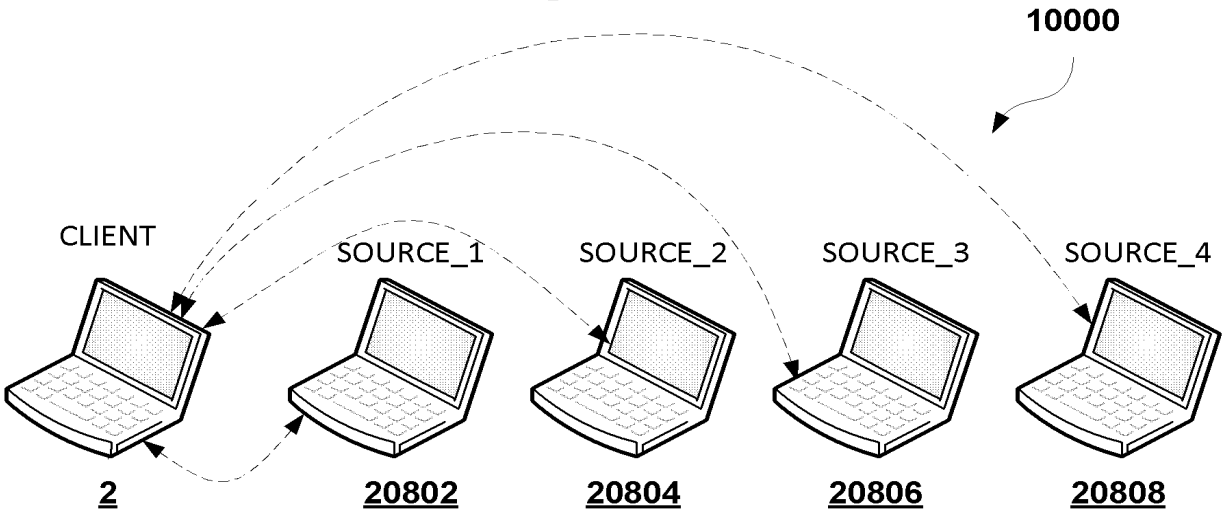

TIMER EXPIRED?
20220
NO
RETURN
20228
YES
SOURCE = SN?
20222
NO
YES
FINISHED LAST CONNECTION STAGE?
20230
YES
NO
SOURCE = NEXT SOURCE IN THE LIST (E.G. IF SOURCE WAS S1, IT IS NOW S2)
20224
STAGENUM = STAGENUM + 1 (I.E. GO TO NEXT STAGE OF CONNECTION)
20232
ADD P% TO ALL STAGE TIMES RECEIVED FROM DATABASE (I.E. ALLOW FOR SLOWER CONNECTION TIMES TO ALL SOURCES)
20226

FIG. 208 – HEURISTICS FOR FINDING BW AND RTT TO NETWORK ELEMENTS – EXAMPLE - DIAGRAM



IP	IP_CLIENT	<	IP1	<	IP2	<	IP3	< IP4
BW			100 MBIT/S		???		???	13 MBIT/S
RTT			30 MS		???		???	95 MS

CLIENT DATABASE

20810

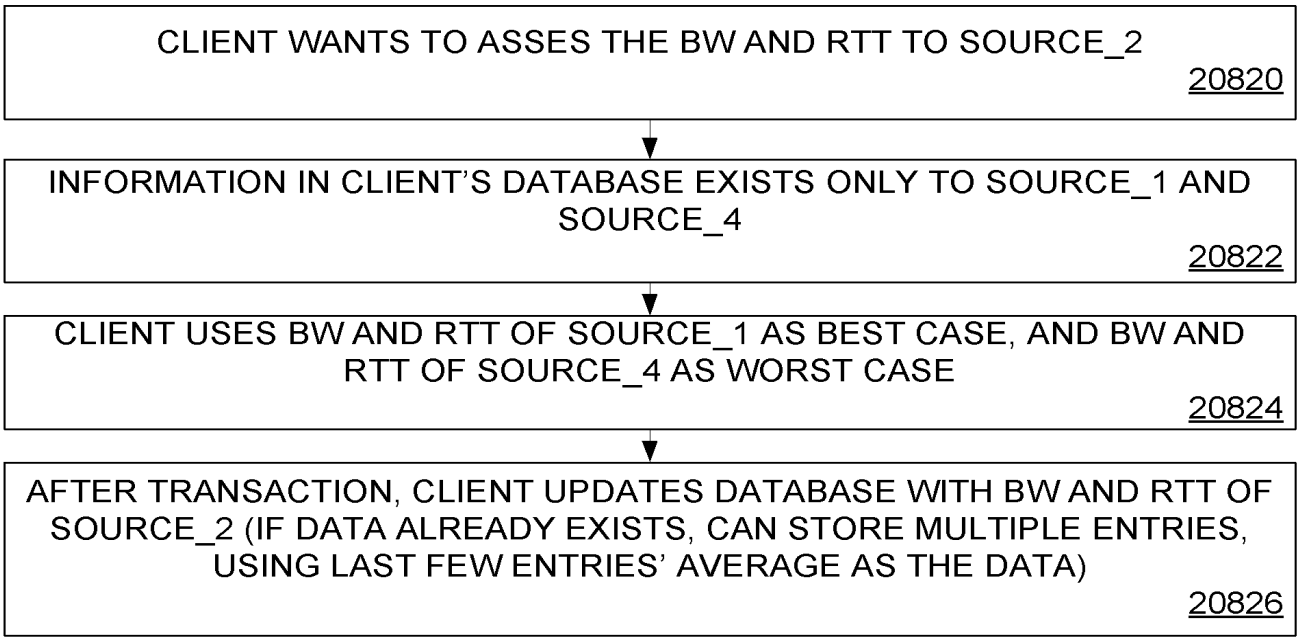


FIG. 210 – HEURISTICS FOR FINDING BW AND RTT TO NETWORK ELEMENTS - FLOWCHART

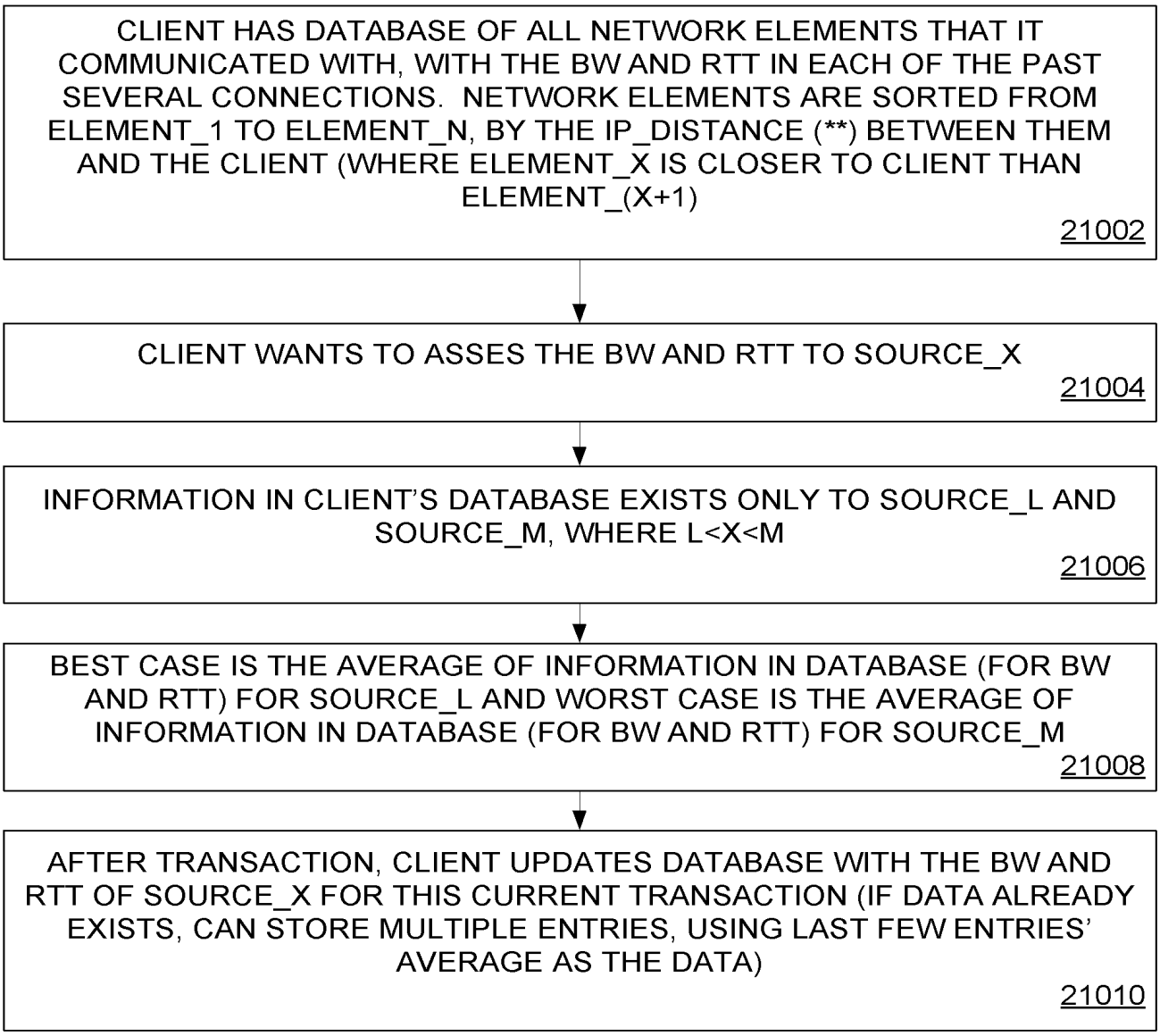


FIG. 212 – HEURISTICS FOR FINDING BW AND RTT TO NETWORK ELEMENTS – MEMORY SAVING NOTATION

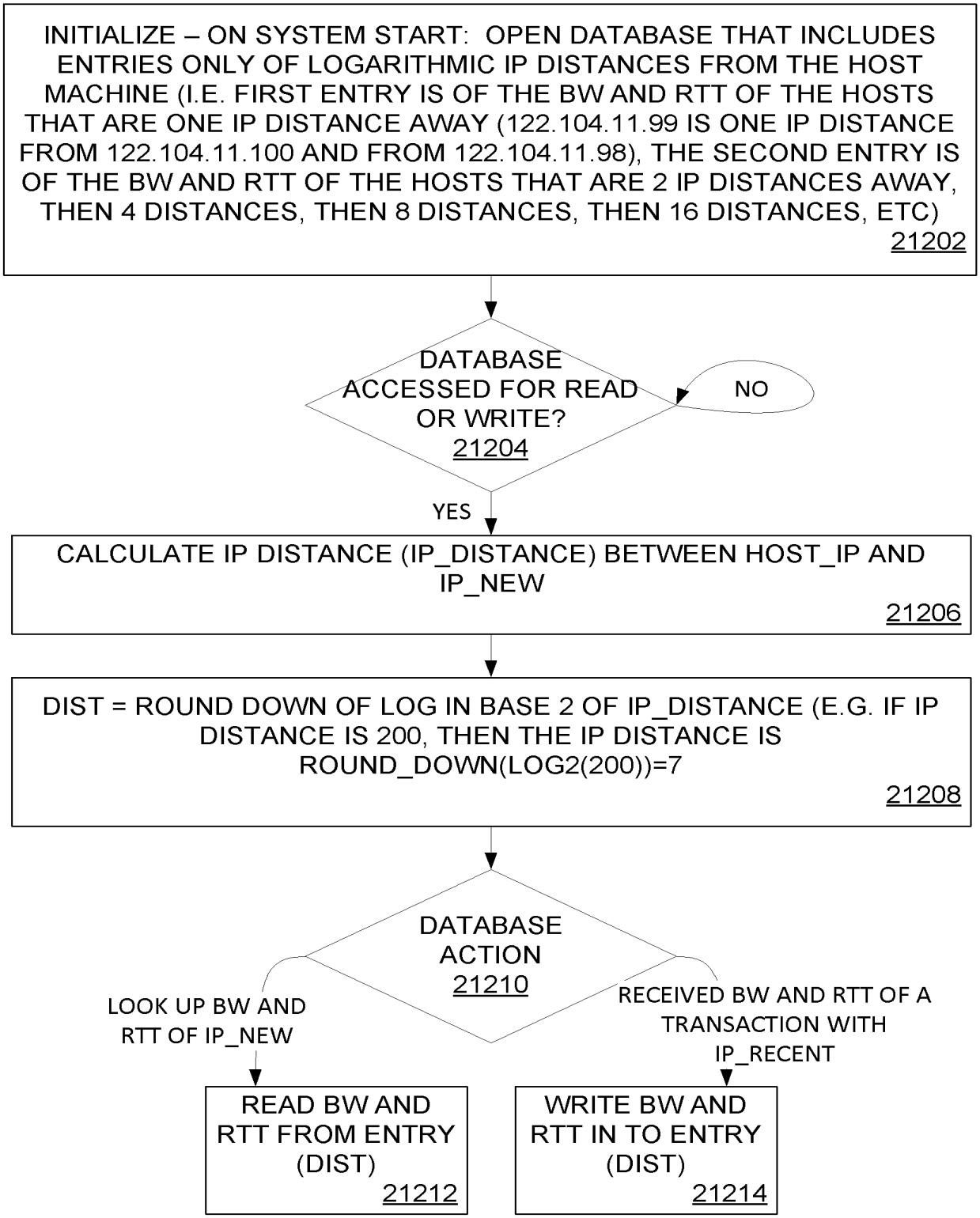


FIG. 214 – HEURISTICS FOR FINDING BW AND RTT TO NETWORK ELEMENTS – FASTER LEARNING NETWORK

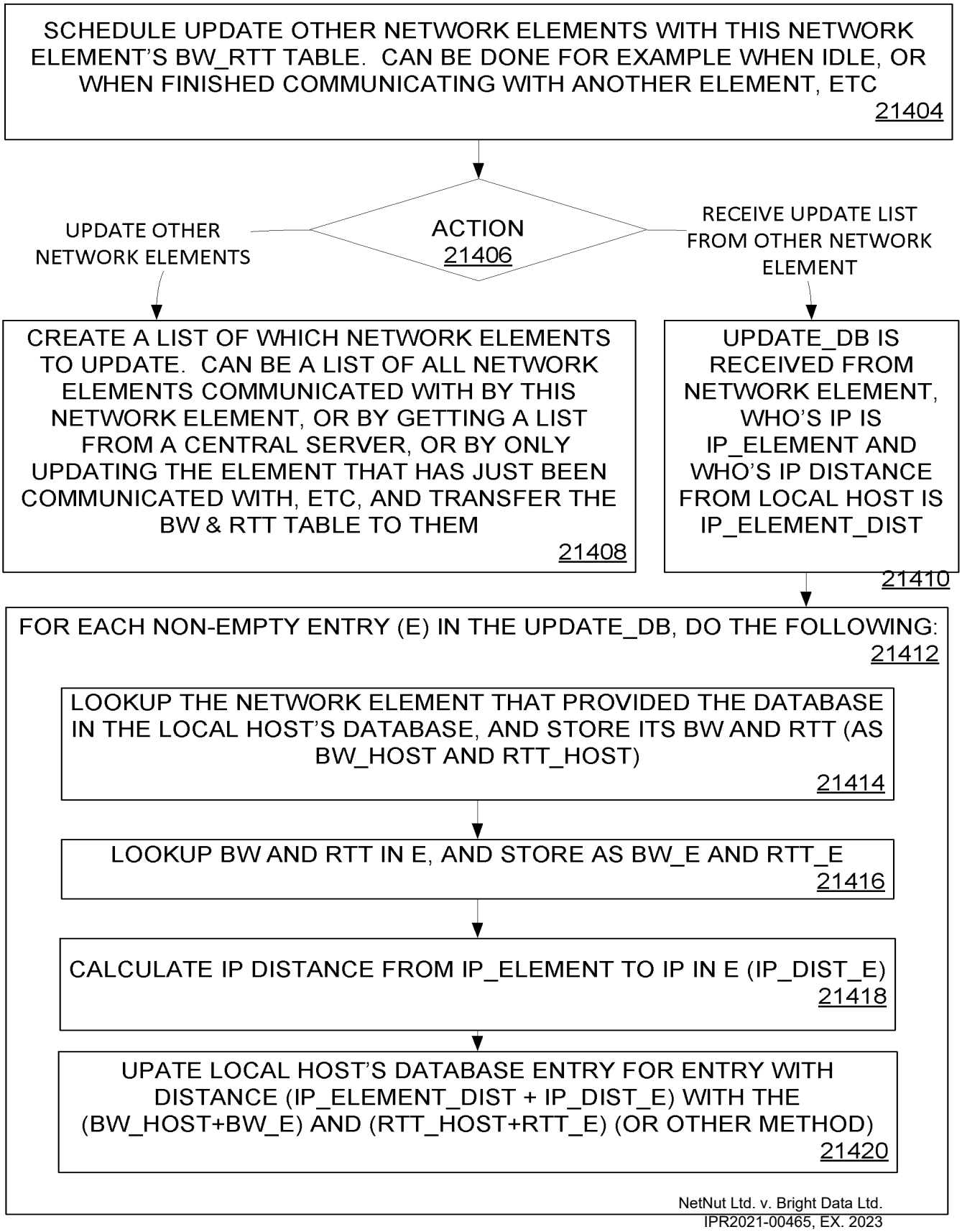


FIG. 216 – GEO IP. CREATING CONTIGUOUS RANGE, OPTIMAL FOR SPEED

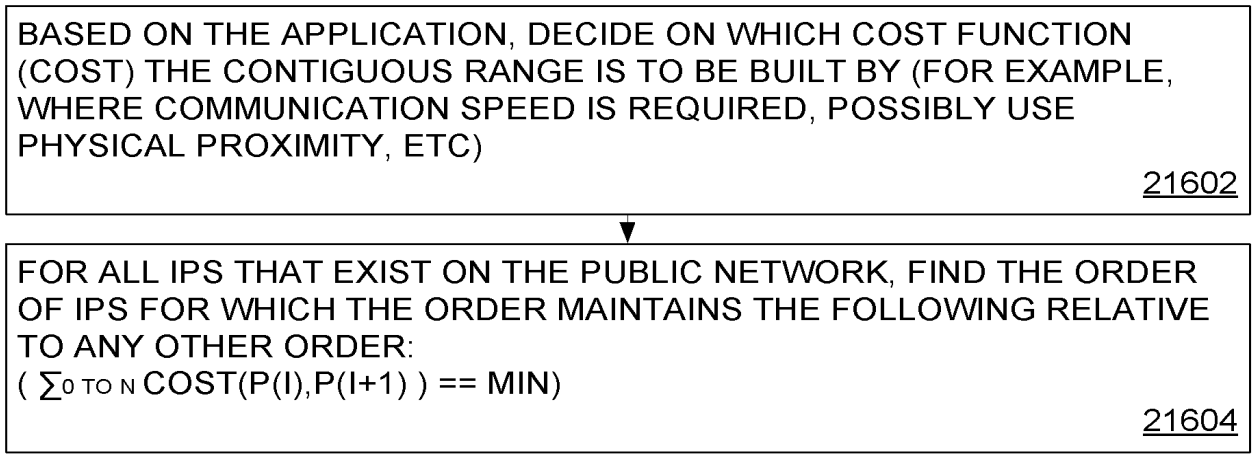


FIG. 218 – FINDING OUT WHICH ELEMENTS IN A NETWORKED SYSTEM ARE ONLINE - DIAGRAM

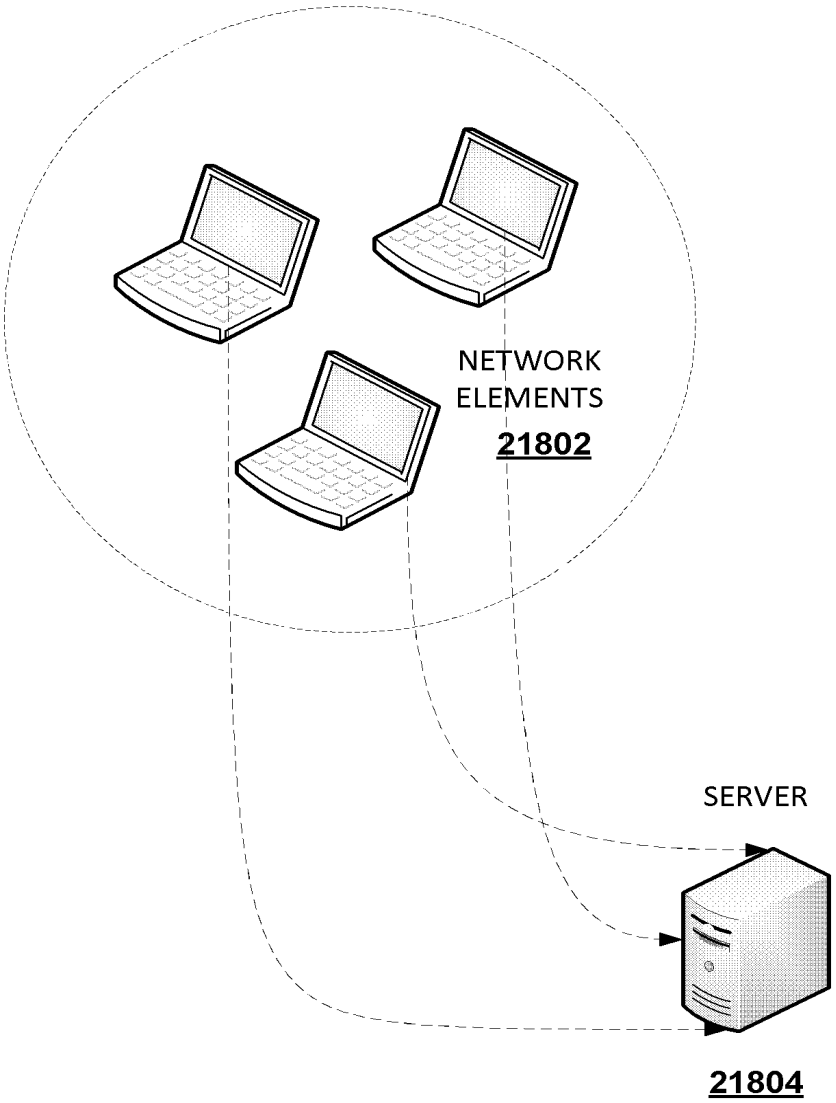


FIG. 220 – FINDING OUT WHICH ELEMENTS IN A NETWORKED SYSTEM ARE ONLINE – NETWORK ELEMENT FLOWCHART

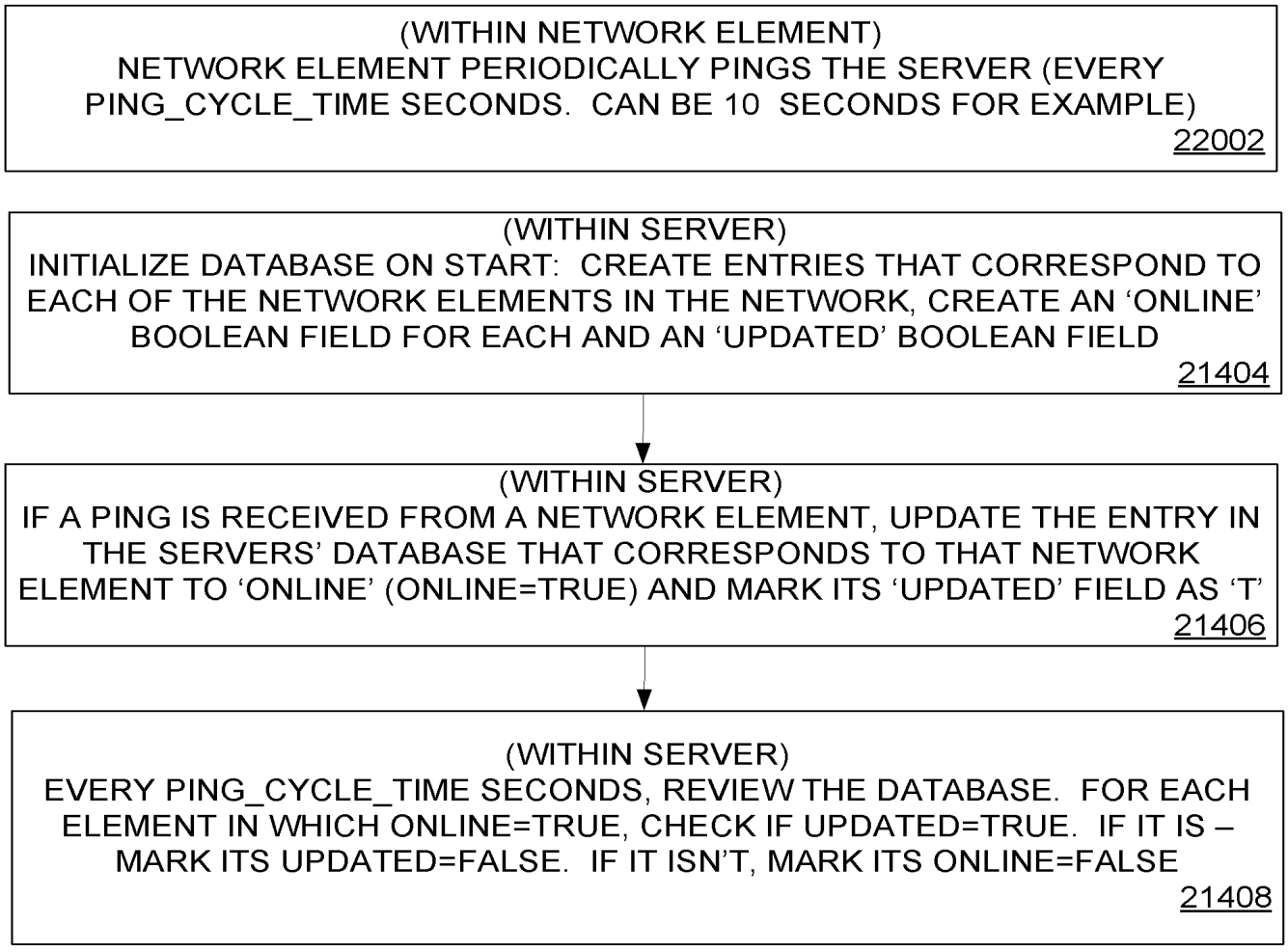


FIG. 224 – DYNAMIC QUEUES FOR SENDING DATA – PRIOR ART

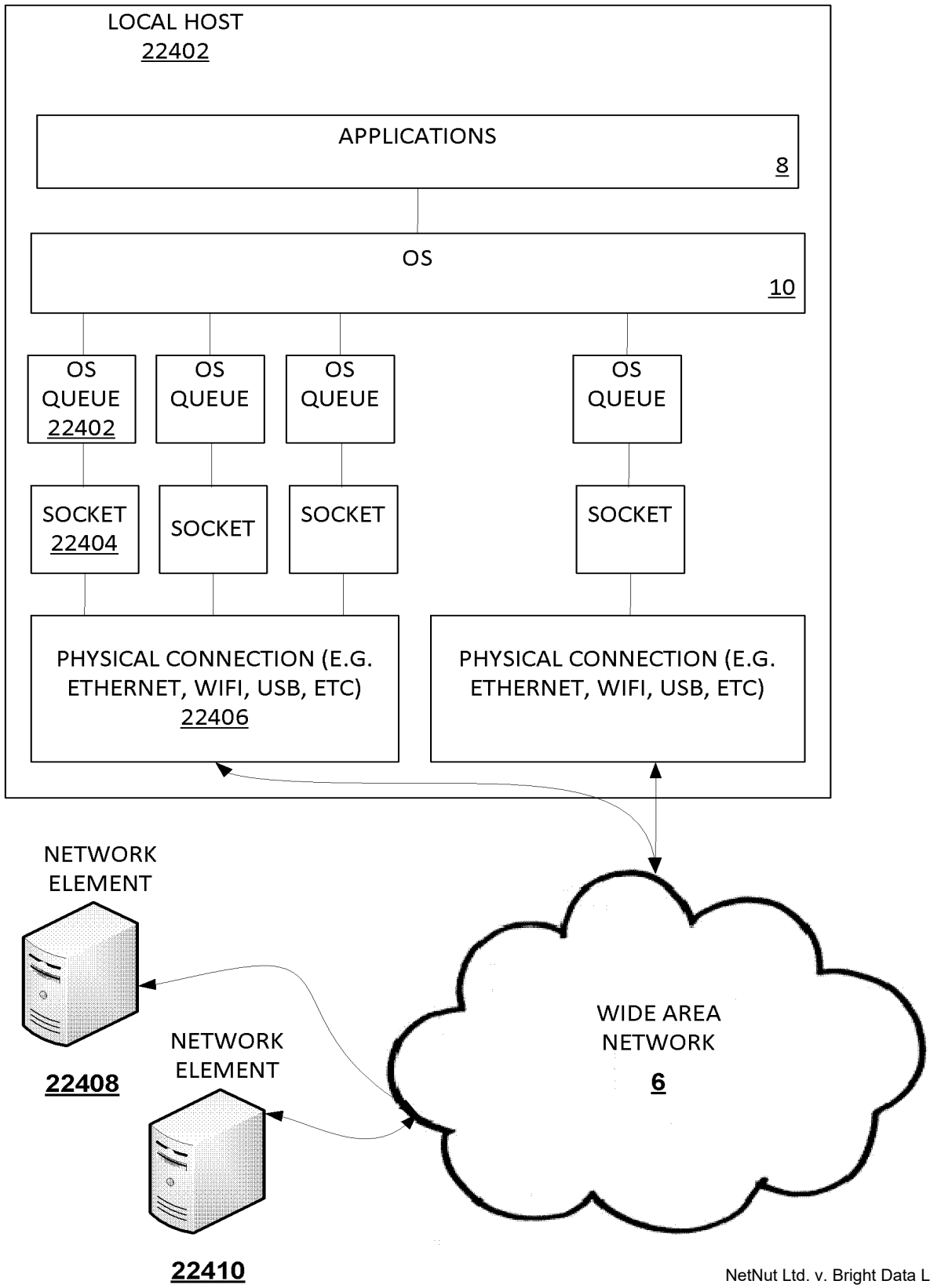


FIG. 226 – DYNAMIC QUEUES FOR SENDING DATA – DIAGRAM

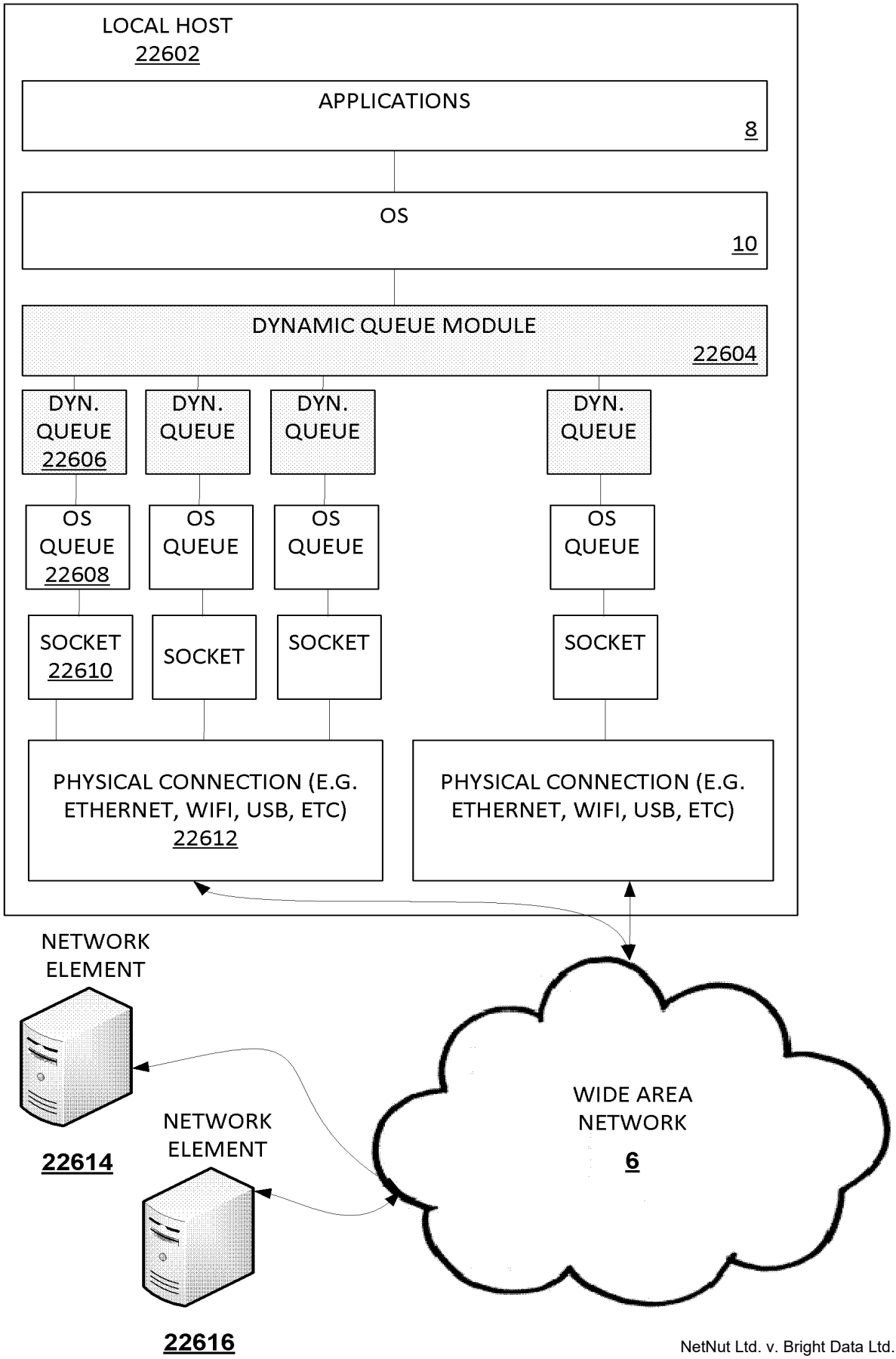


FIG. 228 – DYNAMIC QUEUES FOR SENDING DATA – DIAGRAM

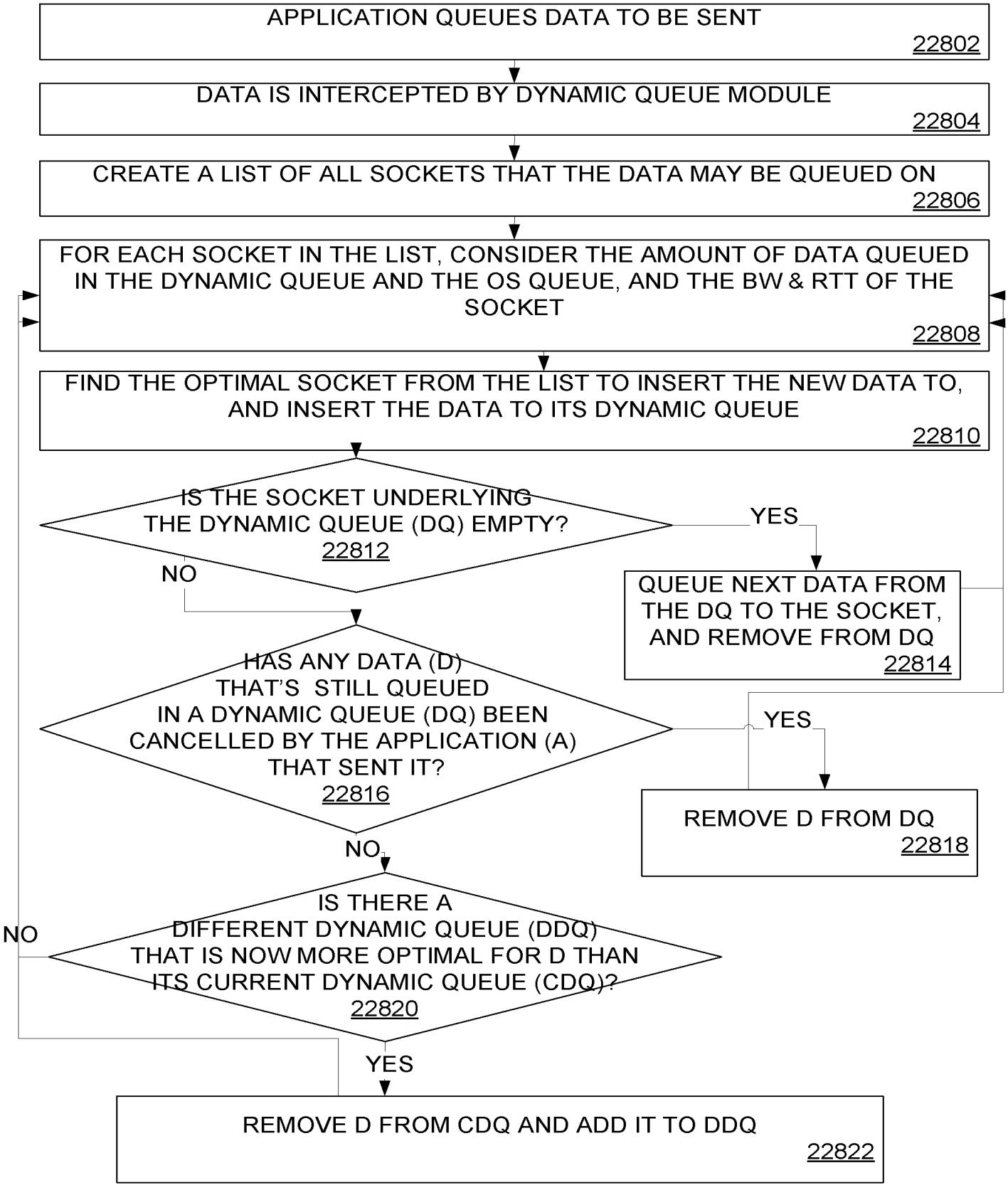


FIG. 232 – VIRTUAL APPLICATION GATEWAY – PRIOR ART

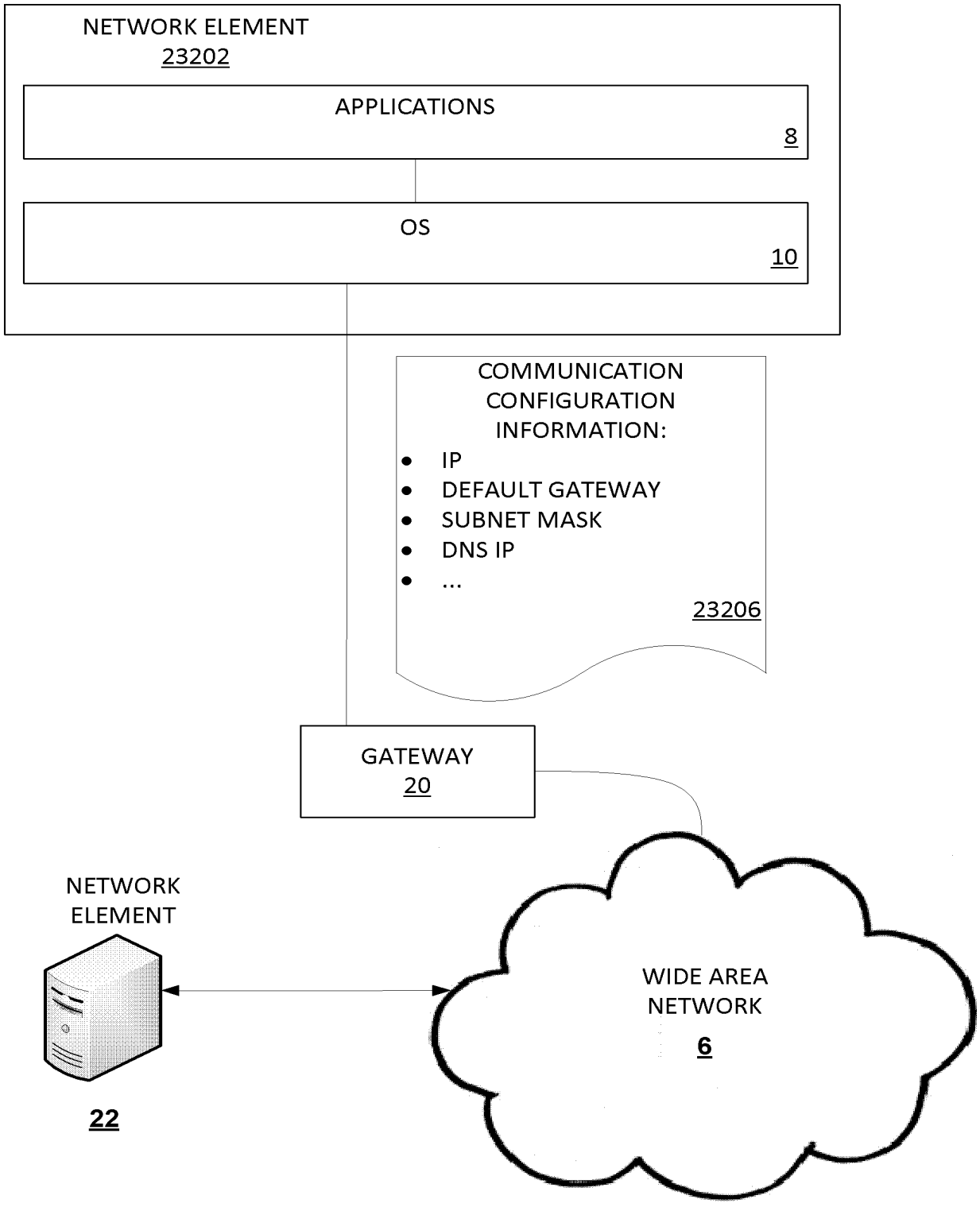


FIG. 234 – VIRTUAL APPLICATION GATEWAY – DIAGRAM

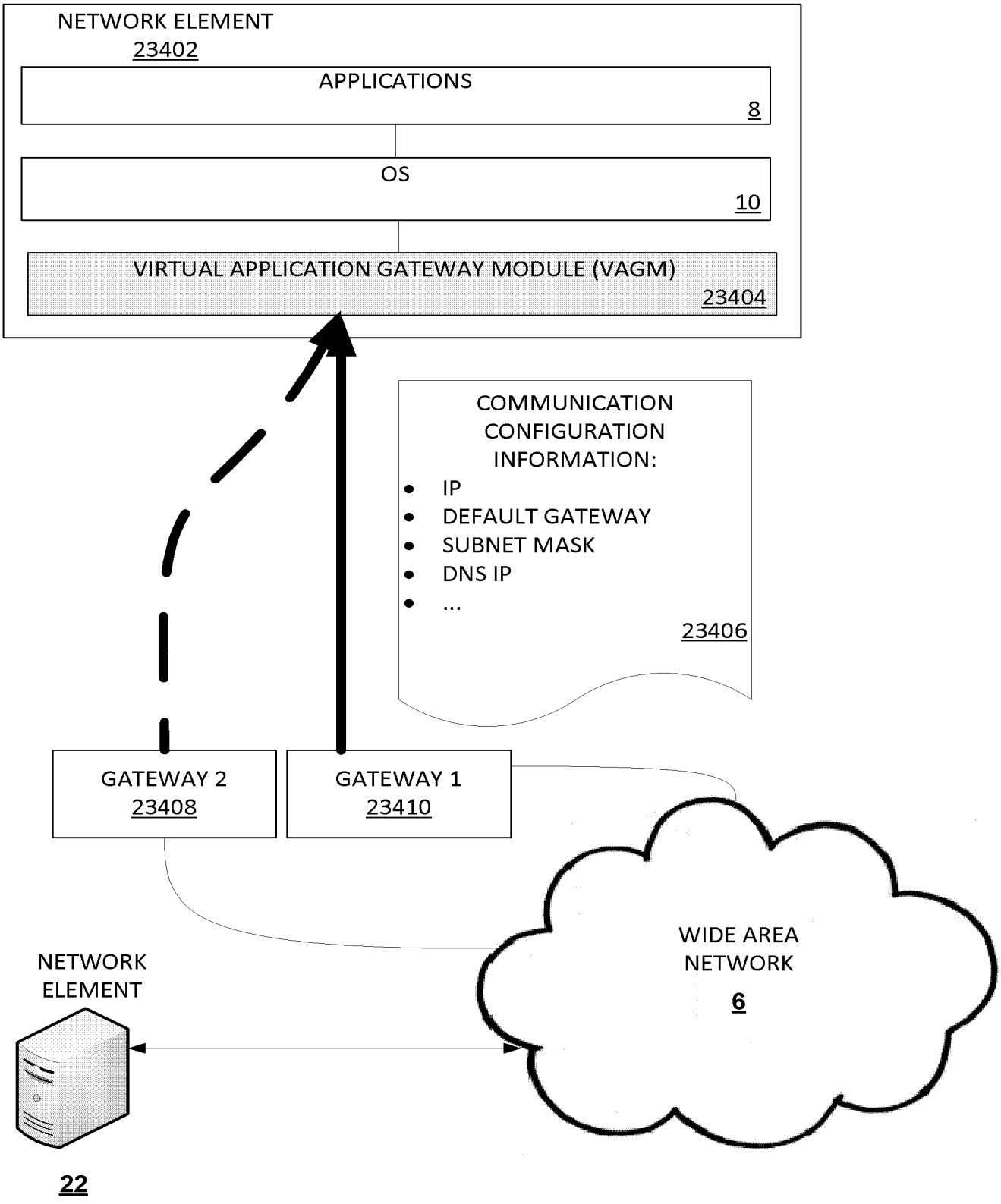


FIG. 236 – VIRTUAL APPLICATION GATEWAY – FLOWCHART

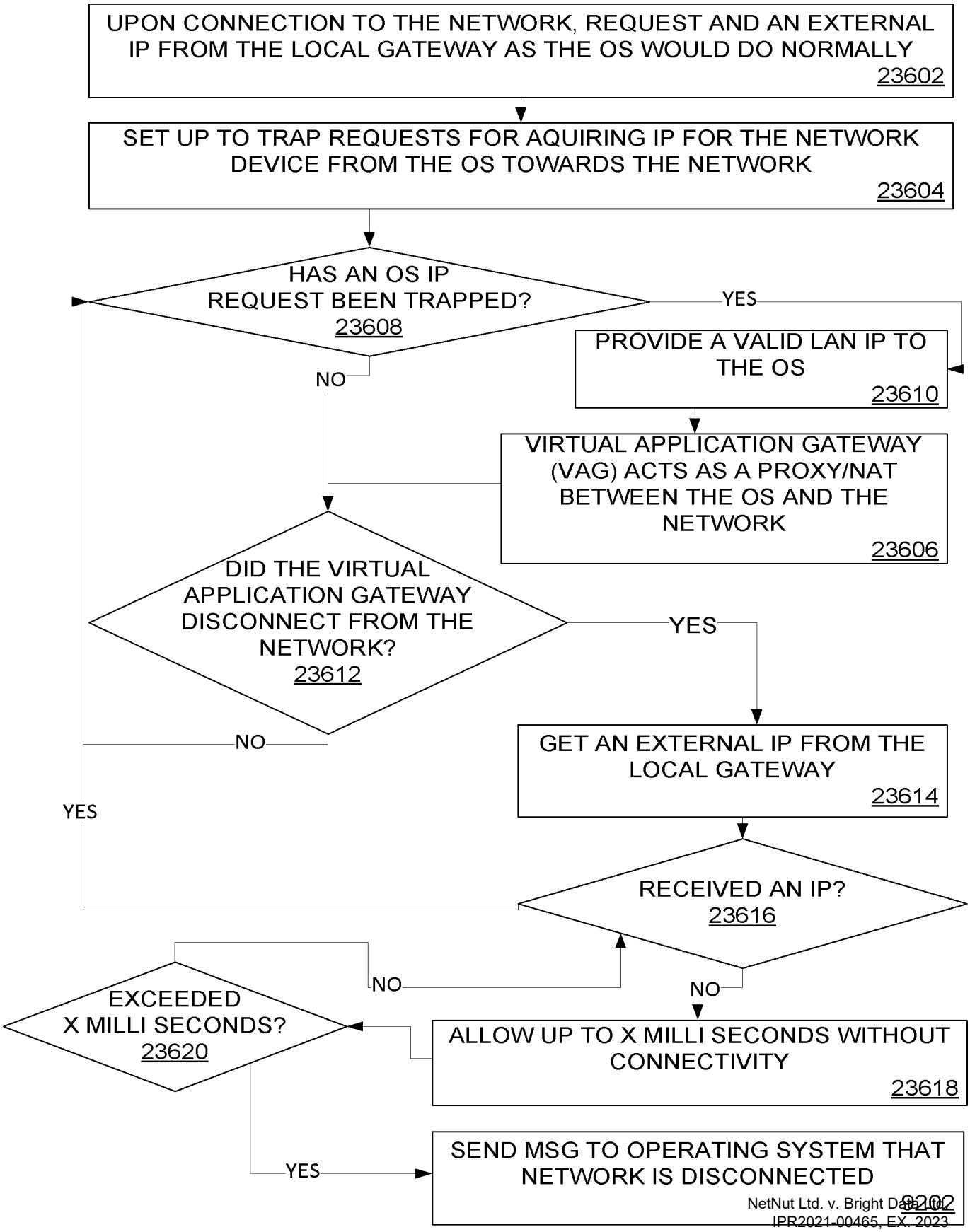


FIG. 240 – RELIABLE CONNECTION PROXY – PRIOR ART

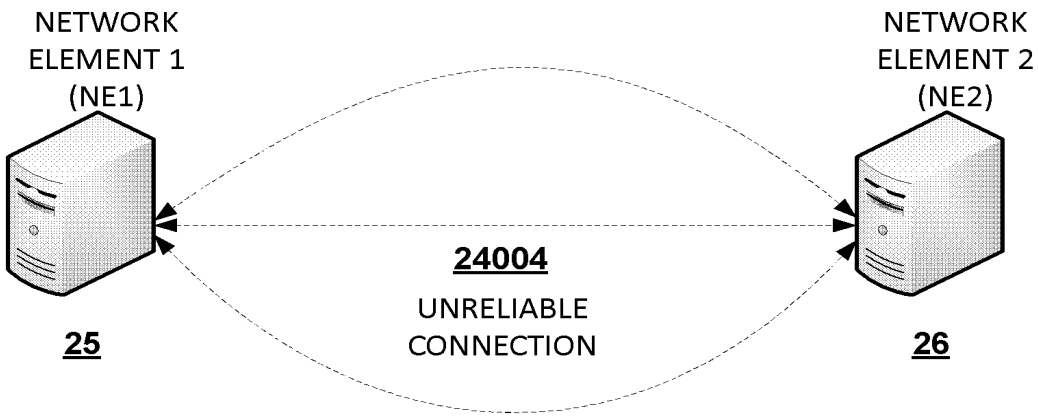


FIG. 242 – DIAGRAM

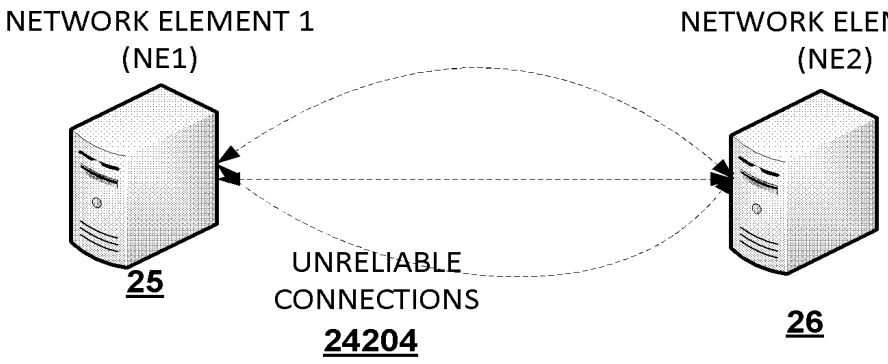


FIG. 243 – DIAGRAM

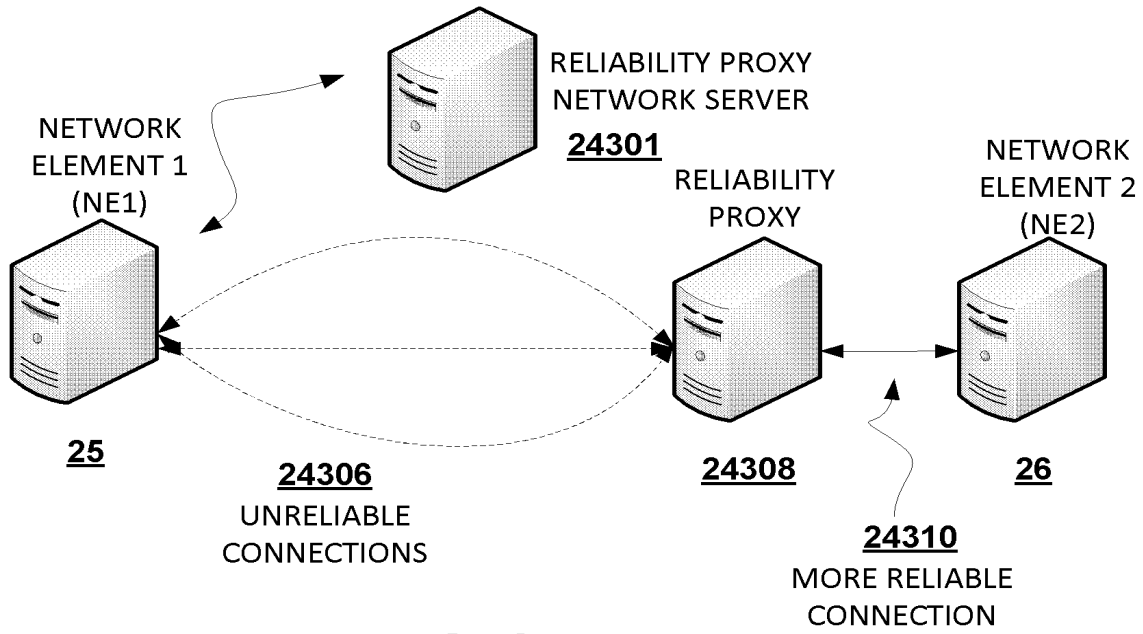


FIG. 244

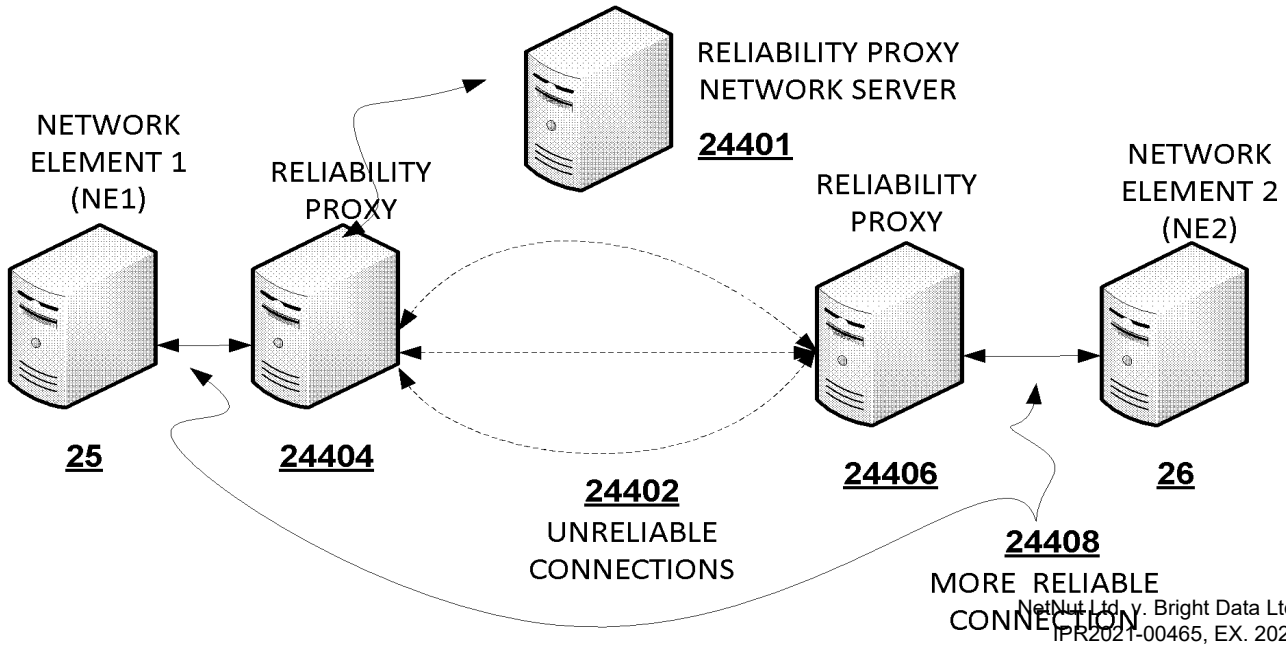


FIG. 246 – RELIABLE CONNECTION PROXY – FLOWCHART

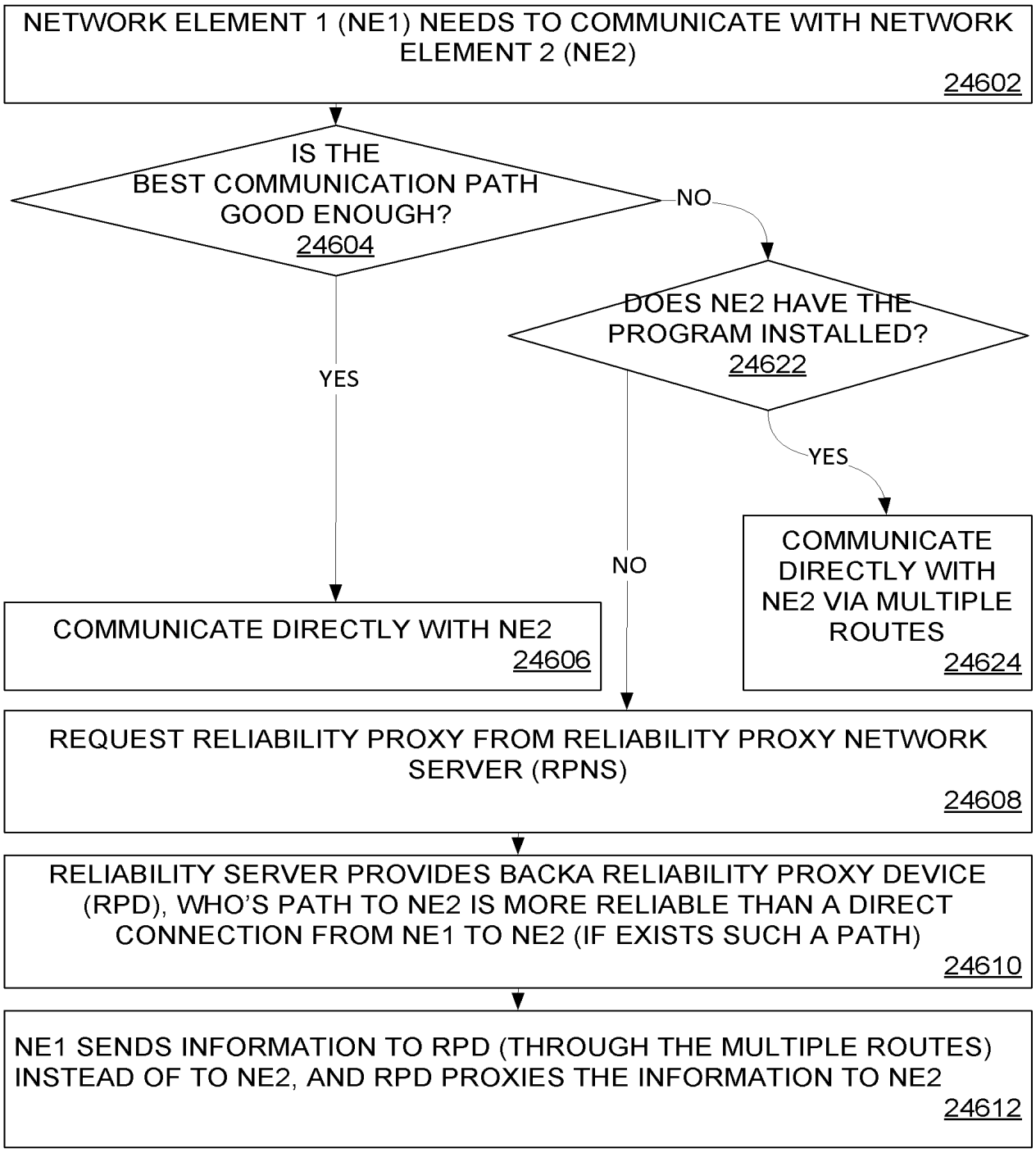


FIG. 247 -TRANSMITTING PACKETS OVER MULTIPLE ROUTES

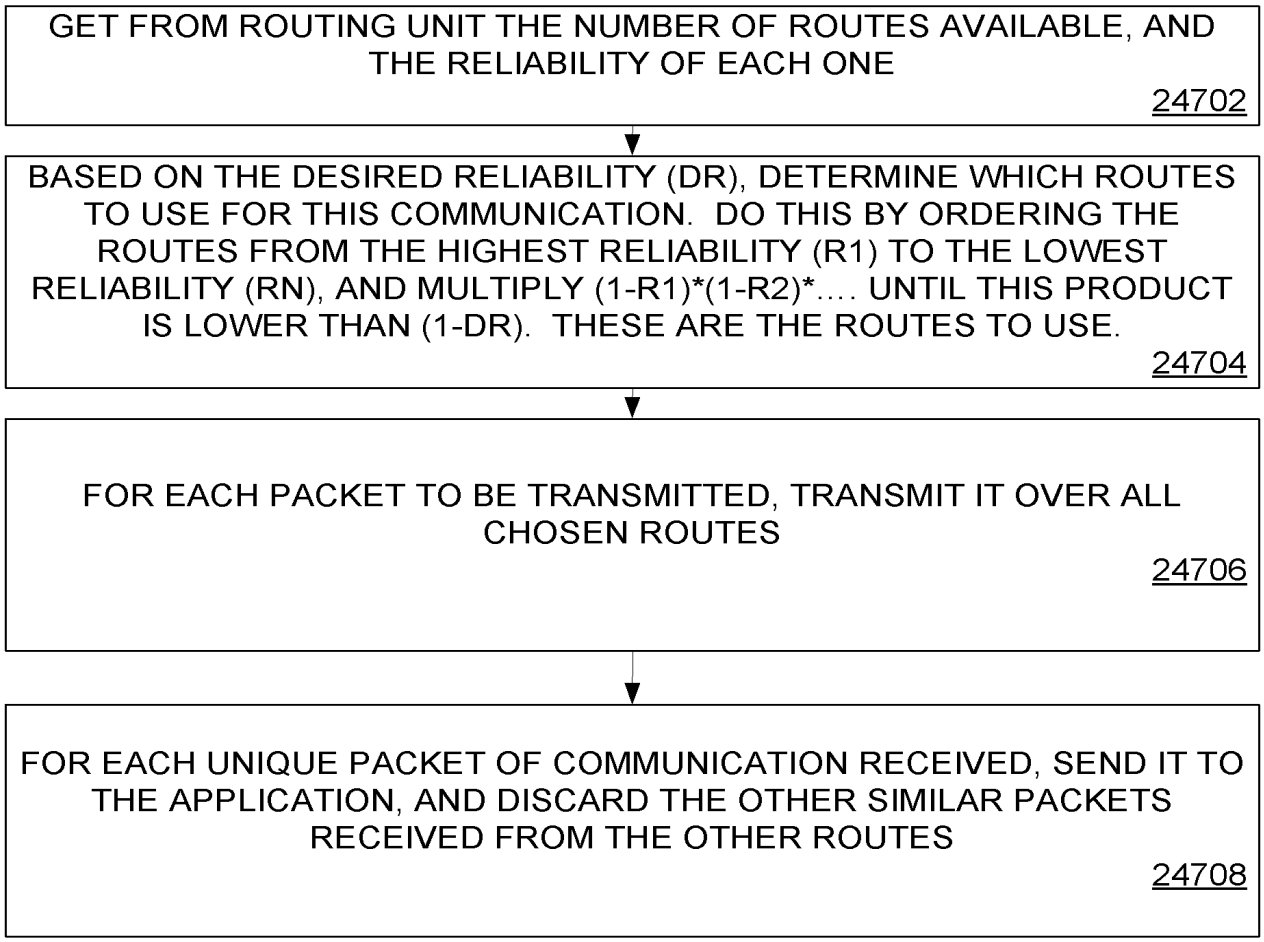


FIG. 248 – RELIABLE CONNECTION PROXY – FLOWCHART

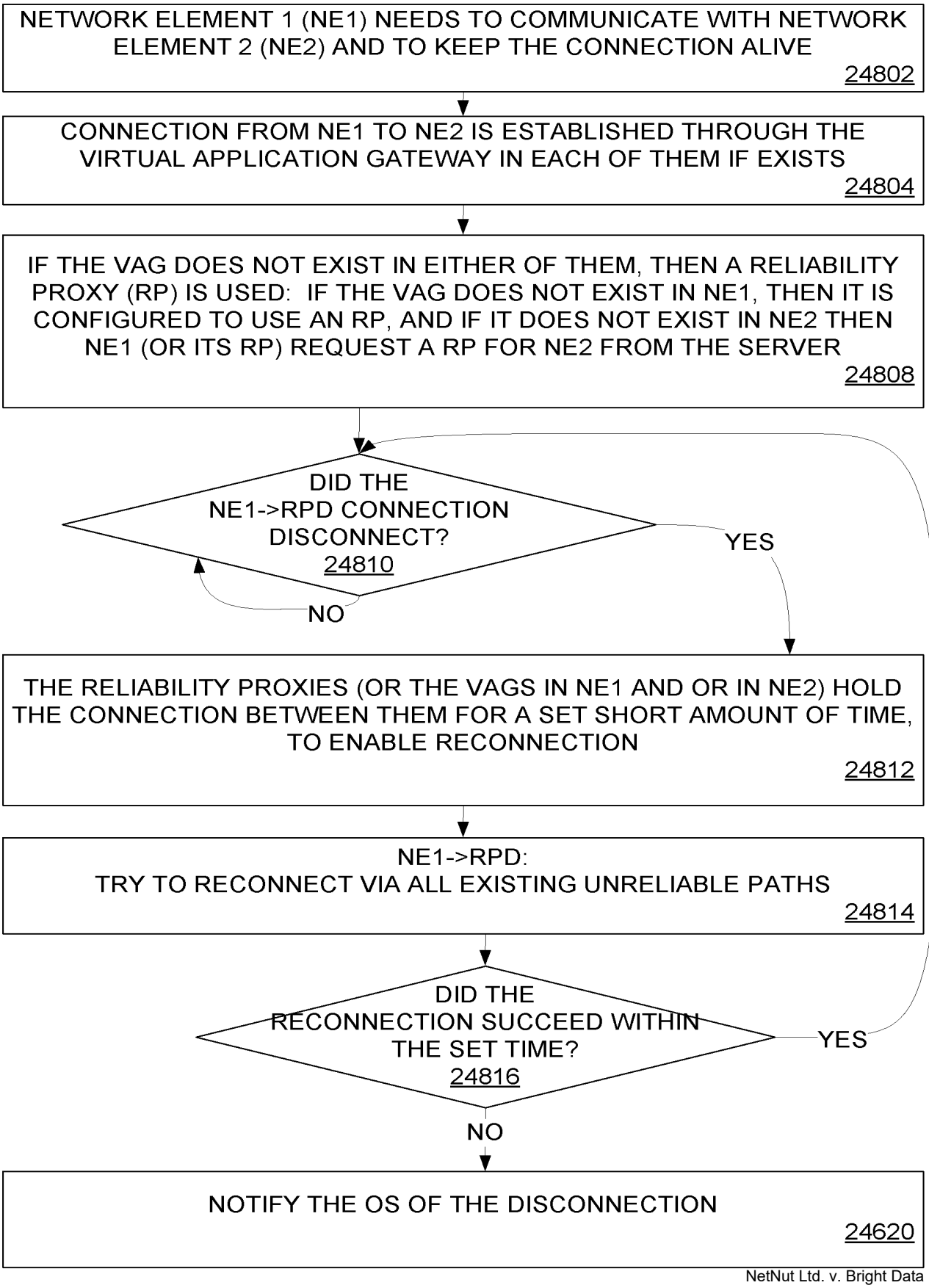


FIG. 260 – AGGRESSIVE DHCP – PRIOR ART DIAGRAM

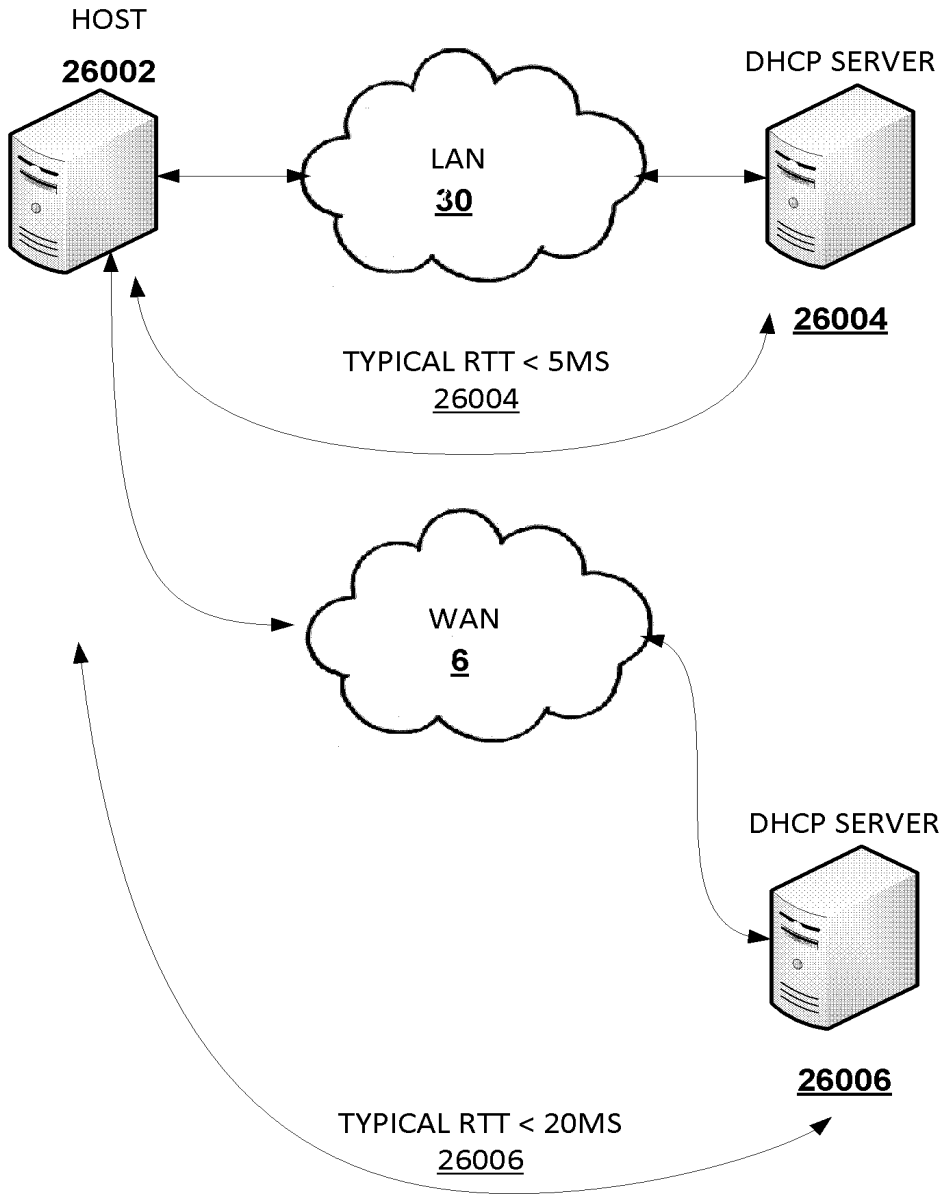


FIG. 262 – AGGRESSIVE DHCP – PRIOR ART FLOWCHART

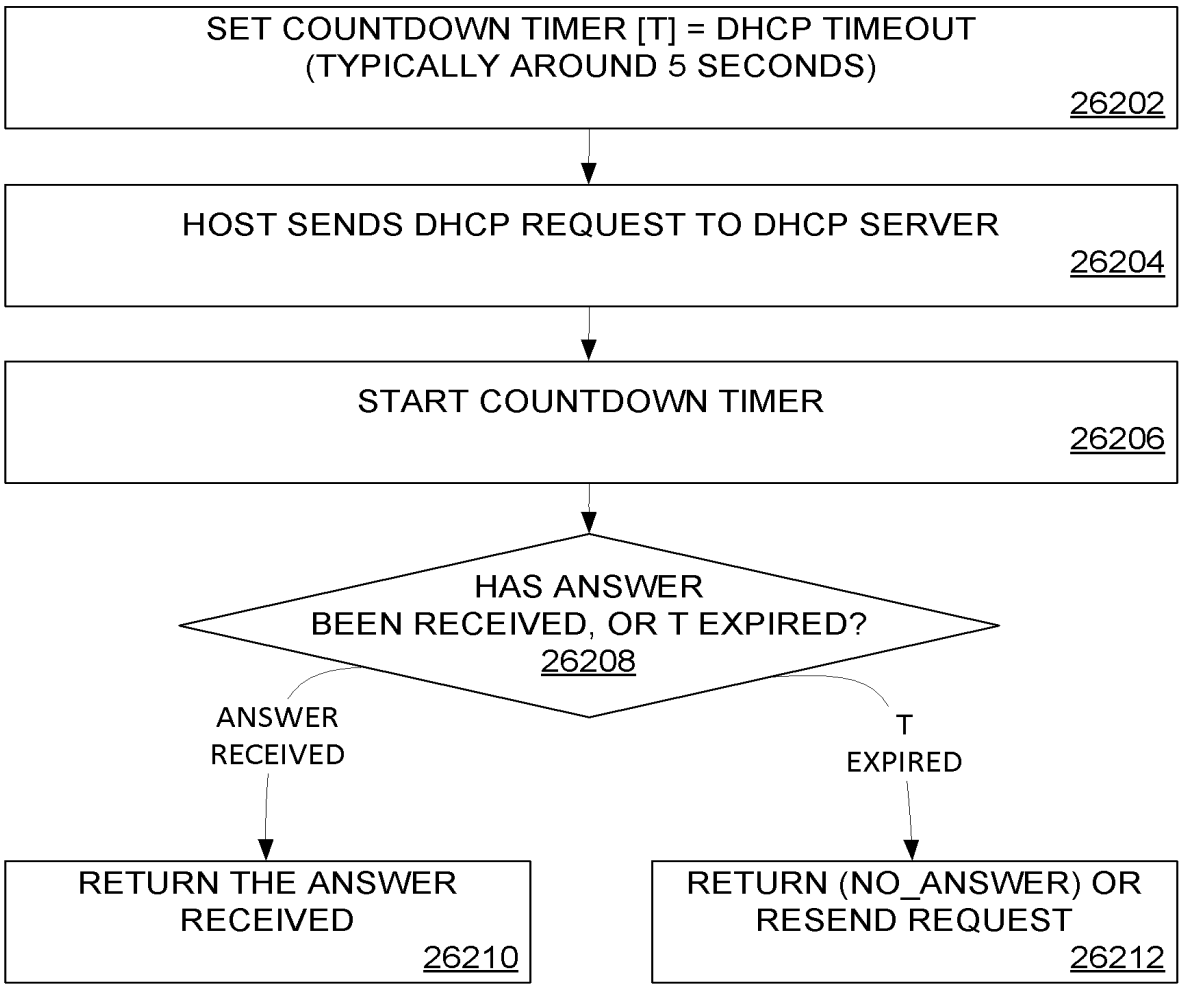


FIG. 264 – AGGRESSIVE DHCP – PRIOR ART FLOWCHART

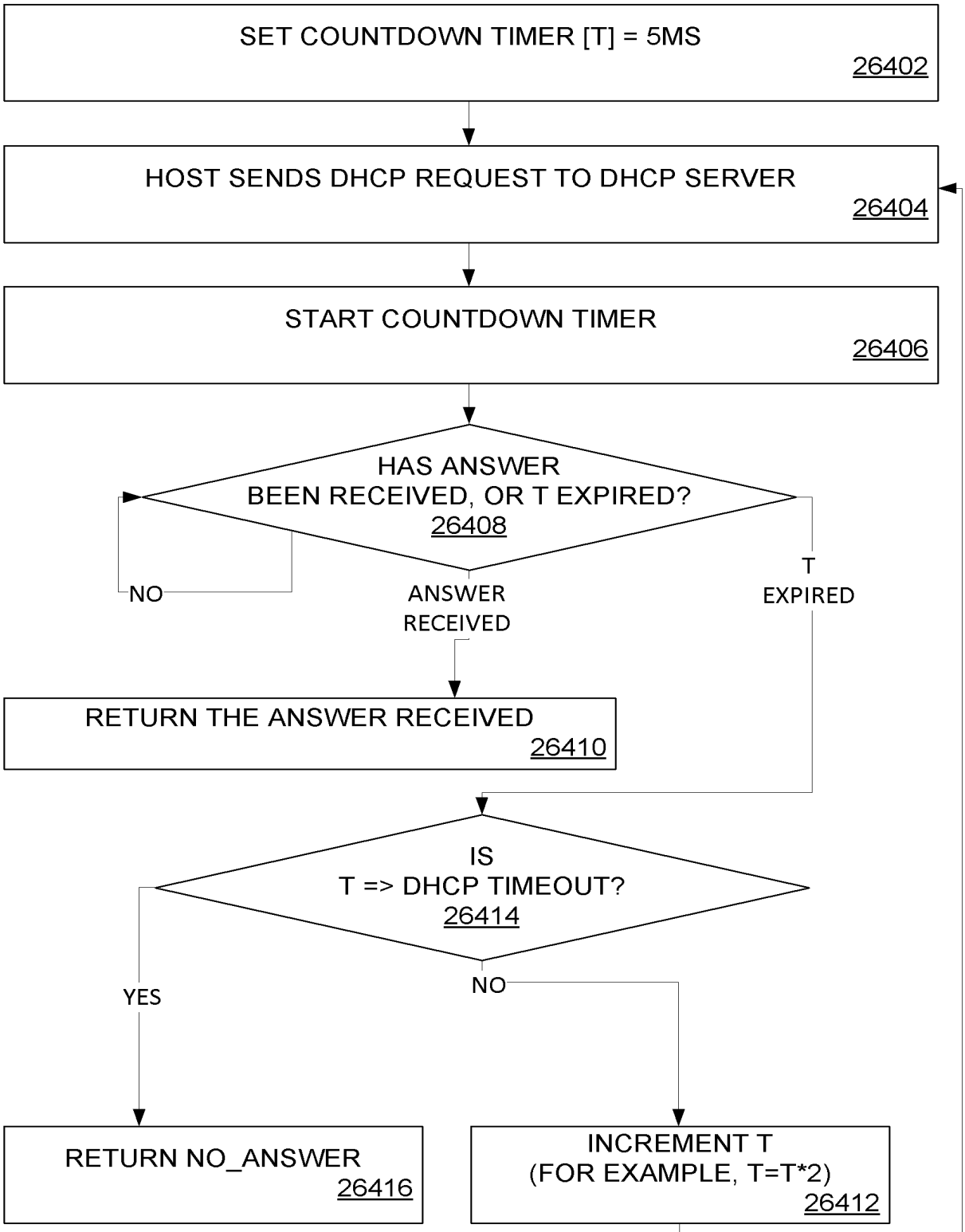


FIG. 270 – DATA / META DATA PRIORITIZATION – PRIOR ART – DIAGRAM

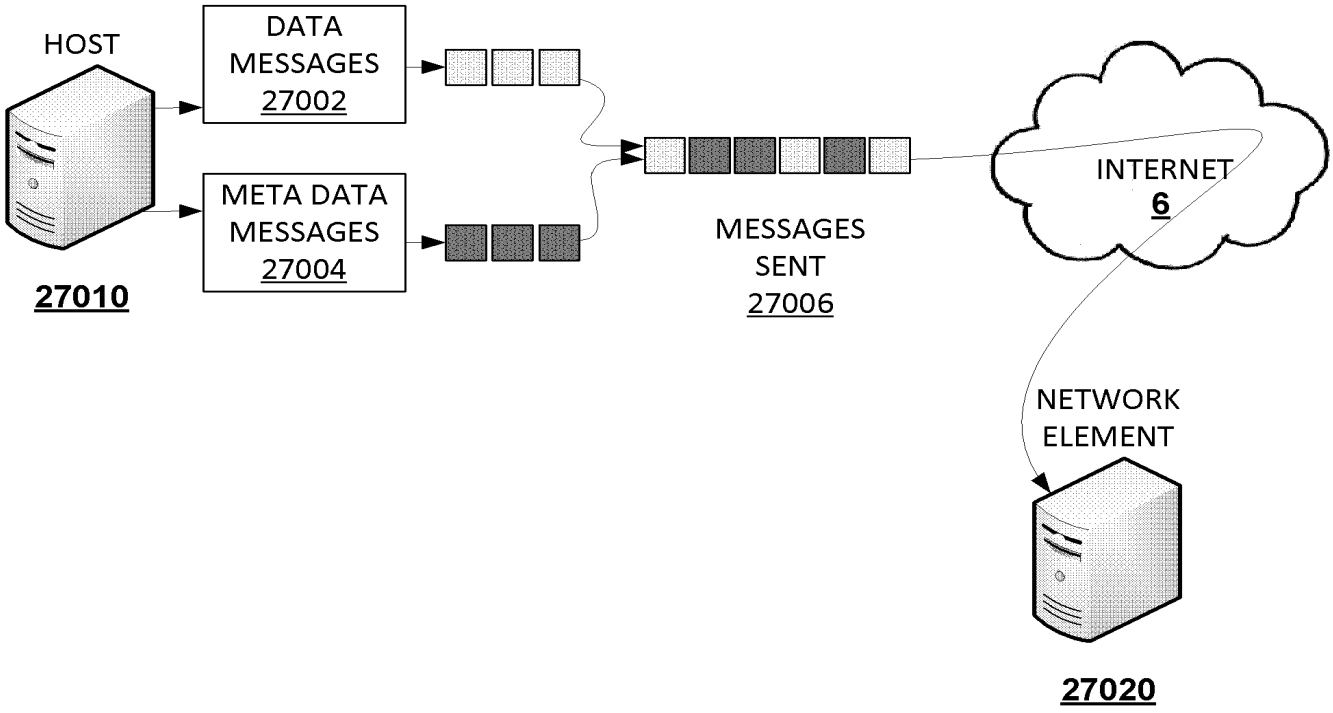


FIG. 272 – DATA / META DATA PRIORITIZATION – DIAGRAM

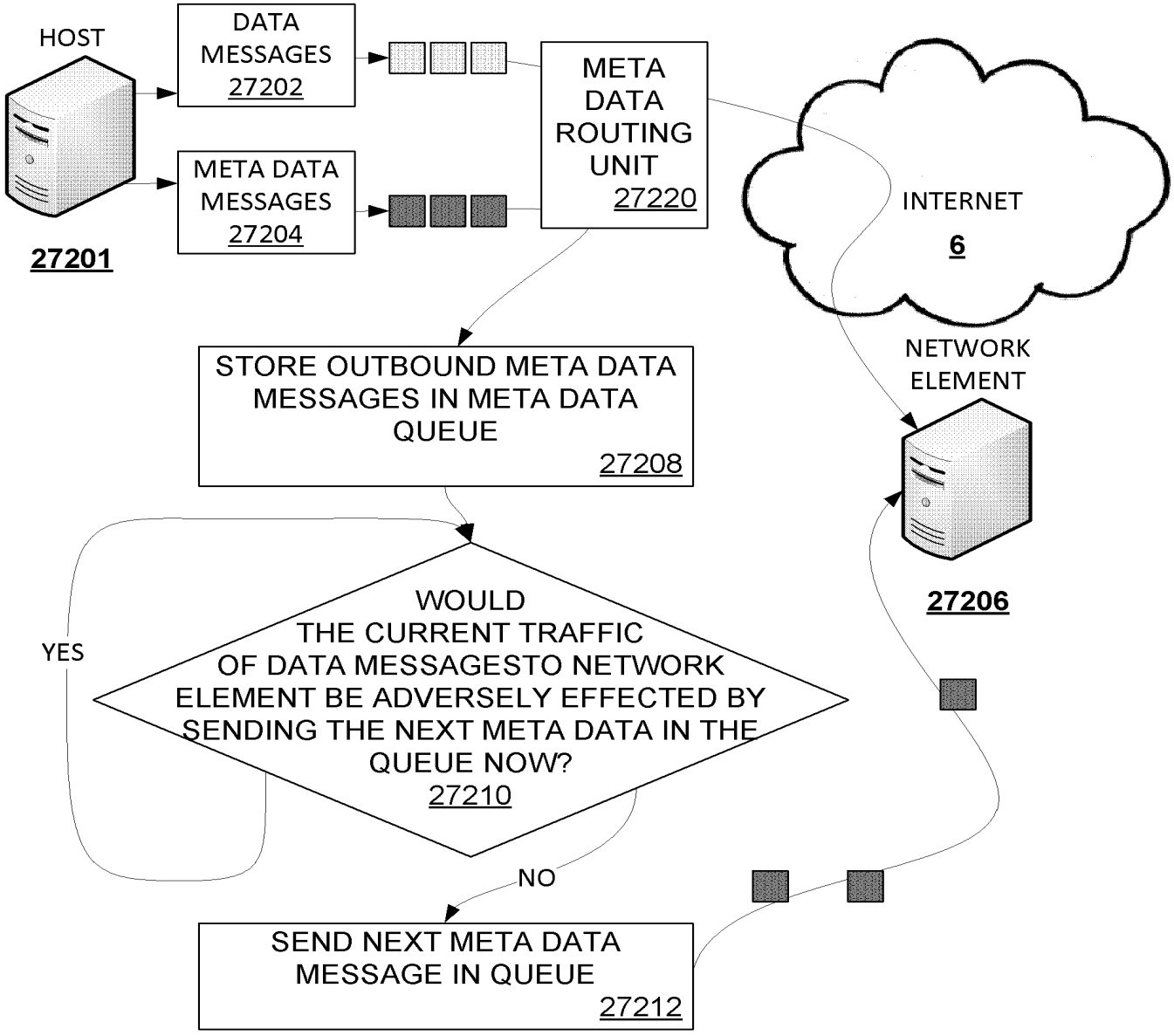


FIG. 300 – COMPRESSION WITH NON DETERMINISTIC SHARED DICTIONARY – PRIOR ART DIAGRAM

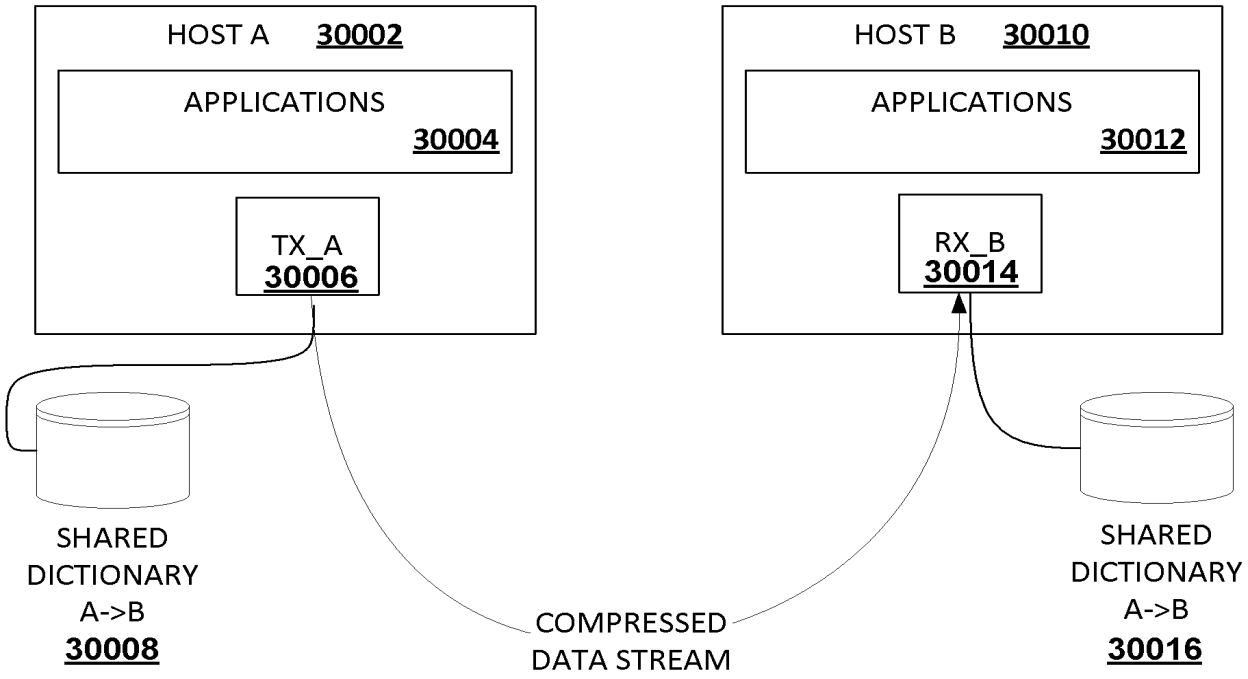


FIG. 301 – COMPRESSION WITH NON DETERMINISTIC SHARED DICTIONARY – DIAGRAM

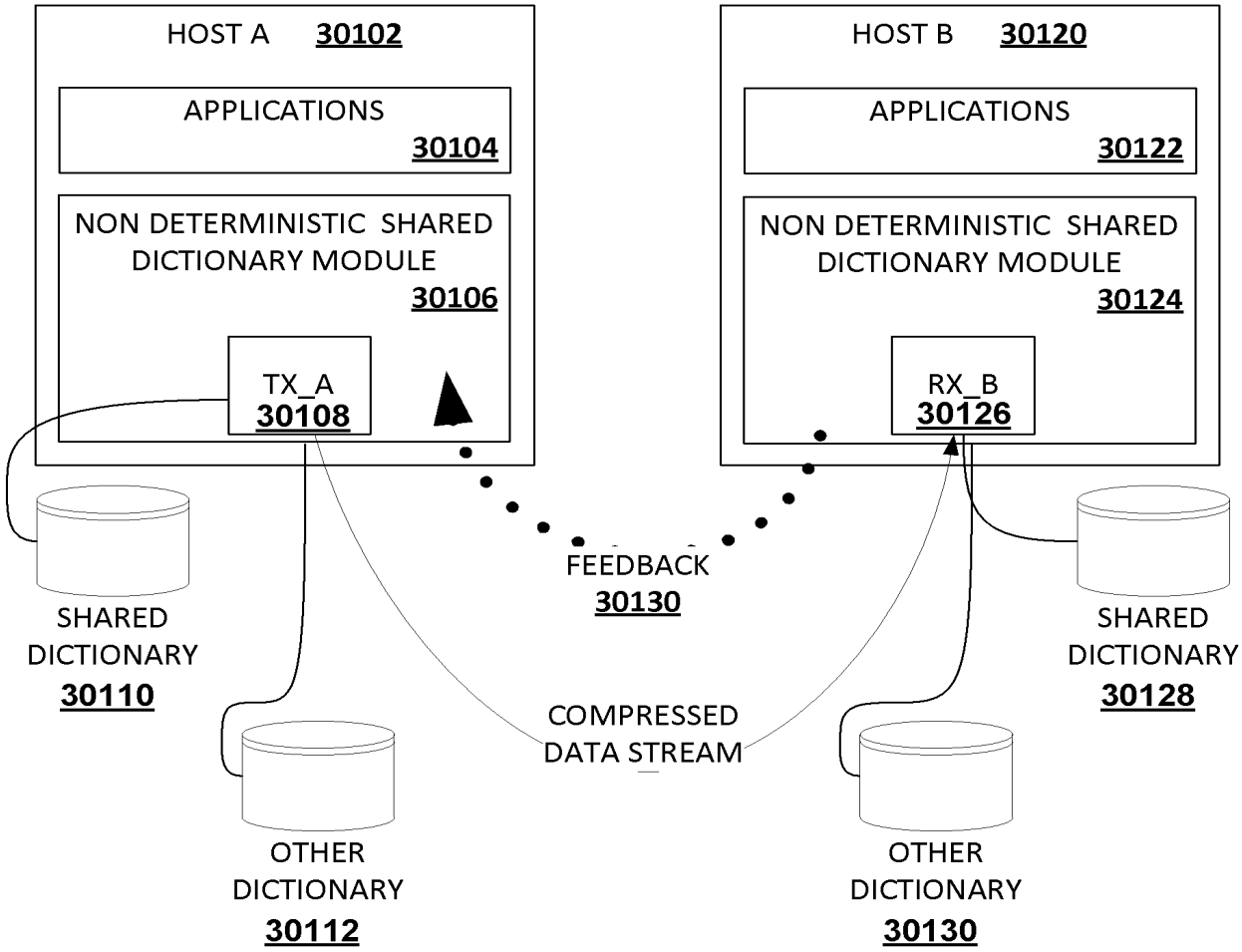


FIG. 302 – COMPRESSION WITH NON DETERMINISTIC SHARED DICTIONARY – FLOWCHART

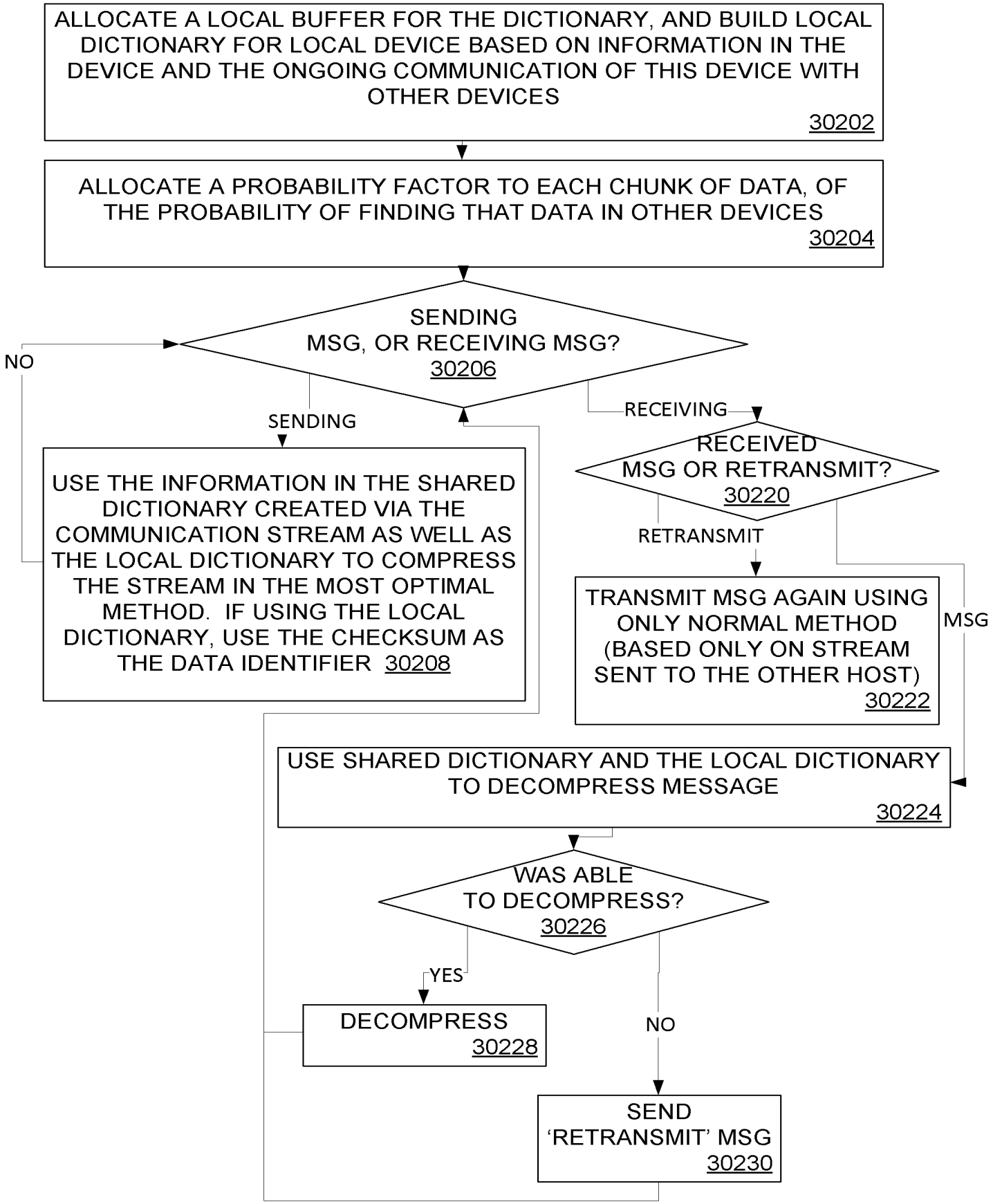


FIG. 304 – COMPRESSION WITH NON DETERMINISTIC SHARED DICTIONARY – IMPROVEMENT

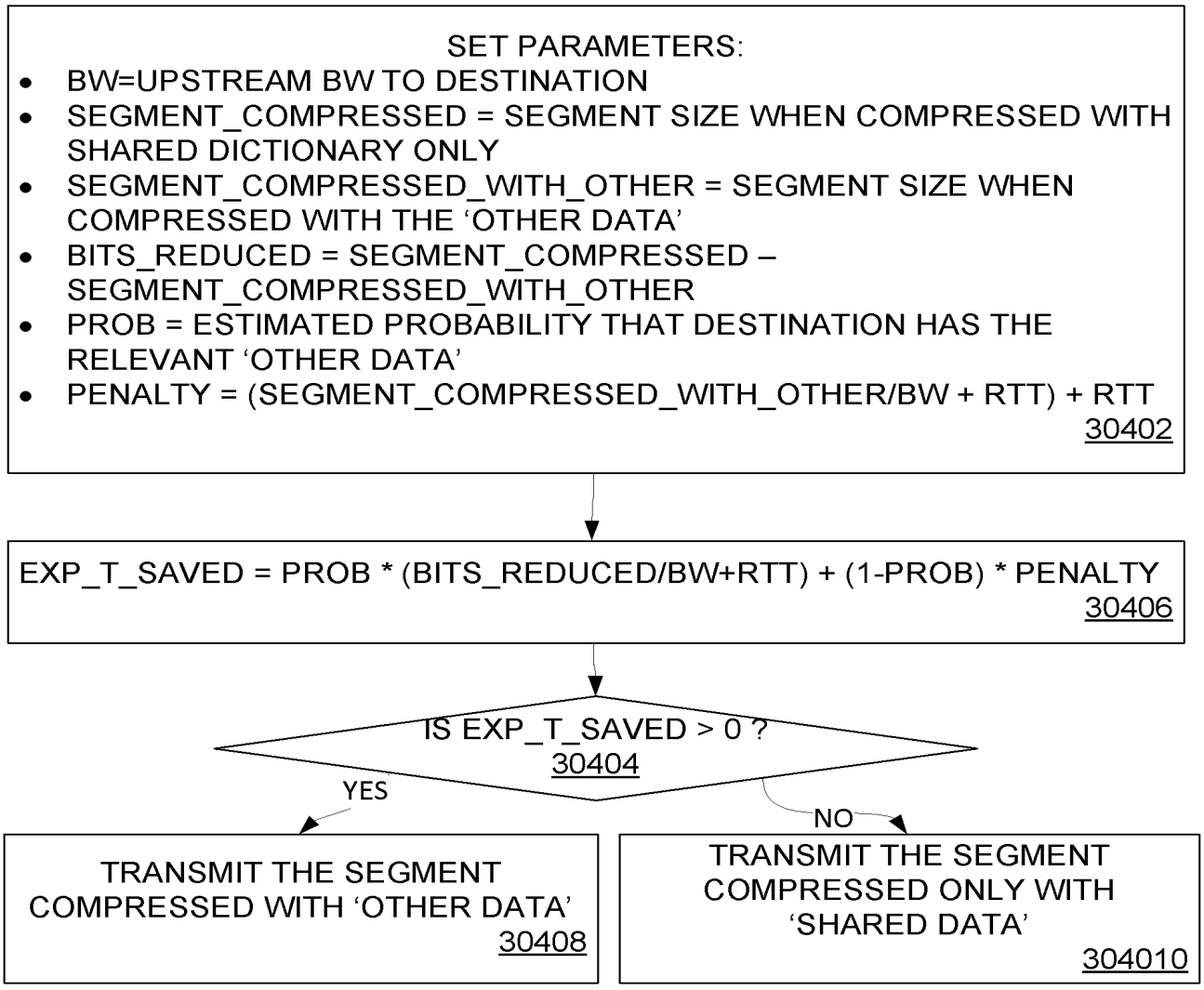


FIG. 310 – TWO WAY STREAM COMPRESSION – PRIOR ART DIAGRAM

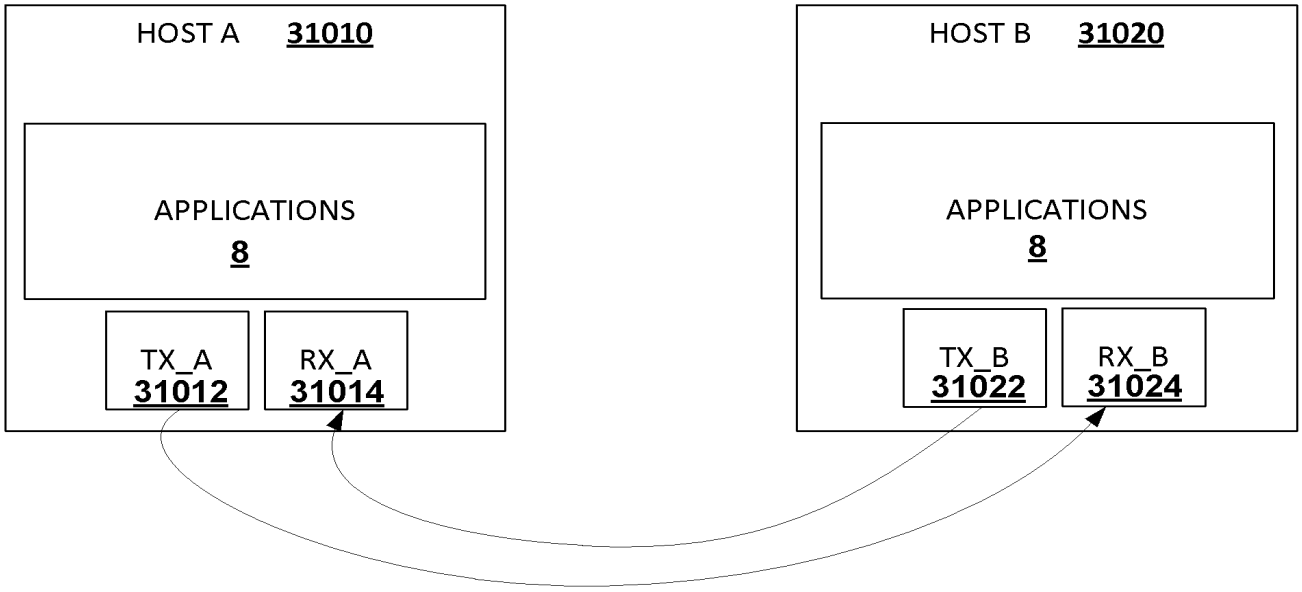


FIG. 312 – TWO WAY STREAM COMPRESSION – DIAGRAM

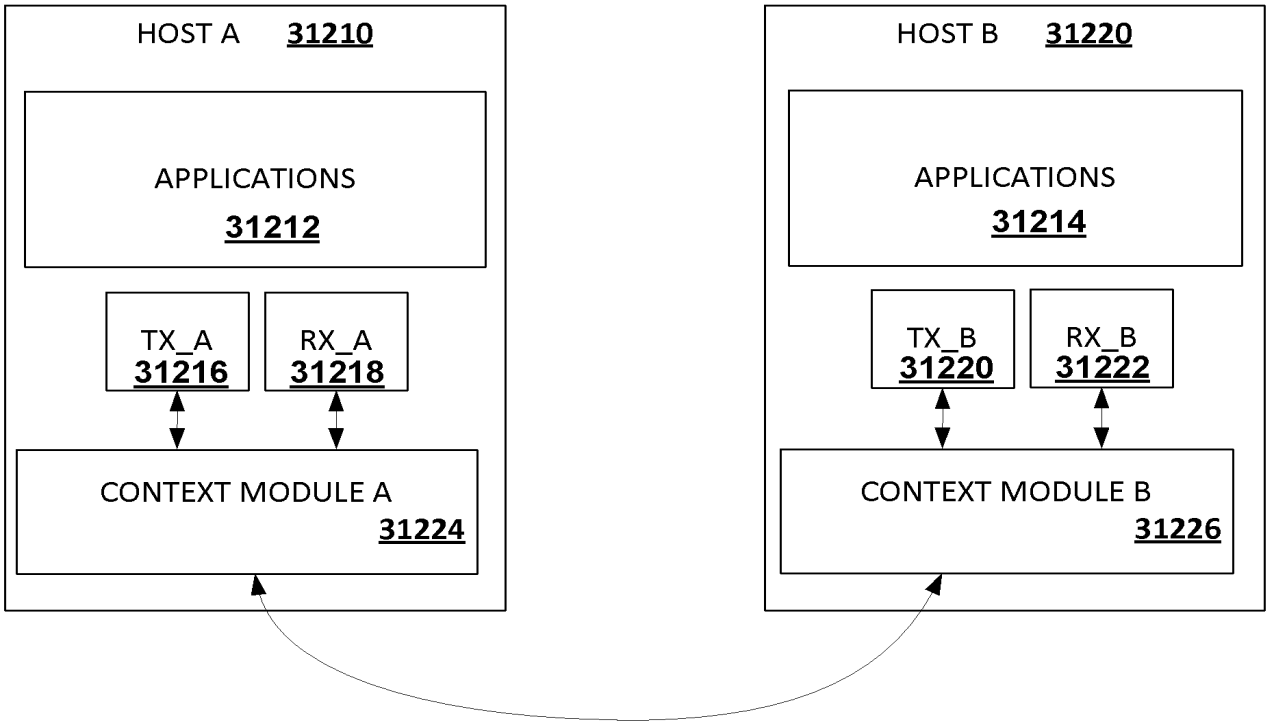
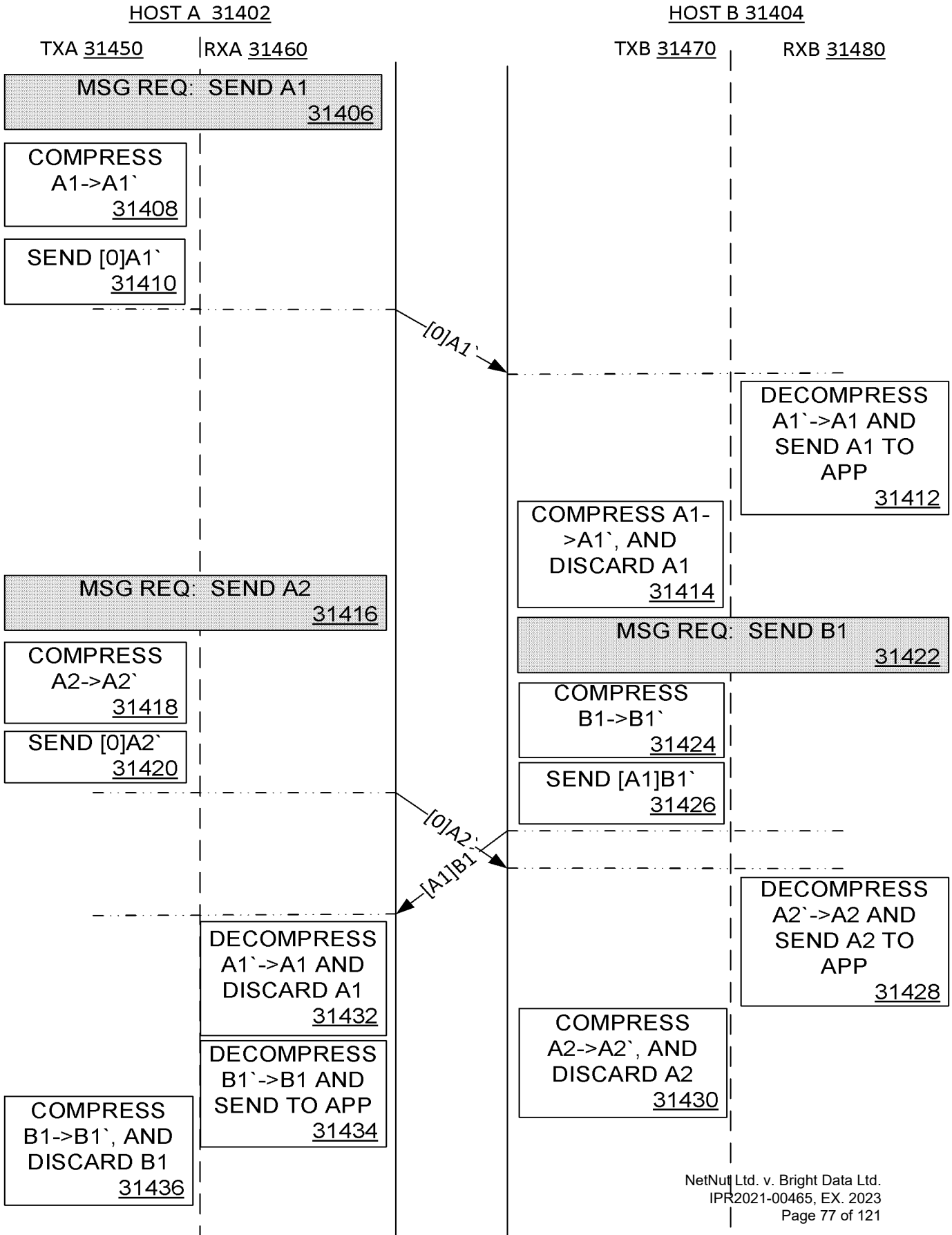


FIG. 314 – TWO WAY STREAM COMPRESSION – FLOW DIAGRAM



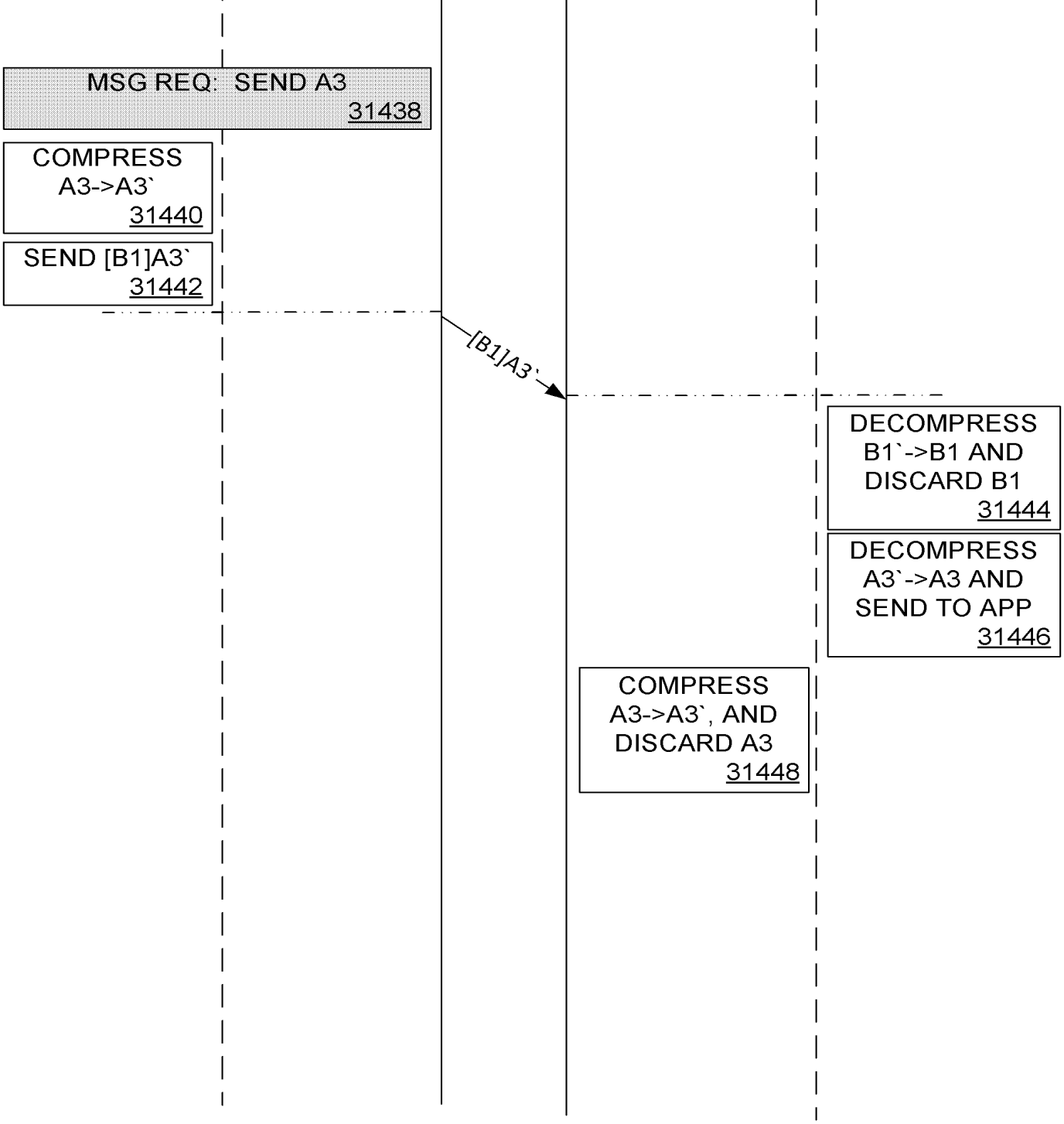

MSG REQ: SEND A3
31438
COMPRESS
A3->A3`
31440
SEND [B1]A3`
31442
[B1]A3`
DECOMPRESS
B1`->B1 AND
DISCARD B1
31444
DECOMPRESS
A3`->A3 AND
SEND TO APP
31446
COMPRESS
A3->A3`, AND
DISCARD A3
31448

FIG. 318 – TWO WAY STREAM COMPRESSION – FLOW CHART

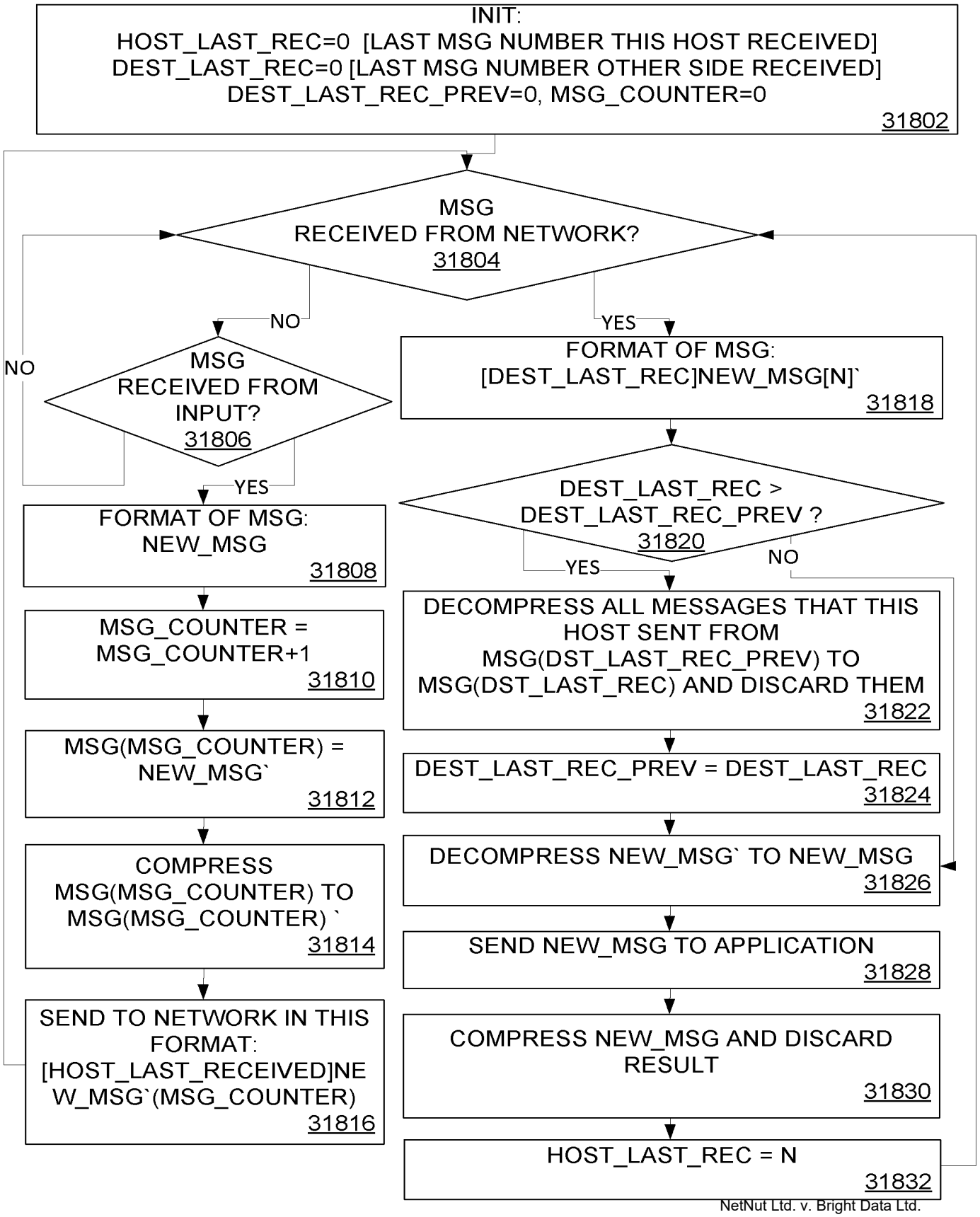


FIG. 400 – ACCESS POINT SETUP

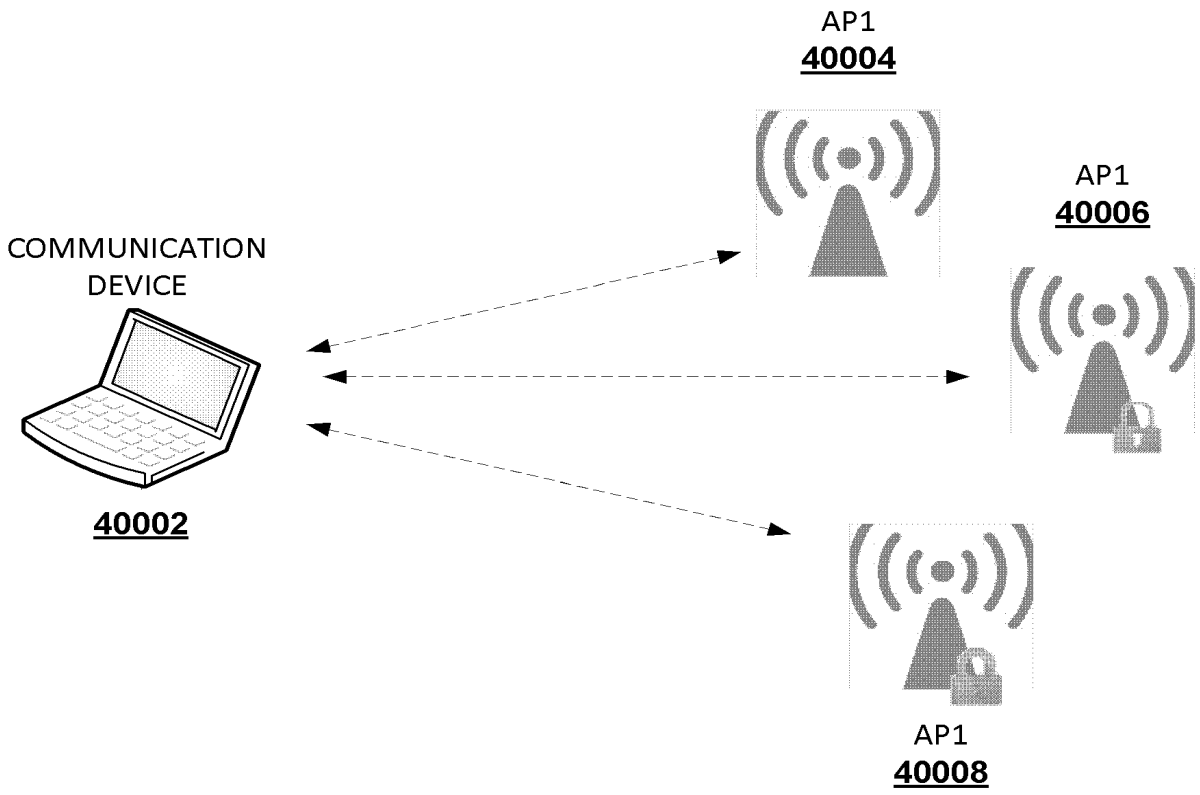


FIG. 402 – GUESSING PASSWORDS FOR LOCKED ACCESS POINTS - FLOWCHART

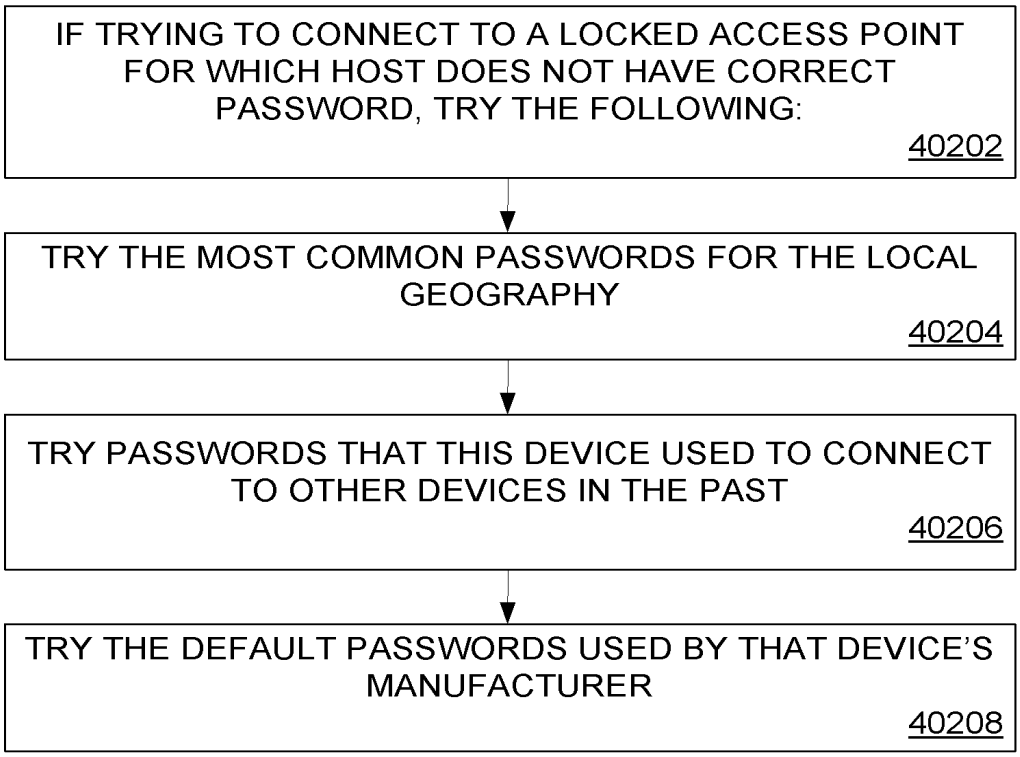


FIG. 404 – PASSWORD SHARING – PRIOR ART - DIAGRAM

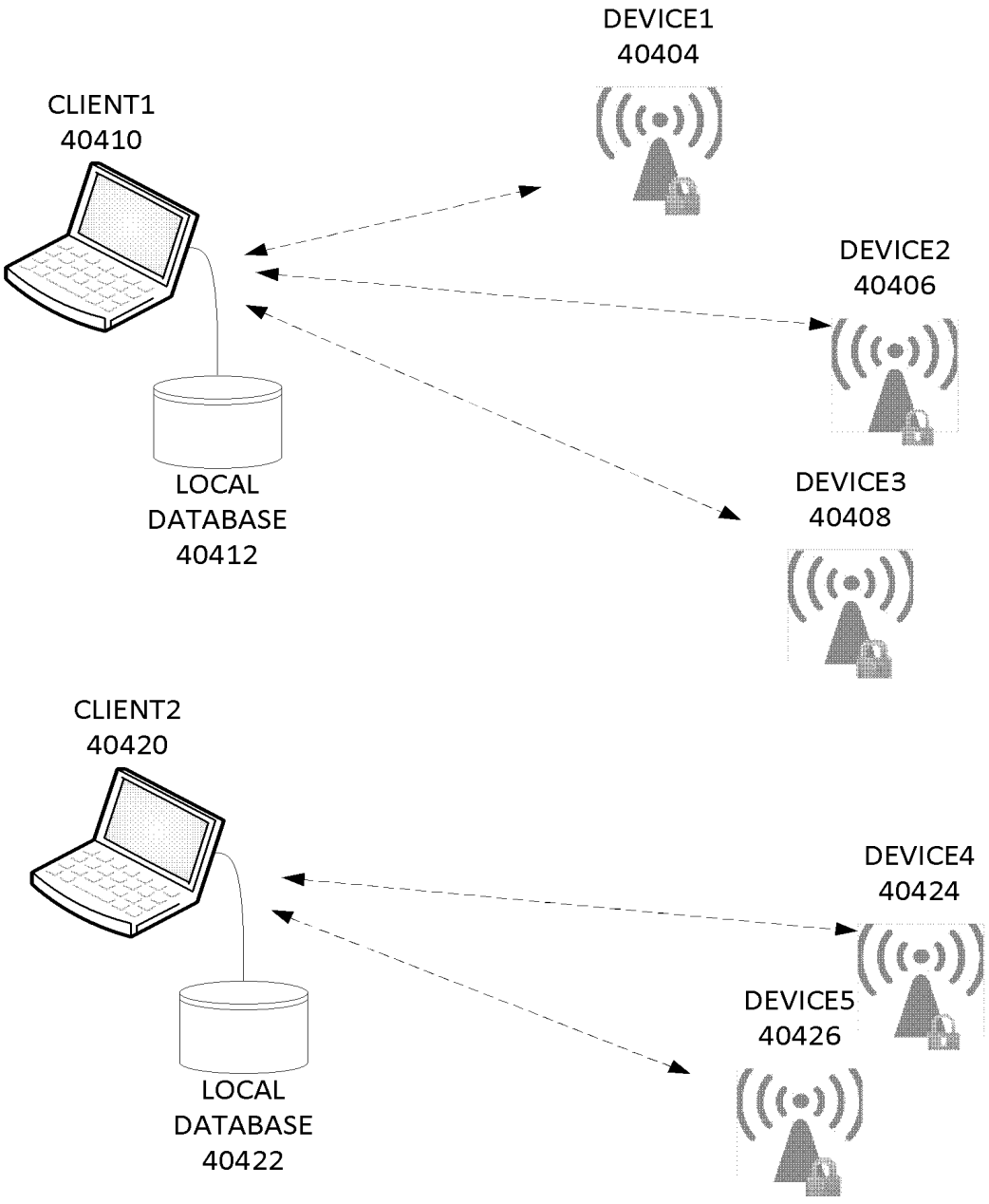


FIG. 406 – PASSWORD SHARING – DIAGRAM

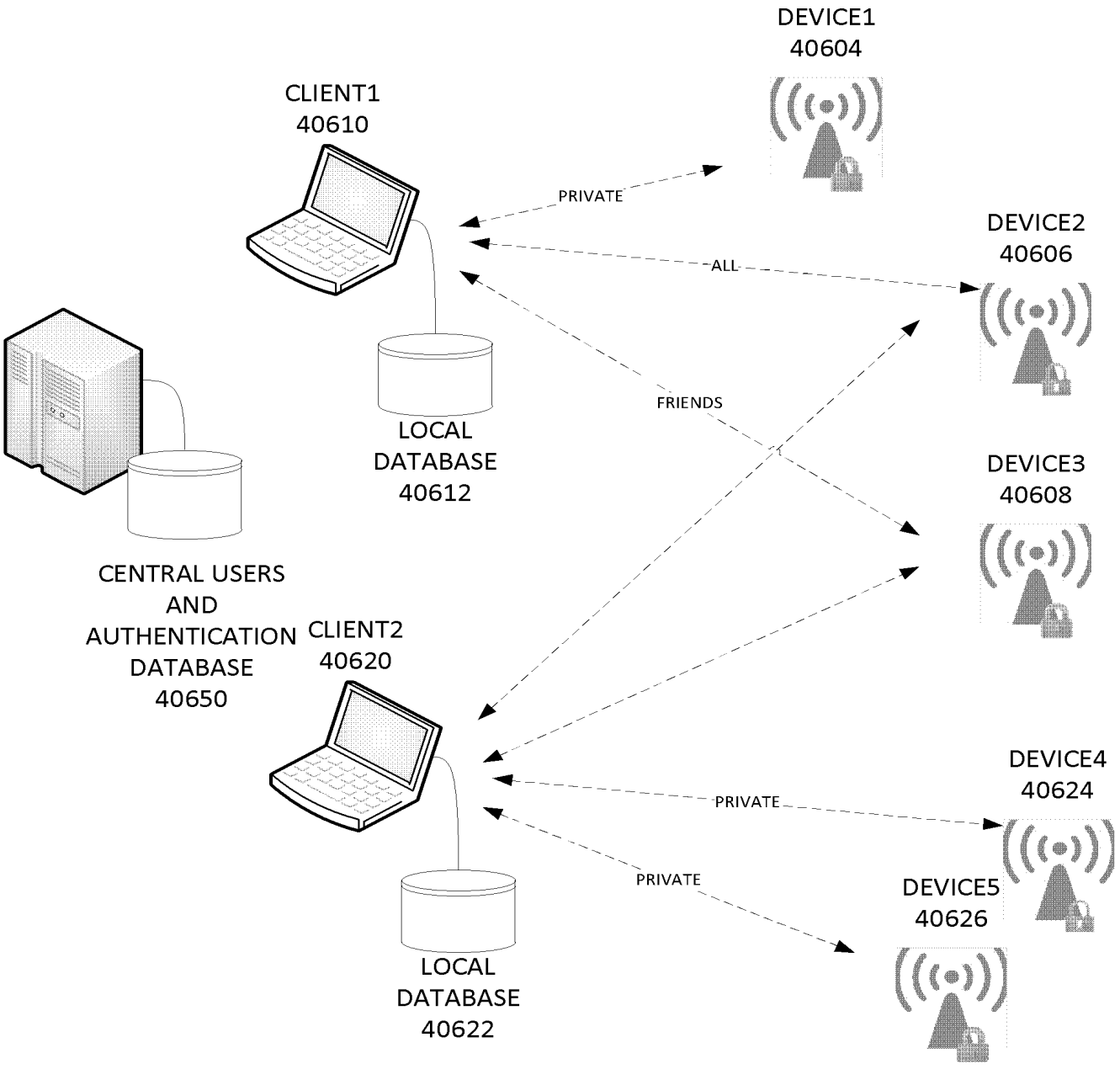


FIG. 408 – PASSWORD SHARING - FLOWCHART

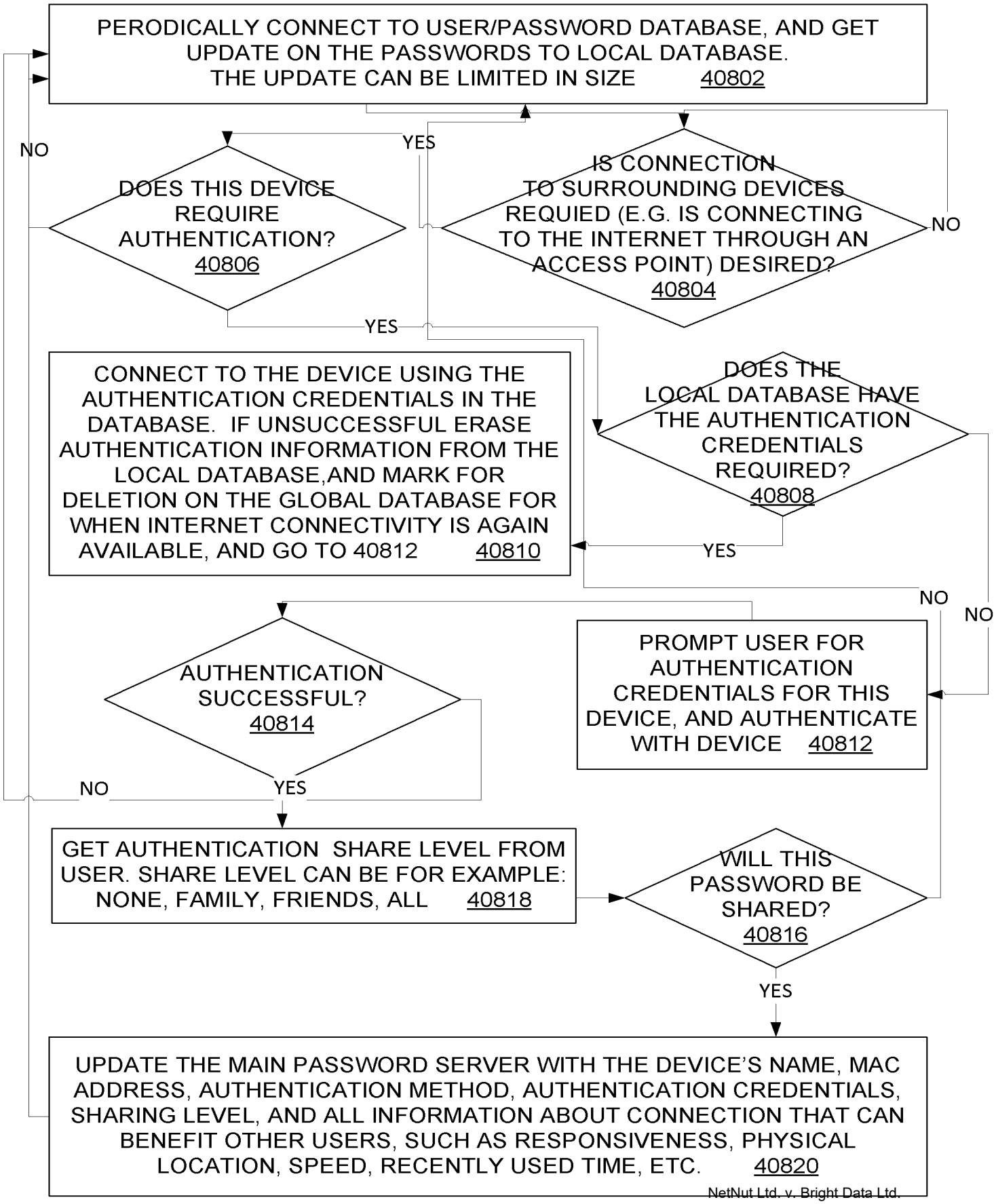
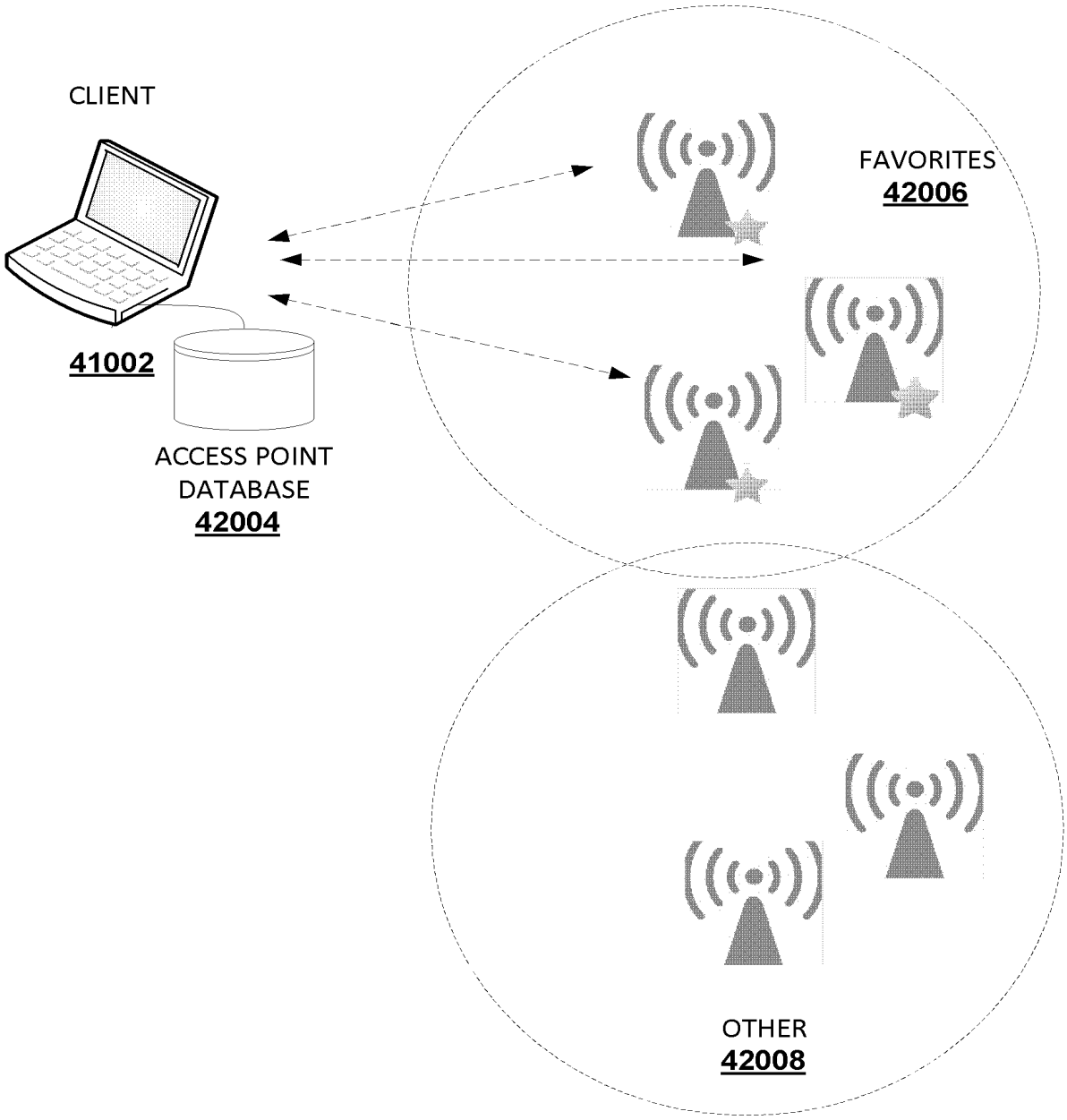


FIG. 410 – SMART CUTOFF – PRIOR ART - DIAGRAM



WHEN CLIENT IS STARTED UP, IT CONNECTS TO ONE OF THE 'FAVORITE' ACCESS POINTS, TYPICALLY SORTED BY SIGNAL STRENGTH

9202

FIG. 412 – SMART CUTOFF – DIAGRAM

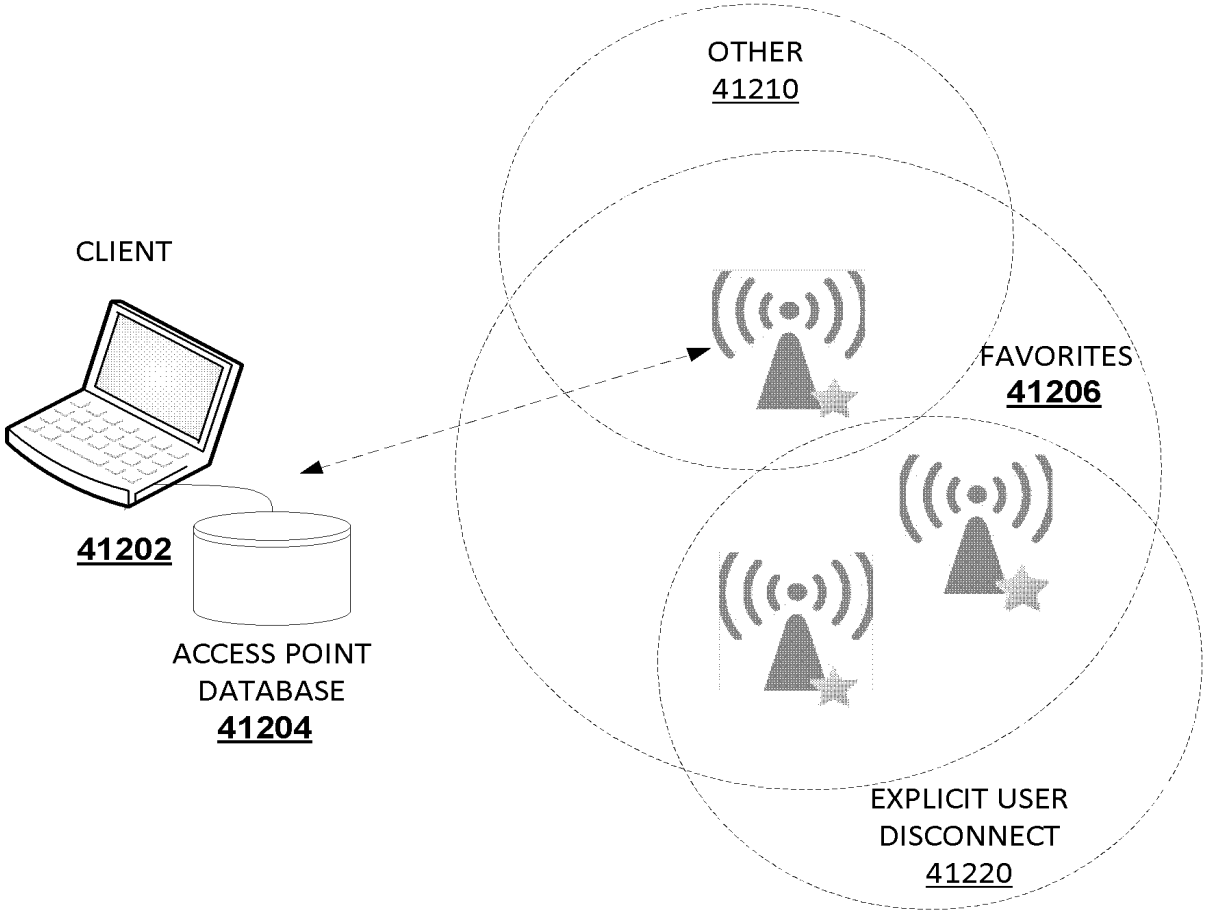


FIG. 414 – SMART CUTOFF – FLOWCHART

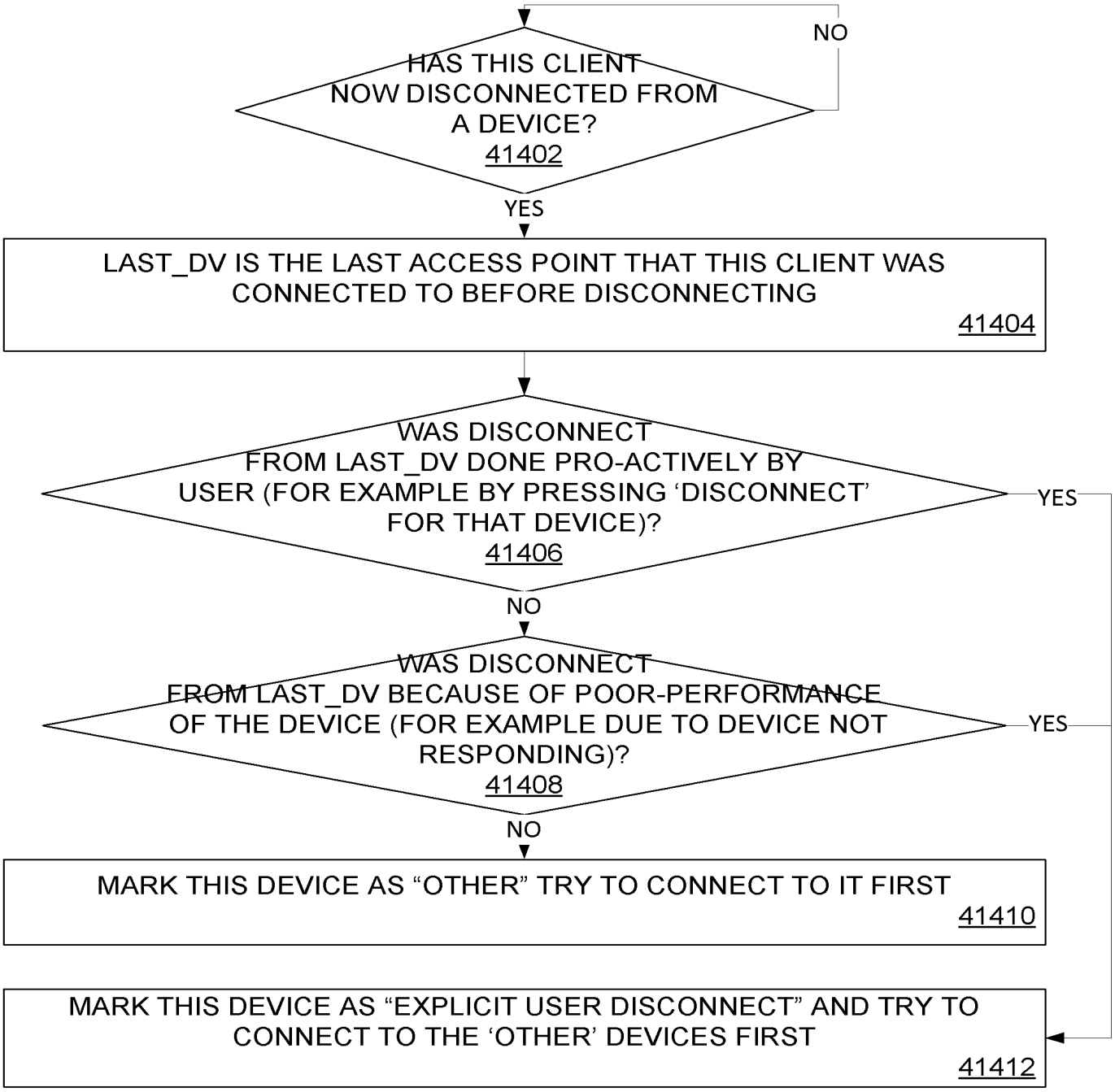


FIG. 500 – QUICK IMAGE RESIZING – PRIOR ART - DIAGRAM

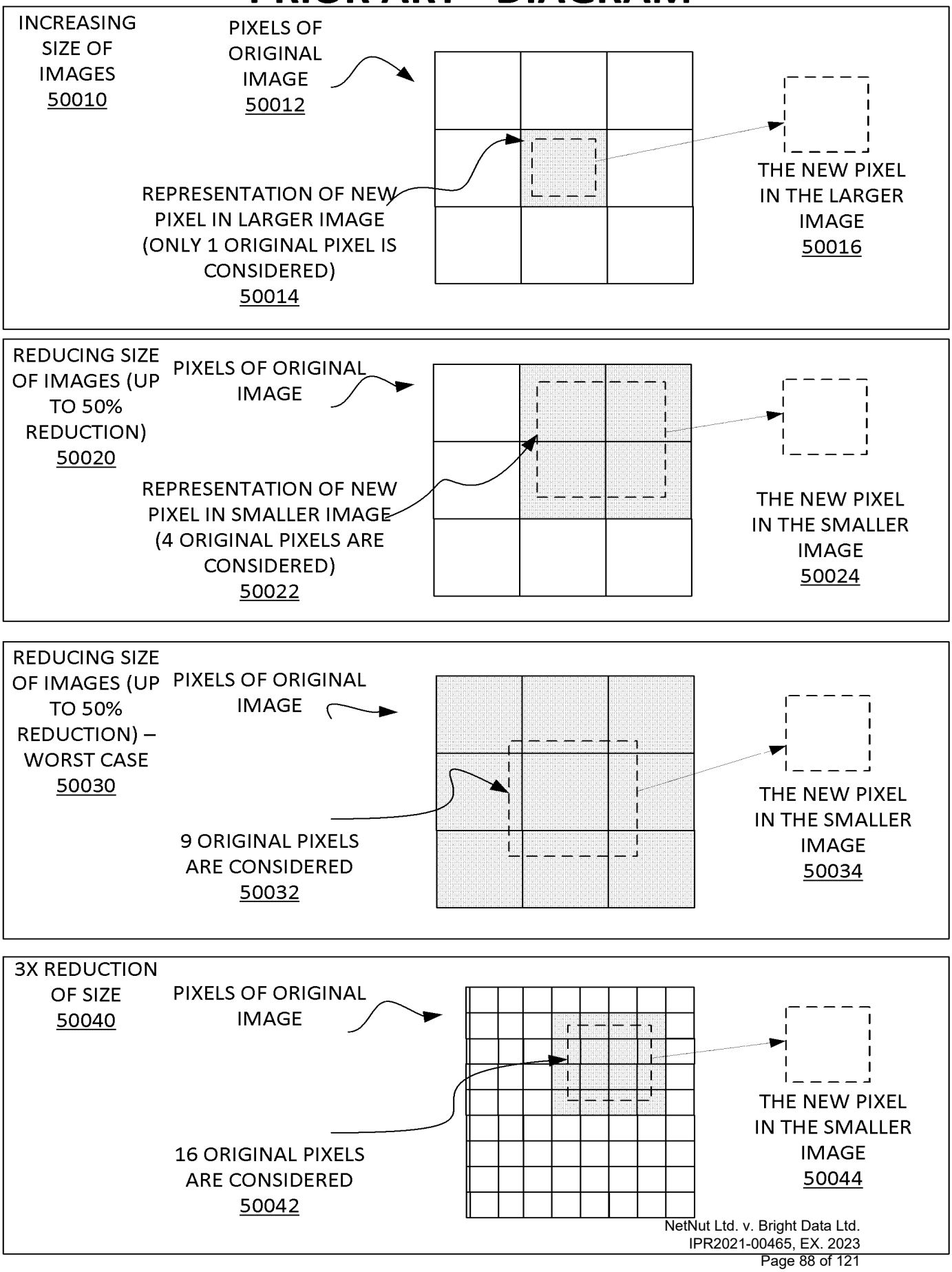


FIG. 504 – QUICK IMAGE RESIZING – FLOWCHART

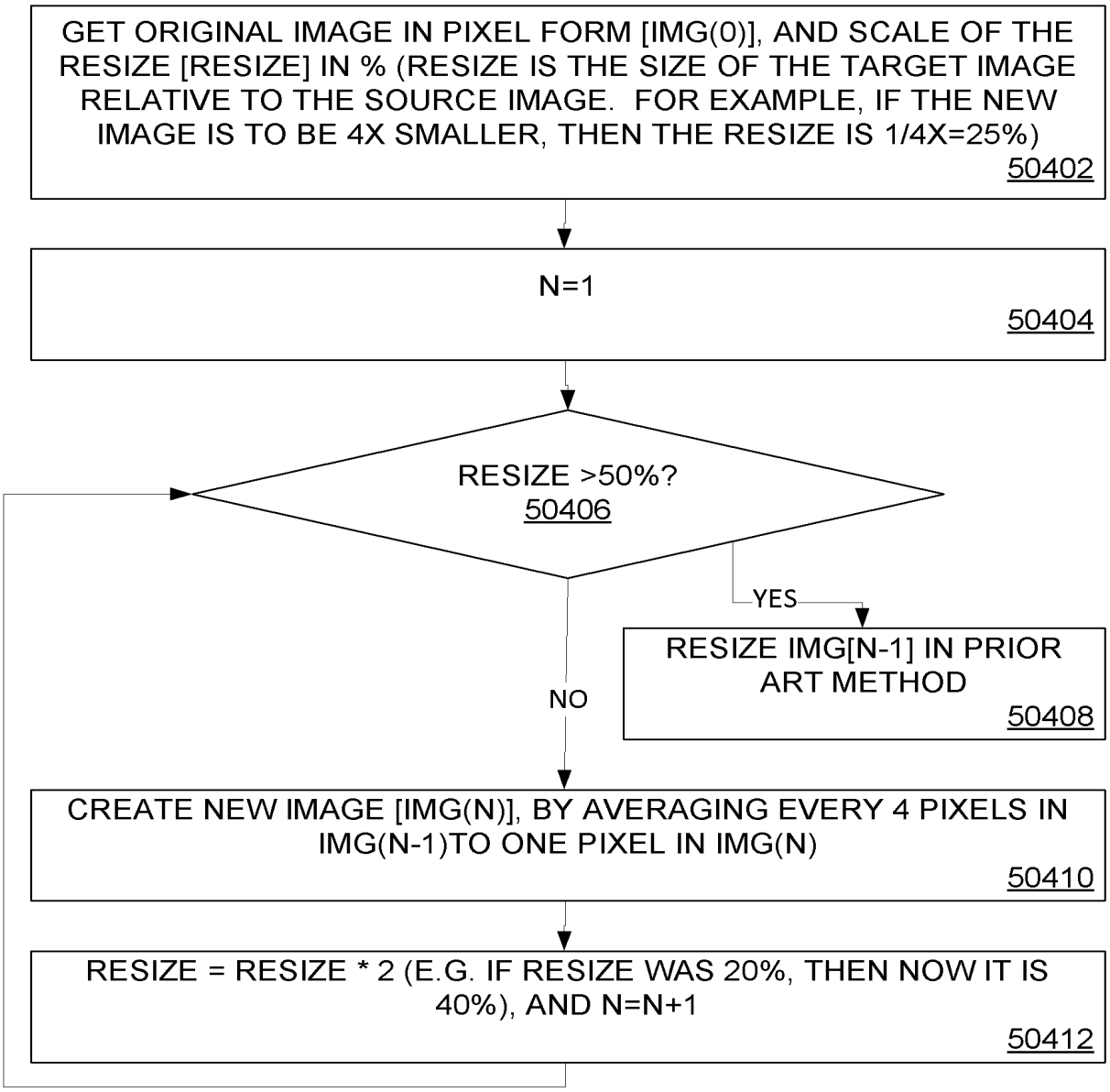
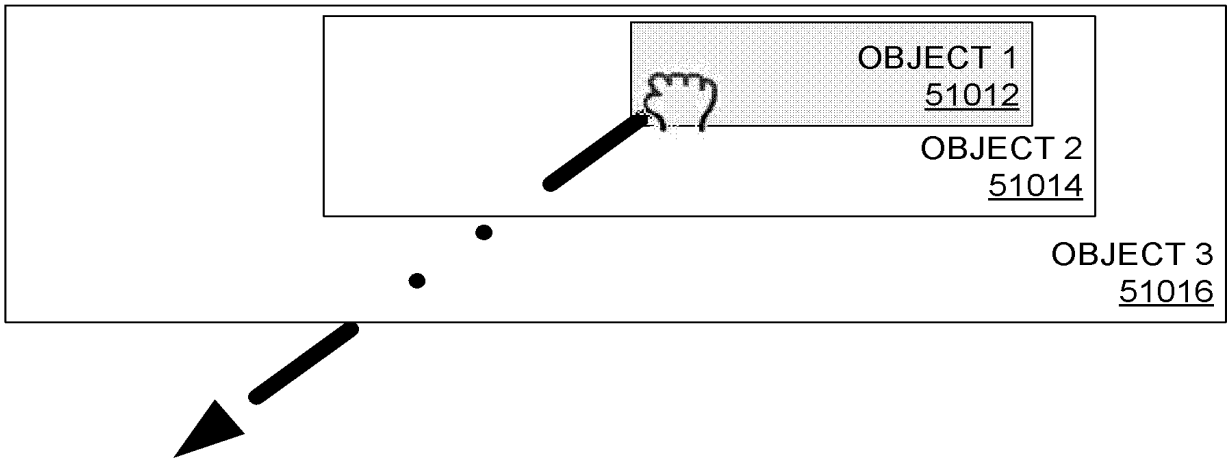


FIG. 510 – OBJECT MOVEMENT HEREDITY – PRIOR ART - DIAGRAM

START OF DRAG
51010



END OF DRAG
51020

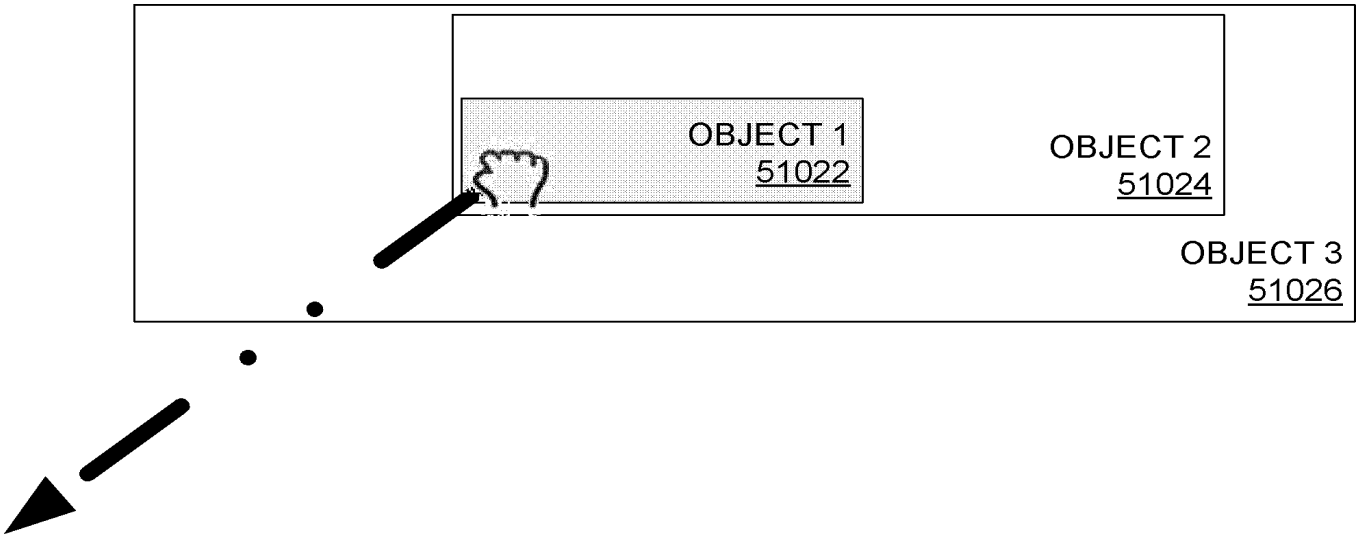
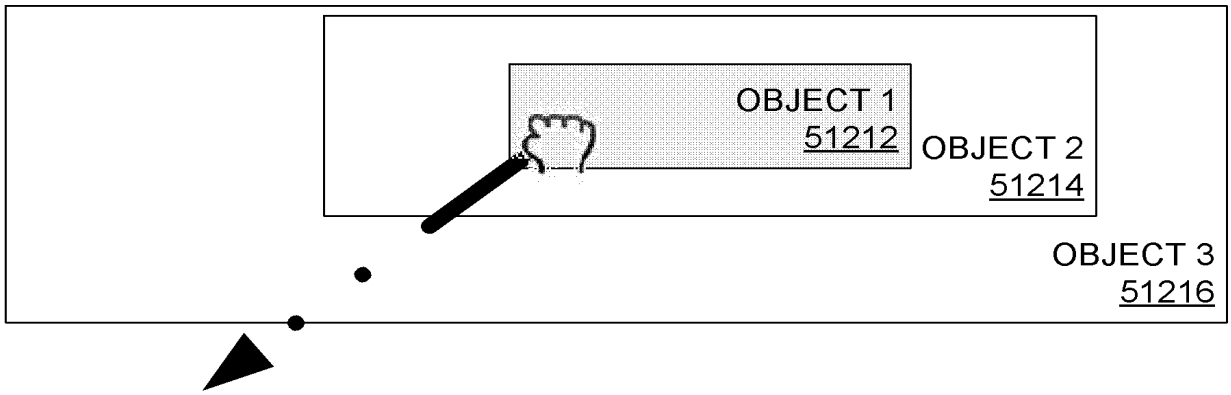


FIG. 512 – OBJECT MOVEMENT HEREDITY – DIAGRAM

START OF DRAG
51210



END OF DRAG
51220

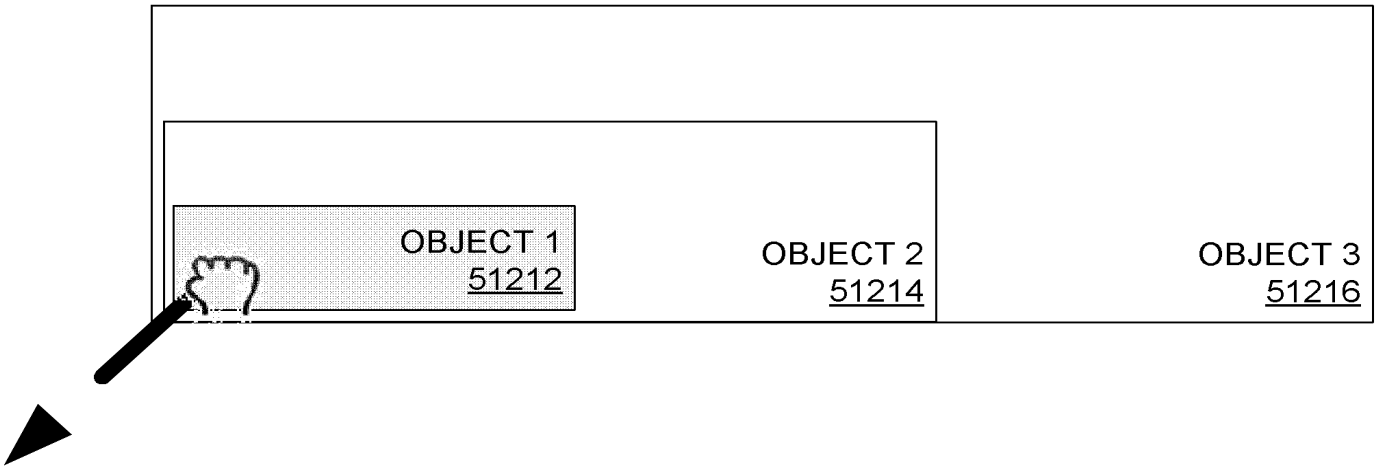


FIG. 514 – OBJECT MOVEMENT HEREDITY – FLOWCHART

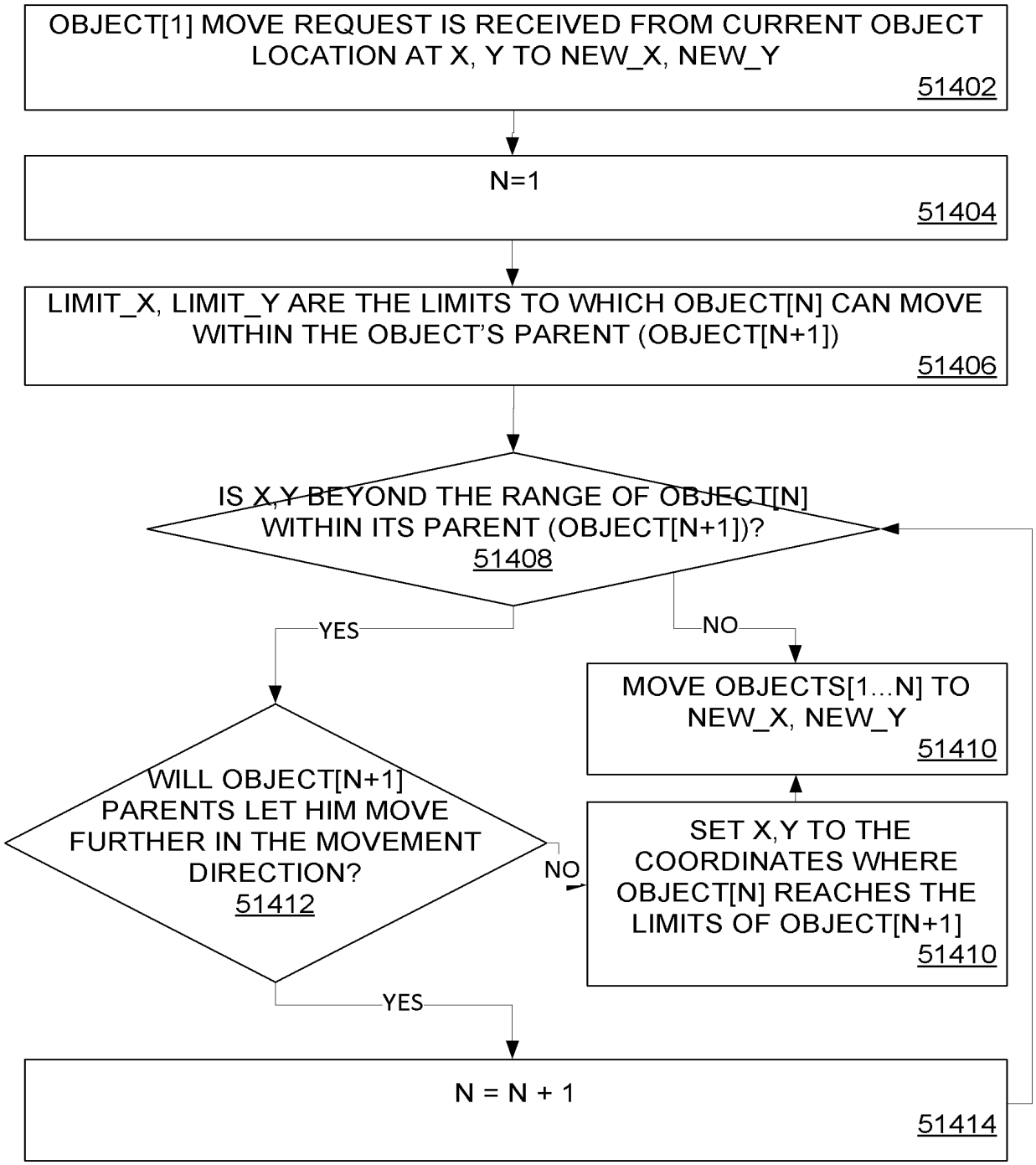
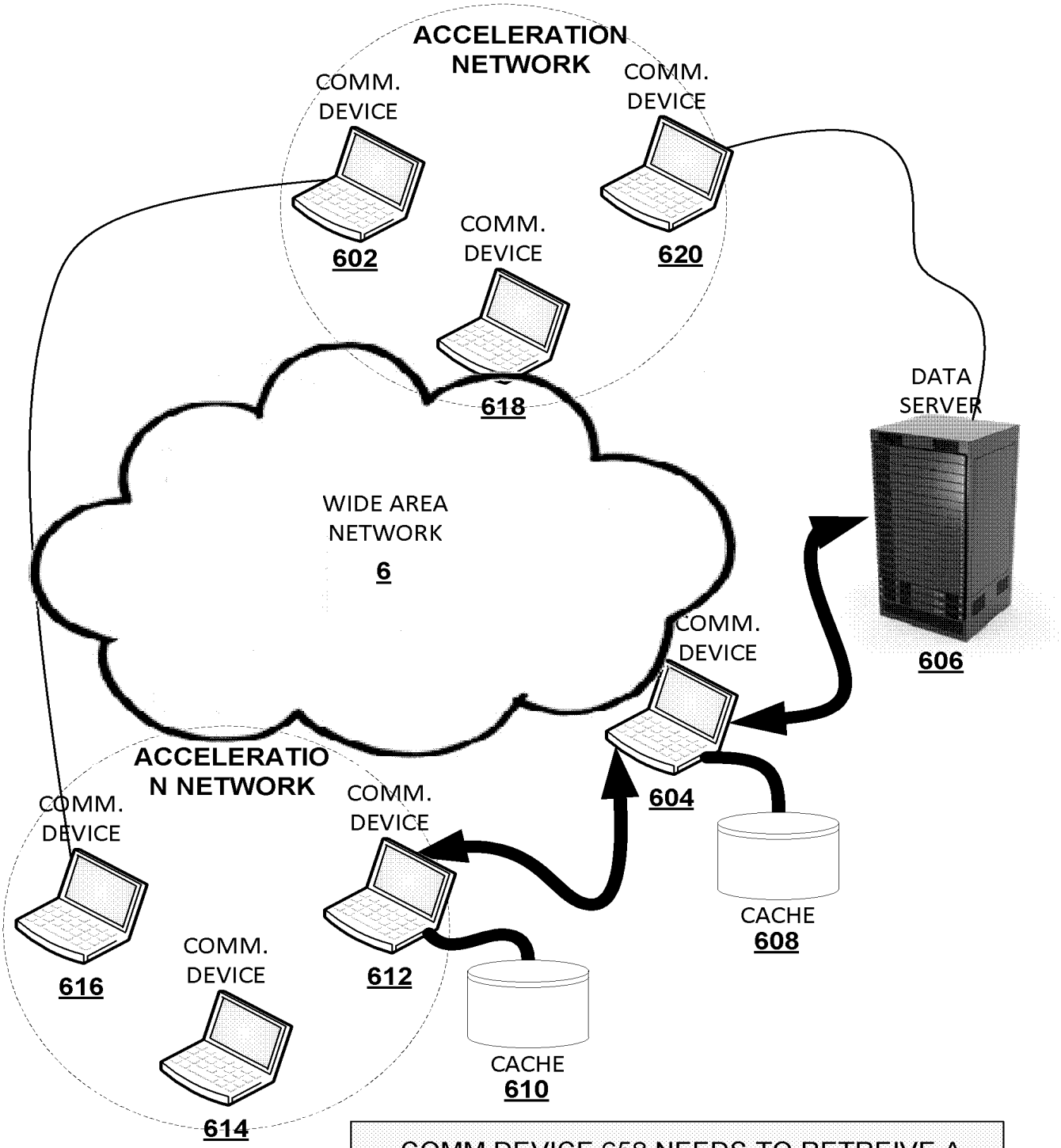


FIG. 600 – SAMENESS BETWEEN URLS FOR CACHING PURPOSES - DIAGRAM



- COMM DEVICE 658 NEEDS TO RETREIVE A URL, FOR WHICH IT HAS A CACHED ENTRY
- SO NEEDS TO FIGURE OUT WHETHER THE STORED DATA IS THE SAME AS THE DATA ON THE DATA SERVER, OR ON A COMM. DEVICE ON THE NETWORK THAT CLAIMS TO HAVE THE DATA CACHED

FIG. 602 – SAMENESS BETWEEN URLS FOR CACHING PURPOSES - FLOWCHART

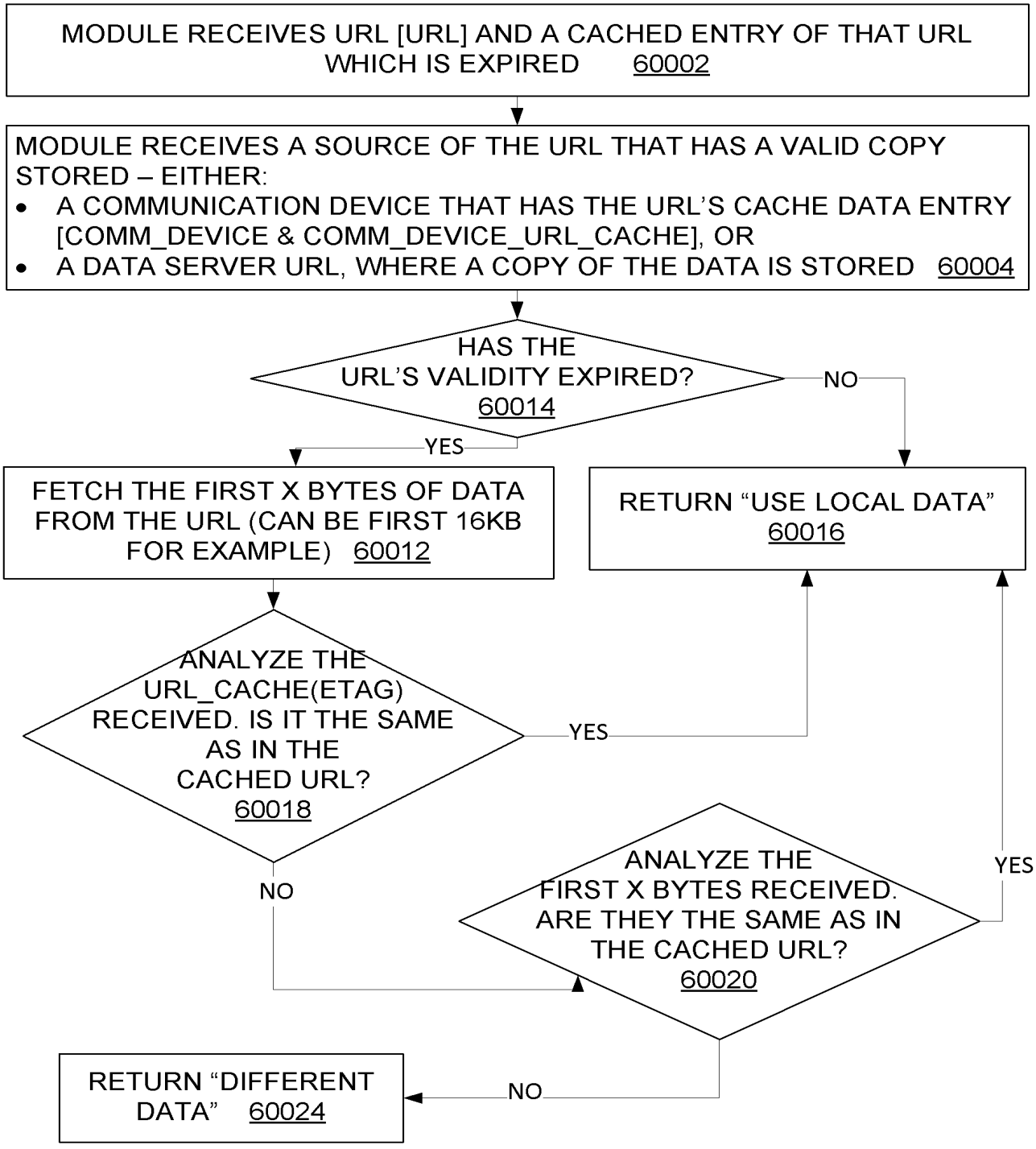


FIG. 610 – REVALIDATING CACHE ENTRIES

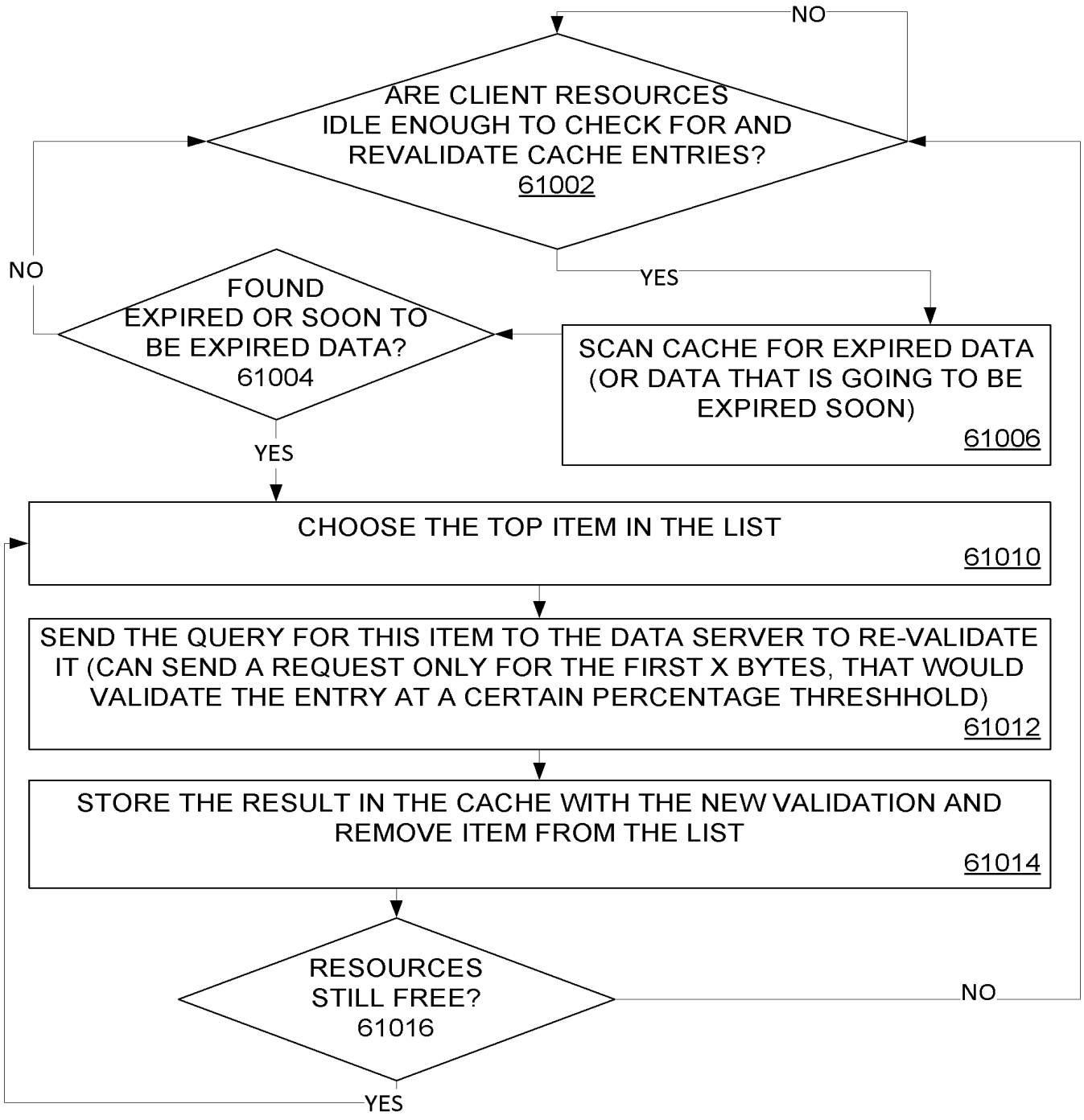


FIG. 620 – NDFS CACHE OVERWRITE REDUCTION AND NDFS CACHE CLEANUP – PRIOR ART

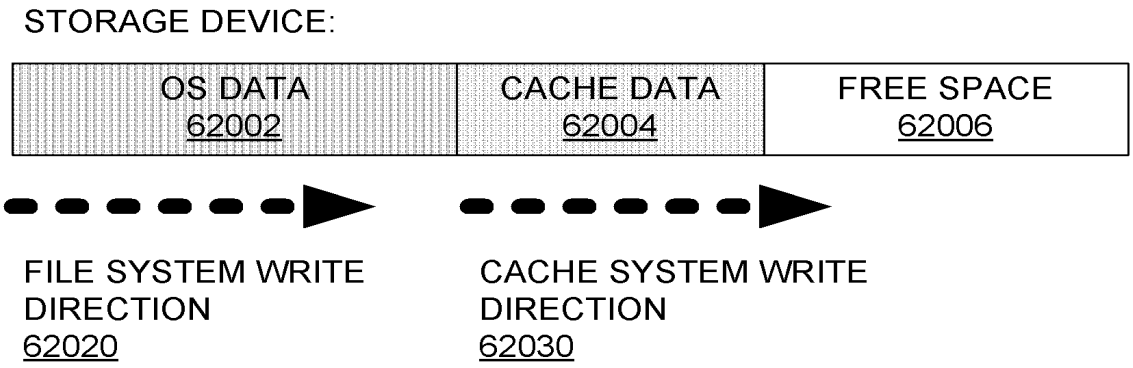


FIG. 622 – NDFS CACHE OVERWRITE REDUCTION AND NDFS CACHE CLEANUP – DIAGRAM

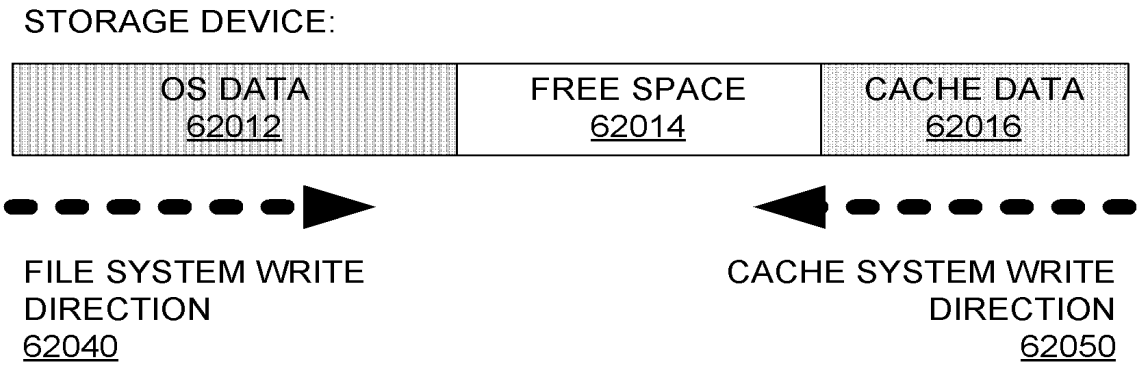


FIG. 622 – NDFS CACHE OVERWRITE REDUCTION AND NDFS CACHE CLEANUP – FLOWCHART

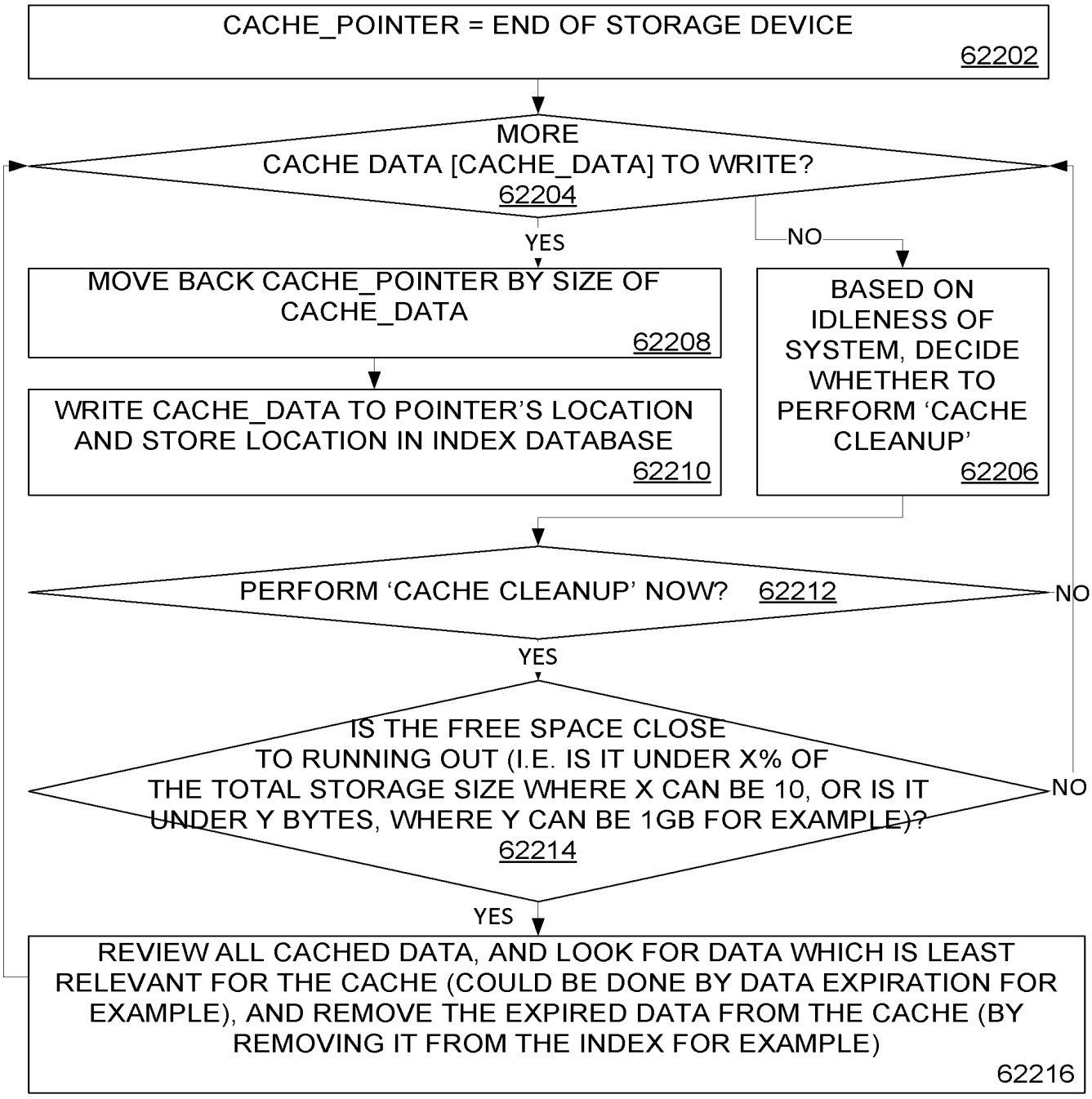


FIG. 630 – NDFS CACHE OVERWRITE REDUCTION – DIAGRAM

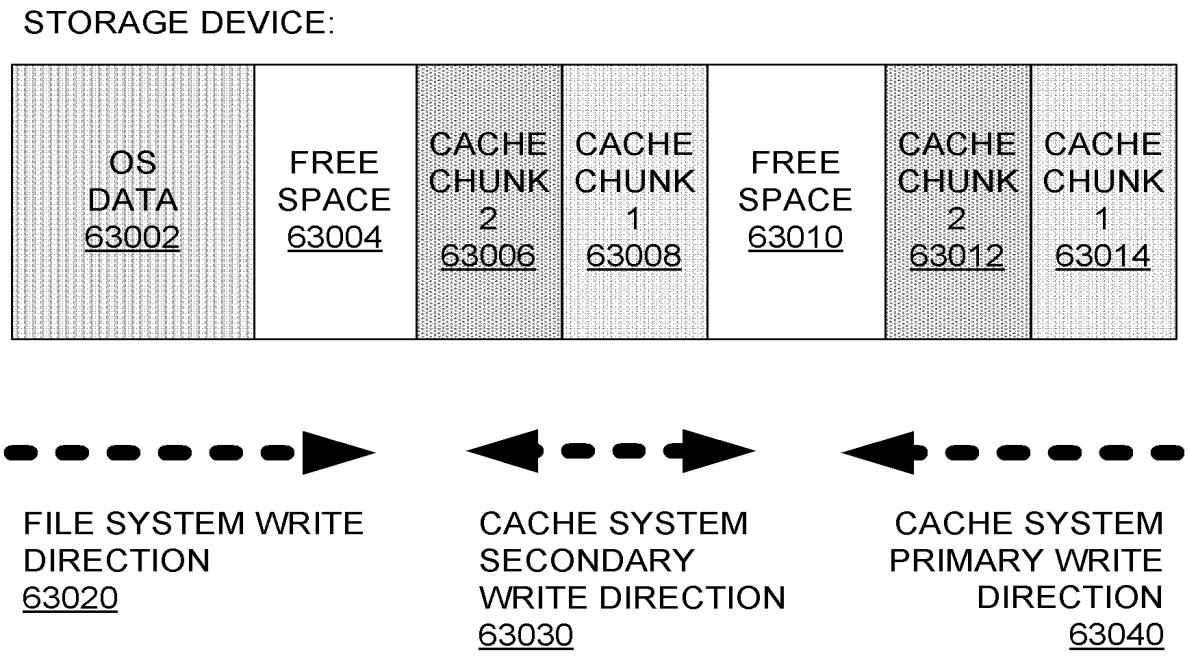


FIG. 632 – NDFS CACHE OVERWRITE REDUCTION – FLOWCHART

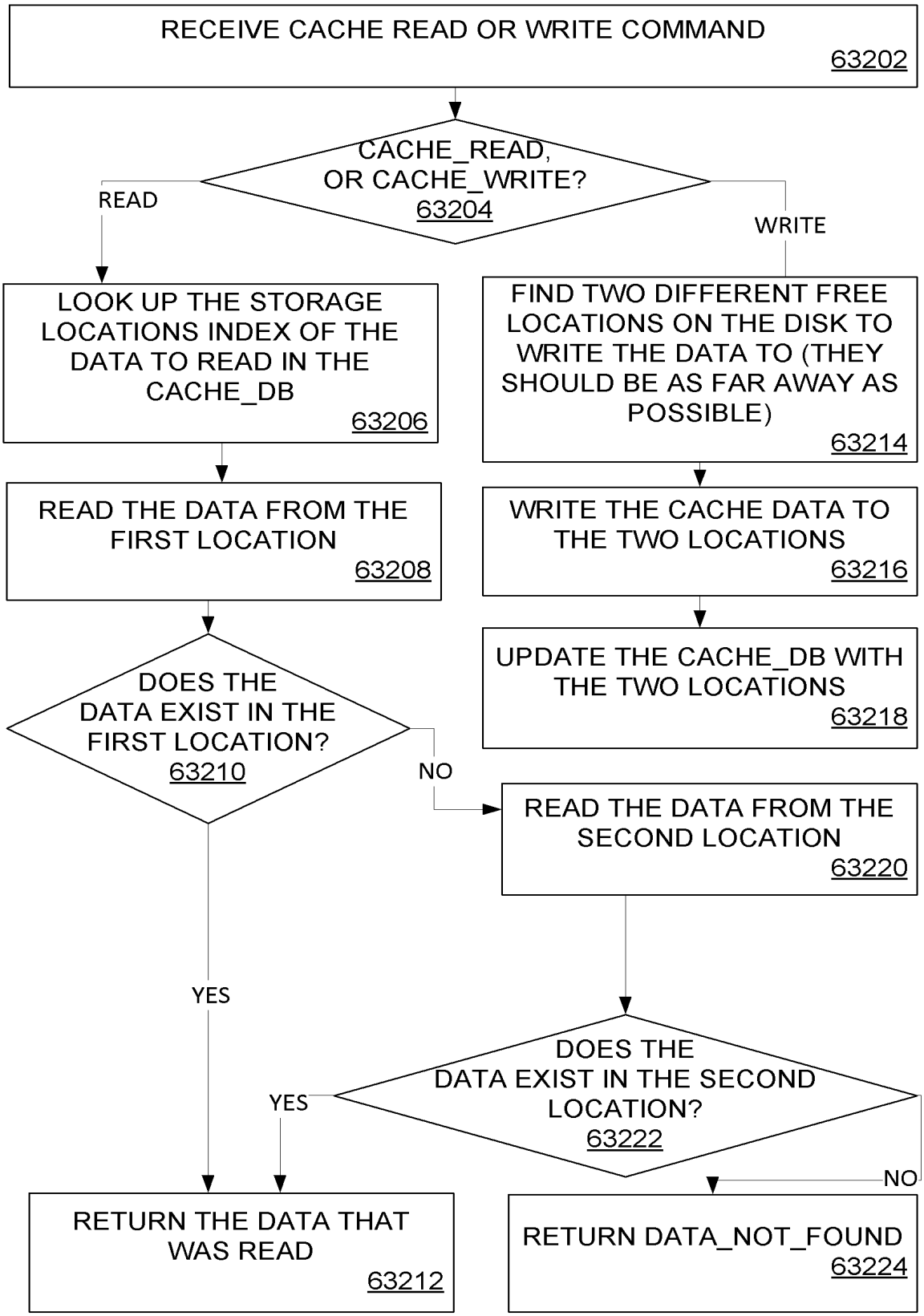


FIG. 634 – NDFS CACHE OVERWRITE REDUCTION – NUMBER OF CHUNK COPIES TO STORE

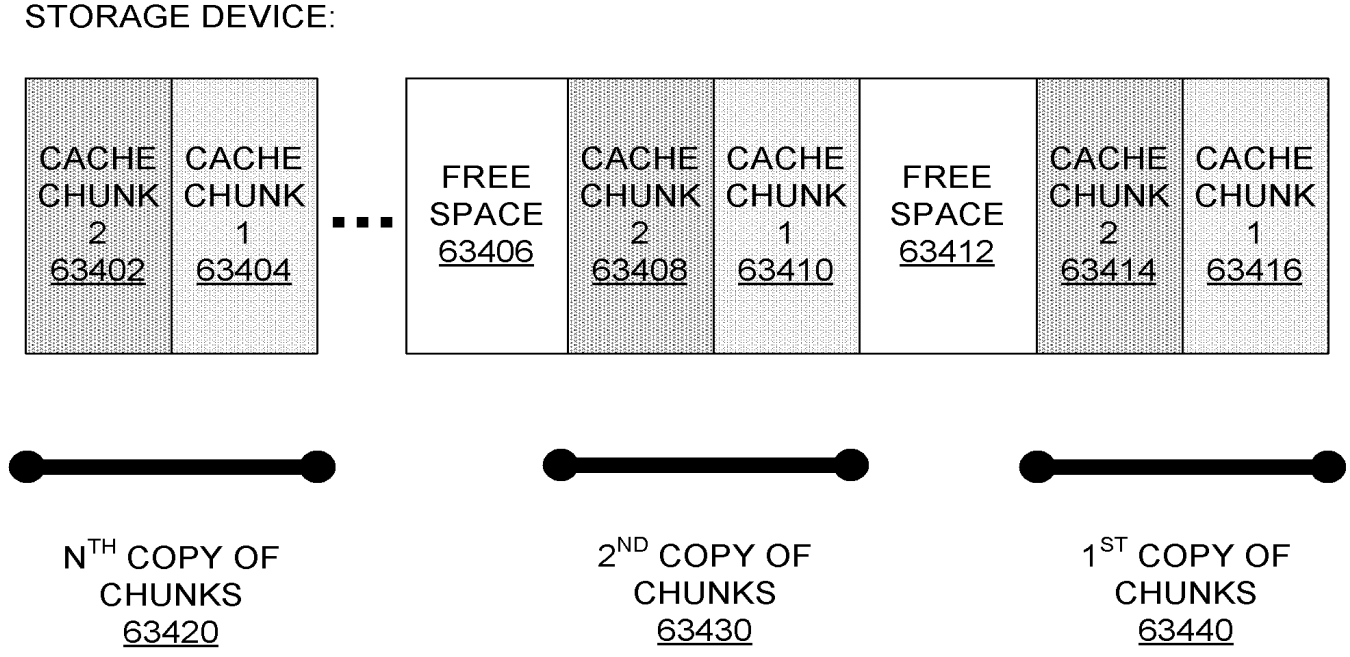


FIG. 640 – NDCACHE – PRIOR ART PHYSICAL ADDRESS VS. VIRTUAL ADDRESS

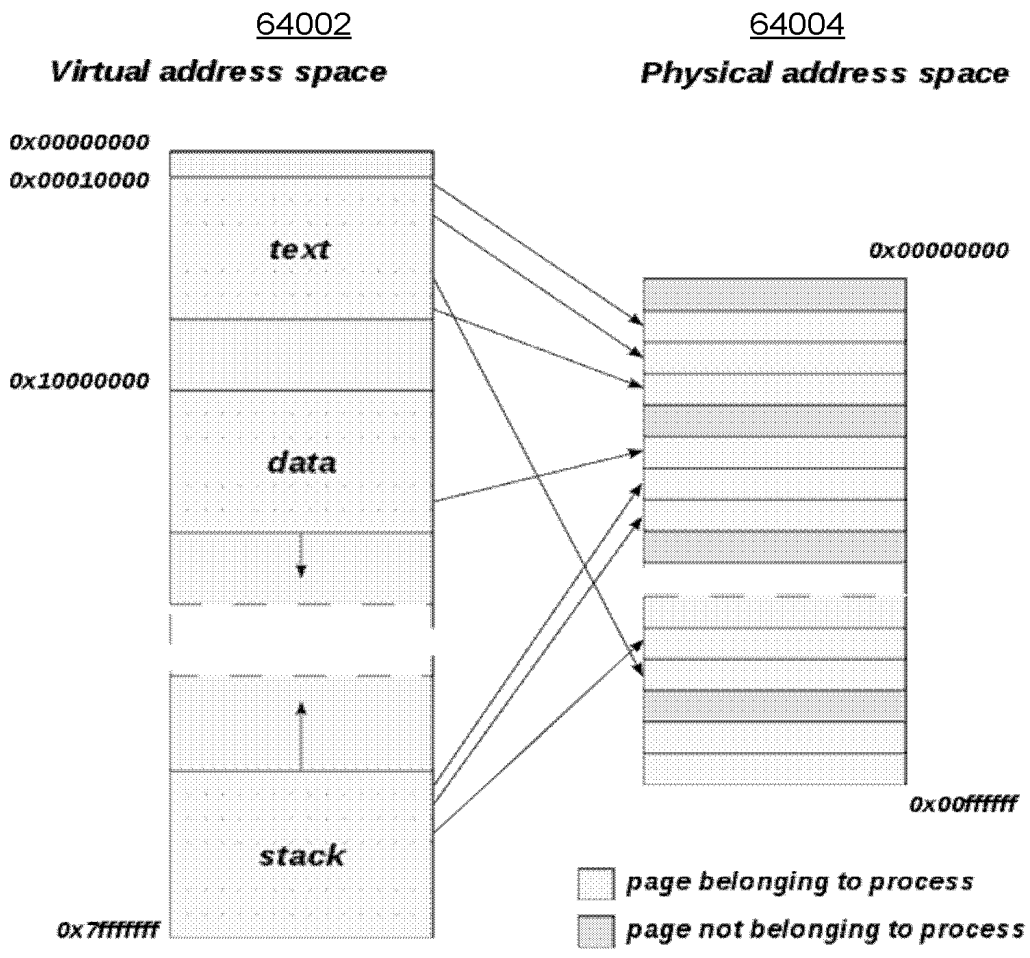


FIG. 642 – NDCACHE – MMU – MEMORY MANAGEMENT UNIT – TRANSLATION BETWEEN VIRTUAL ADDRESS AND PHYSICAL ADDRESS

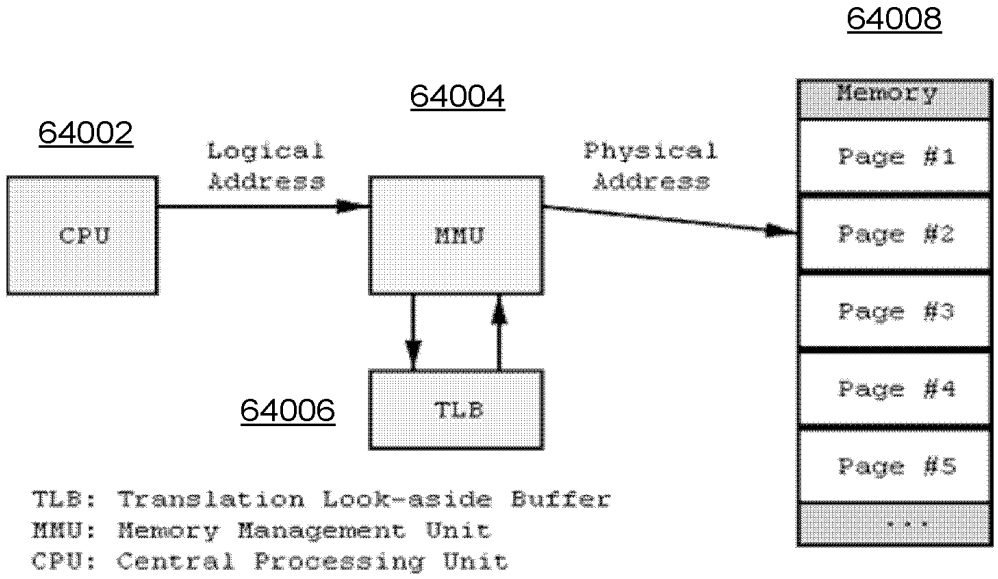


FIG. 644 – NDCACHE – PRIOR ART – MMU OPERATION

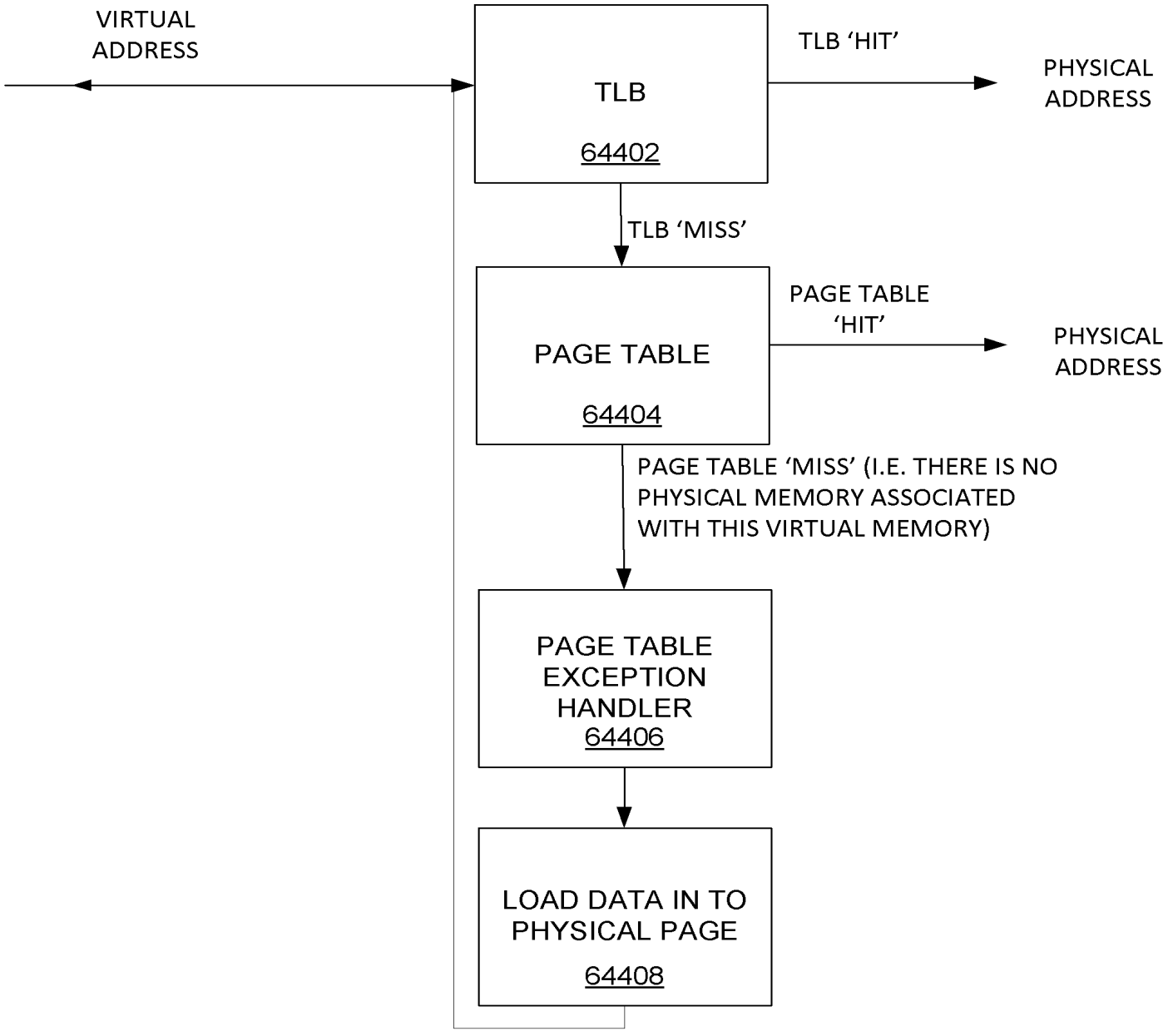


FIG. 648 – NDCACHE – PRIOR ART – EXCEPTION HANDLER

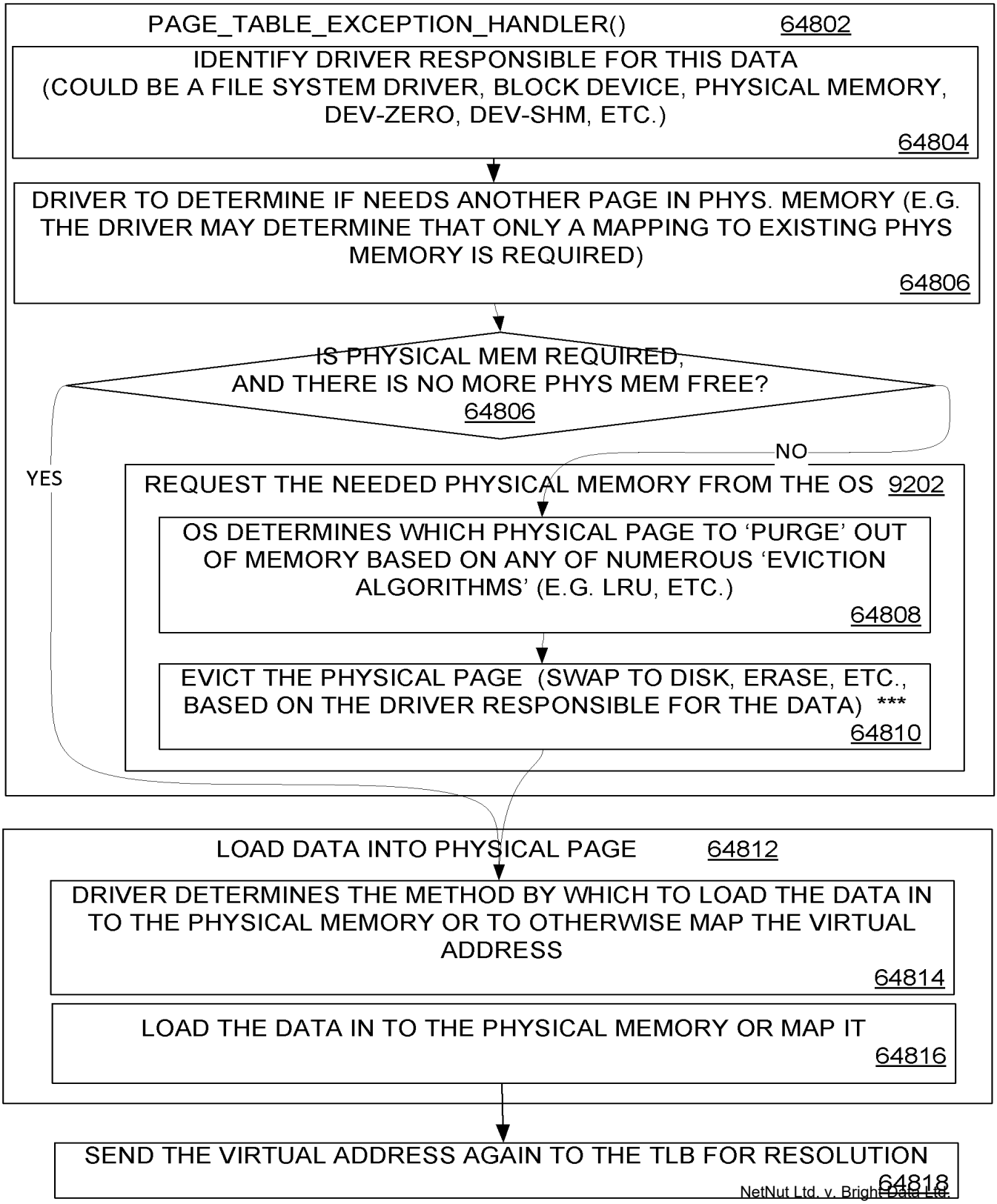


FIG. 650 – NDCACHE FLOWCHART

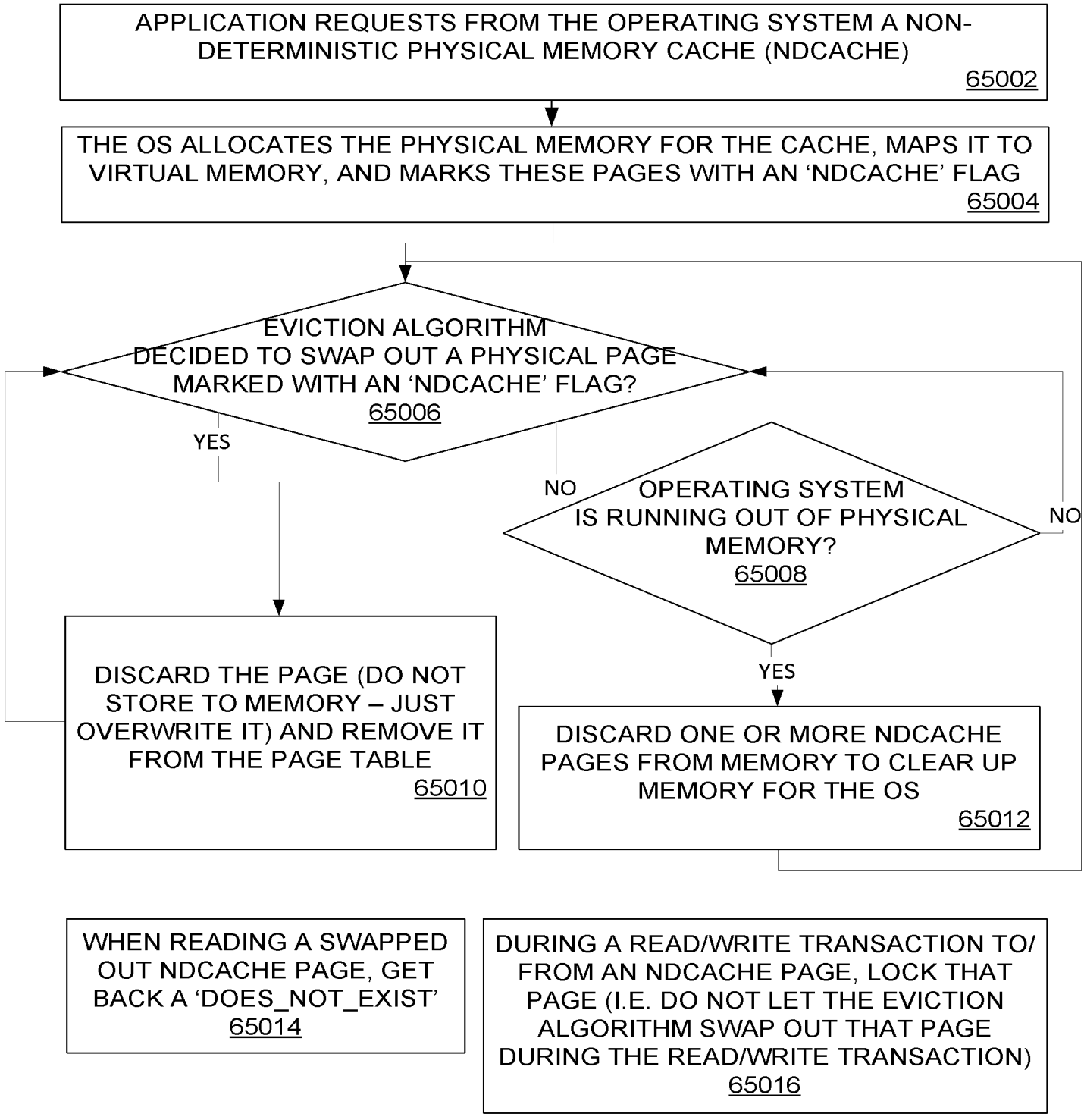


FIG. 654 - NDCACHE – API

API (SIMILAR TO THE FILE API IN POSIX)

FILEDS NDCACHE_OPEN("NAME", CREATE_FLAG, TRUNCATE_FLAG)

- DESCRIPTION: OPEN AN NDCACHE CALLED 'NAME' FOR READING OR WRITING AND RECEIVE A HANDLE [FILEDS] TO IT.
- NAME - THE NAME OF THE NDCACHE TO OPEN OR CREATE. THIS NAME CAN BE USED BETWEEN DIFFERENT PROCESSES ACCESSING THIS PAGE, SIMILAR TO HOW A FILENAME IS USED
- CREATE_FLAG - IS TURNED ON TO CREATE THE CACHE IF SUCH AN NDCACHE CALLED <NAME> DOES NOT EXIST
- TRUNCATE_FLAG - IS TURNED ON TO ERASE THE NDCACHE CALLED <NAME> IF IT ALREADY EXISTS
- RETURN VALUE - THE NDCACHE_OPEN RETURNS A FILE DESCRIPTOR (FILEDS) TO BE USED AS A HANDLE FOR THE OTHER NDCACHE APIS

65402

NDCACHE_READ(FILEDS, BUFFER, SIZE)
NDCACHE_WRITE(FILEDS, BUFFER, SIZE)

- DESCRIPTION: READ OR WRITE TO THE NDCACHE REFERENCED BY FILEDS
- FILEDS - THE FILE DESCRIPTOR OF THE OPEN NDCACHE TO READ FROM OR WRITE TO, RECEIVED FROM NDCACHE_OPEN
- BUFFER - [FOR READ] THE BUFFER TO READ THE INFORMATION IN TO, [FOR WRITE] … TO WRITE FROM
- OFFSET - THE OFFSET WITHIN THE NDCACHE TO START THE READ OR WRITE FROM
- SIZE - THE NUMBER OF BYTES TO READ OR WRITE

65404

NDCACHE_CLOSE(FILEDS)

- DESCRIPTION: CLOSES THE HANDLE TO THE NDCACHE DESCRIBED BY FILEDS. AFTER ALL HANDLES TO AN NDCACHE ARE CLOSED, THE OS MAY DECIDE TO SWAP IT OUT BASED ON ITS EVICTION ALGORITHM
- FILEDS - THE FILE DESCRIPTOR OF THE OPEN NDCACHE TO CLOSE

65406

NDCACHE_UNLINK(FILEDS)

- DESCRIPTION: REMOVE AN NDCACHE ENTRY FROM THE PAGE TABLE
- FILEDS - THE FILE DESCRIPTOR OF THE OPEN NDCACHE TO REMOVE FROM THE PAGE TABLE

65408

FIG. 662 – BACKGROUND - MOUNTING A FILESYSTEM

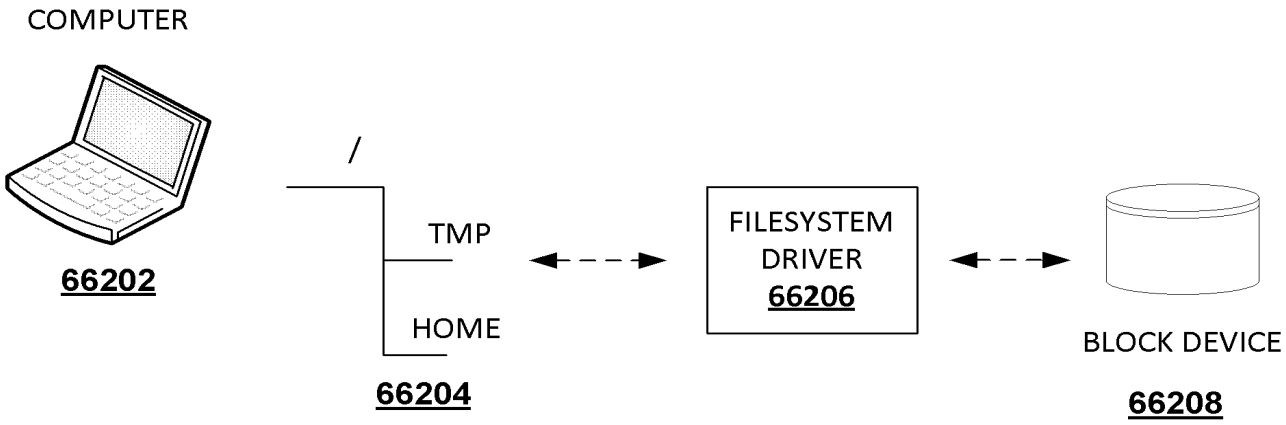


FIG. 664 - BACKGROUND – TMPFS FILE SYSTEM

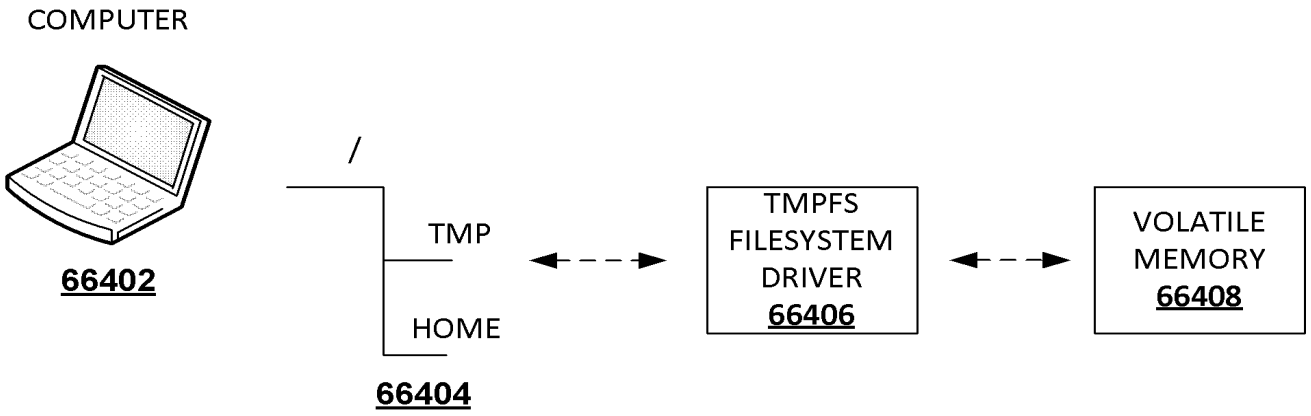


FIG. 666 – NDCACHE – SECONDARY IMPLEMENTATION METHOD USING FS MOUNTING - IMPLEMENTATION

CREATE AND MOUNT A NEW TMPFS WITH AN INITIAL SIZE OF X BYTES (COULD BE 1GB FOR EXAMPLE) TO BE USED FOR ND CACHE
66602

PERIODICALLY CHECK WHETHER THE OS WOULD BENEFIT FROM HAVING ADDITIONAL VOLATILE MEMORY 66604

IT IS POSSIBLE TO CHECK WHETHER OS NEEDS MORE MEMORY BY ONE OR MORE OF THE FOLLOWING METHODS:

1. PERIODIC FREE MEMORY CHECK - EVERY X MS (COULD BE EVERY 20MS) FLAG IF:
   A. LESS THAN A CERTAIN AMOUNT OF FREE MEMORY EXISTS IN THE SYSTEM (COULD BE LESS THAN 200MB)
   B. ABOVE A NUMBER OF MEMORY SWAPS OCCUR IN THE OS (COULD BE MORE THAN 1 SWAP PER 500MS)
   C. KERNEL CACHE SPACE REDUCED TO UNDER A CERTAIN SIZE (COULD BE UNDER 2GB)

2. LISTEN TO 'STRESS' OS CALLS, AND FLAG IF ANY OF THE STRESS CALLS HAVE BEEN INVOKED. THESE COULD BE FOR EXAMPLE "LOW MEMORY" TYPE CALL BACKS

3. ATTACH CALLBACK ON SWAPPING OUT OF THE TMPFS FILE, AND DELETING IT INSTEAD OF SWAPPING OUT

66606

WOULD THE OS BENEFIT FROM MORE VOLATILE MEMORY?
66608

YES

NO

DELETE A NUMBER OF ELEMENTS FROM THE NDCACHE SO THAT A SET CHUNK OF IT WILL BE FREE (CHUNK COULD BE 200MB FOR EXAMPLE)
66610

IT IS POSSIBLE TO DELETE ELEMENTS FROM THE NDCACHE (TMPFS) BY USING ONE OR MORE OF THE FOLLOWING METHODS:

1. LRU – DELETE THE LEAST RECENTLY USED (IN TERMS OF READ/WRITE) ELEMENTS FROM THE TMPFS
2. LFU – DELETE THE LEAST FREQUENTLY USED (IN TERMS OF READ/WRITE) ELEMENTS FROM THE TMPFS
3. SIZE - DELETE THE LARGEST ELEMENTS (BYTES) FROM THE TMPFS
4. SIZE - DELETE THE SMALLEST ELEMENTS (BYTES) FROM THE TMPFS

66612

FIG. 672 – NDCACHE – THIRD IMPLEMENTATION METHOD - USER & KERNEL MODE

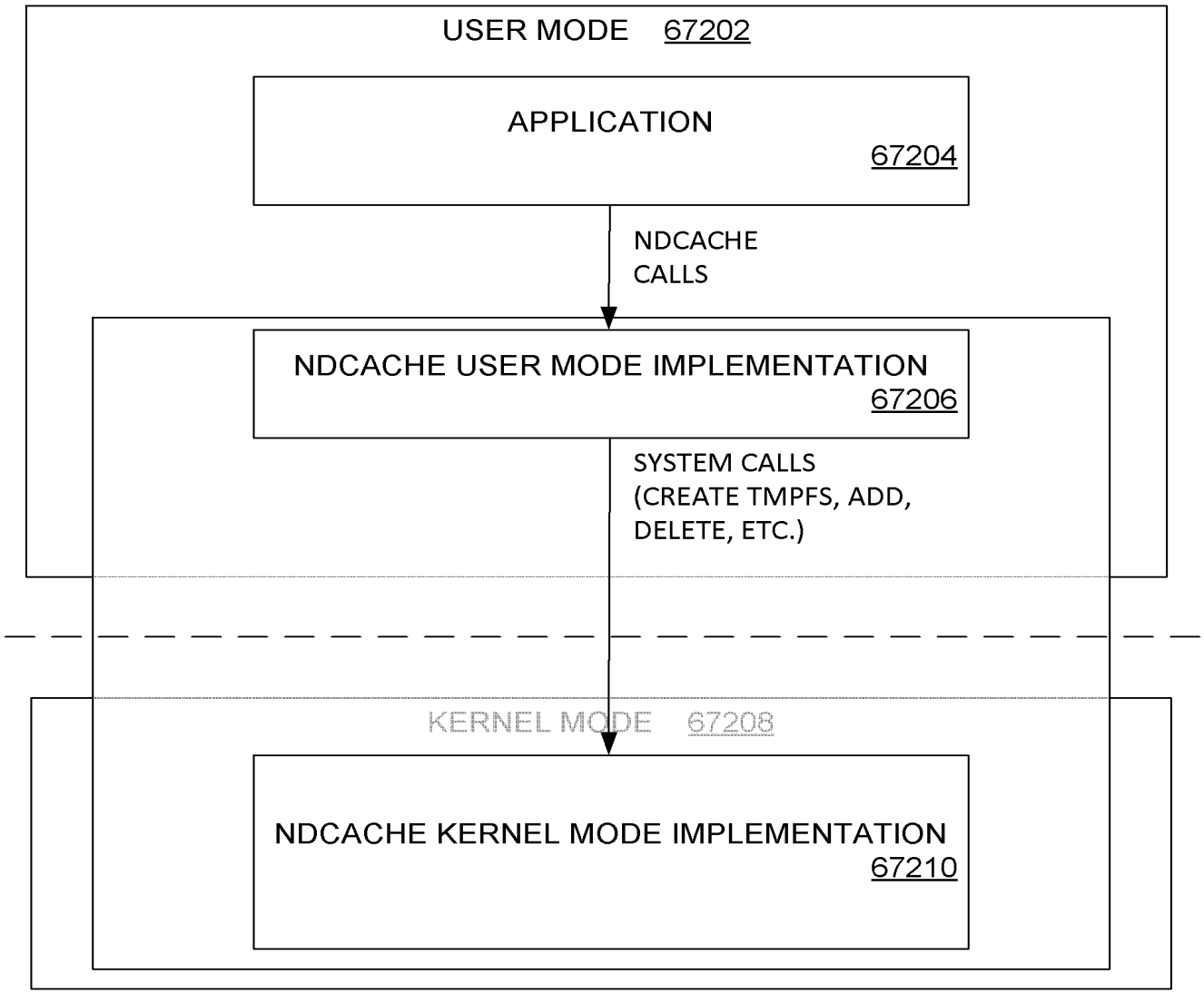


FIG. 674 – NDCACHE – THIRD IMPLEMENTATION METHOD – HIGH LEVEL IMPLEMENTATION

WRITING TO (AND ALLOCATING) THE NDCACHE:

```
FILEDS *FILEDS=OPEN("NDCACHE_DRIVER/NDCACHE_NAME");
/* NDCACHE_DRIVER IS THE KERNEL MODULE FOR NDCACHE EVENTS
   NDCACHE_NAME IS THE NAME OF THIS NDCACHE, TO BE SHARED
   BETWEEN DIFFERENT PROCESSES */

INT *P=NDCACHE_LOOKUP(NDCACHE_NAME);  // GET PTR TO THE NDCACHE

IF (!P)
{
    // NO SUCH NDCACHE EXISTS, AND THUS NEEDS TO BE CREATED

   INT *P = MMAP(1MB, SHARED MEMORY, FILEDES);
   // 1MB AS AN EXMAPLE NDCACHE MEM SIZE REQUESTED
   // SHARED MEMORY SO THAT CAN BE USED BY MULTIPLE PROCESSES

   (INT* P)[0]++;   // INDICATES THAT THERE IS CONTENT IN THE NDCACHE
   P[N]=...;   // WRITE THE NEW CONTENT TO THE NDCACHE

}
ELSE
   DO NDCACHE_WRITE();  // THIS NDCACHE EXISTS, WRITE TO IT
```

67402

FIG. 675 – NDCACHE PAGES AND LOCK FLAG

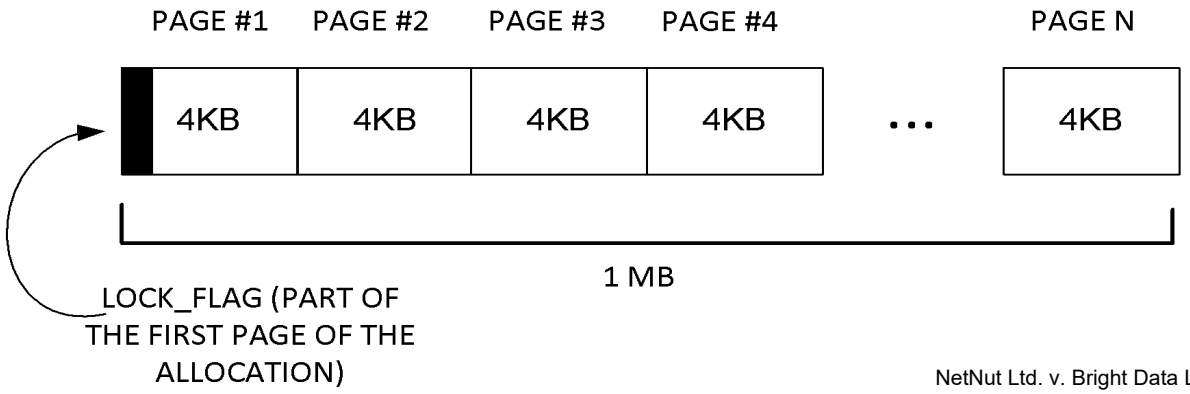


FIG. 676 – NDCACHE – THIRD IMPLEMENTATION METHOD– HIGH LEVEL IMPLEMENTATION (CONT.)

NDCACHE_READ() / NDCACHE_WRITE():

```
// CHECK IF THE NDCACHE STILL EXISTS (IF IT WAS SWAPPED OUT BY THE
OS, IT NO LONGER EXISTS)
IF (!P[0]) RETURN MEM_NOT_FOUND;

// COULD BE THAT AT THIS POINT THE OS SWAPPED OUT THIS NDCACH, SO
// CHECK IF THE NDCACHE STILL EXISTS, AND LOCK IT FOR THE DURATION
// OF THE READ/WRITE
IF ((++P[0])==1)
{
  P[0]--;
  RETURN MEM_NOT_FOUND;
}

// READ/WRITE CONTENT FROM/TO THE NDCACHE
BUF = P[N];    // FOR READ
OR
P[N] = …;   // FOR WRITE

// DECREMENT THE LOCK_FLAG BY ONE (LOCK_FLAG--), INDICATING THAT
THE NDCACHE LOCK FOR THIS READ/WRITE IS RELEASED
P[0]--;

RETURN;
```

67602

FIG. 678 – NDCACHE – THIRD IMPLEMENTATION METHOD – HIGH LEVEL IMPLEMENTATION (CONT.)

NDCACHE KERNEL MODULE (NDCACHE_DRIVER):

EVICTION ALGORITHM – WHEN THE OS NEEDS MORE PHYSICAL MEMORY AND IS CONSIDERING WHICH PHYSICAL MEMORY TO EVICT:

FOR EACH OF THE MEMORY RANGES HANDLED BY THE NDCACHE DRIVER, WHERE LOCK_FLAG=MEM_RANGE[0]:

IF LOCK_FLAG == 0 // MEM RANGE NOT IN USE
THEN SWAP OUT THIS MEMORY RANGE

IF LOCK_FLAG == 1 // NDCACHE MEM RANGE IS IN USE, BUT NOT LOCKED
THEN PREFER NOT TO SWAP OUT THIS MEMORY RANGE

IF LOCK_FLAG > 1 // NDCACHE MEM RANGE IS IN USE AND LOCKED
DON'T SWAP OUT THIS MEMORY RANGE

67802

ON SWAP OF THE NDCACHE MEMORY (I.E. WHEN THE OS'S EVICTION ALGORITHM DECIDES TO SWAP OUT THE NDCACHE'S PHYSICAL MEMORY ALLOCATION):

MAP THE VIRTUAL MEMORY POINTERS TO AN 'ALL ZERO' RANGE IN READ ONLY MODE (FOR EXAMPLE BY MAPPING TO DEVZERO)

67804

FIG. 680 – NDCACHE – FOURTH METHOD – MULTIPLE SEGMENTS IN CACHE

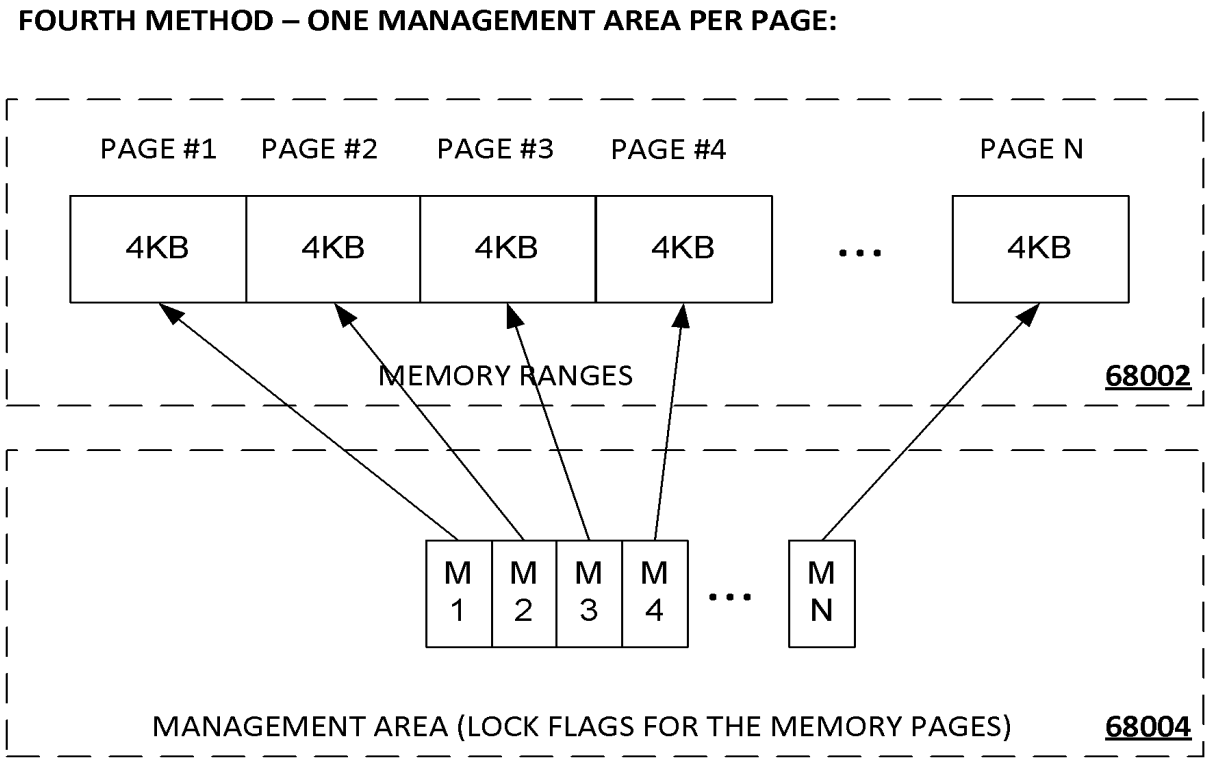


FIG. 682 – NDCACHE – FOURTH METHOD – FLOW CHART

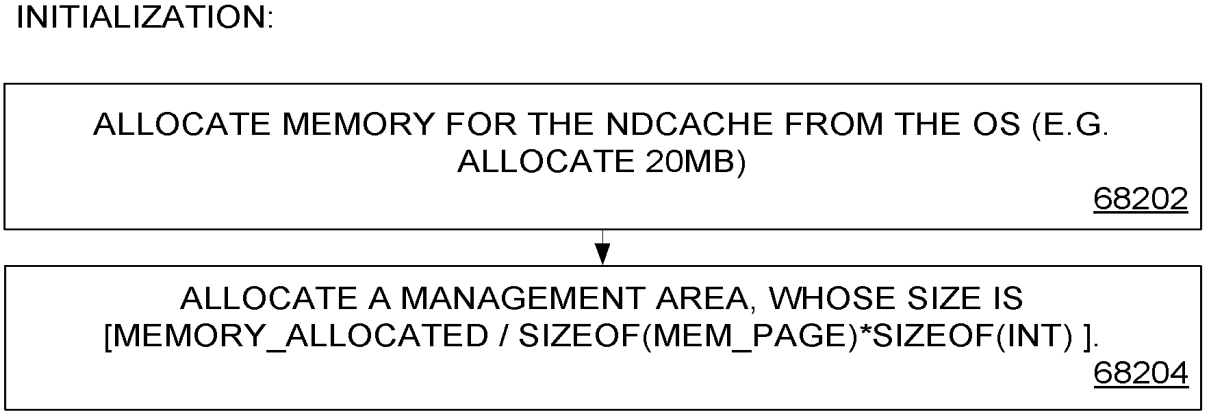


FIG. 690 – NDCACHE – IMPROVEMENT ON THE FOURTH METHOD – MEMORY LAYOUT

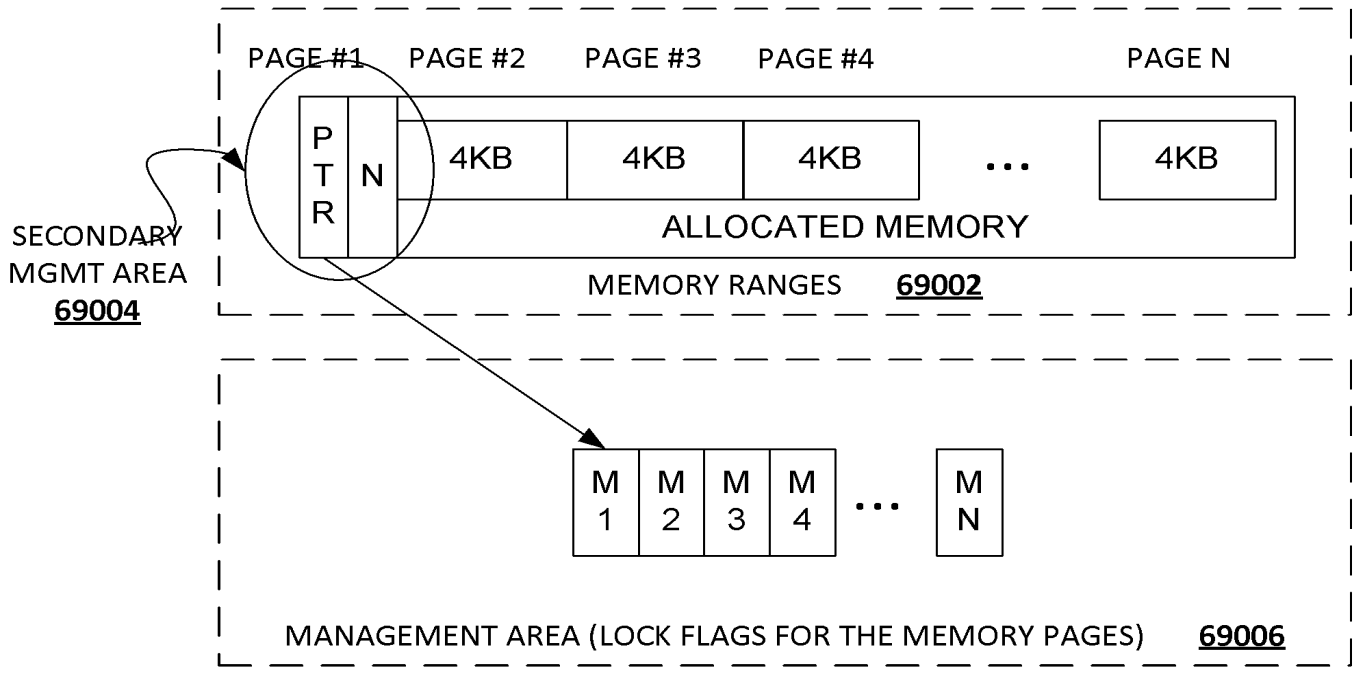


FIG. 692 – NDCACHE – IMPROVEMENT ON THE FOURTH METHOD – FLOWCHART

NEW NDCACHE ELEMENT ALLOCATION: P = NDCACHE_CREATE(SIZE):

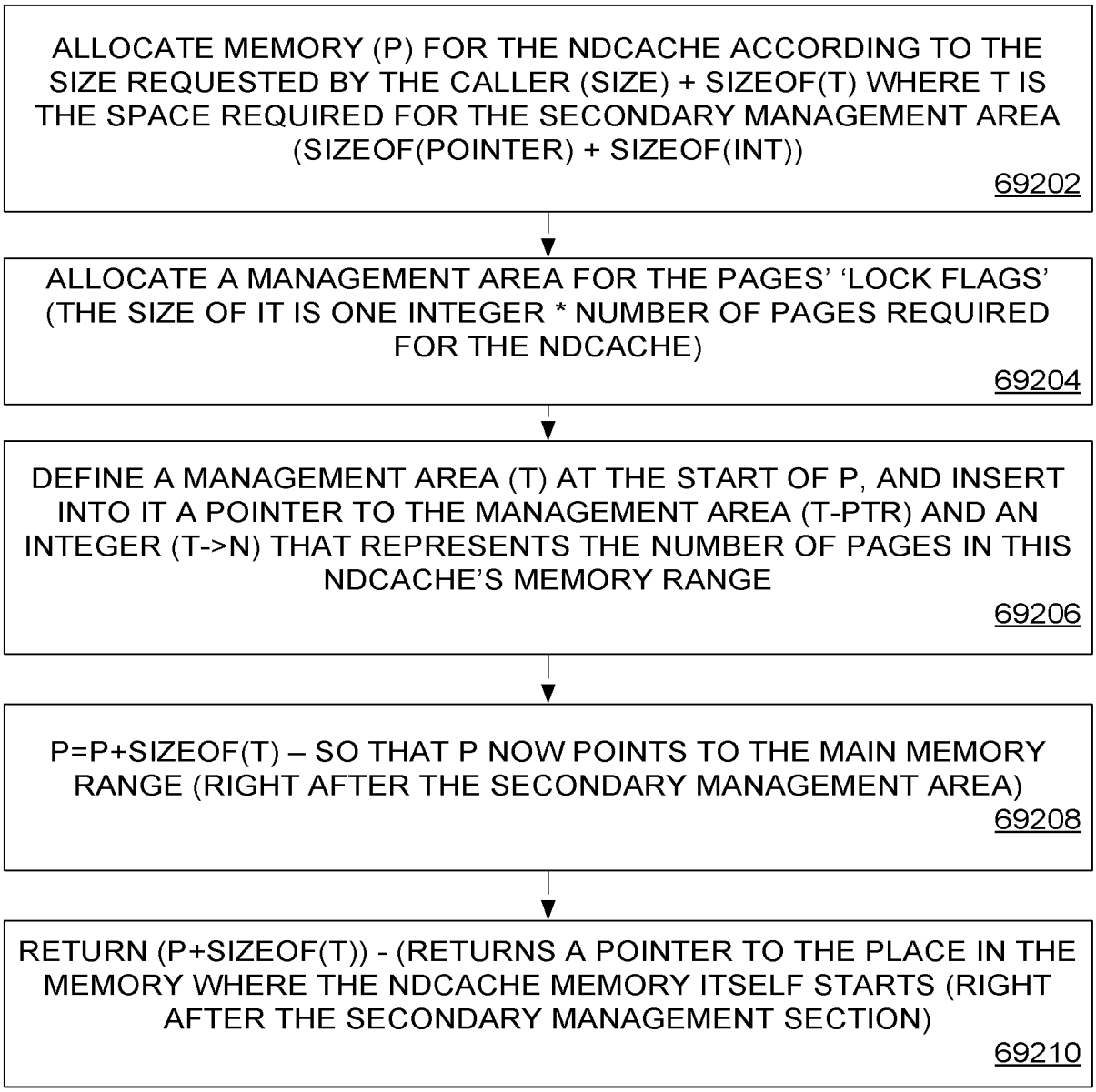


FIG. 696 – NDCACHE – IMPROVEMENT ON THE FOURTH METHOD – FLOWCHART (CONT.)

READING FROM OR WRITING TO THE NDCACHE ELEMENT:

USE THE SAME ALGORITHM AS IN THE FOURTH IMPLEMENTATION, WITH TWO EXCEPTIONS:

1. LOCK(P) / UNLOCK(P) –
  A. FIRST EVALUATES THAT T->N IS NOT ZERO. IF IT IS, THEN THE NDCACHE WAS SWAPPED AND NEEDS TO RETURN MEM_NOT_FOUND
  B. IF T->N IS NOT ZERO, LOCK ALL PAGES THAT THIS NDCACHE ELEMENT SPANS BY LOCKING THE N LOCK_FLAGS STARTING AT THE T->PTR LOCK_FLAG

2. WHEN READING OR WRITING, NOT LIMITED TO ONE PAGE OF MEMORY (CAN USE THE ALLOCATED SIZE)

69602

DELETING THE ALLOCATION: NDCACHE_CLOSE (P)

FREE (P - SIZEOF(T)) – FREES THE SECONDARY MANAGEMENT AREA AS WELL AS THE MAIN MEMORY AREA ALLOCATED TO THIS NDCACHE ELEMENT

69606

FIG. 700 – IDLE MONITOR

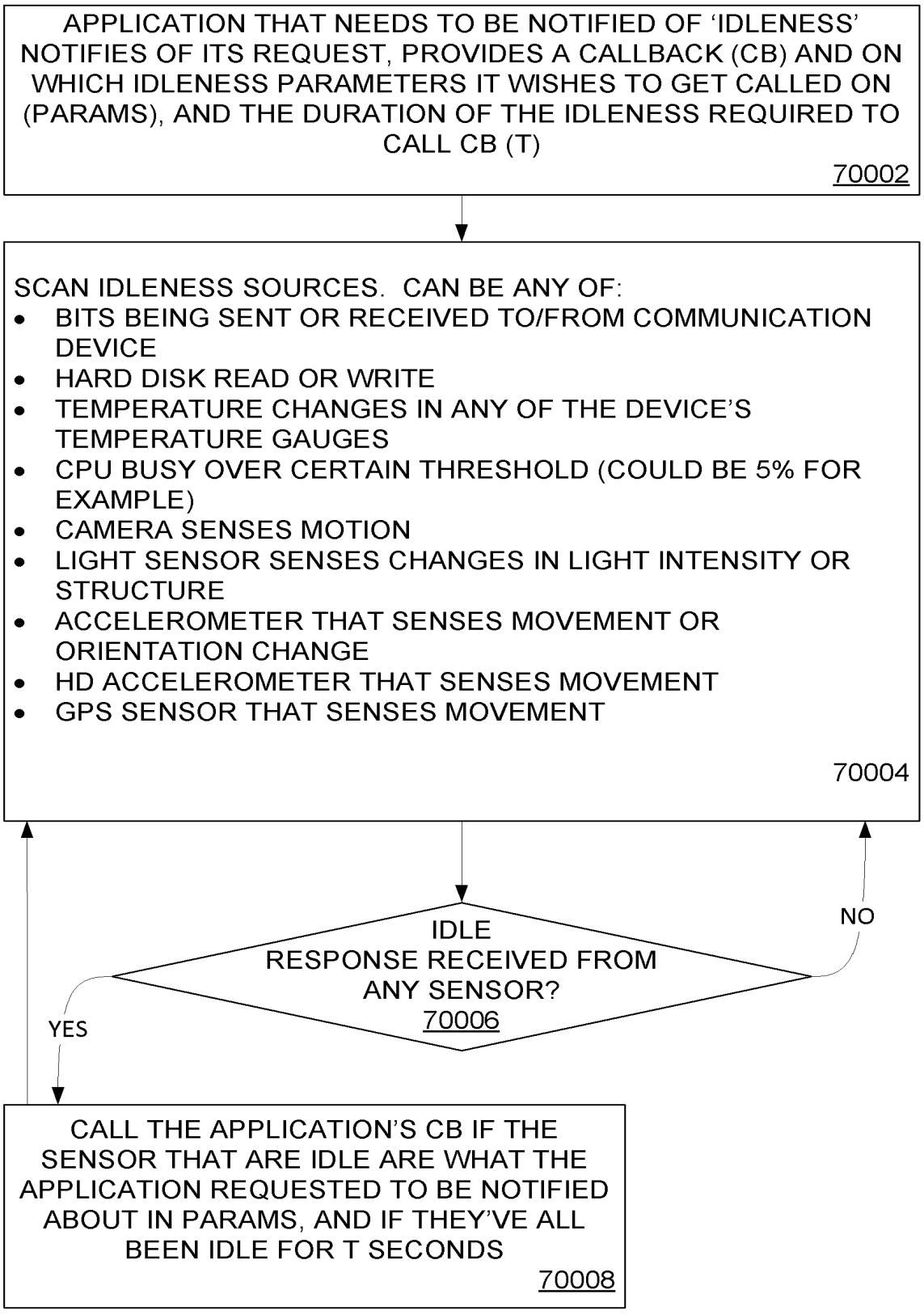


FIG. 710 – STORAGE WRITE ON IDLE; REDUCING STORAGE READ TIMES - DIAGRAM

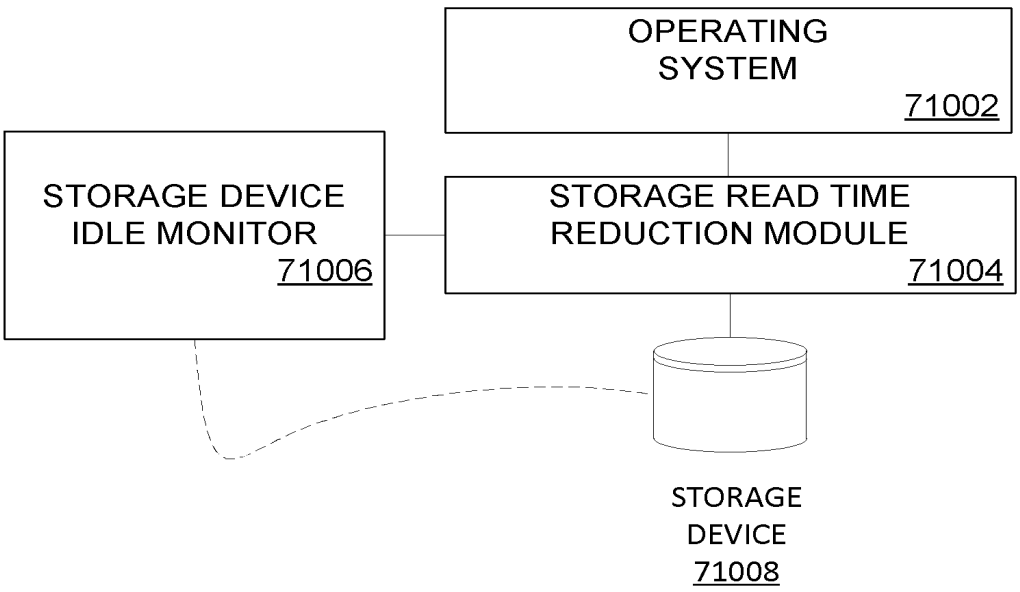


FIG. 712 – STORAGE WRITE ON IDLE; REDUCING STORAGE READ TIMES - FLOWCHART

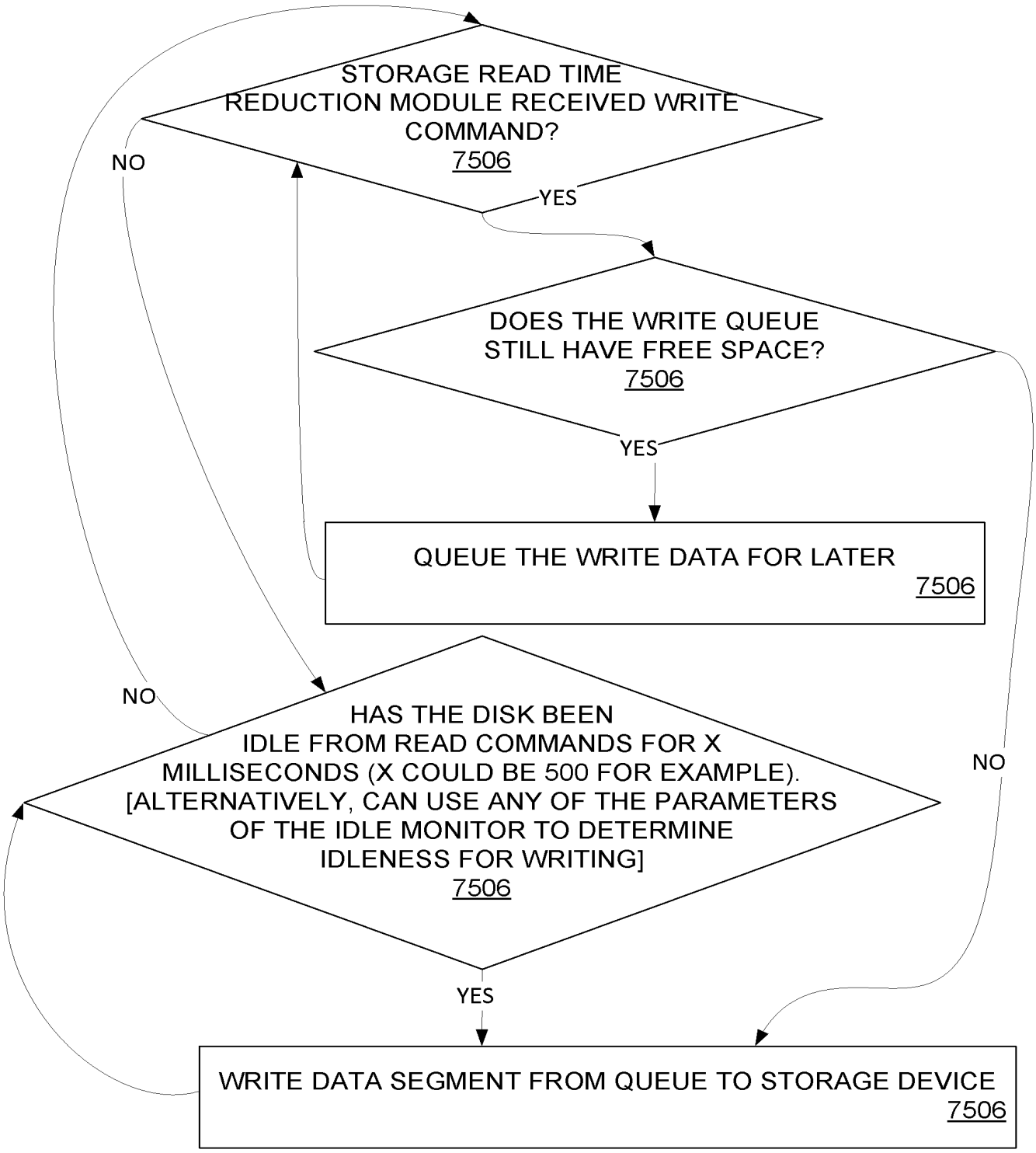


FIG. 420 – AP SELECT ALGORITHM - FLOWCHART