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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
2 February 2017 (02.02.2017)

WIPO | PCT

(10) International Publication Number
WO 2017/019036 A1

(51) International Patent Classification:
G06F 17/30 (2006.01)

(21) International Application Number:
PCT/US2015/042447

(22) International Filing Date:
28 July 2015 (28.07.2015)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant: **HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP** [US/US]; 11445 Compaq Center Drive West, Houston, TX 77070 (US).

(72) Inventor: **WICKERAAD, John A.**; 8000 Foothills Blvd., Roseville, California 95747 (US).

(74) Agents: **TONG, Kin-Wah** et al.; Hewlett Packard Enterprise, 3404 E. Harmony Road, Mail Stop 79, Fort Collins, CO 80528 (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, 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).

Declarations under Rule 4.17:

— *as to the identity of the inventor (Rule 4.17(i))*

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

Published:

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

(54) Title: HASH VALIDATION KEY

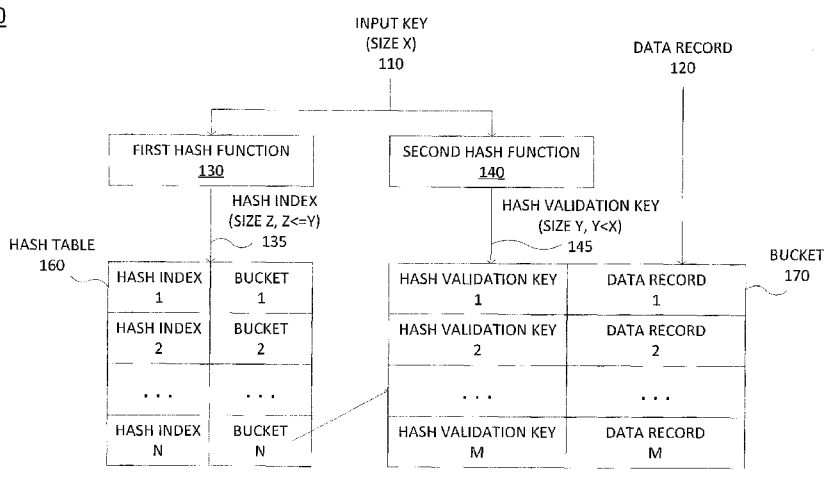


FIG. 1

(57) Abstract: In one example, a method for storing a hash validation key is described. The method may include a processor applying a first hash function to an input key to determine a hash index. In one example, the input key is associated with a data record. The method may further include the processor applying a second hash function to the input key to generate a hash validation key, and storing the hash validation key and the data record in a bucket for the hash index in a hash table.

WO 2017/019036 A1

HASH VALIDATION KEY

BACKGROUND

[0001] Hash tables are used in a variety of data processing contexts, such as for network routing functions, image matching, language recognition, and so forth. Hash tables may be stored as data structures in computing device memories. A variety of hash functions may be used in connection with such hash tables to create hash indexes from input keys. In general, the hash indexes that are created are of a smaller size, e.g., less bits, than the input keys. In some cases, aliasing can occur, where two or more input keys map to the same hash index. This may cause undesirable performance, such as incorrect router behavior in the case where the hash table is used in connection with network routing functions.

BRIEF DESCRIPTION OF THE DRAWINGS

[0002] FIG. 1 illustrates an example system of the present disclosure;

[0003] FIG. 2 illustrates a flowchart of a first example method for maintaining a hash table;

[0004] FIG. 3 illustrates a flowchart of a second example method for maintaining a hash table;

[0005] FIG. 4 illustrates a flowchart of a third example method for maintaining a hash table;

[0006] FIG. 5 illustrates a flowchart of a fourth example method for maintaining a hash table;

[0007] FIG. 6 illustrates an additional example system of the present disclosure; and

[0008] FIG. 7 depicts a high-level block diagram of an example computer that can be transformed into a machine capable of performing the functions described herein.

DETAILED DESCRIPTION

[0009] In one example, the present disclosure describes a device, method, and non-transitory computer-readable medium for maintaining a hash table. For example, a processor may apply a first hash function to an input key to determine a hash index. In one example, the input key is associated with a data record. The processor may further apply a second hash function to the input key to generate a hash validation key, and store the hash validation key and the data record in a bucket for the hash index in a hash table.

[0010] In another example, the present disclosure describes a device, method, and non-transitory computer-readable medium for maintaining a hash table. For example, a processor may apply a first hash function to a search key to generate a hash value and locate a bucket for a hash index in a hash table using the hash value. In one example, the hash index corresponds to the hash value. The processor may further apply a second hash function to the search key to generate a validation value, compare the validation value to a hash validation key in the bucket for the hash index, and retrieve a data record associated with the hash validation key when the validation value corresponds to the hash validation key.

[0011] In another example, the present disclosure describes a device for maintaining a hash table. In one example, the device may include a hardware memory storing the hash table that comprises a bucket associated with a hash index. In one example, the bucket may comprise a hash validation key and a data record associated with the hash validation key. In one example, the hash validation key is associated with an input key. The device may further include

hardware logic to apply a first hash function to a search key to generate a hash value and to locate the bucket for the hash index in the hash table using the hash value. In one example, the search key corresponds to the input key and the hash value corresponds to the hash index.

[0012] Hash tables are employed in many data processing tasks. For example, a router hash table may include many hash entries for various functions. Input keys used in connection with such hash tables may average 100-200 bits or more. For example, an input key may comprise a plurality of fields such as: a layer 2 source address, a layer 2 destination address, a flow (n-tuple), and so forth. Hash indexes created from such input keys are often 20 bits or less, which can result in aliasing and incorrect router behavior. Aliasing, or collisions, may similarly arise in the context of hash tables used for image processing, language recognition, and so forth.

[0013] In one example, the present disclosure implements a first hash function for generating a hash index for an input key and a second hash function to generate a hash validation key. In one example, the hash validation key may comprise a wider hash of the input key than the hash index, e.g., a greater number of bits or other symbols. For instance, when input keys are between 100 and 200 bits, the hash indexes may be 12-20 bits, while hash validation keys may comprise 64-72 bits. As such, while the second hash for the hash validation keys is a "wide" hash, it comprises less bits or symbols than the original input keys. The hash validation key may be stored in the hash table in a hash entry for confirming matches, rather than the entire input key. In this way, storage overhead for the hash table can be reduced or more hash entries may be accommodated in the hash table without increasing the storage volume.

[0014] It should be noted that the hash index and the hash validation key are similar insofar as both are the results of applying a hash function to an input key. However, the hash index may be used for locating a bucket and/or a hash entry in a hash table, while the hash validation key may be used to confirm a match, as described in greater detail below. In one example, the hash validation key and a data record associated with the input key are stored in a hash table and associated with the first hash index. For example, the hash

validation key and associated data record may comprise a "hash entry" that is stored in a bucket in the hash table associated with the first hash index.

[0015] In accordance with the present disclosure, a bucket can store anywhere from zero, one, or several hash entries. Thus, in one example, multiple input keys may map to the same hash index, and hence the same bucket. However, in another example, a bucket may have at most a single hash entry. For example, in some implementations of a two-way hash, a four-way hash, a cuckoo hash, or the like, each hash index may have a single storage location in the hash table, e.g., a bucket, where a single hash entry may be stored. However, in any of these examples, a hash validation key may be used as a match validation, e.g., to confirm that a data record should be associated with a particular input key mapping to the bucket.

[0016] A hash entry that requires storing the original input key for match validation is costly in terms of the number of bits or other symbols to store. For example, in the case of a router hash table for routing functions, the original input key may be up to 400-600 bits, e.g., for Internet Protocol (IP) Version 6 (IPV6) functions. The return data may vary from several bits up to 200 bits or more. Thus, the entire hash entry may be up to 800 bits or more. In addition, storing the original input key may occupy up to half or more of the total storage space for the hash entry. This impacts the number of hash entries that can be placed on a given storage volume, e.g., a random access memory (RAM), or other storage type. At the same time, there are a number of router/switch functions that utilize large hash tables. By storing a validation key from a second hash, e.g., 64-72 bits of data, instead of the original input key, the amount of storage required for each hash entry can be reduced. Thus, a smaller application specific integrated circuit (ASIC), e.g., with a smaller RAM, or other logic/memory device may be used, or the hash table may store more hash entries in the same storage area. For example, each hash entry may require ten to more than fifty percent less storage bits or other symbols as compared to storing the entire input key. The savings may depend on the original input key width relative to the hash validation key width. The data record size may also impact the overall percentage of storage savings.

[0017] In one example, the width of the hash validation key is selected to be wide enough to reduce the possibility of a false match to an acceptable level (e.g., near zero). In one example, using 64-72 bits for the hash validation key reduces the possibility of undesired false match to a low enough likelihood that it can be treated as zero, e.g., for routing functions where the original input keys are from 100-600 bits. However, a hash validation key may be selected to comprise any number of sizes/widths, e.g., 32 to 96 bits. This may depend upon the willingness to tolerate collision risks and the type of use of the hash table. For instance, using a 64 bit validation key in a system with inputs ranging from 100 to 200 bits, there may be less than a 1 / (2^{64}) likelihood of a collision/aliasing. In one example, the likelihood may approach $1/(2^{64}) \times 1/(2^{n})$, where "n" is the original hash index width that is used. With 96 bit validation keys, the likelihood of a collision is even less. Another advantage associated with storing a wide hash is a reduction in the power requirement per hash entry. The power savings per hash entry may be reduced by approximately the same percentage as the storage area reduction per hash entry.

[0018] Although various hash functions may be used for both the first hash and the second hash, the hash functions may be selected to ensure good random values for all input keys (considering strides of input fields, etc). In one example, using different hash functions for the original hash and the verification hash better insulates against the possibility of a hash collision. However, in one example, the same hash function may be used to produce the hash indexes and the hash verification keys. For instance, the 12 most significant bits may comprise the hash index, while a remaining 64 bits may be used for the hash verification key, but it may be more likely for a hash collision to occur.

[0019] In addition, examples of the present disclosure are configurable such that different hash functions may be used for the first hash function and the second hash function, and such that the hash functions may be changed and substituted. For instance, a single router may have a number of hash tables associated with different functions. Some of the hash tables may take the same format and same size inputs, but have an entirely different set of possible outputs. Others of the hash tables can take entirely different formatted and

sized inputs, and also have different sets of possible outputs. The hash functions used in connection with the different hash tables may be the same, or may be entirely different for different hash tables, in addition to the various sizes of the hash indexes and verification keys. These and other aspects of the present disclosure are described in greater detail below in connection with the example FIGs. 1-7.

[0020] FIG. 1 illustrates an example system 100 of the present disclosure for maintaining a hash table. In one example, the system 100 may comprise a portion of a router in a communication network, e.g., an IP network. In one example, the system 100 receives an input key 110. In one example, the input key 110 is of size "X", e.g., 100-600 bits, or any other number of bits or symbols. The input key 110 may comprise, for example, an n-tuple characterizing a packet of a flow in an IP network. The data record 120 may comprise a particular output, a routing decision or other action to take in response to an input corresponding to the input key 110, e.g., a packet or other unit of data of a flow characterized by the n-tuple. In this regard, it should be noted that input key 110 may comprise the first instance of a packet or other data unit of a network flow received by a router, or may comprise additional data of a network flow that has already been detected and processed by the router. Thus, the system 100 may process input key 110 to locate an existing hash entry in hash table 160, or may determine that input key 110 is a new instance, requiring a new hash entry in the hash table 160.

[0021] As shown in FIG. 1, a first hash function 130 is applied to the input key 110 to generate a hash index 135. In one example, the hash index 135 is of size "Z," where Z is less than X, the size, or width, of the input key. In one example, Z is between 12 and 30 bits, where X is between 100 and 200 bits. A second hash function 140 is also applied to the input key 110 to generate a hash validation key 145. In one example, the hash validation key 145 is of size "Y," where Y is less than X. In one example, Y is also greater than Z, such that the hash validation key 145 is of a larger size than the hash index 135. In one example, the first hash function and the second hash function comprise any hash function that is suitable for randomly or semi-randomly mapping an input

key to a smaller-size hash of the input key. In other words, the hash functions should produce a good distribution of input keys over a range of hash indexes or hash verification key values. In addition, the input data may include fields with strides. As such, the hash functions may also be selected such that any strides present in the input data do not affect the randomness of the hash function output in a significant way or manifest in other problems. Experimentation may confirm which hash function meets these criteria. In addition, different hash functions may be more advantageous than others depending upon the particular use of the hash table and the type of data to which it relates, e.g., whether it is used for network routing functions, image processing, speech recognition, and so forth.

[0022] FIG. 1 also illustrates hash table 160 that includes various hash indexes and associated buckets. As shown, hash table 160 includes hash indexes 1 through N and buckets 1 through N, where "N" is a maximum number of buckets in the hash table 160, or a current number of buckets that are contained in the hash table 160. In one example, the system 100 may compare hash index 135 to the hash indexes that are present in the hash table 160 to locate a corresponding bucket. For instance, it may be determined that hash index 135 matches one of the hash indexes and may therefore be associated with the corresponding bucket.

[0023] The buckets in hash table 160 may take the form of bucket 170, as further illustrated in FIG. 1. In the present example, bucket 170 may represent a bucket that is located by looking-up hash index 135 in the hash table 160. The bucket 170 may include a number of hash entries, each comprising a hash validation key and an associated data record. As shown, bucket 170 includes hash entries with hash validation keys 1 through M and data records 1 through M, where "M" is a maximum number of hash entries in the bucket 170, or a current number of hash entries that are contained in the bucket 170. In one example, hash validation key 145 is compared to existing hash validation keys in the bucket 170. If there is match between the hash validation key 145 and an existing hash validation key in the bucket 170, the data record associated with the matching hash validation key may be retrieved. In this way, the input key

110 may be mapped to the appropriate data record using the hash validation key 145 for resolving any conflicts with respect to other input keys that may map to the same hash index and the same bucket.

[0024] However, if there is no match for hash validation key 145 in the bucket 170, then the hash validation key 145 may be stored in a new hash entry in bucket 170. In one example, data record 120 may also be obtained and stored in the new hash entry in association with the hash validation key 145. For instance, if input key 110 is the first instance of receiving data of a particular flow, a router may not yet have a routing decision for the flow. Thus, in one example, additional logic may process the input key and/or other data of the flow, make a routing decision, and provide the routing decision as data record 120. With data record 120 storing the routing decision, any additional input keys derived from a packet for the same flow may be matched to the data record 120 and the packet routed accordingly. In another example, the data record 120 may comprise logic for making a routing decision, or may comprise a pointer to another storage location for obtaining further instructions for making a routing decision.

[0025] It should be noted that the foregoing example is described with respect to a system 100 implementing a hash table 160 for a router. However, the examples of the present disclosure are broadly applicable to hash tables that may be used for various purposes where faster searches and greater storage efficiency would improve existing systems. For example, hash table 160 may store N-grams for language processing, may be used to store and process features for image, fingerprint, and facial recognition, may be utilized to correlate data from disparate sources, e.g., for multiple antennas in radio communications, and so forth. Thus, it should be appreciated that the description of FIG. 1 is provided for illustrative purposes and that other, further and, different examples may be implemented in the same or a similar system. In addition, the hash table 160 and bucket 170 are provided for illustrative purposes. Thus, it should also be appreciated that in other, further, and different examples of the present disclosure, hash table 160 and/or bucket 170 may be implemented in a different format than that which is illustrated in FIG. 1.

[0026] FIG. 2 illustrates a flowchart of an example method 200 for maintaining a hash table. The method 200 may be performed, for example, by the system 100 of FIG. 1, or by at least one of the components of the system 600 illustrated in FIG. 6. For example, the method 200 may be performed by at least one of the hardware logic units 611-614 and 630 in conjunction with at least one of the storage units 651-654, comparison logic units 661-664, output enable gates 671-674, and so forth. However, the method 200 is not limited to implementation with the systems illustrated in FIGs. 1 and 6, but may be applied in connection with any number of devices that store hash tables and/or perform hash-related services. Alternatively, or in addition, at least one of the blocks of the method 200 may be implemented by a computing device having a processor, a memory, and input/output devices as illustrated below in FIG. 7, specifically programmed to perform the blocks of the method. Although any one of the elements in system 600, or in a similar system, may be configured to perform various blocks of the method 200, the method will now be described in terms of an example where blocks of the method are performed by a processor, such as processor 702 in FIG. 7. As used in connection with the description of FIG. 2, the term "processor" may also include multiple processors, hardware state machines, or hardware logic units, e.g., an application specific integrated circuit (ASIC), a programmable logic device (PLD), such as a field programmable gate array (FPGA), and so forth.

[0027] The method 200 begins in block 205. In block 210, the processor applies a first hash function to an input key to generate a hash index. In one example, the input key is associated with a data record. In one example, the input key may comprise a source IP address, a destination IP address, a destination port number, an n-tuple, or the like of a packet or other data unit that is received by a router. The data record may comprise a routing decision or routing logic for making a routing decision regarding the packet. Alternatively, or in addition, the first data record may comprise another action to take in response to the input key and/or a processing to be performed on the packet associated with the input key. For instance, the data record may direct that a packet associated with the input key should be rerouted by changing a

destination IP address of the packet, that the packet should be dropped, and so forth. In one example, the first hash function may comprise any hash function that randomly or semi-randomly maps the input key to a smaller-size hash index.

[0028] In block 220, the processor applies a second hash function to the first input key to generate a hash validation key. In one example, the second hash function may comprise any hash function that randomly or semi-randomly maps the input key to a smaller-size hash validation key. In one example, the second hash function is different from the first hash function. For instance, this may help ensure that two input keys that map to the same hash index via the first hash function would not also map to the same hash validation key via the second hash function. In one example, the second hash function produces a hash validation key that is smaller than the input key, but larger than the hash index (in terms of the width, or the number of bits or symbols comprising the input key, hash index, and hash validation key, respectively). It should be noted that the hash validation key is similar to a hash index insofar as both a hash index and the hash validation key comprise outputs from a hash function. However, the hash index may be used to locate a hash entry in a hash table, whereas the hash validation key may be used to differentiate among different hash entries in the hash table associated with the same hash index, or to confirm a matching hash entry.

[0029] In block 230, the processor stores the hash validation key and the data record in a bucket for the first hash index in a hash table. For example, the hash index may be used to locate a bucket in the hash table. The bucket may comprise a chain of hash entries, with a first hash entry in the bucket containing a pointer to a next hash entry in the bucket, and so forth. In another example, the bucket for each hash index may comprise an array, e.g., a dynamic array that may be resized as more or less hash entries are contained in the bucket. In still another example, the bucket may comprise a sequence of addresses in the hash table, starting with the hash index. For instance, if there is only one hash entry for a hash index, the hash entry will be placed in the hash table at the hash index location, which may also be referred to as an address in the hash

table. If there is a second hash entry that is mapped to the same hash index, it will be placed at the next address in the hash table adjacent to the address/location of the hash index. Thus, the "bucket" will comprise the sequence of addresses starting from the hash index and continuing through adjacent hash indexes that have hash entries until an empty hash index, or address is found. In any case, once the address, or location of the bucket for the hash index is located, the hash validation key and the data record may be stored in the bucket, e.g., as a hash entry. In one example, block 230 may be performed following a confirmation that there is not an existing hash entry for the hash validation key and the data record in the bucket. In one example, block 230 may be performed following the eviction of a previous hash entry in the bucket, e.g., in the case where a cuckoo hashing scheme is employed.

[0030] Following block 230, the method 200 proceeds to block 295 where the method ends.

[0031] FIG. 3 illustrates a flowchart of an example method 300 for maintaining a hash table. The method 300 may be performed, for example, by the system 100 of FIG. 1, or by at least one of the components of the system 600 illustrated in FIG. 6. For example, the method 300 may be performed by at least one of the hardware logic units 611-614 and 630 in conjunction with at least one of the storage units 651-654, comparison logic units 661-664, output enable gates 671-674, and so forth. However, the method 300 is not limited to implementation with the systems illustrated in FIGs. 1 and 6, but may be applied in connection with any number of devices that store hash tables and/or perform hash-related services. Alternatively, or in addition, at least one of the blocks of the method 300 may be implemented by a computing device having a processor, a memory, and input/output devices as illustrated below in FIG. 7, specifically programmed to perform the blocks of the method. Although any one of the elements in system 600, or in a similar system, may be configured to perform various blocks of the method 300, the method will now be described in terms of an example where blocks of the method are performed by a processor, such as processor 702 in FIG. 7. As used in connection with the description of FIG. 3, the term "processor" may also include multiple processors, hardware

state machines, or hardware logic units, e.g., an application specific integrated circuit (ASIC), a programmable logic device (PLD), such as a field programmable gate array (FPGA), and so forth.

[0032] The method 300 begins in block 305. In block 310, the processor receives an input key, where the input key is associated with a data record. In one example, the input key may comprise a source IP address, a destination IP address, a destination port number, an n-tuple, or the like. For example, the input key may be associated with a packet or other data unit that is received by a network router, e.g., in an IP network or the like. The data record may comprise a routing decision or routing logic for making a routing decision regarding the packet. Alternatively, or in addition, the data record may comprise another action to take in response to the input key and/or a processing to be performed on the packet associated with the input key. However, it should be noted that the present method 300 and other examples of the present disclosure are not limited to router functions, but are broadly applicable to various systems that utilize hash tables and where faster searches and greater storage efficiency would be beneficial, e.g., in connection with language processing, image, fingerprint, and facial recognition, radio communications, and so forth.

[0033] In block 315, the processor applies a first hash function to the input key to generate a hash index. In one example, the first hash function may comprise any hash function that randomly or semi-randomly maps the input key to a smaller-size hash index. In one example, block 315 may comprise the same or similar operation(s) as described above in connection with block 210 of the method 200.

[0034] In block 320, the processor applies a second hash function to the input key to generate a hash validation key. In one example, the second hash function may comprise any hash function that randomly or semi-randomly maps the input key to a smaller-size hash validation key. In one example, the first hash function and the second hash function comprise different hash functions. In one example, the second hash function produces a hash validation key that is smaller than the input key, but larger than the hash index. In one example,

block 320 may comprise the same or similar operation(s) as described above in connection with block 220 of the method 200.

[0035] In block 325, the processor stores the hash validation key and the data record in a bucket for the hash index in a hash table. For example, the hash index may be used to locate a bucket in the hash table. In one example, block 325 may comprise the same or similar operation(s) as described above in connection with block 230 of the method 200.

[0036] In block 330, the processor receives a search key corresponding to the input key. For instance, the search key may be equivalent to the input key. To illustrate, a router may receive a first packet for a flow. The input key may be derived from the first packet, and may comprise an n-tuple of source IP address, destination IP address, destination port number, etc. If there is no hash entry already associated with the flow in the hash table, i.e., it is the first time that the router is detecting the flow, a new hash entry may be created in the hash table for the flow. For example, blocks 310-325 may comprise the detection of a new flow and the creation of a hash table entry associated with the input key, and therefore associated with the flow. A subsequent packet or other data unit of the flow will have a similar input key. For purposes of distinguishing between a first detection of a flow and a subsequent detection of the same flow, the terms "input key," and "search key" are used. However, it should be understood that the input key and search key may have corresponding values, or may be equivalent, if the input key and search key are associated with the same flow.

[0037] In block 335, the processor applies the first hash function to the search key to generate a hash value. In one example, the hash value may be equivalent to the hash index, e.g., where the search key and the input key are associated with the same flow.

[0038] In block 340, the processor locates the bucket for the hash index in the hash table using the hash value. For example, where the search key and the input key are the same, the search key and the input key should be processed the same by the first hash function such that the hash value and the hash index are equivalent, i.e., having corresponding values. Thus, the hash value may be used to locate the address for the hash index in the hash table.

[0039] In block 345, the processor applies the second hash function to the search key to generate a validation value. In one example, the validation value is equivalent to the hash validation key. For instance, when the search key and the input key are equivalent, the validation value and the hash validation key should be processed the same by the second hash function such that the validation value and the hash validation key are equivalent, i.e., having corresponding values.

[0040] In block 350, the processor compares the validation value to at least the hash validation key. For example, the bucket may be traversed to find a correct hash entry having a hash validation key corresponding to the validation value. For instance, the validation value is derived from the search key, which may be associated with the same flow as the input key and the hash validation key. The hash validation key may be stored in a hash entry of the bucket, along with the data record. However, there may be other hash entries in the bucket. For instance, the other hash entries may be associated with other flows, and have different validation keys and different data records. As such, the comparison between the validation value and the hash validation keys of any hash entries that are already present in the bucket allows the identification of the correct hash entry, e.g., the one with a hash validation key that corresponds to the validation value. In this case, since the search key corresponds to the input key, the validation value will also correspond to the hash validation key. For example, the hash validation key is already stored in a hash entry in the bucket by virtue of the operations of block 325.

[0041] In block 355, the processor retrieves the data record when the validation value is equivalent to the hash validation key. For instance, in one example the data record may be output to another device or component. In another example, the data record may be used by the processor to make a routing decision or to modify a packet or other data unit, in the case where the method relates to processing network flows, and so forth.

[0042] Following block 355, the method 300 proceeds to block 395 where the method ends.

[0043] FIG. 4 illustrates a flowchart of an example method 400 for

maintaining a hash table. The method 400 may be performed, for example, by the system 100 of FIG. 1, or by at least one of the components of the system 600 illustrated in FIG. 6. For example, the method 400 may be performed by at least one of the hardware logic units 611-614 and 630 in conjunction with at least one of the storage units 651-654, comparison logic units 661-664, output enable gates 671-674, and so forth. However, the method 400 is not limited to implementation with the systems illustrated in FIGs. 1 and 6, but may be applied in connection with any number of devices that store hash tables and/or perform hash-related services. Alternatively, or in addition, at least one of the blocks of the method 400 may be implemented by a computing device having a processor, a memory, and input/output devices as illustrated below in FIG. 7, specifically programmed to perform the blocks of the method. Although any one of the elements in system 600, or in a similar system, may be configured to perform various blocks of the method 400, the method will now be described in terms of an example where blocks of the method are performed by a processor, such as processor 702 in FIG. 7. As used in connection with the description of FIG. 4, the term "processor" may also include multiple processors, hardware state machines, or hardware logic units, e.g., an application specific integrated circuit (ASIC), a programmable logic device (PLD), such as a field programmable gate array (FPGA), and so forth.

[0044] The method 400 begins in block 405. In block 410, the processor applies a first hash function to a search key to generate a hash value. In one example, the first hash function may comprise any hash function that randomly or semi-randomly maps the search key to a smaller-size hash value. In one example, block 410 may comprise the same or similar operation(s) as described above in connection with block 335 of the method 300.

[0045] In block 420, the processor locates a bucket for a hash index in a hash table using the hash value. For example, where the search key and an input key are the same, the search key and the input key should be processed the same by the first hash function such that the hash value and a hash index generated from the input key via the first hash function have corresponding values. Thus, the hash value may be used to locate the address for the hash

index in the hash table. In one example, block 420 may comprise the same or similar operation(s) as described above in connection with block 340 of the method 300.

[0046] In block 430, the processor applies a second hash function to the search key to generate a validation value. The second hash function may comprise any hash function that randomly or semi-randomly maps the search key to a smaller-size validation value. In one example, the first hash function and the second hash function are different hash functions. In one example, the second hash function produces a validation value that is smaller than the search key, but larger than the hash value. In one example, block 430 may comprise the same or similar operation(s) as described above in connection with block 345 of the method 300.

[0047] In block 440, the processor compares the validation value to a hash validation key in a bucket for the hash index in the hash table. For example, the bucket may be traversed to find a correct hash entry corresponding to the validation value, e.g., the hash entry with the hash validation key, where the hash validation key is equivalent to the validation value. The bucket may comprise zero, at least one hash entry, each with a different hash validation key and different data records. However, in the method 400 it is assumed that the hash validation key that corresponds to the validation value is already present in the bucket.

[0048] In one example, the search key is determined to match an input key associated with the hash validation key when the validation value and the hash validation key are determined to be equivalent. For example, the search key and the input key may comprise a same n-tuple identifying a same flow in a communication network. As mentioned above, a bucket associated with the hash index may include multiple hash entries, each having a different hash validation key and different data record. Each of the hash entries may therefore be associated with a different network flow. As such, a determination that a validation value and a hash validation key are equivalent is a confirmation that a packet or other data unit is part of the same flow. In addition, by storing the hash validation key in the hash entry along with the data record, the size of the

hash table may be reduced, or the number of hash entries in the hash table may be increased as compared to a hash table where a full input key is stored along with each data record for verification purposes.

[0049] In block 450, the processor retrieves a data record associated with the hash validation key when the validation value is equivalent to the hash validation key. For instance, the data record may be contained in a hash entry along with the hash validation key. In one example the data record may be output to another device or component. In another example, the data record may be used by the processor to make a routing decision or to modify a packet or other data unit, in the case where the method relates to processing network flows, and so forth.

[0050] Following block 450, the method 400 proceeds to block 495 where the method ends.

[0051] FIG. 5 illustrates a flowchart of an example method 500 for maintaining a hash table. The method 500 may be performed, for example, by system 100 of FIG. 1, or by at least one of the components of the system 600 illustrated in FIG. 6. For example, the method 500 may be performed by at least one of the hardware logic units 611-614 and 630 in conjunction with at least one of the storage units 651-654, comparison logic units 661-664, output enable gates 671-674, and so forth. However, the method 500 is not limited to implementation with the systems illustrated in FIGs. 1 and 6, but may be applied in connection with any number of devices that store hash tables and/or perform hash-related services. Alternatively, or in addition, at least one of the blocks of the method 500 may be implemented by a computing device having a processor, a memory, and input/output devices as illustrated below in FIG. 7, specifically programmed to perform the blocks of the method. Although any one of the elements in system 600, or in a similar system, may be configured to perform various blocks of the method 500, the method will now be described in terms of an example where blocks of the method are performed by a processor, such as processor 702 in FIG. 7. As used in connection with the description of FIG. 5, the term “processor” may also include multiple processors, hardware state machines, or hardware logic units, e.g., an application specific integrated

circuit (ASIC), a programmable logic device (PLD), such as a field programmable gate array (FPGA), and so forth.

[0052] The method 500 begins in block 505. In block 510, the processor applies a first hash function to a search key to generate a hash value. In one example, the first hash function may comprise any hash function that randomly or semi-randomly maps the search key to a smaller-size hash value. In one example, block 510 may comprise the same or similar operation(s) as described above in connection with block 335 of the method 300 and/or block 410 of the method 400.

[0053] In block 520, the processor locates a bucket for a hash index in a hash table using the hash value. For example, where the search key and an input key are the same, the search key and the input key should be processed the same by the first hash function such that the hash value and a hash index generated from the input key via the first hash function have corresponding values. Thus, the hash value may be used to locate the address for the hash index in the hash table. In one example, block 520 may comprise the same or similar operation(s) as described above in connection with block 340 of the method 300 and/or block 420 of the method 400.

[0054] Following block 520, the method 500 proceeds to block 595 where the method ends.

[0055] It should be noted that although not explicitly specified, at least one of the blocks, functions, or operations of the methods 200, 300, 400, and 500 described above may include storing, displaying, and/or outputting. In other words, any data, records, fields, and/or intermediate results discussed in the methods can be stored, displayed, and/or outputted to another device depending on the particular application. Furthermore, blocks, functions, or operations in FIGs. 2-5 that recite a determining operation, or involve a decision, do not necessarily imply that both branches of the determining operation are practiced. In other words, one of the branches of the determining operation can be deemed as optional.

[0056] It should also be noted that the examples of FIGs. 2-5 are applicable to various types of hashing schemes, such as 2-way associative hashing, 4-way

associative hashing, cuckoo hashing, and so forth. For example, when used in connection with 2-way associative hashing, blocks 310-325 may comprise storing a hash entry in a bucket associated with a first portion of a hash table, where the hash table hash two portions, each portion associated with a different hash function for generating different hash indexes. Similarly, blocks 330-340 may involve using the same hash function associated with the first portion of the hash table to generate a hash value that can be used the search the first portion of the has table to locate the bucket where the hash entry is stored. Operations similar to blocks 330-340 may be repeated using a different hash function associated with a section portion of the hash table. However, no matching hash entry may be found in the second portion of the hash table, since the hash entry has been stored at the hash index in the first portion of the hash table. Thus, in other, further, and different examples of the present disclosure, 2-way associative hashing, 4-way associative hashing, cuckoo hashing, and any other hashing techniques may be used in connection with any of the example methods 200, 300, 400, or 500 of FIGs. 2-5.

[0057] FIG. 6 illustrates an example system 600 of the present disclosure, e.g., for maintaining a hash table. In one example, the system 600 may be implemented on a single device or may comprise hardware that is distributed among several devices. The system 600 includes several storage units 651-654 which may comprise portions any kind of static or dynamic storage media, such as a random access memory (RAM), several RAMs, or the like. For instance, the storage units 651-654 may individually or collectively comprise at least a portion of a lookup RAM, e.g., a dynamic RAM (DRAM), a static RAM (SRAM), or the like. Each of the storage units 651-654 may store a hash table having a plurality of buckets that are addressable by a plurality of hash indexes. Each bucket may include, zero, one, or more hash entries, where each hash entry includes a hash validation key and a data record. Alternatively, or in addition, each of the storage units 651-654 may store respective portions of a hash table, where each portion contains buckets that are separately addressable, and where a same hash function or different hash functions may be associated with the different portions. For instance, system 600 may be

used to implement a cuckoo hashing scheme or a 4-way associative hashing scheme, where a hash entry may be stored in any one of four different storage locations, where each of the four possible storage locations is located in a respective one of the portions of the hash table in storage units 651-654. For example, a hash entry may be stored in a bucket in one of the respective portions of the hash table that is not occupied (in the case where a bucket can have at most one hash entry), or in a bucket that is least occupied.

[0058] System 600 further includes an input port 601 for receiving an input key and/or a search key, hardware logic units 611-614 and 630, select lines 621-624, verification line 640, comparison logic units 661-664, output enable gates 671-674, and an output port 691. To illustrate the functioning of system 600, the following example of a cuckoo hashing scheme is provided. A search key may be received via the input port 601. The search key may comprise, for example, an n-tuple of a packet that is received. For instance, the system 600 may comprise a network router. The search key may then be processed by at least one of the hardware logic units 611-614. For instance, in a cuckoo hashing scheme, hardware logic units 611-614 may implement different hash functions that operate on input keys and/or search keys that are received via input port 601, and generate hash indexes and/or hash values that are output on select lines 621-624. In one example, hardware logic unit 630 also implements a hash function for generating a hash validation key and/or a validation value to output on verification line 640. In another example, a same hash functions may be used in hardware logic units 611-614, in order to provide a 4-way set associative hash system, for instance.

[0059] The hash indexes or hash values that are output on select lines 621-624 may be used to select a bucket from a hash table in one of storage units 651-654. For example, a bucket from a hash table portion of storage unit 651 may be selected via a hash value on select line 621 that is generated by hardware logic unit 611. Any hash entries in the bucket may be accessed and the hash validation key(s) retrieved and passed to the comparison logic unit 661. The comparison logic unit 661 may then compare the hash validation key(s) to a validation value that is received from verification line 640. If there is

a match, the comparison logic unit 661 may provide an enable signal to the output enable gate 671. The output enable gate 671 may pass the data record associated with the hash validation key that matches to the validation value as an output to the output port 691.

[0060] It should be noted that at the same time, buckets from a hash table portions of storage units 652-654 may be selected via respective hash values on select lines 622-624. Any hash entries in these buckets may then be accessed, the hash validation keys compared to the validation value on verification line 640 by the hardware logic units 662-664, and so forth. However, in a multi-way hashing scheme (including 4-way cuckoo hashing), only a single hash entry should exist that matches the search key. Therefore, if a match is found by comparison logic unit 661, comparison logic units 662-664 should not find matches, and therefore should not send enable signals to the enable gates 672-674. It should be noted, that occasionally, the same entry may be found in two hash tables, or two hash table portions. For example, in cuckoo hashing, if it is determined to move an existing entry from one hash table to another to make room for a new entry, a copy may first be made in the new hash table location prior to deleting or overwriting the existing entry in the old hash table location. Thus, there may be a small time window where the same entry exists in both locations. However, in practice any entry that matches the search key should contain the same data record. Therefore, either entry may be used to obtain the same data record to output as the return data.

[0061] The foregoing is just one example use of system 600. In addition, it should be noted that the system 600 of FIG. 6 is provided for illustrative purposes. Thus, other systems that are different from system 600 may be utilized in accordance with the present disclosure. For example, a system may be used where hardware logic portions 613 and 614, storage units 653 and 654, comparison logic units 663 and 664, and enable gates 673 and 674 are omitted. Such as system may be used, for example, in a 2-way associative hashing scheme or a 2-way cuckoo hashing scheme. Additional components may be omitted, e.g., so that hardware logic units 612-614, storage units 652-654, and related components are omitted. In still another example, any two or more of

storage units 651-654 may be implemented in a single physical storage device, such that storage units 651-654 comprise logical partitions of the physical storage device. In addition, various components of the system 600 may be implemented in a form different from that which is illustrated in FIG. 6. For instance, enable gates 671-674 are illustrated as AND gates. However, these components, as well as others, may be implemented by a processor executing computer-readable instructions, in a digital signal processing (DSP) hardware component, in a programmable logic device (PLD), and so forth. As such, variants of the above-disclosed and other features and functions, or alternatives thereof, may be omitted, or may be combined or altered into many other different systems or applications.

[0062] FIG. 7 depicts an example high-level block diagram of a computing device suitable for use in performing the functions described herein. As depicted in FIG. 7, the computer 700 comprises a hardware processor element 702, e.g., a central processing unit (CPU), a microprocessor, or a multi-core processor, a memory 704, e.g., random access memory (RAM), a module 705 for maintaining a hash table, and various input/output devices 706, e.g., storage devices, including but not limited to, a tape drive, a floppy drive, a hard disk drive or a compact disk drive, a receiver, a transmitter, a speaker, a display, a speech synthesizer, an output port, an input port and a user input device, such as a keyboard, a keypad, a mouse, a microphone, and the like. Although one processor element is shown, it should be noted that the computer may employ a plurality of processor elements. Furthermore, although one computer is shown in the figure, if the method(s) as discussed above is implemented in a distributed or parallel manner for a particular illustrative example, i.e., the blocks of the above method(s) or the entire method(s) are implemented across multiple or parallel computers, then the computer of this figure is intended to represent each of those multiple computers.

[0063] It should be noted that the present disclosure can be implemented by machine readable instructions and/or in a combination of machine readable instructions and hardware, e.g., using application specific integrated circuits (ASIC), a programmable logic array (PLA), including a field-programmable gate

array (FPGA), or a state machine deployed on a hardware device, a computer or any other hardware equivalents, e.g., computer readable instructions pertaining to the method(s) discussed above can be used to configure a hardware processor to perform the blocks, functions and/or operations of the above disclosed methods.

[0064] In one example, instructions and data for the present module or process 705 for maintaining a hash table, e.g., machine readable instructions, can be loaded into memory 704 and executed by hardware processor element 702 to implement the blocks, functions, or operations as discussed above in connection with the example methods 200, 300, 400, and 500. For instance, the module 705 may include a plurality of computer-readable components, including a first hash function component 711, a bucket locator component 712, a second hash function component 713, and a data record retrieval component 714. When executed by the hardware processor element 702 the first hash function component 711 may cause the processor to generate a hash value in response to a received search key, the bucket locator component 712 may cause the hardware processor element 702 to locate a bucket corresponding to the hash value in a hash table, the second hash function component 713 may cause the hardware processor element 702 to generate a validation value, and the data record retrieval component 714 may cause the processor 702 to return a data record having a hash validation key that matches the validation value. The foregoing is just one example configuration of module 705 in accordance with the present disclosure. Furthermore, when a hardware processor executes instructions to perform "operations," this could include the hardware processor performing the operations directly and/or facilitating, directing, or cooperating with another hardware device or component, e.g., a co-processor and the like, to perform the operations.

[0065] The processor executing the machine readable instructions relating to the above described method(s) can be perceived as a programmed processor or a specialized processor. As such, the present module 705 for maintaining a hash table, including associated data structures, of the present disclosure can be stored on a tangible or physical (broadly non-transitory) computer-readable

storage device or medium, e.g., volatile memory, non-volatile memory, ROM memory, RAM memory, magnetic or optical drive, device or diskette and the like. Furthermore, the computer-readable storage device may comprise any physical device or devices that provide the ability to store information such as data and/or instructions to be accessed by a processor or a computing device such as a computer or an application server.

[0066] It will be appreciated that variants of the above-disclosed and other features and functions, or alternatives thereof, may be combined into many other different systems or applications. Various presently unforeseen or unanticipated alternatives, modifications, or variations therein may be subsequently made, which are also intended to be encompassed by the following claims.

What is claimed is:

1. A method, comprising:

applying, by a processor, a first hash function to an input key to determine a hash index, wherein the input key is associated with a data record;

applying, by the processor, a second hash function to the input key to generate a hash validation key; and

storing, by the processor, the hash validation key and the data record in a bucket for the hash index in a hash table.

2. The method of claim 1, further comprising:

receiving a search key, wherein the search key corresponds to the input key;

applying the first hash function to the search key to generate a hash value, wherein the hash value corresponds to the hash index; and

locating the bucket for the hash index in the hash table using the hash value.

3. The method of claim 2, further comprising:

applying the second hash function to the search key to generate a validation value;

comparing the validation value to at least the hash validation key; and

retrieving the data record when the validation value is equivalent to the hash validation key.

4. The method of claim 1, wherein the bucket for the hash index comprises:

a plurality of hash validation keys; and

a plurality of data records associated with the plurality of hash validation keys.

5. The method of claim 4, wherein the plurality of hash validation keys and the plurality of data records associated with the plurality of hash validation keys

are associated with different input keys of a plurality input keys that are associated with the hash index.

6. The method of claim 1, wherein the hash table comprises a network routing table.

7. The method of claim 1, wherein the hash validation key is of a greater size than the hash index, wherein the hash validation key is of a lesser size that the input key.

8. The method of claim 1, wherein the first hash function and the second hash function are different hash functions.

9. A device, comprising:

a processor; and

a non-transitory computer-readable medium storing instructions which, when executed by the processor, cause the processor to:

apply a first hash function to a search key to generate a hash value;

locate a bucket for a hash index in a hash table using the hash value, wherein the hash index corresponds to the hash value;

apply a second hash function to the search key to generate a validation value;

compare the validation value to a hash validation key in the bucket for the hash index; and

retrieve a data record associated with the hash validation key when the validation value corresponds to the hash validation key.

10. The device of claim 9, wherein the bucket for the hash index includes a plurality of hash validation keys and a plurality of data records associated with the plurality of hash validation keys.

11. The device of claim 9, wherein the hash validation key is generated by applying the second hash function to an input key, wherein the input key is equivalent to the search key.

12. A device comprising:

a hardware memory storing a hash table, wherein the hash table comprises:

a bucket associated with a hash index, wherein the bucket comprises:

a hash validation key, wherein the hash validation key is associated with an input key; and

a data record associated with the hash validation key; and

a hardware logic to:

apply a first hash function to a search key to generate a hash value, wherein the search key corresponds to the input key, wherein the hash value corresponds to the hash index; and

locate the bucket for the hash index in the hash table using the hash value.

13. The device of claim 12, wherein the bucket associated with the hash index further comprises:

a plurality of hash validation keys; and

a plurality of data records associated with the plurality of hash validation keys.

14. The device of claim 12, wherein the hardware logic is further to:

apply a second hash function to the search key to generate a validation value;

compare the validation value to at least the hash validation key of the bucket associated with the hash index; and

retrieve a data record that is associated with the hash validation key when the validation value is equivalent to the hash validation key.

15. The device of claim 12, wherein the hash validation key is associated with the input key by applying the second hash function to the input key to generate the hash validation key.

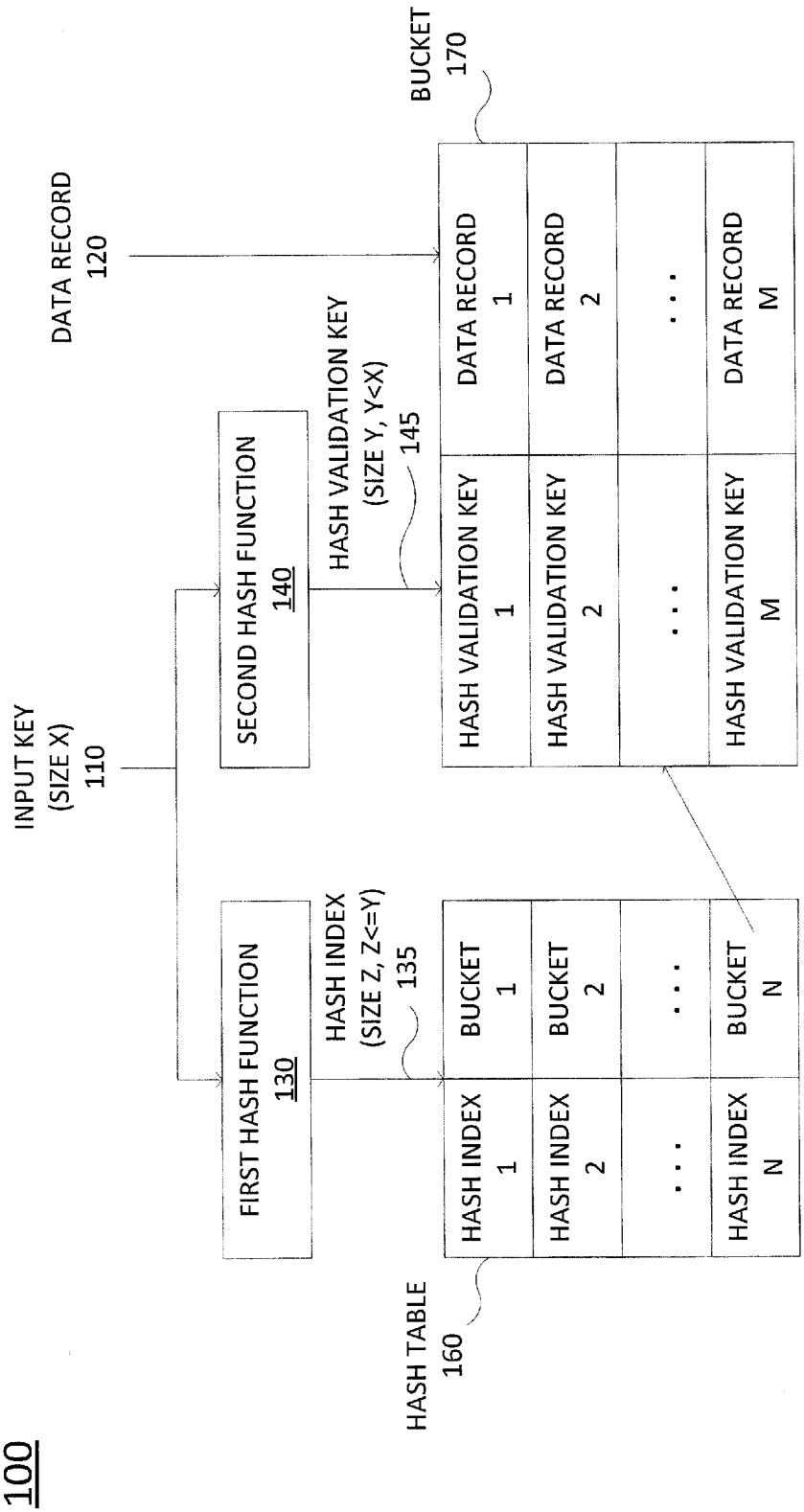

100
INPUT KEY
(SIZE X)
110
DATA RECORD
120
FIRST HASH FUNCTION
130
SECOND HASH FUNCTION
140
HASH INDEX
(SIZE Z, Z<=Y)
135
HASH VALIDATION KEY
(SIZE Y, Y<X)
145
HASH TABLE
160
BUCKET
170
HASH INDEX 1
HASH INDEX 2
HASH INDEX N
BUCKET 1
BUCKET 2
BUCKET N
HASH VALIDATION KEY 1
HASH VALIDATION KEY 2
HASH VALIDATION KEY M
DATA RECORD 1
DATA RECORD 2
DATA RECORD M

FIG. 1

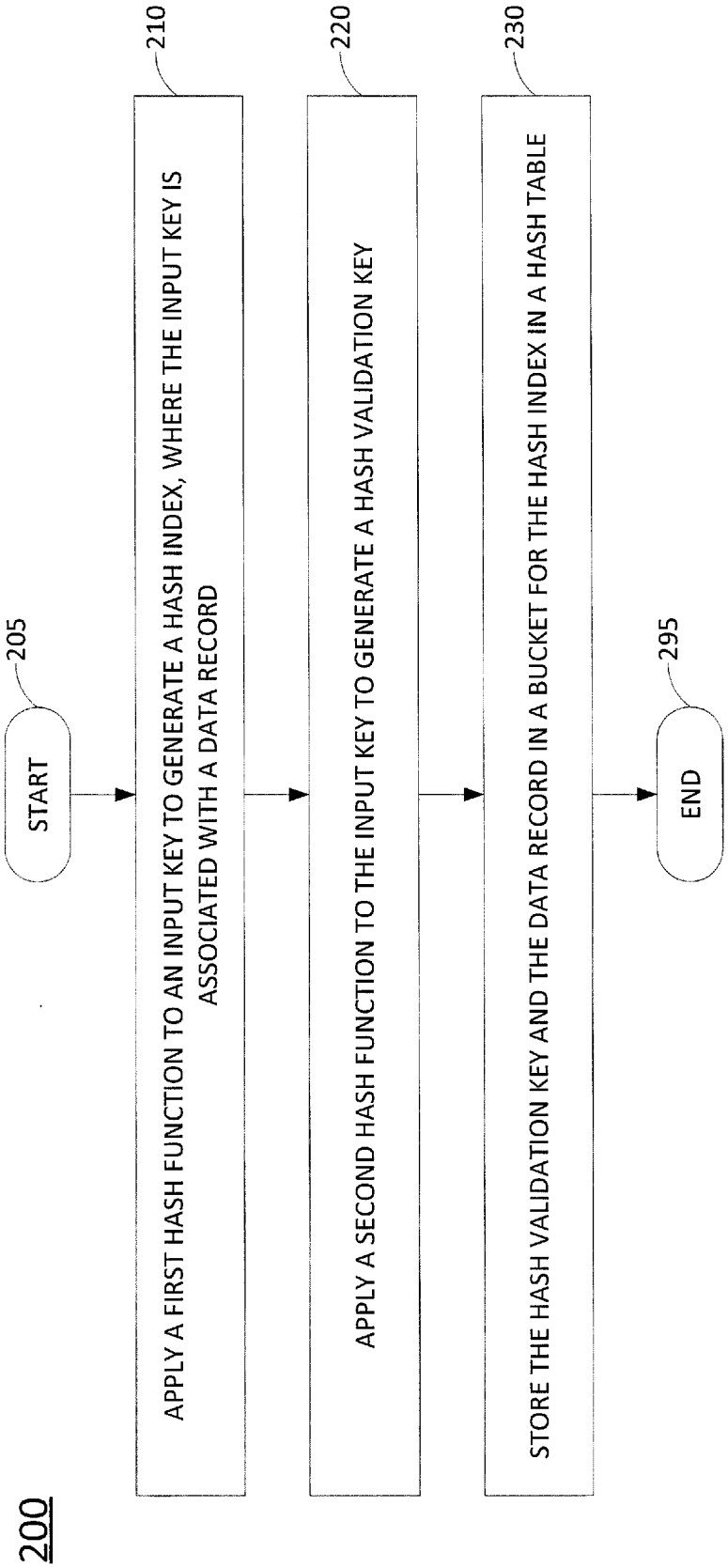

200
205
START
210
APPLY A FIRST HASH FUNCTION TO AN INPUT KEY TO GENERATE A HASH INDEX, WHERE THE INPUT KEY IS ASSOCIATED WITH A DATA RECORD
220
APPLY A SECOND HASH FUNCTION TO THE INPUT KEY TO GENERATE A HASH VALIDATION KEY
230
STORE THE HASH VALIDATION KEY AND THE DATA RECORD IN A BUCKET FOR THE HASH INDEX IN A HASH TABLE
295
END

FIG. 2

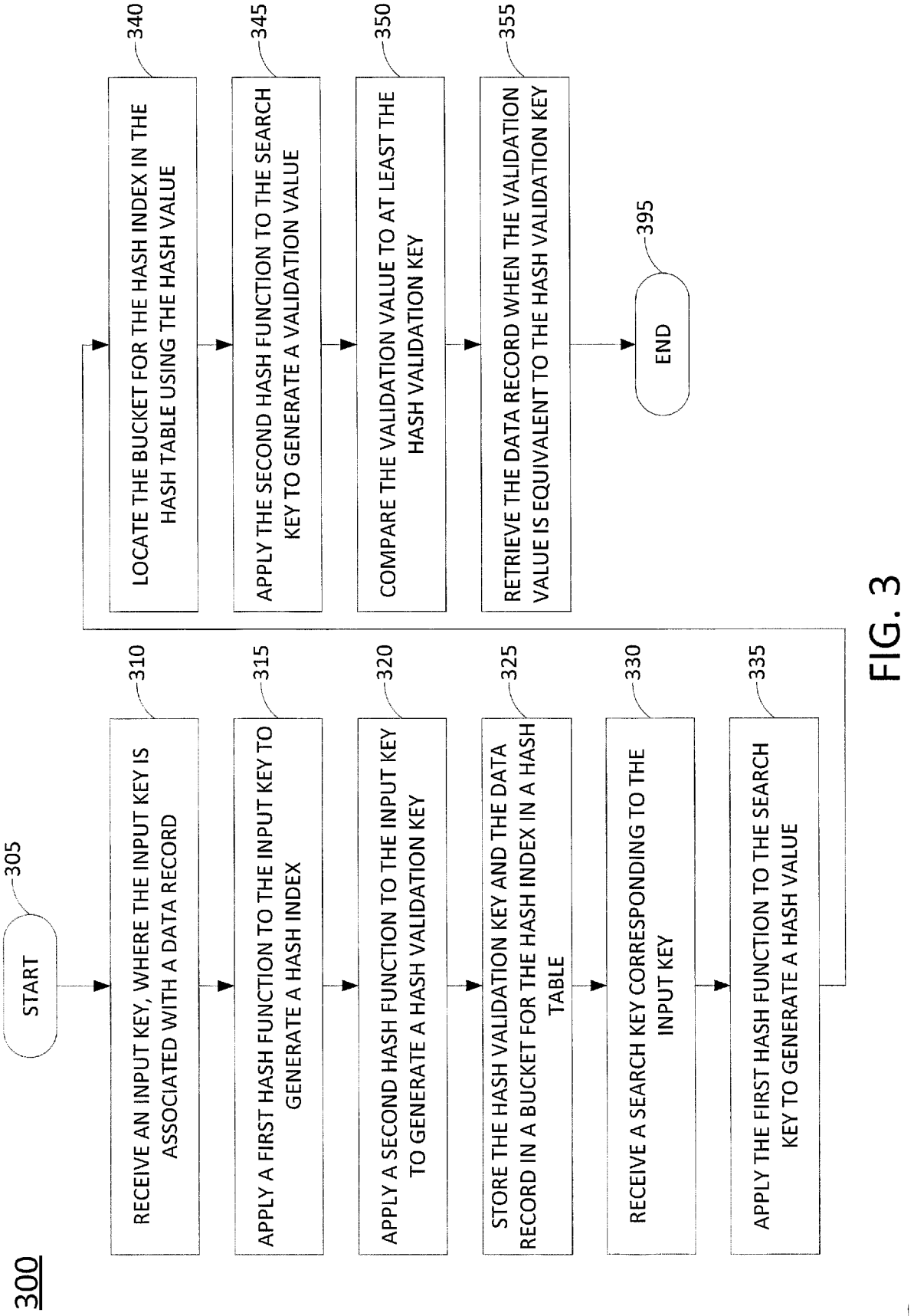

300
START
305
RECEIVE AN INPUT KEY, WHERE THE INPUT KEY IS ASSOCIATED WITH A DATA RECORD
310
APPLY A FIRST HASH FUNCTION TO THE INPUT KEY TO GENERATE A HASH INDEX
315
APPLY A SECOND HASH FUNCTION TO THE INPUT KEY TO GENERATE A HASH VALIDATION KEY
320
STORE THE HASH VALIDATION KEY AND THE DATA RECORD IN A BUCKET FOR THE HASH INDEX IN A HASH TABLE
325
RECEIVE A SEARCH KEY CORRESPONDING TO THE INPUT KEY
330
APPLY THE FIRST HASH FUNCTION TO THE SEARCH KEY TO GENERATE A HASH VALUE
335
LOCATE THE BUCKET FOR THE HASH INDEX IN THE HASH TABLE USING THE HASH VALUE
340
APPLY THE SECOND HASH FUNCTION TO THE SEARCH KEY TO GENERATE A VALIDATION VALUE
345
COMPARE THE VALIDATION VALUE TO AT LEAST THE HASH VALIDATION KEY
350
RETRIEVE THE DATA RECORD WHEN THE VALIDATION VALUE IS EQUIVALENT TO THE HASH VALIDATION KEY
355
END
395

FIG. 3

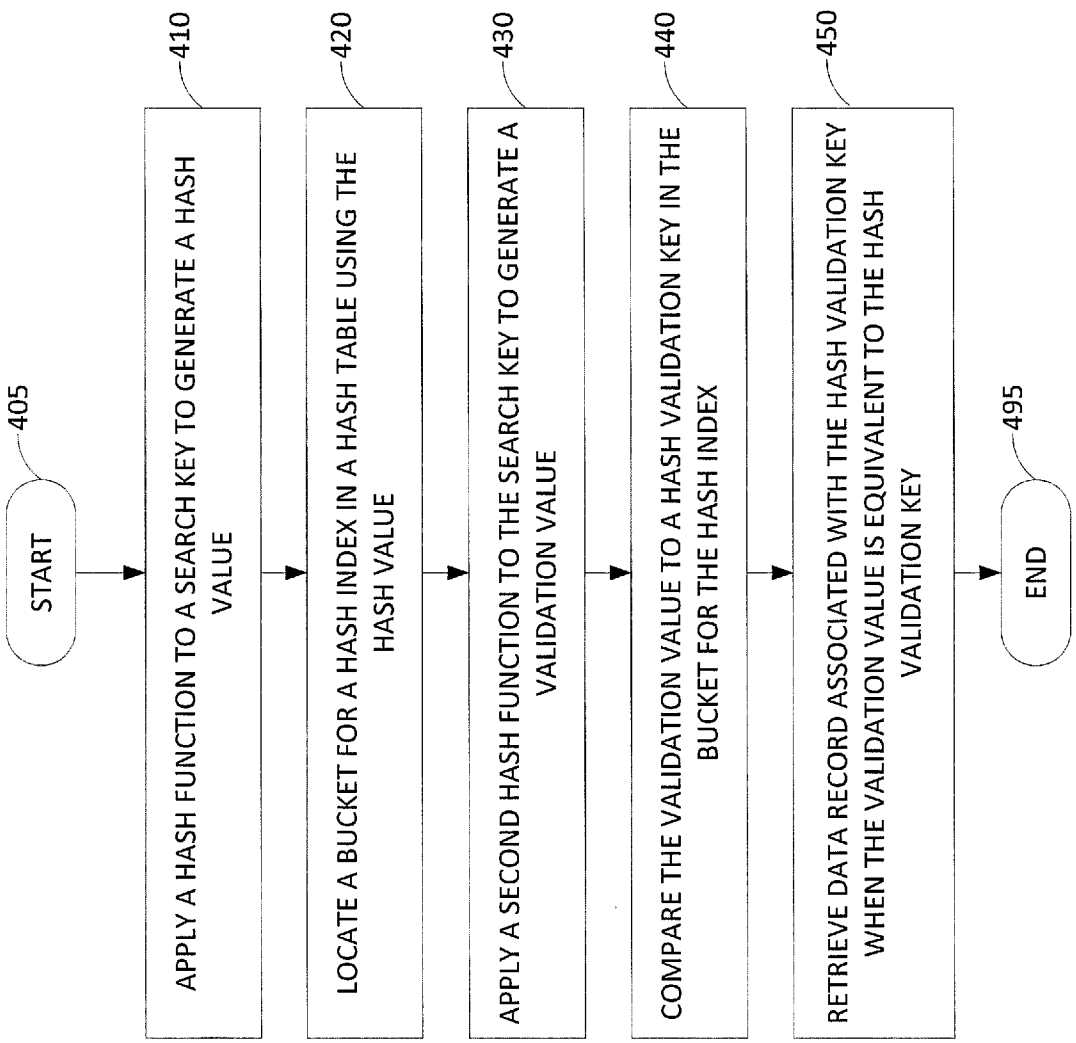
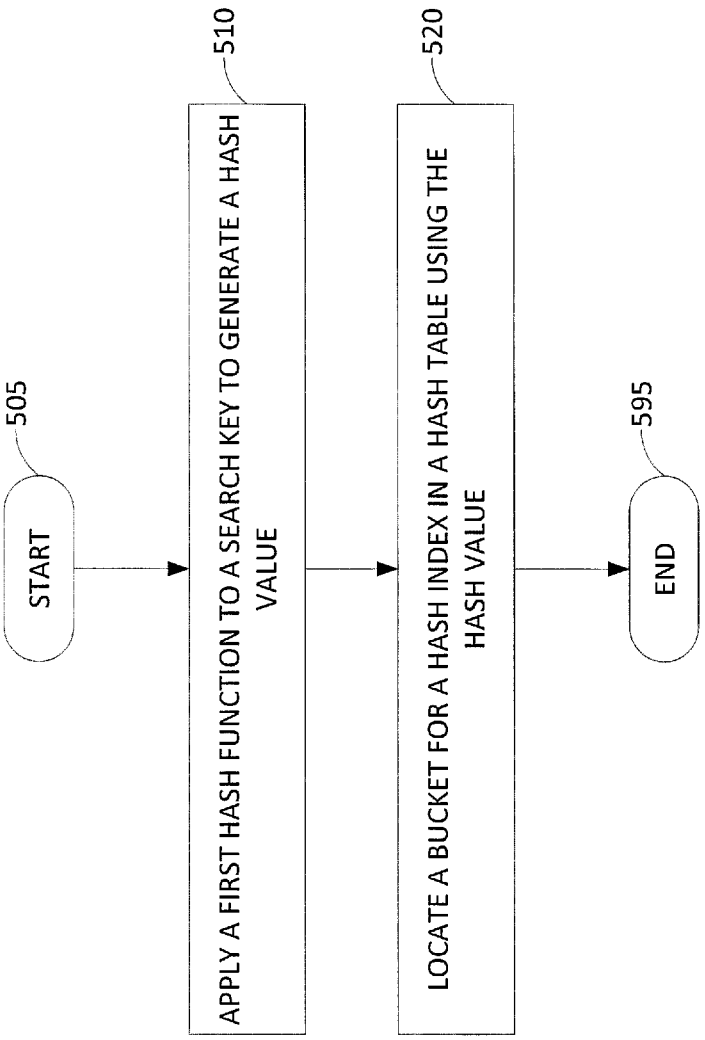

400
405
START
410
APPLY A HASH FUNCTION TO A SEARCH KEY TO GENERATE A HASH VALUE
420
LOCATE A BUCKET FOR A HASH INDEX IN A HASH TABLE USING THE HASH VALUE
430
APPLY A SECOND HASH FUNCTION TO THE SEARCH KEY TO GENERATE A VALIDATION VALUE
440
COMPARE THE VALIDATION VALUE TO A HASH VALIDATION KEY IN THE BUCKET FOR THE HASH INDEX
450
RETRIEVE DATA RECORD ASSOCIATED WITH THE HASH VALIDATION KEY WHEN THE VALIDATION VALUE IS EQUIVALENT TO THE HASH VALIDATION KEY
495
END

FIG. 4


505
START
510
APPLY A FIRST HASH FUNCTION TO A SEARCH KEY TO GENERATE A HASH VALUE
520
LOCATE A BUCKET FOR A HASH INDEX IN A HASH TABLE USING THE HASH VALUE
595
END

500

FIG. 5

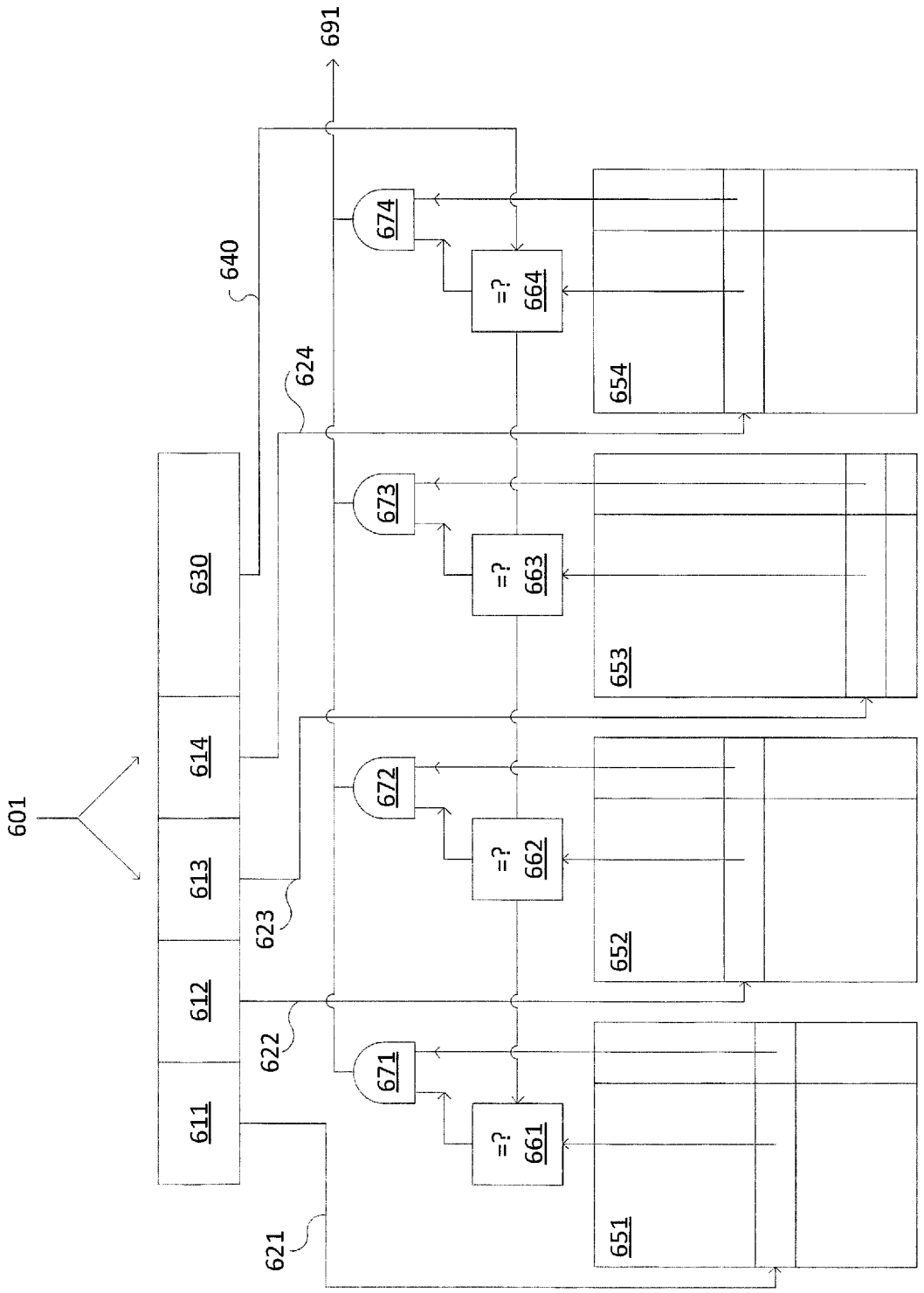

600
601
611
612
613
614
621
622
623
624
630
640
651
652
653
654
661
662
663
664
671
672
673
674
691
=?

FIG. 6

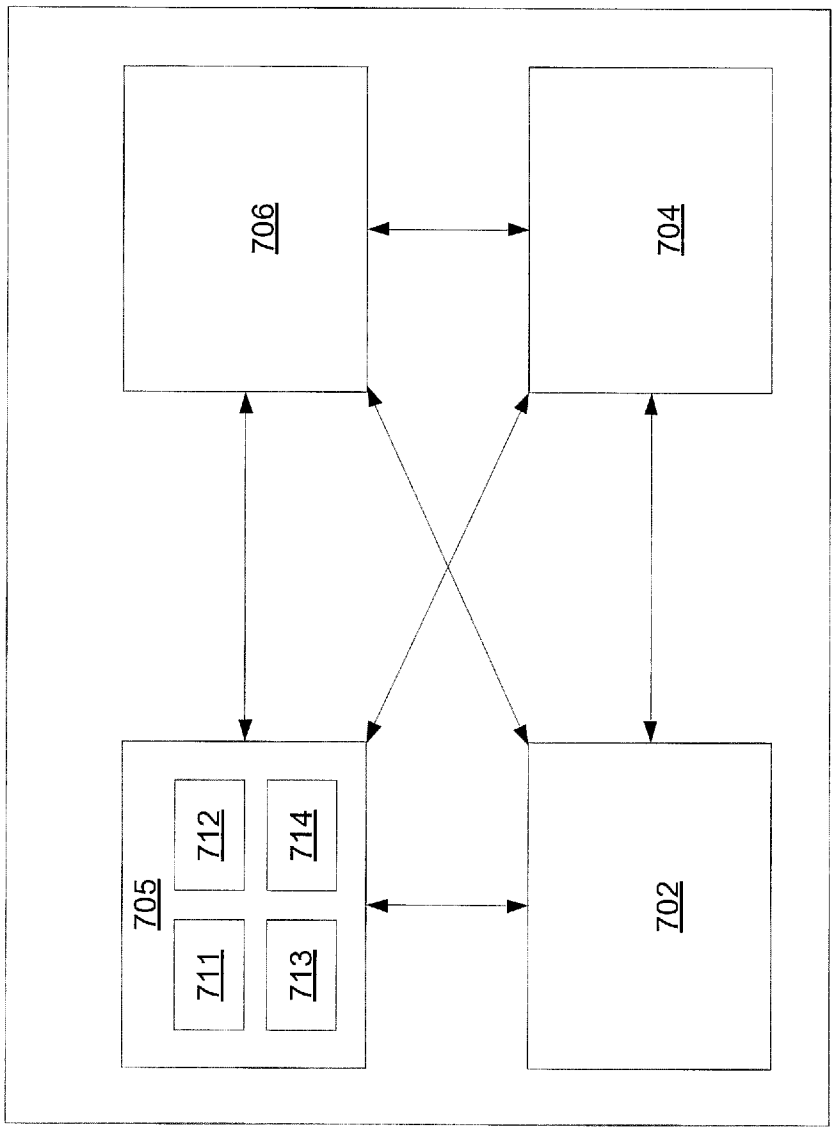


FIG. 7

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2015/042447

A. CLASSIFICATION OF SUBJECT MATTER

G06F 17/30(2006.01)i

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)
G06F 17/30; G06F 12/10; H04L 12/56

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
Korean utility models and applications for utility models
Japanese utility models and applications for utility models

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
eKOMPASS(KIPO internal) & Keywords: second hash function, hash index, hash validation key, bucket, and similar terms.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2013-0151810 A1 (AUTONOMY, INC.) 13 June 2013 See paragraphs [0011], [0034]-[0038], [0041]-[0046], [0054]-[0059], and [0076]-[0078]; claims 1 and 9; and figures 1a-2b.	1-5,7-15
Y		6
Y	US 2014-0064093 A1 (INTERNATIONAL BUSINESS MACHINES CORPORATION) 06 March 2014 See paragraph [0050] and figures 4-5.	6
A	US 2014-0372392 A1 (INTERNATIONAL BUSINESS MACHINES CORPORATION) 18 December 2014 See paragraphs [0043]-[0044] and figure 7.	1-15
A	EP 1970821 A1 (BROADCOM CORPORATION) 17 September 2008 See paragraphs [0024]-[0041] and figures 3-5.	1-15
A	US 2015-0058595 A1 (ORACLE INTERNATIONAL CORPORATION) 26 February 2015 See paragraphs [0038]-[0039]; and figure 1.	1-15

☐ Further documents are listed in the continuation of Box C.

☒ 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
26 April 2016 (26.04.2016)	**27 April 2016 (27.04.2016)**

Name and mailing address of the ISA/KR
International Application Division
Korean Intellectual Property Office
189 Cheongsa-ro, Seo-gu, Daejeon Metropolitan City, 35208, Republic of Korea
Facsimile No. +82-42-481-8578

Authorized officer
NHO, Ji Myong
Telephone No. +82-42-481-8528

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.
PCT/US2015/042447

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2013-0151810 A1	13/06/2013	GB 2479691 A	19/10/2011
		GB 2479691 B	26/03/2014
		US 2010-217953 A1	26/08/2010
		US 8397051 B2	12/03/2013
		US 8806175 B2	12/08/2014
		WO 2010-096750 A2	26/08/2010
		WO 2010-096750 A3	11/11/2010
US 2014-0064093 A1	06/03/2014	US 2014-0064276 A1	06/03/2014
US 2014-0372392 A1	18/12/2014	US 2014-214855 A1	31/07/2014
EP 1970821 A1	17/09/2008	US 2008-0229056 A1	18/09/2008
		US 8266116 B2	11/09/2012
US 2015-0058595 A1	26/02/2015	None	