US009270592B1

(12) **United States Patent**
Sites

(10) **Patent No.: US 9,270,592 B1**
(45) **Date of Patent: Feb. 23, 2016**

(54) **HASH COLLISION AVOIDANCE IN NETWORK ROUTING**

(71) Applicant: **Google Inc.**, Mountain View, CA (US)

(72) Inventor: **Richard Lee Sites**, Mountain View, CA (US)

(73) Assignee: **Google Inc.**, Mountain View, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 156 days.

(21) Appl. No.: **14/163,562**

(22) Filed: **Jan. 24, 2014**

(51) **Int. Cl.**
H04L 12/743 (2013.01)
H04L 12/741 (2013.01)

(52) **U.S. Cl.**
CPC ***H04L 45/7453*** (2013.01); ***H04L 45/745*** (2013.01); ***H04L 45/7457*** (2013.01)

(58) **Field of Classification Search**
CPC ... H04L 45/745; H04L 45/7453; H04L 45/38; H04L 45/74; H04L 45/7457; H04L 47/2483; H04L 47/10; H04L 47/2441; H04L 69/22; H04L 49/3009; H04L 12/5601
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

7,002,965	B1 *	2/2006	Cheriton	370/395.32
7,219,211	B1 *	5/2007	Greene et al.	711/216
7,290,084	B2	10/2007	Miller et al.	
7,653,670	B2	1/2010	Hasan et al.	
2008/0133494	A1 *	6/2008	Won-Kyoung et al.	707/4
2011/0273987	A1 *	11/2011	Schlansker et al.	370/235
2013/0086017	A1	4/2013	Chao et al.	

FOREIGN PATENT DOCUMENTS

EP	1 551 141	B1 7/2005
EP	2 512 073	A1 10/2012

OTHER PUBLICATIONS

Nie, et al., IP Address Lookup Using a Dynamic Hash Function, pp. 1642-1647, Canadian Conference on Electrical and Computer Engineering, IEEE, May 2005.
Pagiamtzis, et al., Content Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey, pp. 712-727, IEEE Journal of Solid-State Circuits, vol. 41. No. 3, Mar. 2006.

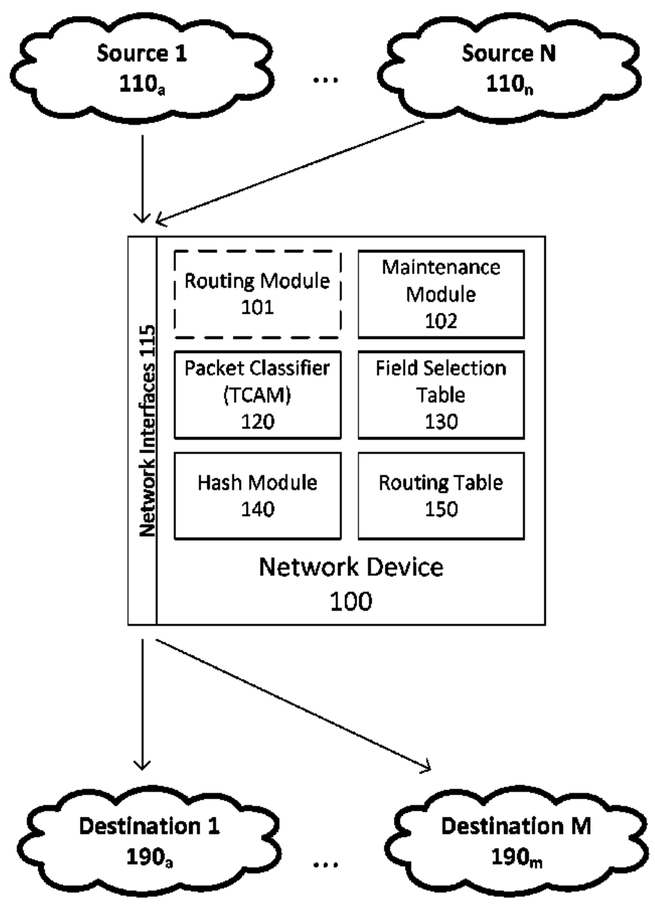
* cited by examiner

Primary Examiner — Kwang B Yao
Assistant Examiner — Ricardo Castaneyra
(74) *Attorney, Agent, or Firm* — Edward A. Gordon; Foley & Lardner LLP

(57) **ABSTRACT**

Network device and method for routing a packet and setting up a new flow. The device includes a packet classifier, a field-selection table, a hash module, and a routing table. A packet is routed by finding an entry in the field-selection table using the packet classifier, selecting bits from the packet based on the entry in the field-selection table, and hashing the selected bits along with an identifier from the packet classifier or the field-selection table, using the hash module. The hash result is used to locate instructions in the routing table. When setting up a new flow, the hash module result may point to an existing entry in the routing table. In such instances, a new entry is added to the packet classifier, such that the hash module will produce a different result that points to an available entry in the routing table.

23 Claims, 7 Drawing Sheets



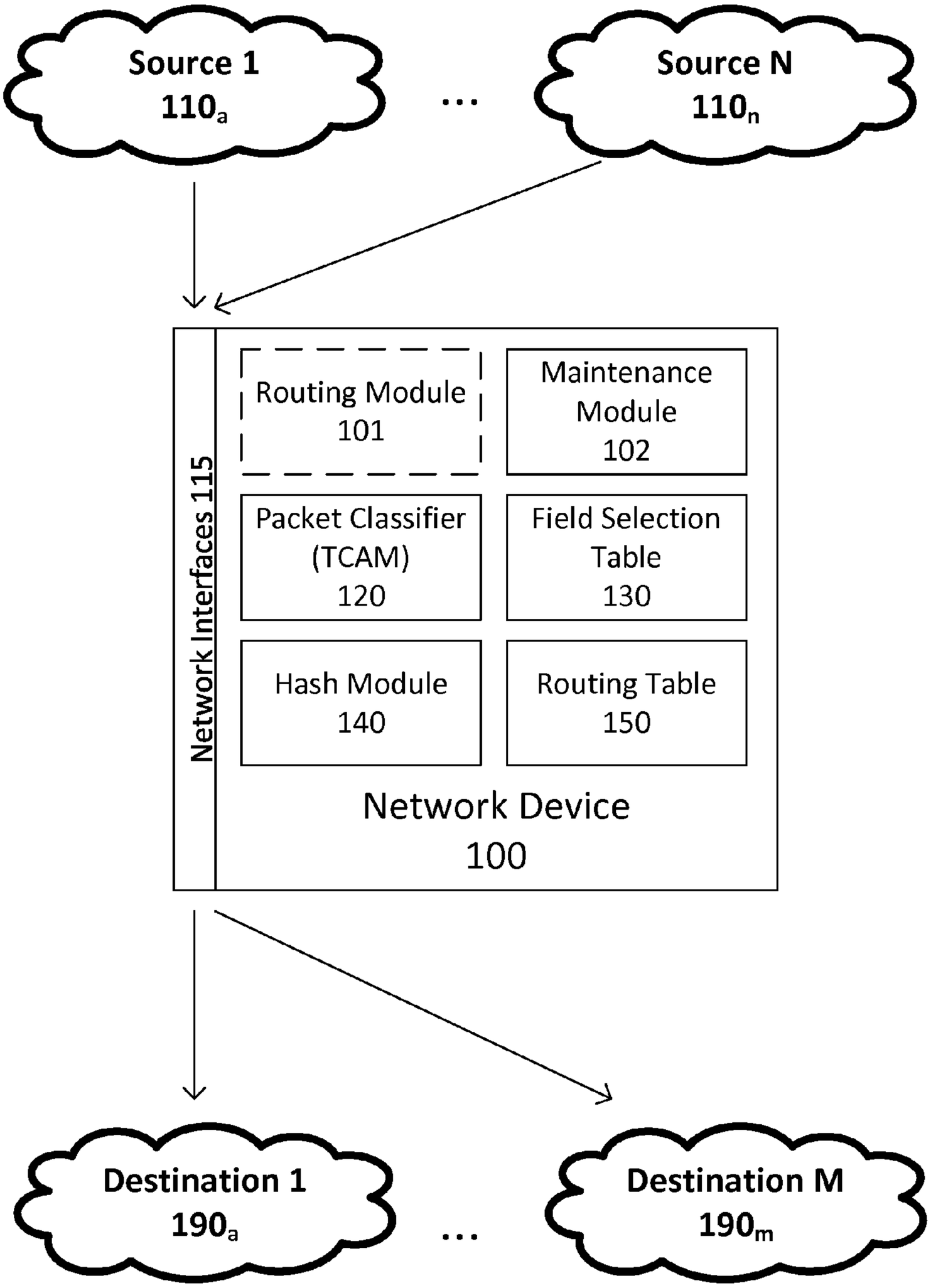

Source 1
110a
...
Source N
110n
Network Interfaces 115
Routing Module
101
Maintenance
Module
102
Packet Classifier
(TCAM)
120
Field Selection
Table
130
Hash Module
140
Routing Table
150
Network Device
100
Destination 1
190a
...
Destination M
190m

Figure 1

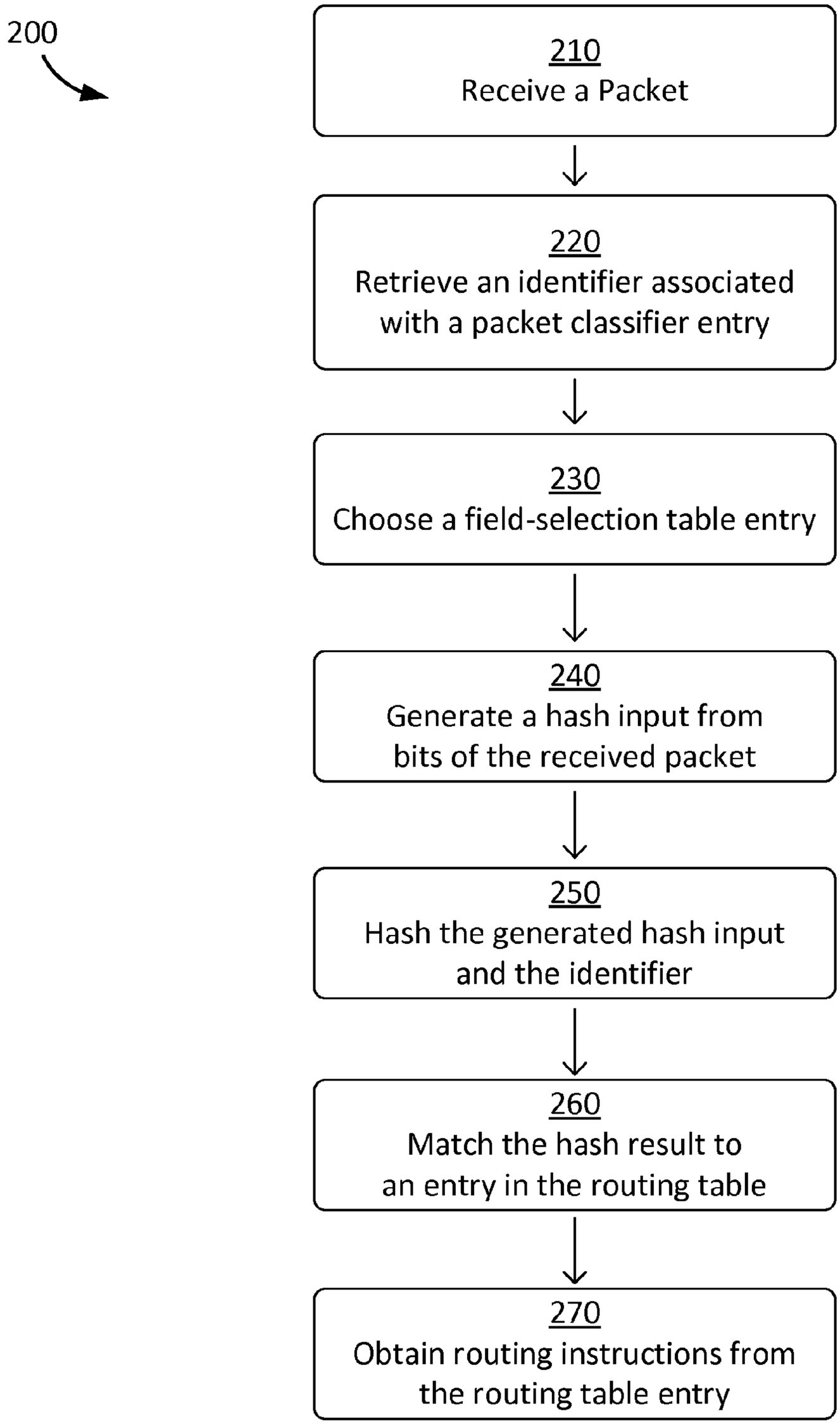

200
210
Receive a Packet
220
Retrieve an identifier associated with a packet classifier entry
230
Choose a field-selection table entry
240
Generate a hash input from bits of the received packet
250
Hash the generated hash input and the identifier
260
Match the hash result to an entry in the routing table
270
Obtain routing instructions from the routing table entry

Figure 2

314

	0000 0123	0000 4567	0011 8901	1111 2345	1111 6789	2222 0123	2222 4567	2233 8901
0	*End of New Packet Preamble and Delimiter Bits*				MAC Destination Address (6 Octets)			
1	MAC Destination (Continued)							
2	MAC Source Address (6 Octets)							
3	MAC Source Address (Cont.)				Ethertype			
4	IP Ver.	IHL *(length)*	DSCP (QoS)	ECN	Packet Length			
5	Identification				Flags	Fragment Offset		
6	Time to Live (TTL)		Protocol		Header Checksum			
7	Source IP Address							
8	Destination IP Address							
9	Source Port				Destination Port			
10	Sequence Number							
11	Acknowledgment Number							
12	Data Offset	Not Used	Control Flags		Window Size			
13	Checksum				Urgent Pointer / Options / Zero Pad			
14	Encapsulated Data ...							

320 340 360 380

Figure 3A

316

	0000 0123	0000 4567	0011 8901	1111 2345	1111 6789	2222 0123	2222 4567	2233 8901
0	*End of New Packet Preamble and Delimiter Bits*				MAC Destination Address (6 Octets)			
1	MAC Destination (Continued)							
2	MAC Source Address (6 Octets)							
3	MAC Source Address (Cont.)				Ethertype			
4	IP Ver.	Traffic Class	Flow Label					
5	Payload Length				Next Header	Hop Limit		
6	Source IP Address (16 Octets)							
7-9	Source IP Address (Continued)							
10	Destination IP Address (16 Octets)							
11-13	Destination IP Address (Continued)							
14	Source Port				Destination Port			
15	Sequence Number							
16	Acknowledgment Number							
17	Data Offset	Not Used	Control Flags		Window Size			
18	Checksum				Urgent Pointer / Options / Zero Pad			
19	Encapsulated Data ...							

320

350

370

390

Figure 3B

420

421	0	**4**5XX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
422	1	**6**XXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
423	2	**4**5XX XXXX XXXX XXXX XX**06** XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
424	3	**6**XXX XXXX XXXX **06**XX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
425	4	**4**5XX XXXX XXXX XXXX XX**06** XXXX 0102 0304 0506 0708 XXXX 0050 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
426	5	**6**XXX XXXX XXXX **06**XX 0102 0304 0506 0708 0910 1112 1314 1516 1718 1920 2122 2324 2526 2728 2930 3132

430

431	0	0000000000**11111111**0000000000000000000
432	1	00000000111111111111111111111111111111
433	2	0000000000**1111111111**0000000000000000
434	3	00000000111111111111111111111111111111
435	4	0000000000**1111111111**0000000000000000
436	5	00000000111111111111111111111111111111

Figure 4

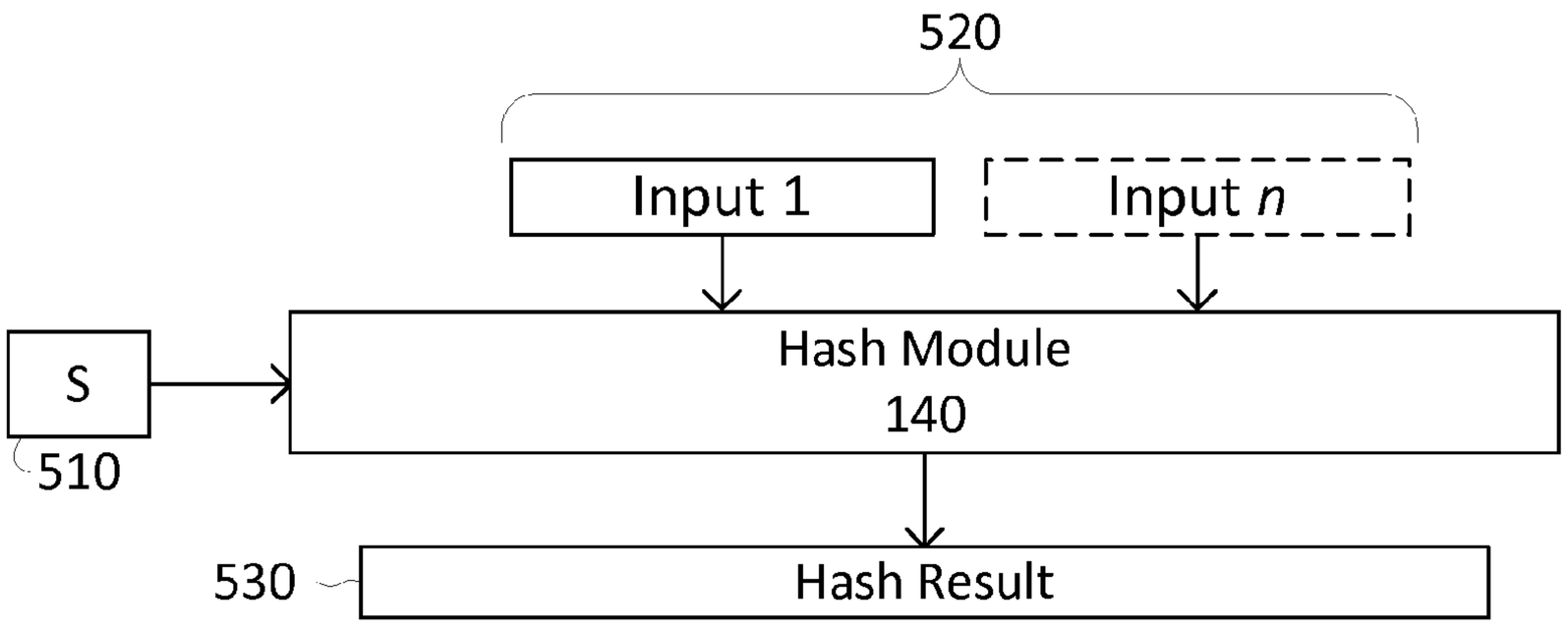


Figure 5

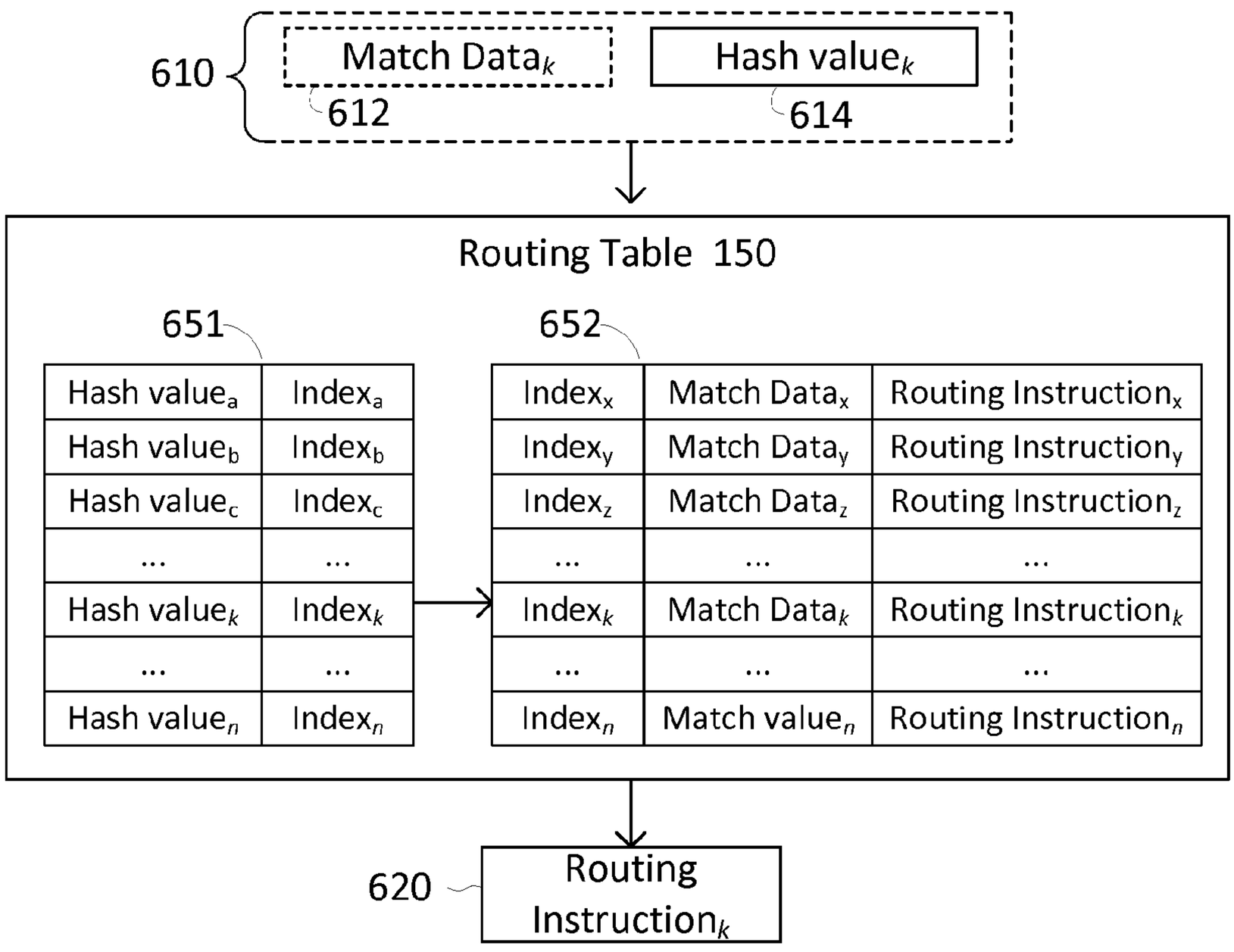


Figure 6

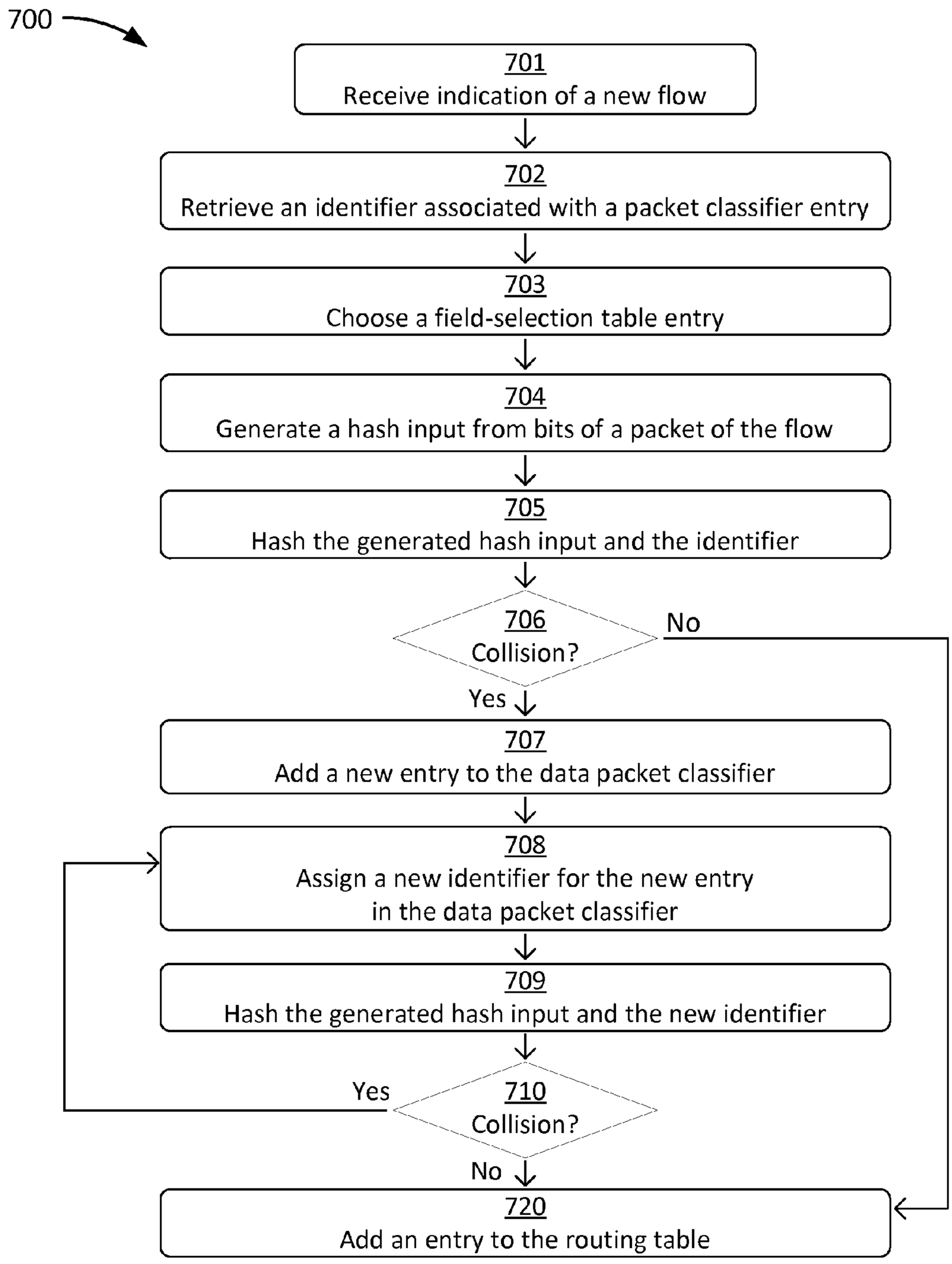

700
701
Receive indication of a new flow
702
Retrieve an identifier associated with a packet classifier entry
703
Choose a field-selection table entry
704
Generate a hash input from bits of a packet of the flow
705
Hash the generated hash input and the identifier
706
Collision?
No
Yes
707
Add a new entry to the data packet classifier
708
Assign a new identifier for the new entry
in the data packet classifier
709
Hash the generated hash input and the new identifier
710
Collision?
Yes
No
720
Add an entry to the routing table

Figure 7

HASH COLLISION AVOIDANCE IN NETWORK ROUTING

BACKGROUND

Network devices, e.g., switches, routers, and filters, play an important role in data communications. Countless amounts of data are transferred as data packets transmitted over different networks across the world. Each data packet must be channeled from its source to its destination, and network devices play an important role in directing the traffic. In order to limit latency, it is important that network device can route traffic efficiently and accurately.

There are many different components within network devices. One component found in some network routers is a routing table. A routing table stores handling instructions, e.g., a next-hop destination or an egress port identifier, for data packets that are to be routed through the device. Each entry in the table corresponds to a route. These routing tables are sometimes stored using volatile integrated circuit memory, e.g., SRAMs. Generally, routing tables have limited capacities. In order to stay within the limited capacity, it is important that network devices store a minimal amount of data for each route in the routing table.

SUMMARY

In one aspect, the disclosure relates to a network device. The network device includes a packet classifier, a field-selection table, a hash module, a routing table, and a routing module configured to route a packet. The routing module is configured to determine an entry in the packet classifier using a received packet, retrieve an identifier associated with the determined packet classifier entry, choose a field-selection table entry using the retrieved identifier, generate a hash module input by identifying a set of bits of the packet based on the chosen field-selection table entry, cause the hash module to compute a hash result based on the generated hash module input and based on the retrieved identifier, match the hash result to an entry in the routing table, and obtain processing data for the data packet from the matching routing table entry.

In another aspect, the disclosure relates to a method. The method includes receiving a packet from a source, determining an entry in a packet classifier using the received packet, retrieving an identifier associated with the determined packet classifier entry, and choosing a field-selection table entry using the retrieved identifier. The method further includes generating a hash module input by identifying a set of bits of the packet based on the chosen field-selection table entry, causing a hash module to compute a hash result based on the generated hash module input and the retrieved identifier, and matching the hash result to an entry in a routing table. The method includes obtaining processing data for the packet from the matching routing table entry.

In another aspect, the disclosure relates to a non-transitory computer-readable medium storing computer-readable instructions that, when executed by one or more computing devices, cause at least one of the one or more computing devices to perform operations that include receiving a packet from a source, determining an entry in a packet classifier using the received packet, retrieving an identifier associated with the determined packet classifier entry, and choosing a field-selection table entry using the retrieved identifier. The operations further include generating a hash module input by identifying a set of bits of the packet based on the chosen field-selection table entry, causing a hash module to compute a hash result based on the generated hash module input and the retrieved identifier, and matching the hash result to an entry in a routing table. The operations further include obtaining processing data for the packet from the matching routing table entry.

BRIEF DESCRIPTION OF THE DRAWINGS

These diagrams and flowcharts are not intended to limit the scope of the present teachings in any way. The devices and methods may be better understood from the following illustrative description with reference to the following figures in which:

FIG. **1** is a diagram of an example network device;

FIG. **2** is a flowchart of an example method for routing a packet using the network device shown in FIG. **1**;

FIG. **3**A is the layout for a typical TCP/IPv4 packet header, including the Ethernet frame;

FIG. **3**B is the layout for a typical TCP/IPv6 packet header, including the Ethernet frame;

FIG. **4** shows an example packet classifier and an example field-selection table, as used by the network device shown in FIG. **1**;

FIG. **5** shows an example of the hash module used in the network device shown in FIG. **1**;

FIG. **6** shows an example routing table; and

FIG. **7** is a flowchart of a method for initiating a new flow using the network device shown in FIG. **1**.

Like reference numbers and designations in the various drawings indicate like elements.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The various concepts introduced above and discussed in greater detail below may be implemented in any of numerous ways, as the described concepts are not limited to any particular manner of implementation. Examples of specific implementations and applications are provided primarily for illustrative purposes.

FIG. **1** shows a diagram of an example network device **100**. The illustrated network device **100** includes a routing module **101**, a maintenance module **102**, a packet classifier **120**, a field-selection table **130**, a hash module **140**, and a routing table **150**. The network device **100** includes network interfaces **115**, through which the network device **100** can receive data packets from one or more of sources $\mathbf{110}_a$-$\mathbf{110}_n$ (generally "source **110**") and can forward the data packets to any number of destinations $\mathbf{190}_a$-$\mathbf{190}_m$ (generally "destination **190**"). The sources **110** and destinations **190** can each be a server, a computer, a processor, a mobile device such as a smart phone or tablet, a set-top device, or any other suitable network device. In general, a set of data packets forming a communication between a source **110** and a destination **190** constitutes a flow. In a two-way exchange between two end points, the end points act as both sources and destinations. Thus, a source **110** in one flow may be a destination **190** for another flow. In some implementations, the network device **100** is an end-host device. For example, the network device **100** can be a server that receives requests from one or more of sources $\mathbf{110}_a$-$\mathbf{110}_n$, and responds to each request, e.g., by originating data packets to transmit, or causing another server to transmit, data packets to the requesting source **110**. In some implementations, the network device **100** uses the routing module **101** to direct packets to another network device, to one of multiple computing processors or modules in the network device **100**, to a particular core of a multi-core proces-

sor, to a particular hypervisor, virtual computer, or operating system, or to a specific application or service instance executing on an end-host server.

In some implementations, one or more of the various components of the network device **100** are implemented as hardware in an integrated circuit, such as an application specific integrated circuit (ASIC) or field programmable gate array (FPGA). In some implementations, one or more of the various components of the network device **100** are implemented as computer executable instructions that are executed by one or more special purpose or general purpose processors. In some implementations, one or more of the various components of the network device **100** are implemented as a mix of special purpose circuits and computer executable instructions that are executed by a processor. For example, in some implementations, the packet classifier **120** is implemented as a ternary content-addressable memory (TCAM) circuit, the hash module **140** is implemented as a special purpose hashing circuit, the field-selection table **130** and the routing table **150** are implemented using random access memory (RAM), and the routing module **101** and the maintenance module **102** are implemented as computer executable instructions that are executed by a general purpose computing processor. In some implementations, one or more of the components or modules are remote from the network device **100**. For example, in some implementations, the maintenance module **102** is implemented in a separate controller, e.g., in a controller for a software-defined network (SDN). In some implementations, the field-selection table **130** and/or the routing table **150** are implemented using volatile memory, such as DRAM, SRAM, FLASH, or other integrated circuit memory. In some implementations, a computing processor is multi-core. In some implementations, the network device **100** is implemented with multiple computing processors.

In operation, the network device **100** receives a packet and determines how to handle the received packet, e.g., by identifying and forwarding the packet to a next-hop network device. The process is generally governed by the routing module **101**. Each packet begins with a sequence of header bits that identify a communication protocol for the packet and any additional packet information in accordance with the communication protocol, e.g., addressing information for the packet. Two example formats are illustrated in FIGS. **3**A and **3**B, described below. The routing module **101** processes the initial bits of the packet to identify an entry in the routing table **150** with handling instructions for the packet. For example, the handling instruction may specify a network interface **115** to use for forwarding the packet. In some implementations, the network device **100** includes multiple computing processors and the handling instructions specify which processor to use for processing the packet. In some implementations, the network device **100** includes multiple service instances for handling network traffic, and the handling instruction may identify a particular service instance to handle the received packet.

The routing module **101** identifies the entry in the routing table **150** by processing information in the header of the packet. In general, each communication protocol defines an assignment of the header bits into meaningful fields, where each field has a number of bits and the values of those bits is the value of the respective header field. The meaning of a header field value is understood within the context of the communication protocol used. The number of bits in a field is typically specified by either the protocol or by the contents of another field in the header. For example, an address in IPv4 is represented by thirty-two bits, starting at the ninety seventh bit of the IPv4 header (as shown, for example, in the IPv4 header **340** in FIG. **3**A), whereas an address in IPv6 is represented by one hundred twenty eight bits, starting at the sixty fifth bit of the IPv6 header (as shown, for example, in the IPv6 header **350** in FIG. **3**B). Furthermore, a typical data packet uses multiple protocols in a layered manner. For example, the transmission control protocol (TCP) defines a communication verification protocol that relies on a separate addressing protocol such as the Internet Protocol (IP), which, in turn, relies on a framing protocol such as Ethernet. Thus, a single TCP/IP packet has at least three layers of header information. For example, as shown in FIG. **3**A, a typical TCP/IPv4 packet, after a frame delimiter, begins with a fourteen-byte Ethernet frame header **320**, followed by a twenty-byte IPv4 header **340**, followed by a TCP header **360**, which is also typically twenty bytes. Each protocol header contributes information used by the routing module **101**. For example, the IPv4 header includes address information and the TCP header includes port information. Therefore, the routing module may be configured to distinguish between protocols and extract information from the different protocol headers of a packet using the correct bits, or fields of bits, of the packet.

Accordingly, still referring to FIG. **1**, the routing module **101** uses a packet classifier **120** to classify the packet. In some implementations, the packet classifier **120** determines which communication protocols are used by the packet. In some implementations, the packet classifier **120** compares the initial bits of a packet to one or more patterns, each associated with an entry in the field selection table **130**. If a pattern matches to the initial bits of a packet, then the associated entry in the field selection table **130** indicates which fields (sets of bits) in the packet to use. In some implementations, the pattern matching is performed using ternary-logic content-addressable memory (TCAM). In a TCAM, the pattern for each bit can match to a **1**, to a **0**, or to either. That is, the pattern can designate some of the bits as "don't care" bits that will satisfy a comparison with the pattern regardless of the value of those particular bits. The "don't care" bits are typically indicated by an "x". In some implementations, the patterns are ordered in such that if multiple patterns can may match to the same packet, the first pattern in the ordering is used. The last pattern can then be a generic pattern that will match all packet headers and is associated with a default rule. In some implementations, the packet classifier **120** returns an identifier for an entry in the field selection table **130**. In some implementations, the packet classifier **120** returns two values: a classification result and an identifier for an entry in the field selection table **130**. In some implementations, the classification result is unique to the pattern matched.

The field selection table **130** is used to identify which data fields in the header of a packet to process based on the determined classification. In some implementations, the field selection table **130** specifies, for each communication protocol or combination of communication protocols, which bits of the header(s) to use. The selected bits are used as part of an input to the hash module **140**. For example, an entry in the field selection table **130** may be a bit mask that, when applied to the packet header with a logical AND operation, zeros-out or clears the header bits that are not selected (effectively leaving only the selected field values). In some implementations, the routing module **101** extracts the results into a data structure, an n-tuple, holding the selected field data. In such implementations, the n-tuple is passed to the hash module **140**. In some implementations, the bit mask identified by an entry in the field selection table **130** is applied to the header, or to a single contiguous block of bits from the header, and the result is passed to the hash module **140**. That is, the entire packet prefix, or a single contiguous block of bits from the

packet, is used with bits for non-selected fields simply set to a constant, e.g., zero. In some implementations, the bit mask is stored in a compressed manner, where each bit of the mask represents multiple bits, e.g., one bit in a byte-mask represents eight bits of a bit-mask. In some implementations, an entry in the field selection table **130** identifies specific bits, or sets of bits, to use from the header. In some implementations, an entry in the field selection table **130** identifies specific sets of bits using a compressed encoding wherein each bit of the encoding corresponds to a range of bits in the header. For example, if a bit is "on" (i.e., set to 1), then the corresponding range (e.g., the n^{th} octet or byte, or the bits from bit x to bit y of the header) is extracted. In some implementations, the resulting value(s) is copied into a memory register.

The network device **100** includes a hash module **140**, used to hash an input value (or values). For example, the hash module **140** may be used to hash bit values selected from a packet's header data, as indicated by the entry in the field selection table **130**. In some implementations, values in addition to the bits selected from a packet header are included in the input values to the hash module **140**. The hash module **140** processes the input values and produces a hash value. In some implementations, the hash value is of a fixed bit-length. In some implementations, an identifier associated with a result from the packet classifier **120** is included in the input values. For example, in some implementations, the classification result is included with the bit values from the packet's header data. In some implementations, the hash module **140** produces a hash result value from a sequential stream of input values. For example, the hash module may accept any number of input values. In some such implementations, each input value is limited to a predetermined number of bits (e.g., 32 or 64). In some such implementations, the hash module result is updated as each input value is received, such that the result is impacted by every bit received as input. In some implementations, the hash module is placed in an initial state prior to generating a result. In some implementations, the first input to the hash module is a seed value. In some implementations, the hash module implements a hash algorithm in special hardware, e.g., in an ASIC or FPGA.

The routing module **101** uses the resulting hash value, or a portion of the resulting hash value, to select an appropriate entry in the routing table **150**. In some implementations, the hash module **140** produces a **2**N-bit (e.g., 32 bit) hash value and the routing module **101** only uses the lower order (or, alternatively, the higher order) N bits (e.g., 16 bits) of the hash value. In some implementations, the routing table **150** is stored in a manner facilitating fast look-ups using the hash value. For example, in some implementations, the routing table **150** may be keyed to, or indexed by, the hash module **140** output values. In some implementations, the routing table **150** may be implemented as an array, two-way associative array, four-way associative array, n-way associative array, or successive-row-lookup table.

The maintenance module **102** maintains, and modifies as necessary, the contents of the packet classifier **120**, field selection table **130**, and routing table **150**. In some implementations, the maintenance module is implemented as hardware in an integrated circuit, such as an ASIC or FPGA. In some implementations, the maintenance module **102** is implemented as computer executable instructions that are executed by a special purpose or general purpose processor. In some implementations, the maintenance module **102** is on a separate network controller, such as an SDN controller, remotely maintaining the components of the network device **100**. In some implementations, the maintenance module **102** updates the routing table **150** for each new packet flow. In some implementations, the maintenance module **102** updates the routing table **150** for a new packet flow if the new flow meets certain requirements. For example, in some implementations, the maintenance module **102** updates the routing table **150** to ensure a consistent route when a flow indicates a need for a certain quality of service (QoS) or in-order delivery. In some implementations, the maintenance module **102** updates the packet classifier **120**. For example, in some implementations, the maintenance module **102** updates the packet classifier **120** with an additional classifier pattern used to differentiate between two distinct packet flows that result in the same hash result value from the hash module **140**. The new pattern will have a new, internally unique, classification result, and may be associated with an existing entry in the field selection table **130** or may be associated with a new entry in the field selection table **130**.

FIG. **2** is a flowchart of an example method **200** for routing a packet using the network device shown in FIG. **1**. In brief overview, the method **200** includes receiving a packet (step **210**), retrieving an identifier associated with a packet classifier entry (step **220**), and choosing a field selection table entry (step **230**). The method further includes generating a first hash module input by identifying a set of bits of the received packet (step **240**), hashing the first hash module input together with the identifier (step **250**), matching the hash result to an entry in the routing table (step **260**), and obtaining a routing instruction from the routing table entry (step **270**). The network device then processes the packet according to the obtained instruction.

As indicated above, the method **200** begins with receiving a packet (step **210**). The packet can be received from any of the sources **110** connected to the network device **100**. The packet can be received via a wired network connection or wirelessly. In general, each packet begins with a sequence of header bits that identify a communication protocol for the packet and any additional packet information in accordance with the communication protocol, e.g., addressing information for the packet. A data packet includes, after the header bits, additional data bits referred to as the payload. The payload may encapsulate another packet, e.g., a packet in a format of another protocol. The payload for an encapsulated packet begins with another header. Thus, as shown in FIGS. **3**A and **3**B, an Ethernet packet may encapsulate an IP (IPv4 or IPv6) packet, which may encapsulate a TCP packet (or a UDP packet, or an ICMP packet, or any other protocol packet). Each encapsulation is a layer, and the header for each layer specifies information that may be useful in determining how to handle the packet. For example, the IP layer includes a source address, a destination address, and a protocol indicator for the next-layer protocol of an encapsulated packet (e.g., **1** for ICMP, **6** for TCP, **17** for UDP, etc.). Similarly, the TCP layer includes identifiers for a source port and a destination port, and also includes control flags indicating a flow state, e.g., a synchronization (SYN) flag used to initiate a flow and a final packet (FIN) flag used to terminate an existing flow.

Continuing with FIG. **2**, after receiving a packet (step **210**), the routing module **101** retrieves an identifier for the packet using the packet classifier **120** (step **220**). A packet classifier **120** matches a received packet with a packet classifier entry. In some implementations, the packet classifier **120** will parse the packet header to determine specific information associated with the packet, such as protocol, source IP address and destination IP address. In some implementations, the packet classifier **120** will compare the packet header to one or more classification patterns. For example, in some implementations, the packet classifier **120** is a TCAM. In general, a packet may satisfy conditions for multiple possible classifi-

cations. In some implementations, the packet classifier **120** prioritizes or orders the classification patterns such that the packet is classified according to the highest priority (highest order or "first") pattern it satisfies. For example, the packet classifier **120** may have a low order generic pattern for all IPv4 packets, a higher order pattern for all TCP/IPv4 packets, and a higher order pattern for TCP/IPv4 packets addressed to a particular address or range of addresses (e.g., a sub-net). If a TCP/IPv4 packet arrives addressed to an address in that range, it would satisfy all three patterns, but the prioritization determines that it should be classified using the highest order pattern. In some implementations, the packet classifier **120** has a maximum number of entries. In some implementations, a smaller number of entries may result in a reduced level of electrical power consumption. For example, where the packet classifier **120** is implemented as a TCAM, a TCAM with at most 128 or 256 entries will use significantly less power than a TCAM with thousands of entries.

The routing module **101** uses the retrieved identifier to identify an entry in the field-selection table (step **230**). Each entry in the field-selection table **130** indicates how to parse or process the header information for a packet. In some implementations, an entry indicates which bits (or sets of bits) of the packet are to be used as input values to a hash module **140**. In some implementations, multiple packet classifier entries may correspond to a same field-selection table entry. In some implementations, the packet classifier **120** is implemented as a TCAM and each entry in the TCAM corresponds to (or indexes) an entry in the field-selection table **130**. In some such implementations, the entries in the field-selection table **130** are data structures including an identifier (e.g., an identifier for an entry in the packet classifier **120**) and a field selection indicator (e.g., a bit selection pattern, as described above). In some implementations, the identifier is an arbitrary number selected as a "Seed" value that, when passed to the hash module **140** as an input value, causes the hash module **140** to generate a particular hash result value (or causes the hash module **140** to generate a hash result value other than a particular hash result value). In brief, as discussed in more detail below, in some implementations, the hash result value corresponds to an entry in the routing table **150** and the Seed value may be selected so that hash result value corresponds to a particular entry in the routing table **150**. Thus, in some implementations, at steps **220** and **230**, the routing module **101** uses the packet classifier **120** to identify an entry in the field-selection table that specifies an identifier or Seed value and a bit-selection instruction.

The routing module **101** then selects bits from the header (s) of the received packet (step **240**) based on the bit-selection instruction from the entry in the field selection table. One or more of the fields of the packet header may be selected. For example, the entry may indicate selection of bits representing the packet's source IP address, destination IP address, next level protocol, destination port, and source port. As another example, the entry may indicate selection of bits representing the packet's destination IP address sub-net (e.g., the first 24 bits of an IPv4 address), next level protocol, destination port, and the TCP synchronization control flag (SYN). In some implementations, the routing module **101** extracts the designated bits from the packet header and passes the designated bits to a hash module **140** as input. In some implementations, the routing module **101** identifies a single contiguous block of bits from the packet, applies a bit mask to the block, the bit mask identified in the entry in the field selection table, and passes the result to a hash module **140** as input. For example, the single contiguous block of bits may be the first forty-four bytes (three hundred fifty two bits) after the Ethernet header, which is sufficient to include the twenty bytes of an IPv4 header or the forty bytes of an IPv6 header, and the first few bytes of an encapsulated header. In some implementations, the routing module **101** also passes the identifier (from step **220**) or Seed value (from step **230**) to the hash module **140** as input.

The routing module **101** uses the hash module **140** to determine a hash value for the input values (step **250**). In general, the hash module **140** accepts the bits selected from the packet header, and any additional input bits (e.g., an identifier or Seed value), as input. The hash module **140** implements a hash function which generates a hash value based on the input values. In some implementations, the hash module **140** calculates the hash result using a hash function such as MD5, Jenkins, or MurMur. In some implementations, the hash module **140** uses a table of random numbers for generating hash values. In some implementations, the hash module **140** uses a linear-feedback shift register (LFSR). Typically, a hash function uses every input value such that a change in any one input bit will result in a different output value. The output value for a hash function is typically represented with fewer bits than the input value. For example, the input values may be 128 bits that include two 32-bit IPv4 addresses, port information, protocol information, and a seed value, and the input values may be reduced to, for example, a 32-bit hash result value. This is a form of lossy compression, which means that, for such functions, there must be at least one output value that can be reached from at least two different input values. When this happens, there is a collision between the different input values that resulted in the same output hash value. As introduced above, in some implementations, the Seed value may be adjusted to avoid collision events. Further discussion of collisions, and methods of addressing collision events, is presented below.

Continuing to refer to FIG. **2**, in the method **200**, the routing module **101** takes the output of the hash module **140** and matches it to an entry in the routing table (step **260**). In some implementations, the routing table **150** is a hash table keyed to the results of the hash module **140**. In some implementations, the hash result value is an index into the routing table **150**. In some implementations, the routing module **101** uses the hash result, or a portion of the hash result, to calculate an index into the routing table **150**. In some implementations, an index into the routing table **150** is a memory address allowing for direct access to memory storing an entry in the routing table **150**. In some implementations, the routing table **150** has a fixed number of entries such that each possible hash result value can be translated to a specific entry in the routing table **150**. For example, there could by 2^{16} entries and the lower-order 16 bits of the hash result value identifies a respective entry. Each entry is either empty or is populated with routing instructions that corresponds to a packet with header information that results in a corresponding hash value. In some implementations, the entry identified is a generic entry for packets flowing to a subnet.

In some implementations, matching the hash result to an entry in the routing table includes verifying the match. For example, each routing table entry may include match data that can be used to confirm the entry is correct for a particular packet. Match data is described in more detail below. In some implementations, matching the hash result to an entry in a the routing table includes deriving a routing table index from a first subset of a set of hash result bits (the binary representation of the hash result), locating an entry in the routing table from the routing table index, identifying a match data item stored in the entry, and verifying that the match data item is equal to a second subset of the set of hash result bits. In some

implementations, the second subset of bits includes at least one bit not in the first subset of bits. In some implementations, the first subset of bits does not intersect with the second subset of bits. In some implementations, the first subset of bits is the x lower order bits of an n-bit hash result, and the second subset of bits is the upper n−x bits of the n-bit hash result. In some implementations, another characteristic of the packet is used to verify the match.

In some implementations, the entry identified is a specific entry created for a particular flow of data packets. For example, in some implementations, a new entry is added to the routing table **150** when a new flow is detected, and the new entry indicates specific instructions for the new flow. A new flow may be detected, for example, when a TCP/IP packet arrives with the SYN flag set, indicating the beginning of a TCP handshake. In some implementations, if a new flow is detected and the resulting hash value points to (or indicates) an entry in the routing table **150** that is already in use, this indicates a collision. In some implementations, when a collision is detected, a new entry is added to the packet classifier **120** such that the identifier for the entry in the packet classifier **120** is changed, thereby altering the input to the hash module **140** and generating a new hash result value. In some implementations, a collision may be detected in other ways, as described below.

The routing module **101** obtains the routing instruction from the entry in the routing table (step **270**) and processes the packet using the routing instruction. In some implementations, the routing table entry identifies a network interface **115** through which the network device **100** forwards the packet. In some implementations, the routing table entry identifies a next-hop address. In some implementations, the routing table entry includes an instruction to allow or drop the packet. In some implementations, the routing table entry includes an instruction to process the packet before forwarding, e.g., to fragment the packet or to update a time-to-live field or a hop limit field. The network device processes the packet using the routing instructions. Thus, for example, the network device can forward the packet to the proper destination **190**.

FIG. **3**A shows the format **314** for the headers of a typical TCP/IPv4 packet transmitted via Ethernet. In broad overview, the illustrated format includes an Ethernet frame **320**, an Internet Protocol (IP) version 4 header **340**, a transmission control protocol (TCP) header **360**, and the beginning of the encapsulated data **380**, i.e., the payload.

A TCP/IPv4 packet, as shown in FIG. **3**A, begins with a new packet preamble and delimiter, most of which is not shown. After the delimiter, an Ethernet frame header **320** includes a media access control (MAC) address for the packet's immediate destination (i.e., the network device receiving the packet) and a MAC address for the packet's immediate source (i.e., the network device transmitting the packet). A MAC address is 48 bits, or six 8-bit octets. The Ethernet frame header **320** also includes a 16-bit "Ethertype" indicator, which may indicate the size of the frame or the protocol for the Ethernet payload (i.e., the next level protocol). The Ethernet frame header **320** is followed by the Ethernet payload, which begins with a header for the encapsulated packet.

FIG. **3**A shows the format **314** for the headers of a typical TCP/IPv4 packet, thus the Ethernet frame header **320** is followed by an IPv4 header **340**. The first four bits indicate the Internet Protocol version (i.e., 4). The next sets of bits indicate the header length (IHL), flags to differentiate service requirements (DSCP, used, e.g., to express quality of service (QoS) requirements), explicit congestion notification (ECN), a length for the IP packet, a packet identification shared across packet fragments, IP flags, and a fragment offset. After the packet fragmentation bits, the IPv4 header **340** indicates a time to live (TTL) for the packet, which may be measured in time (e.g., seconds) or hops (number of network devices that can forward the packet). After the TTL, the IPv4 header **340** indicates the protocol for the next level encapsulated packet. For example, a 1 indicates the Internet control message protocol (ICMP), a 6 indicates TCP, and 17 indicates the user datagram protocol (UDP). The IPv4 header **340** further includes a header checksum, which must be recalculated every time the header changes, e.g., whenever the TTL is updated. The IPv4 header **340** next specifies a 32-bit source address and a 32-bit destination address. Additional header fields may be used, but may be omitted and are not shown in FIG. **3**A.

After the IPv4 header **340**, FIG. **3**A shows a TCP header **360**. The typical TCP header begins with a 16-bit source port identifier and a 16-bit destination port identifier. A TCP port is a virtual port, typically used to indicate the type of data in the payload so that the receiver can pass the packet to the correct application. The TCP header **360** then specifies sequencing information including a sequence number for the packet, an acknowledgement number, and a data offset. The TCP header **360** includes control flags, e.g., SYN, FIN, and ACK, and additional control information such as the window size, a checksum, and other options. The data encapsulated **380** begins after the TCP header **360**.

FIG. **3**B shows the format **316** for the headers of a typical TCP/IPv6 packet transmitted via Ethernet. In broad overview, the illustrated format includes an Ethernet frame **320**, an Internet Protocol (IP) version 6 header **350**, a transmission control protocol (TCP) header **370**, and the beginning of the encapsulated data **390**, i.e., the payload. The Ethernet frame **320** in the illustrated packet is identical to the Ethernet frame **320** in FIG. **3**A.

FIG. **3**B shows the format **316** for the headers of a typical TCP/IPv6 packet, thus the Ethernet frame header **320** is followed by an IPv6 header **350**. The first four bits indicate the Internet Protocol version (i.e., 6). The next sets of bits indicate a traffic class, a flow label, and the payload length. After the payload length, the IPv6 header **350** indicates a Next Header, which is the same as the protocol identifier used in IPv4. That is, it may be a 1 for ICMP, a 6 for TCP, a 17 for UDP, or any other number indicating an associated protocol for the next header in the packet. The IPv6 header **350** then indicates a hop limit for the packet, equivalent to the TTL of IPv4 when used to specify the number of network devices that can forward the packet. After the hop limit, the IPv6 header **350**, specifies a 128-bit source address and a 128-bit destination address. Additional header fields may be used, but may be omitted and are not shown in FIG. **3**B. There is no checksum for an IPv6 header, eliminating one of the bottlenecks in IPv4 packet processing.

After the IPv6 header **350**, FIG. **3**B shows a TCP header **370**. The TCP header **370** is identical to the TCP header **360** shown in FIG. **3**A, but the offsets from the Ethernet frame **320** are increased because the size of an IPv6 header **350** is larger than the size of an IPv4 header **340**. The data encapsulated **390** begins after the TCP header **370**.

FIG. **4** illustrates an example of a packet classifier **420** and an example of a field selection table **430**. The illustrated packet classifier **420** is shown with patterns expressed in hexadecimal, such that each four bit section of the header is represented by a value in the range 0-9, A-F, or by an X for "don't care." The patterns illustrated begin with the IP header, although in some implementations the patterns begin with the Ethernet frame, e.g., so that the source MAC address can be

used in the patterns. The illustrated packet classifier **420** begins with a lowest priority filter **421** matching any IPv4 packet with an IP header of twenty octets and a filter **422** matching any IPv6 packet. The packet classifier **420** also includes a filter **423** matching any TCP/IPv4 packet and a filter **424** matching any TCP/IPv6 packet. The packet classifier **420** includes a higher priority filter **425** matching a TCP/IPv4 packet from address 1.2.3.4 to address 5.6.7.8, with a destination port of 80 (shown in hexadecimal as 0050). The packet classifier **420** also includes a higher priority filter **426** matching a TCP/IPv6 packet from address 0102:0304:0506:0708:0910:1112:1314:1516 to address 1718:1920:2122:2324:2526:2728:2930:3132.

Each row of the example field selection table **430** corresponds to a row of the example of a packet classifier **420**. Thus a packet that satisfies the highest priority filter **426** is associated with a corresponding entry **436** that indicates where the address bits are in the header. An IPv4 packet that does not satisfy any of the higher priority filters will match the lowest priority filter **421**, which corresponds to an entry **431** that indicates where IPv4 address bits are located in the header. As indicated above, while the example field selection table **430** includes one entry for each entry in the packet classifier **420**, in some other implementations, more than one entry in the packet classifier **420** may correspond to a common entry in the field selection table **430**. For example, in some implementations, the filters **421**, **423**, and **425**, which each match to an IPv4 packet, each correspond to a single entry in the field selection table **430** that identifies, for example, bits common to any IPv4 header.

FIG. **5** shows an example structure of the inputs and outputs of a hash module used in the network device shown in FIG. **1**. The example structure of a hash module **140** takes, as input, bits from the packet **520** and a seed value **510**, and generates a hash result value **530**. In some implementations, the bits from the packet **520** are the values for various protocol fields indicated by an entry in the field selection table **130**. In some implementations the bits from the packet **520** are a block of bits from the header with some of the bits therein set to zero. Although shown in FIG. **5** as multiple fields **520**, the bits from the packet **520** may only be one field, may be a single block of data representing multiple fields, may be multiple fields, and/or may include bit values that have been set to a constant. In some implementations, the seed value **510** is an identifier specified by the packet classifier **120**. In some implementations, the seed value **510** is an identifier or Seed value specified by an entry in the field selection table **130**. Because the hash module processes all of the input bits, different values for the seed value **510** will cause the hash module **140** to produce different values for the hash result **530**, without any change to the input values **520** selected from the packet header. In some implementations, the hash result **530** is a fixed number of bits. In some implementations, the lower order bits of the hash result **530** are used to locate an entry in the routing table **150**. For example, in some implementations, the routing table **150** has space for 2^{16} entries, and the lower order sixteen bits of the hash result **530** uniquely identify one of those 2^{16} spaces in the routing table **150**. In some implementations, the routing table **150** is dynamically sized and an intermediary index maps hash result values to indices into the dynamically sized routing table **150**. In some such implementations, the intermediary index has the same number of entries as possible hash result values **530**, e.g., 2^{16} entries for a 16 bit result. In some such implementations, the intermediary index has a smaller number of entries, e.g., m, and the entry for a hash result values is found at the hash result value modulo m. These implementations have an increased likelihood of hash collisions.

In some implementations, the hash result **530** is an n-bit value (e.g., a 64-bit value or a 32-bit value) and the lower order n−x bits (e.g., 16 bits) are used to determine an index into the routing table **150**. In some implementations, the routing module **101** confirms that the entry at the determined index is the correct entry (and not an entry reached via a hash collision) by matching one or more stored values with confirmation values (referred to as "match data"). In some such implementations, the full n-bit hash result value **530** may be used as match data. For example, in some implementations, the routing table **150** includes, in each entry, a stored copy of the n-bit value that matches to the entry even though only n−x bits are used to locate the entry. In some such implementations, only the additional bits of the hash value not included in the n−x bits used to locate the entry are stored as match data. The routing module **101** can confirm that the entry is correct by matching the stored n-bit value (or stored portion thereof) to the n-bit hash result value **530**. As an example, in some implementations, the hash result **530** is 64 bits and only the higher order 48 bits are stored as match data. The lower order 16 bits are used to locate an entry in the routing table **150** and the remaining bits are used to confirm a match. In some such implementations, no other match data is stored. In some implementations, the match data includes bits from one or more of the input fields **520**. For example, in some implementations, the match data includes an address field from the packet header. In some implementations, the match data is an input value **520** that is common to all packets for which the entry should match. In some implementations, the match data is stored in a compressed form. In general, a routing table storing less match data will use less memory and require less circuitry and electrical power for verifications with match data. In some implementations, no match data is stored and no verification is performed.

FIG. **6** shows an example structure of a routing table **150** with stored match data. The example structure shows the routing table **150** implemented as an associative array containing two tables, an index table **651** and a data table **652**. The routing table **150** is accessed using key data **610**, which includes a hash value **614**, e.g., bits from a hash result **530**. When a routing module **101** accesses the routing table **150** to identify routing instructions for a packet, the routing module **101** passes in the hash value **614**, which is then converted to an index using the index table **651**. The index is used to locate a corresponding entry in the data table **652**. In some implementations, the key data **610** also includes match data **612**.

The index table **651** maps hash values to indices. The index table **651** is ordered by hash values such that an entry for an input hash value **614** can be quickly located in the index table **651**. For example, in some implementations, the index table **651** is structured such that each entry is at a memory location addressed by a start address plus the respective input hash value multiplied by a constant (e.g., the size of an entry in the index table **651**).

The data table **652** is ordered by the indices from the index table **651**. As shown in FIG. **6**, in some implementations, each populated entry in the data table **652** includes confirmation match data and routing instructions. In some implementations, the data table **652** does not include the confirmation match data. In some implementations, the confirmation match data includes multiple values, e.g., a value representative of a source or destination address, a value representative of a source or destination sub-net, a hash value, or any other data that may be used as confirmation match data. In some implementations, the confirmation match data is the hash

result value **530** produced by the hash module **140** (see FIG. **5**). In some such implementations, the hash value **614** used as key data **610** for the routing table **150** is a portion of the hash result value **530** stored as confirmation match data. For example, the input hash value **614** may be only the lower (or higher) order bits of the hash result value **530**. In some implementations, the key data **610** is a subset of the bits representing the hash result value **530**, and the match data is a different subset of the bits representing the hash result value **530**. In some implementations, the confirmation match data is the same data that was passed into the hash module **140**. In some implementations, the confirmation match data is a subset of the data that was passed into the hash module **140**. In some such implementations, the confirmation match data may include less information than the data that was passed into the hash module **140**. In some implementations, the confirmation match data is compressed. In some implementations, the index table **651** and the data table **652** are n-way associative. For example, the index table **651** and the data table **652** may be two-way associative, such that the data table **652** includes, for each entry, the hash value that corresponds to that entry (from the index table **651**).

In some implementations, the match data **612** is also passed in as an input value **610**. In such implementations, if the match data stored in the entry matches the input match data **612**, then the routing instruction stored at the entry is returned **620**. If there is no entry, the packet may belong to a new flow and the maintenance module **102** updates the tables accordingly. If there is an entry, but the match data in the data table **652** does not match to the input match data **612**, then there is a collision. That is, the values selected from the packet hashed into a hash value that is already in use by packets in another flow. This packet, too, is for a new flow and the maintenance module **102** updates the tables accordingly. In some implementations, the match data **612** is only passed in as an input value **610** for a new flow. For example, in some implementations, the maintenance module **102** may receive an instruction to establish a new flow and uses the match data **612** to verify that the newly established flow does not have a collision with an existing flow.

In some implementations, when there is a new flow with a hash collision with an existing entry in the routing table **150**, the maintenance module **102** updates the packet classifier **120** to add a new entry for the new flow, such that the new flow is associated with a new identifier. The new entry in the packet classifier **120** may correspond to an already existing entry in the field selection table **130**, or the new entry in the packet classifier **120** may correspond to a new entry in the field selection table **130**. In either case, the updates result in a configuration for the new flow such that the data extracted from packets in the flow, along with the new identifier, cause the hash module **140** to generate a distinct hash result **530** (as shown in FIG. **5**). The new hash result can be used, e.g., as an input value **610**, to identify an entry in the routing table **150** for the new flow. In some implementations, the maintenance module **102** configures the entry with routing instructions for the flow. In some implementations, the maintenance module **102** verifies that the entry is not already in use, and, if it is, modifies the identifier associated with the flow so that the hash module **140** will generate a different result.

FIG. **7** is a flowchart of an example method for initiating a new flow using the network device shown in FIG. **1**. In brief overview, the method **700** includes receiving an indication of a new flow (step **701**), retrieving an identifier associated with packet classifier entry (step **702**), and choosing a field selection table entry (step **703**). The method further includes generating a hash module input by processing a packet of the new flow (step **704**), hashing the generated input along with the identifier (step **705**), and detecting a collision (step **706**). If a collision is detected, the method **700** includes adding a new entry to the data packet classifier (step **707**), assigning a new identifier for the new entry in the data packet classifier (step **708**), and hashing the generated hash module input and the new identifier to produce a new hash result (step **709**). The method **700** then verifies that the new hash result does not have a collision with an existing routing table entry (step **710**). If there is a collision at step **710**, the method **700** repeats step **708**, assigning a different new identifier for the new entry. Once there are no collisions (at step **706** or step **710**), the method **700** includes adding an entry to the routing table (step **720**).

As indicated above, the method **700** begins with the maintenance module **102** receiving an indication to initiate a new data communication flow (step **701**). In some implementations, the indication is an instruction received by the maintenance module **102**. For example, in some implementations, an application or service instance on an host server may initiate a flow by sending an instruction to the maintenance module **102**. In some such implementations, the application or service instance designates specific packet handling instructions for the new flow. For example, the host server can request that acknowledgment packets for a new data stream be directed to a particular stream management instance. In some implementations, the maintenance module **102** detects a new flow as a previously unseen flow. In some implementations, the packet classifier **120** classifies a packet as one initiating a new flow. For example, the packet classifier **120** may detect the beginning of a TCP handshake, as indicated by a SYN flag set in the TCP packet header. In some implementations where the data packet classifier **120** is implemented as a TCAM, there is a TCAM pattern to match one or more flow initiation packets. For example, in some implementations, the TCAM has a high priority pattern for identifying a TCP packet with the SYN flag set, which is interpreted by the network device to indicate a new flow. In some implementations where the data packet classifier **120** is implemented as a TCAM, receiving a packet with a packet header that matches to the default row of the TCAM may initiate a new flow. In some implementations, an indication of a new flow includes an indication of a desired routing instruction for the new flow. In some implementations, the maintenance module **102** determines a desired routing instruction for a new flow. For example, the maintenance module **102** may access additional network topology information to identify a next-hop device for packets in the new flow.

The method **700** includes the maintenance module **102** retrieving an identifier associate with a packet classifier entry (step **702**). This step is analogous to step **220** in FIG. **2**. The method **700** includes the maintenance module **102** choosing a field-selection table entry (step **703**), this step is similarly analogous to step **230** in FIG. **2**. The method **700** includes the maintenance module **102** selecting bits from a packet of the flow (step **704**). This step is analogous to step **240** in FIG. **2**. The method **700** also includes hashing the selected bits and the identifier (step **705**). This step is analogous to step **250** in FIG. **2**. As with step **250** in FIG. **2**, step **705** in FIG. **7** includes using the identifier as a seed value to a hashing function.

In the method **700**, the maintenance module **102** then detects whether a hash collision has occurred (step **706**). A collision is detected, for example, when the routing table **150** does not have an empty location at the index specified by the hash result **530** for the new flow. In some implementations, a collision is detected if there exists an entry with different match data as compared to match data for the new flow. In

some implementations, a collision is detected if there exists an entry with a different routing instruction as compared to the desired routing instruction for the new flow, and no collision is detected if the routing instruction in the routing table **150** is the desired routing instruction for the new flow. That is, if a new flow is established and packets for that new flow would cause the routing module **101** to access an entry in the routing table that is populated with handling instructions prior to establishing the flow, then there is a collision. However, if those handling instructions are the same as the desired handling instructions, in some implementations, the collision is ignored because the new entry would have the same handling instructions as the existing entry. In some implementations, a collision is detected unless the returned location in the routing table **150** is empty, which indicates that the routing instruction for the new data communication flow may be stored at the returned location in the routing table **150**.

If a collision is detected at step **706**, the method **700** includes establishing a new identifier for the flow, e.g., by adding a new entry to the data packet classifier (step **707**), assigning a new identifier for the new entry in the data packet classifier (step **708**), and hashing the generated hash module input and the new identifier to produce a new hash result (step **709**). When adding a new entry to the data packet classifier (step **707**), the new entry in the data packet classifier **120** is selected to match packets of the new flow, but not packets of flows already being processed. In some implementations, the next entry is selected to match packets only of the new flow. In some implementations, the maintenance module **102** identifies the pattern previously used to classify a packet for the flow and adds additional criteria to the pattern, specific to the packet classified. For example, if the packet matched a classification pattern that specified the packet's destination subnet, but not the full destination address, the maintenance module **102** adds a new classification pattern that specifies the packet's full destination address.

The new classification pattern corresponds to a new identifier assigned for the new entry in the data packet classifier (step **708**). In some implementations, the new identifier is an output value from the packet classifier. In some implementations, the new identifier is stored in the field selection table **130**, e.g., as a Seed value. In some implementations, the new identifier is selected at random. In some implementations, the new identifier is selected in a deterministic manner, e.g., by incrementing a counter.

If a collision was detected at step **706**, the method **700** includes, after establishing a new identifier for the flow in step **708**, hashing the generated hash input from step **704** and the new identifier (step **709**). Because the inputs are now different, the hashing at step **709** results in a new a new hash value. The result of the hashing at step **709** is used to identify an entry location in the routing table. The new hash value will almost always correspond to a different entry in the routing table **150**. If there is a collision (step **710**), the method **700** returns to step **708** and assigns a new identifier for the entry in the data packet classifier (added in step **707**). In some implementations, steps **708**, **709**, and **710** are repeated until there is no collision. In some implementations, after a predetermined number of iterations without avoiding a collision, an error condition is triggered. In some implementations, after a predetermined number of iterations without avoiding a collision, additional steps are taken to modify the hash output, e.g., by associating the new entry in the packet classifier with a different field-selection table entry.

When there are no collisions detected at step **706** and/or step **710**, the method **700** includes adding, by the maintenance module **102**, a new entry to the routing table (step **720**), the entry containing the desired routing instruction and match data for the flow. In some implementations, the new entry also contains the hash result from step **709**. In some implementations, the match data is a portion of the hash result from step **709**. For example, in some implementations, the hash result is a 64-bit value and the match data is the higher order 48 bits of the hash result. In some such implementations, the lower order 16 bits of the hash result are used to locate an entry in the routing table and the higher order 48 bits are used to verify a match. Packets received after completion of the method **700** are routed using the method explained in relation to FIG. **2**, which locates and retrieves the routing instructions and associated data stored in step **720**.

In some implementations, the maintenance module **102** removes entries from the routing table **150** for flows that have ended, terminated, or become stale. For example, the packet classifier **120** may detect a TCP teardown, as indicated by a FIN flag set in the TCP packet header. In some implementations where the data packet classifier **120** is implemented as a TCAM, there is a TCAM pattern to match one or more flow termination packets. In some implementations, the maintenance module **102** periodically removes entries from the routing table **150** that have not been used for a threshold length of time or that were established more than some predetermined period of time prior.

Implementations of the subject matter and the operations described in this specification can be implemented in digital electronic circuitry, or in computer software embodied on a tangible medium, firmware, or hardware, including the structures disclosed in this specification and their structural equivalents, or in combinations of one or more of them. Implementations of the subject matter described in this specification can be implemented as one or more computer programs embodied on a tangible medium, i.e., one or more modules of computer program instructions, encoded on one or more computer storage media for execution by, or to control the operation of, a data processing apparatus. A computer storage medium can be, or be included in, a computer-readable storage device, a computer-readable storage substrate, a random or serial access memory array or device, or a combination of one or more of them. The computer storage medium can also be, or be included in, one or more separate components or media (e.g., multiple CDs, disks, or other storage devices). The computer storage medium may be tangible and non-transitory.

The operations described in this specification can be implemented as operations performed by a data processing apparatus on data stored on one or more computer-readable storage devices or received from other sources.

A computer program (also known as a program, software, software application, script, or code) can be written in any form of programming language, including compiled or interpreted languages, declarative or procedural languages, and it can be deployed in any form, including as a stand-alone program or as a module, component, subroutine, object, or other unit suitable for use in a computing environment. A computer program may, but need not, correspond to a file in a file system. A program can be stored in a portion of a file that holds other programs or data (e.g., one or more scripts stored in a markup language document), in a single file dedicated to the program in question, or in multiple coordinated files (e.g., files that store one or more modules, sub programs, or portions of code). A computer program can be deployed to be executed on one computer or on multiple computers that are located at one site or distributed across multiple sites and interconnected by a communication network. Examples of communication networks include a local area network

("LAN") and a wide area network ("WAN"), an inter-network (e.g., the Internet), and peer-to-peer networks (e.g., ad hoc peer-to-peer networks).

The processes and logic flows described in this specification can be performed by one or more programmable processors executing one or more computer programs to perform actions by operating on input data and generating output. The processes and logic flows can also be performed by, and apparatus can also be implemented as, special purpose logic circuitry, e.g., an FPGA (field programmable gate array) or an ASIC (application specific integrated circuit).

While this specification contains many specific implementation details, these should not be construed as limitations on the scope of any inventions or of what may be claimed, but rather as descriptions of features specific to particular implementations. Certain features that are described in this specification in the context of separate implementations can also be implemented in combination in a single implementation. Conversely, various features that are described in the context of a single implementation can also be implemented in multiple implementations separately or in any suitable sub-combination. Moreover, although features may be described above as acting in certain combinations and even initially claimed as such, one or more features from a claimed combination can in some cases be excised from the combination, and the claimed combination may be directed to a sub-combination or variation of a sub-combination.

Similarly, while operations are depicted in the drawings in a particular order, this should not be understood as requiring that such operations be performed in the particular order shown or in sequential order, or that all illustrated operations be performed, to achieve desirable results. In certain circumstances, multitasking and parallel processing may be advantageous. Moreover, the separation of various system components in the implementations described above should not be understood as requiring such separation in all implementations, and it should be understood that the described program components and systems can generally be integrated together in a single software product or packaged into multiple software products.

References to "or" may be construed as inclusive so that any terms described using "or" may indicate any of a single, more than one, and all of the described terms. The labels "first," "second," "third," and so forth are not necessarily meant to indicate an ordering and are generally used merely to distinguish between like or similar items or elements.

Thus, particular implementations of the subject matter have been described. Other implementations are within the scope of the following claims. In some cases, the actions recited in the claims can be performed in a different order and still achieve desirable results. In addition, the processes depicted in the accompanying figures do not necessarily require the particular order shown, or sequential order, to achieve desirable results. In certain implementations, multitasking or parallel processing may be utilized.

What is claimed is:

1. A network device, comprising:

a packet classifier;

a field-selection table;

a hash module;

a routing table comprising entries each associated with a respective hash value; and

a routing module configured to route a packet by:

determining an entry in the packet classifier using the packet,

retrieving a first identifier associated with the determined packet classifier entry,

choosing a first field-selection table entry using the first identifier, wherein the first field-selection table entry specifies a first set of bits,

generating a first hash module input by identifying values of the first set of bits of the packet specified by the chosen first field-selection table entry,

causing the hash module to compute a first hash result based on the first hash module input and based on the first identifier,

matching the first hash result to a first entry in the routing table, and

obtaining processing data for the packet from the first routing table entry associated, by the matching, with the first hash result; and

a maintenance module configured to resolve a collision between the first hash result associated with the first entry in the routing table and a second hash result for a new data communication flow by:

adding, responsive to detecting the collision, a new entry to the packet classifier corresponding to the new data communication flow, wherein the new entry includes a new identifier that is different from the first identifier;

adding, to the field-selection table, a new field-selection table entry corresponding to the new identifier, wherein the new field-selection table entry specifies a second set of bits;

generating a second hash module input by identifying values of the second set of bits of a packet of the new data communication flow;

causing the hash module to compute a third hash result based on the second hash module input and the new identifier; and

adding, in association with the third hash result, an entry to the routing table comprising processing data associated with the new data communication flow.

2. The network device of claim **1**, wherein the new entry added to the packet classifier matches at least one field of a packet associated with the new data communication flow.

3. The network device of claim **1**, wherein the packet classifier is a ternary content addressable memory (TCAM).

4. The network device of claim **3**, wherein the first identifier associated with the first packet classifier entry comprises an index in the TCAM.

5. The network device of claim **1**, wherein the field-selection table comprises a table of byte-masks.

6. The network device of claim **1**, wherein causing the hash module to compute the first hash result based on the first hash module input and based on the first identifier comprises causing the hash module to use the first identifier as a seed value.

7. The network device of claim **1**, wherein the routing module is configured to match the first hash result to the first routing table entry by:

deriving a routing table index from a first subset of a set of hash result bits, wherein the hash result bits are a binary representation of the hash result;

locating an entry in the routing table from the routing table index;

identifying a match data item stored in the entry; and

verifying that the match data item is equal to a second subset of the set of hash result bits, the second subset comprising at least one bit not in the first subset.

8. The network device of claim **1**, wherein the first hash module input consists of the results of an application of a mask associated with the chosen first field-selection table entry to a single contiguous block of bits from the first packet,

wherein the single contiguous block of bits comprises at least a portion of a header of the first packet.

9. The network device of claim **1**, the maintenance module configured to detect the collision between the first hash result and the second hash result for the new data communication flow.

10. The network device of claim **1**, the routing module configured to retrieve the first identifier associated with the first packet classifier entry by retrieving an index value of the first packet classifier entry.

11. The network device of claim **1**, wherein the second set of bits specified in the new field-selection table entry is different from the first set of bits specified by the first field-selection table entry.

12. A method, comprising:

receiving a first packet associated with a first data communication flow;

determining a first entry in a packet classifier using the received first packet;

retrieving a first identifier associated with the determined first packet classifier entry;

choosing a first field-selection table entry using the retrieved first identifier, wherein the first field-selection table entry specifies a first set of bits;

generating a first hash module input by identifying values of the first set of bits of the received first packet specified by the chosen first field-selection table entry;

causing a hash module to compute a first hash result based on the first hash module input and the retrieved first identifier;

matching the first hash result to a first entry in a routing table;

obtaining processing data for the first packet from the matching first routing table entry associated, by the matching, with the first hash result;

detecting a collision between a second hash result for a second data communication flow and the first hash result associated with the first entry in the routing table;

adding, responsive to detecting the collision, a new entry to the packet classifier corresponding to the second data communication flow, wherein the new entry includes a new identifier that is different from the first identifier;

adding, to the field-selection table, a new field-selection entry corresponding to the new identifier, wherein the new field-selection entry specifies a second set of bits;

generating a second hash module input by identifying values of the second set of bits of a packet of the new data communication flow;

causing the hash module to compute a third hash result based on the second hash module input and the new identifier; and

adding an entry to the routing table comprising processing data associated with the new data communication flow, the added entry associated with the third hash result.

13. The method of claim **12**, wherein adding the new entry to the packet classifier comprises adding an entry that matches at least one field of a packet associated with the new data communication flow.

14. The method of claim **12**, wherein retrieving the first identifier associated with the first packet classifier entry comprises retrieving an index value of the first packet classifier entry.

15. The method of claim **12**, wherein the first hash module input consists of the results of an application of a mask associated with the chosen first field-selection table entry to a single contiguous block of bits from the received packet.

16. The method of claim **12**, wherein matching the first hash result to the first routing table entry in the routing table comprises:

deriving a routing table index from a first subset of a set of hash result bits, wherein the hash result bits are a binary representation of the hash result;

locating an entry in the routing table from the routing table index;

identifying a match data item stored in the entry; and

verifying that the match data item is equal to a second subset of the set of hash result bits, the second subset comprising at least one bit not in the first subset.

17. The method of claim **12**, wherein the second set of bits specified in the new field-selection table entry is different from the first set of bits specified by the first field-selection table entry.

18. A non-transitory computer-readable medium storing computer-readable instructions that, when executed by one or more computing devices, cause at least one of the one or more computing devices to:

receive a first packet associated with a first data communication flow;

determine a first entry in a packet classifier using the received first packet;

retrieve a first identifier associated with the determined first packet classifier entry;

choose a first field-selection table entry using the retrieved first identifier, wherein the first field-selection table entry specifies a first set of bits;

generate a first hash module input by identifying values of the first set of bits of the received first packet specified by the chosen first field-selection table entry;

cause a hash module to compute a first hash result based on the first hash module input and the retrieved first identifier;

match the first hash result to a first entry in a routing table;

obtain processing data for the first packet from the first routing table entry associated, by the matching, with the first hash result;

detect a collision between a second hash result for a second data communication flow and the first hash result associated with the first entry in the routing table;

add, responsive to detecting the collision, a new entry to the packet classifier corresponding to the second data communication flow, wherein the new entry includes a new identifier that is different from the first identifier;

add, to the field-selection table, a new field-selection entry corresponding to the new identifier, wherein the new field-selection entry specifies a second set of bits;

generate a second hash module input by identifying values of the second set of bits of a packet of the new data communication flow;

cause the hash module to compute a third hash result based on the second hash module input and the new identifier; and

add an entry to the routing table comprising processing data associated with the new data communication flow, the added entry associated with the third hash result.

19. The non-transitory computer-readable medium in claim **18**, further comprising additional instructions that, when executed by one or more computing devices, cause at least one of the one or more computing devices to add the new entry to the packet classifier by adding an entry that matches at least one field of a packet associated with the new data communication flow.

20. The non-transitory computer-readable medium in claim **18**, further comprising additional instructions that,

when executed by one or more computing devices, cause at least one of the one or more computing devices to: retrieve the first identifier associated with the first packet classifier entry by retrieving an index value of the first packet classifier entry.

21. The non-transitory computer-readable medium in claim **18**, wherein the first hash module input consists of results of an application of a mask associated with the chosen first field-selection table entry to a single contiguous block of bits from the received first packet.

22. The non-transitory computer-readable medium in claim **18**, further comprising additional instructions that, when executed by one or more computing devices, cause at least one of the one or more computing devices to match the first hash result to the first routing table entry in the routing table by:

deriving a routing table index from a first subset of a set of hash result bits, wherein the hash result bits are a binary representation of the hash result;

locating an entry in the routing table from the routing table index;

identifying a match data item stored in the entry; and

verifying that the match data item is equal to a second subset of the set of hash result bits, the second subset comprising at least one bit not in the first subset.

23. The non-transitory computer-readable medium in claim **18**, wherein the second set of bits specified in the new field-selection table entry is different from the first set of bits specified by the first field-selection table entry.

* * * * *