

(19) **DANMARK**

(10) **DK/EP 3120528 T3**



Patent- og
Varemærkestyrelsen

(12) Oversættelse af
europæisk patentskrift

-
- (51) Int.Cl.: **H 04 L 29/08 (2006.01)** **G 06 F 9/06 (2006.01)** **H 04 L 29/06 (2006.01)**
- (45) Oversættelsen bekendtgjort den: **2021-12-06**
- (80) Dato for Den Europæiske Patentmyndigheds bekendtgørelse om meddelelse af patentet: **2021-09-08**
- (86) Europæisk ansøgning nr.: **15725457.4**
- (86) Europæisk indleveringsdag: **2015-05-07**
- (87) Den europæiske ansøgnings publiceringsdag: **2017-01-25**
- (86) International ansøgning nr.: **US2015029736**
- (87) Internationalt publikationsnr.: **WO2015171922**
- (30) Prioritet: **2014-05-09 US 201414273953**
- (84) Designerede stater: **AL AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HR HU IE IS IT LI LT LU LV MC MK MT NL NO PL PT RO RS SE SI SK SM TR**
- (73) Patenthaver: **Google LLC, 1600 Amphitheatre Parkway, Mountain View, CA 94043, USA**
- (72) Opfinder: **MAO, Yuhong, 1600 Amphitheatre Parkway, Mountain View, CA 94043, USA**
- (74) Fuldmægtig i Danmark: **AWA Denmark A/S, Strandgade 56, 1401 København K, Danmark**
- (54) Benævnelse: **System og fremgangsmåde til tilpasning til netværksprotokolopdateringer**
- (56) Fremdragne publikationer:
EP-A2- 1 158 724
EP-A2- 2 632 086
US-A1- 2001 033 580
US-A1- 2002 163 935
US-A1- 2005 078 704

DESCRIPTION

BACKGROUND

[0001] Most networking application specific integrated circuits (ASICs) implement a small and fixed set of packet header editing capabilities for predefined network protocols. Examples of such header editing capabilities for particular network protocols include generic routing encapsulation (GRE) tunnel encapsulation and decapsulation, virtual local area network/multiprotocol label switching (VLAN/MPLS) tag push/pop, etc. New network protocols require either a respin or a complete redesign of the ASIC, which is both costly and time consuming. For example, redesigning and replacing the ASICs in a network can take several years and cost millions of dollars.

[0002] US 2002/0163935 discloses a system and method for facilitating packet transformation of multi-protocol, multi-flow, streaming data. Packet portions subject to change are temporarily stored, and acted upon through processing of protocol-dependent instructions, resulting in a protocol-dependent modification of the temporarily stored packet information.

[0003] US2001033580 discloses a translator unit e.g. for translating packet formatted according to protocol into packet according to second protocol.

[0004] US2013326045 implements the protocol conversion of various protocols by configuring a linkage proxy.

[0005] US2005078704 involves reading header fields of a data packet from one network along with one of translation templates in a translation template cache and generating a new header for transmission into another network. Data payload of the data packet from the former network is removed from its header. The data payload of the data packet is appended to a constructed header for the latter network.

SUMMARY

[0006] The object of the invention is achieved by the subject-matter of the independent claims. Advantageous embodiments are defined in the dependent claims. Further examples are provided for facilitating the understanding of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

[0007]

Fig. 1 is a system diagram according to an aspect of the disclosure.

Fig. 2 is another system diagram according to aspects of the disclosure.

Figs. 3A-B illustrate example instruction formats according to aspects of the disclosure.

Fig. 4 is another system diagram according to aspects of the disclosure.

Figs. 5A-F illustrate example instructions according to aspects of the disclosure.

Fig. 6A-F illustrate example instructions according to aspects of the disclosure.

Fig. 7 is another system diagram according to aspects of the disclosure.

Fig. 8 is a flow diagram of an example method according to aspects of the disclosure.

DETAILED DESCRIPTION

[0008] This disclosure provides a system and method for expediently updating network protocols. Rather than redesigning and replacing application specific integrated circuits (ASICs) when a network protocol is updated, a new ASIC is designed to adapt to future network protocols. The ASIC includes a Protocol Independent Packet Editor (PIPE) which may be programmed to support packet header editing requirements of current and possible future network protocols.

[0009] According to one embodiment, the ASIC includes a Central Editing Unit (CEU) that executes an editing program to operate on the packet header. The Central Editing Unit is a processing pipeline of three distinct editing units: a Deletion Unit, an Insertion Unit, and a Modification Unit. Each unit operates on a packet modified from the previous unit. This CEU pipeline enables flexible packet header editing at high bandwidth and packet rate with efficient hardware implementation in silicon.

[0010] The Deletion Unit enables the removal of arbitrary protocol headers from the packet. The Insertion Unit enables the insertion of arbitrary protocol headers into the packet. The Modification Unit enables flexible rewriting (either overwrite, or read-modify-write) of header fields.

[0011] The packet editing program executed by the CEU includes a set of editing instructions (deletion, insertion, and field modification), and a set of editing data (configuration, meta, shared, and dynamic). One or more different editing programs may exist for different packet header editing capabilities, such as encapsulation, decapsulation, encryption, decryption, etc. A logic unit parses and classifies the incoming packet to determine pointers to program instructions and the associated data set.

[0012] The editing instructions may be, for example, 32 bytes or 64 byte in length. Editing instructions can be shared by the different editing programs for different network operations. Each editing program may include pointers to three sets of data: Static Data; Shared Data (e.g., the [flowID,pkt_cnt,encryption key] of a security flow state that is shared among different packet editing programs); and Dynamic Data (e.g., dedicated data to the program). In addition, there are two sets of global data available to the editing instructions: Meta Data, and Regular Data.

[0013] Fig. 1 illustrates an ASIC 100 according to aspects of the disclosure. The ASIC 100 is adapted to receive a packet, such as the packet 150, and update the packet in accordance with an updated network protocol. According to some examples, the updated network protocol may be a future network protocol that is not currently developed. In this regard, rather than replacing each ASIC in a network each time a protocol is updated, the ASIC and packets transmitted through the network may be efficiently updated.

[0014] The ASIC 100 may be any of a number of ASICs interconnected in a network. According to another example, the ASIC may be designated for protocol updates. While the ASIC 100 is described as having particular components, it should be understood that other components, such as those typically found in an ASIC, may be included. Moreover, while the components are shown as residing within the ASIC 100, it should be understood that the components may reside elsewhere in the network in communication with the ASIC 100.

[0015] The ASIC 100 includes user logic 110 and a packet editor 120. The packet editor 120 further includes at least one editing program 130, including editing instructions 132 and editing data 134, and a central editing unit 140.

[0016] The user logic 110 parses packets received in the ASIC and classifies the packets. For example, the user logic 110 may determine which of a number of editing programs should operate on the packet 150. Different editing programs may be created to update different types of packets. For example, a first editing program may be used to update a packet from a first protocol to a new protocol, while a second editing program is used to update a packet from a second protocol to the new protocol. According to another example, a third editing program may be used to update encapsulated packets, while a third editing program is used to update decapsulated packets.

[0017] Each editing program includes editing data and editing instructions. For example, the editing program 130 includes editing instructions 132 and editing data 134. Further details of the editing program are described below with respect to Fig. 4

[0018] The central editing unit 140 is a processing pipeline which updates the packet 150 according to the editing program identified by the user logic 110. Accordingly, the ASIC 100 can send updated packet 150' through the network.

[0019] Fig. 2 illustrates further details of the central editing unit 140. As shown, the central

editing unit 140 includes a deletion unit 210, an insertion unit 220, and a modification unit 230. Each of these units 210, 220, 230 operates on the incoming packet 150 in accordance with a set of instructions for a particular editing program, such as the instructions 132. As discussed below with respect to Fig. 4, the instructions 132 may include a separate set of instructions for each unit 210, 220, 230. The insertion unit 220 operates on the packet 150 after the packet has been modified by the deletion unit 210. Similarly, the modification unit 230 operates on the packet after the packet has been modified by the insertion unit 220. Each unit may realign a byte stream for the packet after its operation and before passing the packet to the next unit. This central editing unit pipeline enables flexible packet header editing at high bandwidth and packet rate. Moreover, the pipeline provides for efficient hardware implementation in silicon.

[0020] The deletion unit 210 enables the removal of arbitrary protocol headers from the packet. For example, for packets including multiple headers, such as an inner IP header and an outer IP header in a tunneled network, the deletion unit 210 may remove all packet headers or selected packet headers. According to another example, the deletion unit 210 may delete particular fields of packet, while keeping others that are relevant to the updated network protocol.

[0021] The insertion unit 220 enables the insertion of arbitrary protocol headers into the packet. For example, the insertion unit 220 may generate a new set of header fields related to the update network protocol and add these generated fields to the packet 150. In the example where only particular fields are deleted by the deletion unit 210, the insertion unit 220 may determine that the remaining fields do not need to be regenerated, and instead only generate the needed fields. In other examples, the insertion unit 220 may generate the header in its entirety.

[0022] The modification unit 230 enables flexible rewriting, such as overwrite or read-modify-write, of header fields. For example, the modification unit 230 may fill the packet header fields with information enabling the packet to be appropriately transmitted through the updated network.

[0023] As discussed above in connection with Fig. 1, each packet editing program includes editing instructions and an editing data set. For example, a program may have a 16-bit pointer to its editing instructions. According to some examples, editing instructions can be shared by different editing programs. For example, encapsulation under a first network protocol and encapsulation under a second network protocol may utilize the same editing instructions.

[0024] Figs. 3A and 3B illustrate example formats of the editing instructions. In the example of Fig. 3A, the instructions are 32B. Bytes 0-7 are dedicated to deletion, bytes 8-15 are dedicated to insertion, and bytes 16-31 are dedicated to modification. In the example of Fig. 3B, the instructions are 64B. Here, similar to the 32B format, bytes 0-7 are dedicated to deletion. However, in contrast to the 32B format, bytes 8-31 are dedicated to insertion in the 64B format, and bytes 32-63 are dedicated to modification.

[0025] Fig. 4 provides a more detailed illustration of an editing program 430. The editing program 430 includes editing instructions 440 and editing data 450.

[0026] Editing data 450 includes static data 452, shared data 454, and dynamic data 456. For example, each editing program may include pointers to these data sets, which may be stored in a memory in communication with the ASIC. According to one example, the static data 452 and shared data 454 may each be 32B, and the dynamic data 456 may be 32B-64B of data dedicated to the particular editing program. According to one example, multipath load balancing can be supported by selecting the shared/dynamic data sets from a group of shared/dynamic data sets with a hash generated from selected n-tuple header fields.

[0027] Though not shown, additional data may be available to the editing instructions, such as meta data and register data. For example, 32B of meta data may be available, in which the first 8x2B halfword are metadata, such as pkt_length, ip_pkt_id, flow_hash, slice_ID, etc. The second 8x2B halfwords are 2B header fields collected from the packet header, e.g. IP_PROTO, TTL, TOS etc. For example, through the parsing of the user logic, the packet format is identified and 8x2B offsets are specified to select the desired header fields. As another example, 64B of register data may be available to the editing instructions, with 8x8B registers. In this example, Register 0 is hardcoded to all zeros; Register 1 is optionally loaded with 64bit packet count; Register 2 is optionally loaded with 64bit timestamp; and Registers 3-7 are configured by software with static data.

[0028] Editing instructions 440 include delete instruction 442, insert instruction 444, and modify instructions 446. Examples of particular insert instructions 444 are shown, such as <INS NOP> and <INS MOV>, and described in further detail in Figs. 5A-5F. Examples of particular modify instructions 446, such as <MOD NOP> and <MOD WRITE META>, are also shown and described in further detail in Figs. 6A-6F.

[0029] Delete instructions 442 may be, for example, 8B in length, with each bit representing a 2B halfword of a 128B packet header. When a bit is set, the corresponding 2B halfword is deleted from the packet header. A 64bit zero is effectively a no operation (NOP) deletion instruction.

[0030] Insert instructions 444 may be executed by, for example, the insertion unit 220 of Fig. 2. According to one example, the insertion unit 220 maintains an active header pointer (e.g., in units of 2B halfwords) which starts at zero. A MOV instruction, described in further detail below with respect to Fig. 5A, moves the header pointer forward. The insert instruction inserts the specified data at the current active header pointer, and moves the header pointer forward by the number of halfwords inserted. All insert instructions 444 may be, for example, 8bit wide.

[0031] Figs. 5A-5F illustrate example insert instructions 444, including move, and operations on particular data sets. Not shown is a no operation insertion instruction. For example, 8 bits of zeros is a NOP, indicating the end of the insertion instructions.

[0032] Fig. 5A illustrates an example INS MOV instruction. This instruction advances a current insertion pointer by a specified count (cnt). For example, as shown, bits 0-5 include the count, while bits 7-8 are set to 0.

[0033] Fig. 5B illustrates an example INS STATIC instruction. This instruction inserts specified (cnt) halfwords from the Static Data Set (e.g., static data 452 of Fig. 4). If h=0, the data starts from the lower 16B. If h=1, the data starts from the upper 16B.

[0034] Fig. 5C illustrates an example INS SHARED instruction. This instruction may operate on the shared data set (e.g., shared data 454 of Fig. 4) the same way that the INS STATIC instruction operates on the static data set. Similarly, Fig. 5D illustrates an example INS DYNAMIC L instructions that also operates the same way on a first 32B of dynamic data, such as the dynamic data 456 of Fig. 4. Further, Fig. 5E illustrates an example INS DYNAMIC H instructions, which also operates the same way on the second 32B of dynamic data.

[0035] Fig. 5F illustrates an example INS REG instruction, which inserts cnt halfwords from selected global register data.

[0036] Modify instructions 446 may generally operate on a halfword, with optional bits or bytes selected in some instructions. According to some example, two modify instructions may operate on same halfword, each operating on a different byte. MOD NOP indicates a no operation modification instruction, and may be represented by 16 bits of 0. This indicates the end of the modify instructions.

[0037] Fig. 6A illustrates an example MOD WRITE META instruction. According to this instruction, a write operation is performed at the offset (in unit of halfwords) with the selected (sel[3:0]) 2B field from the Meta Data set, according to the specified op_code. Examples of supported write op_code rewriting 4 nibbles of metadata include: 1. ABCD > ABCD; 2. ABCD > ABXX; 3. ABCD > XXCD; 4. ABCD > XXAB; 5. ABCD > CDXX; 6. ABCD > XCDX; 7. ABCD > XXBC. In these examples, "X" indicates that the nibble is not changed. Examples 6 and 7 may be usefully, for example, in copying DSCP/ECN between IPv4 and IPv6 headers.

[0038] Fig. 6B illustrates an example MOD_CHKSUM instruction. This instruction computes the 1's complement checksum (as in IPv4 header) starting at hdr_offset (within the first 64B of the header), for hdr_len halfwords, and writes the computed checksum field at the specified csum_offset location relative to the hdr_offset field. The checksum field itself is not included in the checksum accumulation.

[0039] Fig. 6C illustrates an example MOD_ALU instruction, which executes operations specified by the op_code, for the byte/halfword (specified by byte enable "ben") at the header offset. Examples of supported op_code include: 0- set to zero; 1- set to 1; 2- add 1; 3- add 1 but stops at 8hF or 16hFF; 4- subtract 1; 5- subtract 1 but stops at zero.

[0040] Fig. 6D illustrates an example MOD_CKSUM_INC instruction, which performs an

incremental checksum update of the checksum field at `csum_offset`, with the incremental value specified in `imm_data`.

[0041] Fig. 6E illustrates an example `MOD_WRITE_IMM` instruction, which overwrites the header field at offset with the 16bit immediate data `imm_data`. Each bit of the `nibble_en` enables one nibble in the halfword to be overwritten.

[0042] Fig. 6F illustrates an example `MOD_WRITE_META_IMM` instruction, which writes at the offset (in unit of halfwords) with the result of operations (specified by the `op_code`) between the `imm_data` and the selected (`sel[3:0]`) 2B field from the Meta Data. Examples of supported `op_code` include: 0- Add; 1- Bitwise OR; 2- Bitwise AND.

[0043] While a number of example instructions have been explained, it should be understood that these examples are not exhaustive, and that additional instructions may be used to update the packet headers. Moreover, it should be understood that these example instructions may be modified while still achieving the intended effect.

[0044] Fig. 7 illustrates an example computing device which may be implemented to update the packet headers as described herein. The computing device may be an ASIC or other device, or multiple computing devices or components in communication with each other, including a processor 730 and a memory 720. The computing device 700 may further include one or more input ports 750 for receiving data packets or other information from other computing devices. Similarly, the computing device 700 may have one or more output ports 760 for transmitting packets, for example, after the packet headers have been updated according to the new network protocol.

[0045] The memory 720 stores information accessible by processor 730, including instructions 728, and data 722 that may be executed or otherwise used by the processor 730. The memory 720 may be of any type capable of storing information accessible by the processor, including a computer-readable medium, or other medium that stores data that may be read with the aid of an electronic device, such as a hard-drive, memory card, ROM, RAM, DVD or other optical disks, as well as other write-capable and read-only memories. According to one example, referring back to Fig. 4, the editing data 450 and editing instructions 440 may share a single logic SRAM, enabling dynamic allocation of SRAM resources among them. Systems and methods may include different combinations of the foregoing, whereby different portions of the instructions and data are stored on different types of media.

[0046] The instructions 728 may be any set of instructions to be executed directly (such as machine code) or indirectly (such as scripts) by the processor 730. For example, the instructions may be stored as computer code on the computer-readable medium. In that regard, the terms "instructions" and "programs" may be used interchangeably herein. The instructions may be stored in object code format for direct processing by the processor, or in any other computer language including scripts or collections of independent source code modules that are interpreted on demand or compiled in advance. Upon receipt of a packet, the

processor 730 may execute the instructions 728 to, for example, classify the packet, delete the packet header, insert a new packet header based on the classification, and modify the packet header based on the classification.

[0047] The data 722 may be retrieved, stored or modified by processor 730 in accordance with the instructions 728. For instance, although the system and method is not limited by any particular data structure, the data may be stored in computer registers, in a relational database as a table having a plurality of different fields and records, XML documents or flat files. The data may also be formatted in any computer-readable format. The data may comprise any information sufficient to identify the relevant information, such as numbers, descriptive text, proprietary codes, references to data stored in other areas of the same memory or different memories (including other network locations) or information that is used by a function to calculate the relevant data.

[0048] The processor 730 may be any conventional processor, such as processors in commercially available routers. Alternatively, the processor may be a dedicated controller such as an ASIC or other hardware-based processor. The processor and memory may actually comprise multiple processors and memories that may or may not be stored within the same physical housing. For example, memory may be a hard drive or other storage media located in a server farm of a data center. Accordingly, references to a processor, memory, or computer will be understood to include references to a collection of processors, memories or computers that may or may not operate in parallel.

[0049] The computing device 700 may communicate with other computing devices, for example, over a network. For example, the devices may communicate via a wired connection or wirelessly. The network, and intervening nodes, may comprise various configurations and protocols including the Internet, World Wide Web, intranets, virtual private networks, wide area networks, local networks, private networks using communication protocols proprietary to one or more companies, Ethernet, WiFi (such as 802.11, 802.11b, g, n, or other such standards), and HTTP, and various combinations of the foregoing. Such communication may be facilitated by any device capable of transmitting data to and from other computers, such as modems (e.g., dial-up, cable or fiber optic) and wireless interfaces.

[0050] Fig. 8 illustrates an example method of updating packets to comply with a new network protocol. For example, this method may be performed by a computing device, such as the computing device described above. It should be understood that the operations involved in this method need not be performed in the precise order described. Rather, various operations may be handled in a different order or simultaneously, and operations may be added or omitted.

[0051] In block 810, a packet is received. In block 820, the received packet is classified. For example, classification may include determining a format of the received packet and determining a new protocol in accordance with which the received packet should be updated. According to one example, classification includes determining whether the received packet includes pointers to a particular packet editing program and associated data set.

[0052] In block 830, the packet header is deleted. For example, the header may be deleted in its entirety, or deleted in part based on the new protocol header format.

[0053] In block 840, a new protocol header is inserted based on the classification. For example, the new header may be inserted in whole or in part, based on the deletion. The header may be inserted using, for example, any of the operations described above in connection with Figs. 4 and 5A-F.

[0054] In block 850, the header may be modified. For example, the new header fields may be written and/or existing header fields may be overwritten or modified. Modification of the header fields may be performed using, for example, any of the instructions described above in connection with Figs. 4 and 6A-F. Once the packet header is modified, the updated packet may be output, for example, through an output port of the computing device.

[0055] The systems, methods, and examples described above are advantageous in that they provide for adapting existing networks to new protocols that have not yet been defined. Rather than redesigning and replacing numerous ASICs each time a protocol update is to take place, packet headers may be modified for use by the existing ASICs. In this regard, substantial time, effort, and cost are saved.

REFERENCES CITED IN THE DESCRIPTION

Cited references

This list of references cited by the applicant is for the reader's convenience only. It does not form part of the European patent document. Even though great care has been taken in compiling the references, errors or omissions cannot be excluded and the EPO disclaims all liability in this regard.

Patent documents cited in the description

- [US20020163935A \[0002\]](#)
- [US2001033580A \[0003\]](#)
- [US2013326045A \[0004\]](#)
- [US2005078704A \[0005\]](#)

P A T E N T K R A V

1. Fremgangsmåde til at editere pakker ved anvendelse af en applikationsspecifik integreret kreds, hvor fremgangsmåden omfatter:

5 at modtage en pakke ved én eller flere processorer (100), som omfatter en applikationsspecifik integreret kreds, der omfatter en pakkeeditor (120), hvor pakkeeditoren (120) omfatter en central editeringsenhed (140), hvor den centrale editeringsenhed (140) omfatter en sletteenhed (210), en indsættelsesenhed (220) og en modifikationsenhed (230), hvor pakkeeditoren (120) er programmeret til at understøtte pakkeheader-editeringskrav af nuværende og mulige netværksprotokoller, og omfatter en flerhed af pakkeediteringsprogrammer, hvor hvert pakkeediteringsprogram anvendes til at opdatere en pakke mellem et par af protokoller, hvor hvert par af protokoller er forskelligt fra parret af protokoller af hvert af de andre pakkeediteringsprogrammer; med de én eller flere processorer (100) at klassificere pakken, hvor klassifikationen omfatter med den en eller flere processorer at bestemme en ny protokol, til hvilken pakken skal opdateres, og markører til et bestemt pakkeediteringsprogram og tilknyttet datasæt; med sletteenheden (210) at slette en eksisterende header af pakken; med indsættelsesenheden (220) at indsætte en ny header i pakken baseret på klassifikationen, omfattende at generere nye headerfelter relateret til den nye protokol; og med modifikationsenheden (230) at modificere den nye heade baseret på klassifikationen.

25 2. Fremgangsmåde ifølge krav 1, hvor sletningen af den eksisterende header omfatter at slette udvalgte dele af den eksisterende header, som ikke anvendes af den nye protokol.

30 3. Fremgangsmåde ifølge krav 2, hvor indsættelsen af den nye pakkeheader omfatter at indsætte dele af den nye pakkeheader baseret på den nye protokol og dele, som resterer fra en tidligere pakkeheader.

35 4. Fremgangsmåde ifølge krav 1, hvor modifikationen af den nye header omfatter at eksekvere mindst én af en overskrivningsoperation eller en læs-modificér-skriv-operation.

5. Fremgangsmåde ifølge krav 1, som desuden omfatter at transmittere pakken omfattende den nye header gennem et netværk baseret på den nye header.

6. System som omfatter:

- mindst én hukommelse; og
- mindst én processor (100) i kommunikation med den mindst ene hukommelse, hvor den mindst ene processor (100) omfatter en applikationsspecifik integreret kreds, der omfatter en pakkeeditor, hvor pakkeeditoren (120) omfatter en central editeringsenhed (140), hvor den centrale editeringsenhed (140) omfatter en sletteenhed (210), en indsættelsesenhed (220) og en modifikationsenhed (230), hvor pakkeeditoren (120) er programmeret til at understøtte pakkeheader-editeringskrav af netværksprotokoller, og omfatter en flerhed af pakkeediteringsprogrammer, hvor hvert pakkeediteringsprogram anvendes til at opdatere en pakke mellem et par af protokoller, hvor hvert par af protokoller er forskelligt fra parret af protokoller af hvert af de andre pakkeediteringsprogrammer, hvor den mindst ene processor er konfigureret til:
- at modtage en pakke;
 - at klassificere pakken, hvor klassifikationen omfatter at bestemme en ny protokol, til hvilken pakken skal opdateres, og markører til et bestemt pakkeediteringsprogram og tilknyttet datasæt;
 - ved hjælp af sletteenheden (210) at slette en eksisterende header af pakken;
 - ved hjælp af indsættelsesenheden (220) at indsætte en ny header i pakken baseret på klassifikationen; og
 - ved hjælp af modifikationsenheden (230) at modificere den nye header baseret på klassifikationen.

25

7. System ifølge krav 6, hvor den mindst ene hukommelse omfatter mindst ét pakkeediteringsprogram, hvor hvert pakkeediteringsprogram omfatter markører til dets editeringsinstruktioner og markører til sæt af statisk data, delt data og dynamisk data.

30

8. System ifølge krav 6, hvor bestemmelsen af den nye protokol, til hvilken pakken skal opdateres, omfatter at identificere markører til et pakkeediteringsprogram.

35

9. System ifølge krav 6, hvor sletningen af den eksisterende pakkeheader omfatter at slette udvalgte dele af den eksisterende header, som ikke anvendes af den nye protokol.

10. System ifølge krav 9, hvor indsættelsen af den nye pakkeheader omfatter at indsætte dele af den nye pakkeheader baseret på den nye protokol og dele, som resterer fra en tidligere pakkeheader.

11. System ifølge krav 6, hvor modifikationen af den nye pakkeheader omfatter at eksekvere mindst én af en overskrivningsoperation eller en læs-modificér-skriv-operation.

12. Fremgangsmåde ifølge krav 6, hvor den mindst ene processor desuden er konfigureret til at transmittere pakken omfattende den nye header gennem et netværk baseret på den nye header.

13. Ikke-transitorisk computerlæsbart medie, som lagrer instruktioner, der kan eksekveres af én eller flere processorer for at udføre en fremgangsmåde ifølge ethvert af kravene 1 til 5.

DRAWINGS

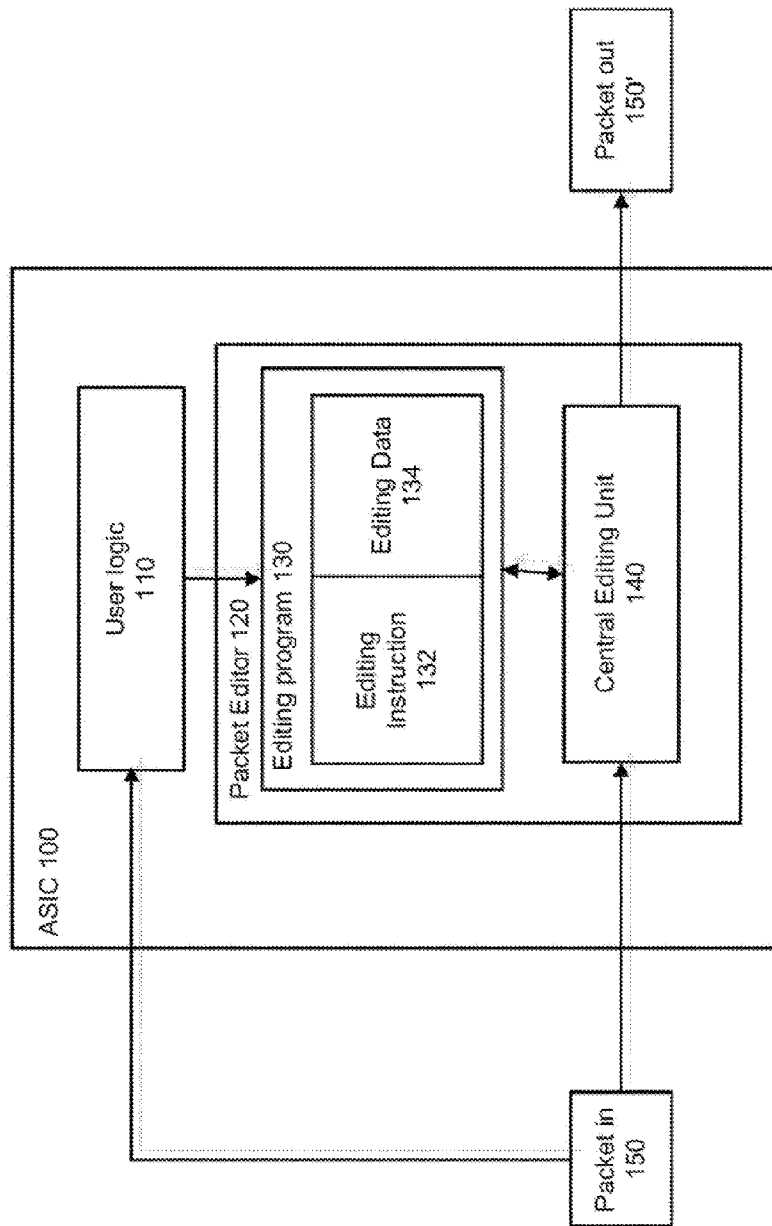


Fig. 1

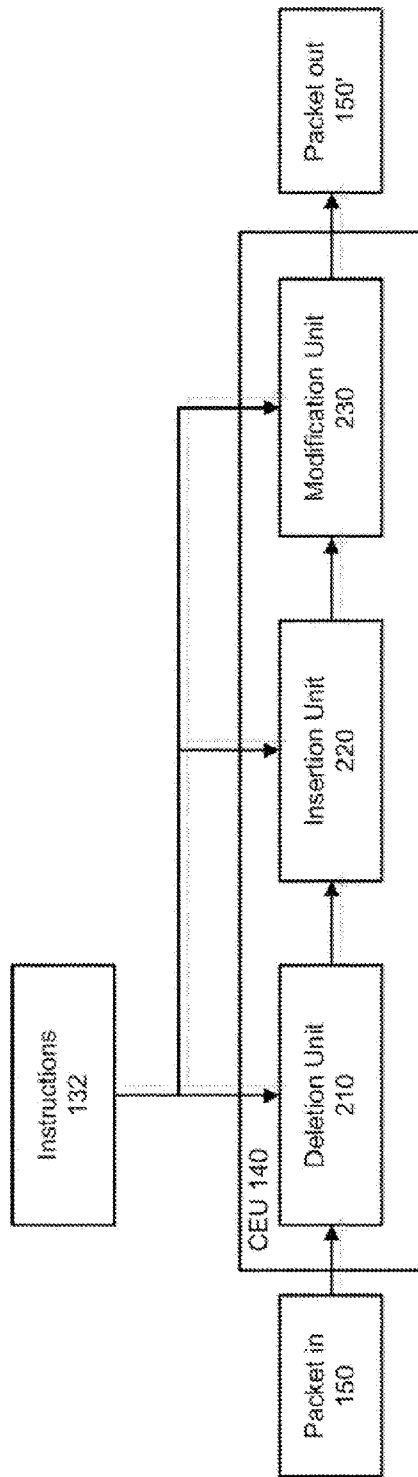


Fig. 2

Bytes 0:7	Bytes 8:15	Bytes 16:31
Delete Instructions	Insert Instructions	Modify Instructions

Fig. 3A

Bytes 0:7	Bytes 8:31	Bytes 32:63
Delete Instructions	Insert Instructions	Modify Instructions

Fig. 3B

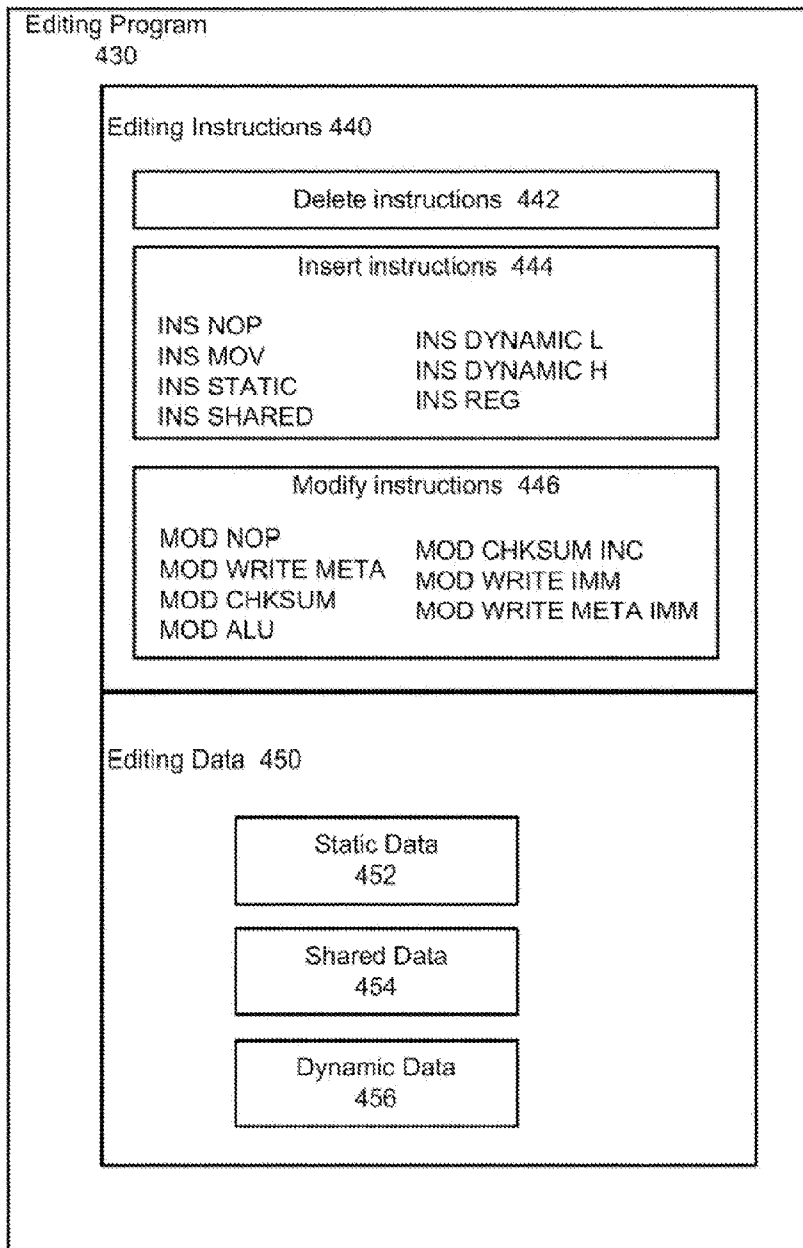


Fig. 4

Fig. 5A

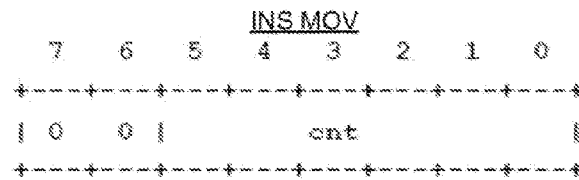


Fig. 5B

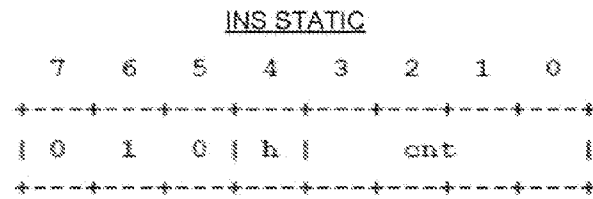


Fig. 5C

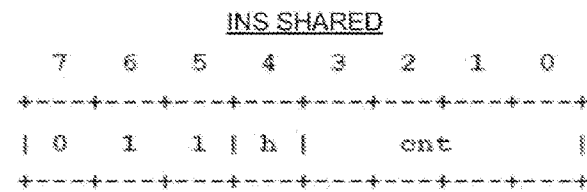


Fig. 5D

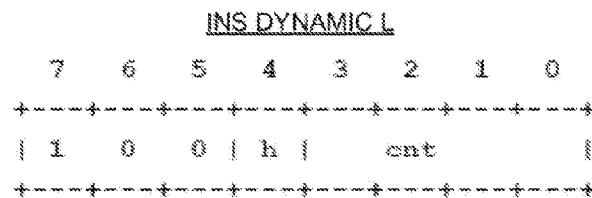


Fig. 5E

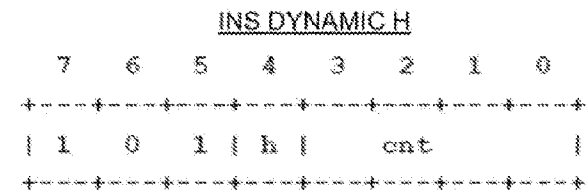


Fig. 5F

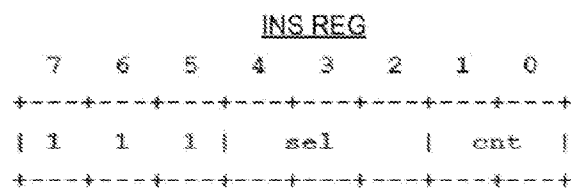


Fig. 6A

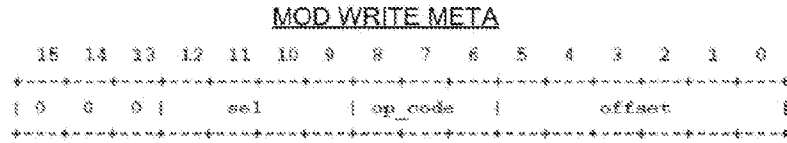


Fig. 6B

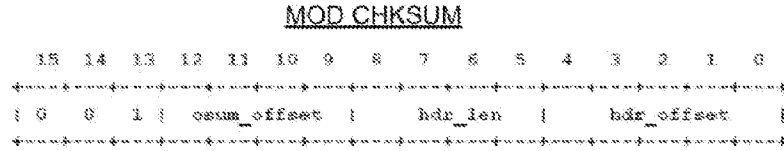


Fig. 6C

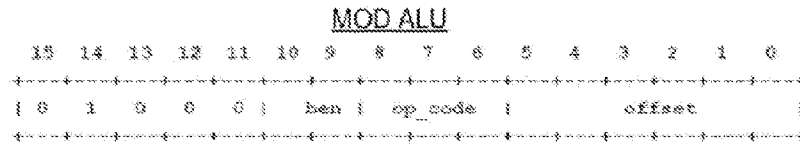


Fig. 6D



Fig. 6E

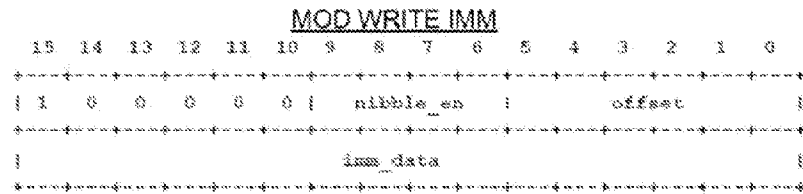
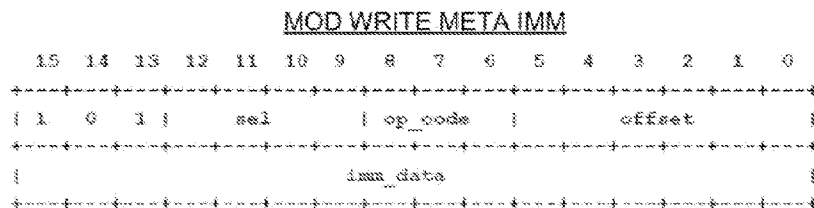


Fig. 6F



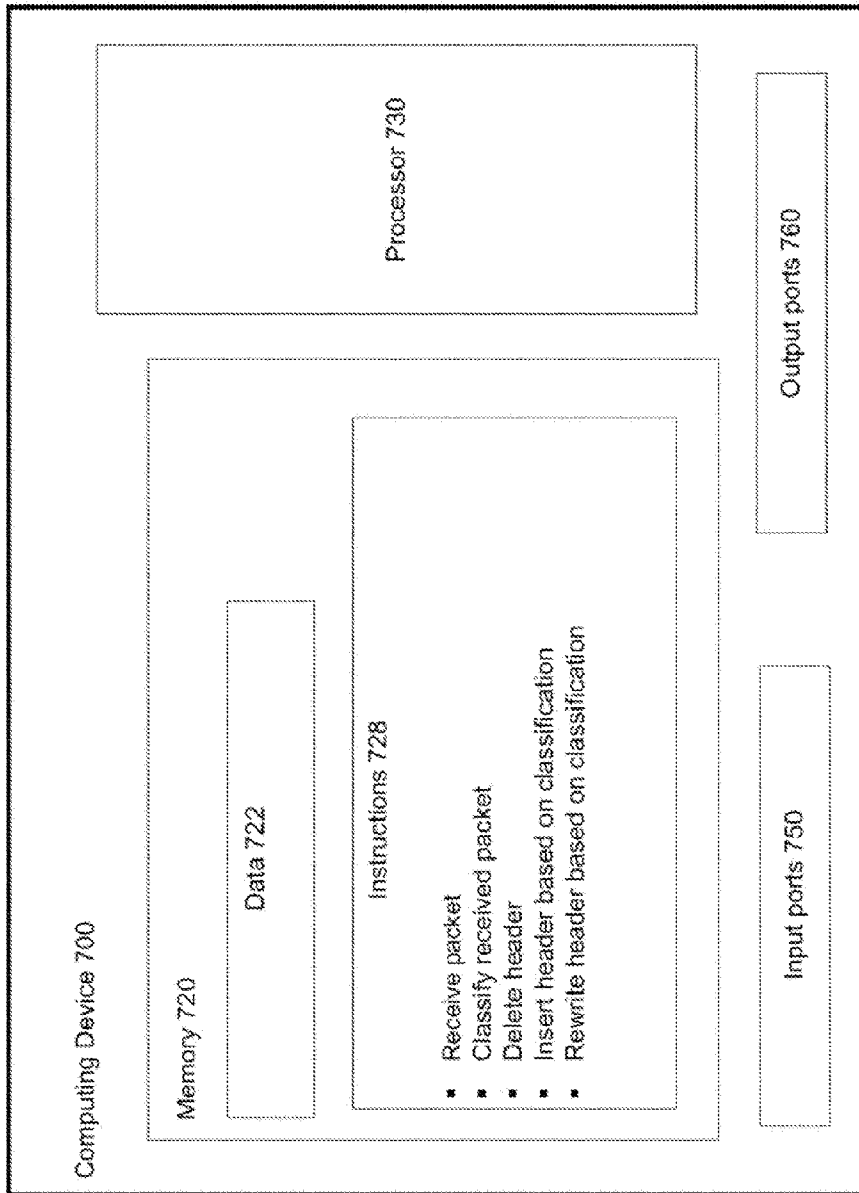


Fig. 7

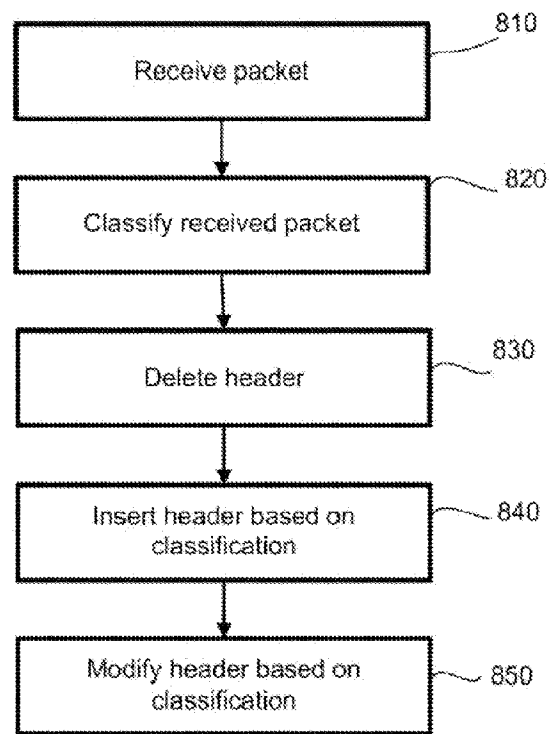
800

Fig. 8