A large capacity programmable packet forwarding device

1st Qinshu Chen[1], 2nd Hua Lu[1*], 3rd Fusheng Zhu[1], 4th Hui Li[2]
{chenqinshu@gdcni.cn[1], luhua@gdcni.cn[2], zhufusheng@gdcni.cn[3]}

Guangdong Communications&Networks Institute[1]

Abstract. In the 5G/B5G era, the need of various new services for network bandwidth, delay and cost is far different, so various emerging services may have their own connective protocols according to their own special applications, which requires the core exchange equipment in the network to support programmability. However, neither using network processor to realize packets forwarding nor using fixed pipeline to realize packets forwarding can not meet the needs of 5G/B5G new applications. In this paper, a new architecture of switching chip is proposed. By using programmable packets parser, programmable packet process unit and programmable packet editer, the requirements of 5G/B5G new applications' rapid deployment, large capacity, low delay and low energy consumption are met.

Keywords: programmable packet parser, programmable packet processor, programmable packet editer

1 Introduction

In 5G/B5G era, with the rise of new services such as automatic driving, telemedicine and virtual reality, various emerging services may define their own communication protocols according to their own special applications due to the distinct demands of various new services for network bandwidth, delay and cost, requiring the core equipment in the network to support programmability. The programmability of services depends on that of switching chip, making it necessary to have programmable packet parser, programmable packet process and programmable editing function in the switching chip of the switching equipment. At present, the implementation of switching chip either uses network processor to make packets forwarded or uses fixed pipeline to make packets forwarded. Although making packets forwarded by using network processor supports software programming according to the needs of services, it leads to large search delay and inability to support large bandwidth, can not reach the level of T bps switching capacity and not meet the requirements of 5G/B5G high-capacity & low delay due to the multi-core time-sharing reuse of the same piece of RAM; while fixed pipeline forwarding can support the level of T bps switching capacity, but once the design is completed, it cannot be modified through software programming to satisfy the needs of 5G/B5G new applications' rapid deployments due to

*Cooresponding author: Hua Lu. {luhua@gdcni.cn}. Guangdong Communications & Networks Institute.

EAI MOBIMEDIA 2020, August 27-28, Harbin, People's Republic of China
Copyright © 2020 EAI
DOI 10.4108/eai.27-8-2020.2294711

the need to design in accordance with the networks protocol in advance. These contradictions become more and more serious with the development of 5G/B5G mobile Internet and Internet of things. In this paper, we study a new programmable switching architecture and realize programmable packet parser, programmable packet processing unit and programmable packet editer to solve the above problems.

2 Programmable packet process flow

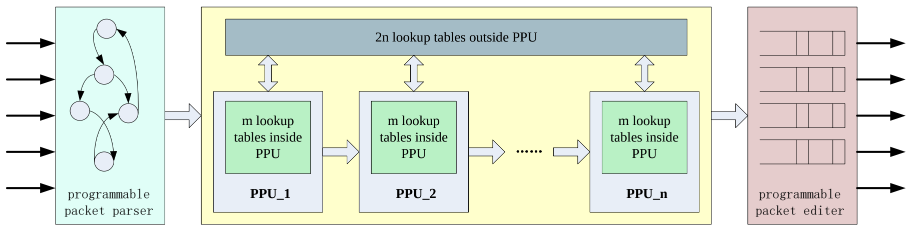

2n lookup tables outside PPU
m lookup tables inside PPU
PPU_1
m lookup tables inside PPU
PPU_2
......
m lookup tables inside PPU
PPU_n
programmable packet parser
programmable packet editer

Fig. 1. Programeble whitch chip block diagram

The structure of programmable switch chip is shown in 0, which is composed of programmable packet parser module, several packet processing units and programmable packet editer module. Firstly, the packets enter the programmable packet parser module for analysis, obtaining the structure of the packet, putting the parse results (mainly packet type, destination MAC address, source MAC address, VLAN, ETH type, DIP, SIP, IP protocol number etc. The keys carried by the packet in this paper can be user-defined keys, not limited to the keys described above) into the meta-data of the packet and send them to the subsequent module PPU (packet process unit) for processing. The structure of each PPU is the same, as shown in 0. Each PPU takes the corresponding keys from the meta-data and those configured in the lookup table to match. If matched, the contents of the corresponding entries in the table are taken out and put into the meta-data of the packet, and then sent to the next PPU for processing. After processed by PPUs, the meta-data of the packet is sent to the packet editer module for generating new packet. Finally, forward the packet to the destination port indicated by the meta-data.

3 Programmable packet parser

The main function of the programmable packet parser is to analyze the packet according to the packet parse database configured by users and move the key fields of the packet to the meta-data. The parse flow is to determine the next node (i.e. the next state) of the packet according to the current node state in the parse tree and the matching keys extracted from the packet, then jump to a new node in the packet parse tree, and then to determine the next node (i.e. the next state) in the parse tree according to the state of the new node and the next keys to be matched, and so on to the end node of the packet parse tree.

To realize the parse process described above, a parse database is required which is composed of the matching rule table and the action table. The matching rule table in the parse database needs to support the masked matching function, as can be realized by the ternary content addressable memory. The matching rule table needs to store the keywords to be compared and the current status, while the former need to support masked matching as its length may be less than the width of the matching rule table and the bits not concerned can be masked. The action table stores the next state, the address of the key to be extracted from the packet next time and the operation instructions that are needed to move the keys of the packet header to the meta-data.

The packet parser first obtains one initial state and four initial offset addresses according to the input port register, obtaining four corresponding keys from the packet according to the latter. After splicing these keys with the initial state looks up to the matching rule table. If hitting the entry of the matching rule table, it reads the action table according to the address of the entry, obtaining the next state and the four offset addresses from the action table in the database. Each key offset can extract one byte key from the packet header, while data offset and data length, corresponding to each other one by one, are given in the packet parse database. Data length is two bits, indicating how length can be extracted by data offset which can be one byte, two bytes, four bytes and six bytes. Once matching can get up to eight valid data offsets and data lengths, that is to say, up to eight fields can be extracted from the packet and put on the meta-data. According to the key offset obtained from the matched entry of the packet parse database, the keyword corresponding to four bytes is extracted from the packet, after that the next state is obtained from this table, with the corresponding search result obtained from the packet parse database again, that is, the next state and the offset addresses of four matched keys. At the same time, the packet parse database provides up to eight offset addresses and data lengths. According to the offset addresses and data lengths, up to eight data are extracted from the packet and stored on the meta-data. Then the process described above is repeated to match the packet parse database again until the next state is 255. Since the packet parse database can be configured by software, the packet parser can analyze any packet by configuring packet parse database according to the format of the packet.

4 Programmable packet process unit

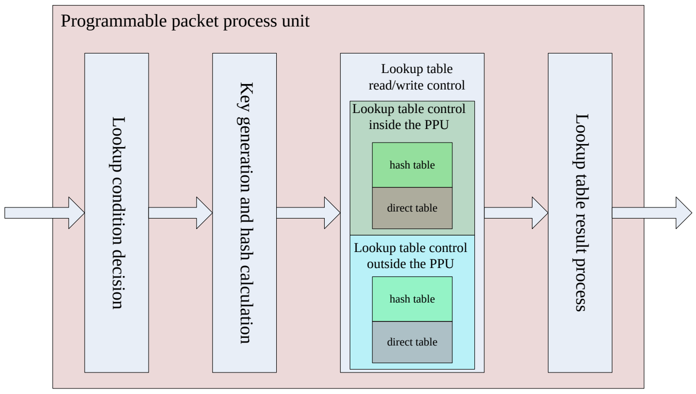

Programmable packet process unit
Lookup condition decision
Key generation and hash calculation
Lookup table read/write control
Lookup table control inside the PPU
hash table
direct table
Lookup table control outside the PPU
hash table
direct table
Lookup table result process

Fig. 2. Programmable packet process unit

The whole programmable packet process unit is composed of the lookup table condition decision module, lookup key generation & hash calculation module, lookup table read/write control module and lookup table result process module, as shown in 0.

4.1 Lookup condition decision

The conditional decision module of lookup table consists of m lookup table decision makers, each of which is composed of i j-bit comparators that is able to perform the judgment of greater than, less than, equal and not equal; each comparator has a control register which stores the immediate number or the address of the match key in the packet meta-data and the flag of the match key is an immediate or not. If it is an immediate number, the value of the compared number is stored; if not, it’s the match key address of the packet meta-data. Simultaneously, the mask of the match key is also included. Before the comparison operation, the match key first needs to do and operation with the mask register by bit, then performing the comparison operation. In the end, the judgment results of i comparators and k key fields from the meta-data according to the address indicated by the lookup table adjudicator are combined to search the TCAM. The m lookup table decision makers can be used independently, each of which corresponds to a lookup table; on the other hand, multiple lookup table decision makers can be used jointly to search a lookup table as a decision condition, at that time the number of lookup tables supported by the PPU becomes smaller yet the supported lookup table decision conditions are more complex than the single lookup table decision maker. The flag of whether the lookup table inside the PPU or outside the PPU and the number of the lookup table to indicate which lookup table to look up are given before it is read.

4.2 Key generation and hash calculation

Each lookup table contains *p* configuration registers, which are used to indicate how to generate keywords to search the lookup table. Each register contains a byte of the corresponding key in the position of the packet meta-data and the bit length of the byte's valid bit. In terms of the register, each byte of the match key is extracted from the meta-data, which extract the bytes that are combined into a temporary lookup table key according to the register's order. Finally, the invalid bits inside the temporary lookup table key are wiped to generate the lookup table key. According to the configuration, the corresponding hash function is selected to calculate the hash index. In this paper, the double hash lookup is supported. Thus, for each lookup key, two different hash functions are used to calculate its hash index simultaneously.

4.3 Lookup table read/write control

PPU lookup table read/write control is to flexibly change the size of each lookup table by configuring registers according to the work scenario of the switch, so that the packet forwarding chip can meet the needs of various services in various scenarios as well as make the best use of RAM resources. This module is divided into the read/write control of the lookup table inside the PPU and that outside the PPU, but the two of lookup table read-write control modules are almost the same.

The hash table attribute register includes the depth of the hash table, the byte width of the hash table, the bit width of the hash table, whether the hash table supports single hash or double hash, whether the hash table carries the statistical counter pointer or not and the base address of the statistical counter pointer, whether the hash table carries the flag of the instruction pointer or not and the base address of the instruction pointer, etc. At the same time, each RAM inside the PPU read/write control module also contains the attribute register describing the RAM block. The register includes the logical table number of the RAM, the RAM block's location in the row and column of the lookup table and the hash function number support by the RAM block. In order to reduce the hash conflict, multiple entries are usually stored in a hash index.

When the hash table read/write control logic inside the PPU receives the lookup table request, it is decoded into the logical table number of the hash table in accordance with the request source. Each RAM block in the PPU checks its RAM attribute register according to the request and the address of lookup table, then checking the logic number of the request lookup table and the number of configured in RAM block attribute register, judging the address of the lookup table in the RAM block of the lookup table. In the meantime, the RAM address is generated according to lookup table address and the RAM is read, after which the data is put into the result register according to the logical table column number in the attribute register of RAM block.

The hash_mux_result is divided in light of the table byte width configuration in the attribute register of the hash table, finally obtaining multiple hash_result data. The hash table supports a hash index to contain 8 entries in general, causing hash_mux_result to split 8 hash_result data of the same byte width in the end. The width of partitions and the number of data to be partitioned are all configured by software. Then the match key, whose width is configured by the hash table attribute register, is taken out from the hash_result as well as compared with the hash table. If match, the final result of the hash table is taken out according to the property register configuration of the hash table, including the instruction pointer offset and the statistics count offset. Afterwards, the instruction pointer offset and the instruction pointer base address configured in the attribute register of the table are added to send to the lookup table result processing module as the instruction pointer. At last, the offset of the statistics count and the base address of the statistics count in the attribute register of the lookup table are added to generate the statistics count pointer, after which initiate the request of searching the statistics count lookup table.

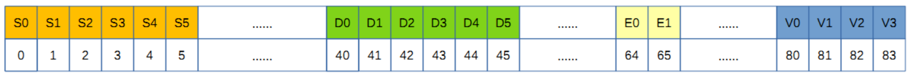
4.4 Lookup table result process

The lookup table result process module is mainly based on the processing results of the previous PPU or the packet parser module and the results of this lookup table to merge and generate a new meta-data, transferring to the next PPU or programmable packet editer module. The lookup table process module generates an instruction database, each entry of which contains u ALU (arithmetic logical unit) instructions. Each instruction can perform the following actions:

- Move the lookup table result data to the meta-data
- Move the data on the meta-data to other locations of the meta-data
- Perform logical operation, including operation and, or, not, exclusive-OR
- Perform arithmetic operation, add and subtract, but not multiply or divide
- Perform shift operations including left shift and right shift
- Set the immediate number to the meta-data

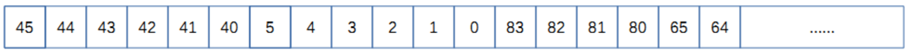
5 Programmable packet editer

The programmable packet editer module uses the results of the PPU process to extract the meta-data from the key field and merge them into a new packet header, then taking out the payload from the packet buffer, splicing the new packet header and the payload to a new packet. The principle of the programmable packet editer is that the sequence of protocol header key fields of a packet is sequential and fixed, for example, a UDP packet must be Ethernet L2 header + IPv4 header + UDP header + payload; a TCP packet must be Ethernet L2 header + IPv4 header + TCP header + payload, while the packet of Ethernet L2 header + UDP header + IPv4 header + payload will not occur. Therefore, we only need to build a most complete protocol header as well as define the location where each protocol header key fields exist in the meta-data as a packet editing database. During packet editing, we can take out each byte from the meta-data according to the location indicated by the packet editing database and then splice it to form a new packet header.



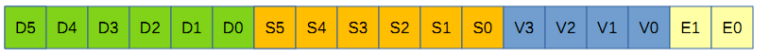
S0	S1	S2	S3	S4	S5	......	D0	D1	D2	D3	D4	D5	......	E0	E1	......	V0	V1	V2	V3
0	1	2	3	4	5	......	40	41	42	43	44	45	......	64	65	......	80	81	82	83

Fig. 3. the meta-data send to packet editer by PPU



45	44	43	42	41	40	5	4	3	2	1	0	83	82	81	80	65	64	......

Fig. 4. packet editing database



D5	D4	D3	D2	D1	D0	S5	S4	S3	S2	S1	S0	V3	V2	V1	V0	E1	E0

Fig. 5. the new edited packet header

In order to better describe the above packet editing flow, this paper presents the L2 packet editing process. The first line of 0 is the meta-data content sent by the previous PPU, while the second line describes the address of each byte on the meta-data, which does not exist in reality, just for the convenience of description. On the meta-data, the 0 to 5 bytes store the SA (MAC source address) of the packet which is 48 bits and 6 bytes in total. The highest byte of SA is S5 which is stored in the position 5 of the meta-data, with the second highest byte S4 stored in the position 4 of that, the other bytes are stored as shown in 0, which is not described in this paper. In 0, the orange field is SA field, 6 bytes; the green field is DA field, 6 bytes; the yellow field is Ethernet type field, 2 bytes; the blue field is VLAN tag field, 4 bytes, including 16 bits VLAN tag protocol field, 3 bits priority COS field, 1 bit CF field and 12 bits VLAN ID field.

The packet editing database stores the position information of the corresponding field of the packet in the meta-data, as shown in 0. The first is the DA field of the packet, which has 6 bytes in total. There lies the location of the packet in the meta-data, with the next one of the SA field in the meta-data, the third one of the VLAN field in the meta-data and the last one of the Ethernet type field in the meta-data. During packet editing, the corresponding fields are taken out from the meta-data according to the address stored in the packet editing database, spliced into packets. For example, when editing a packet, the highest byte of DA is taken from the packet editing database at the address 45 of the meta-data, using which to take the corresponding content D5 from the meta-data, placed at the position of the first byte of the packet. Then the position 44 of the next byte of DA is taken from the packet editing database, from which of the meta-data takes the content D4, spliced after the D5. Similar is the other fields until a new packet header is assembled finally as in 0. Afterwards, a new data packet composed of the original payload is spliced with a new packet header, forwarded to the output port indicated by the meta-data.

6 Summary

In this paper, a large capacity programmable packet forwarding scheme is proposed to solve the problem that the current network processor cannot meet the requirements of low latency and large capacity exchange as well as to avoid the problem that the fixed pipeline forwarding cannot meet that of the rapid deployment of new services. The forwarding behavior can be changed according to the needs of the services by software programming, which can meet the needs of 5G/B5G new services rapid deployment, large capacity, low latency and low energy consumption.

Acknowledgement. This work was supported by the National Key Research and Development Program of China under Grant (2019YFB1804400).

References

1. Evans, Dave. The Internet of Things How the Next Evolution of the Internet Is Changing Everything. CISCO white paper (2011).
2. Liotou, E.; Tseliou, G.; Samdanis, K.; Tsolkas, D.; Adelantado, F.; Verikoukis, C., Multi-tenancy for Virtualized Network Functions, in Quality of Multimedia Experience (QoMEX), 2015 Seventh International Workshop on. pp.1-2, May 2015.
3. S. Scott-Hayward, G. OCallaghan, and S. Sezer, SSDN security: A survey, in Proceedings of the IEEE SDN for Future Networks and Services. pp.1-7, 2013.
4. O. Flauzac, C. Gonzalez, and F. Nolot SDN Based Architecture for Clustered WSN. Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 9th International Conference on, Blumenau, 2015, pp. 342-347.
5. El-Mougy, Amr; Ibnkahla, Mohamed; Hegazy, Lobna, Software-defined wireless network architectures for the Internet-of-Things, in Local Computer Networks Conference Workshops (LCN Workshops), pp.804-811, Oct. 2015.