國立陽明交通大學

資訊學院 資訊工程學系

大學部專題論文

RDMA-based Adaptive Remote Lookup using P4

組員一: 0716206 陳昱丞

組員二: 0716221 余忠旻

組員三: 0717035 王偉軒

指導教授: 林靖茹

中文摘要

在一個複雜的 SDN 網路架構中，交換機上的路由表有可能因記憶體空間有限而不敷使用，此時需要向遠端控制器詢問路由規則，造成延遲。本研究提出一個路由規則安裝機制，將大流量資料的路由資訊盡可能存放在交換機記憶體中，而小流量資料路由資訊則留在遠端控制台。本研究成功設計出一套有效的路由規則遷移機制，使用 P4 語言、count- min sketch 演算法與 RDMA 機制，以達到即時且有效率的路由更新。此路由搬遷演算法能使交換機路由表使用率高達 88%，而 RDMA 根據不同的負載量，能使交換機和遠端控制器的溝通延遲縮短達最多 7 倍。本架構不僅能大幅降低交換機與遠端控制器的溝通次數及延遲，增進網路效能與路由資訊即時性，也能作為一種新型且高效的網路遙測手段。

Abstract

In a complex SDN network architecture, a switch usually has a limited memory size, which cannot install the forwarding rules for all the flows. To overcome this limitation, an SDN integrates local table lookup with remote table lookup to ensure reliable forwarding. However, remote lookup typically incurs a much longer latency, which would lower the overall throughput of a network. In this project, we introduce an efficient rule migration algorithm that installs the rules of heavy flows in the data plane while migrating the rules of light flows to the remote controller. Since traffic patterns change dynamically, we designed a P4 pipeline that leverages count-min sketch and RDMA to enable adaptive rule migration. Using our migration algorithm, the usage rate of a switch's routing table reaches 88%. Furthermore, the signaling overhead reduces at most 7 times according to the load of RDMA. We have shown that the proposed migration algorithm not only enhances forwarding efficiency but also reduces signaling overhead. This can additionally become a new and efficient network telemetry method.

List of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1. Research Motivation and Questions

In recent years, due to the rapid development and growth of cloud computing, big data and other technologies, SDN (Software Define Network) has become one of the most popular network technologies.

In traditional network architecture, switches include a control plane and a data plane. This design let network managers have no flexibility in changing switch behavior. Together with the fact that the design of the switch does not have a lot of computing power. This design such that the switch cannot combine with the huge amount of data processing such as cloud computing and machine learning. It is inappropriate in this era when the amount of network routing messages is increasing.

The SDN separates the control layer from the data layer of the switch, and the control layer is managed by a large server, which is the remote controller of the entire network. The switch only needs to transmit data according to the routing messages given by the remote controller. This makes network managers easier to manage the state of the entire network. By controlling only one remote controller, they can operate an infinite number of switches without actually managing the behavior of the switches to achieve so-called network virtualization. In addition, the remote controller has a large amount of computing power and can use deep learning to calculate the best transmission path and send the path information to the switch for execution.

Traditional transport protocols, such as OSPF (Open Shortest Path First) or RIP (Routing Information Protocol), need to transmit additional packets or headers to allow switches to exchange path messages with each other and update their routing tables. However, in SDN, all Path-Related messages are centrally managed by remote controllers. Switches only need to perform packet transfers, do not need to compute. This can eliminate redundant packets and headers, alleviate the overall burden on the network. The flexibility of path calculation has also become richer and more diverse. Traditional networks use Dijkstra's algorithm or Bellman-Ford algorithm to calculate the best route. However, now due to the larger amount of routing table information and the more complex network environment, traditional algorithms are becoming less flexible in the face of fast-changing network conditions. SDN allows a flexible management network. Unlike traditional network systems, routers can only find the shortest path, which may cause congestion. SDN can instantly monitor network traffic, dynamically and effectively shunt by algorithms to avoid congestion and optimize network resource utilization.

The recently proposed P4 (Programming Protocol-Independent Packet Processors) switch language is not limited to the hardware and protocol used, only the corresponding compiler is required to complete the network deployment. Its flexibility makes it become the most suitable “programmable switch language” for SDN. In the SNN architecture, the remote controller controls the packet forwarding tables (flow tables) of the switch and communicates with the switch via RPC (Remote Procedure Calls) or socket. However, to achieve dynamic routing, data routing mechanisms need to be installed in switches. When network traffic is too large, the routing table will be inadequate due to the limited memory space of the switches. If the table cannot find routing information, it needs to ask the remote controller for routing rules. Communicating with remote controllers takes much longer time than looking up tables directly in the switch. Therefore, this project will design an installation algorithm for routing information so that more frequently used packet forwarding tables are installed in the memory of the switch, and less frequently used packet forwarding tables are stored on the remote side. This reduces the number of remote message exchanges and the burden on the network. In addition, this study introduces RDMA (Remote Direct Memory Access) technology to replace RPC communication and further increase routing management performance.

In addition, while performing routing installation algorithms, we need to count the flow of each flow through a switch. We will use the P4 register function to support the transfer of RDMA. This is a new method of network telemetry. In the past, many methods of network telemetry had some unavoidable shortcomings. For example, although IPMP (IP Measurement Protocol) can measure information such as packet loss rate, it will occupy a lot of network bandwidth and affect operation. In-band Network Telemetry (INT) can reduce network bandwidth and provide more comprehensive information, but its ever-increasing INT Metadata is prone to cause problems when forwarding measurements through switches. In our method, traffic information is stored directly in each P4 switch. No Metadata and no extra packet bandwidth are required, and each switch connects to the Controller via RDMA to transmit the remote data through RDMA in the most immediate way.

Therefore, the method we designed can not only make the complex network architecture more efficient for transmission, but also for efficient network telemetry. It is a good way to get the best of both worlds.

2. Literature Review and Discussion

To perform the RDMA transfer, both ends of the transport must first support RDMA on the hardware. Infiniband(IB) is a computer network communication technology for efficient computing, and RDMA must be performed on hardware devices that have this technology. Currently, these hardwares, such as switches and server adapter wafers, are mainly produced by companies such as Intel and Mellanox.

RDMA achieves transmission acceleration through CPU bypass and zero-copy design [1]. It uses the operating system to create a dedicated transmission channel through which applications can receive data directly without the participation of the operating system and CPU, thus enabling CPU bypass. Traditional TCP/IP transmission requires data replication at both ends of the channel, whereas RDMA transmission does not require data replication. When a local application receives an RDMA Read/Write request, the IB hardware packages the data and sends it directly to the virtual buffer of the target application, where it reorganizes the package back to the original data. There is no need to copy data from the beginning to the end of transmission, which is the so-called zero-copy mechanism.

On software, RDMA can be programmed by calling the API in C. Two important operations are as follows[2]: (1) RDMA Read - Caller host specifies the remote virtual address for reading. Before calling, the remote host must set the permissions of the target memory to be readable. After the operation, the remote host will not be notified when the operation is complete. (2) RDMA Write / RDMA Write With Immediate - The former is the same as Read, but changes data reading to data writing; In addition, the data writing operation will notify the remote host that the operation has ended after the operation is completed.

In the study by Beltman et al. [3], they wanted to make a server capable of efficiently collecting network data through RDMA. They wanted to use RDMA to write packets to the server while the P4 switch was receiving them. They created a network topology that connects servers to each other. Once servers initially pass some initial parameters to the switches for establishing RDMA connections over TCP, they can connect to each other using RDMA. They implemented the RoCEv1 (RDMA over Converged Ethernet) protocol [4] with P4, successfully enabling the packets of this protocol to be transmitted to the P4 switch, and successfully writing the payload to the server's file through the RDMA Write operation. This confirms the existence of P4 switch binding to RDMA. Their research is to write data from the switch to the server, and the plan is to write data from the server back into the memory of the switch.

In a study by Beltman et al. [3], they wanted to make a server capable of efficiently collecting network data through RDMA. They wanted to use RDMA to write packets to the server while the P4 switch was receiving them. They created a network topology that connects servers to each other. Once servers initially pass some initial parameters to the switches for establishing RDMA connections over TCP, they can connect to each other using RDMA. They implemented the RoCEv1 (RDMA over Converged Ethernet) protocol [4] with P4, successfully enabling the packets of this protocol to be transmitted to the P4 switch, and successfully writing the payload to the server's file through the

RDMA Write operation. This result confirms that it is possible to bind the P4 switch with RDMA. Their research is to write the information of the switch to the server, and our plan is to write the data of the server into the memory of the switch.

The hardware or software that executes the P4 language is called the P4 target. In the P4 architecture, there is an intermediate layer on the P4 target. After finishing the P4 programming, it first passes through a front-end compiler, converts the P4 program into a target-independent HLIR (High-Level Intermediate Representation), then dismantles and assigns the corresponding parts of the program to the corresponding hardware (P4 target). Finally, the hardware is converted into executable machine code by a back-end compiler for the hardware to execute. The part in P4 that controls the packet forwarding path is MATs (Match-Action Tables). In the SDN architecture, the P4 program only needs to declare MATs, and its entries are controlled remotely by the control layer through the P4 Runtime API. Remote servers and P4 switches communicate through gRPC [5].

In the study by Yuhan Xue et al. [6], they tried to let switches read data using RDMA instead of using traditional gRPC. The switch here uses a PDP (Programmable Data Plane) switch, of which the P4 switch is one. Their research points out that SRAM (Static Random-Access Memory) on switches can only store thousands of packet forwarding tables, which is not enough to store today's huge amount of network path information. They created a system structure that allows the switch to first search its SRAM space when finding a path. If the table miss, it requests path data from the server through RDMA, and the server returns the data to the switch for execution. A server can store a large amount of data, and for a switch, the server's memory is equivalent to the switch's external memory.

In this study, we hope that the server not only transmits path data to the switch for execution but also writes it to the memory of the switch through RDMA operations. There are usually two memory devices on a programmable switch: TCAM (Ternary Content Addressable Memory) and RAM (Random Access Memory) [8,9]. The advantage of having memory is that if this packet forwarding table is reused, it can be found directly from the memory of the switch, thereby eliminating redundant RDMA access time. This study will measure whether this method can effectively increase network performance by transmitting a large number of packets and a large number of different packet forwarding tables.

In the study by Isaias Martinez-Yelmo et al. [7], they wanted to actually make a switch that uses P4 Runtime to control the packet forwarding path, but if the path does not exist in the remote controller, the switch automatically uses the traditional ARP (Address Resolution Protocol) Path for the packet forwarding. They implemented this idea using P4 extern syntax with the BMv2-based switch. However, when a large number of packets are transmitted, the ARP performance of this switch is worse than that of traditional ARP-only switches. The reason is that BMv2-based switches use RR (Round Robin) mechanisms to control the order in which packets are forwarded, not what they think of as FIFO (First In First Out).

The algorithm in this study uses registers in P4 to implement count-min sketch[13] to estimate traffic. Count-min sketch is a sketch method that utilizes the Bloom Filters algorithm. It is stored as a two-dimensional array and indexed by several independent hash functions, taking out the values in the array and adding one. Count-min sketch uses hash function calculations to reduce space usage. According to the data results [14], it can get rid of Ω (1/ε^2) which is the traditional lower bond of space complexity when dealing with data flow problems, let the space complexity of sketch reduce to the upper bond of space complexity O (1/ε), significantly reduce the memory space used by sketch. However, the cost of reducing memory space usage is that the CPU is used to compute the hash function. In this study, a hash function is performed multiple times for each network traffic, so there is a computational burden on the switch's CPU when it is executed.

On the other hand, in this study, RDMA needs to be able to transmit on the P4 switch of the Ethernet interface, so RoCE technology will be used to achieve the purpose of the RDMA transmission on the Ethernet interface. RoCE uses the same zero replication data transfer mode as RDMA, so it can still maintain the low CPU utilization characteristic as native RDMA [15]. However, because it is still not a native RDMA transmission, the performance of bandwidth and delay of transmission will still be different. Comparing the differences between RoCE, native RDMA, and non-RDMA transmission (Table I) [16], the measurement results show that although RoCE has a decrease in efficiency compared with native RDMA, there is still 5.7 times faster in delay reduction and 3.8 times larger in bandwidth when compared with non-RDMA transmission. P4 is a Turing-complete language [17], meaning that statements such as conditional jumps, while and goto cannot be executed, loops cannot be used directly, and memory allocation cannot be declared arbitrarily as in C. To enable the P4 switch to carry out RDMA transmission, the following RoCE header (Fig. 1) [3] must be made and modified the original transmission way so that the P4 switch can carry out RDMA transmission smoothly on the Ethernet interface.

Table 1. Transfer comparison of RDMA, RoCE, Software RoCE and without RDMA

	RDMA	RoCE	Software RoCE	No RDMA
Delay (microseconds)	1.96	3.7	11.6	21.09
Bandwidth (MB/s)	5481.9	2284.7	x	1136.1

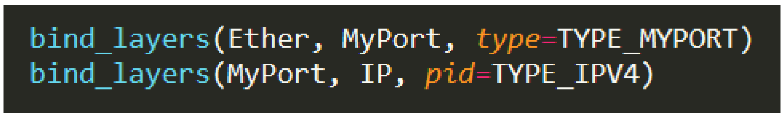
Ethernet	Global Route Header	Base Transport Header	RDMA Extended Transport Header	Payload	Invariant CRC

Figure 1. RoCE header

Chapter 2 Experimental tools and theories

1. Python Scapy

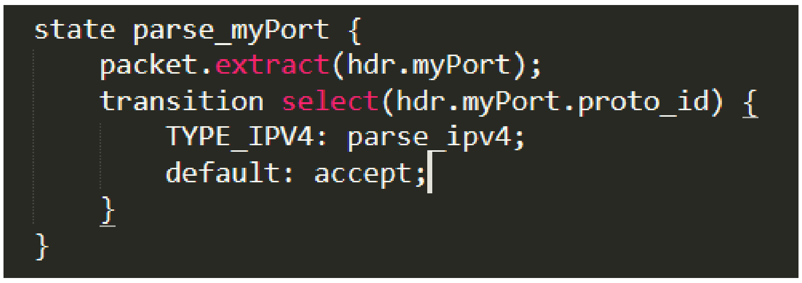
Scapy is a network packet processing tool written in Python, which allows users to edit packet content arbitrarily and send it. In addition to the basic Ethernet, IP, TCP, UDP and other headers, we can also customize the user-defined header (Figure 2). When a packet with user-defined meaning passes through the P4 switch, it can be parsed through the P4 language (Figure 3).



```
bind_layers(Ether, MyPort, type=TYPE_MYPORT)
bind_layers(MyPort, IP, pid=TYPE_IPV4)
```

Figure 2. bind_layers function

Through bind_layers [19] function in Scapy, we can add the customized MyPort header between the Ether and IP layers.



```
state parse_myPort {
    packet.extract(hdr.myPort);
    transition select(hdr.myPort.proto_id) {
        TYPE_IPV4: parse_ipv4;
        default: accept;
    }
}
```

Figure 3. extract and select functions in the parser pipeline of P4

In the parser pipeline of P4, the extract and select functions can also be implemented customized header in Scapy.

2. Mininet

Mininet [20] is a software network simulation tool running in a single Linux kernel, which can simulate the data transmission behavior of network devices such as switches, routers and client-side on a single computer. We can use it to establish a network topology environment, and use SSH and other remote online behaviors to enter the network and run the program. This research uses Mininet to establish the topology that a single switch runs in LAN, and control the behavior of the switch by Python and P4 in the switch.

3. Bmv2 (Behavior model version 2)

Bmv2 [11] is a software to simulate P4 switches. It provide a platform for running P4 language.

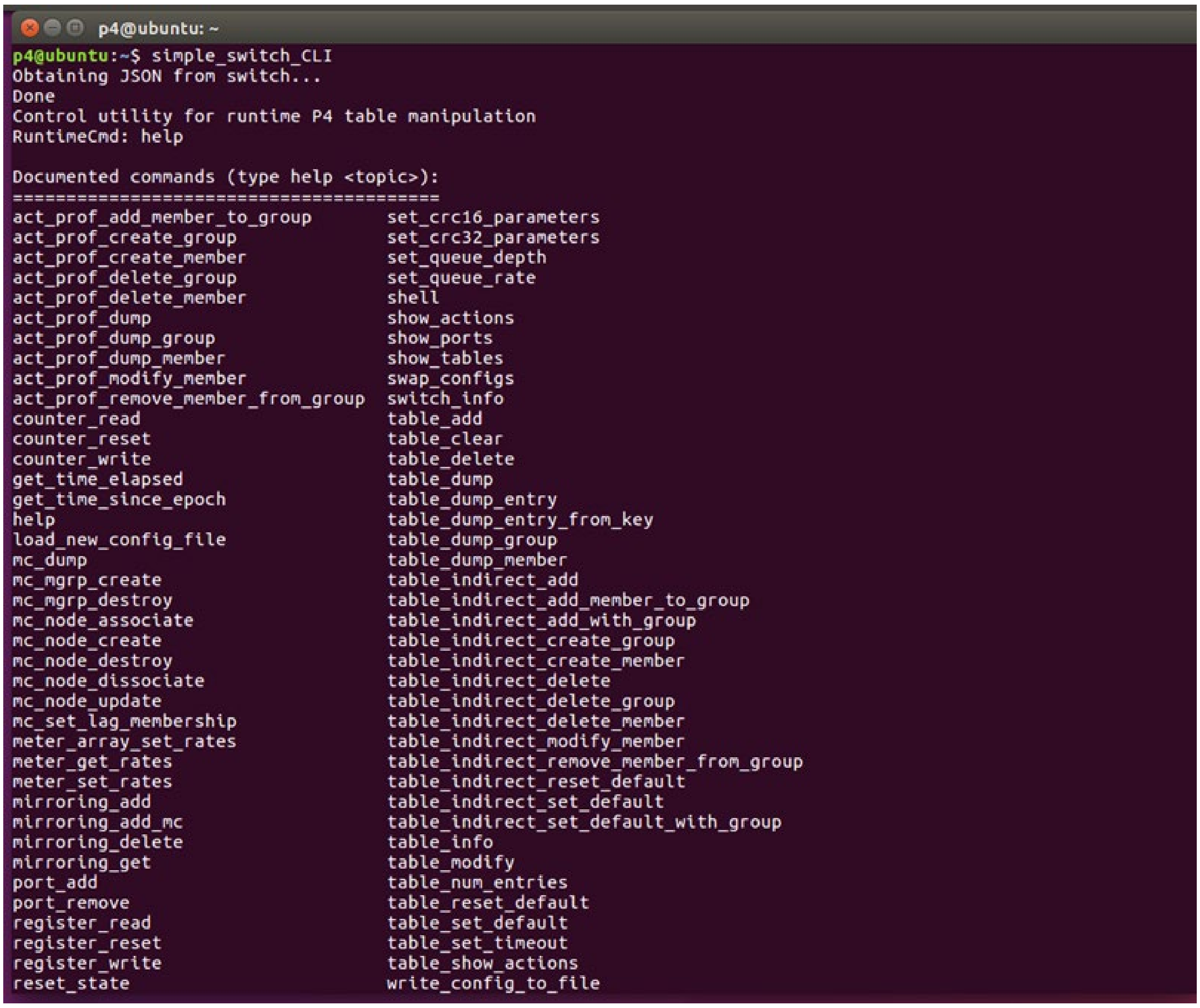
After a P4 source code complies to a JSON file, bmv2 can simulate the packet processing and forwarding behavior in the P4 switch through the JSON file.

4. P4 (Programming Protocol-Independent Packet Processors)

P4 [12], a new programming language for switches published in 2013, uses software to control the behavior of the switch. It introduced the idea of processing packets by partition and segmentation in the way of pipeline, including packet analysis, disassembly, reorganization and forwarding. The advantages of P4 are target independence and protocol independence. As long as there is a corresponding P4 compiler, the P4 program can run on any network element. P4 can handle any customized protocol that is not limited by traditional protocols such as Ethernet, TCP and UDP. and it. This research uses the bmv2 to simulate the switch and run a P4 program to control its behavior.

5. P4 Runtime API

We can use P4 runtime interface to implement the routing table of dynamic control switch. Open the switch interface and enter simple_ switch_ CLI command, and then we can operate the P4 runtime console. Enter help command, and then we can see all control instructions (Figure 4).


```
p4@ubuntu: ~
p4@ubuntu:~$ simple_switch_CLI
Obtaining JSON from switch...
Done
Control utility for runtime P4 table manipulation
RuntimeCmd: help

Documented commands (type help <topic>):
========================================
act_prof_add_member_to_group      set_crc16_parameters
act_prof_create_group             set_crc32_parameters
act_prof_create_member            set_queue_depth
act_prof_delete_group             set_queue_rate
act_prof_delete_member            shell
act_prof_dump                     show_actions
act_prof_dump_group               show_ports
act_prof_dump_member              show_tables
act_prof_modify_member            swap_configs
act_prof_remove_member_from_group switch_info
counter_read                      table_add
counter_reset                     table_clear
counter_write                     table_delete
get_time_elapsed                  table_dump
get_time_since_epoch              table_dump_entry
help                              table_dump_entry_from_key
load_new_config_file              table_dump_group
mc_dump                           table_dump_member
mc_mgrp_create                    table_indirect_add
mc_mgrp_destroy                   table_indirect_add_member_to_group
mc_node_associate                 table_indirect_add_with_group
mc_node_create                    table_indirect_create_group
mc_node_destroy                   table_indirect_create_member
mc_node_dissociate                table_indirect_delete
mc_node_update                    table_indirect_delete_group
mc_set_lag_membership             table_indirect_delete_member
meter_array_set_rates             table_indirect_modify_member
meter_get_rates                   table_indirect_remove_member_from_group
meter_set_rates                   table_indirect_reset_default
mirroring_add                     table_indirect_set_default
mirroring_add_mc                  table_indirect_set_default_with_group
mirroring_delete                  table_info
mirroring_get                     table_modify
port_add                          table_num_entries
port_remove                       table_reset_default
register_read                     table_set_default
register_reset                    table_set_timeout
register_write                    table_show_actions
reset_state                       write_config_to_file
```

Figure 4. P4 runtime command set

Use table_ add、table_ delete commands can add or delete entries in the routing table. Use table_ dump command can see the current routing table of the switch (Figure 5).

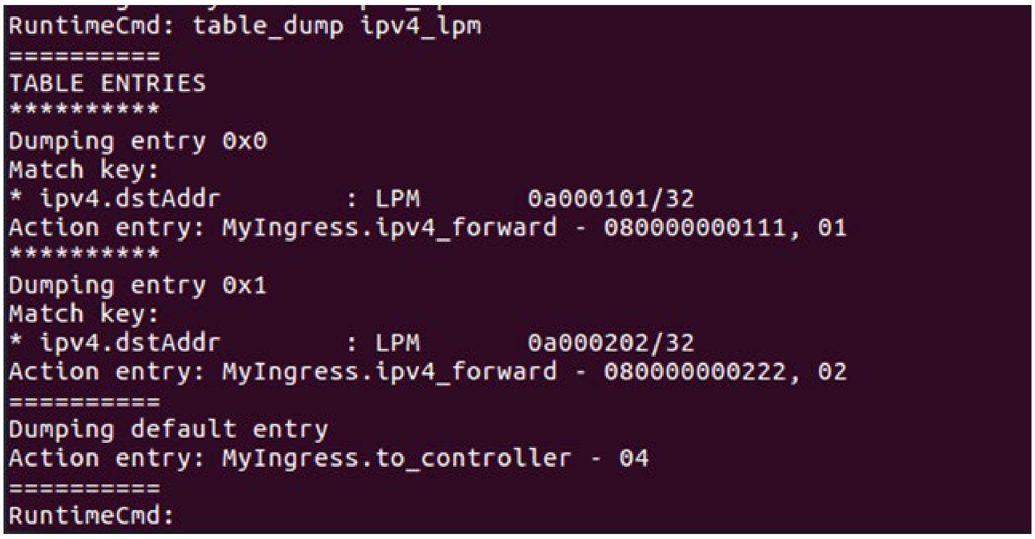

```
RuntimeCmd: table_dump ipv4_lpm
==========
TABLE ENTRIES
**********
Dumping entry 0x0
Match key:
* ipv4.dstAddr        : LPM       0a000101/32
Action entry: MyIngress.ipv4_forward - 080000000111, 01
**********
Dumping entry 0x1
Match key:
* ipv4.dstAddr        : LPM       0a000202/32
Action entry: MyIngress.ipv4_forward - 080000000222, 02
==========
Dumping default entry
Action entry: MyIngress.to_controller - 04
==========
RuntimeCmd:
```

Figure 5. the routing table in the P4 switch

Take the above figure as an example. In entry 0, the destination address IP 0a000101 can be disassembled as 0a.00.01.01, replaced by decimal is 10.0.1.1, and its corresponding port is 1. Similarly, the port corresponding to 10.0.2.2 is 2. When none of the previous entries match, the switch will select default entry. In the figure above, that is port 4. Then, the packet will be sent from port 4. In this research, we use the script cooperating with the P4 runtime to automate the process of dynamically increasing and decreasing the routing table. To reduce the number of remote requests for routing information, we design some rules. The more frequent entries will be automatically added to the routing table of the switch, and the less frequent entries will be eliminated.

6. RDMA Protocol

For traditional network card, we need to repack the data and send it out by the physical layer for the purpose of getting the data sent from the memory. However, the network card needs to tell the CPU the data transmitting request firstly. Secondly, the CPU will copy the data to the memory buffer in its internal register. Finally, the CPU will copy it to the storage space of the I / O device and send it out. If the amount of data is too large, the CPU will be busy moving data and cannot engage in computing. In this background, DMA mechanism emerges.

DMA, direct memory access, is a special mechanism for reading and writing memory (Figure 6). When the network card wants to copy data, most of the process will be completed by the DMA controller. The DMA controller will copy the memory data to the register of the DMA controller firstly. Then, the controller will copy it to the register of the I / O device and send it out. In all the process, the CPU only needs to communicate with the DMA controller and does not need to participate in the data replication. Therefore, other work can be done at this time to improve the overall efficiency.

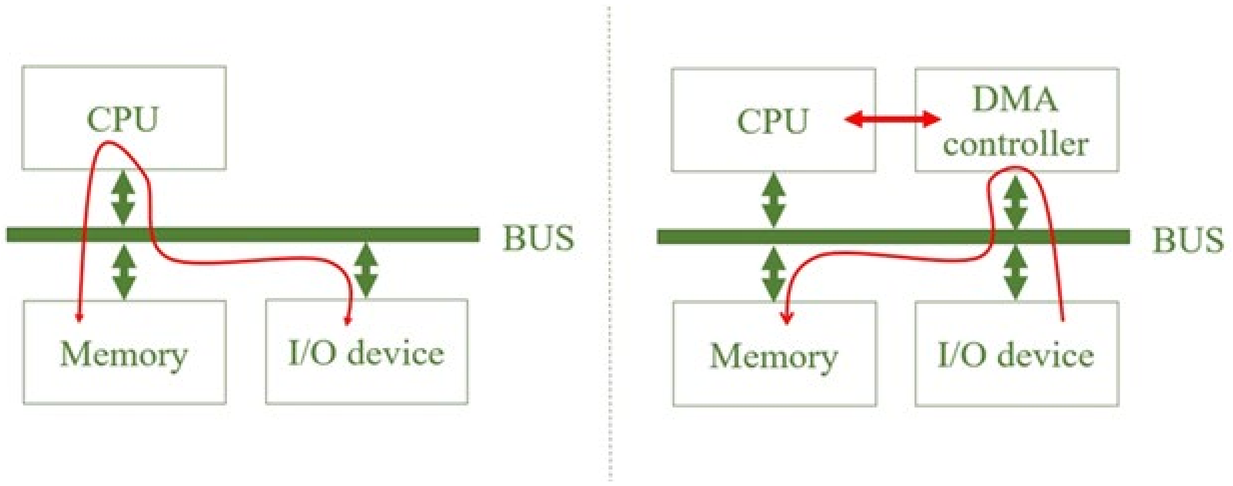


Figure 6. the transmission process of traditional network card on the left half figure
the transmission process of DMA mechanism on the right half figure

In the traditional network, we send data from node A to node B. Both sender a and receiver B need CPU control. The sender needs to copy the data from user space to kernel space through CPU before the network card access it. After the network card receives the data, it will copy the data to kernel space through DMA firstly. Then, it will copy the data from kernel space to user space through CPU. The CPU needs to complete data replication between kernel space and user space, network card control, data encapsulation and analysis, etc. The data transmission of reading and writing the remote memory is completed through the TCP / IP protocol. However, even with the DMA mechanism, the CPU load is still quite large (Figure 7).

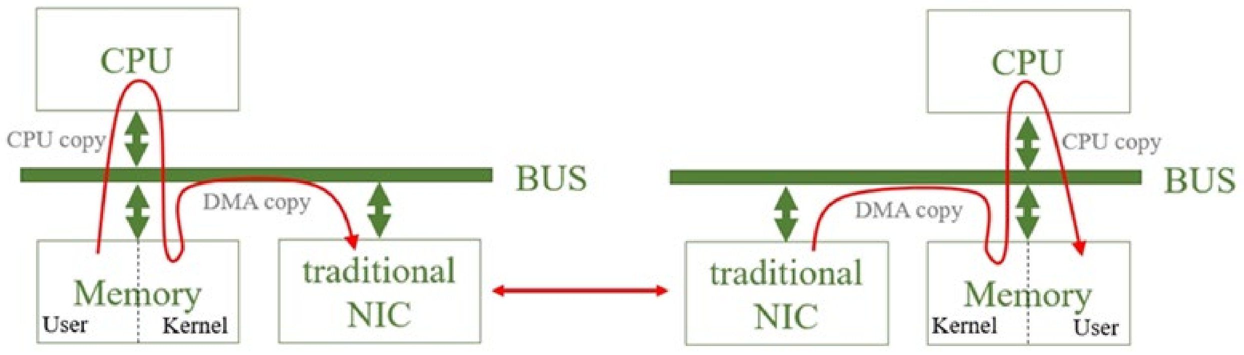


Figure 7. the transmission of DMA mechanism between two ends

RDMA, remote direct memory access, is the mechanism that further improve the remote transmission efficiency. Most of the reading and writing processes are completed through hardware. This can directly access the memory of the remote node. Through RDMA, the CPUs at both sender A and receiver B can hardly participate in data transmission. The network card at the sending A will directly copy data from the user space of the memory to the internal storage space firstly. Secondly, data will be sent to receiver B through disassembly and assembly. Finally, the CPU at receiver B will copy the data directly to the user space of the memory. Compared with traditional networks, RDMA does not need to replicate data between user space and kernel space. That is called Zero Replication mechanism, which can reduce the use of CPU in data transmission (Figure 8).

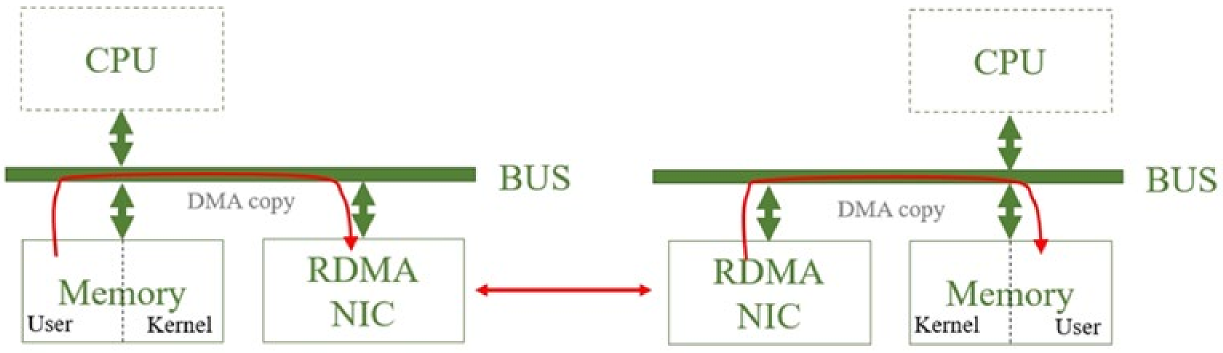


Figure 8. the transmission of RDMA

In addition to the Zero Replication mechanism, RDMA also has a kernel bypass mechanism. RDMA can bypass the kernel during data transmission, and the application layer can directly prepare data to notify the network card to send and receive. This can avoid the overhead of system call and context switching (Figure 9). If RDMA has the key to a remote area memory, it can directly continue writing to the remote end without notifying the remote CPU. This is achieved by using hardware to complete payload disassembly and analysis, which is called CPU Unloading mechanism. Under this mechanism, it can reduce the overhead of CPU analysis of payload in each layer.

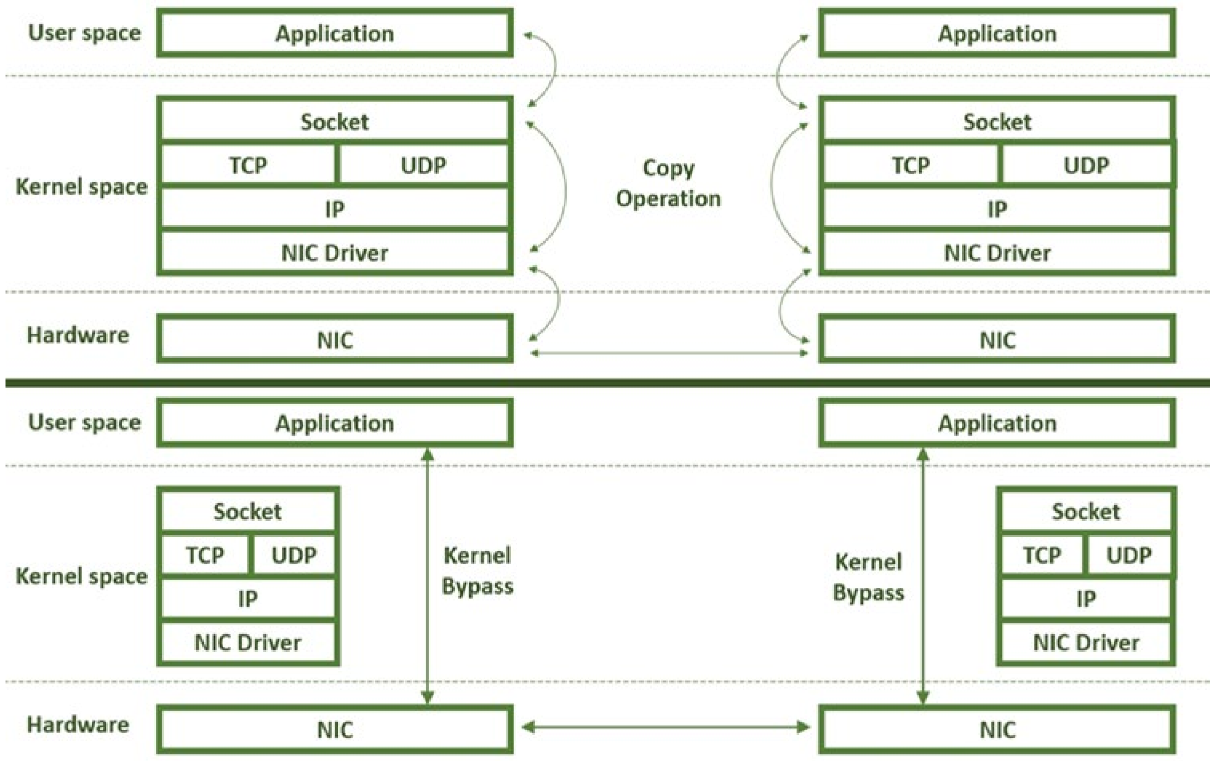


Figure 9. the transmission architecture of socket on the top half figure
the transmission architecture of RDMA, which shows kernel bypass mechanism

7. The Basic Unit and the Transmission of RDMA

Before RDMA transmits data to memory, the RDMA network card needs to register the memory region first. This means that register the operated memory as the RDMA memory region, and let the memory region be protected by RDMA. During this process, the application cannot modify the memory where the data is located, nor can it page out the memory where the data is located. This can avoid illegal access behavior and data loss when memory is paging again.

The basic unit of RDMA transmission is QP, queue pair. Each queue pair will have a unique QPN, queue pair number, to determine which queue pair is used by transmitting data. RDMA has three queue pairs to support RDMA transmission, namely send queue, receive queue, completion queue.

Send queue and receive queue are used to store sending transmission tasks and receiving transmission tasks respectively. Work queue refers to these two queues specially. Work queue is a queue used to store task requests, and WQE, work queue element, is a task request sent from software to hardware. Work queue element will record detailed information about tasks and be handed over to software to put them into work queue one by one. Then, the hardware will take it out and assigns the task. This architecture is shown in Figure 10 below. In other words, send queue and receive queue mentioned above are actually examples of work queue. As shown in Figure 11 below, each queue pair will be paired with a queue pair at the remote end for transmission. At the sender, the software will put work queue element into send queue and hand it over to hardware for execution, which is called post send. On the contrary, at the receiver, the software will put work queue element into receive queue and hand it over to hardware for execution, which is called post receive.

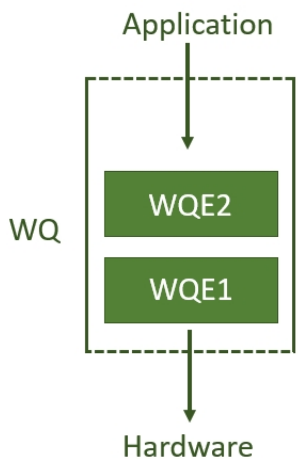


Figure 10. the architecture of work queue

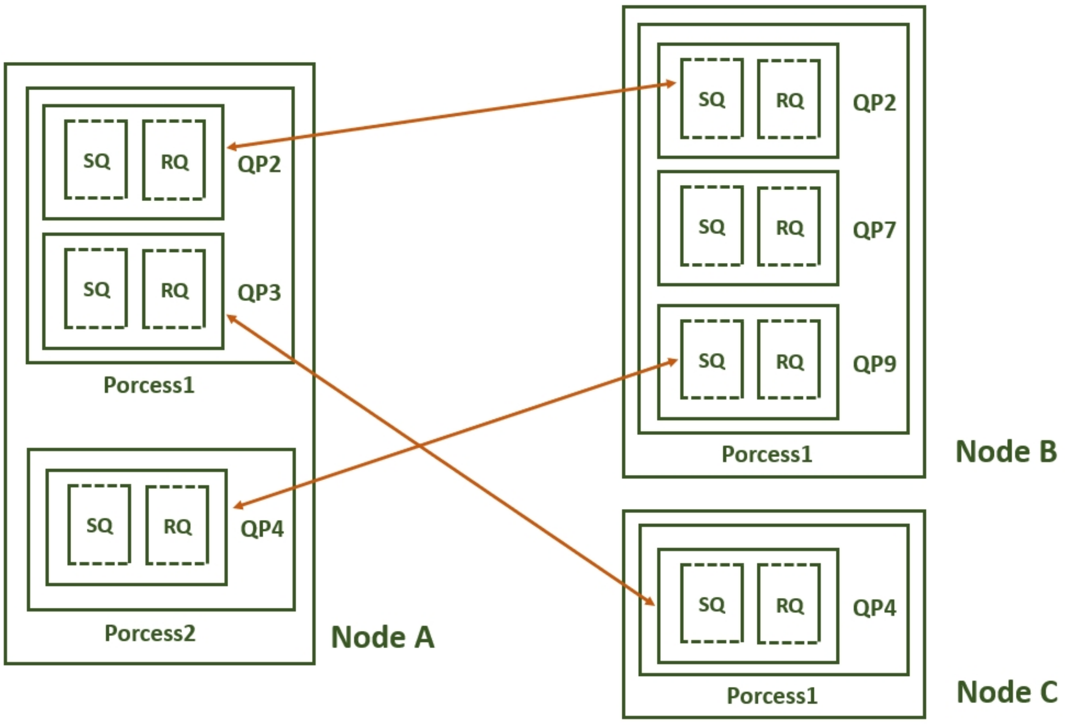


Figure 11. the pairing of queue pair

Completion queue is used to store the queue for task completion. When the hardware is completed, it will return CQE, completion queue element, to the software. Completion queue element will describe the completion information of work queue element, including whether the task has been correctly executed and what errors will be reported if errors are encountered. This architecture of completion queue is shown in Figure 12 below.

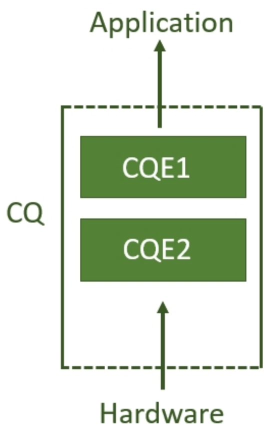


Figure 12. the architecture of completion queue

In summary, RDMA transmission operation can be divided into five steps by these queue pairs. This architecture is shown in Figure 13 below.

1. The user submits a WR, work request, to work queue through the API. Work queue element is

invisible to the user, so we need to submit work request in the application and make the driver put work queue element into work queue.

2. RDMA hardware will take out work queue element from work queue and execute tasks.
3. When the task is completed, RDMA hardware will put completion queue element into completion queue.
4. Since completion queue element is also invisible to the user, we need to take out the completion queue element in completion queue through the driver. Then, the driver will report WC, work completion, to the application to let the user know that the task is completed.
5. Repeat steps 1-4 until work queue is empty.

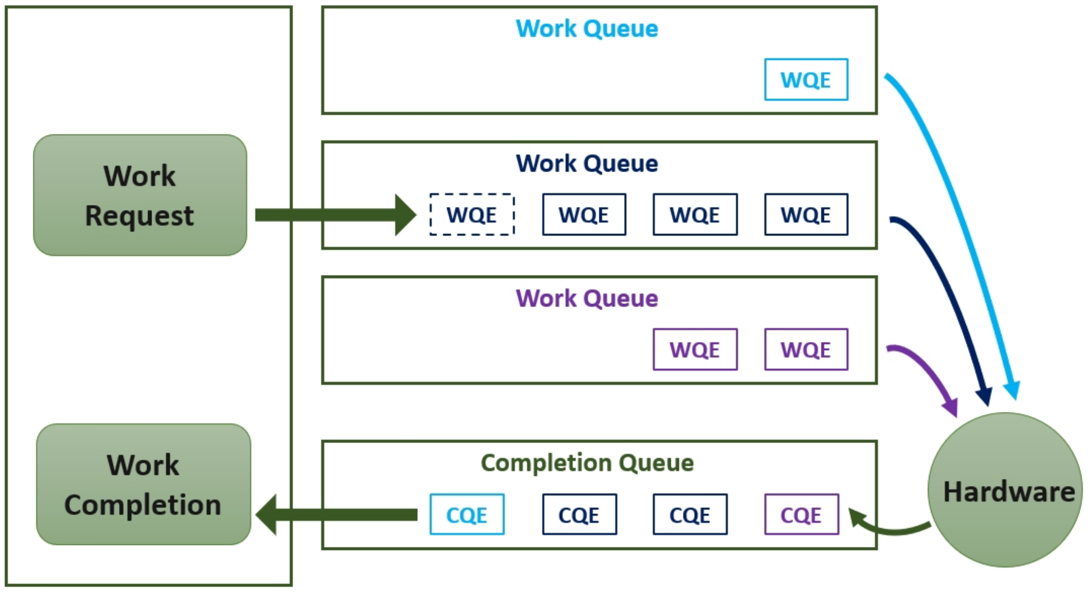


Figure 13. the detailed transmission architecture of RDMA

8. Count-Min Sketch

In this research, count min sketch can be regarded as a method to estimate the flow size. The flow size is proportional to number of packets. The larger the flow, the more packets. The memory space of the switch is limited. Count min sketch reduces a lot of storage space although its accuracy will slightly decrease. Take the following figures (figures 14, 15, 16 and 17) as an example to explain the count min sketch operation in the switch. A packet in a flow is represented by f1.

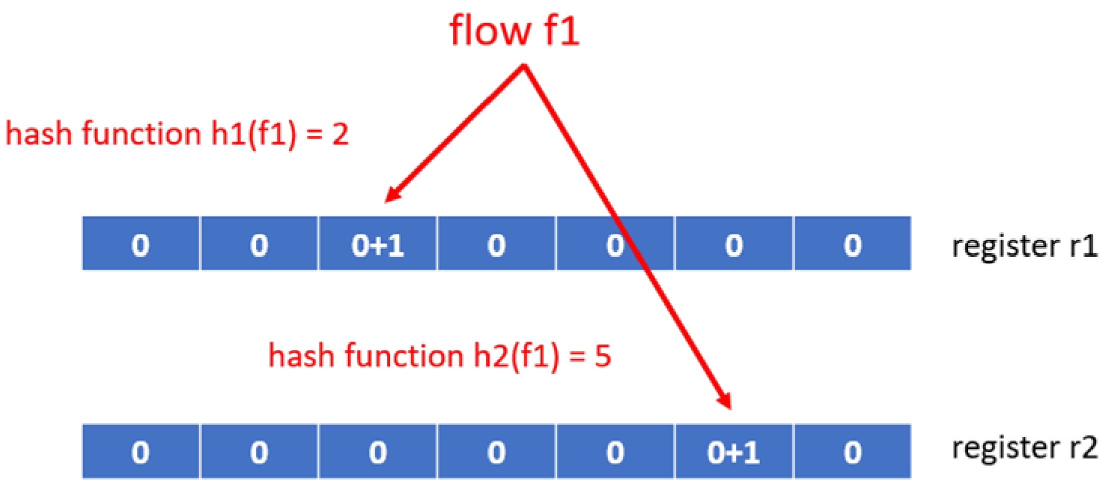


Figure 14. the first step of count-min sketch

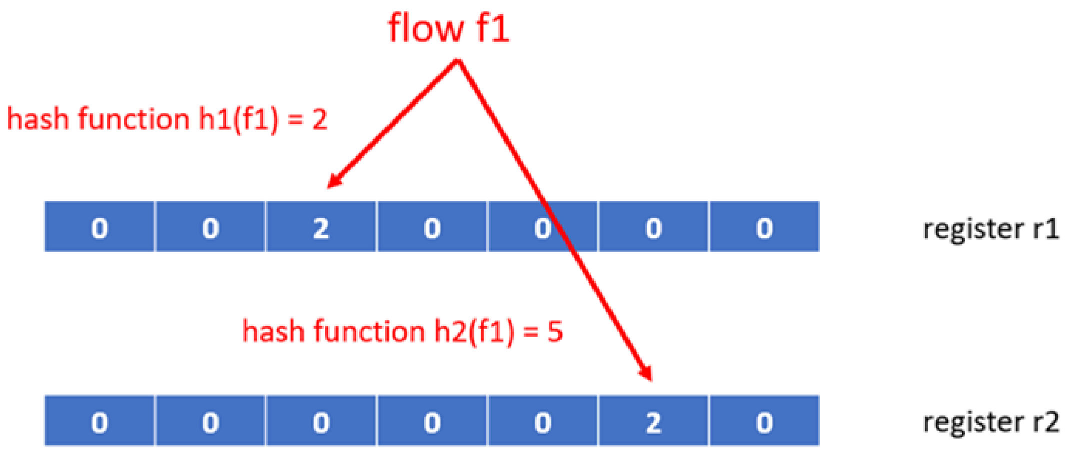


Figure 15. the second step of count-min sketch

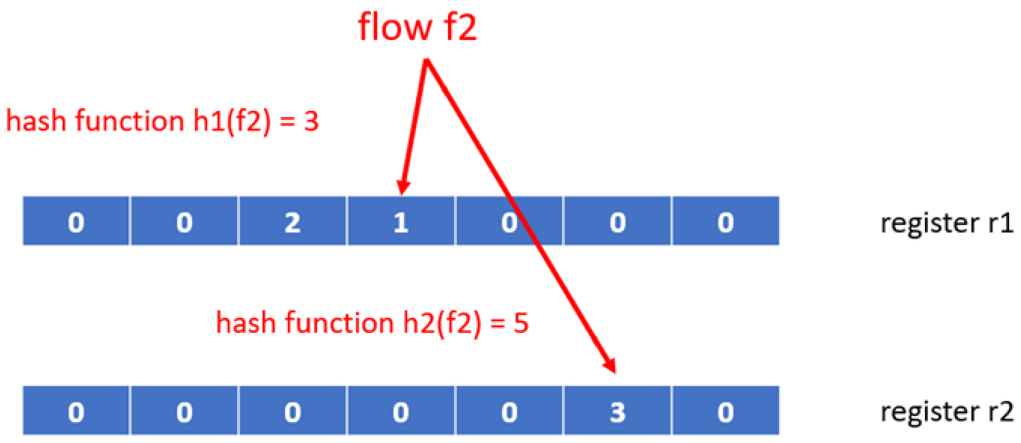


Figure 16. the third step of count-min sketch

In Figure 14 above, both registers r1 and r2 are zeroed first. Like array in other languages, register in P4 has the same function of storing data. Firstly, substitute f1 into the first hash function h1 in the form of tuple. In the flow, the tuple is composed of five elements: source IP, source port, destination IP, destination port and layer four protocol. If the result of substitution is 2, add one to the value of r1 index 2. Similarly, if the result is 5 by substituting h2, add one to the value of r2 index 5. Secondly, if f1 comes in another packet, repeat the above step again (Figure 15). Thirdly, another flow f2 is added. Similarly, substituted f2 into two hash functions to get 3 and 5 respectively, and add one to the corresponding index value (Figure 16). Finally, when we want to estimate the size of f1, we substitute f1 into two hash functions h1 and h2 and get index values 2 and 5. Then, we take the values of r1 index 2 and r2 index 5. The minimum between these two values is the result of count min sketch. In this example, we get the values are 2 and 3, so the size of f1 is estimated to be 2 (Figure 17).

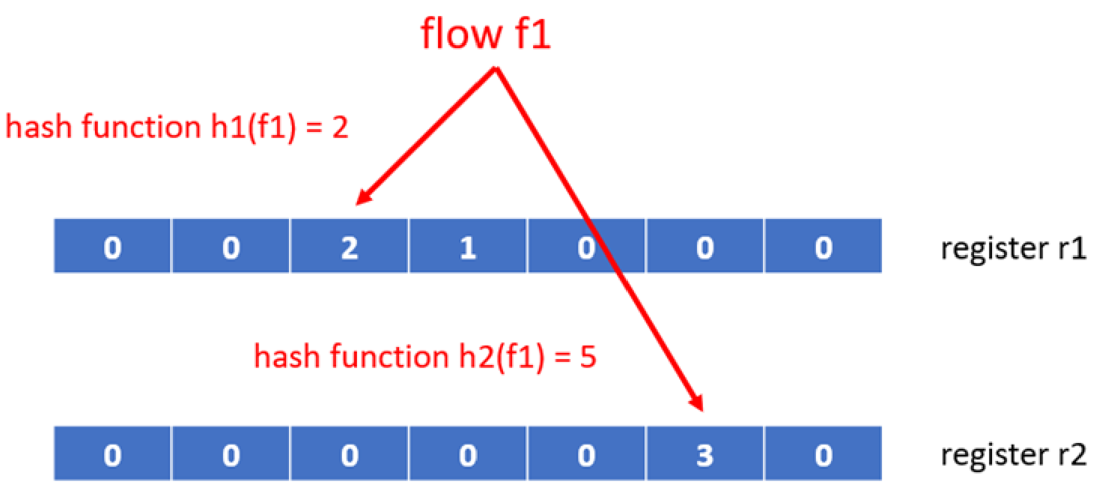
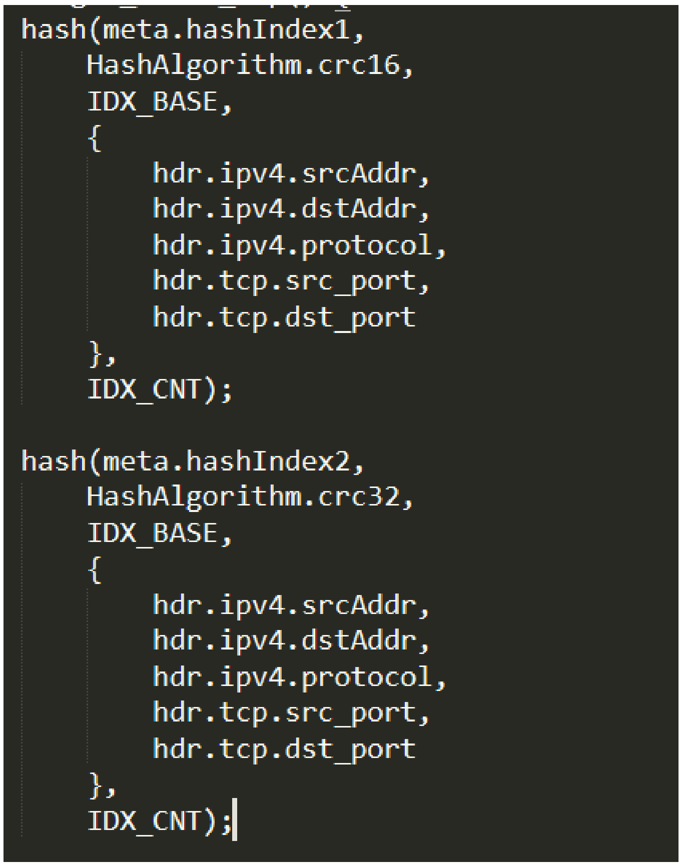


Figure 17. the final step of count-min sketch that shows count of f1= min(r1[2], r2[5])= min(2, 3)=2



```
hash(meta.hashIndex1,
    HashAlgorithm.crc16,
    IDX_BASE,
    {
        hdr.ipv4.srcAddr,
        hdr.ipv4.dstAddr,
        hdr.ipv4.protocol,
        hdr.tcp.src_port,
        hdr.tcp.dst_port
    },
    IDX_CNT);

hash(meta.hashIndex2,
    HashAlgorithm.crc32,
    IDX_BASE,
    {
        hdr.ipv4.srcAddr,
        hdr.ipv4.dstAddr,
        hdr.ipv4.protocol,
        hdr.tcp.src_port,
        hdr.tcp.dst_port
    },
    IDX_CNT);
```

Figure 18. the hash function of the P4 switch

In the figure above, the second parameter represents hash function. In count-min sketch implementation, we need to select two different hash functions. In this example, we select CRC16 as the second parameter of h1 and CRC32 as the second parameter of h2. The fourth parameter is passed into the tuple of flow, which is composed of five elements.

9. The Network Environment of System Module

The network system of this study mainly has four elements.

1. Remote Controller
2. Switch Flow Monitor
3. Switch Local Lookup
4. Remote Lookup

Firstly, the remote controller transmits control instructions to control the behavior of P4 switch. It can store all routing information for the entire LAN. Secondly, the flow monitor in the switch counts the number of packets in the flow and triggers routing table update under fixed conditions. The detailed algorithm will be described in the following P4 switch of system architecture part. Thirdly, switch local lookup will do corresponding behavior when a packet enters the switch. It will first do local lookup to check whether its own routing table has corresponding routing information. If so, it will be transferred directly. If not, perform the fourth step of remote lookup and ask the remote controller for routing information (Figure 19).

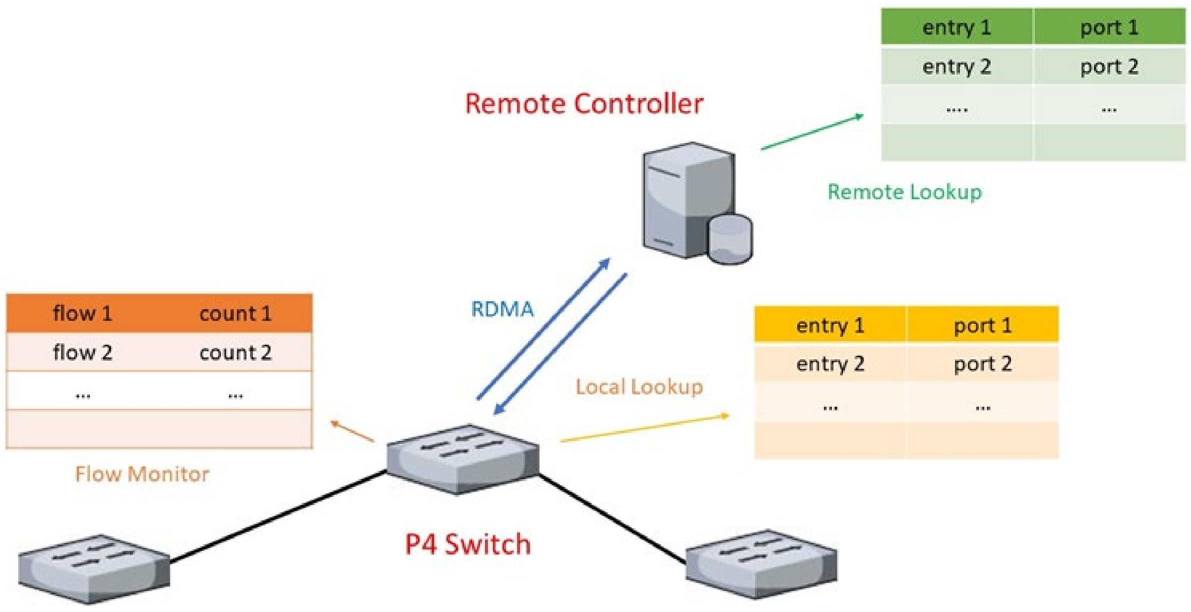


Figure 19. the overall diagram of the network system

10. RDMA of System Module

Since we need to simulate the behavior of RDMA transmission, we need to establish two virtual machines. In these two virtual machines, we use RDMA core to run Soft-RoCE, software RDMA over converged Ethernet, to simulate the transmission of RDMA in the Ethernet interface by software. To do this, we need to complete the following five steps.

1. Install two virtual machines whose operating systems are Ubuntu 20.04 LTS.
2. Install libibverbs and librdmacm in the virtual machine.
3. Download the rdma-core in linux-rdma and install the related packages.
4. Enter the command of rdma link add to start the operation of virtual RDMA network card.
5. Enter command of rxe_ cfg status to confirm whether the virtual RDMA network card is running normally.

11. P4 Switch of System Architecture

Next, we will talk about the detailed process of internal operation of P4 switch. When a packet enters the switch, the switch will first check its routing table, which is also called local lookup. When the routing information is in its own routing table, we will call it table hit. We can directly find the matching port for forwarding. Otherwise, we will call it table miss. If a table hit occurs, the flow count register will be updated before forwarding. This can record the number of hits of the flow packet and add one to the value of the corresponding register. If a table miss occurs, the count-min sketch register will be updated. The update process is showed on the above count-min sketch part.

Then, the switch will transfer the packet to the remote controller. After remote lookup, the remote controller will add the corresponding port to the packet header and return it to the switch. The switch can send out the packet according to the port in the header. In this way, the switch does not have routing information but can complete the packet forwarding without losing any packets. When a certain number of misses occur continuously, routing table will be updated, which called table update.

When table update occurs, the switch will check the count-min sketch value of the current packet (Figure 17). Then, we will compare it with the value in the flow count register in a linear manner. If the value is greater than any value in the flow count register, the forwarding rule corresponding to the register will be deleted from the routing table. Instead, we add the forwarding rule of this packet to the routing table. This make the routing rules of the larger major flow be saved into the routing table and the smaller and less used rules be deleted from routing table.

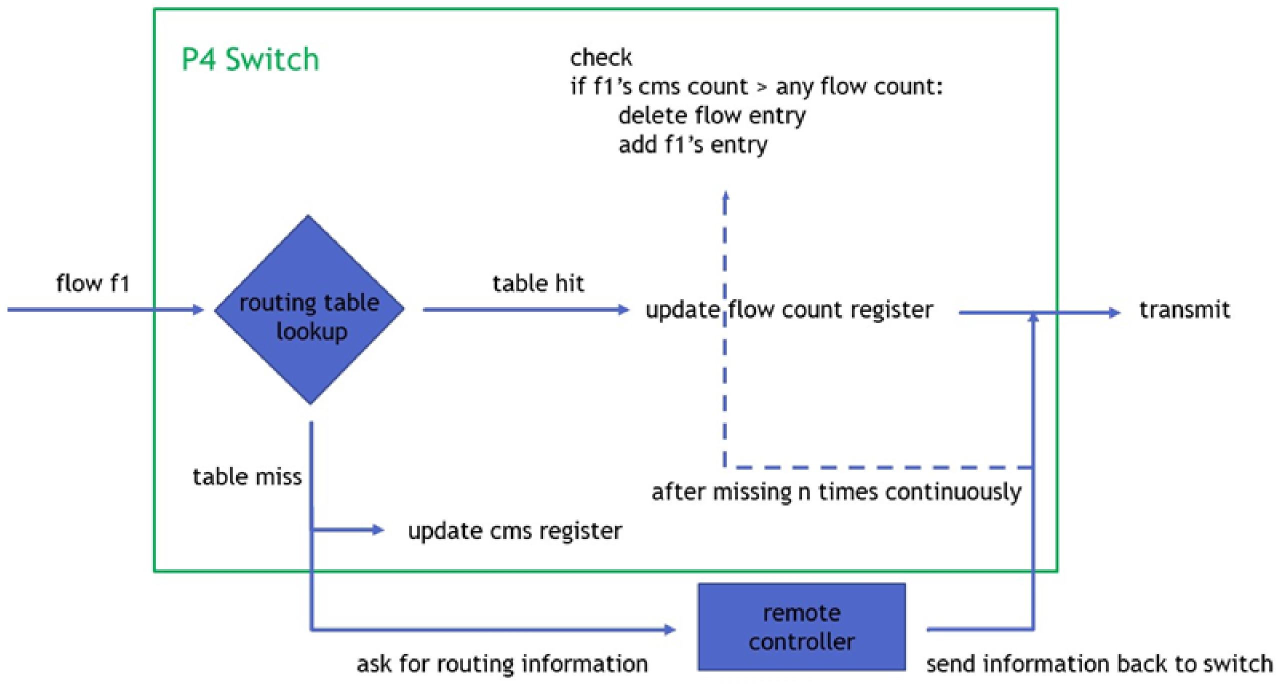

P4 Switch
check
if f1's cms count > any flow count:
delete flow entry
add f1's entry
flow f1
routing table lookup
table hit
update flow count register
transmit
table miss
after missing n times continuously
update cms register
remote controller
ask for routing information
send information back to switch

Figure 20. the flow diagram of P4 switch algorithm

In the complex network experimental setting, we will make the number of flows far exceed the capacity of the routing table of the P4 switch. This means that it will far exceed the memory capacity of the register. Obviously, it is not feasible and efficient to save all flow counts of all flows into the register. Under this condition, the spatial advantages of count-min sketch can be brought into play.

12. RDMA of System Architecture

Since RDMA requires IP location and port when connecting, the network adapter of the two virtual machines need to set bridged to obtain IP location. In the experimental process, one side acts as a server and the other as a client. Both ends use Post instructions to control the transmission process and confirm the state.

Chapter 3 Research methods and procedures

1. Simple Network Architecture

Firstly, we use bmv2 combined with mininet to simply simulate the behavior of switches in the above architecture. The first step is to establish a simple network topology (Figure 21). This architecture consists of a P4 switch s1 connecting with three hosts and a remote controller. The IP of host h1 is 10.0.1.1, h2 is 10.0.2.2, h3 is 10.0.3.3, and the controller is 10.0.4.4. There are forwarding rules for h1 and h2 in the routing table of s1, but there is no forwarding rule for h3. It represents that currently the packet cannot be transmitted to h3. h4 is the controller and does not belong to the host, so there is no path in the routing table. The controller stores forwarding rules for all hosts.

In the first step, h1 sends a packet to h2. After receiving the packet, switch S1 checks its routing table and finds that the packet destination h2 is in the routing table, and its corresponding port is 2. It means that the packet can be successfully sent from h2 by sending the packet to port 2. Therefore, the switch adds one to its corresponding flow count register value and sends the packet to port 2. The state of routing information in the routing table is called table hit.

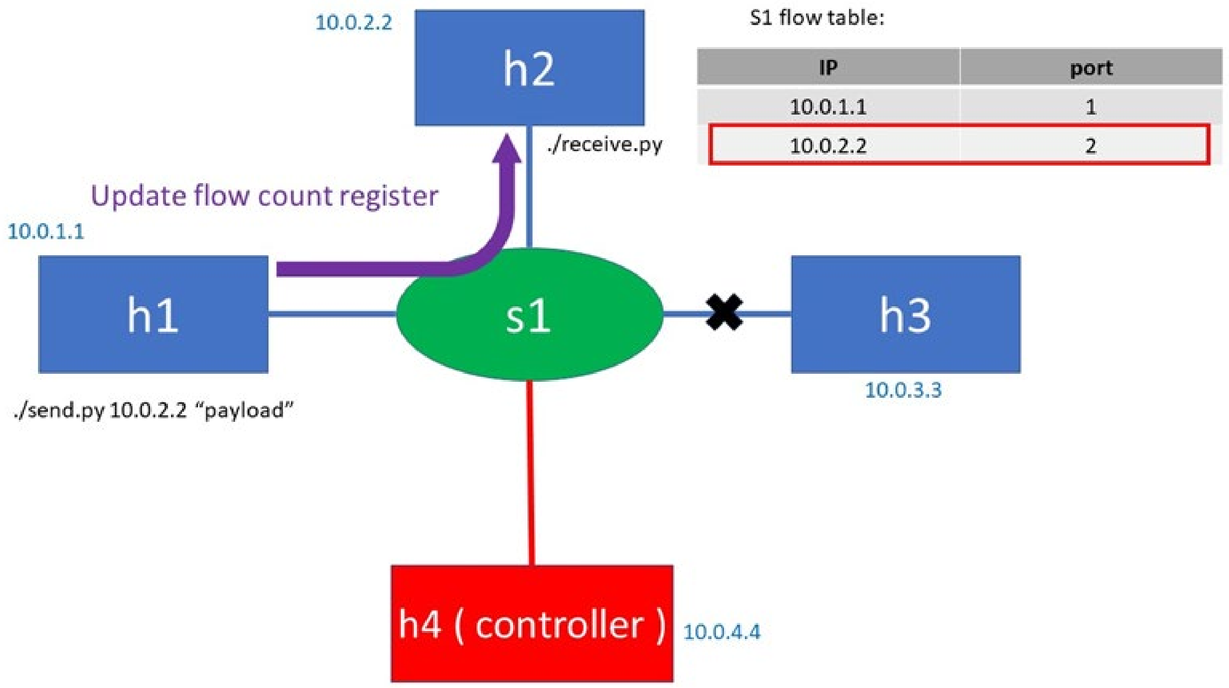
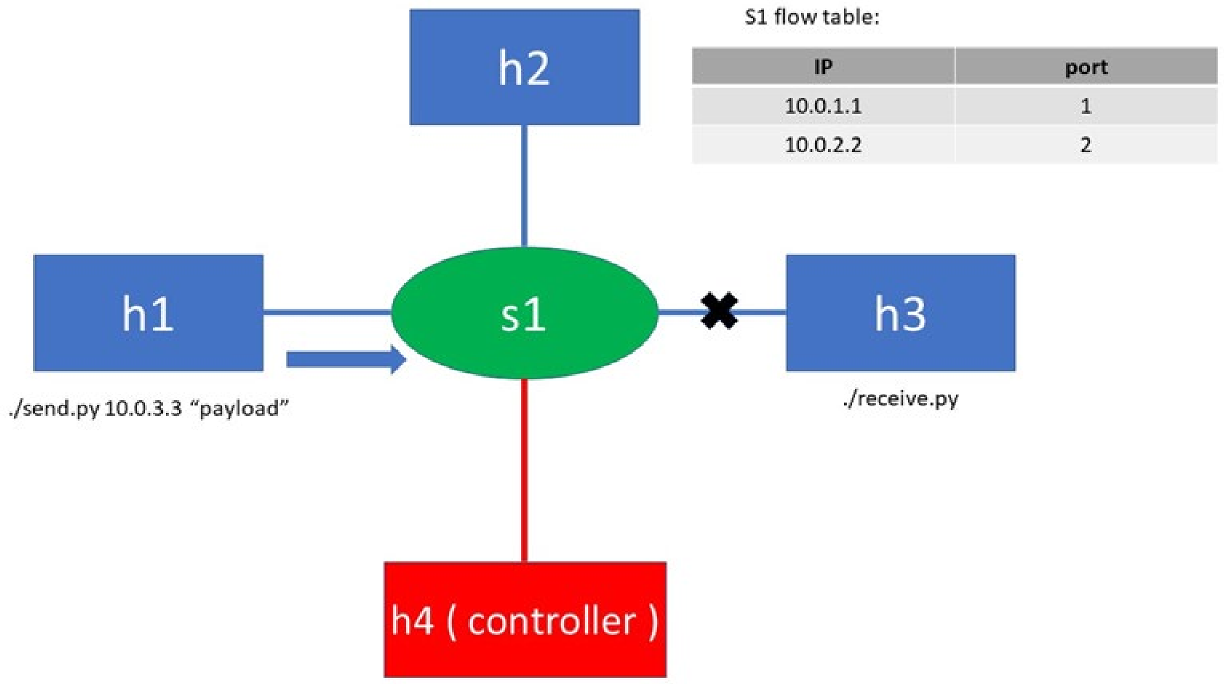


Figure 21. h1 sends a packet to h2

Next, h1 attempts to send a packet to h3. After receiving the packet, switch s1 will check its own routing table first. This time, it finds that it does not have a forwarding rule for h3, which means it does not know which port to send the packet to (Figure 22). The state in which there is no routing information in the routing table is called table miss.

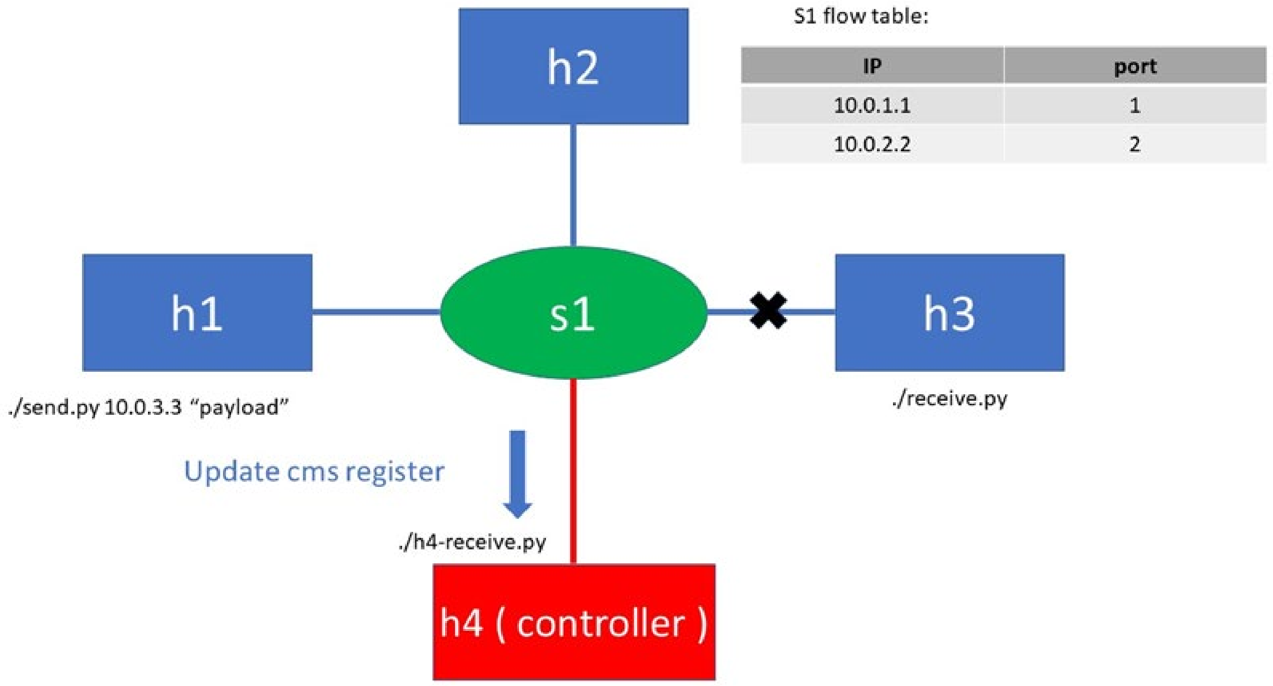


S1 flow table:

IP	port
10.0.1.1	1
10.0.2.2	2

Figure 22. h1 sends a packet to h3, which occurs table miss

At this time, s1 will ask the controller for the routing information. It sends the packet to h4 and updates the count min sketch register (Figure 23).



S1 flow table:

IP	port
10.0.1.1	1
10.0.2.2	2

Figure 23. s1 forwards a packet to controller

After receiving the packet from s1, the controller queries the routing information summary table according to its destination IP and obtains that the forwarding rule is port 3. It represents the packet of this IP must be sent from port 3 of s1. The controller modifies the packet field, packs the message into the packet (Figure 24), and sends it back to s1 in reverse direction. After s1 receives the packet, it can successfully send the packet to h3 according to this message.

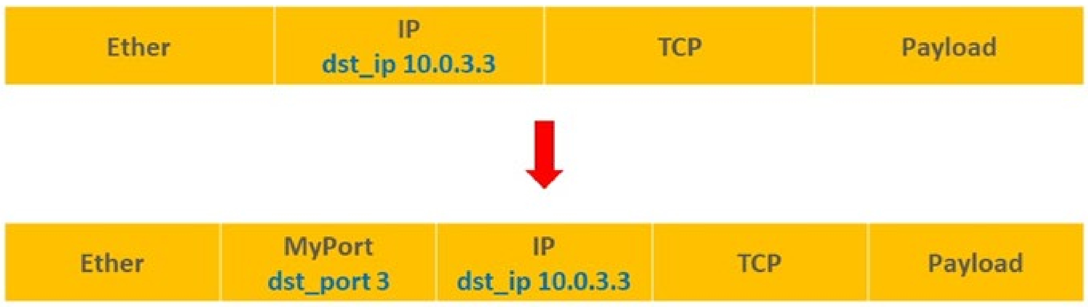


Figure 24. controller modifies the packet field and load the port information

Whenever s1 a certain number of consecutive misses occur, the routing table update is performed. It will check whether the value, hereinafter referred to as x1, obtained by the current flow, hereinafter referred to as f1, executing count min sketch is greater than the value in the flow count register. The comparison method is from front to back linear comparison. If it is found that the value in the flow count register is less than x1, delete the routing table entry corresponding to the register index, and add the forwarding rule of f1 into the routing table of s1. The register also needs to perform the same operation to maintain the correspondence between table and register. In the above example, update is required if s1 miss five times. If h1 sends five consecutive packets to h3, there will be five consecutive misses. And trigger table update at the fifth miss. It will check the values in the flow count register in sequence, assuming that it is 7 and 2, and the corresponding table entry is 10.0.1.1 and 10.0.2.2. Suppose that the value obtained by count min sketch is 4. In the first step, compare 7 > 4, so there is no action; But then compare 2 < 4, it will trigger the swap mechanism. s1 will delete the entry of 10.0.2.2 and add the entry of 10.0.3.3 to the table. The flow count register will also change to 7,0, as shown in Figure 25 below.

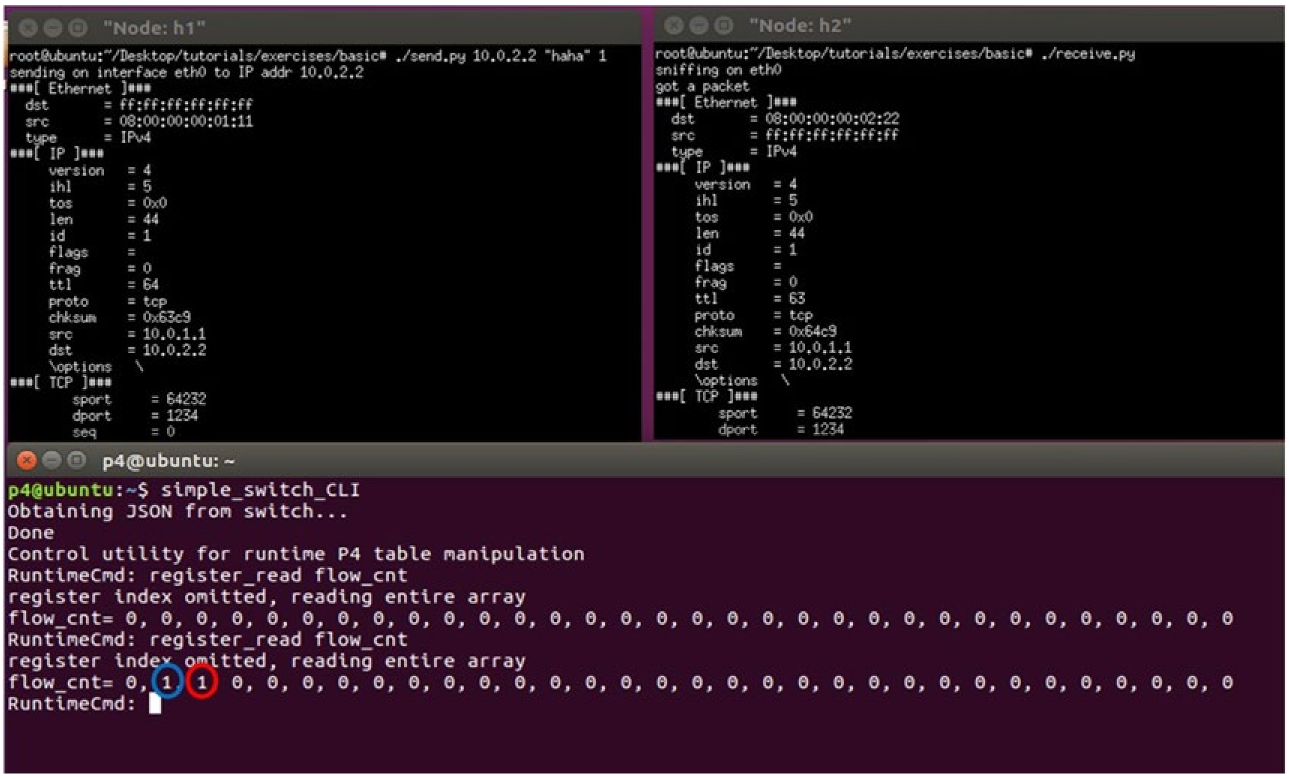
IP	port	count
10.0.1.1	1	7
10.0.2.2	2	2

$\Downarrow$

IP	port	count
10.0.1.1	1	7
10.0.3.3	3	0

Figure 25. routing table update

Finally, it is the picture of actual operation on Linux system. First, in Figure 26, h1 sends a TCP packet to h2. At this time, the switch has the routing information to h2, which can be forwarded directly, and the P4 runtime API is used to check the flow count register.



"Node: h1"

```
root@ubuntu:~/Desktop/tutorials/exercises/basic# ./send.py 10.0.2.2 "haha" 1
sending on interface eth0 to IP addr 10.0.2.2
###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 08:00:00:00:01:11
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 44
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0x63c9
     src       = 10.0.1.1
     dst       = 10.0.2.2
     \options   \
###[ TCP ]###
        sport     = 64232
        dport     = 1234
        seq       = 0
```

"Node: h2"

```
root@ubuntu:~/Desktop/tutorials/exercises/basic# ./receive.py
sniffing on eth0
got a packet
###[ Ethernet ]###
  dst       = 08:00:00:00:02:22
  src       = ff:ff:ff:ff:ff:ff
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 44
     id        = 1
     flags     =
     frag      = 0
     ttl       = 63
     proto     = tcp
     chksum    = 0x64c9
     src       = 10.0.1.1
     dst       = 10.0.2.2
     \options   \
###[ TCP ]###
        sport     = 64232
        dport     = 1234
```

p4@ubuntu: ~

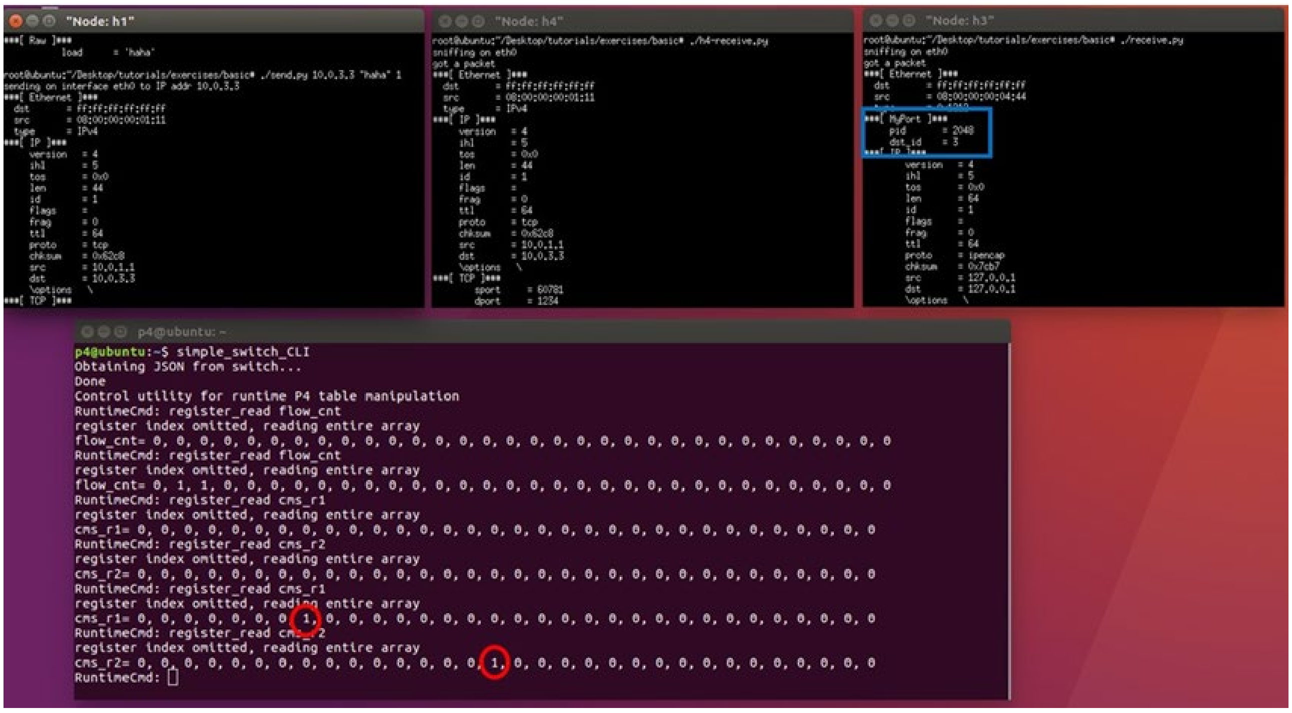
```
p4@ubuntu:~$ simple_switch_CLI
Obtaining JSON from switch...
Done
Control utility for runtime P4 table manipulation
RuntimeCmd: register_read flow_cnt
register index omitted, reading entire array
flow_cnt= 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
RuntimeCmd: register_read flow_cnt
register index omitted, reading entire array
flow_cnt= 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
RuntimeCmd:
```

Figure 26. demonstration screen (1)

In the above figure, h1 is in the upper left corner, and send.py is executed on h1 to send a packet to h2. h2 is in the upper right corner, executing receive.py to receive the packet. The following is the

P4 runtime API, using register_ Read view flow_ CNT content. The red circle indicates that h1 sends a TCP packet to h2, while the blue circle indicates that h2 sends an ACK packet back to h1.

Next, in Figure 27, h1 sends a packet to h3. At this time, the switch has no routing information to h3 and needs to transfer it to h4 acting as the controller. We check the two count min sketch registers r1 and r2 in the same way.



```
"Node: h1"
###[ Raw ]###
        load      = 'haha'

root@ubuntu:~/Desktop/tutorials/exercises/basic# ./send.py 10.0.3.3 "haha" 1
sending on interface eth0 to IP addr 10.0.3.3
###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 08:00:00:00:01:11
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 44
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0x62c8
     src       = 10.0.1.1
     dst       = 10.0.3.3
     \options   \
###[ TCP ]###
```

```
"Node: h4"
root@ubuntu:~/Desktop/tutorials/exercises/basic# ./h4-receive.py
sniffing on eth0
got a packet
###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 08:00:00:00:01:11
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 44
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0x62c8
     src       = 10.0.1.1
     dst       = 10.0.3.3
     \options   \
###[ TCP ]###
        sport     = 60781
        dport     = 1234
```

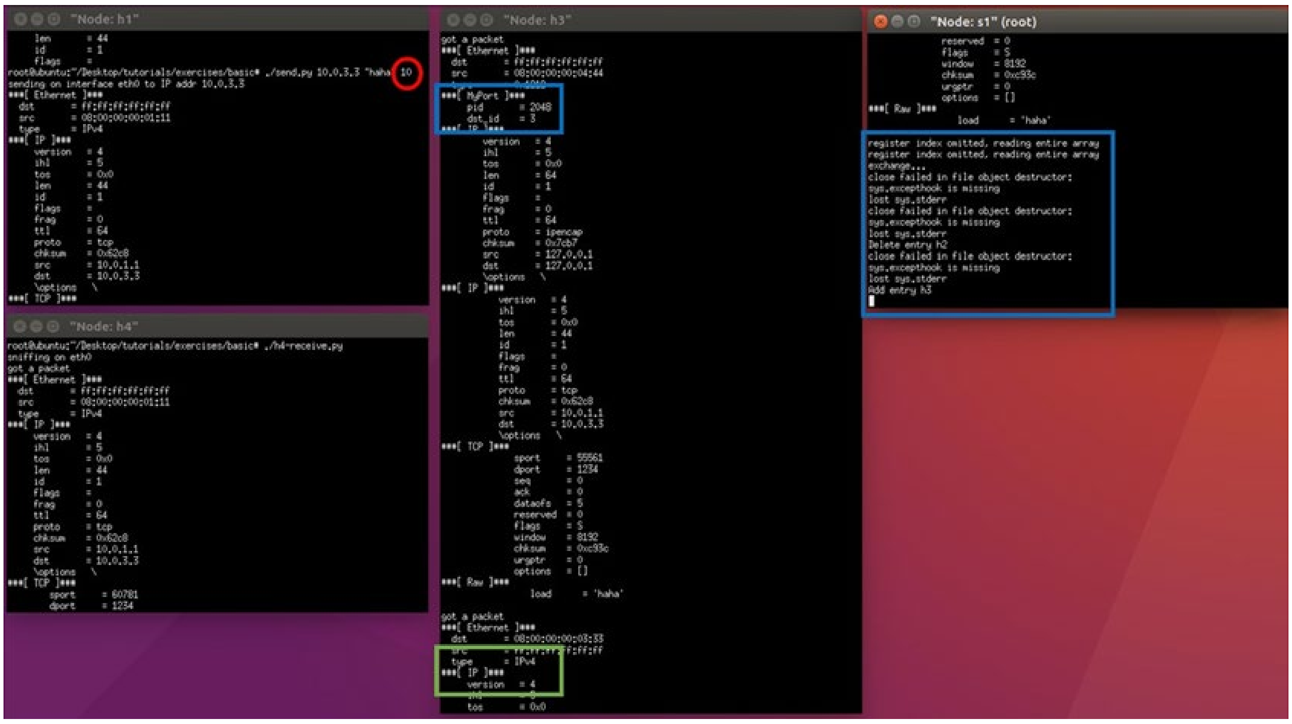
```
"Node: h3"
root@ubuntu:~/Desktop/tutorials/exercises/basic# ./receive.py
sniffing on eth0
got a packet
###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 08:00:00:00:04:44
  type      = [illegible]
###[ MyPort ]###
     pid       = 2048
     dst_id    = 3
###[ IP ]###
        version   = 4
        ihl       = 5
        tos       = 0x0
        len       = 64
        id        = 1
        flags     =
        frag      = 0
        ttl       = 64
        proto     = ipencap
        chksum    = 0x7cb7
        src       = 127.0.0.1
        dst       = 127.0.0.1
        \options   \
```

```
p4@ubuntu: ~
p4@ubuntu:~$ simple_switch_CLI
Obtaining JSON from switch...
Done
Control utility for runtime P4 table manipulation
RuntimeCmd: register_read flow_cnt
register index omitted, reading entire array
flow_cnt= 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
RuntimeCmd: register_read flow_cnt
register index omitted, reading entire array
flow_cnt= 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
RuntimeCmd: register_read cms_r1
register index omitted, reading entire array
cms_r1= 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
RuntimeCmd: register_read cms_r2
register index omitted, reading entire array
cms_r2= 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
RuntimeCmd: register_read cms_r1
register index omitted, reading entire array
cms_r1= 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
RuntimeCmd: register_read cms_r2
register index omitted, reading entire array
cms_r2= 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
RuntimeCmd: 
```

Figure 27. demonstration screen (2)

In the above figure, h1 is on the left, h4 is in the middle and h3 is on the right of the upper half. It can see that h1 sends a packet to h3 and forwards it through h4. The packet received by h3 will also have a MyPort field (blue box). cms_r1 and cm1_ r2 in below has a value that changes from 0 to 1 (red circle).

Finally, figure 28 shows that h1 sends 10 consecutive packets to h3. The switch still has no routing information to h3, so the first five packets will have five consecutive misses. At this time, execute table update to install the routing information of h3 into the routing table.



```
"Node: h1"
     len       = 44
     id        = 1
     flags     =
root@ubuntu:~/Desktop/tutorials/exercises/basic# ./send.py 10.0.3.3 "haha" 10
sending on interface eth0 to IP addr 10.0.3.3
###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 08:00:00:00:01:11
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 44
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0x62c8
     src       = 10.0.1.1
     dst       = 10.0.3.3
     \options   \
###[ TCP ]###
```

```
"Node: h4"
root@ubuntu:~/Desktop/tutorials/exercises/basic# ./h4-receive.py
sniffing on eth0
got a packet
###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 08:00:00:00:01:11
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 44
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0x62c8
     src       = 10.0.1.1
     dst       = 10.0.3.3
     \options   \
###[ TCP ]###
        sport     = 60781
        dport     = 1234
```

```
"Node: h3"
got a packet
###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 08:00:00:00:04:44
  type      = 0x1010
###[ MyPort ]###
     pid       = 2048
     dst_id    = 3
###[ IP ]###
        version   = 4
        ihl       = 5
        tos       = 0x0
        len       = 64
        id        = 1
        flags     =
        frag      = 0
        ttl       = 64
        proto     = ipencap
        chksum    = 0x7cb7
        src       = 127.0.0.1
        dst       = 127.0.0.1
        \options   \
###[ IP ]###
           version   = 4
           ihl       = 5
           tos       = 0x0
           len       = 44
           id        = 1
           flags     =
           frag      = 0
           ttl       = 64
           proto     = tcp
           chksum    = 0x62c8
           src       = 10.0.1.1
           dst       = 10.0.3.3
           \options   \
###[ TCP ]###
              sport     = 55561
              dport     = 1234
              seq       = 0
              ack       = 0
              dataofs   = 5
              reserved  = 0
              flags     = S
              window    = 8192
              chksum    = 0xc93c
              urgptr    = 0
              options   = []
###[ Raw ]###
                 load      = 'haha'

got a packet
###[ Ethernet ]###
  dst       = 08:00:00:00:03:33
  src       = ff:ff:ff:ff:ff:ff
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
```

```
"Node: s1" (root)
                 reserved  = 0
                 flags     = S
                 window    = 8192
                 chksum    = 0xc93c
                 urgptr    = 0
                 options   = []
###[ Raw ]###
                    load      = 'haha'

register index omitted, reading entire array
register index omitted, reading entire array
exchange...
close failed in file object destructor:
sys.excepthook is missing
lost sys.stderr
close failed in file object destructor:
sys.excepthook is missing
lost sys.stderr
Delete entry h2
close failed in file object destructor:
sys.excepthook is missing
lost sys.stderr
Add entry h3
```

Figure 28. demonstration screen (3)

In the figure above, h1 continuously transmits 10 packets to h3. In the middle is h3. The packets received in the front have the MyPort field (blue frame in the middle), while the packets received later have no (cyan frame). On the right is switch s1. You can see that a table update message occurs in s1 (blue box on the right), h3 routing information is installed and h2 routing information is deleted.

2. Complex Network Architecture

Then, bmv2 and mininet are still used to simulate a complex network architecture. Suppose there is a LAN with host IP of 10.0.0.1 ~ 10.0.0.254, 10.0.1.1 ~ 10.0.1.254, 10.0.2.1 ~ 10.0.2.254, 10.0.3.1 ~ 10.0.3.254, total of 1016 hosts. A P4 switch processes 1000 flows in total, and only one packet in one flow is processed per unit time. In Kaustubh Gadkari's research [10], it is pointed out that in ISP (Internet service provider) networks, 1.93% of flow entries will be used 99.5%. It means that about 2% of the flows are major flows, so 20 flows are randomly made to be major flows. Major flow has 500 ~ 1000 packets in total, with a duration of 500 ~ 1000 unit time; Normal flow is 1 ~ 50 packets with a duration of 1 ~ 50 unit time. All packets transmitted are UDP, so there will be no ack. Overall, the number of packets to be processed by this switch is about $20 \times 750+980 \times 25.5 = 39990$, about 40000 packets. Assuming that the unit time is 1 second, that is, only one packet is transmitted per second (collision avoidance), it will take about 40000 seconds for all packets to be transmitted.

Due to the limitation of computer performance, we can't implement the LAN of thousands of hosts with mininet. However, what we care about is the number of routing table hit and miss of the switch. Therefore, it can be assumed that the packet will not be lost due to other external factors. As long as the switch selects the right port to send the packet, it can be preset that the packet will be transmitted to the host. In other words, we only need to focus on the behavior of the switch, not the transmission of packets in the network. For this purpose, we connect a packet generator to a P4 switch. For the packet generator, we can run Python on it and use scapy to form packets and transmit them. The advantage of using scapy is that it can help us freely generate packets of any form and content. We can freely decide the source IP and destination IP. Based on this, we can simulate different flows and different packet incoming switches, as shown in Figure 29 below.

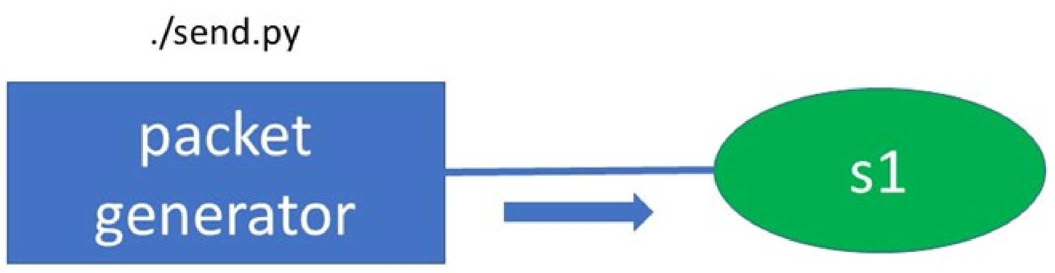


Figure 29. complex network architecture

3. The Experimental Process of RDMA

In the actual network, it takes some time for SDN switch to communicate with remote controller, which reduces the timeliness of routing information and network telemetry. Therefore, based on the above complex network, we use RDMA to improve the communication time between SDN switch and remote controller to ensure the timeliness of information. In this experiment, firstly, the RDMA transmission needs to be established in advance, so that the search requirements can be sent in time. Next, the pre search IP needs to be put into the post of RDMA transmission, so that the remote controller can know the route that the local switch wants to find. During the operation of the whole search architecture, due to the rigorous verification mechanism of RDMA, we need to ensure that the search can be carried out continuously without disconnection. Finally, we need to ensure that the routing information can be placed in the specific memory location of each switch when the controller serves multiple local switches to prevent the local switch from reading the route information belonging to other switches when reading the route.

Finally, in order to verify whether using RDMA can indeed reduce the CPU occupation and search delay of local switches when searching routes compared with traditional transmission methods, we need to compare the transmission delay of local switches under different CPU loads and remote controllers under serving different numbers of local switches, and analyze the results.

4. The Experimental Steps of RDMA

In order to achieve the RDMA transmission of this architecture, we need to complete the following five steps.

1. We need to establish connecting first. This can avoid that the connecting time influence immediacy during the lookup.
2. Take the local SDN switch handing over the remote controller and remote controller performing lookup as the trigger event. We send the post containing the lookup IP location and retrieve the routing rule by RDMA read.
3. Continuously perform transmission between local SDN switch and remote controller.
4. The remote controller's memory that will be accessed between local SDN switches cannot be the same. Thus, remote controller can determine the storage location of the routing rules that each SDN switch controls.
5. Taking the number of switches and CPU load of switches as variable factors. We perform transmission and compare the results.

Next, we will talk about the above five steps in detail. The First point is establishing connecting in advance. When running this architecture, you need to establish connecting first to avoid establishing connecting time when searching. This will affect the immediacy of finding routes. RDMA transmission is basically divided into five steps.

1. Establish an event channel. This opens the connecting between two machines.
2. Both the local SDN switch and the remote controller allocate memory first. Then, register it in the memory region.
3. In the Post, fill in transmission information, memory information and key of its memory region.
4. After sending the Post containing the transmission information and verifying the content, the transmission will begin.
5. Confirm the transmission status and disconnect.

In order to achieve the goal that the remote controller can directly enter the transmission stage during the lookup, the remote controller will receive connecting request from the local SDN switch first. Then, both ends will complete the above parts 1 to 3 so that the Post containing the request can be sent directly during the lookup. Post has the information of routing rules can be read directly.

The second point is the lookup routing. During connecting of RDMA, Post plays the role of enabling both ends to verify the information of transmission. Since connecting and information exchange have been completed previously, the remote controller is waiting for the Post sent by the local SDN switch now. When the local SDN switch needs to lookup a route from the remote controller, it will write a Post containing the lookup IP location and RDMA read request and send it. The remote controller will let the local SDN switch read its corresponding memory after verifying the Post content, such as key, connecting status and other information correct.

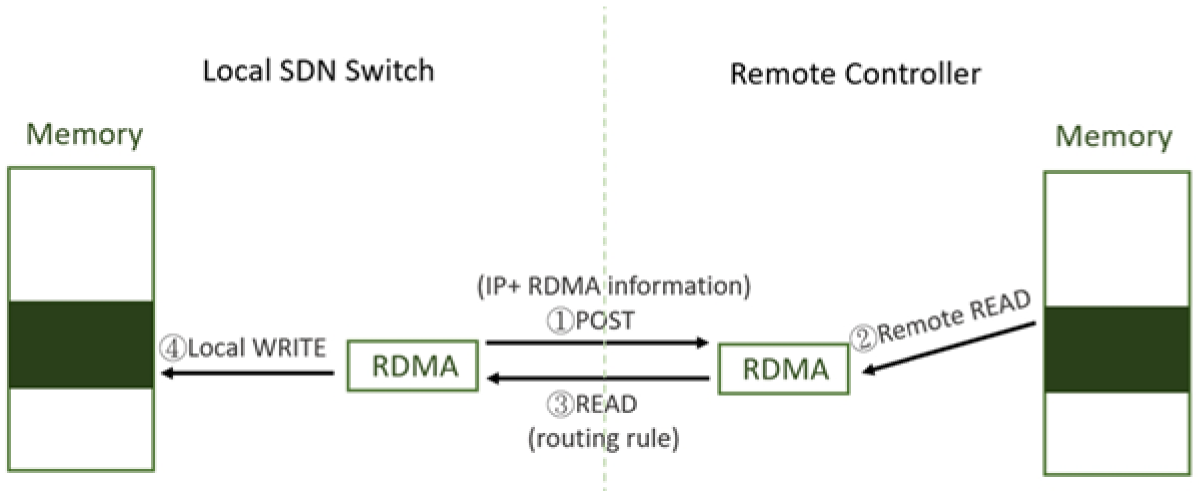


Figure 30. the steps of lookup routing rules

The third point is the transmission continuously between the local SDN switch and the remote controller. Work queue is a queue as the medium between application and RDMA, which stores the work assigned to RDMA by application. The basic unit of work is work queue element. We can divide work queue into send queue and receive queue in detail. These two are used to store work queue element of send and receive respectively.

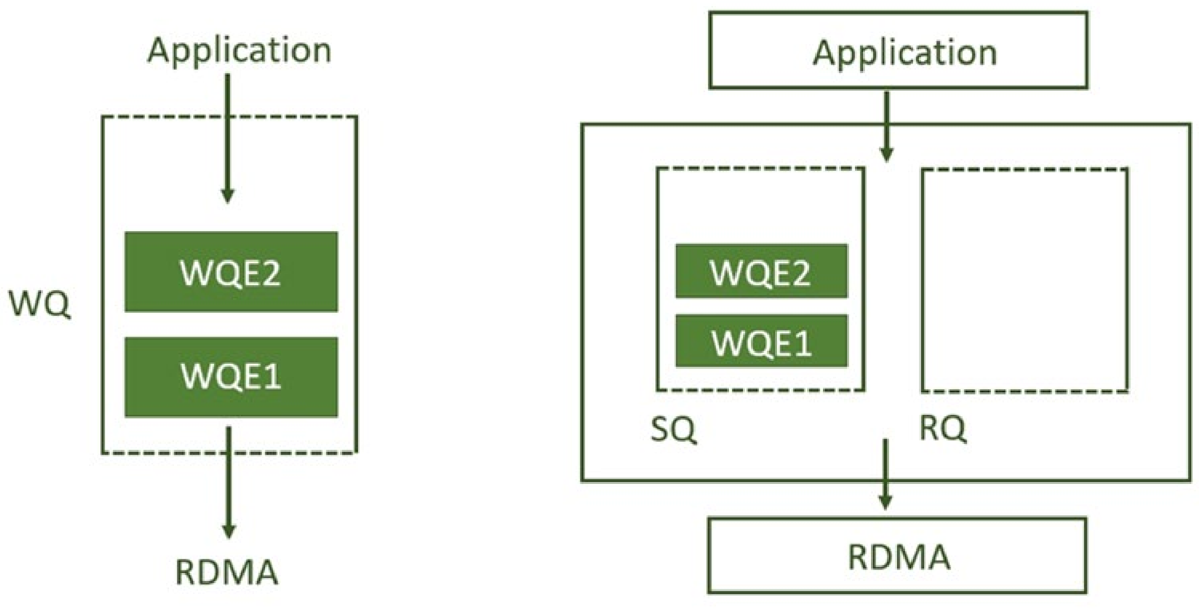


Figure 31. the architecture of RDMA assigning work

During RDMA transmission, work queue will be used to confirm the local status, and post will be used as the medium of the transmission for both ends. To enable the local SDN switch to perform RDMA read continuously, both ends need to put work queue element in their respective send queue after an RDMA transmission is completed. Then, the remote controller needs to send back the Post of MSG_READY in order that local SDN switch get the information that RDMA read transmission can continue. Although the local SDN switch receives data, both ends will put work queue element into send queue during performing RDMA read. Only when performing RDMA receive, both ends will put work queue element into send queue.

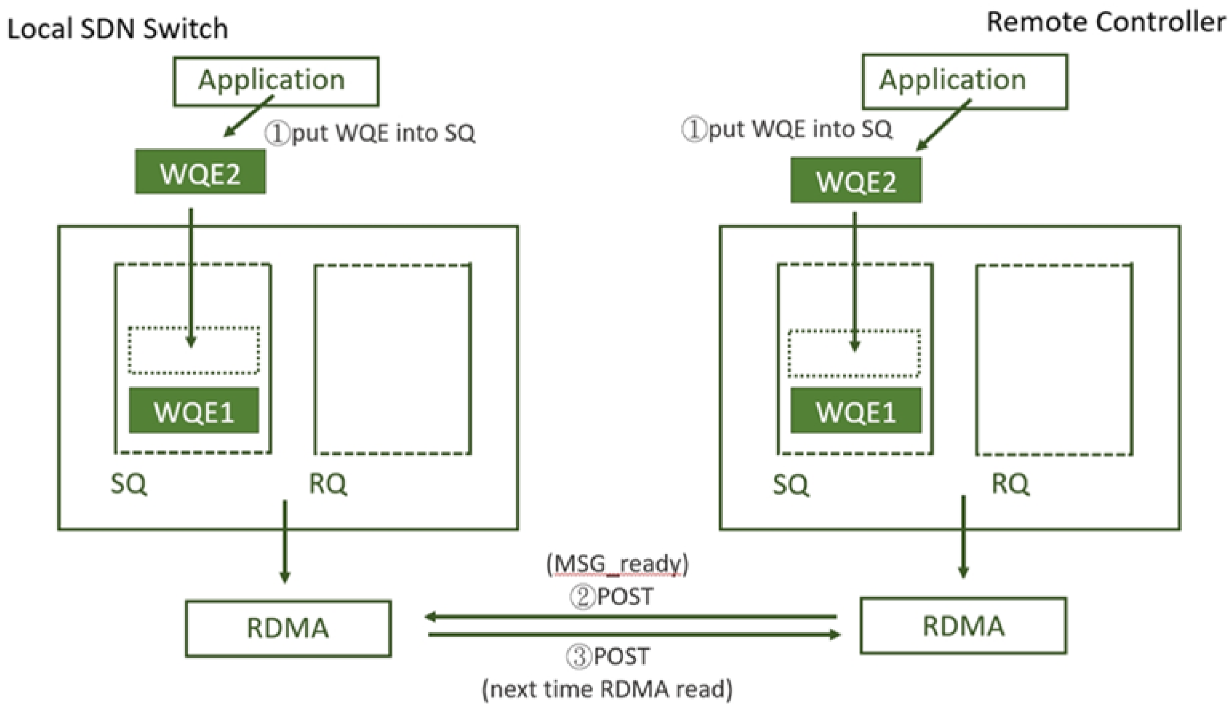


Figure 32. the implementation of continuous transmission

The fourth point is that the remote controller's memory accessed by each local SDN switch must be unique. After receiving the connecting request from the local SDN switch, the server, remote controller of RDMA, will allocate the memory location and register it in the memory region of RDMA. At the same time, the memory region will be locked. This can prevent SDN switches from occupying the memory location accessed by other connecting SDN switches.

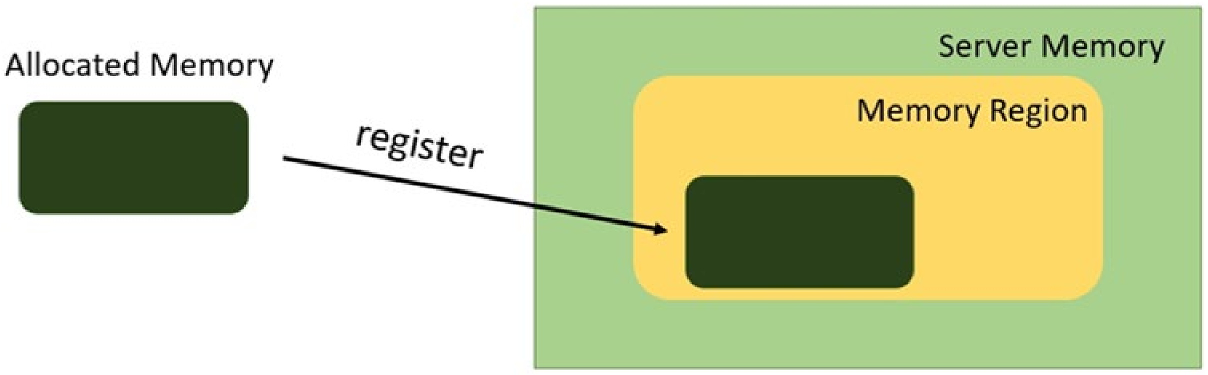


Figure 33. the lock of memory region

Finally, the fifth point is verifying the difference between RDMA and traditional transmission methods. Take the remote controller serving different numbers of local SDN switches and local SDN switches under different CPU loads as variables. We measure the transmission time and compare the results. When taking the remote controller serving different numbers of local SDN switches as a variable measurement, we use multithread to simulated local SDN switches simultaneously on the client side. When taking the local SDN switch under different CPU loads as the variable measurement, we write the dead loop in Python and execute it on the client side to simulate local SDN switches performing under different CPU loads. Finally, measure and record the transmission delay of two experiments.

Chapter 4 Results and Analysis of Experiment

1. Complex Network Architecture

In the above experiment, there are 1016 hosts, so the routing table space of S1 can only save 32 entries, that is, when the number of table entries reaches 32, we will not let it increase again. In this way, we need a flow count register size of 32. We have two count-min sketch register R1 and R2 with sizes of 32. Table updates occur five consecutive misses. The following table shows the results of five experiments.

Table 2. experimental results of switches table hit ratios

	P4 Switch using cache			**Normal P4 Switch**	
	Total packet	Table entry hit	Hit rate	Table entry hit	Hit rate
1st	38966	34501	88.54%	1646	4.22%
2nd	39196	34703	88.54%	2201	5.62%
3rd	39103	34627	88.55%	1501	3.84%
4th	40288	35826	88.92%	1701	4.22%
5th	39732	35272	88.77%	3363	8.46%

In the table above, P4 Switch using cache is a switch with the above routing algorithm installed; Normal P4 Switch is a random selection of 32 entries from 1016 flow table entries to fill in the switch. It can be found from the table that switches with routing algorithm can reach almost 90% hit rate, while switches without routing algorithm can only reach less than 10%.

Next, we pick one of them to see if entries in the routing table are stored in all major flows. The following table shows the results of the experiment.

Table 3. Comparing the routing table entry content, count value and the number of packets in the

major flow of the switch

entry	major flow	packets	table entry	count
1	10.0.1.119	982	10.0.1.119	1008
2	10.0.2.80	975	10.0.2.4	43
3	10.0.0.118	946	10.0.1.16	36
4	10.0.3.227	941	10.0.3.227	936
5	10.0.2.171	940	10.0.2.171	935
6	10.0.1.154	934	10.0.1.154	932
7	10.0.1.157	927	10.0.1.157	970
8	10.0.0.48	895	10.0.0.48	898
9	10.0.0.121	867	10.0.0.121	862
10	10.0.2.103	864	10.0.2.103	859
11	10.0.0.218	821	10.0.1.141	34
12	10.0.3.46	810	10.0.3.46	805
13	10.0.1.246	754	10.0.1.6	33
14	10.0.0.217	681	10.0.0.217	676
15	10.0.0.104	606	10.0.0.104	601
16	10.0.2.156	602	10.0.2.156	597
17	10.0.2.112	601	10.0.2.112	596
18	10.0.3.24	543	10.0.2.56	31
19	10.0.2.196	506	10.0.2.196	501
20	10.0.0.31	500	10.0.0.31	545
21	x	x	10.0.2.148	30
22			10.0.2.249	28
23			10.0.0.243	26
24			10.0.0.12	23
25			10.0.1.148	20
26			10.0.3.37	20
27			10.0.0.167	16
28			10.0.1.169	12
29			10.0.3.41	7
30			10.0.0.214	7
31			10.0.1.235	4
32			10.0.1.213	3

As we can see from the table above, 15 of the 20 major flows are stored in routing tables. This method has proved to be quite efficient.

Therefore, we can conclude that in LANs with 1016 hosts, we only need one flow count register of size 32 in the P4 switch and two count-min sketch registers of size 32, a total of 96 register spaces, so that we can substantially increase the probability of table hit. The number of forwarding rules and the space consumption ratio of registers are 1016:96, approximately 32:3.To confirm this conclusion, we will do another experiment. According to the mininet official web page, mininet can successfully open 4096 hosts [18] in a single core. We also don't need to actually turn on 4096 hosts, just add 4096 entries to the switch table. This can be done with simple_Switch_CLI plus script to complete automatically. Proportionally, we give the flow count register size 128, and both count-min sketch registers are 128. To simulate larger and more complex network systems, we enlarge the number of flows by 10 times, from 1000 to 10000, and major flows to 200 by scales, as shown in the table below.

Table 4. Experiments on large-scale systems

	P4 Switch using cache			**Normal P4 Switch**	
	Total packet	Table entry hit	Hit rate	Table entry hit	Hit rate
1st	397207	352365	88.71%	16301	4.10%
2nd	405482	360594	88.93%	19436	4.79%
3rd	402394	357695	88.89%	25074	6.23%
4th	402164	356924	88.75%	23018	5.72%
5th	401649	356784	88.83%	14528	3.62%

From the table above, you can get similar results as Table 1. It represents a method that remains stable and performs well in complex network structures.

2. RDMA Simulation

For RDMA, this study uses the following three ways to experiment and compare.

a. Comparison of number and transmission time of different local SDN switches under low CPU load.

This experiment is to simulate the transmission time comparison between RDMA transmission mode and traditional transmission mode when the remote controller serves different numbers of local SDN switches with low CPU load. Since the local SDN switch will have a certain degree of workload in the daily network environment, in the simulation process, we set the CPU load of the local SDN switch to 30%, and use this load to test the transmission time of the exchange route between the remote controller and the local switch.

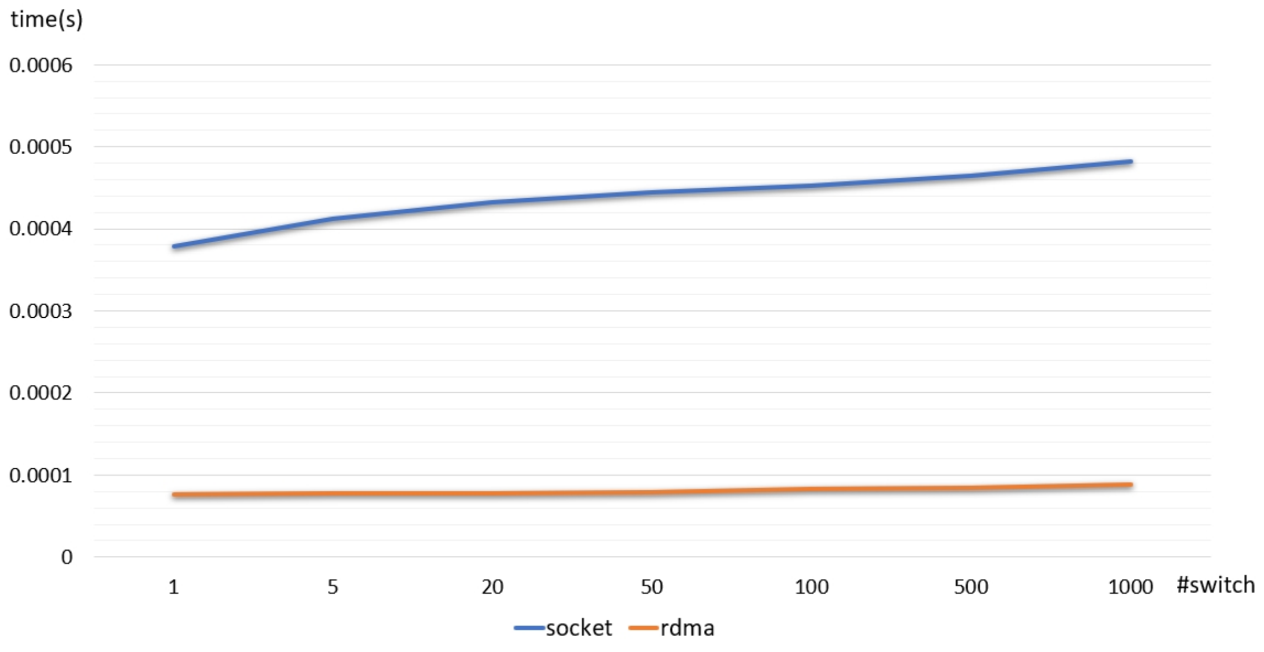


Figure 34. comparison of number of local SDN switches and transmission time under the same low CPU load

It can be seen from the above chart that when the remote controller serves the same number of local SDN switches, the transmission mode of RDMA is much faster than the traditional transmission mode; With the increase of the number of local SDN switches served by the remote controller, it can also be seen that the growth rate of RDMA transmission mode time is also smaller than that of traditional transmission mode time.

b. Comparison of CPU load and transmission time under the same number of local SDN switch connections.

This experiment is to compare the transmission time of RDMA transmission mode and traditional transmission mode by simulating the remote controller serving the same number of local SDN switches with different CPU loads. In this experiment, we simulate that the remote controller is serving 1000 local SDN switches, and measure the transmission time of the exchange route between the remote controller and the local switch under different CPU loads.

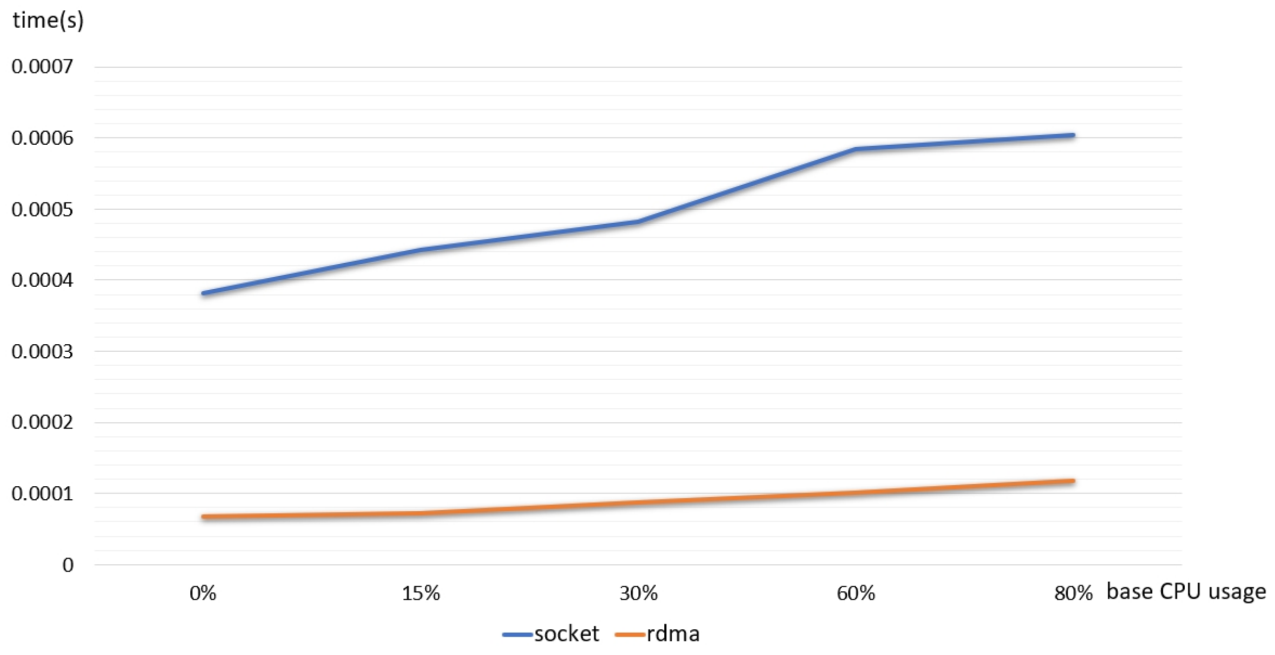


Figure 35. comparison of CPU load and transmission time under the same number of local SDN switch connections

It can be seen from the above chart that under the same number of local SDN switches served by the remote controller, the transmission time of RDMA transmission mode is still much lower than that of traditional transmission mode. With the increase of CPU load, the transmission time of traditional transmission mode increases to a great extent, while the growth rate of transmission time of RDMA transmission mode is much smaller than that of traditional transmission mode.

c. Comparison of the transmission time magnification of two transmission modes under the different number of local SDN switches.

This experiment is to compare the transmission time magnification of RDMA transmission mode and traditional transmission mode when the remote controller serves different CPUs and different numbers of local SDN switches.

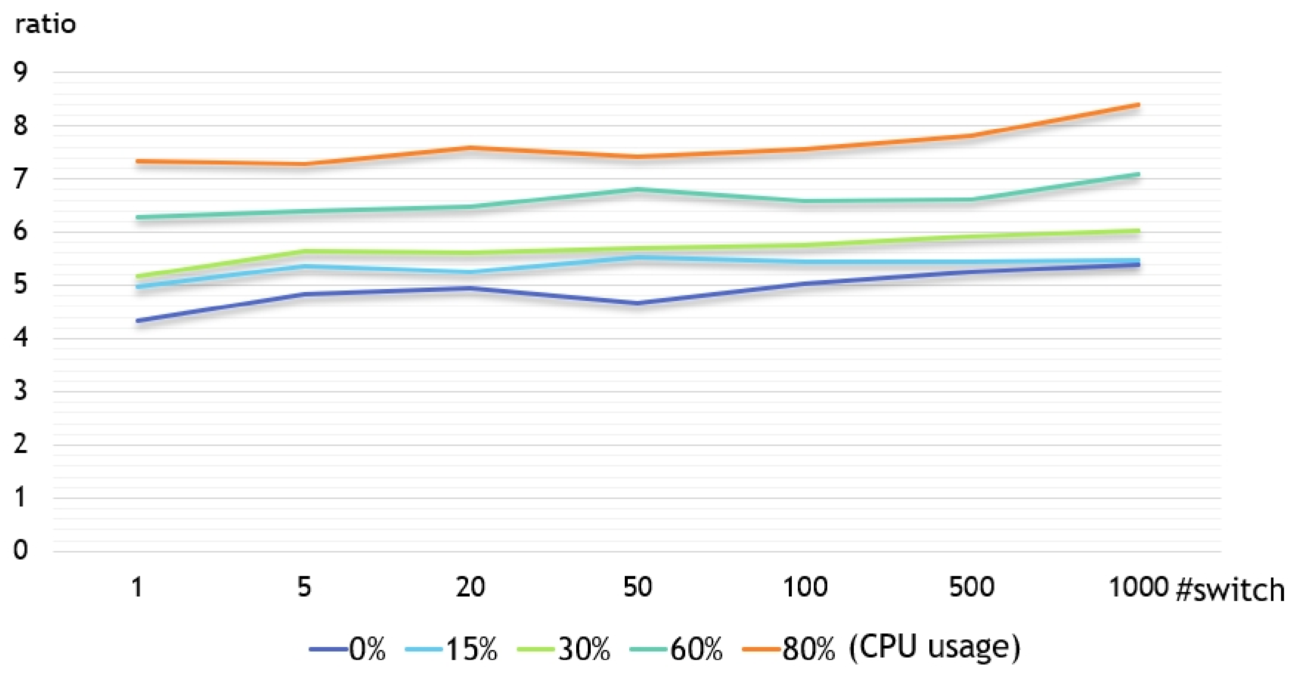


Figure 36. comparison of the transmission time magnification of two transmission modes under the different number of local SDN switches

It can be seen from the above chart that RDMA transmission mode can maintain higher transmission efficiency when the local SDN switch served by the remote controller increases or the CPU load of the local SDN switch increases, and the advantages of RDMA transmission mode are more obvious when the local SDN switch has higher CPU load (i.e. workload).

Chapter 5 Conclusions and Dissucssion

This study successfully implements the routing installation algorithm using P4 language and installs the commonly used routing rules into P4 switch without losing any packets in theory. The register in P4 is used to store flow count and count min sketch to assist the installation of algorithms for estimation and routing rule exchange. However, we only focus on whether there is a required routing message in the switch routing table, that is, the table hit times mentioned in the article. The goal of the routing installation algorithm is to reduce the table Miss times and the communication times between the switch and the remote controller while ignoring the delay that may be caused by large-scale register operation. In addition, it also takes a little time to install and delete routing rules in the routing table. If you do not use mininet, but connect a large number of actual hosts, it can not guarantee that it will not affect the packet reception quality. For the internal operation mechanism of the routing installation algorithm, when searching for the flow count value, we use the linear search of the index value from small to large because the table in the P4 runtime API table_add method is similar to the push back method, adding a new routing rule to the last side of the routing table (with the largest index value). The newly added count value is 0. In order to prevent the newly added rule from being deleted immediately, we put it behind so that we can avoid the probability of being retrieved. The advantage of this method is that it is easy to implement and does not consume too much redundant computing time or space. However, the disadvantage is that it may not be accurate enough to find the routing rule that should be deleted most. There may be more suitable search and comparison methods, which can find a better balance between efficiency and accuracy. In the experiment, the ratio of routing rules to register space is 32:3, and there may be a more suitable ratio to make the performance better.

The research also combines the transmission mode of RDMA for route search and forwarding routing. We run Soft-RoCE (software RDMA over converged Ethernet) through Linux RDMA core to simulate the transmission of RDMA in the Ethernet interface by software, so as to simulate the communication behavior between the local SDN switch and the remote controller in the whole complex network. With the above changes, we can solve the problems of low efficiency and resource occupation of traditional transmission methods.

The experiment takes the CPU load and the number of local SDN switches as the control variables to simulate the routing lookup behavior of the remote controller and the local SDN switch and measure the transmission time. The experimental results show that RDMA can complete the lookup and transmission at a faster speed regardless of the high CPU load or more local SDN switches, greatly improve the transmission efficiency and reduce the resource occupation of the local SDN switch.

We hope to use RDMA transmission mode to conduct count min sketch network telemetry through the cooperation between the remote controller and the local SDN switch, so as to solve the problem that in the past in order to pursue accuracy, the remote controller frequently sampled the local SDN switch resulting in the decline of network service, and also solve the problem of poor accuracy caused by sketching only in the local switch. This study provides a new solution for the traditional network telemetry.

References

[1] U. Wickramasinghe, A. Lumsdaine, S. Ekanayake and M. Swany, "RDMA Managed Buffers: A Case for Accelerating Communication Bound Processes via Fine-Grained Events for Zero-Copy Message Passing," 2019 18th International Symposium on Parallel and Distributed Computing (ISPDC), Amsterdam, Netherlands, 2019, pp. 121-130, doi: 10.1109/ISPDC.2019.00025.

[2] RDMA Aware Programming user manual, Mellanox Technologies, 2015. [Online]. Available: https://www.mellanox.com/related-docs/prod_software/RDMA_Aware_Programming_user_manual.pdf

[3] R. Beltman, S. Knossen, J. Hill and P. Grosso, "Using P4 and RDMA to collect telemetry data," 2020 IEEE/ACM Innovating the Network for Data-Intensive Science (INDIS), GA, USA, 2020, pp. 1-9, doi: 10.1109/INDIS51933.2020.00006.

[4] J. Vienne, J. Chen, M. Wasi-Ur-Rahman, N. S. Islam, H. Subramoni and D. K. Panda, "Performance Analysis and Evaluation of InfiniBand FDR and 40GigE RoCE on HPC and Cloud Computing Systems," 2012 IEEE 20th Annual Symposium on High-Performance Interconnects, Santa Clara, CA, 2012, pp. 48-55, doi: 10.1109/HOTI.2012.19.

[5] F. Hauser, M. Häberle, M. Schmidt and M. Menth, "P4-IPsec: Site-to-Site and Host-to-Site VPN With IPsec in P4-Based SDN," in IEEE Access, vol. 8, pp. 139567-139586, 2020, doi: 10.1109/ACCESS.2020.3012738.

[6] Y. Xue and Z. Zhu, "Hybrid Flow Table Installation: Optimizing Remote Placements of Flow Tables on Servers to Enhance PDP Switches for In-Network Computing," in IEEE Transactions on Network and Service Management, doi: 10.1109/TNSM.2020.3045711.

[7] I. Martinez-Yelmo, J. Alvarez-Horcajo, M. Briso-Montiano, D. Lopez-Pajares and E. Rojas, "ARP-P4: A Hybrid ARP-Path/P4Runtime Switch," 2018 IEEE 26th International Conference on Network Protocols (ICNP), Cambridge, 2018, pp. 438-439, doi: 10.1109/ICNP.2018.00062.

[8] G. Grigoryan, Y. Liu and M. Kwon, "PFCA: A Programmable FIB Caching Architecture," in IEEE/ACM Transactions on Networking, vol. 28, no. 4, pp. 1872-1884, Aug. 2020, doi: 10.1109/TNET.2020.3001904.

[9] T. Nguyen-Viet and D. Le, "TCAM-based flow lookup design on FPGA and its applications," 2015 International Conference on Advanced Technologies for Communications (ATC), Ho Chi Minh City, 2015, pp. 378-382, doi: 10.1109/ATC.2015.7388355.

[10] K. Gadkari, M. L. Weikum, D. Massey and C. Papadopoulos, "Pragmatic router FIB caching," 2015 IFIP Networking Conference (IFIP Networking), Toulouse, 2015, pp. 1-9, doi: 10.1109/IFIPNetworking.2015.7145296.

[11] BEHAVIORAL MODEL (bmv2) [Online] Avaliable: https://github.com/p4lang/behavioral-model

[12] p4language [Online] Available: https://github.com/p4lang

[13] Cormode, G., & Muthukrishnan, M. Count-Min Sketch. 2009. URL: http://dimacs.rutgers.edu/~graham/pubs/papers/cmencyc.pdf

[14] Cormode, G., & Muthukrishnan, S. (2005). An improved data stream summary: the count-min sketch and its applications. Journal of Algorithms, 55(1), 58-75

[15] Chelsio Communication. RoCE (RDMA over Converged Ethernet) : FAQ. 2012. URL: https://www.chelsio.com/roce/

[16] Guo, C., Wu, H., Deng, Z., Soni, G., Ye, J., Padhye, J., & Lipshteyn, M. (2016, August). RDMA over commodity ethernet at scale. In Proceedings of the 2016 ACM SIGCOMM Conference (pp. 202-215).

[17] Budiu, M. (2019). Programming networks with P4. VMware Research Blog.

[18] Mininet Overview [Online] Available: http://mininet.org/overview/

[19] Scapy Adding new protocols [Online] Available: https://scapy.readthedocs.io/en/latest/build_dissect.html

[20] Introduction to Mininet [Online] Available: https://github.com/mininet/mininet/wiki/Introduction-to-Mininet