Design and Implementation of VCP Network for Open Flow

Jie Ke, Changhong Zhu, and Yisen Lin(✉)

School of Computer Science and Engineering, Guilin University of Aerospace Technology, Guilin 541004, Guangxi, China
linyisen0409@163.com

Abstract. The traditional TCP congestion control mechanism is implicit feedback. In the high-speed network environment, it is easy to have problems such as unclear congestion indication and low efficiency, resulting in low link utilization. To solve this problem, this paper designs and implements a feedback Variable-structure congestion Control Protocol (VCP) network based on Open Flow to improve the link utilization in the high-speed network environment. Firstly, based on the POX controller, a new OpenFlow switch and controller is designed to replace the router to realize the network intermediate node function of the VCP protocol. Secondly, the TCP/IP protocol stack of Linux is studied, and the end system function of the VCP congestion control mechanism is realized by using the modular architecture of the Linux kernel function, so as to realize the VCP network terminal. Finally, the VCP is deployed on the mininet platform for the operation and test verification, and the network is built on the mininet platform to test the efficiency and fairness of the VCP. The experimental results show that the VCP protocol can achieve higher link utilization and ideal fairness in Gbit/s network.

Keywords: Congestion control mechanism · Open Flow · VCP · Linux

1 Introduction

The TCP protocol adopts the conservative congestion window growth method AIMD [1, 2] (Additive Increase Mutilplicative Decrease), which cannot effectively utilize the network bandwidth, and with the increase of high-latency data streams, the unfairness of TCP is increasingly prominent. The focus of congestion control mechanisms can be divided into two categories: based on end systems and based on intermediate nodes (based on implicit feedback [3, 4] and based on explicit feedback [5]). Implicit feedback mechanisms rely solely on end nodes to gather information (usually packet loss and delay increases) to guess the state of network congestion. The research of congestion control protocol based on explicit feedback mechanism has become an important research trend for network congestion problems.

In IP network, there are mainly two types of explicit feedback mechanisms: one is based on congestion notification feedback, such as ECN [6–8], Anti-ECN [9], Multilevel

ECN [10], MaxNet [11, 12], the routing node marks a few bits in the data packet header according to the network load information, the marking information is fed back by the data receiving end to the data sending terminal, and the sending terminal adjusts the sending rate according to the marking information. The second is based on explicit rate feedback, such as Quick Start [13, 14], XCP [15, 16], VCP [17, 18], RCP [19], CADPC [20], ACP [21], JexMax [22]. These algorithms transmit data flow state information and network state information between end systems and intermediate routing nodes by introducing congestion headers, achieving high network utilization and better fairness. Whether it is an implicit congestion control mechanism or an explicit congestion control mechanism, the focus is on the traditional network architecture. In recent years, a new network architecture, namely SDN [23] (Software Defined Network), has been proposed, which abstracts all network devices in the network as a whole and abstracts them into a network operating system, separates the control layer and the forwarding layer, and provides a global Regulates network traffic and efficiently utilizes network bandwidth, which alleviates network congestion to a certain extent.

Among the many protocols of SDN, OpenFlow is only one of the most respected and widely recognized protocols, but it is not the only way for SDN. At this stage, SDN is difficult to promote, and the implicit congestion control mechanism has shown many deficiencies. Therefore, the research on explicit congestion control protocols is still an important trend to solve congestion control. Among many explicit congestion control protocols, it is possible to The Variable-structure Congestion Control Protocol (VCP) can maintain high efficiency, fairness, and stability in both traditional and high-speed networks, and its network load feedback requires only two ECN bits. Mark. This paper studies the VCP protocol uses the OpenFlow technology to realize the VCP network intermediate node, uses the Linux TCP/IP protocol stack to realize the VCP terminal node, forms the VCP network, and verifies the performance of the VCP protocol in the high-speed network environment.

2 Related Works

2.1 VCP Protocol

Typical representatives of explicit rate-based feedback methods include VCP and XCP. This kind of algorithm transmits information such as network state information and data flow state by introducing congestion headers between network terminal systems and network intermediate nodes. However, XCP has many problems, such as increasing the load of the network, not using a promotion, increasing the burden of routers, and relatively complex network promotion projects.

(1) Load factor VCP divides the degree of network congestion into three levels: Low Load, High Load, and Overload. The Load Factor is used to reflect the degree of congestion in the network. VCP uses different combinations of the two bits of ECN in the data packet header to indicate the three levels of network congestion respectively: (00) 2 indicates that the sending terminal does not support the VCP protocol; (01) 2 indicates that the network congestion level is low load; (10) 2

indicates that the network congestion level is high load; (11) 2 network congestion level is overload. The formula for calculating the VCP load factor is:

$$\rho_l = \frac{\lambda_l + \kappa_q \cdot \widetilde{q_l}}{\gamma_l + C_l \cdot t_p}$$

Among them, research shows that the RTT value of 75%–90% of the data flow on the Internet is less than 200 ms, so take 200 ms; t_p is the incoming traffic (in bytes) within the time; λ_l is the steady-state queue length during this time; κ_q take 0.5; γ_l is the expected utilization rate of the link, take 0.98; C_l is the link bandwidth. The ECN bits are encoded according to ρ_l: when $0 \le \rho_l < 80\%$, mark (01) 2; when $80\% \le \rho_l < 100\%$, mark (10) 2; when $\rho_l \ge 100\%$, mark (10) 2.

(2) Adjustment of congestion window

The sending terminal dynamically adjusts the congestion window cwnd according to the encoding of the ECN bits in the returned ACK, and the adjustment algorithm of the congestion window is as follows: ① The ECN code is (01)2, indicating that the current network load index belongs to the low load range, and the MI algorithm is used. The expression of the congestion window cwnd is: $cwnd(t + rtt) = cwnd(t) \times (1 - \xi)$. Among them, $\xi = 0.0625$. Since each data stream has a different RTT, adjust the parameter ξ to $\xi \leftarrow (1+\xi)^{\frac{rtt}{t_\rho}} - 1$. ② The ECN code is (10)2, indicating that the current network load index belongs to the high load range, and the AI algorithm is used. The expression of the congestion window cwnd is: $cwnd(t + rtt) = cwnd(t) + \alpha$. ③ The ECN code is (11)2, indicating that the current network load index belongs to the overload range, and the MD algorithm is used. The original calculation expression of the congestion window cwnd is: $cwnd(t + \delta_t) = cwnd(t) \times \beta$. Among them, $\beta = 0.875,\ \delta_t \to 0_+$. Since MD is to recover the network from congestion and must be performed quickly, similar to an impulse behavior, the parameter β is independent of RTT, so it is not necessary to adjust according to the different RTT of each data stream.

2.2 OpenFlow Technology

The OpenFlow technology was first proposed in the literature [24]. This technology enables the network to have programmability. Nick Mckeown and his team further proposed the concept of SDN (Software Defined Network, Software Defined Network), which aims to enable users to develop applications through software, only focusing on the upper management interface without considering the underlying complex network topology. With the in-depth study of SDN, control platforms such as NOX, POX, Onix [25], Floodlight, Beacon, and Maestro have emerged. These current mainstream controllers are encapsulating the OpenFlow protocol that communicates with switches. In this paper, the network intermediate routing node is the OpenFlow switch and controller based on the OpenFlow protocol. The switch and the controller communicate through the OpenFlow secure channel. The controller configures and manages the switch functionally. The datagram is sent to the controller for processing.

2.3 OpenFlow Technology

Mininet [26] is a lightweight software-defined network and test platform, which uses lightweight virtualization technology to make a single system look like a complete network running the corresponding kernel system and user code, or simple It is understood as a process-based virtualization platform in the SDN network system, which supports various protocols such as OpenFlow and OpenvSwitch.

In summary, OpenFlow switches and controllers can be designed using OpenFlow technology to replace routers to implement intermediate nodes in VCP networks.

3 Design and Implementation of OpenFlow-Oriented VCP Network

This paper designs and implements the VCP network terminal system based on the Linux protocol stack, and designs and implements the VCP network intermediate nodes based on the Mininet platform for OpenFlow, namely the OpenFlow switch and the OpenFlow controller.

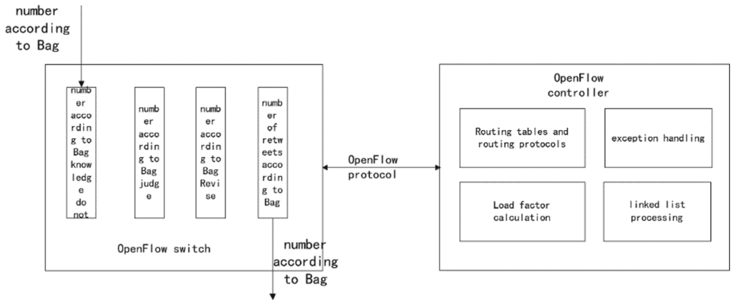
3.1 Design and Implementation of Intermediate Nodes in the VCP Network

The middle node design of the VCP network adopts the third-party controller POX, and based on the POX controller, new OpenFlow switches and controllers are designed. The new OpenFlow switch required in this paper can be designed using the POX controller. In addition, the POX controller supports compiling new controllers. Therefore, this paper uses the POX controller to design a new OpenFlow switch and controller to replace the router to realize the intermediate node of the VCP network. In this paper, the intermediate routing node is divided into a control management part and a data processing part. Among them, the control management part is responsible for the new OpenFlow controller designed according to the POX controller, responsible for the calculation of the VCP load factor, monitoring the real-time information of the switch and the modification of the flow table information; and the data processing part is responsible for the data channel function. From Fig. 1, it can be seen that the OpenFlow switch needs to realize the identification, ECN marks and forwards data packets to the corresponding ports.

This paper uses the controller POX to design a new OpenFlow switch. In order to meet the required functions, refer to the switch under the /pox/forwarding file to design a new OpenFlow switch, and its source code is also placed in the corresponding In the /pox/forwarding file, namely vcp_of_switch.py, in addition to the normal switch function, the functions that need to be implemented are as follows:

① Check whether the incoming data packet is a VCP data packet, that is, check whether the protocol number field of the IP header of the data packet is marked as the VCP protocol.

② Data collection, that is, record the length of each output queue and the data flow of each port.



number according to Bag
number according to Bag knowledge do not
number according to Bag judge
number according to Bag Revise
number of retweets according to Bag
OpenFlow switch
number according to Bag
OpenFlow protocol
OpenFlow controller
Routing tables and routing protocols
exception handling
Load factor calculation
linked list processing

Fig. 1. The overall design structure of the intermediate routing node of VCP based on OpenFlow

③ Update the ECN mark of the packet header. When a packet passes through the OpenFlow switch, it must be matched by the flow table before it can be forwarded. The design of the flow table matching rule is shown in Fig. 2:

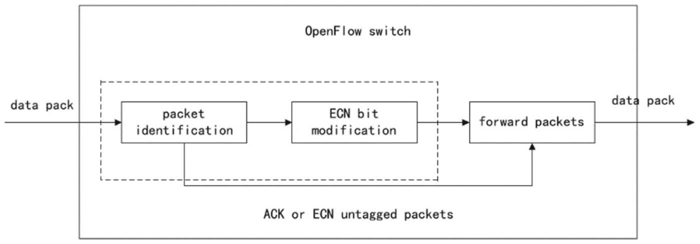


Fig. 2. OpenFlow switch packet matching rules

In order to implement the OpenFlow switch, this paper defines three classes in vcp_of_switch.py, namely VCP_Of_Switch (object), OF_Connection (object) and Switch_Features (object).

A. Class OF_Connection (object)

When the entire network starts up, the OpenFlow switch needs to establish a connection with the controller through a secure channel and maintain communication. And OF_Connection (object) is responsible for encoding and decoding the information between the switch and the controller. The main functions are shown in Table 1.

Table 1. Main functions in OF_Connections (object)

Function	Features
__init__ (self, io_worker)	Initialize the module
set_message_handler (self, handler)	set message processor
send_message (self, data)	Send information to the switch
read_message (self, io_worker)	Decode the information sent by the switch
_error_handler (self, reason, info)	Handling error messages
_extract_message_xid (self, message)	Extract the information sent by the switch
close (self)	Turn off the information processor
get_controller_id (self):	Get switch ID

B. VCP_Of_Switch (object)

VCP_Of_Switch (object) is the core part of the whole module and the main class of the OpenFlow switch function implementation. Its functions can be divided into three parts: one is the setting of the basic parameters of the switch, such as port, buffer size and flow table, etc.; The operation of the flow table entry; the third is the interaction between the switch and the controller. The implementation of these three parts is described below.

① Setting of the basic parameters of the switch The OpenFlow switch is different from the traditional switch. In addition to the basic functions of the traditional switch, it also includes the function of the flow table and the function of communicating with the controller. It is not only necessary to define the information of each port and cache of the switch in the VCP_Of_Switch (object), but also need to define the flow table in the OpenFlow protocol. The definition of each parameter of the switch and the definition of the parameters of the information that the switch interacts with the controller, in VCP_O_Switch (object), the core function that implements the above functions is __init__ (self, dpid, name, ports, miss_send_len, max_buffers, max_entries, features), in the header of the class, that is, the parameters that the initialization switch must have.

② Flow entry In order to further improve the definition and operation of the flow table, the control POX compiled the library file flow_table.py, which defines the basic composition of the flow table entry in detail, such as the definition of matching settings, instruction sets, and counters, and also defines the convection table. The operation includes adding, modifying, and deleting the flow table. The controller calls this library file, sends information to the switch, and sets the flow table, and the switch reconfigures the flow according to the information sent by the controller according to this library file. Surface. Therefore, in VCP_Of_Switch (object), the functions and functions of the controller to the flow table direction management are shown in Table 2.

Table 2. Flow table operation functions

Function	Features
_handle_FlowTableModification (self, event)	Process flow table
_flow_mod_modify (self, flow_mod, connection, table, strict=False) _flow_mod_modify_strict (self, flow_mod, connection, table)	Modify the flow table modification of the switch
_flow_mod_delete (self, flow_mod, connection, table, strict) flow_mod_delete_strict (self, flow_mod, connection, table)	Delete the flow table of the switch
_flow_mod_add (self, flow_mod, connection, table)	Add new flow table

③ Interaction between switch and controller After the switches and controllers in the network are turned on, they need to interact to establish connections. First, the switch and the controller are connected through a secure channel, and the switch will default the secure channel as a local connection. Once the secure channel is established, both the switch and the controller will send a "hello" message to each other. At the same time, the switch and the controller can send each other a message. The "echo" message measures the delay, whether the connection is maintained, etc. These two kinds of messages belong to the symmetric message, and some functions that implement the definition of the symmetric message are shown in Table 3.

Table 3. Switch and controller message part functions and functions

Function	Features
rx_message (self, connection, msg)	Process the message sent by the controller
set_connection (self, connection)	Secure Channel Connection Settings
send (self, message, connection=None)	Send a message to the controller
_rx_hello (self, ofp, connection):	process hello message
_rx_echo_request (self, ofp, connection)	Handling echo requests
_rx_echo_reply (self, ofp, connection)	Reply to the echo message sent by the controller
send_hello (self, force=False)	Send a hello message to the controller
rx_packet (self, packet, in_port, packet_data)	Handling secure channel information

In this paper, the controller not only needs to manage the flow table regularly, but also needs to obtain the port information regularly, that is, the controller calculates the average queue length every 200 ms. The calculation formula is: Among them, the weight; represents the current read average queue length; represents the weighted average queue length. Therefore, the switch must count the buffer queue length of each output queue. The controller needs to obtain the statistical information of each output port, and the controller obtains the statistical information by sending a read-state message to the switch.

2) OpenFlow controller design and implementation

The OpenFlow controller designed and implemented in this paper mainly realizes three functions: the first is the interaction between the controller and the switch; the second is the flow table management of the switch; the third is the calculation of the VCP load factor. The controller POX defines a large number of library files to support its own implementation. This paper uses it to compile a new OpenFlow controller. The implementation of the controller function is in the compiled VCP_Of_Controller.py module, which defines three classes to implement the controller. The functions are Of_Connection (object), VCP_Of_Controller (event) and Load_Factor_Computing (event). The three classes are described below. ① Of_Conection (object) The controller's class Of_Connection (object) is used to encode and decode the information that the controller communicates with the switch. Its design and implementation are the same as the function of the class Of_Connection (object) in the switch. It realizes the "hello" message when the controller communicates with the switch. At the same time as the encoding and decoding of the "echo" message, this class also needs to encode and decode the controller-to-switch message, and the sub-messages "Features", "Configuration", "Read-State", "Modify-State" under the message" and "Send-Packet" play an important role in the controller controlling the switch. ② VCP_Of_Controller (event) VCP_Of_Controller (event) is one of the core functions of the controller. It realizes the operation of the controller on the switch flow table, including the deletion, addition, and modification of the switch flow table entry, the matching rules in the flow table entry, and the action command of the instruction set. Its basic parameter settings are set according to the library file of the controller POX. In the design of this paper, the switch is connected to three hosts, so the data packet goes through the flow table matching process in Table 4.

The controller needs to delete the flow table rules of the switch and define the function _handle_table (event) to delete the flow table rules of the switch and add flow table entries. The parameters of msg, which represents a message, that is, a carrier that carries information in the communication between the controller and the switch. ③ Load_Factor_Computing(event) The main functions implemented by the defined class Load_Factor_Computing (event) are: first, the counter should keep synchronized with the switch, regularly obtain the output port queue information of the switch, and calculate the load factor; the second is to calculate the load factor, and realize the function of the calculation of its load factor in Table 5.

Table 4. Flow table matching process

Flow table ID	Flow table matching description
0	If the ECN bit of the header of the matching data packet is 00, the data packet does not support the ECN protocol and enters the flow directly; if it is not 00, it enters the next flow table 8
1	Whether the matching data packet is a returned ACK or a sent data packet, if the ACK enters the flow table 5 and matches, otherwise, it enters the next flow table;
2–3	Match the source address of the data packet, if the match is successful, enter flow table 6, otherwise enter the next flow table
4	Match the source address of the data packet, if the match is successful, enter the flow table 7, otherwise send the data packet to the controller
5–7	The ECN tag of the matching data packet matches. If the value of the data packet is less than the ECN value of the switch, the ECN of the data packet is updated to the ECN value of the switch, and then it enters the flow table 8
8	Match the source address of the packet, send the packet to the output queue of the corresponding port and wait

Table 5. Functions for calculating load factor calculations

Function	Function description
_timer_func ()	timer, synchronized with the switch
_handle_flow_stats_received (event)	Get information about each flow of the switch
_handle_ports_tats_received (event)	Obtain the traffic information of each port of the switch
_load_factor_computing (event)	Calculate persistent output queues and load factors

After the load factor is calculated, the calculated load factor should be sent to the switch, that is, the controller sends the switch flow table entry modification command, that is, to modify the switch flow Table 2, 3, 4 and 5, and define the function _flow_modify(event) to implement.

3.2 Design and Implementation of OpenFlow Monitoring Module

After completing the design and implementation of the intermediate nodes and terminals of the VCP network, the VCP network is built on the Mininet platform to verify the efficiency, fairness, and stability of the VCP protocol. This article uses the Netfilter-iptables mechanism of the Linux kernel. When the host of the Linux kernel receives and sends data packets, the data packets must be processed by Netfilter. As shown in Fig. 3, Netfilter provides four mechanisms, namely Filter, NAT, Mangle, and Raw. When the data packets pass through the linked list under each mechanism, Netfilter operates on the data packets according to the rules of the linked list. Among them, Filter is the core mechanism of Netfilter, which performs the filtering operation on the data packets. Filtering, for example, all data packets that can be specified to the TCP Port 80 of the machine are discarded. Since this article does not need to compile Netfilter, it just uses the configuration tool iptables for Netfilter, adds rules through the command line, counts the packets sent by each sender, and calculates the rate of the packets sent by each sender at the receiver.

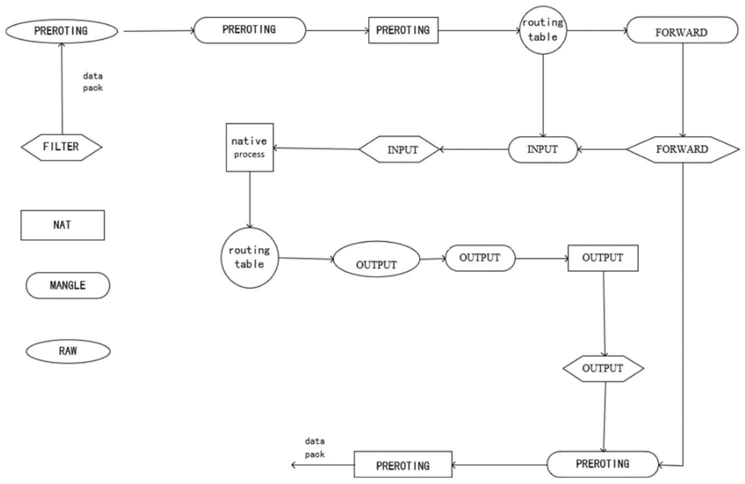


Fig. 3. Data packets go through Netfilter

In addition to realizing the above functions, the monitoring module also needs to monitor the output queue information of each output terminal. In this paper, a python program is used to write a monitoring module to monitor the bandwidth rate of the link and the output queue of each port. Its important function definitions are shown in Table 6.

3.3 Design and Implementation of VCP Terminal System

In this paper, VCP is regarded as a congestion control protocol belonging to TCP, and when the datagram is processed in the VCP part and sent to TCP, the datagram will be treated as a normal TCP datagram. When sending a data packet, TCP calls the vcp_queque_xmit() function instead of the ip_queque_xmit() function, and passes it to the VCP, which processes the data packet and forwards it to the IP. In this process, the VCP is responsible for adding the data header added at the IP layer and marking the protocol code bit used to mark the upper-layer protocol number in the header as 200 and marking the ECN mark. When implementing the VCP congestion control module, call the int inet_add_protocol (struct net_protocol *prot, unsigned char protocol) function to establish a connection with the IP. In this way, in the congestion control part of the TCP protocol, a new congestion control algorithm is defined: tcp_vcp. The function of this algorithm is to calculate and adjust the cwnd of TCP through AI, MI, and MD algorithms according to the load factor marked in the ACK header.

Table 6. Monitoring module functions

function	parameter	Function description
monitor_qlen()	iface, interval_sec, fname,	Use TC to monitor the output queue size of each port of the switch and write it into the qlen.txt of the current file
monitor_count()	ipt_args, interval_sec, fname,	Use iptables to monitor the rate at which the receiver receives the data sent by each sender, and write it into the txt file of the current file
monitor_devs()	dev_pattern, interval_sec, fname	Use /proc/net/dev to monitor the speed of sending and receiving packets on each port, and write it into the current file txt file
monitor_dev_ng	interval_sec, fname	Command line call bwm-ng network tool to monitor bandwidth and bandwidth speed

1) Send terminal For a datagram to be sent: after the user data of the application layer is added with the application header and the TCP header and encapsulated into a TCP datagram, the datagram is transmitted to the VCP by calling the vcp_queue_xmitt() function, where the member variable sk_protocol in the vcp_queue_xmitt() function = 200 (VCP protocol number), the IP header (IP_VCP) of the VCP protocol registration datagram. The VCP marks "01" in the ECN bit of the IP_VCP header, and marks the transport layer protocol number as 200, indicating that the datagram supports the VCP protocol. Calling the function ip_queue_xmit(), the VCP transmits the datagram to the IP layer. Figure 4 shows the encapsulation process of a datagram to be sent.
2) Receiving terminal For a datagram to be received: If the transport layer protocol number is marked as VCP, the IP datagram enters the VCP processing part through the IP layer. After reading the congestion level information marked by the ECN in the datagram header, the VCP forwards the processed datagram to TCP. If the transport layer protocol is marked as TCP, the datagram is passed directly to the TCP layer. The TCP protocol stack processes standard TCP datagrams. Figure 5 shows the processing of a datagram to be received. In the standard data packet receiving process, if it is confirmed that the data packet is a TCP data packet, TCP will call the tcp_v4_rcv function to receive the data packet. From the Linux kernel version 2.6.9, the data structure of the TCP protocol is no longer modularized. If it needs to be modified The data structure of TCP can only recompile the source code of the kernel. Therefore, in order to reduce the workload, a modular congestion control algorithm is used. Therefore, in the receiving process of VCP data packets, VCP is sent from IP to VCP for the first time. The VCP reads and saves the ECN mark of the data packet, the data packet is returned to the Backlog again, and the data packet

is processed again according to the standard receiving process of the data packet. As shown in Fig. 5.

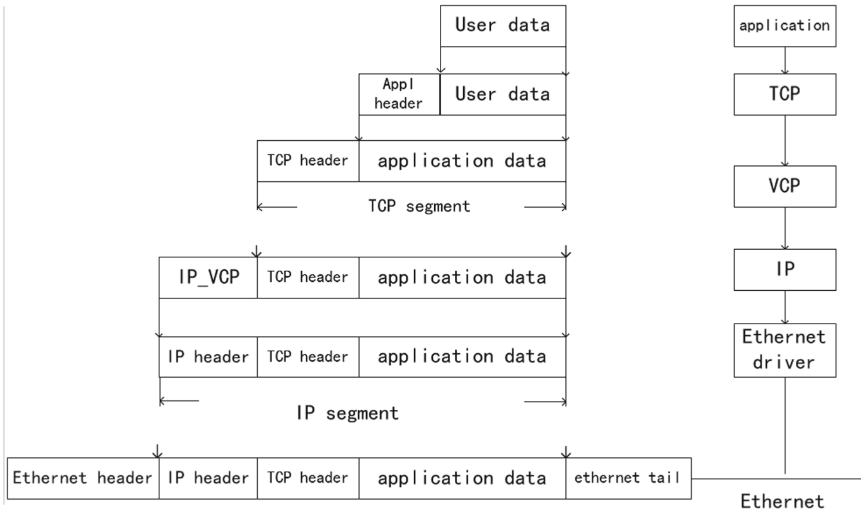


Fig. 4. Sending datagram encapsulation process

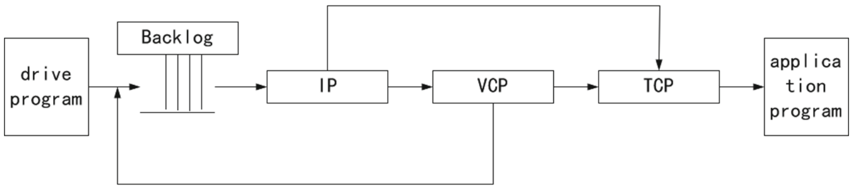


Fig. 5. Received datagram processing process

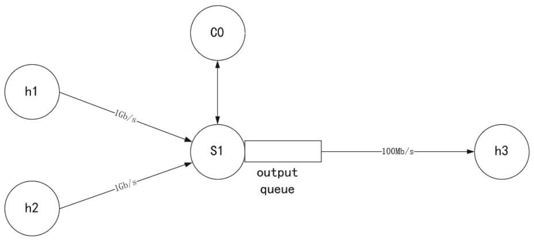


Fig. 6. Experimental topology

4 Validation Experiments

4.1 Construction of the Experimental Network

A PC with a CPU frequency of 3.2 GHz is used as the platform, and VirtualBox is installed to build a Mininet platform to simulate a small VCP single-bottleneck local area network to verify the efficiency and fairness of the VCP congestion control protocol. As shown in Fig. 6, a typical VCP network topology is constructed for verification, in which h1, h2, and h3 respectively represent three end systems, namely hosts; s1 represents an OpenFlow switch; c0 represents a controller. Among them, the link bandwidth between h1 and h2 and the switch is 1 Gbps, and the link bandwidth between h3 and the switch is 100 Mbps, that is, the bottleneck bandwidth. The Mininet platform runs on Ubuntu 13.04 that supports the VCP protocol, and its simulation network is built as shown in Fig. 7.

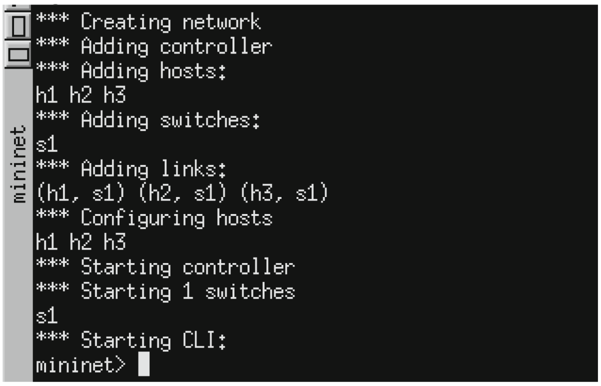

```
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
```

Fig. 7. Network topology simulated by Mininet

4.2 VCP Efficiency Test

1) Link utilization test In the VCP network, the network bottleneck is set on the s1-h3 link, and the bandwidth utilization of the VCP protocol on the s1-h3 link is tested. The bandwidth is set to 100 Mbps, and the bandwidth of the s1-h1 and s1-h2 links is is 1Gbps. Use the command line commands of the xterm interface of the host h1 and the host h2 to continuously send data to the host h3 to test the link bandwidth utilization of the switches s1-h3. In the Mininet platform, the Drop Tail mechanism, namely TCP/DT, is implemented. When output queues are full, all subsequent datagrams are discarded. In the experiment, the utilization ratio of VCP network and ordinary TCP (TCP/DT) network in the bottleneck section is compared. The experimental results are shown in Fig. 8, and the horizontal axis represents time.

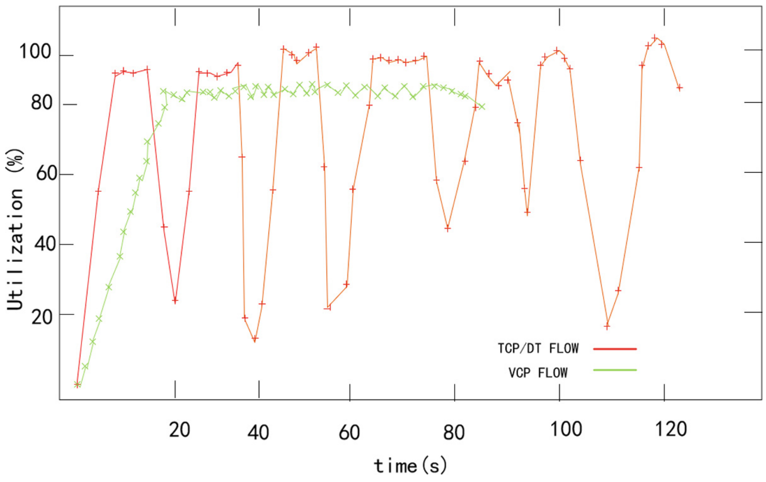


Fig. 8. Bottleneck link utilization

2) Transmission delay test In order to test the link transmission delay of the VCP protocol as the throughput changes, in the experiment, the host h1 sends the data packet to the host h3, and the bandwidth of the link s1-h1 and s1-h3 segments is set to the default value of 1Gbps. Mininet simulates the network that supports the VCP mechanism and the traditional TCP (TCP/DT) network, respectively, and calculates the change of the packet transmission delay with the throughput, as shown in Fig. 9: the horizontal axis represents the throughput (%), and the vertical axis represents the link transmission Delay (ms). It can be seen from Fig. 9 that when the throughput is low (below 60%), both the VCP and TCP/DT mechanisms can maintain a low transmission delay. However, as the throughput increases, the transmission delay of the TCP/DT congestion control mechanism increases significantly, almost showing an exponential growth trend. When the throughput is high, it even reaches 13ms; relatively speaking, the transmission delay of the VCP protocol increases with the change in throughput does not fluctuate significantly, and it can basically be kept below 3ms. The experimental results show that the VCP protocol can maintain a relatively ideal transmission delay.

4.3 VCP Fairness Test

In order to test the fairness of the VCP protocol, that is, to measure whether each user or each link in the network bottleneck link segment can share network resources fairly, the test still adopts the configuration in Sect. 3.1: the experimental bottleneck is the s1-h3 segment. The link bandwidth of the bottleneck segment is set to 100 Mbps, and the link bandwidth of the s1-h1 segment and the s1-h2 segment is 1Gbps. After the experiment starts, the host h1 and the host h2 send data packets to the host h3 at the same time. By testing the bandwidth utilization of the intermediate network segment between the switch s1 and the host h3, it is checked whether the flow1 sent by the sending host h1 and the flow2 sent by the sending host h2 are fair. Share the bandwidth of the bottleneck link s1-h3 segment. Figure 9 shows the change of the bottleneck link utilization rate of flow1 and flow2 with time, the horizontal axis represents time, and the vertical axis is the bandwidth utilization rate of each sending host on the link bottleneck s1-h3 segment link.

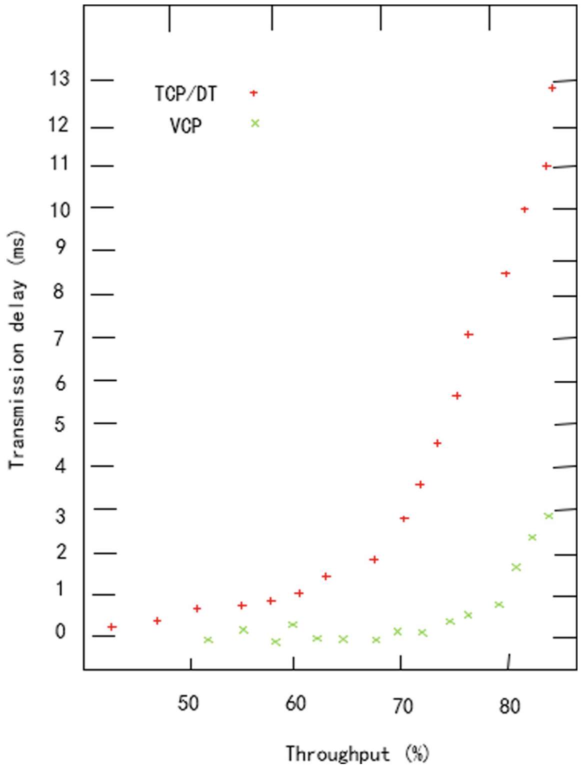


Fig. 9. Transmission delay at different throughputs

Figure 10 VCP fairness test As can be seen from Fig. 10, the total utilization of VCP flow 1 and VCP flow 2 to the bottleneck link s1-h3 is still relatively ideal. Although it fluctuates slightly after stabilization, it can generally be maintained between 35% and 45%. Although there is a slight inequity in the allocation of network resources, on the whole, the two links can better share the bandwidth of the bottleneck link. 5 Conclusion This paper designs and simulates a realistic VCP network verification platform. First, the Linux network protocol stack is used to implement the VCP network end system; secondly, the VCP network intermediate routing node composed of OpenFlow switches and controllers is implemented using POX; Mininet platform builds a typical VCP experimental verification network. Among them, the host in the network runs the VCP network end system, the OpenFlow switch and the controller realize the intermediate routing node of the VCP network, and an experiment is designed to verify the efficiency and fairness

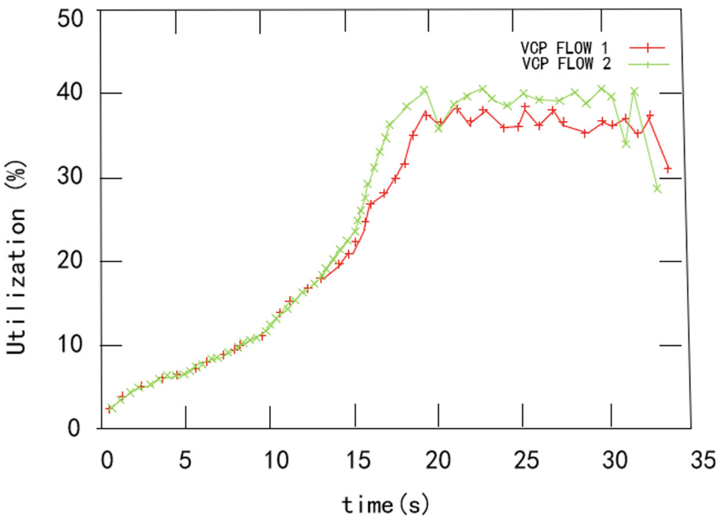


Fig. 10. VCP fairness test

of the VCP protocol in the real network environment. The experimental results show that, compared with the traditional TCP/DT mechanism, the VCP protocol can make better use of the resources provided by the link, the efficiency of the protocol is still higher; and the VCP protocol can better realize the network resources without the link. The distribution among the roads has better fairness. The VCP protocol only uses two binary bits of the ECN to encode the congestion level information, which can achieve relatively good performance in a real network environment.

Acknowledgments. This work is supported in part by Guangxi Young and Middle-aged Research Capacity Enhancement Project "Research and Application on The Key Technology of Intelligent Experiment Management Platform (Grant No. 2020ky21026) and the 2020 school-level Scientific Research Project in Guilin University of Aerospace Technology (Grant No. XJ20KT18: Research on Technology of Error-tolerance Recognition of Channel Coding parameters in non-cooperative communication).

References

1. Dah-Ming, C., Raj, J.: Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. Comput. Netw. ISDN Syst. **17**(1), 114 (1989)
2. Yang, Y.R., Lam, S.S.: General AIMD congestion control. In: Proceedings of IEEE ICNP, pp. 187–198 (2000)
3. Lefelhocz, C., Lyles, B., Shenker, S., Zhang, L.: Congestion control for best-eort service: why we need a new paradigm. IEEE Netw. **10**(1), 10–19 (1996)
4. Gevros, P., Crowcroft, J., Kirstein, P., Bhatti, S.: Congestion control mechanisms and the best effort service model. IEEE Netw. **15**(3), 16–26 (2001)

5. Bakshi, B.S., Kirshna, P., Vaidya, N.H., Pradhan, D.K.: Improving performance of TCP over wireless networks. In: Proceeding of the 1st Annual International Conference on Mobile Computing and Networking, pp. 2–11. ACM, New York (1995)
6. Ramakrishnan, K., Floyd, S., Black, D.: The addition of explicit congestion notification (ECN) to IP (2001)
7. Oljira, D.B., Grinnemo, K.J., Brunström, A., et al.: MDTCP: practical latency-aware multipath congestion control for datacenter networks (2020)
8. Chen, C., Fang, H.C., Iqbal, M.S.: QoSTCP: provide consistent rate guarantees to TCP flows in software defined networks. In: ICC 2020-2020 IEEE International Conference on Communications (ICC), pp. 1–6. IEEE (2020)
9. Hu, J., Huang, J., Li, Z., et al.: AMRT: anti-ECN marking to improve utilization of receiver-driven transmission in data center. In: 49th International Conference on Parallel Processing-ICPP, pp. 1–10 (2020)
10. Durresi, A., Sridharan, M., Liu, C., Goyal, M., Jain, R.: Traffic management using multilevel explicit congestion notification. In: Proceedings of SCI (2001)
11. Gronát, P., Aldana-Iuit, J.A., Bálek, M.: MaxNet: neural network architecture for continuous detection of malicious activity. In: 2019 IEEE Security and Privacy Workshops (SPW), pp. 28–35. IEEE (2019)
12. Wydrowski, B., Andrew, L., Mareels, I.: MaxNet: Faster Flow Control Convergence. In: Mitrou, N., Kontovasilis, K., Rouskas, G.N., Iliadis, I., Merakos, L. (eds.) NETWORKING 2004. LNCS, vol. 3042, pp. 588–599. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24693-0_49
13. Floyd, S., Allman, A., Jain, M., Sarolahti, P.: Quick-Start for TCP and IP. Draft-ietf-tsvwg-quickstart-07 txt (2006)
14. Zhang, D., Zheng, K., Zhao, D., et al.: Novel quick start (QS) method for optimization of TCP. Wirel. Netw. **22**(1), 211–222 (2016)
15. Katabi, D., Handley, M., Rohrs, C.: Congestion control for high bandwidth-delay product networks. In: Proceedings of the 2002 SIGCOMM Conference, vol. 32, pp. 89–102. ACM, New York (2002)
16. Jiang, N., Huang, J., Liu, S., et al.: Achieving fast convergence and high efficiency using differential explicit feedback in data center. In: ICC 2020–2020 IEEE International Conference on Communications (ICC), pp. 1–6. IEEE (2020)
17. Xing, S., Yin, B., Yao, J., et al.: A VCP-based congestion control algorithm in named data networking. In: 2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), pp. 463–468. IEEE (2018)
18. Zhang, H., Zhou, H., Chen, C., et al.: Fast fairness convergence through fair rate estimation in Variable-structure congestion control protocol. Comput. Commun. **70**, 54–67 (2015)
19. Barreto, L.: XCP-Winf and RCP-Winf: improving explicit wireless congestion control. J. Comput. Netw. Commun. (2015)
20. Teymoori, P., Welzl, M., Gjessing, S., et al.: Congestion control in the recursive internetworking architecture (RINA). In: 2016 IEEE International Conference on Communications (ICC), pp. 1–7. IEEE (2016)
21. Shreedhar, T., Kaul, S.K., Yates, R.D.: ACP: age control protocol for minimizing age of information over the internet. In: Proceedings of the 24th Annual International Conference on Mobile Computing and Networking, pp. 699–701 (2018)
22. Zhang, Y., Leonard, D., Loguinov, D.: Jetmax: scalable max-min congestion control for high-speed heterogeneous networks. Comput. Netw. **52**, 1193–1219 (2008)
23. Jasim, M.N.: A proposed adaptive least load ratio algorithm to improve resources management in software defined network OpenFlow environment. Karbala Int. J. Mod. Sci. **7**(1), 6 (2021)
24. McKeown, N., et al.: OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Comput. Commun. Rev. **38**(2), 69–74 (2008)

25. Lanversin, J.T., Kütt, M., Glaser, A.: ONIX: an open-source depletion code. Ann. Nucl. Energy **151**, 107903 (2021)
26. Chaudhary, R.: Software-defined networking based control flow optimization for multi-cloud environment (2021)