DOI: 10.3969/j.issn.1673-5188.2017.04.006
http://kns.cnki.net/kcms/detail/34.1294.TN.20171020.1814.002.html, published online October 20, 2017

Technical Analysis of Network Plug-in Flannel for Containers

YANG Yong, DONG Xiugang, and DONG Zhenjiang
(Nanjing R&D Center, ZTE Corporation, Nanjing 210012, China)

Abstract

The development of cloud computing has made container technology a hot research issue in recent years. The container technology provides a basic support for micro service architecture, while container networking plays an important role in application of the container technology. In this paper, we study the technical implementation of the Flannel module, a network plug-in for Docker containers, including its functions, implementation principle, utilization, and performance. The performance of Flannel in different modes is further tested and analyzed in real application scenarios.

Keywords

container technology; Docker; network plug-in; Flannel

1 Overview

With the development of cloud computing, container technology has become one of the hot research issues in recent years. In order to utilize container technology in the production environment, container network should be addressed firstly [1]. The container networking technology [2] connects inter-host containers and isolates each container network with others. Currently, the container networking technology has two solutions [3]: overlay networking and network routing. In overlay networking, application layer protocols are encapsulated and overlaid on the existing IP layer. Typical applications include Weave, Flannel, and Open vSwitch. In network routing, container networking is implemented through routing on network Layer 3 or 2. Typical applications include Calico and Macvlan. The overlay networking solution has no additional requirements on the existing network and container networking is available without the need to change the underlying network, so this mature solution is widely applied. However, with the increase of network nodes, the network is getting more and more complicated, and it is becoming more difficult and costly to maintain the network [4]. In addition, overlay networking is implemented through encapsulation on the application layer, so performance is also a large concern. Network routing features high network transmission performance and easy network maintenance, but has particular requirements on routers and switches in networking, which affects its scope of application.

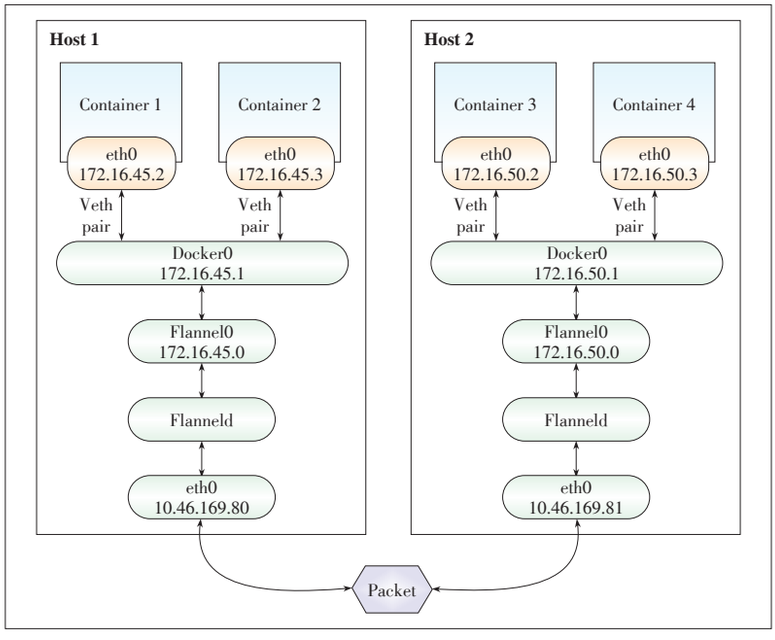
Flannel, a representative of the overlay networking solution for container networking, is a network plug-in service designed by CoreOS for Kubernetes. With Flannel, each Docker container created on different hosts in the container cluster can have a unique virtual IP address in the cluster [6], and Docker container networks can interwork with each other. Flannel is an independent open-source project using the Apache License Version 2.0 protocol and compiled with the Go language. Flannel is easy to use with a comprehensible implementation principle and has become a mainstream network plug-in recommended by many container solution providers.

This paper will provide an in-depth analysis of the implementation principle, as well as its utilization methods and modes of Flannel.

2 Implementation Principle of Flannel Network

Flannel provides an independent subnet for each host, and the network information of the entire cluster is stored on etcd, a distributed lock service. The IP address of a target container in the subnet of the host is queried from etcd for forwarding data across hosts. **Fig. 1** shows the implementation principle.

In Fig. 1, Host 1 and Host 2 are two hosts, on each of which two containers are operating. If the user datagram protocol (UDP) mode [7] is used for a request that needs to be sent from Container1 to Container3, the virtual network interface controller (NIC) Flannel0 (the virtual NIC name in virtual extensible



▲**Figure 1. Implementation principle of flannel network.**

LAN (VXLAN) mode may be different) is created.

The implementation procedure is described as follows:

1) IP datagrams are encapsulated and sent through eth0 of the container.
2) Container1's eth0 interacts with Docker0 through veth pair and sends a packet to Docker0. Docker0 then forwards the packet.
3) Docker0 determines that the IP address of Container3 points to an external container by querying the local routing table, and sends the packet to the virtual NIC Flannel0.
4) The packet received by Flannel0 is forwarded to the Flanneld process. The Flanneld process encapsulates the packet by querying the routing table maintained by etcd and sends the packet through eth0 of the host.
5) The packet determines the target host in the network across hosts.
6) The Flanneld process that listens to the 8285 port on the target host decapsulates the packet.
7) The decapsulated packet is forwarded to the virtual NIC Flannel0.
8) Flannel0 queries the routing table, decapsulates the packet, and sends the packet to Docker0.
9) Docker0 determines the target container and sends the packet to the target container.

3 Network Forwarding Modes Supported by Flannel

The Flannel configuration file is stored under the /coreos.com/network/config directory of etcd and can be viewed by running the etcdctl command. The directory where the configuration file is stored can be changed through --etcd-prefix.

The configuration file is a json file, consisting of the following five key parameters:

- Network (character string): IPv4 Classless Inter-Domain Routing (CIDR) of the entire Flannel network. It is the unique required keyword.
- SubnetLen (integer type): length of the subnet allocated. 24 is set as default (eg. /24).
- SubnetMin (character string): start IP address allocated to a subnet. The first address of the subnet is set as default.
- SubnetMax (character string): end IP address allocated to a subnet. The last address of the subnet is set as default.
- Backend (dictionary): type and specific configuration used by the back end. The keywords supported in the dictionary are described below (default: UDP).

Backend specifies the transfer mode of an encapsulated packet and supports the following modes:

1) UDP: encapsulating packets through UDP [8]
- type (character string): udp
- Port (digit): UDP port number used for sending encapsulated packets, with 8285 set as default

2) vxlan: encapsulating packets through kernel VXLAN [9]
- type (character string): vxlan
- VNI (digit): ID used by the VXLAN [10](VNI), with 1 set as default
- Port (digit): UDP port number used for sending encapsulated packets, with 8472 set as default (kernel default value)

3) host-gw: creating an IP route to a subnet through the IP address of the remote host. This requires the hosts that are operating Flannel are directly interconnected on Layer 2.
- type (character string): host-gw

4) aws-vpc: used for the containers operating on AWS
5) gce: used for containers operating on GCE
6) alloc: only implementing network distribution, with no packet forwarding.

Fig. 2 shows a json file for packet forwarding in UDP mode. The technical implementations in UDP, VXLAN, and host-

```
{
  "Network": "10.0.0.0/8",
  "SubnetLen": 20,
  "SubnetMin": "10.10.0.0",
  "SubnetMax": "10.99.0.0",
  "Backend": {
    "Type": "udp",
    "Port": 7890
  }
}
```

Figure 2. ▶ A json file in UDP mode for sending packets.

GW modes are described below.

4 Implementation Principle and Analysis of Flannel Network Forwarding Modes

Flannel provides the networking service for containers and uses etcd to store the network configuration information [11], [12]. Flannel obtains subnet information from etcd, declares the network segment of its subnet, and stores it in etcd. Before data forwarding across hosts, Flannel queries the IP address of the host corresponding to the subnet and sends data to flanneld of the host, and the flanneld forwards the data. The following analyzes the data stream sending and receiving flows in UDP, VXLAN, and host-GW modes.

4.1 UDP Mode

UDP is the default mode used by Flanneld. The virtual NIC flannel0 is added on a host and uses the 8285 as the listening port by default.

According to the Wireshark analysis (**Fig. 3**), the data is forwarded in the UDP mode. The source and destination IP addresses of the source packet are the IP address of the host, and the IP addresses of containers are encapsulated into the packet.

4.2 VXLAN Mode

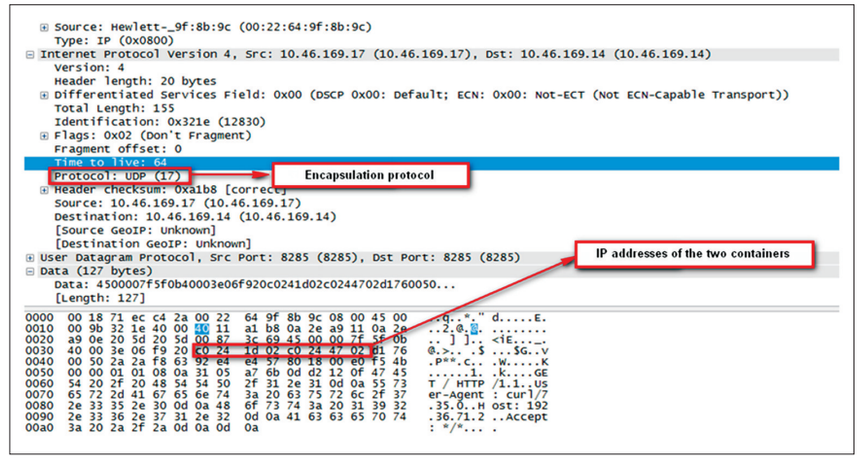
In VXLAN mode, Flannel runs a flanneld process on each host and creates a VXLAN device (virtual NIC Flannel.x, where x indicates the VNI). In this test, VNI is 1, so the virtual NIC Flannel.1 is created.

According to the Wireshark analysis as shown in **Fig. 4**, the data is forwarded also through UDP, and the source and destination IP addresses are the IP addresses of the host. Some VXLAN encapsulation protocols are added in the packet (which is different from the UDP mode), and the IP addresses of the two containers are encapsulated into the packet.

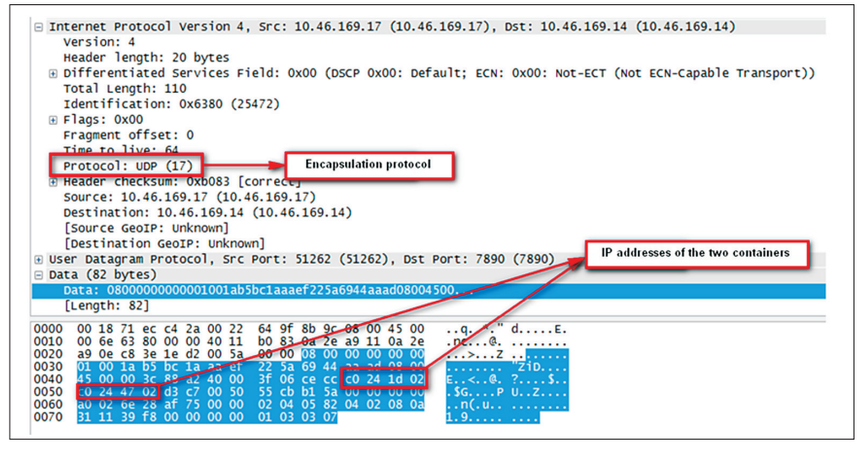
4.3 Host-GW Mode

Different from that in UDP and VXLAN modes, no Flannel virtual NIC is created in the host-GW mode, but the hosts need to be interconnected on Layer 2. In this test, the IP addresses of two containers are 192.38.88.2 and 192.38.95.2.

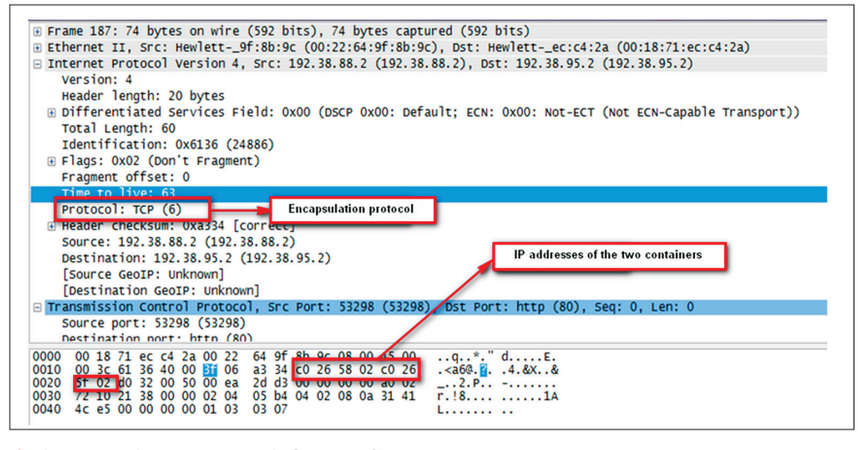
According to the Wireshark analysis (**Fig. 5**), the source and destination IP addresses in the packet are the IP addresses of the two containers but not the IP address of the host. No UDP-based packet encapsulation is implemented. Instead, the pack-



▲Figure 3. Wireshark analysis for UDP mode.



▲Figure 4. Wireshark analysis for VXLAN mode.



▲Figure 5. Wireshark analysis for host-GW mode.

et is transmitted through TCP.

5 Conclusions

Our tests were conducted in two servers with the same configuration, including the servers with 100M network card, operating system ubuntu14.04, Flannel v0.7.0, and etcd v2.3.1. Besides, Iperf, the network performance testing tool for testing the maximum bandwidth performance of TCP or UDP, is used, and two containers are running on each of the hosts. **Table 1** describes the results of the performance tests.

▼Table 1. Performance comparision of direct data transmission between two hosts with UDP/VXLAN/host-GW-based transmission through Flannel.

Mode	Rate (M/s)	Description
Direct transmission between two hosts	94.3	There is a normal performance loss in direct transmission between two hosts.
UDP	92.7	There is greater performance loss.
VXLAN	91.2	
Host-GW	94.3	The performance is approximate to that of direct transmission between two hosts.

GW: gateway
UDP: user datagram protocol
VXLAN: virtual extensible local area network

According to the testing results, the host-GW mode provides the best performance, because no virtual NIC is created in this mode, which thus reduces the number of forwarding operations and improves network performance. However, in this mode, the destination address is the container's IP address and media access control (MAC) addressing is used, which requires that the two hosts be interconnected on Layer 2.

References

[1] J. Turnbull, *The Docker Book: Containerization is the New Virtualization*. Seattle, USA: Amazon Digital Services, Inc., 2014.
[2] dotCloud. (2017, Apr. 30). Docker: build, ship and run any app, anywhere [Online]. Available: https://www.docker.com
[3] B. Yang, W. Dai, and Y. Cao, *Docker Primer (in Chinese)*. Beijing, China: China Machine Press, 2014.
[4] H. Sun Hongliang, *The Source Code Analysis of Docker (in Chinese)*. Beijing, China: China Machine Press, 2015.
[5] K. Wang, G. Zhang, and X. Zhou, "Research on the container-based virtualization technology," *Computer Technology and Development*. vol. 25, no. 8, pp. 138–141, Aug. 2015. doi: 10.3969/ j.issn.673-629X.2015.08.029.
[6] S. Du, "The application of virtual reality technology in computer network platform," *Computer Engineering & Software*, vol. 34, no. 1, pp. 45–46, Jan. 2013.
[7] G. Peng, "UDP-based point-to-point fast and reliable transmission model," Dissertation, Sun Yat-Sen University, Guangzhou, China, 2013.
[8] F. Zhao and Z. Ye, "Contract analysis of UDP and TCP and improvement of UDP in reliability," *Computer Technology and Development*, vol. 16, no. 9, pp. 219–221, Sept. 2006.
[9] H. Zhao, Y. Xie, and F. Shi, "Network Virtualization and Network Function Virtualization," *ZTE Technology Journal*, vol. 20, no. 3, pp. 8–11, Jun. 2014. doi: 10.3969/j.issn.1009-6868.2014.03.002.
[10] Z. Lu, Z. Jiang, and B. Liu, "A VXLAN-Based Virtual Network Access Control Method," *Computer Engineering*, vol. 40, no. 8, pp. 86–90, Aug. 2014.
[11] L. Wu Longhui, *Kubernetes*. Beijing, China: Publishing House of Electronics Industry, 2016.
[12] SEL of Zhejiang University, *Docker Container and Container Cloud*. Beijing, China: Posts & Telecom Press, 2015.

Manuscript received: 2017-07-26

Biographies

YANG Yong (yang.yong3@zte.com.cn) works with ZTE Corporation and focuses on ICT application R&D and software architecture technology research. His research interests include service platforms, capability opening, multimedia technologies, and cloud computing.

DONG Xiugang (dong.xiugang20@zte.com.cn) works with ZTE Corporation. His research interests include the Web and container technologies (including the micro service and container cluster management technologies) and their application, and natural language processing.

DONG Zhenjiang (dong.zhenjiang@zte.com.cn) works with ZTE Corporation. His research interests include security technologies , cloud computing and artificial intelligence.