Dynamic reverse proxy chain generation for networks in data centers

1st Yang Bai
School of Data and Computer Science
Sun Yat-sen University
Guangzhou, China
baiy8@mail2.sysu.edu.cn

2nd Guixin Guo
National Supercomputing Center
Guangzhou, China
guixin.guo@nscc-gz.cn

3rd Yong Wang
School of Data and Computer Science
Sun Yat-sen University
Guangzhou, China
wangy579@mail2.sysu.edu.cn

4th Kangyou Zhong
National Supercomputing Center
Guangzhou, China
kangyou.zhong@nscc-gz.cn

5th Jiang Li
National Supercomputing Center
Guangzhou, China
jiang.li@nscc-gz.cn

6th Yunfei Du
School of Data and Computer Science
Sun Yat-sen University
Guangzhou, China
yunfei.du@nscc-gz.cn

***Abstract*—Reverse proxy, as one of the important components required by the data center to provide application services, has functions of access control, load balancing, connecting to different types of networks, etc. In the future, as the application services requiring reverse proxy further increase, network types become more and more diverse and complex, and the network hierarchy becomes higher and higher, reverse proxy will change from a single layer to multiple layers to form a reverse proxy chain. The construction of the reverse proxy chain will become one of the bottlenecks of data center networking operation and maintenance.**

In this paper, we propose a method of automatically constructing reverse proxy chains to avoid the problem of manual static configuration of the reverse proxy chain which is time-consuming, laborious, and difficult to maintain. In our software-defined networking experiment, we simulated a full-binary-tree-like topology of 1534 nodes. We recorded the time to generate and remove proxy chains of various lengths. The average time to generate all reverse proxy chains in the topology consisting of 1534 nodes with 100ms delay is around 5100ms, much smaller than manual configuration, which usually needs several hours.

***Keywords*—reverse proxy chain, dynamic generation, data center**

I. Introduction

In recent years, many tech companies such as Google, Amazon, and Huawei have built their own data centers to support many kinds of cloud services. Most data centers use multi-layer network architectures and include many different types of networks, such as 10 Gigabit Ethernet networks, 25 Gigabit Ethernet networks, and InfiniBand networks. Because different networks have different interfaces and different requirements for packet size and packet rate, making it difficult to achieve compatibility on the same router. Even if the compatibility is really realized, its commercial value is very small. Therefore, the current solution is to use a proxy server equipped with multiple types of network interface cards to solve the communication problem among different networks. For example, communication between a 10GbE network and a 25GbE network requires a proxy server with a 10GbE network interface card and a 25GbE network interface card. If a request is forwarded to a destination through the help of multiple proxy servers, then the proxy servers involved constitute a proxy chain. An illustration of multiple networks in a data center is shown in Figure 1.

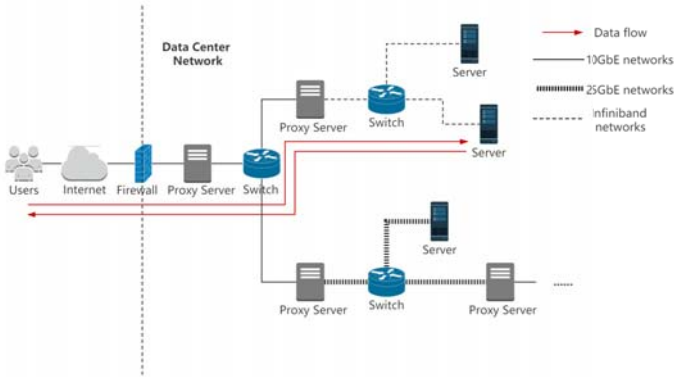

Fig. 1. An illustration of multiple networks in a data center

Traditionally, the responsibility usually falls upon network administrators to manually configure layers of reverse proxy servers. However, it is time-consuming, complicated and difficult to scale in large-scale scenarios, especially when a new application server is added to the network. Besides, the traditional static configuration practice usually leads to poor availability. In the case of a single node failure in a proxy chain, application servers which rely on this reverse proxy chain, are no longer able to provide services.

In this paper, we propose a method of dynamically generating reverse proxy chains. It can help save network administrators the trouble of manual configuration. Reverse proxy chains are dynamically generated when new applications servers are added and automatic adjustments are made in the case of node failures, to provide users with highly available services.

II. Basic concept and contribution

A. Basic concept

A forward proxy is an Internet-facing proxy used to retrieve data from a wide range of sources. A reverse proxy is a type of proxy server that retrieves resources on behalf of a client from one or more servers. These resources are then returned to the client, appearing as if they originated from the proxy server itself. The proxy chain discussed in this paper is a type of reverse proxy. We mainly study the automatic construction of reverse proxy chain in the data center.

B. Contribution

Research on proxies has received growing attention over the past decade. Many researchers have focused on the performance issues of proxy services. Reference [1] proposes a proxy server by-passing scheme for the chained HTTP proxy networks to reduce response time. Reference [2] builds proxy services on the real-time link to reduce network congestion and improve the efficiency of proxy server. There are also many cache algorithms [3] [4] [5] for proxy services. And security issues with proxy service [6] [7] [8] are also a concern.

However, most of these studies only focus on a single proxy server, without considering proxy chains. In our work, we designed a method of dynamically generating reverse proxy chains, which has the following advantages:

a. Proxy chains can make automatic adjustments based on changes in their network topology.
b. Support for proxies to serve multiple application servers at the same time.
c. Both access control policies and load balancing policies can be formulated.

III. Design and implementation

In this section, we introduce the architecture design of the proxy program first. We then demonstrate the construction process of the reverse proxy chain.

A. Architecture design

The architecture design of the proxy program is shown in Figure 2. On the left side of the dotted line are the Broadcast module, Log module and Heartbeat module involved in the construction and maintenance of reverse proxy chains. The right side of the dotted line contains the HTTP Proxy module, the Authentication module and the MySQL database. The Authentication module and MySQL database in the dashed box will only appear on top-level reverse proxy servers, which is used to formulate access control policies.

B. The process of constructing a reverse proxy chain

Before constructing a reverse proxy chain, a non-intrusive program for broadcasting is started on an application server. By non-intrusiving we mean not changing the original program of the application service. The non-intrusive program encapsulates the application server's IP address, HTTP service's port number, etc. into a broadcast data packet. The reverse proxy servers listen to these broadcasts at all times, and store the information in the broadcasts into the proxy forwarding table after filtering. Also, reverse proxy servers use the same packet data structure for broadcasting proxy information. The process is repeated until the top-level reverse proxy servers receive the proxy information. The construction of a reverse proxy chain from an application server to top-level reverse proxy servers is thus completed.

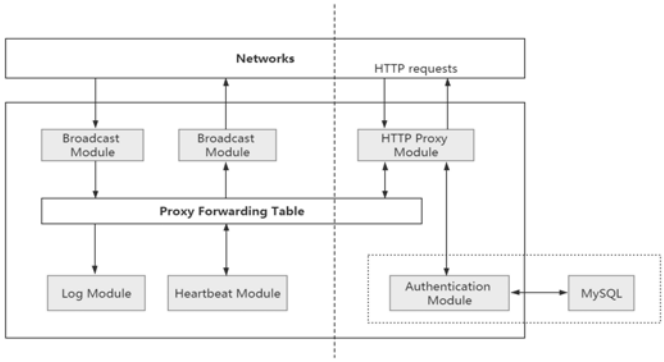

Fig. 2. Architecture design

In the initial version of the design, we encountered problems of broadcast storm and circular dependencies of two reverse proxy servers.

There is at least one network interface card on each reverse proxy server, and each network interface card will have a different IP address for connecting to different networks. If the reverse proxy server gets a broadcast data packet from one of its network interface cards, and then sends a new encapsulated data packet back to the same network interface card, a broadcast storm will occur. A broadcast storm can consume sufficient network resources so as to render the network unable to transport normal traffic [9]. Therefore, a check is performed before sending a broadcast data packet to each network interface card. If the IP address of the data source and the IP address of the network interface card are inside the same network, the broadcast data packet will not be sent to the network again.

The problem of circular dependency of two reverse proxy servers not only produces data redundancy but also prevents the heartbeat mechanism from performing correctly. Therefore, we added the Proxy_Chain field to broadcast data packets and the proxy forwarding tables. Upon receiving a broadcast data packet, a reverse proxy server will check whether the Proxy_Chain field in the broadcast data packet contains its own identity. If it does, the reverse proxy server will ignore the broadcast data packet. During the next round of broadcasting, the current reverse proxy server will add its identifier at the end of the Proxy_Chain field. In this way, the problem of circular dependencies is avoided.

Also, we have implemented a heartbeat mechanism to ensure the high availability of the system. If it is found that the Time_Stamp field of an entry in the proxy forwarding table has not been updated for more than two consecutive lifetimes

of a broadcast data packet (a total of 60s), the entry will be marked as invalid and then removed. And its goroutine for proxy service will be shut down.

IV. Performance Evaluation

The experimental environment is a Ubuntu 18.04 server with a 48-core Intel(R) Xeon(R) CPU E5-2692, and 64 gigabytes memory. We used Mininet for our software-defined network (SDN) simulation experiments, which is a network emulation orchestration system. We also deployed Ryu as our SDN controller, which is a component-based software-defined networking framework. We carried out two experiments. The first one has a full-binary-tree-like topology of 1534 nodes, including 1023 servers and 511 switches. We define the length of a reverse proxy chain as the number of servers on the reverse proxy chain, that is, the number of reverse proxy servers plus one application server, and the distance between two nodes as the length of the shortest reverse proxy chain between them.

A. A full-binary-tree-like topology of 1534 nodes

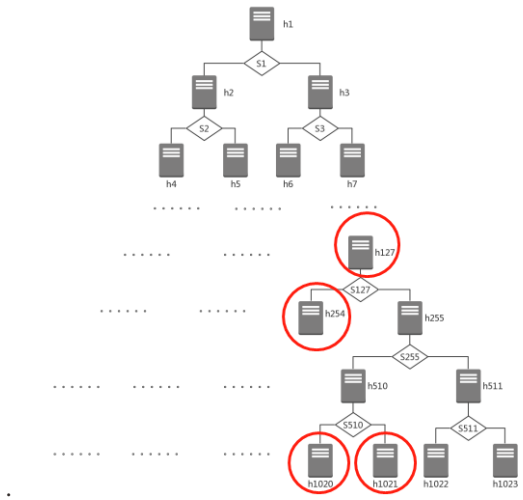

Fig. 3. A full-binary-tree-like topology illustration

The 1023 servers are nodes in a full binary tree, being connected by switches. An illustration is shown in Figure 4, the S means switch and h reverse proxy server. The root node numbered 1 is the top-level reverse proxy server. The length of the longest reverse proxy chain in this topology is 18. The bandwidth between two adjacent devices was set to 10Mbit, the packet loss rate 1%, and the link delay 30ms, 100ms, 500ms, and 1000ms, respectively each time. An application service and a non-intrusive program for broadcasting were deployed on the leaf node 1023. And a proxy program was running on the other 1022 servers. Due to the nature of the topology above: the path from one node to another without visiting a node twice is unique. Therefore, the distance from a node to an application service node is the length of the reverse proxy chain. A total of 1022 reverse proxy chains are generated. The proxy chain from node 1023 to itself is not considered. We recorded the time required to complete the construction of these 1022 reverse proxy chains. After the program for broadcasting on node 1023 was closed, the time required for all the information of the 1022 reverse proxy chains to be removed from their proxy forwarding tables was recorded. For different link delays, we calculated the average time of same-length proxy chain construction and destruction. Finally, the statistical results are drawn as two scatter plots as shown in Figures 5 and 6.

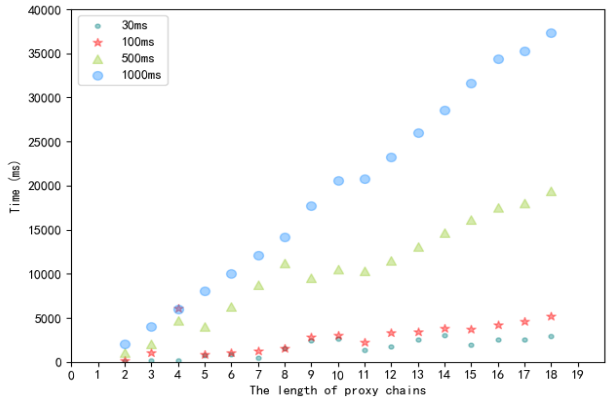

Fig. 4. The time to construct proxy chains in full-binary-tree-like topology

The abscissa of Figure 5 is the distance from a node to the application server, which is also the length of the proxy chain in the current network topology. And the ordinate is the time from the start of broadcasting to the completion of the construction of the proxy chain, in milliseconds. As is shown in Figure 5, the construction time of a proxy chain is proportional to its length, which is consistent with our expectations. However, in the 100ms delay network, the time required for the construction of a proxy chain of length 4 (6118.5ms) varies greatly, which almost coincides with the corresponding 1000ms delay point (6032.67ms). It even exceeded the construction time of proxy chains of length 18 in 100ms delay network.

As is shown in Figure 4, there are only 4 proxy chains of length 4 ending at node 1023:

$h127 \rightarrow h255 \rightarrow h511 \rightarrow h1023$
$h254 \rightarrow h255 \rightarrow h511 \rightarrow h1023$
$h1020 \rightarrow h510 \rightarrow h511 \rightarrow h1023$
$h1021 \rightarrow h510 \rightarrow h511 \rightarrow h1023$

Since the average construction time of proxy chains of length 3 does not vary much and there is no abnormality in the average construction time of proxy chains of length 5, the following two links were working correctly:

$h127 \rightarrow h255 \rightarrow h511 \rightarrow h1023$
$h254 \rightarrow h255 \rightarrow h511 \rightarrow h1023$

This means that there must be something wrong with the link from node 510 to node 1020 or 1021, which caused a great deviation in the average construction time of the two proxy chains:

$h1020 \rightarrow h510 \rightarrow h511 \rightarrow h1023$

$h1021 \rightarrow h510 \rightarrow h511 \rightarrow h1023$

After some investigation and analysis, we found that Mininet uses processes to simulate nodes and links. Due to the limitation of the memory, a few of processes are abnormally closed. Since only two links were found abnormal in more than 1,000 links, the error rate was less than 1%, so we retained the results of this experiment.

For the current network topology, the path from a reverse proxy server to an application server is unique, and because a switch connects two reverse proxy servers, the link delay needs to be multiplied by 2, so the theoretical time is:

T (time to construct all proxy chains) = longest proxy chain length * link delay * 2

In this case, it takes 37290ms, which is 37.29 seconds, in the 1000ms delay network, which is of the same order of magnitude as the theoretical value 36,000ms. The same conclusion is true for the other three levels of latency.

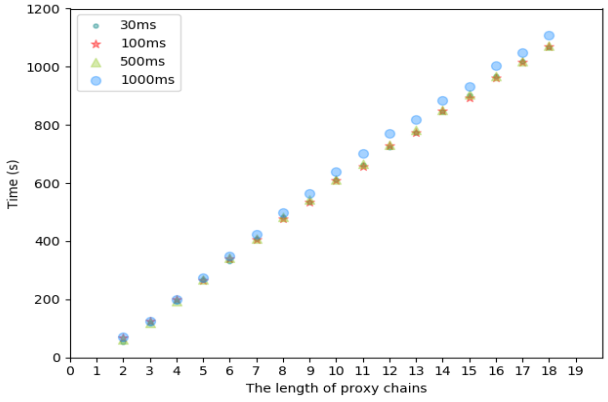

Fig. 5. The time to remove proxy chains in full-binary-tree-like topology

The abscissa of Figure 6 is the distance from a node to the application server, which is also the length of the proxy chain in the current network topology. And the ordinate is the time from the termination of broadcasting to the removal of proxy chains from the network, in seconds. It can be seen from Figure 6 that there is little difference in the time required for the proxy chains of the same length to be removed from the four different delay networks. Because of the existence of the heartbeat mechanism, for each reverse proxy server, if no new broadcast data packet is received within two consecutive lifetimes (a total of 60s), then the proxy chain is considered invalid, and broadcasting of the proxy information to higher-level reverse proxy servers is terminated. The message that the proxy chain is invalid is gradually passed from the application server to the top-level reverse proxy server, so it takes a long time to remove all the proxy chains from the network. The theoretical time is:

T (time to remove all the proxy chains from the network) = length of the longest proxy chain * 60s + length of the longest proxy chain * link delay * 2

It takes 1109820ms in a 1000ms delay network, which is about 18.5 minutes and is of the same order of magnitude as the theoretical value 18.6 minutes, which is in line with our expectations. Experiments with the other three levels of network delay setting have also reached the same conclusion. It should be additionally noted that the removal of the expired reverse proxy chains does not affect the use of other proxy chains.

In addition to the above experiments, we also tested the impact of deliberately deleting a key node on a proxy chain in the two topologies mentioned above. In the 1534-node topology, the proxy chain between any two nodes is unique. Therefore, all proxy chains containing deleted nodes can not be used, and they are gradually removed from the network. The time required for removal is proportional to the length of a proxy chain, which is consistent with our previous conclusions. Besides, the correctness of the proxy chain has also been verified, which fully meets our expectations.

V. Conclusion and Future Work

In this paper, we propose a method of dynamically generating reverse proxy chains that helps avoid the time-consuming and labor-intensive manual configuration for distributed networks in data centers. In our experiments, we set up a full-binary-tree-like topology network with 1534 nodes. We collected the time required to construct all the proxy chains and the time required to remove the proxy information of a service in a network after the services and the program for broadcasting were closed. And we verified the automatic adjustment of proxy chains after designating a faulty node.

References

[1] G. Kim and S. Lee, "A proxy server by-passing scheme for the chained http proxy networks," in *2014 International Conference on Information and Communication Technology Convergence (ICTC)*, Oct 2014, pp. 886–887.

[2] G. Du, Z. Zhang, and X. Wu, "Http proxy server based on real-time link," in *2010 International Conference on Multimedia Information Networking and Security*, Nov 2010, pp. 169–173.

[3] R. Gupta and S. Tokekar, "Preeminent pair of replacement algorithms for l1 and l2 cache for proxy server," in *2009 First Asian Himalayas International Conference on Internet*, Nov 2009, pp. 1–5.

[4] Y. Niranjan, S. Tiwari, and R. Gupta, "Average memory access time reduction in multilevel cache of proxy server," in *2013 3rd IEEE International Advance Computing Conference (IACC)*, Feb 2013, pp. 44–47.

[5] Zhenzhong Yang and Xiaojun Huang, "Dynamic configuration of reverse proxy cache based on multi-dimensional time series prediction of visit traffic," in *2016 7th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, Aug 2016, pp. 237–240.

[6] C. Lin, J. Liu, and C. Lien, "Detection method based on reverse proxy against web flooding attacks," in *2008 Eighth International Conference on Intelligent Systems Design and Applications*, vol. 3, Nov 2008, pp. 281–284.

[7] W. Yanhua, Y. Kuihe, and Z. Yun, "Research and realization of security proxy based on ssl protocol," in *2007 8th International Conference on Electronic Measurement and Instruments*, Aug 2007, pp. 2–264–2–267.

[8] Wen-Guang Long and Jian-Ping Li, "Designing secure session based on reverse proxy," in *2012 International Conference on Wavelet Active Media Technology and Information Processing (ICWAMTIP)*, Dec 2012, pp. 299–301.

[9] N. Wisitpongphan, O. K. Tonguz, J. S. Parikh, P. Mudalige, F. Bai, and V. Sadekar, "Broadcast storm mitigation techniques in vehicular ad hoc networks," *IEEE Wireless Communications*, vol. 14, no. 6, pp. 84–94, 2007.