CSDI Projects: NFV and RDMA

Yang Hong
hy.styx at gmail.com

May 5, 2015

1 Firewall

Firewall is a security network function provided by many operations systems. Firewall controls the incomming and outgoing packets according to installed rules. On Linux, you can use user-land application `iptables` to configure the firewall in kernel.

This project will build a userspace firewall to understand how firewalls work. Basically, you capture a series of packets using `tcpdump` and save them to a file. Replay the packets to your firewall configured with some rules to filter out packets with specific behaviors. This is different from the real firewall because some packets are not possible because their precedents should have been filtered out. But it gives you the basic idea of the functionality of firewalls.

1.1 Requirements

Packet Recording
Recording some sample packets is the first step you need to do. You should use tools listed below to capture packets on your system. Run the packet capture tools for 5 minutes and you should open some web pages to increase the amount of packets. Record the all the packets in a file for replaying afterwards. To help verify your firewall, your should do following operations:

- Visit http://www.sjtu.edu.cn.
- Visit at least 10 other websites.

Firewall Implementation
You should use either `libpcap` or `PF_RING` to implement your firewall application. Since you are not actually going to intercept and filter packets received or sent by OS, your firewall reads recorded packets from a dump file and emulates the filter behaviors. Your firewall should be able to implement at least the following rules:

(a) Filter out all incomming ARP packets

(b) Filter out all outgoing DNS queries. Because you are replaying packets, this operation won't affect future packets.

(c) Limit the number of outgoing TCP connections to 5. When the number of TCP connections reaches 5, drop TCP packets of future connections. When a connection terminates, one more connection is allowed to establish. You need to accurately deal with different type of TCP packets at different stages.

Hint: The last rule is a little difficult. Your firewall need track the establishment and termination of TCP connections and dynamically decide whether to drop packets or forward packets.

Evaluation

- Record packets to `dump.pcap`.
- Feed packets from `dump.pcap` to your firewall, output filtered packets to `filtered.pcap`. Remember to print essential processing decisions to demonstrate how your firewall works.

1.2 Tools

- `tcpdump` A famous tool for capturing packets and replay packets.
- `libpcap` A library that provides APIs to capture raw packets from OS network interfaces. It provides auxiliary functions to parse packet headers and can work with tcpdump and `PF_RING`. The file format of dumped packets, `*.pcap` is the de facto standard for packet capture.
  http://eecs.wsu.edu/~sshaikot/docs/lbpcap/libpcap-tutorial.pdf
- `PF_RING` A high-performance packet capturing framework that provides convenient interfaces to operate on packets. With zero-copy support, it can provide packet processing of 10 Gbps.
  http://www.ntop.org/products/pf_ring/

1.3 Hand In

- Scripts or programs to record packets.
- Firewall code
- Documents describing the design of your firewall.

2 RDMA Emulation

RDMA (Remote Direct Memory Access) is a technology that greatly simplifies application communication between physical machies. RDMA has been applied in high performance computing to reduce I/O bottleneck. This project, `librdmaemu`, uses software to emulate the basic idea of RDMA-style programming. `RDMAemu` allows two applications running on two different physical machines to access the memory region of the other. You need to emulate the operations of RDMA verbs and the programming style of RDMA programs rather than simply passing data through sockets.

2.1 Materials

The following articles provides some insights on RDMA programming.

- http://www.csm.ornl.gov/workshops/openshmem2013/documents/presentations_and_tutorials/Tutorials/Verbs%20programming%20tutorial-final.pdf
- https://github.com/tarickb/the-geek-in-the-corner/tree/master/01_basic-client-server

2.2 Requirements

Your goal is to design a library, `librdmaemu`, which provides APIs similar to RDMA library and accomplishes remote memory access in the background. A real hardware NIC capable of RDMA does not need remote CPU to execute code in order to read/write remote memory. For `librdmaemu`, it does execute code in a separate thread to implement the memory transfer function. You should use the examples provided in Github code as the start point. However that code only provides basic functionality of "Post Send Request". You should extend the test programs to demonstrate that `librdmaemu`'s behavior is close to real RDMA hardware.

Server:

1. Create event channel
2. Bind to an address
3. Create listener on port/address
4. Wait for connection request
5. Create shared memory region, completion queue, send-receive queue pair

6. Accept the request
7. Wait for connection to be established
8. Post operations

Client:

1. Create event channel
2. Create shared memory region, completion queue, send-receive queue pair
3. Connect server
4. Wait for connection to be established
5. Post operations

You library should implement and make use of the essential parts:

- shared memory region
- event mechanism based on completion queue
- send-receive queue pair
- post operations

2.3 Hand In

- Test server and client programs that show how to use RDMA.
- Library code.
- Documents describing your library design.