Development of a multi-threaded NAT

IK2200: Communication Systems Design (CSD) - Project 4

Introduction

A **NAT** (**N**etwork **A**ddress **T**ranslator) allows rewriting the address of a network packet. You probably have one NAT at home, in your home router. This box does NAT to provide Internet connectivity to all your home devices while using a single IP address assigned by your **ISP** (**I**nternet **S**ervice **P**rovider). All your devices have a local IP address. When one of them sends a packet towards the Internet, your home router rewrites the source address to your unique Internet IP address given by your ISP. When a response comes back, it is rewritten the other way around, back to the original address.

Carrier-grade NATs follows the same ideas but are deployed by ISP in their internal network to translate addresses of thousands of devices to a fewer amount of IP addresses. If you have a 4G connection, chances are you are behind a NAT, because the IPv4 address space does not leave enough addresses for the billions of mobile devices to be connected to the Internet soon.

The goal of this project is to study different methods to build a carrier-grade NAT on a single host with multiple CPU cores.

Required knowledge: C++, Linux

High-level task overview

1. Testbed deployment and study of single-core NAT
    a. Arrange a testbed using virtual machines
    b. Deploy a simple single-core NAT using FastClick, an extension of the Click Modular Router
    c. Measurement of bandwidth and latency
    d. Profiling of the NAT to find performance bottlenecks
2. Implement multiple multi-core scaling techniques
    a. Implement scaling of the NAT in FastClick using a global lock around the flow table
    b. Duplicate the NAT per-core using source port ranges to dispatch packets to the right core in software
    c. Per-core duplication using hardware dispatching instead of software to dispatch packets to the right core
3. Method from the literature
    a. Conduct a literature review about methods to scale a NAT using multiple CPU cores
    b. Implement a method given by the literature review
    c. Final review

Task 1

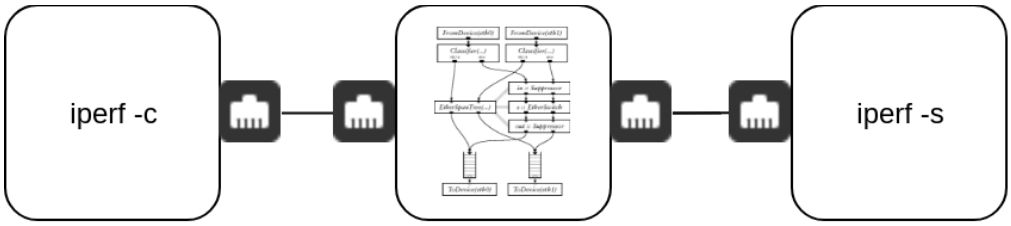
In this task, you will deploy a single-core NAT using FastClick[1], an high-speed extension of the Click Modular Router[2]. You should start by reading [2] then [1]. The FastClick general tutorial[3] will guide you with the baby steps to learn FastClick, while the FastClick DPDK Tutorial[4] will guide you in installing DPDK to allow high-speed packet processing. For this work, you must use DPDK.

1.1 Testbed deployment

You will need to build a setup of 3 machines. One will generate traffic towards a second machine that will run the NAT. But at this point, make **it simply forward packets using FastClick**. The NAT machine will in turn be connected to a third machine which will act as a sink.

1.1.1 Using IPerf

Iperf is a tool that allows computing the bandwidth between one instance running as a client, and an instance running as a server.



1.1.2 Using FastClick to generate & echo UDP flows

Iperf is limited in term of the number of flows it can support. Iperf is not built to repeat multiple requests. FastClick itself is actually very well suited to generate a very large amount of UDP flows. To simulate millions of clients, you need to stress the NAT with million of flows.

---

[1] Barbette, T., Soldani, C., & Mathy, L. (2015, May). Fast userspace packet processing. In *Architectures for Networking and Communications Systems (ANCS), 2015 ACM/IEEE Symposium on* (pp. 5-16). IEEE. http://hdl.handle.net/2268/181954

[2] Kohler, E., Morris, R., Chen, B., Jannotti, J., & Kaashoek, M. F. (2000). The Click modular router. *ACM Transactions on Computer Systems (TOCS)*, *18*(3), 263-297. https://dl.acm.org/citation.cfm?doid=354871.354874

[3] https://github.com/tbarbette/fastclick/wiki/Tutorial

[4] https://github.com/tbarbette/fastclick/wiki/High-speed-I-O#dpdk

How to create a UDP generator and a UDP echo with FastClick is discussed in the Packet generation[5] wiki page.

Completion criteria

- ❏ Your report explains the Click and FastClick components and shows that you understand what you're doing.
- ❏ Ensure the IPerf flow from generator to sink goes through FastClick and vice versa.
- ❏ Use the Print element of FastClick to ensure that packets flow as expected.

1.2 Deploy a simple single-core NAT using FastClick

Deploy a NAT using FastClick. We suggest starting with the UDPRewriter element[6] for the UDP case before moving to support the TCPRewriter[7] element for the IPerf test case.
You should use the IPClassifier[8] element to dispatch TCP and UDP traffic to the right element. This will allow you to have a one-for-all configuration.

Completion criteria

- ❏ The packets on sink appear to have been rewritten.
- ❏ All tests cases of 1.1 still work.

1.3 Throughput and latency characterization

Compare the throughput and latency with and without the NAT using the multiple tools. Your toolset must be able to repeat experiment multiple times and show the mean and standard deviation of the individual tests results. Show the impact of forwarding less or more flows on throughput and latency.
The chosen plot type should also display the "tail latency", that is the latency for the worst 1% of the flows.

Completion criteria

- ❏ You produced graphs comparing both tests cases, with and without NAT under a varying amount of flows and flow size.
- ❏ Your report explains the performance change you see.

---

[5] https://github.com/tbarbette/fastclick/wiki/FastClick-for-packet-generation
[6] https://github.com/tbarbette/fastclick/wiki/UDPRewriter
[7] https://github.com/tbarbette/fastclick/wiki/TCPRewriter
[8] https://github.com/tbarbette/fastclick/wiki/IPClassifier

1.4 Profiling

Use Linux's "perf" tool[9] to find where the performance is lost, bottlenecks, and potential improvements. Before moving to the next task, you must make sure the NAT is indeed the bottleneck, as scaling the NAT would not improve performance if it's not the bottleneck. You may give more cores to the machines running the generator and/or the sink to be sure that the NAT machine is the bottleneck.

Completion criteria

- ❏ You proof in your report the observations of 1.3 with the performance counters observed with perf
- ❏ You list the biggest hit points in your report

[9] https://jvns.ca/blog/2016/02/24/perf-top-my-new-best-friend/

Task 2

The goal of this task is to scale the NAT using multiple cores.

Completion criteria

- ❑ For each method below, repeat the experiments of 1.3 and explain the results you see with the methods developed in 1.4.

2.1 Global locked flow table

Use a lock around the flow table, allowing to protect the shared states between cores.
When you use FastClick, the FromDPDKDevice with the "SCALE parallel" parameter will automatically scale across all cores. Therefore your NAT will be traversed by multiple cores as soon as you run FastClick with multiple cores. You should try first to scale using one of the Spinlock elements, and then a finer-grained lock in the Rewriter elements.

2.2 Per-core duplication, with software classification

Fully duplicate the NAT element per-core, using a CPUSwitch element. In this configuration the problem is that packets that come back from the sink must be piped to the right element that originally handled the packet. One way to do so is by choosing carefully the ports that each per-core NAT will use, and classify (with IPClassifier) those ports to forward returning packets to the right core (using the Pipeliner element) that contains the corresponding flow in the flow tables.

2.3 Per-core duplication, with hardware classification

This solution is identical to 2.2, but you will use hardware capabilities of the NIC to classify packets and directly direct them towards the right hardware queue. Simulation of hardware dispatching, or using a real classification capable NIC will be discussed in time.

Task 3

The goal of task 3 is to go beyond the given "simple" scaling examples and draw a full conclusion from your experience. It is unlikely that you'll beat the hardware-based scaling method, but you may find better techniques for purely software-based scaling.

3.1 Literature review

You will conduct a literature review about scaling NAT using multiple CPU cores. Google scholar is a good start.

Completion criteria

- ❏ Your review cites multiple papers and highlight the best contributions regarding scalable NATs. You may find papers for very similar functions, such as load balancing, that describe techniques applicable to NATs.

3.2 Implement one method from the literature review

Implement one (or more if you wish) methods from the literature review that you think will perform best.

3.3 Final review

Compare the difference in term of performance against the other methods as in 1.2.

Completion criteria

- ❏ Provide a/multiple final graph(s) comparing all methods.
- ❏ Review clearly the limitation of each method. Is it possible to add dynamically one more core? Does one method take more memory than another? (...?)
- ❏ It is clear to the reader of your report which method he or she should use according to his or her use case
- ❏ Your code proves to be sufficiently ready and clean that a pull request to the main FastClick repository for a multi-thread safe version of UDPRewriter and TCPRewriter has been accepted.
- ❏ Optionally, you may add the others methods you developed in the FastClick research package so everybody can re-execute your experiments and come to the same conclusions as you