

Design and Implementation of Practical Software Middleboxes

Summary

University Politehnica of Bucharest



Submitted for the degree of
Doctor of Philosophy

Vladimir Olteanu

Supervisor: Valentin Cristea

Keywords

- networking
- datacenter
- middlebox
- protocol

Summary

This thesis discusses the design of horizontally scalable software middleboxes.

Middleboxes are machines that process network packets, performing tasks as simple as basic filtering, or as complex as detecting leakage of sensitive information. They are a staple of today's Internet.

Traditionally, middleboxes were monolithic hardware appliances. Upgrading them either in terms of performance or functionality was typically costly, as it meant outright replacing the machine with a different model. Moreover, they had to cope with peak hour traffic; during off-peak hours the extra capacity was essentially wasted.

To address these issues, a new field of research emerged, called Network Function Virtualization. The key proposal is to move traffic processing from dedicated appliances to software running in a distributed manner on commodity hardware. Such middleboxes can scale out or in on the fly as the demand for capacity increases or decreases: extra capacity can be conjured up by adding more machines, and then removed by powering down some of them.

The most basic obstacle to scaling out middleboxes is that they may hold state, both per-flow and shared across multiple flows. Such state is key to the correct handling of the traffic; were it to become unavailable, even temporarily, all corresponding flows may experience disruptions. Therefore, scaling out not only entails redirecting a portion of the incoming traffic to another machine, but also migrating all relevant state to said machine.

There have been efforts to find a general framework that enables near-seamless scale-up and scale-down events, but they tend to incur a rather steep performance cost. The subtext of this thesis is that we argue against such generalized solutions; to that end, we design two separate middleboxes, and use tailor-made solutions to solve their scalability issues.

The first middlebox we discuss is a large scale NAT. It is a rather straightforward middlebox to implement, that holds both per-flow and per external IP. Our version features a novel state migration algorithm aimed at incurring a minimal performance penalty. Here we show that by simply risking reordering some packets while keeping all other guarantees intact, the performance cost of scale-out or scale-in events is minimal. Occasional packet reorderings are generally something that protocols at Layer 4 and up can cope with easily.

The second middlebox we discuss is Beamer, a stateless cloud-scale load-balancer, that also works with MPTCP. Stateful load-balancers remember associations between connections and back-end servers. This, in essence, attempts to duplicate information kept by the servers themselves in their connection tables. By not holding any per-flow state, Beamer muxes are simple to implement both in software and hardware, and avoid many of the pitfalls of scaling up and down. They can

easily handle SYN floods, something stateful designs struggle with.

Finally, we will discuss something that is a middlebox only in the loose sense of the word: we have designed version 6 of the decades-old SOCKS protocol, and built a proxy around it. Version 6 addresses many of the shortcomings of version 5: its RTT overhead in most cases is zero (or even negative!), it offers DNS from the proxy's vantage point, and it plays well with the new features introduced in TLS 1.3. Since the proxy is technically a service running on top of TCP, large SOCKSv6 deployments can be made using Beamer.

Contents

1	Introduction	3
2	Background	5
2.1	Middleboxes in the Internet architecture	5
2.2	OpenFlow and SDN	6
2.3	Multipath TCP	7
3	Principles in building scalable network processing	9
3.1	Status quo	12
3.2	Thesis Contributions	13
3.3	Publications	16
4	Carrier-grade NAT	17
4.1	Introduction and Motivation	17
4.2	Designing a Carrier-Grade NAT	19
4.3	Implementation	22
4.4	Evaluation	24
4.5	Lessons Learned	29
5	Beamer	31
5.1	Introduction	31
5.2	Background	32
5.3	Limits of stateful load balancing	34
5.4	Beamer: stateless load-balancing	36
5.4.1	Stable hashing	37
5.4.2	Daisy chaining	39
5.4.3	Mux data plane algorithm	40
5.5	Handling Multipath TCP	41
5.5.1	MPLB's approach	42
5.5.2	Beamer's approach	46
5.6	Beamer control plane	47
5.7	Defragmentation	49
5.8	Implementation	51

5.9	Evaluation	52
5.9.1	Micro-benchmarks	52
5.9.2	Scalability and robustness	55
5.9.3	Load balancing HTTP over MPTCP	59
5.9.4	Controller scalability	61
5.9.5	Stable hashing	62
5.9.6	Defragmentation	62
5.10	Conclusions	63
6	SOCKS Protocol Version 6	65
6.1	Introduction	65
6.2	Mode of operation	66
6.3	Requests	68
6.4	Version Mismatch Replies	69
6.5	Authentication Replies	70
6.6	Operation Replies	71
6.6.1	Handling CONNECT	72
6.6.2	Handling BIND	72
6.6.3	Handling UDP ASSOCIATE	73
6.7	SOCKS Options	77
6.7.1	Stack options	77
6.7.2	Authentication Data options	84
6.7.3	Authentication Method options	84
6.7.4	Session options	86
6.7.5	Idempotence options	88
6.8	Username/Password Authentication	90
6.9	SSL Authentication	92
6.10	TCP Fast Open on the Client-Proxy Leg	92
6.11	False Starts	92
6.12	DNS provided by SOCKS	92
6.13	Security Considerations	93
6.13.1	Large requests	93
6.13.2	Replay attacks	94
6.13.3	Resource exhaustion	94
6.14	Privacy Considerations	94
6.15	SOCKS timing	94
6.16	Implementation	96
6.17	Evaluation	96
6.18	Conclusion	97
7	Related work	99
8	Conclusions	101