Contents

Acknowledgements **v**

Contents **vii**

1 Introduction **1**

2 Working at Cloudflare **3**

2.1 Economic Context . . . 3
2.1.1 Cloudflare . . . 3
2.1.2 Denial of Service Attack . . . 5
2.1.3 Cloudflare Denial of Service Protection Product . . . 5
2.2 Human Context . . . 6
2.3 Scientific Context . . . 6
2.3.1 System: eBPF . . . 6
2.3.2 Networking: XDP . . . 7
2.3.3 Challenges of Benchmarking . . . 7
2.3.4 Data . . . 8
2.3.5 Hardware . . . 8
2.3.6 Software . . . 9

3 Problem Statement **11**

3.1 Problem Analysis . . . 11
3.1.1 Existing System . . . 11
3.2 Aims . . . 14
3.2.1 Problem Statement . . . 14
3.2.2 Requirements Specification . . . 14

4 State of the Art **17**

4.1 eBPF & XDP . . . 17
4.1.1 eBPF . . . 17
4.1.2 Hook Points . . . 19

4.1.3 Improvements in eBPF . . . 22
4.2 Denial of Service Attacks . . . 23
4.2.1 Network Level Attacks . . . 23
4.2.2 Application Level Attacks . . . 24
4.3 Benchmark . . . 24
4.3.1 Aims and Challenges . . . 24
4.3.2 Statistical Tests . . . 24
4.3.3 Tooling . . . 26

5 Preparatory Work and Benchmarks Creation 29
5.1 Method . . . 29
5.2 Writing a Functional Specification . . . 30
5.2.1 Current Design . . . 30
5.2.2 Solutions Considered . . . 30
5.2.3 Measurements . . . 31
5.2.4 Adoption Criteria . . . 32
5.3 Writing Benchmarks . . . 33
5.3.1 Benchmark 1 . . . 33
5.3.2 Benchmark 2 . . . 34
5.4 Running Benchmark . . . 36
5.4.1 Hardware . . . 36
5.4.2 Run Time Observations . . . 37
5.4.3 Benchmark Results Analysis . . . 37

6 Improvements and Alternative Implementations Considered 39
6.1 Kernel Patch Tail-Call . . . 39
6.1.1 Context . . . 39
6.1.2 Main Results . . . 39
6.1.3 Benchmark 1 . . . 40
6.1.4 Benchmark 2 . . . 41
6.2 Dispatcher . . . 43
6.2.1 Context . . . 44
6.2.2 Main Results . . . 45
6.2.3 Benchmark 1 . . . 45
6.2.4 Benchmark 2 . . . 46
6.3 Switch eBPF Programs Dynamically at Run Time . . . 47

6.4 Merge Rulesets . . . 48
6.5 Adopted Solution . . . 48
6.5.1 Refining the Dispatcher . . . 49

7 Impact and Future of the Project **50**
7.1 Economic Impact and Knowledge Acquired . . . 50
7.1.1 Lower Infrastructure Cost . . . 50
7.1.2 External Communication . . . 50
7.2 Future and Improvements . . . 51

8 Project Management **53**
8.1 Working Methods . . . 53
8.2 Gantt Charts . . . 53

9 Conclusion **55**

Bibliography **57**

List of Figures **65**

List of Tables **67**

Listings **69**

Glossary **71**

Acronyms **72**

Appendices **73**

A Organizational Chart **74**

B XDP Driver Support **75**

C Functional Specification Ideas **77**

D Kernel Patch Tail Call: Benchmark Results **78**
D.1 Benchmark 1 . . . 78
D.2 Benchmark 2 . . . 81
D.2.1 CPU . . . 81

D.2.2 Throughput 84

E Dispatcher: Benchmark Results **87**

E.1 Benchmark 1 87

E.2 Benchmark 2 87

E.2.1 CPU 87

E.2.2 Throughput 87

Résumé **89**

Abstract **89**

Résumé

Mon stage s'est déroulé chez Cloudflare, une entreprise améliorant la sécurité et la performance des applications Internet. L'un de ses produits est une protection contre les attaques par déni de service (DoS). L4Drop, un des composants logiciels de cette protection, est basé sur des règles de rejets de paquets entrant. L4Drop utilise eBPF (extended Berkeley Packet Filter), un jeu d'instructions pour une machine virtuelle dans le noyau Linux. Ainsi, les différentes règles sont placées dans différents programmes et ceux-ci sont chaînés par des instructions appelées « tail call ». En effet, si une règle n'est pas vérifiée, on saute à la suivante. Le sujet de stage est de mesurer le coût d'un tail call et d'optimiser l'agencement des programmes eBPF de L4Drop. La première étape est de créer des benchmarks pour mieux comprendre les performances de ce programme. La deuxième étape consiste à comparer les performances de versions modifiées du programme L4Drop à la version de base.
Un premier benchmark exploite l'infrastructure de test des programmes eBPF du noyau. Un second simule du trafic tout en mesurant le temps d'exécution des programmes eBPF. Ainsi, on montre que le coût d'un tail call entre deux programmes passe de 25 ns à 5 ns avec les récentes optimisations du noyau. Ceci a été présenté à la Linux Plumbers Conference 2020. Les améliorations d'implémentation réalisées permettent de gagner 50 secondes de temps de calcul sur l'infrastructure de Cloudflare, par seconde de pique d'attaque DoS. Il en résulte des besoins de matériels réduits.

Mots-clés : eBPF, XDP, réseaux, Linux

Abstract

My internship took place at Cloudflare, a company whose mission is to enhance the security and the speed of Internet applications. One of its products is a Denial of Service (DoS) attack protection. L4Drop, one of the software components of this protection, is based on rules to drop incoming packets. L4Drop uses eBPF (extended Berkeley Packet Filter), an instruction set for a virtual machine in the Linux kernel. Thus, the different rules are placed in different programs and these are chained by instructions called "tail call". Indeed, if a rule is not verified, we skip to the next one. The subject of the internship is to measure the cost of a tail call and to optimize the layout of L4Drop's eBPF programs. The first step is to create benchmarks to better understand the performance of this program. The second step is to compare the performance of modified versions of the L4Drop program to the baseline.
A first benchmark leverages the testing infrastructure for eBPF programs from the kernel and a second one simulates traffic while recording metrics on the execution times. The cost of a tail call is shown to drop from 25 ns to 5 ns thanks to recent improvements in the kernel. This work was presented at the Linux Plumbers Conference 2020. The implementation improvements carried out during this internship result in a gain of 50 seconds of computation time across Cloudflare's infrastructure, per seconds of peak DoS attack; translating into reduced hardware requirements.

Keywords: eBPF, XDP, networks, Linux