University of Pretoria
Software Engineering - COS 301

Defendr Specification

Dark nITes
24 May 2019

Team members:

MI(Muhammed) Carrim	R(Ruslynn) Appana	SNL(Sisa) Khoza	CP(Christiaan) Opperman	J(Jeandre) Botha

Contents

1 Introduction

1.1 Purpose

The intent of this software requirements specification document is to provide the requirements and implementation plan for the project named ***Defendr***. This document serves to clarify and communicate all stakeholder's understandings and expectations of ***Defendr***.

This document is written for the perusal of the aforementioned stakeholders: our client and customer Advance, including the COS 301 module and Computer Science department of the University or Pretoria as clients too. The final audience member is the development team itself, Dark nITes.

1.2 Scope

Defendr is to be a blackbox implementation of a DoS protection service, as well as a network load-balancer for various back-end applications (henceforth called service collectively). The service is to be situated between the client and server; request from the client are to pass through the service, dropping/blacklisting offending packets/IPs. The service should employ direct server return; responses from the server are to be sent directly back to the client, and not routed back via the service.

Packets that are permitted to pass through DoS protection will then be subject to the load-balancer. Various load-balancing pools with multiple instances of a back-end situated in them. According to the algorithm that is managing the particular pool's load-balancing, packets will get passed to the intended back-end instance.

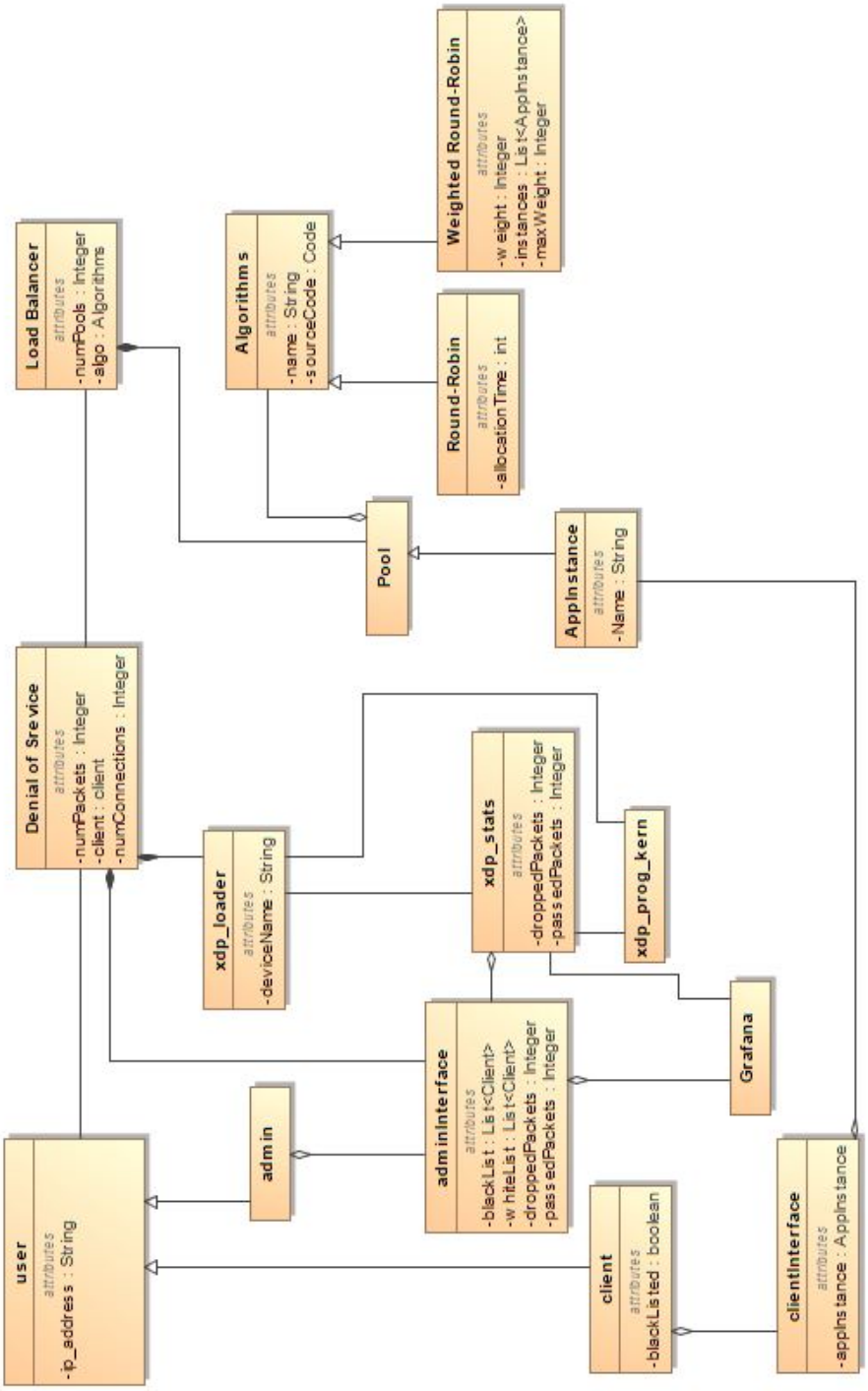
The service will measure the validity of request packets sent to a protected service by two criteria: # of packets per second, and # of connections. The limitations will be specified by the owner of the back-end being protected.

The service will also comprise of two user interfaces; an administration interface, and client interface. The interface will provide access to metrics comprising of:

- Current servers being protected
- Current status of the server, i.e. total # of packets, # of packets being let through (success rate), # of packets being dropped (failure rate)
- A heat-map that displays the geo-location of the origin of client request to protected back-ends
- List of blacklisted IPs
- Internal overhead

in a raw and graphical data format

1.3 Domain Modal



1.4 Definitions, Acronyms, and Abbreviations

Term	Definition
Blackbox	A method of software testing that examines the functionality of an application without peering into its internal structures or workings.
DOS	Disk Operating System.
DoS	Denial of Service.
DDoS	Distributed denial of service.
Load balancer	A subset of the service that will distribute network traffic to various instance of an application, as determined by the current governing algorithm.
Client	The originating device from which a request is received.
IPs	Internet Protocol.
Packets	The units of data that are being transmitted from a client to a protected application.
XDP	eXpress Data Path.
eBPF	extended Berkley Packet Filter.
Prometheus	A monitoring system that has a time-series metrics database and ways to query said metrics.
Grafana	A toolkit that presents data in a graphical form. Can be used in conjunction with Prometheus to graph the service's metrics.

2 User Characteristics

2.1 Client

The client is the primary user of the system. Their main focus is the protection of the integrity of their application. Their actions will only include the sending of packets and receiving of responses. By using this system, they will be able to protect the integrity of their application and experience an increase to throughput by virtue of DoS protection and network load-balancing. The client will use the front-facing segment of the system. These users cannot cause much detriment to the system as their interaction will strictly be limited to interacting with the front-end interface. As such, they are expected to have no more knowledge other than how to use the (protected and balanced) application. These users are required to have some form of computing device which will be able to use the front-end system as well as a connection to the internet.

2.2 Administrator

Administrators will have the duties of installing components of the system and maintaining its health and performance. As these decisions will determine the success of the service as well as the level of security, they will require a level of skill that ensures this .These users would be expected to have experience with networking and some degree of software development and maintenance. These users will be able to make changes to the system that are integral to its running, e.g. manually blacklisting IPs/IP ranges, adjusting packet filtering rules or removing services from protected pools.

3 Functional Requirements

3.1 DoS Protection

- R1.1. The subsystem must be able to detect and mitigate a DoS attack by dropping the offending packets from the source IP or IP ranges, and allow access after an exponential timeout.
- R1.2. The subsystem must provide functionality to gather metrics. These metrics packet success/failure rates, total # of packets per pool/back-end.
- R1.3. The subsystem rules that determine an incipient DoS should be able to be manually altered.

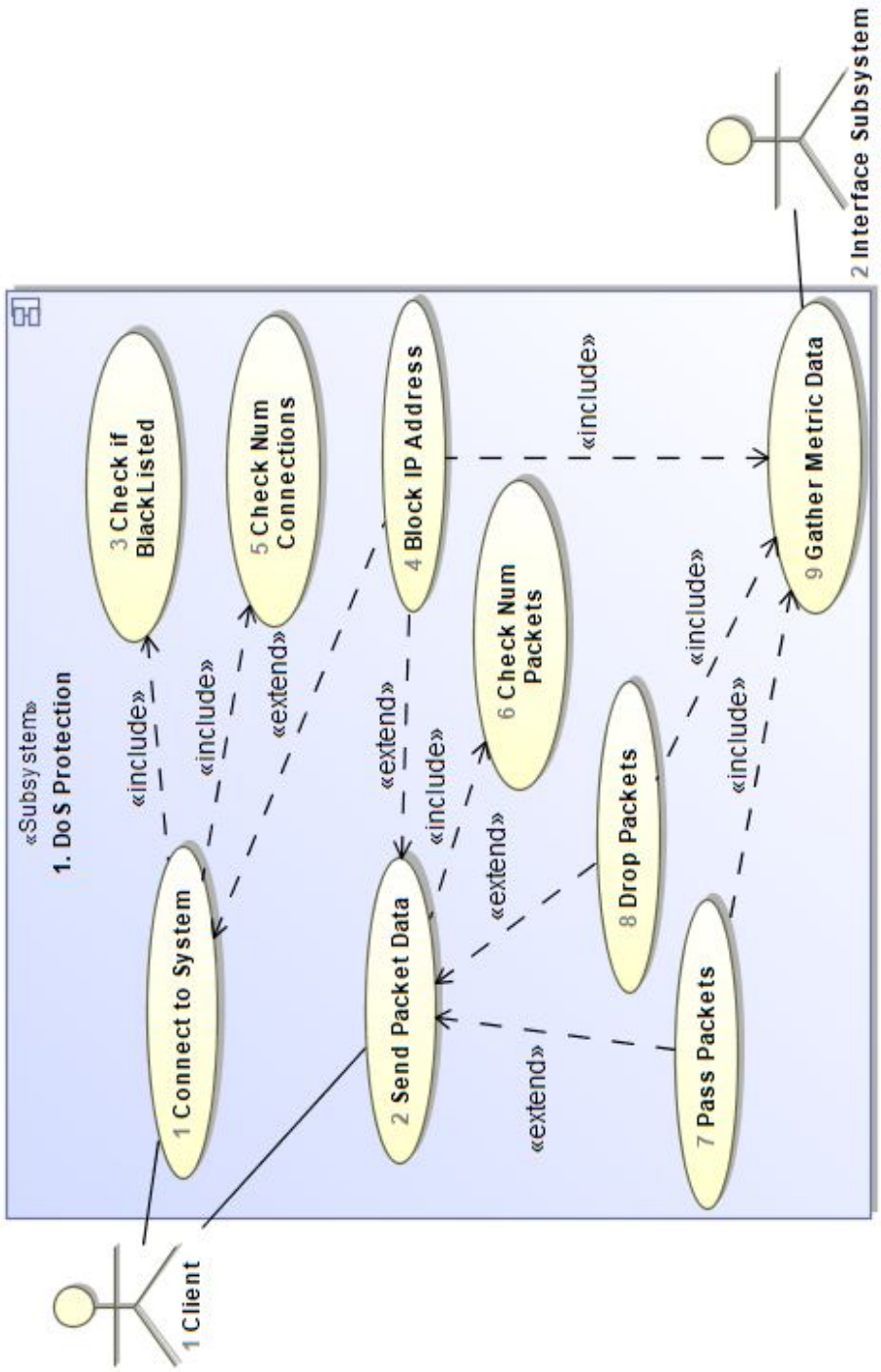
3.2 Load-Balancer

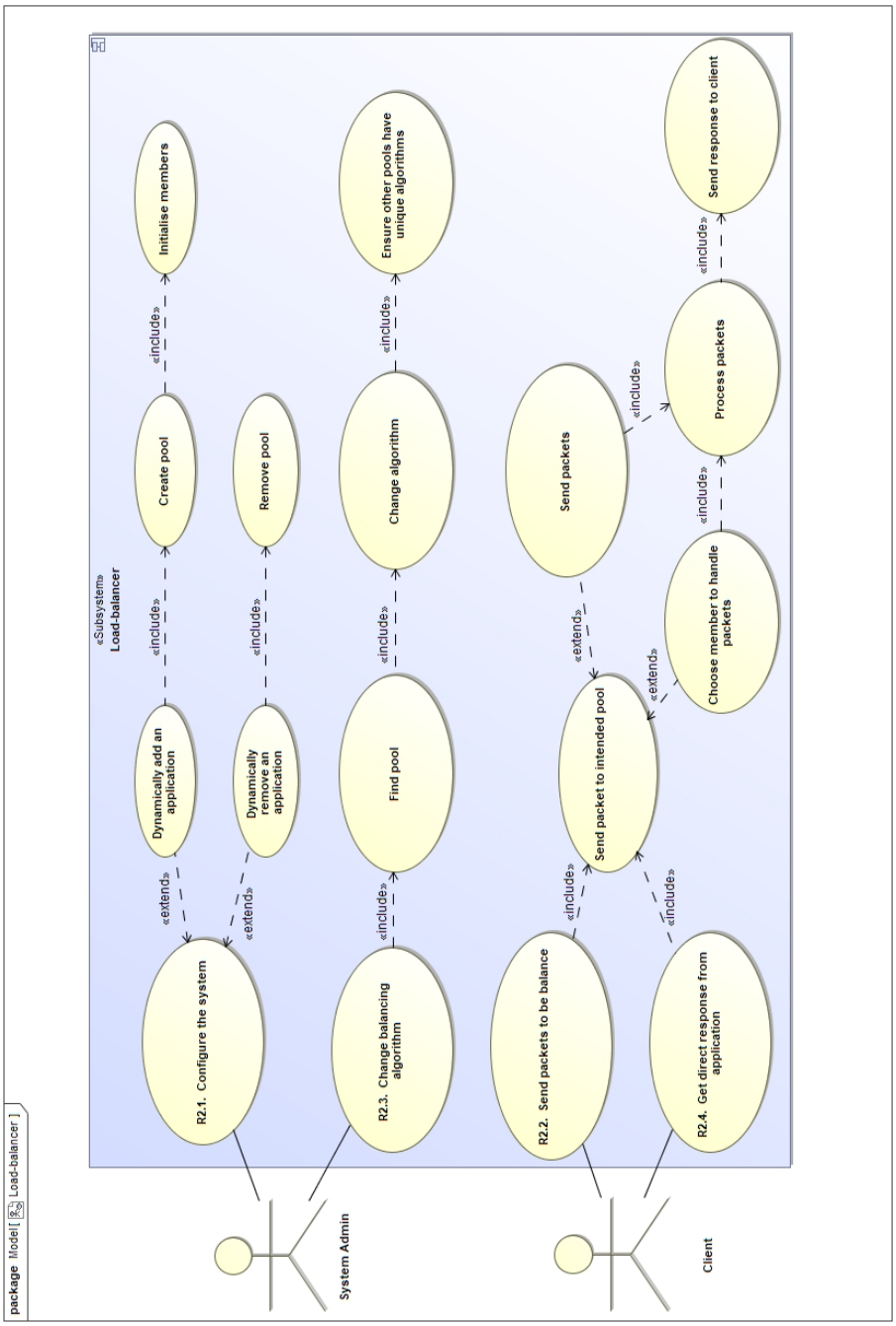
- R2.1. The subsystem needs to be configurable; applications/back-ends should be able to be dynamically added/removed
- R2.2. The subsystem must have multiple load-balancing pools, where pools are defined by the back-ends. Members of each pool are instances of the back-end that are to be load-balanced.
- R2.3. The subsystem must support multiple load-balancing algorithms, of which Round Robin and Weighted Round Robin must be included, different per pool. These algorithms should be changeable on-the-fly. Network load anomaly detection, with an option of prediction, should also be included.
- R2.4. The method of request response should be via a direct server response to the requesting client, that is responses should not return via the service.

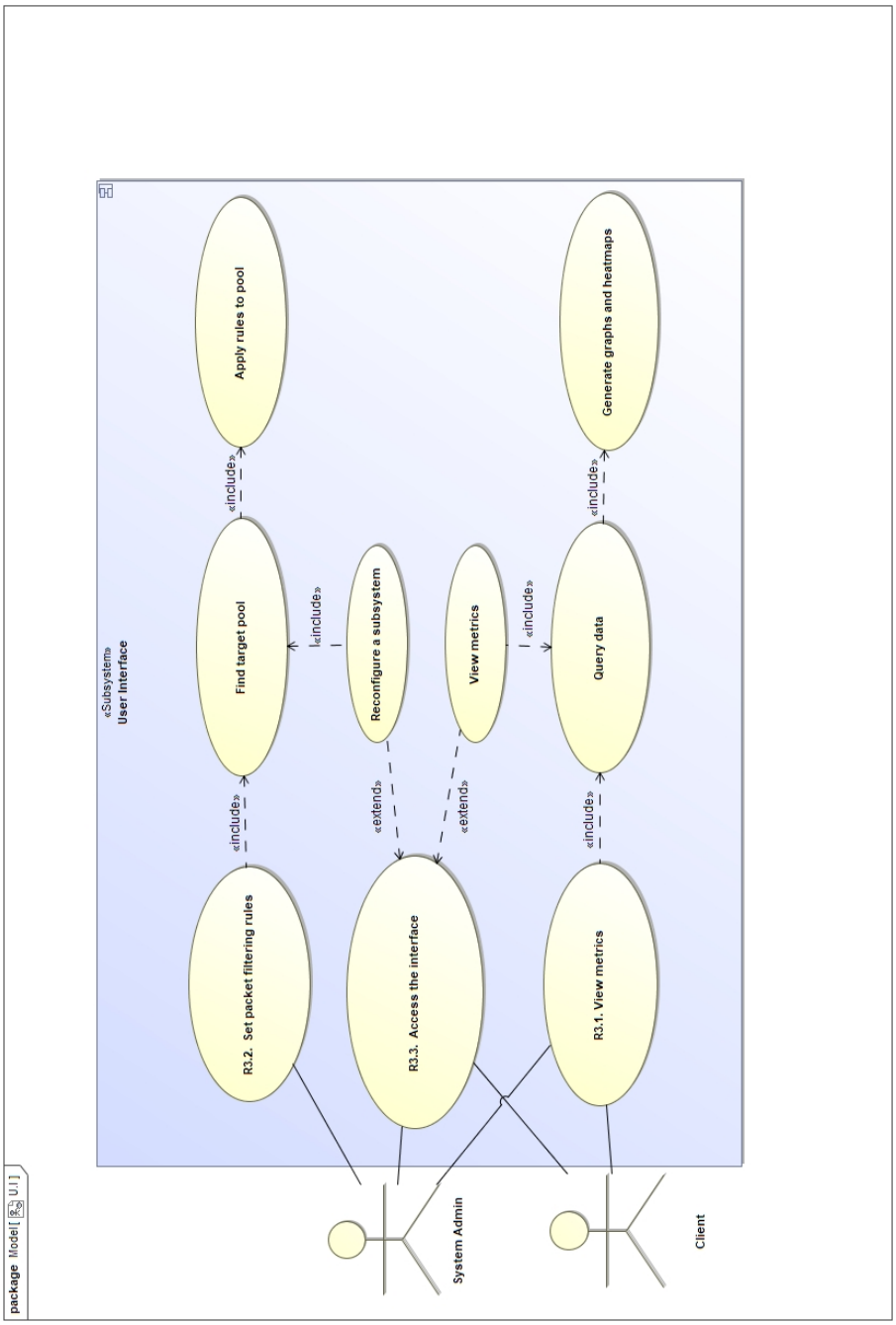
3.3 User Interface

- R3.1. The interface should show the metrics in a graphical medium, i.e. graphs and heat-maps.
- R3.2. The interface should also provide a means of manually configuring the service, e.g. blacklisting/white-listing IPs/IP ranges, defining rules that govern network traffic
- R3.3. The interface must be accessible from anywhere, and not from just a specific machine, i.e. the interface must be hosted somewhere accessible, but for demo purposes localhost will do

3.4 Use Case Diagrams




package Model [Load-balancer]
«Subsystem»
Load-balancer
R2.1. Configure the system
R2.3. Change balancing algorithm
R2.2. Send packets to be balance
R2.4. Get direct response from application
Dynamically add an application
Dynamically remove an application
Find pool
Send packet to intended pool
Choose member to handle packets
Create pool
Remove pool
Change algorithm
Send packets
Process packets
Initialise members
Ensure other pools have unique algorithms
Send response to client
«extend»
«include»
System Admin
Client


package Model [U.I]
«Subsystem»
User Interface
Apply rules to pool
Find target pool
Reconfigure a subsystem
View metrics
Query data
Generate graphs and heatmaps
R3.2. Set packet filtering rules
R3.3. Access the interface
R3.1. View metrics
«include»
«include»
«include»
«include»
«include»
«include»
«extend»
«extend»
System Admin
Client

4 Quality Requirements

4.1 Performance

The system must be able to perform at high speed and have the ability to handle great volumes, i.e the same number of packages that the application which it is protecting. In other words, the system should have the same performance capabilities as the services it is protecting so that it will not cause a bottleneck. This can be measured by looking at the drop rate and number of packages.

The step by step actions taken by the system will be logged to ensure the performance of the system. This includes the time when a packet arrives, the IP address, the location the IP address originates from and as well as the category of threat placed for the IP address. Having this information logged will ensure the performance of the system if any tests had to be done to ensure the validity of the systems work.

4.2 Security

The security of the system has to be high, since one of the main purposes of the software is to protect the services against DOS attacks. The software runs on the kernel which will expose the client to other threats if the security isn't high enough. Role based access control is used for the interface which allows for modification of the systems control values. This ensures security for the system by not allowing incorrect IP addresses to be blacklisted, etc by any unauthorized entities.

4.3 Availability

The system should be available to any application that it needs to protect. Seeing as the system also has an interface in the form of a program, which allows for the modification of the system by allowing it to manually blacklist IPs/IP ranges, adjust packet filtering rules or remove services from protected pools, there will be a possibility that the program will encounter challenges seeing that it is fallible. Thus we have come to an agreement with the client, that the system can be expected to have 99.5% up-time.

4.4 Maintainability

The system maintenance can falls under two distinct categories, that being the upkeep of the code and the addition and removal of services which have to be protected.

- Upkeep of code: The software must be able to be changed so that the software as a whole works on the latest version of Linux. The code also needs to be written in such a manner as to ensure that any developer can understand the code as well as be able to pick up at any point and make changes to the system. The developers of the Defendr system at Dark nITes all follow the practices set out in their coding standard document. This document is based on the current and most used practices of coding in the world and will ensure consistency throughout all code for the system.

- Addition and removal of services: The user should be able to add and remove services as they see fit through the use of a user interface. This can be clearly seen and tested in the

interface as any changes made will be updated and shown to the user. This will allow for example an IP address which was previously blacklisted to be allowed again. The IP address will no longer show up in the blacklisted section and one can monitor the packets arriving from the address for testing purposes to ensure the IP address is no longer blacklisted.

4.5 Scalability

The system should be able to scale to the size of the server which it is protecting, without requiring additional changes to be made to the basic structure of the system. It should also be able to accommodate a different number of load balancing pools.

4.6 Cost

All the technologies that are being used to design the system are free and open sourced. It will not cost anything to create the software, except for the man hours needed to learn how to use the technologies. Installing the software will only cost time, since all that needs to be done is to compile the code and link it with the kernel. Any maintenance of the program will also not cost any money and only require time from system developers. This means that cost will be kept minimal and almost inconsiderable.

4.7 Usability

The system will be easy to use, because of the intuitive user interface and simple design.The user manual created by Dark nITes will also provide that the system and all its different interfaces will be easily understood and easy to use. The system will also only be available to the two different types of users by means of the log in system designed. This ensures only authorized users can use the system as well enable certain users (Admin/SuperUser) to have access to more sensitive aspects of the system.

4.8 Flexibility

Th system must be highly flexible so that it can accommodate different services with different volumes. This will be ensured through the interface designed to allow users to allocate and remove services which have to be protected as well as the modification of the systems control values.

4.9 Monitorability

The system will be monitored through a GUI which displays packet rates (total and per pool), drop rates, heat-map, packet size, internal overhead and white and black listed IPs. The system as a whole will also log any action done. This includes the time when a packet arrives, the IP address, the location the IP address originates from and as well as the category of threat placed for the IP address. This ensures that any action taken by the system can be monitored.

5 Constraints

There are a couple of constraints imposed on the current system. The main component of the system which is in charge of the DOS protection and load balancing must be on a Linux machine and have access to any of the dependencies that the software requires. This component of the system must also be coded and developed in the language C to ensure that it will integrate and work with XDP and eBPF. This machine must be connected to the internet to ensure that all the logging that the system is done is recorded to the systems database. The computers or devices running the program that is the interfaces for the users also must be connected to the internet to ensure the required information is sent to the database that holds all information for the system. The database service used for the logging from the system as well as any lists need to be done on a database that can handle a very large volume of data due to the nature of the system. This means that many free database services cannot be used due to their size limit.

6 Trace-ability Matrices

6.1 Requirements vs Use-cases

Requirement	**Priority**	**DoS**	**Load-balancer**	**U.I.**
R1.1	1	X		
R1.2	1	X		
R2.1	3		X	
R2.2	3		X	
R2.3	3		X	
R2.4	3		X	
R3.1	2			X
R3.2	2			X
R3.3	2			X

6.2 Requirements vs DoS subsystem

Requirement	**Check blacklist**	**Check conn.s**	**Check packets/s**	**Drop**	**Pass**	**Get Metrics**
R1.1	X	X	X	X		
R1.2					X	
R1.3						X

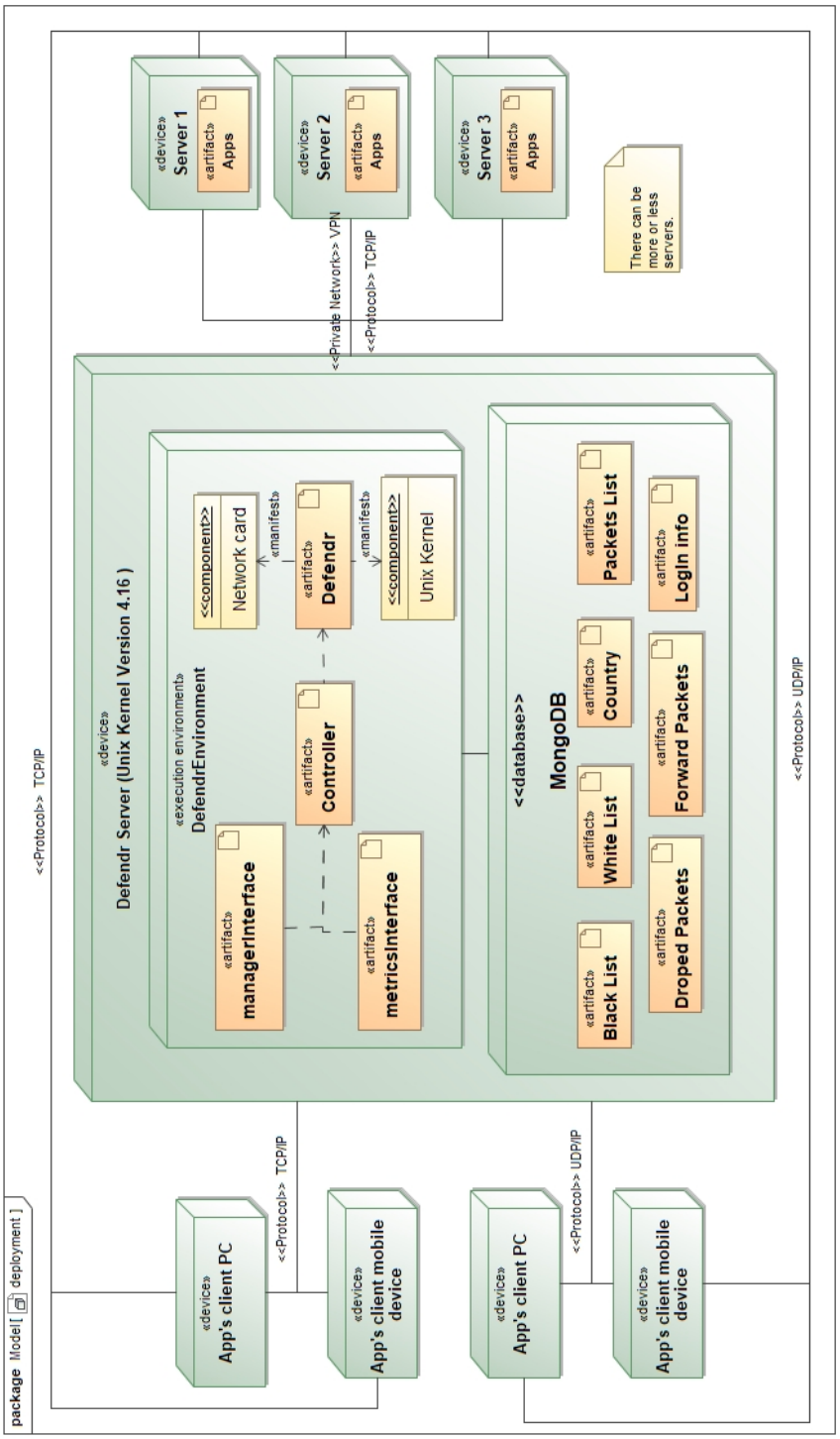
6.3 Requirements vs Load-balancer subsystem

Requirement	Configure	Change alg.	Foward packets	Direct-server resp.
R2.1	X			
R2.2		X		
R2.3			X	
R2.4				X

6.4 Requirements vs U.I. subsystem

Requirement	Set rules	Access interface	View metrics
R3.1		X	
R3.2	X		
R3.3			X

7 Deployment model



8 Architectural Requirements

8.1 System Type

We have implemented a system that makes use of basically two system types. These include a Interactive Subsystem and an Event-Driven Subsystem. These two subsystem types are separated into front-end and back-end systems. The front-end is an instance of an Interactive Subsystem whilst the back-end is an instance of an Event-Driven subsystem.

8.1.1 Interactive Subsystem(Front-End)

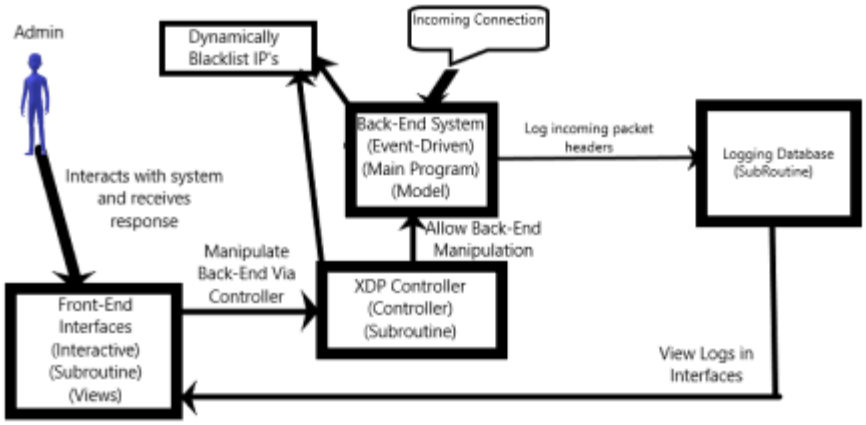
When the administrator logs in to the application, he interacting with the system. He inputs his user name and password and receives a clear feedback from the system. He either enters the rest of the application or an error message appears. Thereafter whatever buttons he clicks on, the application will give him a clear response based on what he pressed. Thus the front-end portion of the system is an Interactive Subsystem.

8.1.2 Event-Driven Subsystem(Back-End)

The back-end portion works as an Event-Driven subsystem because it depends on the incoming packet transmissions from various source IP Addresses before an action is taken. When an IP sends a packet to the system, the system immediately logs that packet into a database. Thereafter, it checks that said IP does not exceed the packet per second limit specified by the system. If it does, the IP is dynamically blacklisted and all corresponding packets are dropped.These are examples of events that take place before the system generates a response for aid events. This in indicative that the back-end is an Event-Driven subsystem.

8.2 Architectural Style

The system currently makes use of three Architectural Styles. These are; The MVC Style, Main program and subroutines finally Event-driven system architecture. These three style are closely linked together and complimenting one another. The front-end interfaces serve as the views in MVC as well as the subroutines of the main program. The XDP controller is the link between the back-end and front end hence it serves as the Controller of the MVC. Finally the back-end serves as the Model in the MVC as well as the Main program which is in fact governed by the Event-driven system Architecture. However the whole system makes use of a Microservices Style. Each part of the system is in essence it's own separate entity that comes together as whole to make the system fully functional. This is seen by the interaction of the interfaces and the back-end that can run separately and are only coupled by the XDP controller that allows for communication between the two. The back-end is further divided into Microservices such as Blacklisting IP's and Logging incoming packets.



9 Technology Decisions

9.1 Interfaces

The interfaces were all coded in Python whilst making use of it's GUI library tkinter. All the GUI's were created using the add-on tool Page which can be found on sourceforge.net. This tool allows for the user to drag and drop tkinter widgets onto the window and thereafter generate Python code that corresponds to what was created with Page. Many python libraries had to be installed. These included pip, dnspython, pymongo, python-tk, urllib3, setuptools and many other packages.

9.2 Packet Dropper

The packet dropper program's kernel files where coded in restricted c and ebpf which where compiled into object and llvm files using the llvm framework and clang compiler. To this end the llvm and clang libraries had to be installed as well as the bcc and kernel-headers packages in order to use bpf maps for key value stores within the kernel. The packett dropper files that are executed in the userspace where however coded using standard c and compilled with the gcc compiler which was installed along side the standard linucx build-essentials package.

9.3 Logging System

The logging system consists of two components: a MongoDB C driver, and one in Python. The proposal was to connect the interface and Defendr system in a way to share information. As such the C driver exists to write data to the database, where Python interfaces can then read and modify data. The database solution is a hosted MongoDB server, with 2 redundancy back-ups. MongoDB was the decision as it is not bound by the constraints of traditional normalisation rules. The database is also built for distributed services, allowing for higher numbers of cheaper servers to be used instead of a monolithic approach. These facts will allow Defendr to grow and adapt in time, providing a more future-proofed approach as it's needs and requirements change.

The software requirements/dependencies:

C

1. MongoDB (mongodb-org)
2. MongoC driver
   (a) Mongo client library (libmongoc-1.0)
   (b) BSON library (libbson-1.14)
   (c) SSL security library (libssl-dev)
   (d) SASL security library (libsasl2-dev)
3. Software builder (CMake)
4. Library querying (pkg-config)

Python

1. MogngoDB Python driver (pymongo)
2. URL parser library (urllib.parse)

9.4 Unit Testing

Unit testing was done using the unittest framework. It supports test automation, sharing of setup and shutdown code for tests, aggregation of tests into collections, and independence of the tests from the reporting framework