Рассмотрим, какие репозитории с Kathara предлагает нам сайт https://github.com. Для этого на сайте в окне поиска введем ключевое слово - Kathara.

Первым мы видим официальный акаунт на github от создателей Kathara - https://github.com/KatharaFramework. На нем можно найти репозитории с описанием проекта, кодом эмулятора, инструкциями по установке на различные операционные системы, примерами лабораторных работ, docker-файлами и скриптами для установки Kathara внутри docker и так далее.

Теперь речь пойдет о неофициальных проектах, связанных с данным средством моделирования. Например, https://github.com/ErCrozzUni/kathara_ebpf - диссертационное исследование от Леонарда Кроццоли и Лоренцо Бенци, цель которого - поэкспериментировать с передовыми методами отслеживания вредоносных потоков в среде эмуляции сети Kathara с помощью программ eBPF. Ebfp (extended Berkeley Packet Filter) - современная технология ядра Linux, которая предоставляет возможность запускать в пространстве ядра программы, заключенные в песочницу. Данный фильтр позволяет расширить возможности ядра без необходимости загружать в него дополнительные модули и запускать компиляцию заново. В области кибербезопасности мониторинг потоков сетевого трафика крайне важен для обнаружения аномальной и потенциально опасной активности, ведь пресечение кибератаки может значительно снизить её воздействие на систему.

Еще один репозиторий, на который мы обратим внимание, - https://github.com/vitome/pnd-labs. Здесь можно найти пять лабораторных работ по Kathara для курса "Practical Network Defense" в Римском университете Sapienza. Данный проект может быть использован для освоения основных методов эмуляции сетей, так как он представляет собой сборник как работающих топологий, так и задач для самостоятельного изучения.

На сайте github.com также представлен репозиторий https://github.com/tcaiazzi/kathara-trex-labs, содержащий готовые Kathara-сценарии для интеграции с TRex (Cisco TRex) — генератора трафика с открытым исходным кодом. TRex использует технологию DPDK (Data Plane Development Kit) - проект Linux Foundation, состоящий из библиотек для ускорения обработки сетевых пакетов на разных архитектурах центрального процессора. В связке с Kathara это позволяет воспроизводить высоконагруженные DDoS-сценарии и исследовать поведение сетевых сервисов и защитных механизмов при реальной нагрузке.

Репозиторий https://github.com/Martolins/Hierarchical-SDN-using-Kathara -- лабораторный проект, моделирующий иерархическую SDN-архитектуру с использованием эмулятора Kathara. Автор создал виртуальную сеть, где управление трафиком происходит

централизованно с помощью нескольких контроллеров, распределенных по уровням. На сценариях лабораторного стенда можно продемонстрировать взаимодействие контроллеров в сети :

- Локальный контроллер управляет трафиком внутри своего домена.
- Корневой контроллер координирует взаимодействие между разными доменами.
- Если один контроллер падает, его часть сети продолжает работать под управлением другого.

Репозиторий https://github.com/buonhobo/Katharsis полноценный фреймворк для управления сетевыми экспериментами, реализующий принцип "Infrastructure as Code". Он предоставляет сетевой интерфейс для использования Kathara, что упрощает описание сложных сетевых топологий, позволяет автоматизировать их развертывание и тестирование.

Остановимся подробнее на репозитории kathara_ebpf. Архив проекта представляет собой лабораторную установку, реализующую анализ сетевого трафика с помощью eBPF. Проект состоит из трех каталогов - docs, ebpf и labs. В каталоге docs присутствуют файлы с командами для eBPF. В каталоге ebpf хранятся файлы, необходимые для создания docker-образа, который будет использоваться внутри узлов kathara (pc1, pc2, r1, r2). При сборке образа создается минимальная linux-среда с утилитами clang, llvm, bpftool, make и библиотекой libbpf, которая позволяет создавать и развертывать программы eBPF в ядре linux. В каталоге labs хранятся лабораторные работы для Kathara. Каждая работа оформлена в своей папке - lab1, lab2.

Рассмотрим lab1 - исследование eBPF-фильтрации UDP-трафика. Внутри мы видим общий файл, описывающий связи в топологии (lab.conf), скрипты инициализации узлов (pc1.startup, pc2.startup, pc3.startup, r1.startup, r2.startup), общий каталог shared и два каталога под узлы pc1 и pc2. В каталоге pc1 хранятся исходный код eBPF-программы (xdp_tree.c), правила компиляции программы (Makefile), скомпилированный объектный файл (xdp_tree.o), скрипт для расшифровки содержимого карты BPF (decode_flow_map.py). В каталоге pc2 хранится Python-скрипт для отправки UDP-пакетов (udp_sender.py).

Запустим топологию - sudo kathara lstart. Посмотрим список запущенных узлов - sudo kathara list.

```
parallels@ubuntu-linux-22-04-desktop:~/Downloads/kathara_ebpf-main/labs/lab1$ sudo kathara lstart
                                Starting Network Scenario

  Description: A simple lab for testing eBPF compatibility
  Version: 1.0
  Author(s): L. Crozzoli, L. Benzi
  Email: contact@kathara.org
  Website: http://www.kathara.org/

⚠Running devices with privileged capabilities, terminals might not open!
[Deploying collision domains]  3/3
[Deploying devices]  6/6
parallels@ubuntu-linux-22-04-desktop:~/Downloads/kathara_ebpf-main/labs/lab1$ sudo kathara list
[sudo] password for parallels:
                                TIMESTAMP: 2025-10-05 23:25:23.476766
```

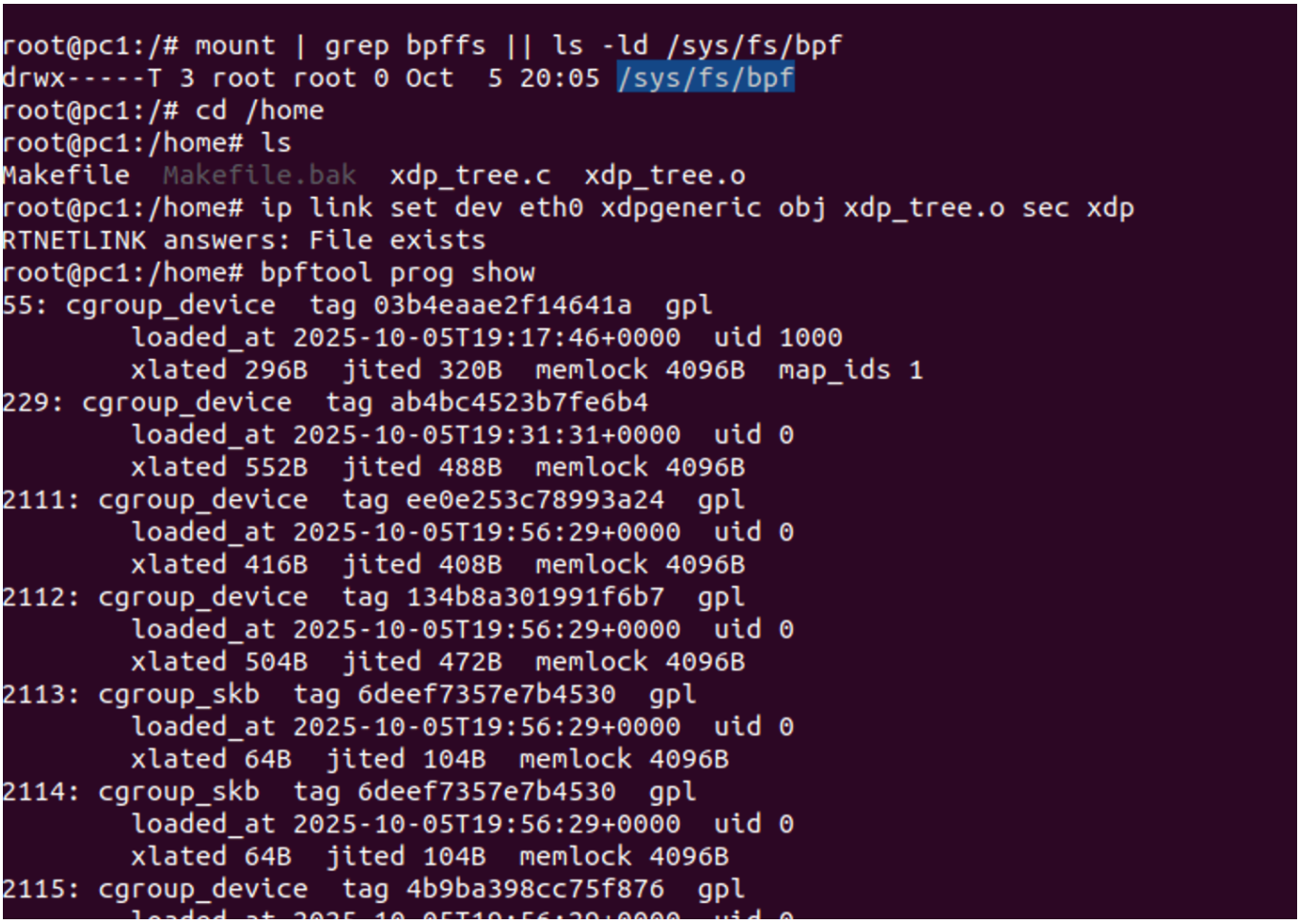
NETWORK SCENARIO ID	NAME	USER	STATUS	IMAGE	PIDS	CPU USAGE	MEM USAGE	MEM PERCENT	NET USAGE	INTERFACES
nWGcwP5LzvSS…	pc3	parallels-cb…	running	kathara/base…	2	0.00%	1004.0 KB / 1.92 GB	0.05 %	170.21 KB / 0 B	0:C
nWGcwP5LzvSS…	wireshark	parallels-cb…	running	lscr.io/linu…	44	1.52%	186.91 MB / 1.92 GB	9.50 %	4.16 KB / 126.0 B	0:Bridged
nWGcwP5LzvSS…	pc2	parallels-cb…	running	kathara/base…	2	0.00%	2.7 MB / 1.92 GB	0.14 %	96.62 KB / 62.42 KB	0:C
nWGcwP5LzvSS…	pc1	parallels-cb…	running	kathara/ebpf…	3	0.00%	2.76 MB / 1.92 GB	0.14 %	73.77 KB / 96.48 KB	0:A
nWGcwP5LzvSS…	r2	parallels-cb…	running	kathara/base…	2	0.00%	980.0 KB / 1.92 GB	0.05 %	170.45 KB / 158.94 KB	0:C, 1:B
nWGcwP5LzvSS…	r1	parallels-cb…	running	kathara/base…	2	0.00%	1.07 MB / 1.92 GB	0.05 %	170.62 KB / 159.07 KB	0:A, 1:B

```
parallels@ubuntu-linux-22-04-desktop:~/Downloads/kathara_ebpf-main/labs/lab1$
```

После запуска топологии был выполнен вход на терминал pc1 командой: sudo kathara connect pc1. В рабочем каталоге /home выполним сборку исходного файла xdp_tree.c командой make.

```
root@pc1:/home# ls
Makefile  Makefile.bak  xdp_tree.c
root@pc1:/home# make
clang -O2 -g -target bpf -Wall -I/usr/include/aarch64-linux-gnu -I/lib/modules/5
.15.0-41-generic/build/include -c xdp_tree.c -o xdp_tree.o
root@pc1:/home#
```

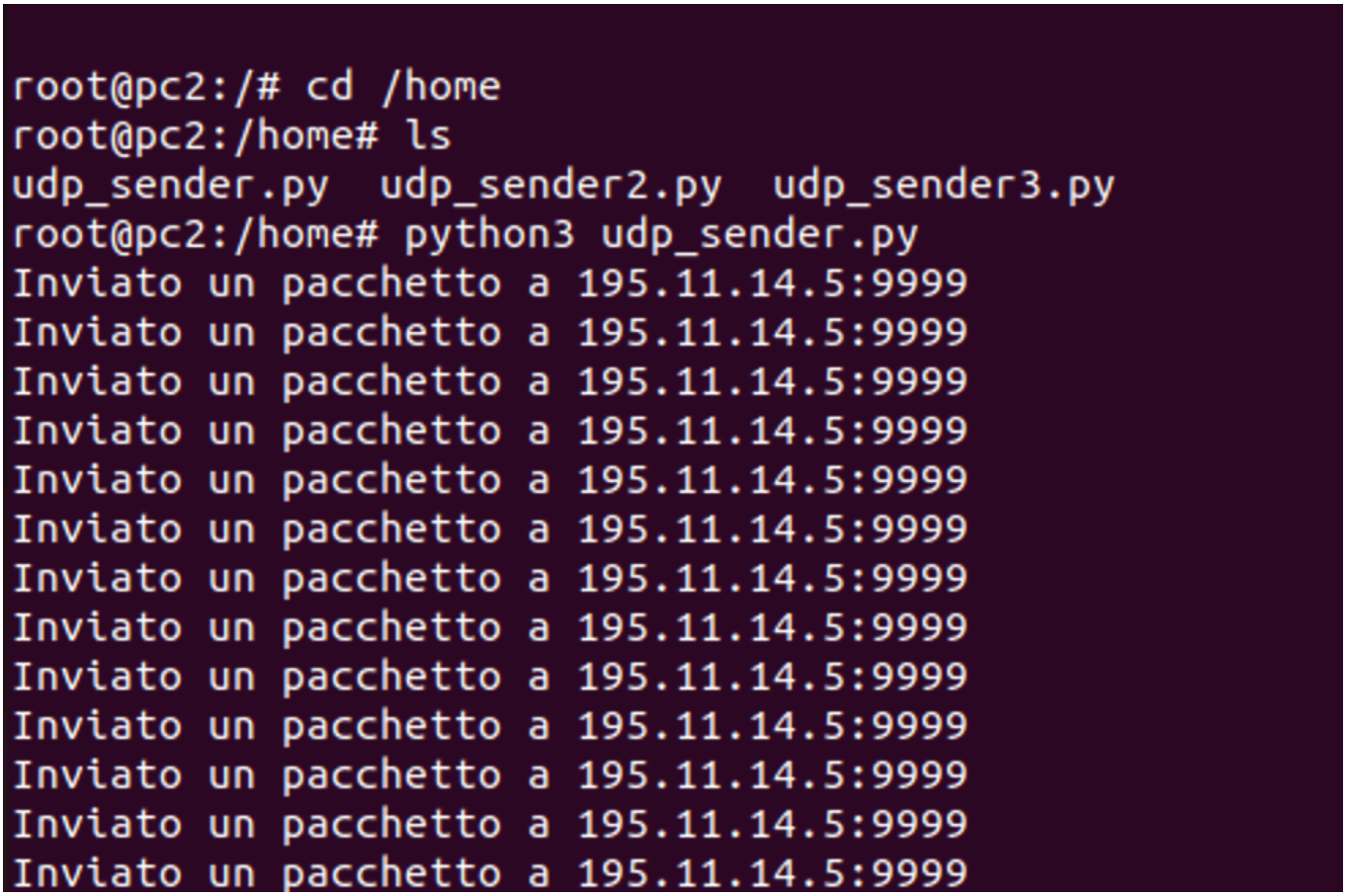
Создается объектный файл xdp_tree.o, который затем загружается в сетевой интерфейс eth0 в виде XDP-фильтра: ip link set dev eth0 xdpgeneric obj xdp_tree.o sec xdp. Для проверки загрузки программы воспользуемся утилитой bpftool, а именно командой bpftool prog show, которая позволит нам посмотреть подробную информацию о загруженных в ядро eBPF-программах.



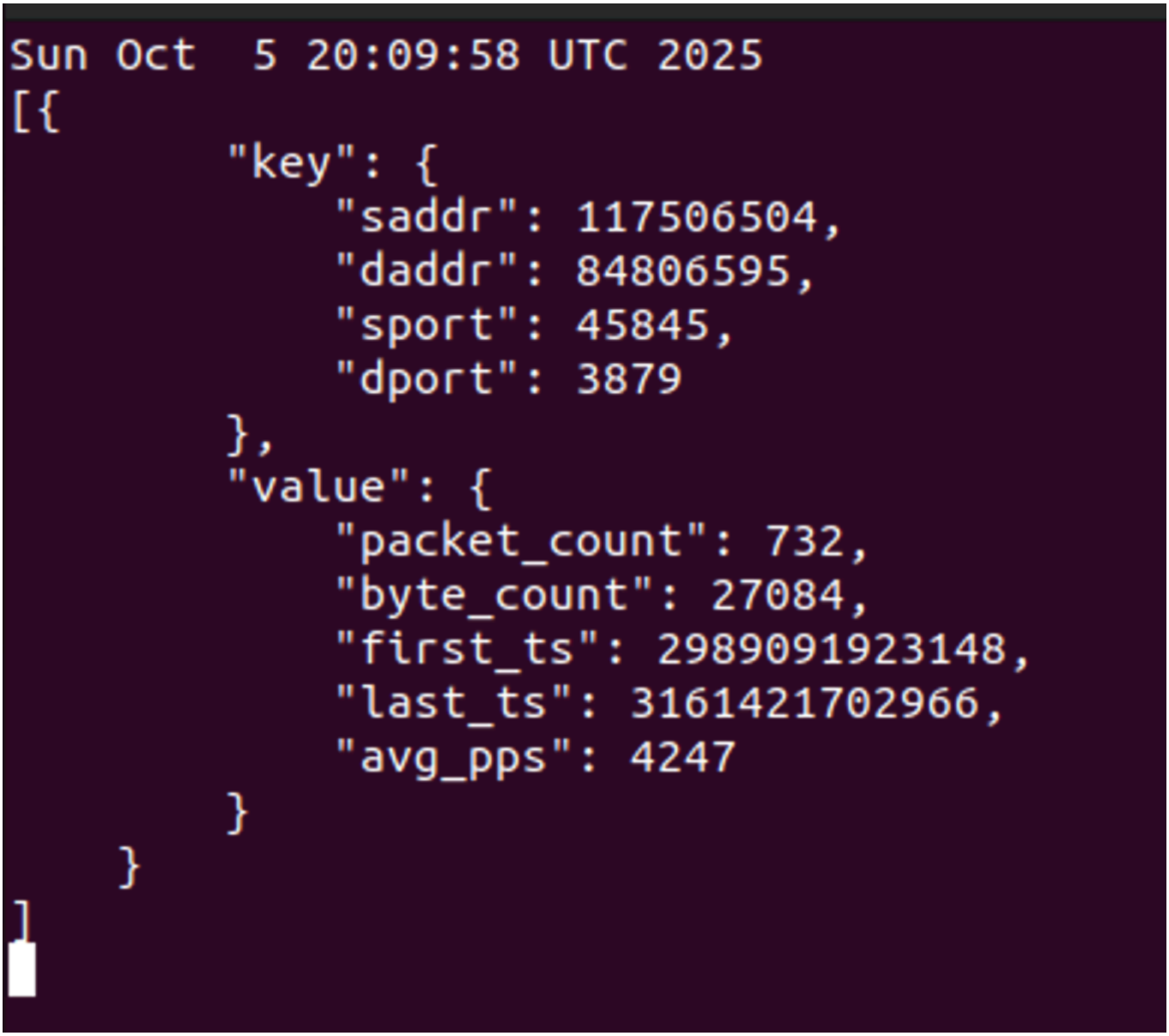
Также воспользуемся командой bpftool map show, которая показывает список всех доступных BPF-карт (maps) в системе. Заметим, что в запись xdp name count_udp_flows подтверждает, что наша программа была успешно загружена в ядро. Увидим, что у нашей программы есть map_ids 45, это означает, что создана и подключена карта flow_map, где будут храниться результаты анализа пакетов.

```
        xlated 552B  jited 488B  memlock 4096B
2184: xdp  name count_udp_flows  tag 27dd72d8e32ade4c  gpl
        loaded_at 2025-10-05T20:05:42+0000  uid 0
        xlated 672B  jited 628B  memlock 4096B  map_ids 45
        btf_id 150
root@pc1:/home# bpftool map show
1: hash  flags 0x0
        key 9B  value 1B  max_entries 500  memlock 8192B
45: hash  name flow_map  flags 0x0
        key 12B  value 40B  max_entries 1024  memlock 57344B
        btf_id 150
68: array  name libbpf_global  flags 0x0
        key 4B  value 32B  max_entries 1  memlock 4096B
69: array  name pid_iter.rodata  flags 0x480
        key 4B  value 4B  max_entries 1  memlock 4096B
        btf_id 192  frozen
        pids bpftool(117310)
70: array  name libbpf_det_bind  flags 0x0
        key 4B  value 32B  max_entries 1  memlock 4096B
root@pc1:/home#
```

Напомним, что цель нашего эксперимента - сгенерировать UDP-нагрузку с узла pc2 и пропустить ее через маршрутизаторы к узлу pc2, где загружена XDP-программа. Подключаемся к узлу pc2. Генерируем нагрузку с помощью запуска Python-скрипта udp_sender.py.



В окне pc1 наблюдаем карту. Видим в значении key поля saddr и daddr адреса источника и получателя, записанные в формате 32-битных беззнаковых целых (uint32) в little-endian. Преобразуя в привычную нам запись формата IPv4 получаем - 117506504 -> 200.1.1.7 и 84806595 -> > 195.11.14.5. Также видим поля sport и dport - это 16-битные целочисленные номера портов отправителя (45845) и получателя (3879). В значении value расписаны packet_count — количество пакетов потока (732), byte_count — суммарный объём в байтах для этого потока, first_ts — временная метка первого пакета в наносекундах, last_ts - временная метка последнего пакета в наносекундах, avg_pps (packets per second) - средняя скорость в пакетах в секунду (4247).



Проведённый эксперимент продемонстрировал работу XDP-программы count_udp_flows, выполняющей анализ UDP-потоков в реальном времени. Программа успешно отлавливает все проходящие пакеты, определяет IP-адреса и порты взаимодействующих узлов, подсчитывает количество и объём пакетов, а также вычисляет интенсивность трафика.