Traffic generator

Створення пакетів за допомогою сирих сокетів. Для Лінукс систем.

Функціонал:

* створення довільних пакетів tcp, udp;
* створення одного або кількох пакетів, інкрементація пакетів;
* обрахування та перевірка чексум; валідація пакетів;
* відправка пакетів (аналіз отриманих пакетів?);
* серіалізація пакетів (у бінарники);
* візуалізація пакетів у форматі json;
* підтримка ipv4, ipv6;
* Робота з кількома сокетами одночасно (багатопоточність?);

Для аплікухи використовуємо паттерн MVC.

Модель: створює пакети, діапазони пакетів. Класи: базовий для пакетів, який реалізує інтерфейс, класси – нащадки. Абстрактна фабрика для пакетів. Клас для серіалазації та десеріалізації (бінарник і джейсон). Структура з параметрами пакету.

Вюшка: клас який реалізує або СЛІ, або ГУІ. Відображає меню, прогрес, рейти, пакети. Для меню реалізувати клас меню, субменю і меню ітем. Клас меню буде містити список субменю, а субменю буде містити меню ітем. Реалізувати клас команди, для взаємодії з контролером.

Контроллер: клас який містить основну логіку, обмінюється коммандами з вюшкою, і з моделлю. Вміє читати та зберігати налаштування аплікухи. (+ взаємодія з вайршарком?)

Корисні посилання:

[простий сніффер](https://habrahabr.ru/post/164901/);

[RAW сокети](https://habrahabr.ru/company/smart_soft/blog/184430/);

[TCP/IP](https://www.opennet.ru/docs/RUS/tcpip/);

[UDP wiki](https://ru.wikipedia.org/wiki/UDP);

[ARP wiki](https://ru.wikipedia.org/wiki/ARP);

[IPv4](https://ru.wikipedia.org/wiki/IPv4)

[TCP wiki](https://uk.wikipedia.org/wiki/TCP);

[сніффер\_1](https://www.opennet.ru/base/dev/traffic_analyze.txt.html)

[сніффер cpp](https://ru.stackoverflow.com/questions/225965/Сниффер-c)

[net\_device\_ioctl](http://man7.org/linux/man-pages/man7/netdevice.7.html)

[IPC](http://tldp.org/LDP/lpg/node7.html)

<http://www.4stud.info/rtos/lecture4.html>

[sockets rsdn](https://rsdn.org/article/unix/sockets.xml)

ipc: <https://users.cs.cf.ac.uk/Dave.Marshall/C/node28.html> <http://www.it.uom.gr/teaching/distrubutedSite/dsIdaLiu/labs/lab2_1/sockets.html>

Need to implement module wich will be analyze traffic in promiscuous mode. This module should communicate with controller class and use relevant class for packet.

***MULTITHREADING***

Main function runs controller process and controller process run other relevant processes – View and Model.

**Threads and responsibility:**

* Main: run all other threads, allocates all main resources, restart process (if needed), handle signals;
* Logger: logs runtime info from all other processes;
* View: user interface (display info, receive commands);
* Controller: processing user commands, model driving, main logic;
* Model: creating traffic items, sending, receiving;
* Each thread able to create own sub process if needed;

**IPC:**

* Using sockets for sending query and response (queue or pipe as variant);
* Items for communications: query. Should have fields ID, size, name, destination (for which module), sequence flag (for “fragmented” queries), sequence ID, query data (size and type);
* each thread should have own work queue, which contain incoming and outgoing queue;
* Perhaps you can allocate some pool of memory as buffer for read and write to socket.

**Thread management:**

* Different priority for threads;
* Restart thread after crash;
* Checking if thread was finished without errors (use shared memory?);

***MAIN FUNCTION***

* Allocate main resources;
* Run other threads (Model, View, Controller, Logger);
* Handle signals;
* Read configuration;

***CLASSES FOR COMMUNICATION***

**Query\_item**:

* For communication between Model, View, Controller, Logger;
* Contain: ID, full\_size, type (get, run, info, answer), sender, destination, sequence flag (for “fragmented” queries), sequence ID, query data (size, command\_name, type, command\_data);

**Query\_serializer**:

* Serialize and deserialize query from/to binary type;

**In\_queue:**

* Receive and deserialize queries;
* Call relevant handler for query;
* Contain map query to handler map (should be registered from outside);
* Check is receiver correct;

**Out\_queue:**

* Serialize and send queries;
* Map query destination to relevant receiver;
* Contain destination adres to receiver map;
* Check is the sender correct;

***LOGGER CLASSES***

**Logger**

* log levels: DEBUG, INFO, NOTICE, WARNING, ERROR, CRITICAL. Default is INFO;
* options to choose maximum size of file, count of files;
* compressing files;
* rotations files (substituting oldest file);
* automatically append date and time;
* adjust date and time format;
* for logging can accept exception, or string;
* option to disable logging;
* logging step by step into file or using internal buffer to temporary storing data and flushing into the file (reducing HDD usage);

**Logger\_server**

* creates and configures logger instance;
* receives items for logging and pass them to logger;

***CONTROLLER CLASSES***

**ControllerCore**

* Loading and saving configurations (using class ConfManager);
* Loading and sawing user defined packets and sequences (TemplateManager)
* Communicates between View and Model (receives and sends querier);
* Creating statistics;
* Main logic (creating, analyzing packets, etc);

**InterfaceDriver**

* Get all available network interface and relevant parameters;
* Set interface to promiscuous mode, and back, to standart mode;
* Configuring network interface (HW addr, IP addr, speed, other parameters);
* Creating and deleting interfaces;

***VIEW CLASSES***

**View:**

* Draws user interface;
* Communicate with user (input and output) and controller (send command, get info for display);
* Create requests for different info from Controller;
* Create request for sending traffic;
* Configuring, e.t.c.

**UserCommand**:

* Exec handler;
* Parameters array;
* Command name;
* Help info;
* Autocompleteion handler;

**Menu:**

* Contains menu items;

**Menu\_Item**:

* Contains command or list of Menu items;

***CLI options***:

* Auto completion (use commands tree and binary search for commands and autocompleting handles for different parameters;
* Progress bar;
* Different colors;

***MODEL CLASSES***

**PacketHeader:** (headers can be different)

* Contains binary representation of packet header;
* Can represent TCP and UDP headers;
* Representation of each field separately;

**Packet:**

* Header (tcp or udp, ip) instance of PacketHeader class;
* Data segment;
* Checksum;
* Serialization and deserialization (binary representation);

**TrafficFactory:**

* Creates instance of packet from input parameters;

**Socket:**

* Creating, managing and deleting sockets;
* Sending single or multiple packets through socket;
* Reading data from socket;

***CLASS DIAGRAMS***

Enums for diferent network protocols – arp, icmp, igmp, ospf…

Flows:

Loading:

Initializing Controller. Controller initialize Model, then controller Initialize View.

View during initialization provide registering callbacks for commands, each successful registered command can be displayed and executed.

Executing command:

View send command to Controller. Controller call relevant handlers, and sends to View response. View print output using response info.

Need to add: graft of classes and scheme of classes and base flows

Need to add comparator for packets