ТЕЛЕКОММУНИКАЦИИ

УДК 681.518.5

АНАЛІЗ ІНФРАСТРУКТУРИ HOST-SWITCH-HOST-VIRTUALBOX ЗАСОБАМИ МОВИ PYTHON3

БЕЗРУК В.М., КРИВЕНКО С.А., НІКОЛЕНКО Д.О.

Розглядається будова інформаційної інфраструктури для створення мережі моніторингу та контролю за станом технології IoT, які традиційно використовують в мережах з низькою швидкістю, низькою потужністю і короткими пакетами даних. Аналіз зосереджений на прикладі реалізації концепції мікросайту Bottle з підтримкою бази даних SQLight. Експериментальні результати показують ефективність і стабільність роботи системи.

Ключові слова – мікросайт, Інтернет речей, база даних, аналізатор пакетів, технологія віртуалізації.

Keywords – Bottle, Internet of Things, SQLight, Sniffers, VirtualBox.

Вступ

Sniffers – це програми, які можуть захоплювати, шифрувати та виявляти пакети мережного трафіку за допомогою пакета програм та аналізувати їх з різних причин. Зазвичай використовується в галузі безпеки мережі. Wireshark – це дуже поширений аналізатор протоколів пакетів [1]. Ця комп'ютерна програма виявляє та реєструє різноманітну обмежену інформацію, особливо секретні паролі, необхідні для отримання доступу до файлів або мереж. Аналізатори пакетів можуть бути також написані мовою python. У цій статті описані мовою python кілька простих аналізаторів пакетів для платформи Linux. Використовується операційна система Linux, оскільки, хоча python є портативним, програми не будуть працювати або давати аналогічні результати, наприклад, на платформі Windows. Це пов'язано з різницею в реалізації сокета API [2].

В даній статті досліджується архітектура "клієнт-сервер", що використовується в навчальному процесі Харківського національного університету радіоелектроніки. Поняття архітектури "клієнт-сервер" для різних людей має різний зміст, залежно від їх спеціалізації і від того, йде мова про програмне забезпечення чи про систему апаратних засобів. У будь-якому випадку визначення цього поняття є досить простим: сервер (апаратний пристрій або програмне забезпечення) надає "послуги", які потрібні одному або кільком клієнтам (користувачам послуг). Єдиним призначенням сервера є очікування запитів (клієнтів), повернення відповідей на них (надання послуги) і очікування наступних запитів.

Клієнти, з іншого боку, звертаються до сервера з конкретним запитом, відправляють всі необхідні дані, а потім чекають відповіді сервера, який може або надавати запитувані дані, або містити вказівку на причину відмови. Сервер працює невизначено довго, безперервно обробляючи запити; клієнти виконують одноразові запити для отримання послуги, отримують цю послугу, після чого завершують поточну операцію. Клієнт може в подальшому виконувати додаткові запити, але вони розглядаються в рамках окремих операцій.

Вступаючи в світ мережевого програмування, найважче зрозуміти, як працює сервер. Що ж стосується функціонування клієнта, то його можна описати набагато простіше в порівнянні з сервером. Завдання клієнта полягає лише в тому, що він повинен створити свою окрему кінцеву точку зв'язку, а потім встановити з'єднання з сервером. Після цього клієнт може виконувати запити, в тому числі здійснювати весь необхідний обмін даними. Після обробки запиту або отримання клієнтом результату, або просто необхідного підтвердження сеанс зв'язку завершується [3].

Широкі можливості відкриває альтернативний підхід на основі застосування мови Python як універсальної для різних операційних систем IoT [4].

Метою дослідження є аналіз можливостей застосування мови Python як універсальної для різних операційних систем при реалізації аналізатора протоколів пакетів в мережі із складною інфраструктурою.

1. Модель інфраструктури

1.1. Апаратна модель

Визначення поняття архітектури "клієнт-сервер" ілюструється на рис. 1, де показаний призначений для користувача, або клієнтський, комп'ютер, за допомогою якого отримується інформація від сервера через Інтернет.

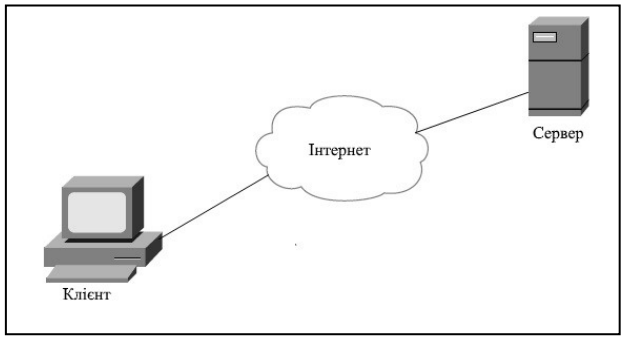

Рис. 1. Типове використання одного з кластерів ZigBee

Хоча така система дійсно може служити прикладом архітектури "клієнт-сервер", це не єдиний варіант даної архітектури. Крім того, архітектура "клієнт-сервер" може бути реалізована не тільки за допомогою програмного забезпечення, але і з застосуванням комп'ютерних апаратних засобів.

Прикладами апаратних серверів можуть служити сервери друку. Вони обробляють завдання на друк і відправляють їх на принтер (або на якийсь інший пристрій друку), підключений до такої системи. Як правило, доступ до сервера друку нада-

ється в мережі, і клієнтські комп'ютери відправляють на нього запити друку.

Ще одним прикладом апаратного сервера може служити файловий сервер. Такі сервери, як правило, являють собою комп'ютери з великим обсягом запам'ятовуючих пристроїв загального призначення, до яких надається дистанційний доступ для клієнтів. Комп'ютери клієнтів монтують диски серверного комп'ютера так, начебто ці диски перебувають на локальному комп'ютері. Однією з найбільш широко застосовуваних мережних операційних систем, які підтримують файлові сервери, є NFS компанії SUN MICROSYSTEMS. Якщо доступ до мережного дискового накопичувача організований так, що неможливо визначити, чи встановлений диск на локальному комп'ютері чи знаходиться в мережі, це означає, що система "клієнт-сервер" успішно виконує свою роботу. Призначення такої системи полягає в тому, щоб користувач у своїй роботі не відчував різниці між локальним і мережним диском. Таке функціонування системи доступу забезпечується за допомогою програмної реалізації.

Програмні сервери також експлуатуються на апаратних засобах, але, на відміну від апаратних серверів, не мають виділених периферійних пристроїв (принтерів, дискових накопичувачів). До основних послуг, що надаються програмними серверами, відносяться виконання програм, пошук і передача даних, агрегування, оновлення даних, а також здійснення інших типів запрограмованих дій або операцій маніпулювання даними. У наші дні до найбільш широко застосовуваних програмних серверів відносяться веб-сервери.

Приватні особи або компанії, які бажають експлуатувати власні веб-сервери, повинні придбати один чи кілька комп'ютерів, підготувати вебсторінки або створити веб-додатки, які вони бажають надати користувачам, а потім запустити веб-сервер. Призначення такого сервера полягає в тому, щоб приймати клієнтські запити, повертати веб-сторінки веб-клієнтам, тобто браузерам на комп'ютерах користувачів, а потім чекати наступного клієнтського запиту. Передбачається, що після запуску ці сервери повинні експлуатуватися невизначено довго. Зрозуміло, це недосяжна мета, але безперервна експлуатація веб-серверів здійснюється настільки довго, наскільки це можливо, за умови відсутності втручання якоїсь зовнішньої сили, яка свідомо чи несвідомо (наприклад, через відмову апаратних засобів) викликає припинення їх роботи.

Ще одним типом програмних серверів є сервери баз даних. Такі сервери приймають клієнтські запитів на збереження або пошук інформації, виконують дії відповідно до цих запитах, а потім очікують надходження завдання на виконання чергової роботи. Ці сервери також призначені для експлуатації протягом невизначено довгого часу. Останнім типом програмного сервера є віконний сервер. Такі сервери можуть розглядатися майже як аналогічні апаратним серверам. Вони експлуатуються на комп'ютері з підключеним пристроєм відображення, наприклад, монітором того чи іншого типу. Клієнтами віконного сервера є програми, для роботи яких потрібно віконне середовище. Такі клієнти прийнято розглядати як додатки з графічним інтерфейсом користувача GUI[1]. Спроба виконання подібних додатків без віконного сервера, іншими словами, в середовищі підтримки текстового режиму, такому як вікно DOS або командний інтерпретатор Unix, призводить до невдалого завершення. Після отримання доступу до віконного сервера додатки з графічним інтерфейсом функціонують нормально. Таке середовище стає ще більш цікавим, якщо застосовується поряд з мережними засобами. Зазвичай як дисплей для віконного клієнта служить сервер на локальному комп'ютері, але в деяких мережних середовищах підтримки вікон, таких як система X Window, для виведення відображення з графічного додатку можна вибрати віконний сервер іншого комп'ютера. У таких ситуаціях програма з графічним інтерфейсом користувача може експлуатуватися на одному комп'ютері, а виводити зображення на іншому!

Як приклад описаних вище систем, можна розглянути передачу інформації в системі, схема якої наведена на рис. 2.

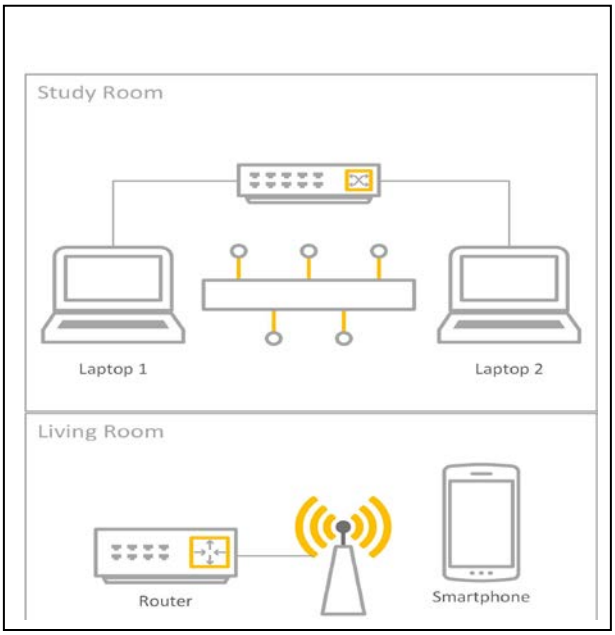

Рис. 2. Апаратна модель

До складу системи входять такі елементи: ноутбуки Lenovo s10-3(Windows 8.1 pro) та Asus X200MA(Windows 10 education); комутатори D-Link DIR100 та TP-LINK WR740N; пасивна оптична мережа.

[1] graphical user interface - GUI

1.2. Налаштування системи

На ноутбук Lenovo s10-3 була встановлена віртуальна машини VirtualBox версії 5.2.8. VirtualBox надає вісім віртуальних карт Ethernet PCI для кожної віртуальної машини. Для кожної такої картки можна самостійно вибрати обладнання, яке буде віртуалізоване, а також режим віртуалізації, в якому віртуальна карта буде працювати у відношенні фізичного мережевого обладнання на хості Asus X200MA (рис. 3).

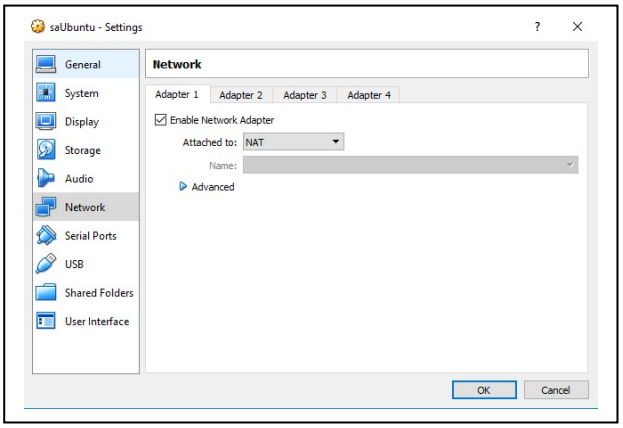

Рис. 3. Перша віртуальна карта

Чотири мережних карти можна налаштувати у розділі "Мережа" діалогового вікна в графічному інтерфейсі користувача VirtualBox. Налаштування другої мережної карти наведено на рис. 4.

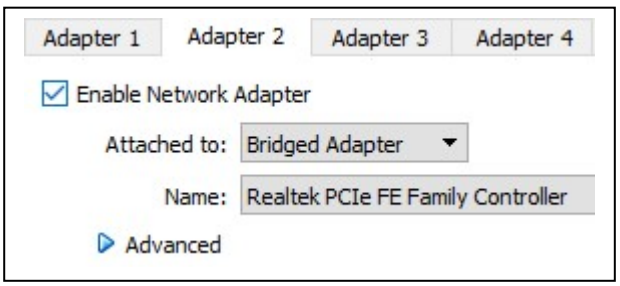

Рис. 4. Друга віртуальна карта

Для другої картки вибрано тип обладнання (Realtec PCIe FE Family Controller), яке буде представлено віртуальній машині.

За допомогою мостових мереж VirtualBox використовує драйвер пристрою на системі WINDOWS 10 хосту Asus X200MA, який фільтрує дані з фізичного мережевого адаптера. Цей драйвер називається "net filter". Це дозволяє VirtualBox перехоплювати дані з фізичної мережі та вставляти в неї дані, ефективно створюючи новий мережний інтерфейс у програмному забезпеченні. Коли гість використовує такий новий інтерфейс програмного забезпечення, він виглядає на хост-системі так, ніби гість був фізично підключений до інтерфейсу за допомогою мережевого кабеля: хост може відправити дані гостю через цей інтерфейс і отримувати від нього дані. Це означає, що можна налаштувати маршрутизацію або міст між гостем та іншою частиною мережі.

Для цього VirtualBox потребує драйвера пристрою на хост-системі. Те, як працює мостова мережа, повністю переписано за допомогою VirtualBox 2.0 та 2.1, залежно від операційної системи хоста. З точки зору користувача, головна відмінність полягає в тому, що складна конфігурація більше не потрібна ні в одній із підтримуваних операційних систем для хостів.

Незважаючи на те, що TAP більше не потрібний для Linux з мостовими мережами, можна використовувати інтерфейси TAP для деяких додаткових налаштувань, оскільки ви можете підключити віртуальну машину до будь-якого хоста-інтерфейсу, який також може бути інтерфейсом TAP.

Щоб увімкнути мережу з мостом, все що потрібно зробити, це відкрити діалогове вікно налаштування віртуальної машини, перейти на сторінку «Мережа» та вибрати «Мостова мережа» у списку для «Прикріплено до». Нарешті, вибрати потрібний головний інтерфейс зі списку внизу сторінки, який містить фізичні мережні інтерфейси систем. Наприклад, на типовому MacBook це дозволить вам вибрати між "en1: AirPort" (який є бездротовим інтерфейсом) і "en0: Ethernet", який представляє інтерфейс із мережевим кабелем.

Перехід до бездротового інтерфейсу виконується інакше, ніж від моста до проводового інтерфейсу, тому що більшість бездротових адаптерів не підтримують нерозбірливий режим. У всьому трафіку потрібно використовувати MAC-адресу бездротового адаптера хоста, тому VirtualBox має замінити вихідну MAC-адресу в заголовку Ethernet вихідного пакета, щоб переконатися, що відповідь буде надіслано на головний інтерфейс. Коли VirtualBox бачить вхідний пакет з IP адресою призначення, що належить одному з адаптерів віртуальної машини, він замінює MAC-адресу призначення в заголовку Ethernet за допомогою MAC-адреси адаптера VM та передає його. VirtualBox розглядає пакети ARP та DHCP, щоб вивчати IP адреси віртуальних машин.

Залежно від операційної системи вашого хосту слід пам'ятати про такі обмеження:

У хостах Macintosh функціональність обмежена, коли використовується AirPort (бездротова мережа Mac) для мостових мереж. В даний час VirtualBox підтримує тільки IPv4 та IPv6 через AirPort. Для інших протоколів (наприклад, IPX) потрібно вибрати провідний інтерфейс.

У хостах Linux функціональність обмежена, коли використовуються бездротові інтерфейси для мостових мереж. В даний час VirtualBox підтримує тільки IPv4 та IPv6 через бездротовий зв'язок.

Для інших протоколів (наприклад, IPX) потрібно вибрати провідний інтерфейс.
Крім того, встановлення MTU на менш ніж 1500 байтів на дротяних інтерфейсах, надане драйвером sky2 на NIC ультра Ethernet UE Marvell Yukon II EC, як відомо, викликає втрати пакетів при певних умовах.

1.3. Взаємодія елементів

Схема організації зв'язку наведена на рис. 5.

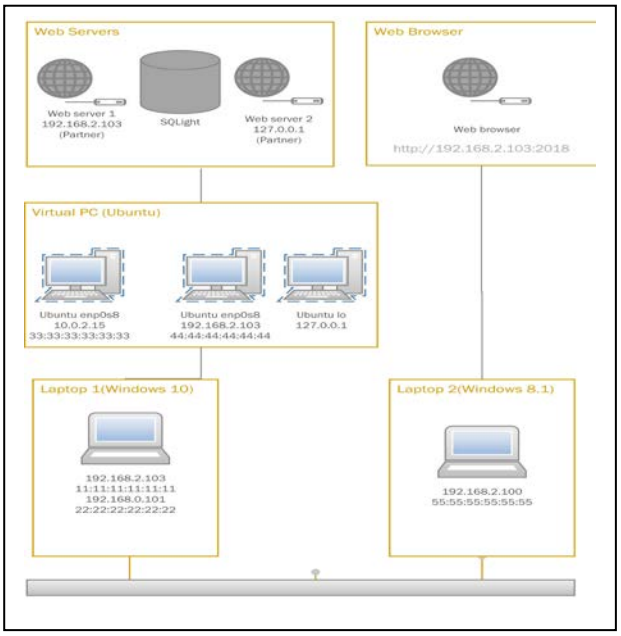

Рис. 5. Схема організації зв'язку

Веб-сервер BOTTLE (один файл мовою PYTHON3) запускається в середовищі IDLE, яке працює під гостьовою операційною системою XUBUNTU 18.04 на віртуальній машині. Операційною системою хоста Asus X200MA є WINDOWS10. Призначення такого сервера полягає в тому, щоб приймати клієнтські запити, повертати веб-сторінки веб-клієнтам, тобто браузерам на комп'ютерах користувачів, а потім чекати наступного клієнтського запиту. Клієнтські запити зберігаються в базі даних під управлінням SQLight (один файл мовою PYTHON3).

2. Результати дослідження

Результати дослідження отримані за допомогою написаних мовою python кількох простих аналізаторів пакетів для платформи Linux. Використовується операційна система XUBUNTU. Аналізується трафік фрейм ворка (Python Web Framework) і SQLight, які підвищують продуктивність мережі і ефективність обслуговування великої кількості пристроїв IoT.

2.1. Аналіз фрейму Ethernet

Пакет даних на дроті та фрейм як його корисне навантаження складаються з двійкових даних. Ethernet передає дані з найбільш значущим октетом (байтом) спочатку; однак в межах кожного октету найменший значний біт переноситься. Внутрішня структура фрейму Ethernet вказана в стандарті IEEE 802.3. На рис. 6 показано повний пакет Ethernet та кадр всередині, як це було надіслано, для розміру корисного навантаження MTU 1500 октетів.
Деякі реалізації Gigabit Ethernet та інших варіантів високої швидкості Ethernet підтримують більші фрейми, відомі як Джамбо фрейми.

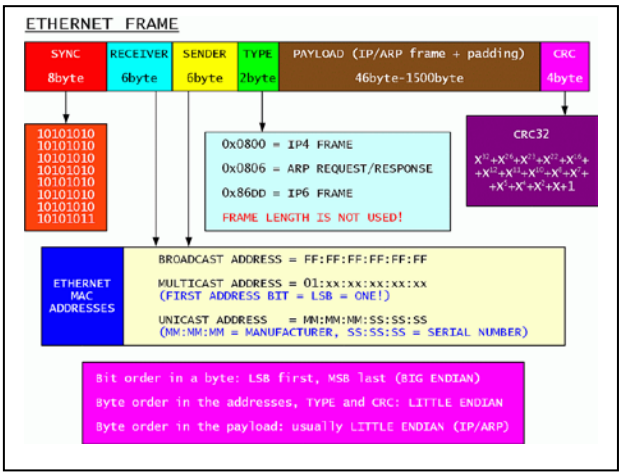

Рис. 6. Фрейм Ethernet

Для видобування фреймів створюється об'єкт сокету

conn = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(3))

з такими параметрами: сім'я адрес – AF_PACKET; тип сокету – SOCK_RAW; тип протоколу – визначається на основі перетворення об'єктів мережі до об'єктів хосту socket.ntohs(3).

2.2. Розпакування заголовків IP-пакетів

IP-пакет складається з розділу заголовків та розділу даних. IP-пакет не має контрольної суми даних чи будь-якого іншого суфікса після розділу даних. Як правило, канальний рівень (Ethernet) інкапсулює IP-пакети в фрейми з суфіксом CRC, який виявляє більшість помилок, і зазвичай контрольна сума TCP-рівня до кінця визначає більшість інших помилок. Заголовок пакетів IPv4 складається з 14 полів, з яких 13 необхідні (рис.7).
Наведений нижче код повертає правильно відформатовану адресу IPv4:

```
import struct

class IPv4:

    def __init__(self, raw_data):

        version_header_length = raw_data[0]

        self.version = version_header_length >> 4

        self.header_length = (version_header_length & 15) * 4

        self.ttl, self.proto, src, target = struct.unpack('! 8x B B
2x 4s 4s', raw_data[:20])

        self.src = self.ipv4(src)
```

```
        self.target = self.ipv4(target)
        self.data = raw_data[self.header_length:]
    # Returns properly formatted IPv4 address
    def ipv4(self, addr):
        return '.'.join(map(str, addr))
```

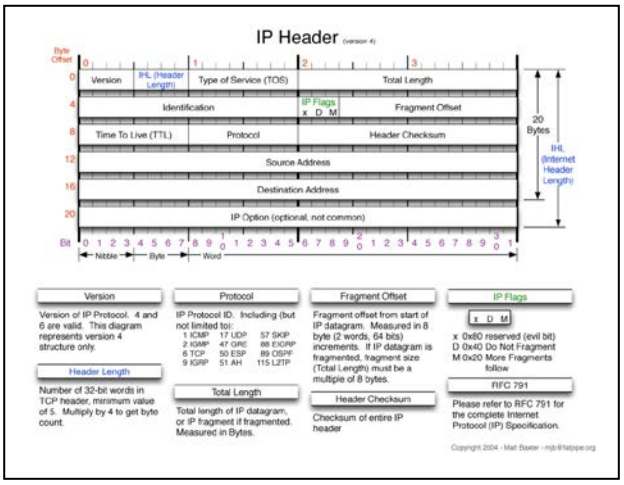

Рис. 7. Заголовок IP-пакетів

2.3. Розпаковка даних ICMP та TCP

ICMP – мережний протокол, що входить в стек протоколів TCP/IP. В основному ICMP використовується для передачі повідомлень про помилки й інші виняткові ситуації, що виникли при передачі даних. Також на ICMP покладаються деякі сервісні функції, зокрема на основі цього протоколу заснована дія таких загальновідомих утиліт як ping та traceroute.

ICMP описаний в RFC 792 (з доповненнями в RFC 950) і є стандартом Інтернету (входить в стандарт STD 5 разом з протоколом IP). Хоча формально ICMP використовує IP (ICMP пакети інкапсулюються в IP пакети), він є невід'ємною частиною IP й обов'язковий при реалізації стека TCP/IP. Поточна версія ICMP для IPv4 називається ICMPv4. В IPv6 існує аналогічний протокол ICMPv6.

ICMP не є протоколом, орієнтованим на з'єднання (як, наприклад, TCP), тобто при втраті пакету ICMP не буде робити ніяких спроб по його відновленню. ICMP повідомлення (тип 12) генеруються при знаходженні помилок у заголовку IP пакета (за винятком самих ICMP пакетів, щоб не призвести до нескінченно зростаючого потоку ICMP повідомлень про ICMP повідомлення).

ICMP повідомлення (тип 3) генеруються маршрутизатором при відсутності маршруту до адресата.

Схема розпаковки сегментів TCP наведена на рис. 8.

Код, наведений нижче, повертає правильно відформатований сегмент TCP:

```
import struct
class TCP:
    def __init__(self, raw_data):
        (self.src_port, self.dest_port, self.sequence,
self.acknowledgment, offset_reserved_flags) =
struct.unpack(
            '! H H L L H', raw_data[:14])
        offset = (offset_reserved_flags >> 12) * 4
        self.flag_urg = (offset_reserved_flags & 32) >> 5
        self.flag_ack = (offset_reserved_flags & 16) >> 4
        self.flag_psh = (offset_reserved_flags & 8) >> 3
        self.flag_rst = (offset_reserved_flags & 4) >> 2
        self.flag_syn = (offset_reserved_flags & 2) >> 1
        self.flag_fin = offset_reserved_flags & 1
        self.data = raw_data[offset:]
```

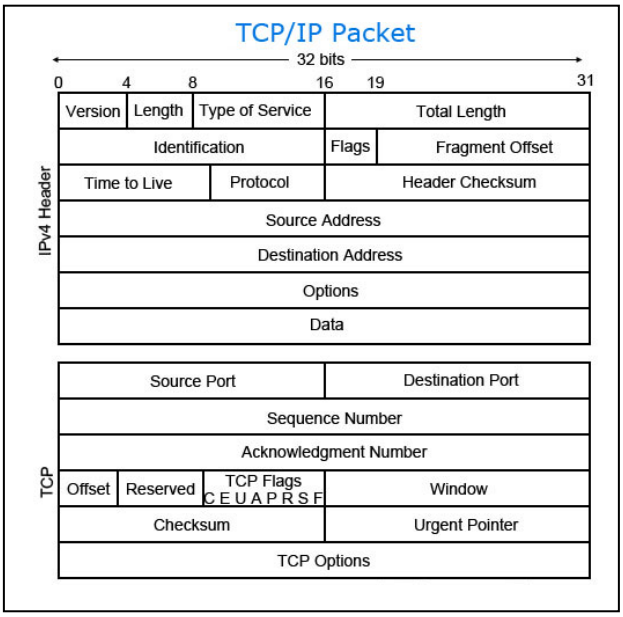

Рис. 8. Сегмент TCP

Код, який дозволяє не друкувати надлишкову інформацію щодо локального хосту, наведено нижче:

```
s = socket.socket(socket.AF_INET, socket.SOCK_RAW,
socket.IPPROTO_TCP)#line01
    resa=re.search('127.0.0.1',str(s.recvfrom(1))) #line02
    if resa is  None : #line03
        print(s.recvfrom(1)) #line04
    else: #line05
        print('nosano') #line06
        break#line07
```

В рядку 1 створюється об’єкт сокету з такими параметрами: сім’я адрес – AF_INET; тип сокету – SOCK_RAW; тип протоколу – IPPROTO_TCP. Крім того, в коді використовується сім’я адрес AF_PACKET, яка вперше введена в сучасну версію python 3.6.

Рядок 2 забезпечує пошук адреси Інтернет протоколу локального хосту.

Рядки 3-7 забезпечують роздруківку адрес Інтернет протоколу (рис. 9).

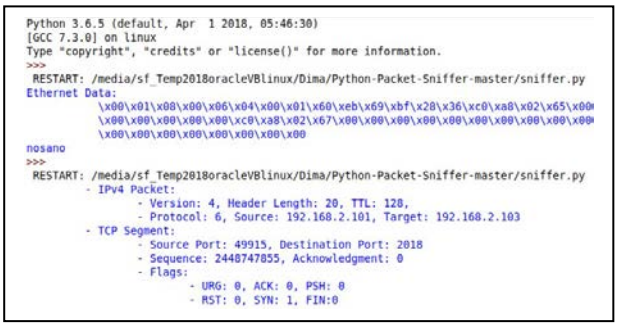

Рис. 9. Результати аналізу

Дані Ethernet роздруковані вбудованими функціями:

```
print('Ethernet Data:')

print(format_multi_line(DATA_TAB_1, eth.data))
```

Інформацію щодо пакетів Інтернет протоколу друкує код

```
print(TAB_1 + 'IPv4 Packet:')

print(TAB_2 + 'Version: {}, Header Length: {}, TTL: {},'.format(ipv4.version, ipv4.header_length, ipv4.ttl))

print(TAB_2 + 'Protocol: {}, Source: {}, Target: {}'.format(ipv4.proto, ipv4.src, ipv4.target))
```

Інформацію щодо сегментів протоколу TCP друкує код

```
print(TAB_1 + 'TCP Segment:')

print(TAB_2 + 'Source Port: {}, Destination Port: {}'.format(tcp.src_port, tcp.dest_port))

print(TAB_2 + 'Sequence: {}, Acknowledgment: {}'.format(tcp.sequence, tcp.acknowledgment))

print(TAB_2 + 'Flags:')

print(TAB_3 + 'URG: {}, ACK: {}, PSH: {}'.format(tcp.flag_urg, tcp.flag_ack, tcp.flag_psh))

print(TAB_3 + 'RST: {}, SYN: {}, FIN:{}'.format(tcp.flag_rst, tcp.flag_syn, tcp.flag_fin))
```

Висновок

Аналіз можливостей застосування мови Python як універсальної для різних операційних систем при реалізації аналізатора протоколів пакетів в мережі із складною інфраструктурою показав ефективність і стабільність роботи системи.

Література:

1. Wireshark User’s Guide [Online] Available: https://www.wireshark.org/download/docs/user-guide.pdf
2. Code a network packet sniffer in python for Linux [Online] Available: https://www.binarytides.com/python-packet-sniffer-code-linux//
3. Wesley J. Chun Core PYTHON Applications Programming. Third Edition, Michigan, Pearson Education, Inc,.2012, pp. 53-93.
4. Analysis ZigBee Alliance initiative to create a universal language for IOT dotdot/ Bezruk V., Vlasova V., Krivenko S. // Radioelektronika i informatika. 2017. N 1. P. 9-13.
5. struct — Interpret bytes as packed binary data [Online] Available: https://docs.python.org/3/library/struct.html#format-characters/
6. Python network packet sniffer. [Online]. Available: https://github.com/buckyroberts/Python-Packet-Sniffer

Надійшла до редколегії 03.06.2018

Рецензент: д-р техн. наук, проф. Бараннік В.В.

Безрук Валерій Михайлович, д-р техн. наук, академік академії зв’язку України, професор, завідувач кафедри «Інформаційно-мережна інженерія» ХНУРЕ. Адреса: Україна, 61166, Харків, пр. Науки, 14, E-mail: valeriy_bezruk@ukr.net.

Кривенко Станіслав Анатолійович, канд. техн. наук, доцент кафедри «Інформаційно-мережева інженерія» ХНУРЕ. Адреса: Україна, 61166, Харків, пр. Науки, 14, E-mail: Stanislav.Kryvenko@nure.ua.

Ніколенко Дмитро Олексійович, студент магістерської підготовки кафедри «Інформаційно-мережна інженерія» ХНУРЕ. Адреса: Україна, 61166, Харків, пр. Науки, 14, E-mail: dmitro.nikolenko@nure.ua.

Bezruk Valeriy, Doctor of Technical Sciences, professor, academician of the Ukrainian Academy of Telecommunications**,** Professor, Head of the Department of Information and network engineering. Address: 14 Nauki av., Kharkiv city, Ukraine, 61166**,** E–mail: valeriy_bezruk@ukr.net

Krivenko Stanislav, Candidate of technical sciences, associate professor of the Information and network-engineering department, Kharkov National University of Radioelectronics. Address: 14 Nauki av., Kharkiv city, Ukraine, 61166**,** E-mail: Stanislav.Kryvenko@nure.ua.

Nikolenko Dmitry, Master's degree student, Department "Information and Network Engineering", Kharkiv National University of Radio Electronics. Address: 14 Nauki av., Kharkiv city, Ukraine, 61166**,** E-mail: dmitro.nikolenko@nure.ua