Технічні науки

УДК 004.428.4

Ткаченко Олександр Миколайович
кандидат технічних наук, доцент
Вінницький національний технічний університет
Ткаченко Александр Николаевич
кандидат технических наук, доцент
Винницкий национальный технический университет
Tkachenko Oleksandr
PhD, Associate Professor
Vinnytsia National Technical University

Комаров Володимир Леонідович
аспірант кафедри обчислювальної техніки
Вінницького національного технічного університету
Комаров Владимир Леонидович
аспирант кафедры вычислительной техники
Винницкого национального технического университета
Komarov Volodymyr
Graduate Student of the Department of Computer Engineering
Vinnytsia National Technical University

РОБОТА З ПРОТОКОЛОМ ETHERNET МОВОЮ ПРОГРАМУВАННЯ GOLANG
РАБОТА С ПРОТОКОЛОМ ETHERNET ЯЗЫКОМ ПРОГРАММИРОВАНИЯ GOLANG
WORKING WITH THE ETHERNET PROTOCOL IN THE GOLANG PROGRAMMING LANGUAGE

Аннотація. *У даній статті розглядаються програмі засоби для реалізації мережевих додатків з використанням низькорівневих мережевих примітивів, таких як Ethernet фрейми. Зокрема розглядається приклад створення Ethernet фрейма, його особливості, відправка та отримання. Для полегшення описаних процесів застосовується бібліотеки відкритого коду, для роботи з низькорівневими примітивами.*

Робота з даними протоколу Ethernet є основою будь якого програмного забезпечення, що працює в мережі, як локальній так і глобальній. Згідно моделі OSI даний протокол працює на канальному рівні. Основними одиницями даних, що передаються за цим протоколом є фрейми. Вони інкапсулюють в собі дані, що передаються мережею а також додають службову інформацію, необхідну для коректного опрацювання цього фрейму пристроями в мережі.

Мова Golang є достатньо швидкою, та включає в себе засоби роботи з мережею. Проте, робота з низькорівневими даними, такими як фрейми Ethernet є досить складною.

В даній статті описано процес створення, відправки та отримання Ethernet фрейму мовою Golang з використанням допоміжної бібліотеки "mdlayher/ethernet".

Ключові слова: *ethernet, golang, frame, ethernet frame, networking, фрейм, OSI.*

Аннотация. *В данной статье рассматриваются программе средства для реализации сетевых приложений с использованием низкоуровневых сетевых примитивов, таких как Ethernet фреймы. В частности рассматривается пример создания Ethernet фрейма, его особенности, отправка и получение. Для облегчения описанных процессов применяется библиотеки открытого кода, для работы с низкоуровневыми примитивами.*

Работа с данными протокола Ethernet является основой любого программного обеспечения, работающего в сети, как локальной так и глобальной. Согласно модели OSI данный протокол работает на канальном уровне. Основными единицами данных, передаваемых по этому протоколу является фреймы. Они инкапсулируют в себе данные, передаваемые сетью а также добавляют служебную информацию, необходимую для корректного проработки этого фрейма устройствами в сети.

Язык Golang достаточно быстрый, и включает в себя средства работы с сетью. Однако, работа с низкоуровневыми данными, такими как фреймы Ethernet является достаточно сложной.

В данной статье описан процесс создания, отправки и получения Ethernet фрейма языке Golang с использованием вспомогательной библиотеки "mdlayher/ethernet".

Ключевые слова: *ethernet, golang, frame, ethernet frame, networking, фрейм, OSI.*

Summary. *This article discusses program tools for implementing network applications using low-level network primitives, such as Ethernet frames. In particular, an example of creating an Ethernet frame, its features, sending and receiving. To facilitate the described processes, open source libraries are used to work with low-level primitives.*

Working with Ethernet data is the basis of any software running on a network, both local and global. According to the OSI model, this protocol works at the channel level. The basic units of data transmitted under this protocol are frames. They encapsulate the data transmitted by the network and also add the service information necessary for the correct processing of this frame by devices on the network.

The Golang language is fairly fast, and includes networking tools. However, working with low-level data such as Ethernet frames is quite complex.

This article describes the process of creating, sending, and receiving an Ethernet frame in Golang using the mdlayher / ethernet helper library.

Key words: *ethernet, golang, frame, ethernet frame, networking, OSI.*

Постановка проблеми. Сучасне життя тісно пов'язане з мережею інтернет. Робота з мережевим трафіком потребує значної уваги. Сучасні мови програмування надають зручний функціонал для роботи з протоколами верхніх рівнів моделі OSI. В свою чергу, канальний рівень, що оперує фреймами Ethernet, відноситься до низькорівневих, тобто таких, що працюють напряму з API операційної системи. Робота з такими протоколами потребує більше зусиль, та часу. Щоб полегшити роботу з низькорівневими примітивами мережевих протоколів доцільно використати спеціалізовані бібліотеки, що інкапсулюють складну логіку викликів методів операційної системи.

Аналіз останніх досліджень та публікацій. Дослідження складають праці таких фахівців у галузі низькорівневого та мережевого програмування, як А. Микитишин [1], C. Hunt [2], A. Donovan [3], J. Newmarch [4], M. Tsoukalos [5].

Формулювання цілей статті. Дослідження механізму взаємодії з низькорівневими мережевими примітивами канального рівня моделі OSI мовою програмування Golang засобами спеціалізованих бібліотек.

Виклад основного матеріалу. Використовуючи пакет "ethernet", Ethernet фрейм може бути створений і використаний для подальших операцій з даними, що інкапсулюються у ньому.

Для прикладу буде використано фрейм, що включає в себе мінімальне корисне навантаження, а саме просту фразу "hello world" з

користувацьким типом фрейму. Він буде поширений в мережу з використанням широкомовної MAC адреси "FF:FF:FF:FF:FF:FF" [6].

Щоб створити фрейм достатньо оголосити структуру відповідного типу та проініціалізувати її поля.

```
// Фрейм, який буде відправлено по мережі
f := &ethernet.Frame{
   // Транслювати фрейм на всі клієнти в цьому сегменті мережі
   Destination: ethernet.Broadcast,
   // MAC адреса відправника
   Source: net.HardwareAddr{0xde, 0xad, 0xbe, 0xef, 0xde, 0xad},
   // Тип фрейму
   EtherType: 0xcccc,
   // Корисне навантаження
   Payload: []byte("hello world"),
}
```

Проте, перед відправкою фрейм необхідно закодувати в двійковий формат.

```
b, err := f.MarshalBinary()
if err != nil {
   log.Fatalf("failed to marshal frame: %v", err)
}
```

Зазвичай, операційна система чи мережевий інтерфейс власноруч обчислює контрольну суму фрейму Ethernet – FCS [6]. В незвичайних випадках, коли виконати це автоматично неможливо, викликається метод "MarshalFCS" Ethernet фрейму щоб додати до вже закодованого фрейму інформацію про контрольні суми даних [7].

Більшість мережевих додатків зазвичай побудовані на основі TCP чи UDP. Проте, оскільки робота з Ethernet фреймами відбувається на значно

нижчому рівні мережевого стеку, потрібно отримати спеціальні дозволи для використання деяких API напряму.

Щоб отримати можливість роботи на нижчому рівні необхідно використовувати "сирі сокети" ("пакетні сокети" в Linux). Дані низькорівневі сокети надають API для відправки та отримання Ethernet фреймів напряму, використовуючи підвищені привілеї від операційної системи.

В операційних системах Linux та BSD можна використовувати бібліотеку "github.com/mdlayher/raw" щоб надсилати та отримувати Ethernet фрейми через мережевий інтерфейс [8]. До прикладу, щоб надіслати попередньо створений фрейм, необхідно спочатку отримати посилання на мережевий інтерфейс, ініціювати сокет лиш потім записати дані в мережевий інтерфейс.

```
// Ініціалізація інтерфейсу eth0
ifi, err := net.InterfaceByName("eth0")
if err != nil {
   log.Fatalf("failed to open interface: %v", err)
}
// Ініціалізація сокету за користувацьким типом фрейму
c, err := raw.ListenPacket(ifi, 0xcccc)
if err != nil {
   log.Fatalf("failed to listen: %v", err)
}
defer c.Close()
// Запис даних в сокет за широкомовною адресою отримувача
addr := &raw.Addr{HardwareAddr: ethernet.Broadcast}
if _, err := c.WriteTo(b, addr); err != nil {
   log.Fatalf("failed to write frame: %v", err)
}
```

На іншій машині, необхідно створити схожий додаток, для прослуховування вхідних даних Ethernet фреймів за користувацьким типом фреймів.

```
// Ініціалізація інтерфейсу eth0
ifi, err := net.InterfaceByName("eth0")
if err != nil {
   log.Fatalf("failed to open interface: %v", err)
}
// Ініціалізація сокету за користувацьким типом фрейму
c, err := raw.ListenPacket(ifi, 0xcccc)
if err != nil {
   log.Fatalf("failed to listen: %v", err)
}
defer c.Close()
// Буфер вхідних даних розміром за замовчуванням для даного інтерфейсу
b := make([]byte, ifi.MTU)
var f ethernet.Frame
// Читання фреймів
for {
   n, addr, err := c.ReadFrom(b)
   if err != nil {
      log.Fatalf("failed to receive message: %v", err)
   }
   // Декодування вхідних фреймів в об'єкт Golang
   if err := (&f).UnmarshalBinary(b[:n]); err != nil {
      log.Fatalf("failed to unmarshal ethernet frame: %v", err)
   }
   // Перевірка отриманих даних як корисного навантаження
```

```
    log.Printf("[%s] %s", addr.String(), string(f.Payload))
}
```

Цього достатньо для реалізації мережевої комунікації засобами Ethernet фреймів. Як корисне навантаження можна записувати будь які дані, навіть реалізувати власний протокол передачі даних, що відповідно, працюватиме на вищому рівні.

Висновки з даного дослідження і перспективи подальших розвідок у даному напрямі. Низькорівневі мережеві примітиви, як Ethernet фрейми та "сирі сокети" є досить могутніми засобами комунікації. Використовуючи такі примітиви з'являється можливість повністю контролювати трафік, що надсилається та отримується додатками. Використовуючи спеціалізовані бібліотеки значно скорочується час на реалізацію мережевих додатків. Це дає змогу продуктам проектувати та розробляти власні мережеві протоколи, які повністю відповідатимуть потребам конкретного продукту. Подальше дослідження у даному напрямі дозволить отримати додаткові можливості використання низькорівневих примітивів, використання службових полів фрейму а також тегування фреймів за протоколом VLAN.

Література

1. Микитишин А. Г. Комп'ютерні мережі. / А. Г. Микитишин, М. М. Митник, П. Д. Стухляк, В. В. Пасічник // Львів: «Магнолія 2006», 2013. С. 256-258.
2. Hunt C. TCP/IP Network Administration / C. Hunt // O'Reilly Media 2002. C. 86-87.
3. Donovan A. A. The Go Programming Language / A. Donovan, B. Kernigan // Addison-Wesley Professional, 2015. C. 187.
4. Newmarch J. Network Programming with Go: Essential Skills for Using and Securing Networks / J. Newmarc // Apress 2017. C. 122.

5. Tsoukalos M. Mastering Go: Create Golang production applications using network libraries, concurrency, machine learning, and advanced data structures, 2nd Edition / M. Tsoukalos // Packt Publishing Ltd. 2019. C. 529-542.
6. Package Net provides a portable interface for network I/O, including TCP/IP, UDP, domain name resolution, and Unix domain sockets / Golang Documents. URL: https://golang.org/pkg/net (дата звернення: 21.01.21)
7. Package Ethernet implements marshaling and unmarshaling of IEEE 802.3 Ethernet II frames and IEEE 802.1Q VLAN tags / GitHub repository. URL: https://github.com/mdlayher/ethernet/ (дата звернення: 19.01.21)
8. Package Raw enables reading and writing data at the device driver level for a network interface / GitHub repository. URL: https://github.com/mdlayher/raw/ (дата звернення: 19.01.21)

References

1. Mykytyshyn A. Gh. Komp'juterni merezhi / A. Gh. Mykytyshyn, M. M. Mytnyk, P. D. Stukhljak, V. V. Pasichnyk // Ljviv: «Maghnolija 2006», 2013. S. 256-258.
2. Hunt C. TCP/IP Network Administration / C. Hunt // O'Reilly Media 2002. S. 86-87.
3. Donovan A. A. The Go Programming Language. / A. Donovan, B. Kernigan. // Addison-Wesley Professional, 2015. C. 187.
4. Newmarch J. Network Programming with Go: Essential Skills for Using and Securing Networks / J. Newmarc // Apress 2017. S. 122.
5. Tsoukalos M. Mastering Go: Create Golang production applications using network libraries, concurrency, machine learning, and advanced data structures, 2nd Edition / M. Tsoukalos // Packt Publishing Ltd. 2019. S. 529-542.

6. Package Net provides a portable interface for network I/O, including TCP/IP, UDP, domain name resolution, and Unix domain sockets / Golang Documents. URL: https://golang.org/pkg/net (data zvernennja: 21.01.21)
7. Package Ethernet implements marshaling and unmarshaling of IEEE 802.3 Ethernet II frames and IEEE 802.1Q VLAN tags / GitHub repository. URL: https://github.com/mdlayher/ethernet/ (data zvernennja: 19.01.21)
8. Package Raw enables reading and writing data at the device driver level for a network interface / GitHub repository. URL: https://github.com/mdlayher/raw/ (data zvernennja: 19.01.21)