Programowanie sieciowe

Transmisja ramek z pakietami warstw wyższych (biblioteka libnet)

Michał Kalewski

Instytut Informatyki
Politechnika Poznańska
ul. Piotrowo 2, 60–965 Poznań

mkalewski@cs.put.poznan.pl
http://www.cs.put.poznan.pl/mkalewski

Poznań · 25 marca 2019 r.

Plan wykładu

Biblioteka libnet

Biblioteka libnet (1/2)

Biblioteka libnet:

- ułatwia tworzenie i transmitowanie ramek (warstwy łącza danych) zawierających pakiety protokołów warstw wyższych;
- pozwala na manipulowanie wszystkimi parametrami w nagłówkach transmitowanych pakietów;
- zawiera funkcje tworzące pakiety kilkudziesięciu najpopularniejszych protokołów od warstwy drugiej do siódmej (m.in. Ethernet II, IEEE 802.2 i 802.2 SNAP, IEEE 802.1q, FDDI, ARP, IPv4, IPv6, ICMP, IGMP, OSPF, RIP, TCP, UDP, BGP, DNS);
- działa w systemach operacyjnych UNIX, GNU/Linux oraz Windows;
- dostępna jest pod następującym adresem internetowym: http://github.com/sam-github/libnet.

Biblioteka `libnet` (2/2)

Głównym przeznaczeniem biblioteki `libnet` jest tworzenie i transmitowanie ramek zawierających pakiety protokołów warstwy wyższych ze specyficznymi parametrami nagłówków definiowanymi przez programistę.

W konsekwencji, biblioteka ta jest powszechnie wykorzystywana w implementacjach narzędzi związanych z bezpieczeństwem sieci komputerowych, a jej uzupełnieniem jest przeważnie biblioteka `libpcap` (do odbioru ramek sieciowych).

Implementacja programów z użyciem biblioteki `libnet` w systemach operacyjny GNU/Linux wymaga instalacji pakietu `libnet-devel` (*development library for libnet*), a kompilacja takich programów (np. kompilatorem gcc) wymaga jawnego linkowania tej biblioteki (`-lnet` w przypadku kompilatora gcc).

Konstruowanie ramek z użyciem biblioteki `libnet`

Schemat konstruowania ramek (1/2)

Przy użyciu biblioteki `libnet` ramki z pakietami warstw wyższych tworzone są w częściach — protokołowi każdej z warstw odpowiada wywołanie osobnej funkcji.

Funkcje tworzące pakiety poszczególnych protokołów posiadają argumenty odpowiadające polom w nagłówkach tych pakietów.

Pakiety, które są kapsułkowane muszą być tworzone w **kolejności od warstwy najwyższej do warstwy najniższej**.

Przed rozpoczęciem pracy z biblioteką `libnet`, konieczne jest jej zainicjowanie (odpowiednią funkcją) i uzyskanie tym samym jej *kontekstu* (*libnet context*).

Komplementarnie, przed zakończeniem pracy z tą biblioteką niezbędna jest jej finalizacja (odpowiednią funkcją), która zwalania zasoby jej kontekstu.

Bezbłędne wywołanie funkcji tworzącej pakiet zwraca *identyfikator pakietu* (*protocol tag*, *ptag*), który jednoznacznie określa utworzony pakiet w kontekście biblioteki `libnet`.

Schemat konstruowania ramek (2/2)

Schematycznie, konstruowanie i transmisja ramek z użyciem biblioteki `libnet` wymaga następujących czynności:

1. Inicjacja biblioteki `libnet` funkcją `libnet_init()` (*alokacja pamięci*).
2. Utworzenie nagłówków pakietów od warstwy najwyższej do warstwy najniższej funkcjami `libnet_build_*()` lub `libnet_autobuild_*()`.
3. Wysłanie utworzonej ramki funkcją `libnet_write()`.
4. Wysłanie kolejnej ramki:
   - uaktualnienie/modyfikacja utworzonych pakietów z uzyskanymi wcześniej *identyfikatorami ptag* i powrót do punktu nr 3, lub
   - usunięcie stworzonej ramki z *kontekstu* biblioteki funkcją `libnet_clear_packet()` i utworzenie nowej ramki zaczynając od punktu nr 2.
5. Zakończenie pracy z biblioteką `libnet` poprzez wywołanie funkcji `libnet_destroy()` (*zwolnienie pamięci*).

Inicjacja biblioteki `libnet` — funkcja `libnet_init()`

```
libnet_t* libnet_init(int intinjection_type,
                      const char *device, char *err_buf);
```

Parametry:

`intinjection_type` określa typ transmisji; dostępne wartości to: `LIBNET_LINK`, `LIBNET_LINK_ADV`, `LIBNET_RAW4`, `LIBNET_RAW4_ADV`, `LIBNET_RAW6` oraz `LIBNET_RAW6_ADV`;

`device` określa interfejs sieciowy, który ma być wykorzystywany przez bibliotekę `libnet` — jego nazwa systemowa lub adres IP (w notacji dziesiętnej z kropkami) skonfigurowany na wybranym interfejsie; wartość `NULL` spowoduje, że biblioteka `libnet` wybierze pierwszy dostępny interfejs sieciowy w systemie;

`err_buf` wskazuje na bufor (o rozmiarze `LIBNET_ERRBUF_SIZE`), do którego zapisany zostanie komunikat o ewentualnym będzie wykonania funkcji.

Wartość zwracana: kontekst biblioteki lub `NULL` w przypadku błędu.

Inicjacja biblioteki `libnet` — typy transmisji

`LIBNET_LINK` Transmisja na warstwie łącza danych — wymaga utworzenia ramki sieciowej.

`LIBNET_LINK_ADV` Transmisja na warstwie łącza danych *w trybie zaawansowanym* — wymaga utworzenia ramki sieciowej.

`LIBNET_RAW4` Transmisja na warstwie sieciowej z użyciem protokołu IPv4 — wymaga utworzenia pakietu IPv4.

`LIBNET_RAW4_ADV` Transmisja na warstwie sieciowej z użyciem protokołu IPv4 *w trybie zaawansowanym* — wymaga utworzenia pakietu IPv4.

`LIBNET_RAW6` Transmisja na warstwie sieciowej z użyciem protokołu IPv6 — wymaga utworzenia pakietu IPv6.

`LIBNET_RAW6_ADV` Transmisja na warstwie sieciowej z użyciem protokołu IPv6 *w trybie zaawansowanym* — wymaga utworzenia pakietu IPv6.

</> Inicjacja biblioteki `libnet` — przykład

```
libnet_t *l;   /* libnet context */
char errbuf[LIBNET_ERRBUF_SIZE];

l = libnet_init(LIBNET_RAW4, argv[1], errbuf);
/* l = libnet_init(LIBNET_RAW4, NULL, errbuf); */
if (l == NULL) {
  fprintf(stderr, "libnet_init() failed: %s\n", errbuf);
  exit(EXIT_FAILURE);
}
...
libnet_destroy(l);
```


Funkcje tworzące pakiety

Istnieją dwa rodzaje funkcji tworzących pakiety: wymagające określenia wartości wszystkich pól nagłówka pakietu (`libnet_build_*()`) oraz wymagające określenia wartości tylko tych pól nagłówka pakietu, których biblioteka `libnet` nie może wyznaczyć na podstawie swojego kontekstu (`libnet_autobuild_*()`).

Funkcje tworzące pakiety zwracają identyfikator pakietu (*ptag*).

Identyfikator pakietu jest też zawsze ostatnim argumentem wywołania funkcji tworzących pakiety — wartość `0` (lub `LIBNET_PTAG_INITIALIZER`) oznacza tworzenie nowego pakietu, wartość odpowiadająca istniejącemu pakietowi oznacza jego modyfikowanie.

Sumy kontrolne, występujące w nagłówkach pakietów niektórych protokołów mogą być przez bibliotekę `libnet` obliczane automatycznie — w tym celu odpowiedni parametr wywołania powinien przyjmować wartość `0`.

zob. `libnet-functions.h(3)` oraz `libnet-headers.h(3)`

https://github.com/sam-github/libnet/blob/master/libnet/include/libnet/libnet-functions.h

Funkcje tworzące pakiety — przykłady (1/2)

```
ip = libnet_build_ipv4(
  LIBNET_IPV4_H + LIBNET_TCP_H + payload_s, /* length    */
  0,                                        /* TOS       */
  128,                                      /* IP ID     */
  0,                                        /* IP fragmentation */
  64,                                       /* TTL       */
  IPPROTO_TCP,                              /* protocol  */
  0,                                        /* checksum  */
  src_ip_addr,                              /* source IP */
  dst_ip_addr,                              /* destination IP */
  NULL,                                     /* payload   */
  0,                                        /* payload size */
  ln,                                       /* libnet context */
  0);                                       /* ptag      */
```


Funkcje tworzące pakiety — przykłady (2/2)

Funkcje `libnet_build_ethernet()` i `libnet_autobuild_ethernet()`:

```
eth = libnet_build_ethernet(
  dst_hw_addr,                      /* destination addr  */
  src_hw_addr,                      /* source addr       */
  ethertype,                        /* ethertype         */
  NULL,                             /* payload           */
  0,                                /* payload size      */
  ln,                               /* libnet context    */
  0);                               /* ptag              */
```

```
eth = libnet_autobuild_ethernet(
  dst_hw_addr,                      /* destination addr  */
  ETHERTYPE_ARP,                    /* ethertype         */
  ln);                              /* libnet context    */
```


Wybrane funkcje dodatkowe

```
struct libnet_ether_addr {
  uint8_t  ether_addr_octet[6];
};

struct libnet_ether_addr* libnet_get_hwaddr(libnet_t *l);

uint32_t libnet_get_ipaddr4(libnet_t *l);

uint32_t libnet_name2addr4(libnet_t *l, char *host_name,
                           uint8_t use_name);

char* libnet_addr2name4(uint32_t in, uint8_t use_name);

int libnet_write(libnet_t *l);

void libnet_destroy(libnet_t *l);
```


Interfejs zaawansowany

Interfejs zaawansowany (ang. *advanced mode*) biblioteki `libnet` umożliwia bezpośredni dostęp do bajtów utworzonej w jej kontekście ramki.

```
libnet_t *ln;
libnet_ptag_t ptag;
u_int8_t *frame;
u_int32_t frame_size;
u_int8_t *header;
u_int32_t header_size;
char errbuf[LIBNET_ERRBUF_SIZE];

ln = libnet_init(LIBNET_LINK_ADV, NULL, errbuf);
...
ptag = libnet_autobuild_ethernet(...);
...
libnet_adv_cull_header(ln, ptag, &header, &header_size);
libnet_adv_cull_packet(ln, &frame, &frame_size);
```

Przykład zastosowania — protokół ARP

</> Żądanie protokołu ARP (1/2)

```
1  libnet_t *ln;
2  u_int32_t target_ip_addr, src_ip_addr;
3  u_int8_t
4    bcast_hw_addr[6] = {0xff, 0xff, 0xff, 0xff, 0xff, 0xff},
5    zero_hw_addr[6]  = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
6  struct libnet_ether_addr* src_hw_addr;
7  char errbuf[LIBNET_ERRBUF_SIZE];
8
9  ln = libnet_init(LIBNET_LINK, NULL, errbuf);
10
11 src_ip_addr = libnet_get_ipaddr4(ln);
12 src_hw_addr = libnet_get_hwaddr(ln);
13 target_ip_addr = libnet_name2addr4(ln, argv[1],
14                                    LIBNET_RESOLVE);
```

cdn.

</> Żądanie protokołu ARP (2/2)

ciąg dalszy

```
15 libnet_autobuild_arp(
16   ARPOP_REQUEST,                 /* operation type       */
17   src_hw_addr->ether_addr_octet, /* sender hardware addr */
18   (u_int8_t*) &src_ip_addr,      /* sender protocol addr */
19   zero_hw_addr,                  /* target hardware addr */
20   (u_int8_t*) &target_ip_addr,   /* target protocol addr */
21   ln);                           /* libnet context       */
22
23 libnet_autobuild_ethernet(
24   bcast_hw_addr,                 /* ethernet destination */
25   ETHERTYPE_ARP,                 /* ethertype            */
26   ln);                           /* libnet context       */
27
28 libnet_write(ln);
29 libnet_destroy(ln);
```


</> Odpowiedź protokołu ARP (1/2)

```
1  libnet_t *ln;
2  u_int32_t target_ip_addr, zero_ip_addr;
3  u_int8_t
4    bcast_hw_addr[6] = {0xff, 0xff, 0xff, 0xff, 0xff, 0xff},
5    zero_hw_addr[6]  = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
6  struct libnet_ether_addr* src_hw_addr;
7  char errbuf[LIBNET_ERRBUF_SIZE];
8
9  ln = libnet_init(LIBNET_LINK, NULL, errbuf);
10
11 src_hw_addr = libnet_get_hwaddr(ln);
12 target_ip_addr = libnet_name2addr4(ln, argv[1],
13                                    LIBNET_RESOLVE);
14 zero_ip_addr = libnet_name2addr4(ln, "0.0.0.0",
15                                  LIBNET_DONT_RESOLVE);
```

cdn.

</> Odpowiedź protokołu ARP (2/2)

ciąg dalszy

```
16  libnet_autobuild_arp(
17    ARPOP_REPLY,                    /* operation type       */
18    src_hw_addr->ether_addr_octet,  /* sender hardware addr */
19    (u_int8_t*) &target_ip_addr,    /* sender protocol addr */
20    zero_hw_addr,                   /* target hardware addr */
21    (u_int8_t*) &zero_ip_addr,      /* target protocol addr */
22    ln);                            /* libnet context       */
23
24  libnet_autobuild_ethernet(
25    bcast_hw_addr,                  /* ethernet destination */
26    ETHERTYPE_ARP,                  /* ethertype            */
27    ln);                            /* libnet context       */
28
29  libnet_write(ln);
30  libnet_destroy(ln);
```


Podsumowanie

Podsumowanie

- Biblioteka `libnet`:
  - schemat konstruowania ramek;
  - kontekst oraz identyfikator pakietu (*ptag*);
  - funkcje tworzące pakiety (`libnet_build_*()` oraz `libnet_autobuild_*()`);
  - wybrane funkcje dodatkowe.
- Przykłady zastosowania: żądania i odpowiedzi protokołu ARP.

Pytania i zadania

Pytania i zadania — praca własna

1. Zapoznaj się z listą wszystkich funkcji tworzących pakiety biblioteki `libnet`, *zob.* `libnet-functions.h(3)`.
2. Zapoznaj się z przykładami zastosowania biblioteki `libnet` dostępnymi pod następującym adresem internetowym: https://github.com/sam-github/libnet/tree/master/libnet/sample.
3. W jakim celu system operacyjny może wykonać transmisję żądania ARP z własnym adresem IP?

Literatura dodatkowa:

Libnet 1.1 tutorial: https://repolinux.wordpress.com/category/libnet/

Dziękuję za uwagę!