Προγραμματισμός Δικτύων – Ε-01

9η Διάλεξη

Διδάσκων: Νίκος Ντάρμος

<ntarmos@cs.uoi.gr>
[http://www.cs.uoi.gr/~ntarmos/Courses/NetworkProgramming/]

Τμήμα Πληροφορικής
Πανεπιστήμιο Ιωαννίνων

Στο σημερινό μάθημα...

- Broadcast/multicast.
- SCTP multihoming/multistreaming.
- RAW sockets.
- Πρόσβαση στο επίπεδο διασύνδεσης (datalink).

Broadcast & Multicast

Εισαγωγή

- Ως τώρα είδαμε μεθόδους unicast επικοινωνίας:
  Ένας αποστολέας επικοινωνεί με έναν παραλήπτη.
- Ωστόσο δεν αρκεί αυτό για όλες τις εφαρμογές.
  - «Ανακάλυψη πόρων» (resource discovery): ARP, DHCP.
  - Αποστολή πληροφοριών προς όλους τους υπολογιστές ενός υποδικτύου: NTP, routed.
  - Ταυτόχρονη αποστολή δεδομένων προς ένα σύνολο πελατών: ζωντανή ροή πολυμεσικού περιεχομένου.
- Επικοινωνία ενός με όλους $\Rightarrow$ broadcast.
- Επικοινωνία ενός με πολλούς $\Rightarrow$ multicast.
  - Μπορεί να υλοποιηθεί στο επίπεδο διαδικτύου και παρακάτω ή στο επίπεδο εφαρμογής (application-level broadcast/multicast). Εδώ ασχολούμαστε με την 1^{η} περίπτωση.
- Επικοινωνία ενός με τουλάχιστον έναν από πολλούς $\Rightarrow$ anycast.

Broadcasting

Broadcasting

- Broadcast: ταυτόχρονη επικοινωνία ενός αποστολέα με όλους τους κόμβους ενός υποδικτύου.
- Γιατί είναι διαφορετικό από ταυτόχρονη αποστολή σε N διαφορετικές συνδέσεις, σε δίκτυο N κόμβων;
  1. «Σπατάλη» πόρων δικτύου.
  2. Κι αν δεν ξέρουμε τη διεύθυνση του παραλήπτη;
- Υποστηρίζεται μόνο από το UDP και sockets τύπου `SOCK_DGRAM` και `SOCK_RAW`.
- Δεν υποστηρίζεται από το IPv6.

Broadcasting

- Υπάρχουν 4 «είδη» broadcast:
  1. Broadcast κατευθυνόμενο προς ένα συγκεκριμένο υποδίκτυο.
     - Διεύθυνση παραλήπτη: <διεύθυνση δικτύου>.<διεύθυνση υποδικτύου>.255.
     - Το πιο κοινό είδος broadcast σήμερα. Πολλές φορές τα άλλα μεταφράζονται σε αυτό.
  2. Broadcast κατευθυνόμενο προς όλα τα υποδίκτυα ενός δικτύου.
     - Διεύθυνση παραλήπτη: <διεύθυνση δικτύου>.255.255
     - Χρησιμοποιείται ελάχιστα έως καθόλου.
  3. Broadcast κατευθυνόμενο προς ένα ολόκληρο δίκτυο.
     - Διεύθυνση παραλήπτη: <διεύθυνση δικτύου>.255
     - Για δίκτυα χωρίς υποδίκτυα. Πρακτικά δεν υπάρχουν εδώ και αρκετό καιρό.
  4. Broadcast περιορισμένης προώθησης.
     - Διεύθυνση παραλήπτη: 255.255.255.255
     - Χρησιμοποιείται όταν δεν υπάρχουν καθόλου ρυθμίσεις δικτύου (π.χ. κατά το DHCP/BOOTP).

Broadcasting

- Για να αποστείλουμε δεδομένα με broadcast πρέπει:
  1. Να ενεργοποιήσουμε την επιλογή `SO_BROADCAST` επιπέδου υλοποίησης sockets (`SOL_SOCKET`).
  2. Να ορίσουμε ως διεύθυνση παραλήπτη την κατάλληλη διεύθυνση broadcast και τον αριθμό θύρας στον οποίο θα περιμένουν οι πελάτες τα δεδομένα.
- Ορισμένα λειτουργικά συστήματα δεν επιτρέπουν τον κατακερματισμό broadcast πακέτων.
  - Όπως και στην περίπτωση «μεγάλων» datagrams, επιστρέφεται σφάλμα `EMSGSIZE`.
  - Για μεταφερσιμότητα, το μέγεθος του datagram πρέπει να είναι $\leq$1472 bytes.
- Τα πακέτα broadcast δεν προωθούνται πέρα από τα όρια του δικτύου στο οποίο δημιουργούνται.

Multicasting

Multicasting

- Multicast: ταυτόχρονη επικοινωνία ενός αποστολέα με ένα σύνολο κόμβων δικτύου.
- Γιατί είναι διαφορετικό από ταυτόχρονη αποστολή σε N διαφορετικές συνδέσεις (multi-unicast);
- Υποστηρίζεται μόνο από το UDP και sockets τύπου `SOCK_DGRAM` και `SOCK_RAW`.
  - Και το SCTP προσφέρει «multicasting» αλλά ανάμεσα σε ένα ζεύγος κόμβων μόνο.
- Υποστηρίζεται τόσο από το IPv4 όσο και από το IPv6.
- Δεν υπάρχει περιορισμός προώθησης στο διαδίκτυο (πρέπει όμως να είναι κατάλληλα ρυθμισμένοι οι δρομολογητές ή χρήση του MBone).
- Δεν υπάρχει πρόβλημα με τον κατακερματισμό των πακέτων.

Multicasting

- Ορίζονται «ειδικού τύπου» διευθύνσεις για επικοινωνία με multicasting.
- Η ανάθεση γίνεται από την IANA.
  - IPv4: 224.0.0.0/4.
    `http://www.iana.org/assignments/multicast-addresses/`
  - IPv6: ff00::/8.
    `http://www.iana.org/assignments/ipv6-multicast-addresses/`
- Η διεύθυνση αναλύεται σε τμήματα:
  - IPv4:
    - 4 bits «πρόθεμα δικτύου».
    - 28 bits «αναγνωριστικό ομάδας» (group id).
    - Κατά την μετατροπή σε διευθύνσεις Ethernet χρησιμοποιούνται μόνο τα 23 λιγότερο σημαντικά από τα 28 bits με πρόθεμα 01:00:5e $\Rightarrow$ δεν έχουμε 1-προς-1 αντιστοιχία.
  - IPv6:
    - 8 bits «πρόθεμα δικτύου».
    - 4 bits επιλογών.
    - 4 bits «scope» διευθύνσεων (πόσο «μακριά» θα φτάσει κάθε πακέτο multicast).
    - 112 bits «αναγνωριστικό ομάδας».
    - Κατά την μετατροπή σε διευθύνσεις Ethernet χρησιμοποιούνται μόνο τα 32 λιγότερο σημαντικά από τα 112 bits με πρόθεμα 33:33 $\Rightarrow$ δεν έχουμε 1-προς-1 αντιστοιχία.

Multicasting

- Ορίζονται 5 επιλογές sockets (`setsockopt(2)`/`getsockopt(2)`):
  - `IP_ADD_MEMBERSHIP`/`IPV6_ADD_MEMBERSHIP`: προσθήκη της διεργασίες στην ομάδα.
  - `IP_DROP_MEMBERSHIP`/`IPV6_DROP_MEMBERSHIP`: αφαίρεση της διεργασίας από την ομάδα.
  - `IP_MULTICAST_IF`/`IPV6_MULTICAST_IF`: επιλογή της διεπαφής δικτύου από την οποία θα στέλνονται πακέτα (δείτε τις `getifaddrs(3)`/`freeifaddrs(3)` και `if_nametoindex(3)`).
  - `IP_MULTICAST_TTL`/`IPV6_MULTICAST_HOPS`: επιλογή μέγιστου ορίου βημάτων για την προώθηση τον εξερχόμενων πακέτων.
  - `IP_MULTICAST_LOOP`/`IPV6_MULTICAST_LOOP`: επιλογή παράδοσης εξερχόμενων δεδομένων και στην διεργασία–αποστολέα.
- Οι πρώτες δύο επιλογές επηρεάζουν την παραλαβή ενώ οι επόμενες τρεις την αποστολή πακέτων.
- Για να στείλει μία διεργασία ένα πακέτο multicast αρκεί απλά να επιλέξει την κατάλληλη διεύθυνση παραλήπτη.
  - Ο πυρήνας θα επιλέξει αυτόματα την διεπαφή, θα θέσει όριο βημάτων $=1$ και θα παραδώσει τα δεδομένα και στη διεργασία–αποστολέα.
- Για να λάβει μία διεργασία δεδομένα από multicasting πρέπει:
  1. Να «εγγραφεί» στην κατάλληλη «multicast ομάδα».
  2. Να δημιουργήσει ένα UDP socket και να το κάνει `bind(2)` στην κατάλληλη θύρα.

SCTP multihoming/multistreaming

SCTP multihoming/multistreaming

- Το πρωτόκολλο μεταφοράς SCTP προσφέρει επίσης κάποιου είδους «multicasting» μέσω της υποστήριξης **multihoming**.
  - Multihoming: Κάθε άκρο σύνδεσης («συσχέτιση» – association) μπορεί να έχει παραπάνω από μία διευθύνσεις, καθεμιά εκ των οποίων να αντιστοιχεί σε διαφορετική διεπαφή δικτύου, υλοποίηση επιπέδου διασύνδεσης, φυσικό δίκτυο, κτλ.
  - Κάθε πακέτο στέλνεται από μία διεπαφή του αποστολέα προς μια διεπαφή του παραλήπτη. Αν η αποστολή αποτύχει, το πρωτόκολλο δοκιμάζει το επόμενο ζεύγος διεπαφών, κοκ.
  - Έτσι κάθε πακέτο μπορεί να ακολουθήσει εντελώς διαφορετική διαδρομή και η σύνδεση να είναι αξιόπιστη ακόμα κι αν κάποιες από τις φυσικές συνδέσεις χαθούν!
- Το πρωτόκολλο μεταφοράς SCTP προσφέρει επίσης υποστήριξη για **multistreaming**.
  - Multistreaming: Κάθε σύνδεση ανάμεσα σε δύο κόμβους μπορεί να αποτελείται από περισσότερες της μίας ροές δεδομένων.
  - Παράδειγμα: μία ροή για δεδομένα ελέγχου, μία ροή για αιτήσεις, μία ροή για απαντήσεις.
  - Το πρωτόκολλο διαχειρίζεται την σειριοποίηση, το buffering, την αποστολή και την ανασύνθεση της πληροφορίας.

SCTP multihoming/multistreaming

- Για να αποκτήσουμε πρόσβαση στις δυνατότητες αυτές του SCTP παρέχονται ειδικές συναρτήσεις.
  - `int sctp_bindx(int s, struct sockaddr *addrs, int num, int type)`: Προσθέτει ή αφαιρεί μία λίστα από `num` διευθύνσεις που βρίσκονται στον πίνακα `addrs`, στις ήδη υπάρχουσες για το socket `s`, ανάλογα με το αν το τελευταίο όρισμα είναι `SCTP_BINDX_ADD_ADDR` ή `SCTP_BINDX_DEL_ADDR`.
  - `int sctp_connectx(int s, struct sockaddr *addrs, int num, sctp_assoc_t *sa)`: λειτουργεί όπως και η `connect(2)` με τη διαφορά ότι υποστηρίζει σύνδεση και στις `num` διευθύνσεις του πίνακα `addrs` στο socket `s`. Αν επιτύχει, το πεδίο `sa` θα έχει το αναγνωριστικό της SCTP «συσχέτισης» (association).
  - `int sctp_opt_info(int s, sctp_assoc_t, int opt, void *arg, socklen_t *size)`: αντίστοιχη της `getsockopt(2)` για το SCTP, αν η `getsockopt(2)` δεν υποστηρίζει το SCTP.

SCTP multihoming/multistreaming

- Για την αποστολή και λήψη δεδομένων πάνω από τέτοιες «συσχετίσεις» SCTP παρέχονται οι συναρτήσεις:
  - `ssize_t sctp_send(int sd, const void *msg, size_t len, const struct sctp_sndrcvinfo *sinfo, int flags)`
  - `ssize_t sctp_sendx(int sd, const void *msg, size_t len, struct sockaddr *addrs, int addrcnt, const struct sctp_sndrcvinfo *sinfo, int flags)`
  - `size_t sctp_sendmsg(int s, const void *msg, size_t len, const struct sockaddr *to, socklen_t tolen, uint32_t ppid, uint32_t flags, uint16_t stream_no, uint32_t timetolive, uint32_t context)`
  - `ssize_t sctp_sendmsgx(int s, const void *msg, size_t len, const struct sockaddr *to, int addrcnt, uint32_t ppid, uint32_t flags, uint16_t stream_no, uint32_t timetolive, uint32_t context)`
  - `ssize_t sctp_recvmsg(int s, void *msg, size_t len, struct sockaddr * restrict from, socklen_t * restrict fromlen, struct sctp_sndrcvinfo *sinfo, int *flags)`

SCTP multihoming/multistreaming

- Η δομή `sctp_sndrcvinfo` είναι ως εξής:

```
struct sctp_sndrcvinfo {
    u_int16_t    sinfo_stream;      /* Stream number */
    u_int16_t    sinfo_ssn;         /* Stream Sequence Number
                                       (ordered recv only) */
    u_int16_t    sinfo_flags;       /* Message flags */
    u_int32_t    sinfo_ppid;        /* Opaque PPID field */
    u_int32_t    sinfo_context;     /* context field */
    u_int32_t    sinfo_timetolive;  /* TTL for PR-SCTP (send only) */
    u_int32_t    sinfo_tsn;         /* Transport sequence number
                                       (recv only) */
    u_int32_t    sinfo_cumtsn;      /* Cumul. ACK point (recv only) */
    sctp_assoc_t sinfo_assoc_id;    /* Association id of the peer */
};
```

- **Προσοχή:** Το SCTP είναι αρκετά «νέο» πρωτόκολλο και δεν υποστηρίζεται πλήρως από όλα τα συστήματα. Συμβουλευτείτε τα man pages του συστήματός σας!

SCTP multihoming/multistreaming παράδειγμα (1/2)

```
1  static const int LOCALTIME_STREAM = 0;
2  static const int GMT_STREAM = 1;
3  static const int MAX_BUFFER = 255;
4
5  int main(int argc, char ** argv)
6  {
7      int sd, cd, ret;
8      struct addrinfo *info, *cur, hints;
9      char buffer[MAX_BUFFER+1];
10     time_t currentTime;
11
12     memset(&hints, 0, sizeof(hints));
13     hints.ai_family = AF_INET;
14     hints.ai_flags = AI_ADDRCONFIG | AI_PASSIVE;
15     hints.ai_socktype = SOCK_STREAM;
16     if ((ret = getaddrinfo(NULL, argv[1], &hints, &info)) < 0) {
17         fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(ret));
18         return 1;
19     }
20
21     sd = socket(AF_INET, SOCK_STREAM, IPPROTO_SCTP);
22     for (cur = info; cur; cur = cur->ai_next)
23         if ((ret = sctp_bindx(sd, (struct sockaddr *)cur->ai_addr, 1,
               SCTP_BINDX_ADD_ADDR)) < 0)
24             perror("sctp_bindx");
25     listen(sd, 5);
```

SCTP multihoming/multistreaming παράδειγμα (2/2)

```
27      while (1) {
28          cd = accept(sd, NULL, NULL);
29          currentTime = time(NULL);
30
31          /* Send local time on stream 0 (local time stream) */
32          snprintf(buffer, MAX_BUFFER, "%s\n", ctime(&currentTime));
33          if ((ret = sctp_sendmsg(cd, buffer, strlen(buffer), NULL, 0, 0, 0,
                LOCALTIME_STREAM, 0, 0 )) < 0)
34              perror("sctp_sendmsg");
35
36          /* Send GMT on stream 1 (GMT stream) */
37          snprintf(buffer, MAX_BUFFER, "%s\n", asctime(gmtime(&currentTime)));
38          if ((ret = sctp_sendmsg(cd, buffer, strlen(buffer), NULL, 0, 0, 0,
                GMT_STREAM, 0, 0 )) < 0)
39              perror("sctp_sendmsg");
40
41          close(cd);
42      }
43      close(sd);
44      freeaddrinfo(info);
45      return 0;
46  }
```

Επικοινωνίες χαμηλού επιπέδου

RAW sockets

RAW sockets

- Πώς μπορούμε να στείλουμε πακέτα πρωτοκόλλων της οικογένειας IP εκτός των TCP/UDP/SCTP;
  - Π.χ.: πώς υλοποιείται το ping (ICMP);
  - Π.χ.: μπορούμε να κατασκευάσουμε ένα πακέτο όπως ακριβώς το θέλουμε εμείς;
  - Π.χ.: θα μπορούσαμε να επεκτείνουμε τα υπάρχοντα πρωτόκολλα (π.χ. ICMP) με νέου τύπου πακέτα, χωρίς να παρέμβουμε στον πυρήνα;
- Το BSD sockets API προσφέρει τον τύπο sockets `SOCK_RAW` ο οποίος επιτρέπει πρόσβαση στη δομή και τις κεφαλίδες των πακέτων.
- Χρησιμοποιούνται όταν θέλουμε:
  - Να επικοινωνήσουμε μέσω των πρωτοκόλλων ICMPv4, IGMPv4 και ICMPv6.
  - Να υλοποιήσουμε λειτουργικότητα νέων πρωτοκόλλων πάνω από το IP τα οποία όμως δεν γνωρίζει ο πυρήνας.
  - Να κατασκευάσουμε εκ του μηδενός τα πακέτα μας, συμπεριλαμβανομένης της κεφαλίδας IP.
- Είναι διαθέσιμα μόνο στον *superuser* (root).

Δημιουργία

RAW sockets

Δημιουργία

- Δημιουργούμε ένα RAW socket με:
  ```
  int sd = socket(AF_INET, SOCK_RAW, IPPROTO_XXX);
  ```
  όπου `IPPROTO_XXX` κάποιος από τους υπάρχοντες τύπους πρωτοκόλλων.
- Για να επεξεργαστούμε την κεφαλίδα IP πρέπει να ενεργοποιήσουμε την επιλογή `IP_HDRINCL`, διαφορετικά η κεφαλίδα συμπληρώνεται από τον πυρήνα του λειτουργικού συστήματος:
  ```
  int on = 1;
  setsockopt(sd, IPPROTO_IP, IP_HDRINCL, &on, sizeof(on));
  ```
- Σε επίπεδα κάτω του διαδικτύου δεν υπάρχει η έννοια της θύρας.
  - Η `bind(2)` θέτει μόνο την IP διεύθυνση από την οποία θα στέλνονται τα πακέτα αν δεν είναι ενεργοποιημένη η επιλογή `IP_HDRINCL` (διαφορετικά ο πυρήνας επιλέγει αυτόματα)
  - Η `connect(2)` θέτει μόνο την IP διεύθυνση προς την οποία θα στέλνονται τα πακέτα (οπότε μπορούμε να χρησιμοποιούμε `write(2)` ή `send(2)` αντί για `sendto(2)` ή `sendmsg(2)`).

Αποστολή/λήψη δεδομένων

RAW sockets

Αποστολή δεδομένων

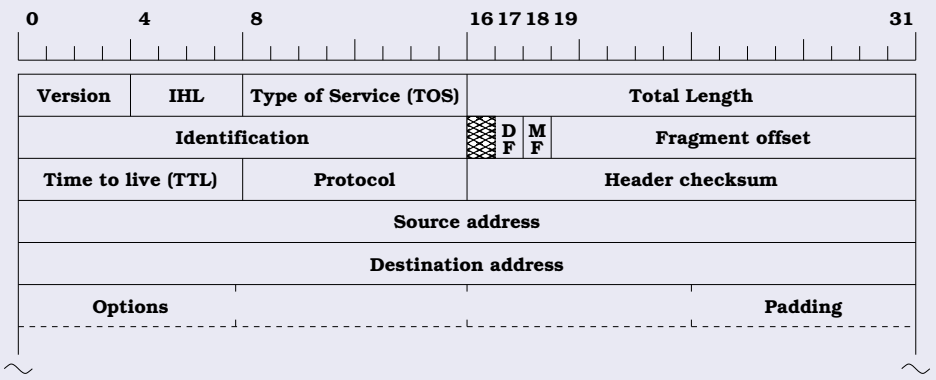
- Για την αποστολή δεδομένων χρησιμοποιούμε είτε τις `sendto(2)`/`sendmsg(2)`, είτε τις `write(2)`/`writev(2)`/`send(2)` αν έχουμε πρώτα καλέσει `connect(2)`.
- Αν δεν έχουμε ενεργοποιήσει την επιλογή `IP_HDRINCL`, τα δεδομένα που δίνουμε στις κλήσεις αυτές πρέπει να είναι αυτά που θα τοποθετηθούν αμέσως μετά την κεφαλίδα IP.
- Αν έχουμε ενεργοποιήσει την επιλογή `IP_HDRINCL`, τα δεδομένα που δίνουμε στις κλήσεις αυτές πρέπει να συμπεριλαμβάνουν και την κεφαλίδα IP.
- Αν τα πακέτα υπερβαίνουν σε μέγεθος το MTU του δικτύου, ο πυρήνας αναλαμβάνει τον κατακερματισμό τους.
- Σε κάθε περίπτωση, είναι ευθύνη του προγραμματιστή τα δεδομένα να υλοποιούν τη δομή του πρωτοκόλλου που χρησιμοποιείται.

RAW sockets

Αποστολή δεδομένων

- Αν κατασκευάζουμε την κεφαλίδα IP:
  - Το πεδίο `identification` παίζει το ρόλο «θύρας». Μπορεί να τεθεί στην τιμή 0, οπότε το συμπληρώνει ο πυρήνας, ή σε κάποια άλλη τιμή που γνωρίζουμε εμείς.
  - Το πεδίο header checksum υπολογίζεται από τον πυρήνα. Ωστόσο είναι ευθύνη του προγραμματιστή να υπολογίσει τα checksums των κεφαλίδων των υπόλοιπων πρωτοκόλλων (π.χ. ICMP, IGMP κτλ.)
  - Το byte ordering εξαρτάται από το λειτουργικό σύστημα.
    - Στο Linux, στο OpenBSD και στο NetBSD, όλα τα πεδία είναι σε network byte order.
    - Στα υπόλοιπα BSD-derived συστήματα (π.χ. FreeBSD, DragonflyBSD) όλα τα πεδία είναι σε network byte order, εκτός από τα total length και fragment offset που είναι σε host byte order.

Κεφαλίδα πακέτου IPv4



RAW sockets

Αποστολή δεδομένων

- Για πρωτόκολλα της οικογένειας IPv6 (π.χ. ICMPv6):
  - Όλα τα πεδία είναι σε network byte order.
  - Δεν ορίζεται η επιλογή `IP_HDRINCL`.
    - Οι επιλογές και τα διάφορα πεδία της κεφαλίδας είναι προσβάσιμα μέσω επιλογών sockets ή δεδομένων ελέγχου (βλπ. `sendmsg(2)`).
    - Για να διαβάσουμε ή να γράψουμε πλήρες πακέτο, πρέπει να κατεβούμε στο επίπεδο διασύνδεσης.
  - Για πακέτα `ICMPv6` το checksum υπολογίζεται πάντα από τον πυρήνα. Για τα υπόλοιπα πρωτόκολλα, μπορούμε να επιλέξουμε να υπολογίζει το σύστημα τα checksums με την επιλογή `IPV6_CHECKSUM`:

```
int offset = 2; // Σε ποιο σημείο θα αποθηκευθεί το checksum.
setsockopt(sd, IPPROTO_IPV6, IPV6_CHECKSUM, &offset,
sizeof(int));
```

RAW sockets

Λήψη δεδομένων

- Κατά την λήψη δεδομένων, ο πυρήνας είναι υπεύθυνος για την ανασύνθεση κατακερματισμένων πακέτων.
- Για να παραδοθεί ένα πακέτο σε ένα RAW socket, υπάρχουν οι εξής κανόνες:
  - Το πακέτο δεν πρέπει να είναι TCP ή UDP. Για να χειριστούμε την κεφαλίδα τέτοιων πακέτων πρέπει να κατεβούμε στο επίπεδο διασύνδεσης.
  - Σχεδόν όλα τα ICMP πακέτα παραδίδονται, αφού ο πυρήνας ολοκληρώσει την επεξεργασία τους. Σε BSD-derived συστήματα, δεν παραδίδονται πακέτα τύπου echo-request, timestamp-request, netmask-request.
  - Παραδίδονται όλα τα πακέτα IGMP, αφού ο πυρήνας ολοκληρώσει την επεξεργασία τους.
  - Παραδίδονται όλα τα πακέτα IP τα οποία έχουν κάποια τιμή που δεν γνωρίζει ο πυρήνας στο πεδίο protocol.

RAW sockets

Λήψη δεδομένων

- Αν υπάρχουν περισσότερα από ένα RAW sockets ανοιχτά σε ένα σύστημα, τότε για κάθε εισερχόμενο πακέτο IP ο πυρήνας ελέγχει όλα τα sockets αυτά και το πακέτο παραδίδεται σε όλα τα κατάλληλα τέτοια sockets.
- Για να είναι ένα RAW socket «κατάλληλο», πρέπει:
  1. Αν έχει οριστεί κάποιο πρωτόκολλο για το socket (3ο όρισμα της `socket(2)`), το εισερχόμενο πακέτο να είναι του ίδιου πρωτοκόλλου.
  2. Αν έχει οριστεί τοπική διεύθυνση IP για το socket (με χρήση της `bind(2)`), το εισερχόμενο πακέτο να έχει την ίδια διεύθυνση παραλήπτη.
  3. Αν έχει οριστεί απομακρυσμένη διεύθυνση IP για το socket (με χρήση της `connect(2)`), το εισερχόμενο πακέτο να έχει την ίδια διεύθυνση αποστολέα.
- Αν κατά τη δημιουργία του socket δεν ορίσαμε τίποτα από τα παραπάνω, τότε αυτό λαμβάνει ένα αντίγραφο κάθε πακέτου IP που ο πυρήνας παραδίδει σε RAW sockets.

RAW sockets

Λήψη δεδομένων

- Για πρωτόκολλα οικογένειας IPv4, το πακέτο που παραδίδεται στην εφαρμογή συμπεριλαμβάνει και την κεφαλίδα IP.
- Για πρωτόκολλα οικογένειας IPv6, το πακέτο δεν συμπεριλαμβάνει την κεφαλίδα και τις κεφαλίδες επέκτασης.
- Για το byte ordering ισχύει ότι και στην περίπτωση αποστολής δεδομένων.
  - Στα BSD-derived συστήματα και το πεδίο identification είναι σε host byte order.

RAW sockets

Λήψη δεδομένων

- Καθώς στο IPv6 το πρωτόκολλο ICMPv6 εμπεριέχει και τη λειτουργικότητα των ARP και IGMPv6, απαιτείται κάποιος μηχανισμός επιλογής των πακέτων που ενδιαφέρουν την εφαρμογή.
- Έτσι, καθορίζεται ένα «φίλτρο» τύπου `struct icmp6_filter` για sockets αυτού του είδους.
  - Ορίζεται και ελέγχεται με κλήσεις των `setsockopt(2)` και `getsockopt(2)` στο επίπεδο `IPPROTO_ICMPV6` και για την επιλογή `ICMP6_FILTER`.
  - Για τη διαχείριση του φίλτρου ορίζονται οι εξής μακροεντολές:
    - `void ICMP6_FILTER_SETPASSALL(filt);`
    - `void ICMP6_FILTER_SETBLOCKALL(filt);`
    - `void ICMP6_FILTER_SETPASS(msgtype, filt);`
    - `void ICMP6_FILTER_SETBLOCK(msgtype, filt);`
    - `void ICMP6_FILTER_WILLPASS(msgtype, filt);`
    - `void ICMP6_FILTER_WILLBLOCK(msgtype, filt);`

    όπου `filt` ένας δείκτης σε δομή `struct icmp6_filter` και `msgtype` ακέραιος (0 – 255) που δηλώνει τον τύπο μηνύματος ICMP.

Επιλογές

RAW sockets

Επιλογές

- Εκτός από την επιλογή `IP_HDRINCL`, ορίζονται επίσης οι παρακάτω (επιπέδου `IPPROTO_IP`), οι οποίες όμως δε μπορούν να χρησιμοποιηθούν παράλληλα με την `IP_HDRINCL`:
  - `IP_OPTIONS`: επιτρέπει την διαχείριση των επιλογών της κεφαλίδας IP.
  - `IP_TOS`: επιτρέπει την διαχείριση του πεδίου «type of service».
  - `IP_TTL`: επιτρέπει την διαχείριση του ελάχιστου «time to live» που πρέπει να έχει ένα πακέτο για να γίνει αποδεκτό.
  - `IP_DONTFRAG`: επιτρέπει την διαχείριση του πεδίου «don't fragment».
  - `IP_RECVDSTADDR`: ζητά την επιστροφή της διεύθυνσης παραλήπτη στα δεδομένα ελέγχου της `recvmsg(2)`.
  - `IP_RECVIF`: ζητά την επιστροφή των στοιχείων της διεπαφής δικτύου από την οποία έγινε λήψη του πακέτου στα δεδομένα ελέγχου της `recvmsg(2)`.
  - `IP_RECTTL`: ζητά την επιστροφή της τιμής του πεδίου TTL στα δεδομένα ελέγχου της `recvmsg(2)`.

Παράδειγμα: ping

RAW sockets παράδειγμα: ping (1/4)

```
1  unsigned short ip_cksum(unsigned short *buf, int len);
2  void _terminate(struct addrinfo **info, char **buf, int *sd, int ret) {
3      if (*sd >= 0) { close(*sd); *sd = -1; }
4      if (*buf)     { free(*buf); *buf = NULL; }
5      if (*info)    { freeaddrinfo(*info); *info = NULL; }
6      exit(ret);
7  }
8  #define terminate(ret) _terminate(&info, &recvbuf, &sd, ret)
9
10 int main(int argc, char ** argv) {
11     int sd, ret, ttl, maxp, ip_hlen, packet_id, replied, timedout, sockerrno;
12     char *recvbuf = NULL;
13     socklen_t sslen;
14     struct sockaddr_storage ss;
15     struct addrinfo hints, *info = NULL;
16     struct ip *ip_ptr;
17     struct icmp *icmp_ptr;
18     struct timeval tv;
19
20     if (argc != 2) {
21         fprintf(stderr, "Usage: %s <hostname>\n", argv[0]);
22         return 1;
23     }
24
25     sd = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
26     sockerrno = errno;
27     setuid(getuid()); /* No root rights needed after this */
28     if (sd < 0) {
29         sockerrno = ret;
30         perror("socket");
31         terminate(1);
32     }
```

RAW sockets παράδειγμα: ping (2/4)

```
34      memset(&hints, 0, sizeof(hints));
35      hints.ai_family = AF_INET;
36      hints.ai_socktype = SOCK_STREAM;
37      hints.ai_flags = AI_ADDRCONFIG | AI_CANONNAME;
38      if ((ret = getaddrinfo(argv[1], NULL, &hints, &info)) < 0) {
39          fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(ret));
40          terminate(1);
41      }
42
43      ttl = 20;                       /* Look at most 20 hops away */
44      setsockopt(sd, IPPROTO_IP, IP_TTL, &ttl, sizeof(ttl));
45      tv.tv_sec = 2; tv.tv_usec = 0; /* Wait for up to 2'' */
46      setsockopt(sd, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv));
47      maxp = IP_MAXPACKET + 128;
48      setsockopt(sd, SOL_SOCKET, SO_RCVBUF, &maxp, sizeof(maxp));
49
50      packet_id = getpid() & 0xFFFF;
51      icmp_ptr = (struct icmp *)calloc(1, sizeof(struct icmp));
52      icmp_ptr->icmp_type = ICMP_ECHO;
53      icmp_ptr->icmp_code = 0;
54      icmp_ptr->icmp_id = packet_id;
55      icmp_ptr->icmp_seq = 0;
56      icmp_ptr->icmp_data[0] = '\0';
57      icmp_ptr->icmp_cksum = ip_cksum((unsigned short *)icmp_ptr, sizeof(*icmp_ptr));
58      ret = sendto(sd, icmp_ptr, sizeof(*icmp_ptr), 0, info->ai_addr, info->ai_addrlen);
59      free(icmp_ptr);
60      if (ret < 0) {
61          perror("sendto");
62          terminate(1);
63      }
64
65      recvbuf = (char *)malloc(maxp);
```

RAW sockets παράδειγμα: ping (3/4)

```
67      for (timedout = replied = 0; !timedout && !replied; ) {
68          sslen = sizeof(struct sockaddr_storage);
69          ret = recvfrom(sd, recvbuf, maxp, 0, (struct sockaddr *)&ss, &sslen);
70          if (ret < 0) {
71              if (errno == EWOULDBLOCK) {
72                  timedout = 1;
73                  break;
74              }
75              if (errno == EINTR)
76                  continue;
77              perror("recvfrom");
78              terminate(1);
79          }
80
81          ip_ptr = (struct ip *)recvbuf;
82          ip_hlen = ip_ptr->ip_hl << 2;
83          icmp_ptr = (struct icmp *)((char *)ip_ptr + ip_hlen);
84          if ((maxp - ip_hlen) < 8) {
85              fprintf(stderr, "No space for ICMP header\n");
86              terminate(1);
87          }
88          if (icmp_ptr->icmp_type == ICMP_ECHOREPLY) {
89              if (icmp_ptr->icmp_id == packet_id)
90                  replied = 1;
91          } else {
92              ip_ptr = (struct ip *)icmp_ptr->icmp_data;
93              icmp_ptr = (struct icmp *)(ip_ptr + 1);
94              if (ip_ptr->ip_p == IPPROTO_ICMP && icmp_ptr->icmp_type == ICMP_ECHO && icmp_ptr->icmp_id
                        == packet_id)
95                  replied = -1;
96          }
97      }
```

RAW sockets παράδειγμα: ping (4/4)

```
99      printf("%s (%s) %s\n", argv[1], info->ai_canonname, replied == 1 ? "is alive" : (replied == -1 ? "
            unreachable" : "timed out"));
100     terminate(0);
101 }
102
103 unsigned short ip_cksum(unsigned short *buf, int len) {
104     unsigned short *w = buf;
105     int sum = 0;
106     for (; len > 1; len -= 2)
107         sum += *w++;
108     if (len == 1)
109         sum += (unsigned short)(*(unsigned char *)w);
110     return ~(unsigned short)(((sum >> 16) + (sum & 0xFFFF)) + (sum >> 16));
111 }
```

Πρόσβαση στο επίπεδο διασύνδεσης

Πρόσβαση στο επίπεδο διασύνδεσης

- Πολλές προχωρημένες λειτουργίες απαιτούν απευθείας πρόσβαση στο επίπεδο διασύνδεσης.
- Συνήθως χρησιμοποιείται όταν:
  - Θέλουμε να εξετάσουμε τις κεφαλίδες και την δομή πακέτων στα οποία δεν έχουμε πρόσβαση μέσω RAW sockets.
  - Θέλουμε να έχουμε πρόσβαση σε όλα τα πακέτα που διαχειρίζεται το επίπεδο διασύνδεσης. Αν παράλληλα ενεργοποιήσουμε την λειτουργία «promiscuous» της διεπαφής δικτύου, μπορούμε να αποκτήσουμε πρόσβαση σε όλα τα πακέτα που κυκλοφορούν στο δίκτυο.
  - Ο πυρήνας δεν παρέχει υποστήριξη για κάποια δικτυακή λειτουργία που χρειαζόμαστε και μπορούμε να την υλοποιήσουμε σε userspace.
- Σε συστήματα UNIX υπάρχουν 3 κύρια «πρότυπα» πρόσβασης στο επίπεδο διασύνδεσης:
  - BSD Packet Filter (BPF).
  - SVR4 Data Link Provider Interface (DLPI).
  - Linux `SOCK_PACKET`.

BPF

Πρόσβαση στο επίπεδο διασύνδεσης

BPF

- Το BPF υλοποιείται εξολοκλήρου στον πυρήνα του λειτουργικού.
- Πριν σταλεί οποιοδήποτε πακέτο και αμέσως αφού παραληφθεί κάποιο νέο, το σύστημα δίνει αντίγραφό του στο BPF.
- Οι εφαρμογές αποκτούν πρόσβαση σε αυτό «ανοίγοντας» (`open(2)`) ένα *νέο* ειδικό αρχείο /dev/bpfX, όπου X ακέραιος (π.χ. /dev/bpf1).
  - Αν το αρχείο υπάρχει ήδη, η κλήση αποτυγχάνει με σφάλμα `EBUSY`.
- Η εφαρμογή μπορεί να ορίσει φίλτρα για τα πακέτα που την ενδιαφέρουν.
  - Παρέχονται σχετικές επιλογές της `ioctl(2)`.
  - Το φιλτράρισμα γίνεται εξολοκλήρου στον πυρήνα.
  - Για κάθε πακέτο που ταιριάζει στο φίλτρο, η εφαρμογή μπορεί να ορίσει να παραδίδεται σε αυτή μόνο ένα τμήμα του (capture length).
  - Ο πυρήνας κρατά τα σχετικά πακέτα σε buffer και τα παραδίδει στην εφαρμογή όταν γεμίσει ο buffer ή όταν παρέλθει ένα χρονικό όριο που αυτή έχει θέσει.
- Αν και συνήθως αποκτούμε πρόσβαση στο BPF για ανάγνωση πακέτων (π.χ. tcpdump), μπορούμε επίσης και να γράψουμε πακέτα με αποτέλεσμα το σύστημα να τα στείλει στο δίκτυο.
  - Γιατί το χρειαζόμαστε αυτό αφού έχουμε RAW sockets;

DLPI

Πρόσβαση στο επίπεδο διασύνδεσης

DLPI

- Υλοποιήθηκε από την AT&T και υποστηρίζεται σε συστήματα SVR4.
- Η βασική ιδέα λειτουργίας είναι η ίδια με αυτή του BPF.
- Οι εφαρμογές αποκτούν πρόσβαση σε αυτό «ανοίγοντας» το αρχείο που αντιστοιχεί στην διεπαφή δικτύου και χρησιμοποιώντας κατόπιν μία κλήση συστήματος για να το «προσαρτήσουν» (DL_ATTACH_REQUEST).
- Εξαιτίας της σχεδίασής του, το DLPI είναι αρκετά πιο αργό (3 με 20 φορές) σε σχέση με το BPF.

SOCK_PACKET

Πρόσβαση στο επίπεδο διασύνδεσης

SOCK_PACKET

- Στο Linux μπορούμε να αποκτήσουμε πρόσβαση στο επίπεδο διασύνδεσης χρησιμοποιώντας ένα socket τύπου `SOCK_PACKET`.
  - Το 3ο όρισμα της `socket(2)` καθορίζει το είδος των πακέτων που θέλουμε να λαμβάνουμε.
  - Παράδειγμα:

```
sd = socket(AF_INET, SOCK_PACKET, htons(ETH_P_ALL));
sd = socket(AF_INET, SOCK_PACKET, htons(ETH_P_IP));
sd = socket(AF_INET, SOCK_PACKET, htons(ETH_P_IPV6));
sd = socket(AF_INET, SOCK_PACKET, htons(ETH_P_ARP));
```

- Μπορούμε επίσης να θέσουμε την διεπαφή δικτύου σε κατάσταση λειτουργίας «promiscuous», χρησιμοποιώντας το flag `IFF_PROMISC` με την επιλογή `SIOCSIFFLAGS` της `ioctl(2)`.
- Σε σχέση με τα BPF και DLPI:
  - Η μέθοδος αυτή δεν προσφέρει κάποιου είδους φιλτράρισμα ή buffering στα πακέτα.
  - Δεν μπορούμε να καθορίσουμε τη διεπαφή δικτύου από την οποία μας ενδιαφέρουν τα πακέτα. Επιστρέφεται ωστόσο ως `sa_data` από τις `recvfrom(2)` και `recvmsg(2)`.

libpcap

Πρόσβαση στο επίπεδο διασύνδεσης

libpcap

- Η βιβλιοθήκη αυτή υποστηρίζει όλες τις προηγούμενες μεθόδους και κάποιες ακόμα...
  - Σχεδόν «ανεξάρτητη πλατφόρμας».
- Υποστηρίζει μόνο ανάγνωση πακέτων από το επίπεδο διασύνδεσης. Ωστόσο η προσθήκη υποστήριξης εγγραφής θεωρείται «εύκολη».
- Για να τη χρησιμοποιήσουμε:
  - Αρχικά «ανοίγουμε» ένα ειδικό «αρχείο συσκευής», με χρήση των `pcap_create(3)` ή `pcap_open_live(3)`.
  - Αρχικοποιούμε και ενεργοποιούμε το φίλτρο για τα εισερχόμενα πακέτα με χρήση των `pcap_compile(3)` και `pcap_setfilter(3)` αντίστοιχα.
  - Διαβάζουμε πακέτα με χρήση των `pcap_next(3)` ή `pcap_next_ex(3)`.
- Δείτε το `http://www.tcpdump.org/` για περισσότερες πληροφορίες.