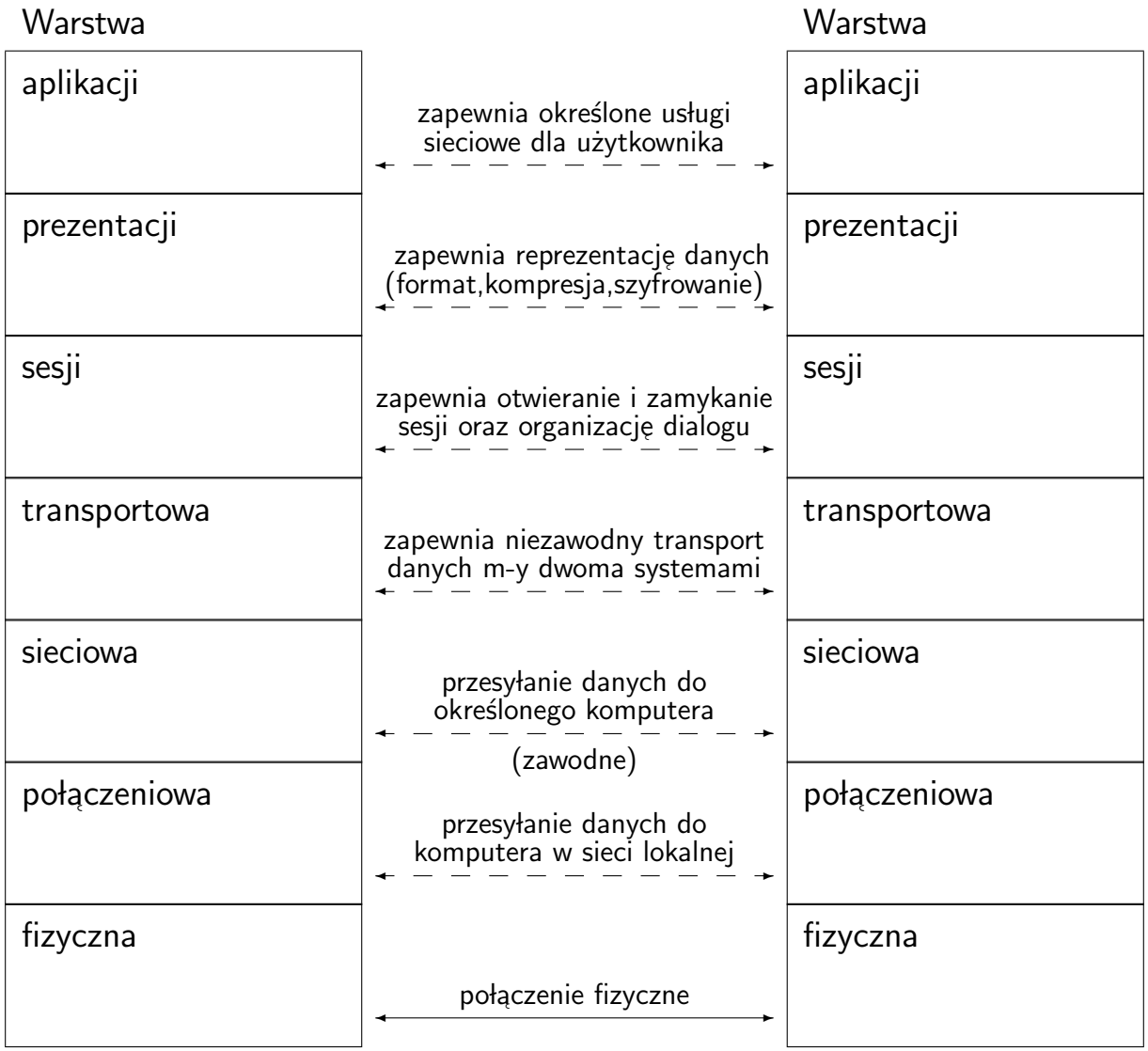
Unix: programowanie sieciowe

Witold Paluszyński
witoldp@pwr.wroc.pl
http://sequoia.ict.pwr.wroc.pl/~witold/

Niniejszy dokument zawiera materiały do wykładu na temat programowania sieciowego w systemie Unix. Jest on udostępniony pod warunkiem wykorzystania wyłącznie do własnych, prywatnych potrzeb i może być kopiowany wyłącznie w całości, razem z niniejszą stroną tytułową.

Warstwowy model sieci ISO-OSI



Model ISO a protokoły internetowe

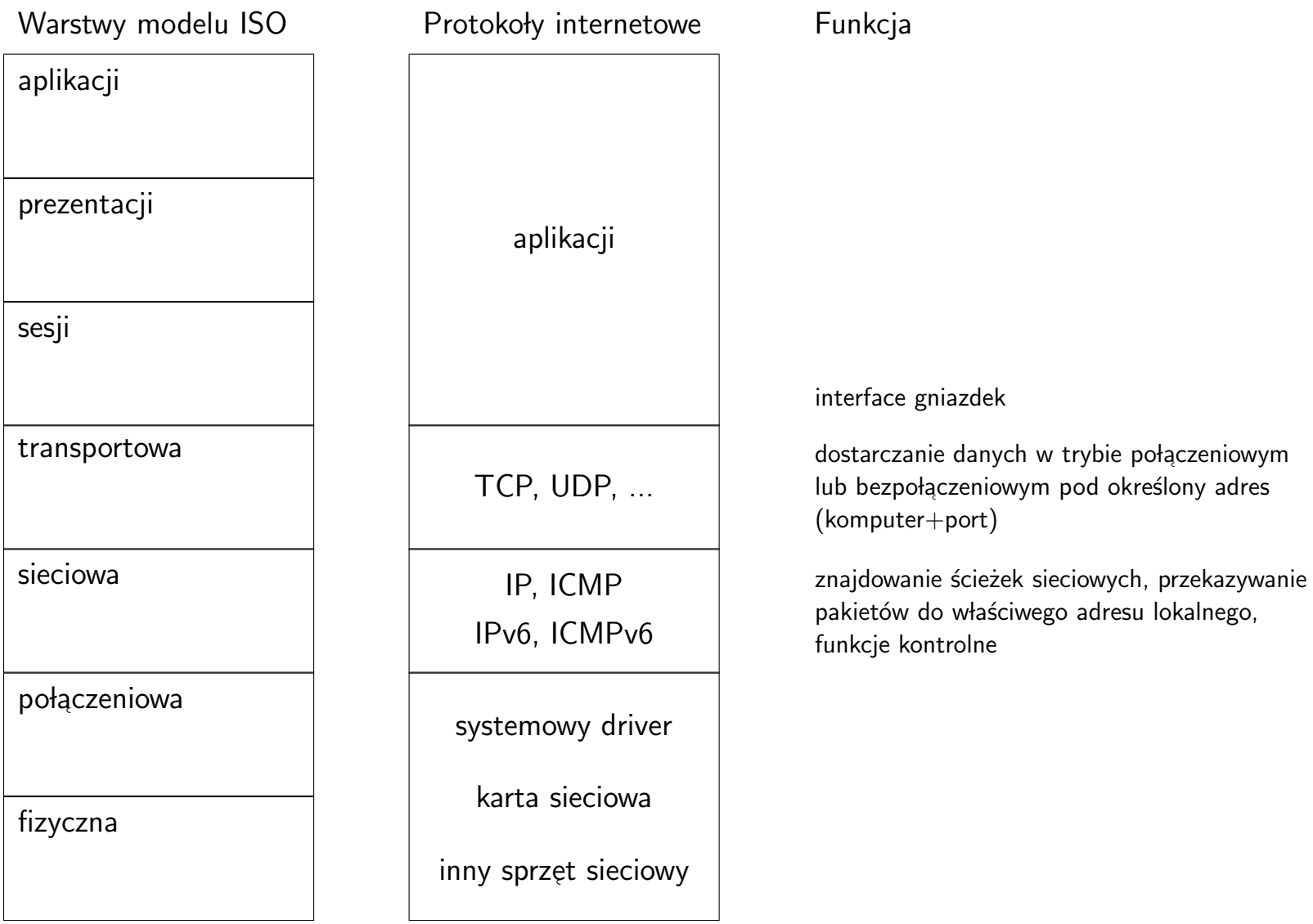
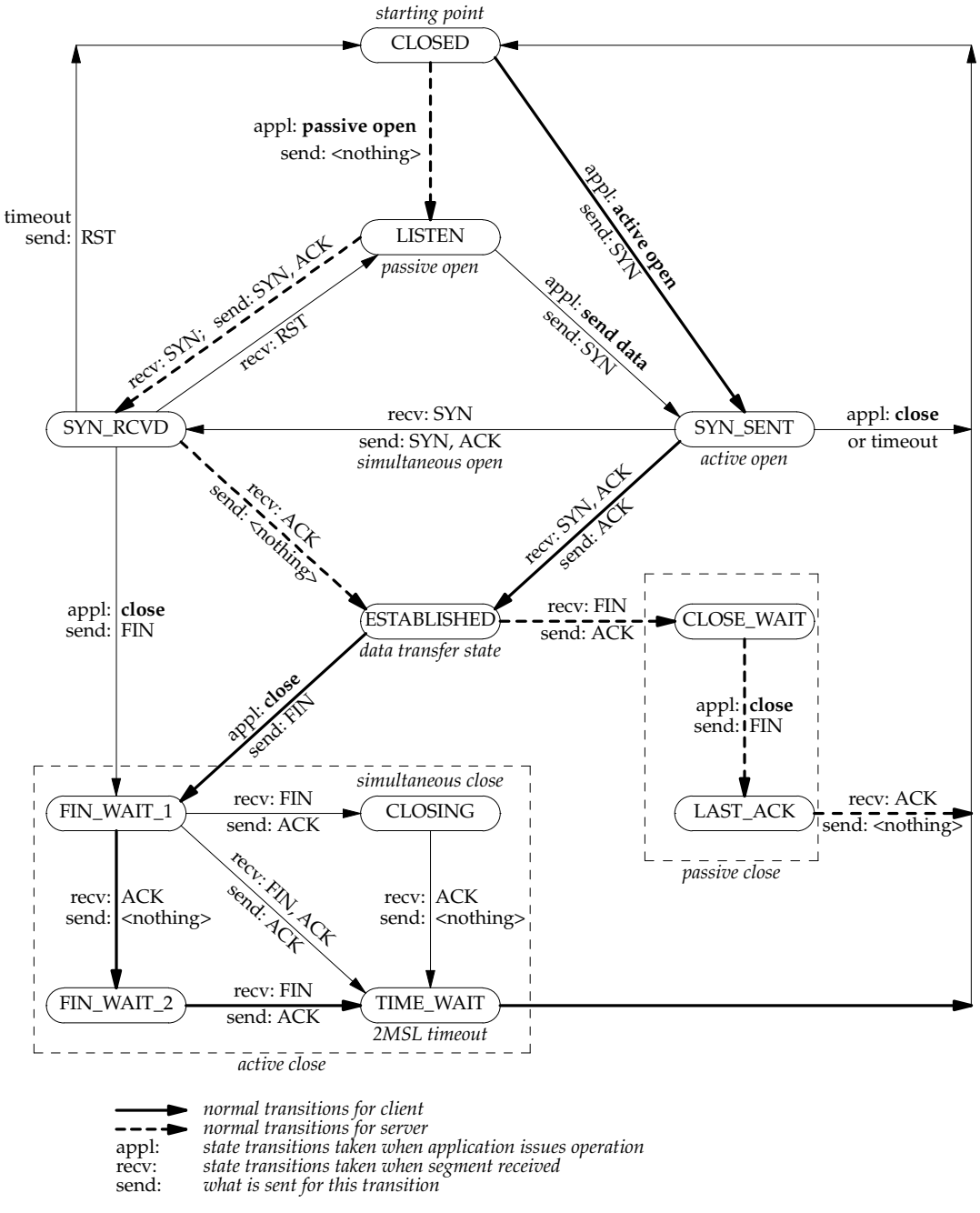


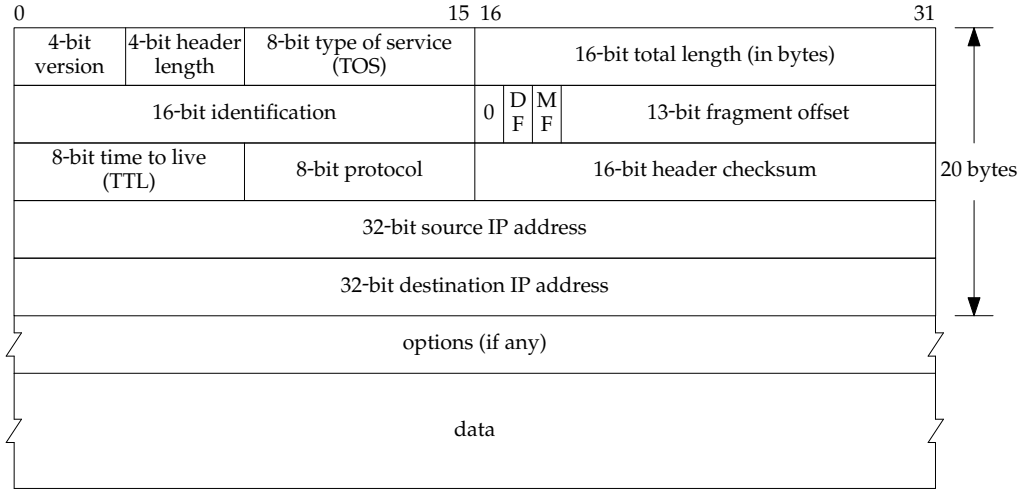
Diagram stanów protokołu TCP



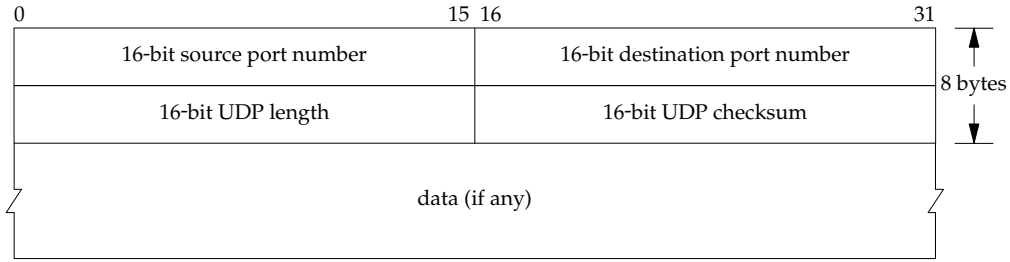
TCP state transition diagram.

Nagłówki pakietów TCP, UDP i IP

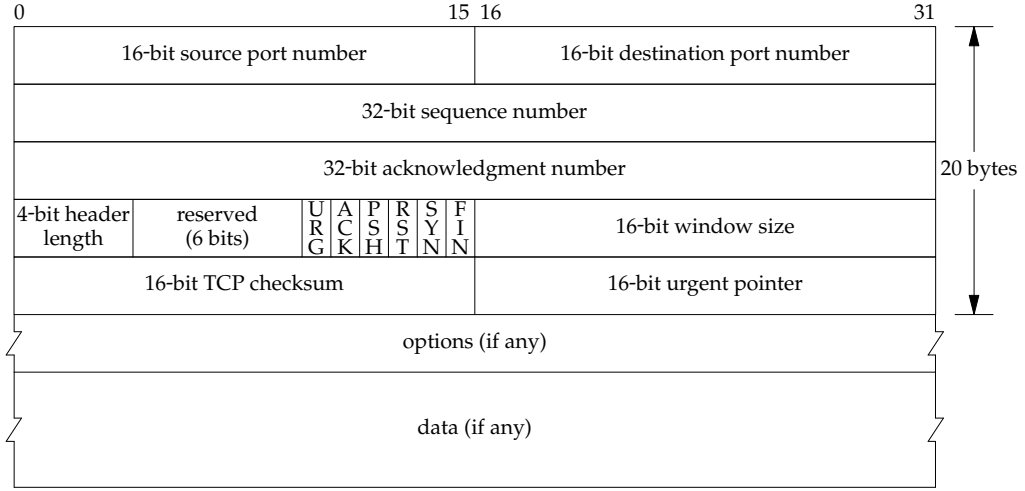
IP Header



UDP Header



TCP Header



Reprinted from *TCP/IP Illustrated, Volume 1: The Protocols*

Interface gniazdek

• Gniazdka — uniwersalny interface do mechanizmów komunikacji międzyprocesowej pozwalający na dwukierunkową komunikację procesów, nawet między różnymi komputerami. Gniazdko stanowi deskryptor pliku reprezentujący koniec „rury" (a raczej „dziury"), który po podłączeniu **gdzieś** drugiego końca można zapisywać i/lub odczytywać.

• Komunikaty wrzucone do dziury wypadają drugim końcem, ale nie na pewno i niekoniecznie we właściwej kolejności albo bez błędów.

• Gniazdka typu `SOCK_STREAM`: komunikacja w modelu połączeniowym — zwykle związana z protokołem TCP warstwy transportowej (analogia: komunikacja telefoniczna).

• Gniazdka typu `SOCK_DGRAM`: komunikacja w modelu bezpołączeniowym — zwykle związana z protokołem UDP warstwy transportowej (analogia: komunikacja listowa).

• Gniazdka typu `SOCK_RAW`: komunikacja na niskim poziomie, wyłącznie bezpołączeniowa — zwykle związana z protokołem IP warstwy sieciowej, a także komunikaty kontrolne protokołu ICMP.

Adresowanie w sieci Internet

32-bitowe (4-oktetowe) adresy IP wersji 4, stosowane od 1.I.1983

- klasa A: pierwszy oktet jest adresem sieci (a pozostałe trzy adresem komputera), z czego pierwszy bit jest zerem; możliwe 128 takich sieci (przedział 0-127) i 16 milionow komputerów w każdej
- klasa B: pierwsze dwa oktety są adresem sieci (a dwa adresem komputera), z czego pierwsze dwa bity są 10; możliwe 16 tysiecy adresów sieci (przedział 128-191) po 65 tysięcy komputerów
- klasa C: pierwsze trzy oktety są adresem sieci (a jeden adresem komputera), z czego pierwsze trzy bity są 110; możliwe dwa miliony takich sieci (przedział 192-223) i 256 komputerów w każdej
- klasa D: cały adres jest jednym adresem, ale cztery pierwsze bity muszą być 1110; adresy te stosuje się do komunikacji multicast
- adresy *broadcast*: same jedynki lub same zera w adresie komputera
- adresy prywatne:
  $10.0.0.0 - 10.255.255.255$
  $172.16.0.0 - 172.31.255.255$
  $192.168.0.0 - 192.168.255.255$

Adresowanie bezklasowe

Przydzielanie adresów IP według schematu pięciu klas jest nieefektywne; pomimo iż istnieje ponad 4 miliardy liczb 32-bitowych już w połowie lat 90-tych zaczęło brakować adresów IP. Ponadto, dowolność w przydzielaniu pojedynczych adresów sieci powodowała przepełnienie globalnych routerów zobowiązanych do odnajdowania ścieżki prowadzącej do każdej sieci. Dla rozwiązania tych problemów wprowadzono tzw. system CIDR (*classless inter-domain routing*). Odrzuca on sztywny podział adresu na sieć i komputer, oraz wprowadza agregację bloków adresów sieci.

Obecnie dowolny adres składa się z *prefixu* (pierwszej części) dowolnej długości stanowiącego adres sieci, i reszty stanowiącej adres komputera. Dla wskazania długości prefixu stosuje się notację *x.y.z.t/p*.

Na przykład, 156.17.9.0/25 oznacza adres sieci, w którym 25 bitów stanowi adres sieci, a pozostałych 7 adres komputera. Oznacza to, że może być 128 adresów w tej sieci, z których pierwszy (same zera w części adresu komputera) jest adresem sieci, a ostatni (same jedynki w części adresu komputera) jest adresem rozgłaszania (*broadcast*), co pozostawia 126 rzeczywistych adresów.

Adresowanie z podsieciami

Schemat adresowania: nr-sieci/nr-komputera pozwala przydzielać organizacji pulę adresów jako całą sieć. Jednak często użytkownicy adresów sieciowych sami chcą wydzielać sobie mniejsze sieci (podsieci) dla usprawnienia swojej własnej infrastruktury sieciowej. Dlatego jest możliwe traktowanie dowolnej początkowej części adresu komputera jako rozszerzenia adresu sieci (adresu podsieci), a reszty jako adresu komputera.

Na przykład: adres sieci 156.17.9.0/25, albo 156.17.9/25, w rzeczywistości składa się z adresu sieci 156.17.9.0/24 przydzielanej jednostkom Politechniki Wrocławskiej, i jednobitowego adresu podsieci (o wartości 0).

Dla odmiany, 156.17.9.128/25 nie jest adresem drugiej takiej samej sieci, ponieważ te adresy zostały podzielone na podsieci inaczej, z trzybitowym adresem podsieci, po 30 rzeczywistych adresów komputerów w każdej. Adresami tych sieci są więc: 156.17.9.128/27, 156.17.9.160/27, 156.17.9.192/27, 156.17.9.224/27.

Maską podsieci jest adres, w którym wszystkie bity adresu sieci są jedynkami, a bity adresu komputera zerami. Maską podsieci 156.17.9/25 jest 255.255.255.128, albo ffffff80, a posieci 156.17.9.128/27, jak i pozostałych trzech powyższych podsieci, jest 255.255.255.224, albo ffffffe0.

Adresy symboliczne

Dla wygody wprowadzono dualną przestrzeń adresów — adresy symboliczne. Są one zorganizowane w hierarchiczną strukturę tzw. domen adresowych. Struktura ta nie ma ograniczeń; domeny położone „niżej" w hierarchii są własnością różnych organizacji, które same dostarczają informacji o domenach w nich zawartych.

Np.: komputer `sequoia.ict.pwr.wroc.pl` (IP:156.17.9.3) należy do domeny `ict.pwr.wroc.pl`, która jest własnością Instytutu Cybernetyki Technicznej Politechniki Wrocławskiej, i która otrzymała prawa do tej domeny od Politechniki Wrocławskiej, właściciela domeny `pwr.wroc.pl`.

Przestrzeń adresów symbolicznych służy tylko wygodzie ludzkiej pamięci, i istnieje odwzorowanie adresów symbolicznych na adresy numeryczne. Do realizacji tego odwzorowania służy specjalny system DNS (*domain name system*) składający się z sieci serwerów wymieniających informacje o tych odwzorowaniach i serwujących te informacje na życzenie.

Adresy IP interface'ów

Komputer może posiadać więcej niż jedną kartę sieciową (tzw. *interface* sieciowy), które mogą być podłączone do różnych sieci lub do tej samej sieci. W obu przypadkach obie karty sieciowe powinny funkcjonować niezależnie i mieć oddzielne adresy IP. W pierwszym przypadku (dwie różne sieci) komputer może, ale nie musi, pełnić rolę łącznika, tzw, *routera* przekazującego ruch między tymi sieciami. Często jednak taki komputer posiada tylko jeden adres symboliczny.

Np. komputer tahoe.ict.pwr.wroc.pl ma dwie karty sieciowe o adresach 156.17.9.16 i 156.17.30.103 i jest routerem między siecią Robotyki 156.17.9/25 a siecią główną ICT 156.17.30.96/27.

Adres 127.0.0.1 (symbol `INADDR_LOOPBACK`) jest na mocy konwencji przypisany na każdym komputerze virtualnemu interface-owi *loopback*, który jest programową implementacją połączenia sieciowego komputera z samym sobą.

Adres 0.0.0.0 (symbol `INADDR_ANY`) jest również czasami używany jako tzw. adres *wild-card*.

Adresowanie — inne zagadnienia

• adresy niskiego poziomu w sieciach lokalnych
Są to tzw. adresy MAC, *medium access control* — unikalne 6-bajtowe adresy przydzielane na stałe urządzeniom sieciowym (patent firmy Xerox dla sieci Ethernet, stosowany również w innych technologiach sieciowych, np. token ring).

$2^{48} \approx 2.8 \times 10^{14}$ adresów co daje ok. 28 tysięcy adresów na każdego mieszkańca planety

• adresy IP wersji 6 (IPv6)
Jest to nowa wersja systemu adresów internetowych z nową wersją protokołu IP warstwy sieciowej. Adresy IPv6 są 128-bitowe i zawierają w sobie adresy MAC (a raczej ich 64-bitowe uogólnienie: EUI-64).

Routing

Routing jest czynnością określania lokalizacji sieci komputerowej na podstawie jej adresu.

Lokalizację rozumiemy tu w sensie połączeń, to znaczy znalezienie ścieżki sieciowej (szeregu połączonych routerów), prowadzącej do lokalizowanej sieci.

Realizacja tej czynności opiera się na związku prefixu adresu sieci z fizyczną lokalizacją sieci. Informacje wymieniane między komputerami znajdującymi ścieżki (routerami) pozwalają im na określanie tych ścieżek przez abstrakcję.

Algorytm routera: jeśli adres jest w mojej podsieci to doręczam pakiet komu trzeba, jeśli jest w jednej z podsieci dołączonej do mojej, lub znam na pamięć ścieżkę do tej podsieci z obsługującym ją routerem to jemu przekazuję pakiet, w przeciwnym wypadku przekazuję pakiet routerowi obsługującemu ruch na „zewnątrz" mojej podsieci; może to ja sam jestem takim routerem, w tym przypadku posiadam drugie połączenie z kimś kto obsługuje ruch wychodzący z mojej podsieci.

Komunikacja klient-serwer

• Pakiety sieciowe z konieczności odbiera system operacyjny, który następnie stwierdza dla kogo pakiet jest przeznaczony. Pakiety adresowane są tzw. numerami portów. Numery portów mogą być związane z gniazdkami otwartymi przez procesy już po nawiazaniu komunikacji, albo otwartymi w celu oczekiwania na zgłoszenie klienta.

• Większość komunikacji sieciowej odbywa się w trybie klient-serwer. Istnieją „dobrze-znane" usługi sieciowe, obsługiwane pod oficjalnymi numerami portów przez serwery, które rejestrują swoje usługi w systemie (związują gniazdko z adresem-numerem portu funkcją `bind`), następnie oczekują na nadejście zgłoszenia, odbierają je od systemu operacyjnego, i przystępują do ich obsłużenia mając jednocześnie na uwadze możliwość nadejścia następnych zgłoszeń.

```
ftp-data        20/tcp
ftp             21/tcp
smtp            25/tcp          mail
http            80/tcp          www www-http
http            80/udp          www www-http
sunrpc          111/udp         rpcbind
sunrpc          111/tcp         rpcbind
```

• Procesy pragnące nawiązać symetryczną komunikację sieciową (tzn. nie w

trybie klient-serwer) również muszą zainicjować komunikację za pośrednictwem zdalnego systemu. Jeden z partnerów musi zarejestrować się jak serwer, i oczekiwać na zgłoszenie się partnera może przejść do komunikacji symetrycznej zgodnie ze swoim protokołem.

• Jeszcze inny model komunikacji stanowi zdalne wywoływanie procedur (*remote procedure calling, RPC*). Proces wywoływania zdalnej procedury rozpoczyna się od kontaktu ze zdalnym serwerem pośredniczącym (brokera), który umożliwia komunikację z właściwym serwerem, który oblicza wartość funkcji dla przysłanych wartości argumentów.

Uzyskiwanie informacji o adresie

```
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <netdb.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    char *host, **names, **addrs;
    struct hostent *hostinfo;
    if (argc == 1) {
        char myname[256];
        gethostname(myname, 255);
        host = myname;
    }
    else
        host = argv[1];
    hostinfo = gethostbyname(host);
    if (!hostinfo) {
        fprintf(stderr, "brak translacji adresu dla: %s\n", host);
        exit(1);
    }
    printf("Wyniki dla adresu symbolicznego %s:\n", host);
```

```
    printf("Nazwa: %s\n", hostinfo -> h_name);
    printf("Aliasy:");
    names = hostinfo -> h_aliases;
    while(*names) {
        printf(" %s", *names);
        names++;
    }
    printf("\n");
    if (hostinfo -> h_addrtype != AF_INET) {
        fprintf(stderr, "Hmm, nie jest to adres internetowy!\n");
        exit(1);
    }
    addrs = hostinfo -> h_addr_list;
    while (*addrs) {
        printf(" %s", inet_ntoa(*(struct in_addr *)*addrs));
        addrs++;
    }
    printf("\n");
    exit(0);
}
```

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
/* In the included file <netinet/in.h> a sockaddr_in is defined as follows:
 * struct sockaddr_in {
 *      short   sin_family;
 *      u_short sin_port;
 *      struct in_addr sin_addr;
 *      char    sin_zero[8];
 * };
 * This program creates a datagram socket, binds a name to it, then reads * from t
 */
main() {
  int sock, length;
  struct sockaddr_in name;
  char buf[1024];

  /* Create socket from which to read. */
  sock = socket(PF_INET, SOCK_DGRAM, 0);
  if (sock < 0) {
    perror("opening datagram socket");
    exit(1);
```

```
  }
  /* Create name with wildcards. */
  name.sin_family = AF_INET;
  name.sin_addr.s_addr = INADDR_ANY;
  name.sin_port = 0;
  if (bind(sock, &name, sizeof(name))) {
    perror("binding datagram socket");
    exit(1);
  }
  /* Find assigned port value and print it out. */
  length = sizeof(name);
  if (getsockname(sock, &name, &length)) {
    perror("getting socket name");
    exit(1);
  }
  printf("Socket has port #%d\n", ntohs(name.sin_port));
  /* Read from the socket */
  if (read(sock, buf, 1024) < 0)
    perror("receiving datagram packet");
  printf("-->%s\n", buf);
  close(sock);
}
```

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>

#define DATA "The sea is calm tonight, the tide is full . . ."

main(int argc, char *argv[])
{
  int sock;
  struct sockaddr_in name;
  struct hostent *hp, *gethostbyname();

  /* Create socket on which to send. */
  sock = socket(PF_INET, SOCK_DGRAM, 0);
  if (sock < 0) {
    perror("opening datagram socket");
    exit(1);
  }

  /*
```

```
     * Construct name, with no wildcards, of the socket to send to.
     * Gethostbyname() returns a structure including the network address
     * of the specified host.  The port number is taken from the command
     * line.
     */
    hp = gethostbyname(argv[1]);
    if (hp == 0) {
      fprintf(stderr, "%s: unknown host\n", argv[1]);
      exit(2);
    }
    memcpy(&name.sin_addr, hp->h_addr, hp->h_length);
    name.sin_family = AF_INET;
    name.sin_port = htons(atoi(argv[2]));
    /* Send message. */
    if (sendto(sock, DATA, sizeof(DATA), 0, &name, sizeof(name)) < 0)
      perror("sending datagram message");
    close(sock);
}
```

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>

#define DATA "Half a league, half a league . . ."

/*
 * This program creates a socket and initiates a connection with the socket
 * given in the command line.  One message is sent over the connection and
 * then the socket is closed, ending the connection.  The form of the command
 * line is streamwrite hostname portnumber
 */

main(int argc, char *argv[]) {
  int sock;
  struct sockaddr_in server;
  struct hostent *hp, *gethostbyname();
  char buf[1024];

  /* Create socket */
```

```
  sock = socket(PF_INET, SOCK_STREAM, 0);
  if (sock < 0) {
    perror("opening stream socket");
    exit(1);
  }
  /* Connect socket using name specified by command line.  */
  server.sin_family = AF_INET;
  hp = gethostbyname(argv[1]);
  if (hp == 0) {
    fprintf(stderr, "%s: unknown host\n", argv[1]);
    exit(2);
  }
  memcpy(&server.sin_addr, hp->h_addr, hp->h_length);
  server.sin_port = htons(atoi(argv[2]));

  if (connect(sock, &server, sizeof(server)) < 0) {
    perror("connecting stream socket");
    exit(1);
  }
  if (write(sock, DATA, sizeof(DATA)) < 0)
    perror("writing on stream socket");
  close(sock);
}
```

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>

#define TRUE 1

/*
 * This program creates a socket and then begins an infinite loop. Each time
 * through the loop it accepts a connection and prints out messages from it.
 * When the connection breaks, or a termination message comes through, the
 * program accepts a new connection.
 */

main()
{
  int sock, length;
  struct sockaddr_in server;
  int msgsock;
  char buf[1024];
  int rval;
```

```
int i;

/* Create socket */
sock = socket(PF_INET, SOCK_STREAM, 0);
if (sock < 0) {
  perror("opening stream socket");
  exit(1);
}
/* Name socket using wildcards */
server.sin_family = AF_INET;
server.sin_addr.s_addr = INADDR_ANY;
server.sin_port = 0;
if (bind(sock, &server, sizeof(server))) {
  perror("binding stream socket");
  exit(1);
}
/* Find out assigned port number and print it out */
length = sizeof(server);
if (getsockname(sock, &server, &length)) {
  perror("getting socket name");
  exit(1);
}
printf("Socket has port #%d\n", ntohs(server.sin_port));

/* Start accepting connections */
```

```
  listen(sock, 5);
  do {
    msgsock = accept(sock, 0, 0);
    if (msgsock == -1)
      perror("accept");
    else do {
      memset(buf, 0, sizeof(buf));
      if ((rval  = read(msgsock, buf,  1024)) < 0)
        perror("reading stream message");
      i = 0;
      if (rval == 0)
        printf("Ending connection\n");
      else
        printf("-->%s\n", buf);
    } while (rval != 0);
    close(msgsock);
  } while (TRUE);
  /*
   * Since this program has an infinite loop, the socket "sock" is
   * never explicitly closed.  However, all sockets will be closed
   * automatically when a process is killed or terminates normally.
   */
}
```

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <sys/select.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>

#define TRUE 1

/*
 * This program uses select() to check that someone is trying to connect
 * before calling accept().
 */

main()
{
  int sock, length;
  struct sockaddr_in server;
  int msgsock;
  char buf[1024];
  int rval;
```

```
fd_set ready;
struct timeval to;

/* Create socket */
sock = socket(AF_INET, SOCK_STREAM, 0);
if (sock < 0) {
  perror("opening stream socket");
  exit(1);
}
/* Name socket using wildcards */
server.sin_family = AF_INET;
server.sin_addr.s_addr = INADDR_ANY;
server.sin_port = 0;
if (bind(sock, &server, sizeof(server))) {
  perror("binding stream socket");
  exit(1);
}
/* Find out assigned port number and print it out */
length = sizeof(server);
if (getsockname(sock, &server, &length)) {
  perror("getting socket name");
  exit(1);
}
printf("Socket has port #%d\n", ntohs(server.sin_port));
```

```
/* Start accepting connections */
listen(sock, 5);
do {
  FD_ZERO(&ready);
  FD_SET(sock, &ready);
  to.tv_sec = 5;
  if (select(sock + 1, &ready, 0, 0, &to) < 0) {
    perror("select");
    continue;
  }
  if (FD_ISSET(sock, &ready)) {
    msgsock =  accept(sock, (struct sockaddr *)0, (int *)0);
    if (msgsock == -1)
      perror("accept");
    else do {
      memset(buf, 0, sizeof(buf));
      if ((rval = read(msgsock, buf, 1024)) < 0)
        perror("reading stream message");
      else if (rval == 0)
        printf("Ending connection\n");
      else
        printf("-->%s\n", buf);
    } while (rval > 0);
    close(msgsock);
  } else
```

```
      printf("Do something else\n");
  } while (TRUE);
}
```