IK1350 Protocols in Computer Networks/ Protokoll i datornätverk Spring 2008, Period 3

Module 5: UDP and friends

KTH Information and Communication Technology

Lecture notes of G. Q. Maguire Jr.

For use in conjunction with *TCP/IP Protocol Suite*, by Behrouz A. Forouzan, 3rd Edition, McGraw-Hill, 2006.

For this lecture: Chapters 11, 16, 17

Last modified: 2008.02.03:15:15

Outline

- UDP
- Socket API
- BOOTP
- DHCP
- DNS, DDNS

Transport layer protocols

The transport layer is responsible for end-to-end delivery of entire messages

- uses Protocol Port and/or Port Number to demultiplex the incoming packet so that it can be delivered to a specific process
- Segmentation and Reassembly
  - Divides a message into transmittable segments and reassemble them at the receive
- Connection control - for connection-oriented transport protocols
- End-to-end Flow Control (in contrast to link level flow control)
- End-to-end Error Control (in contrast to link level error control)

Main Transport layer protocols

Three main transport layer protocols:

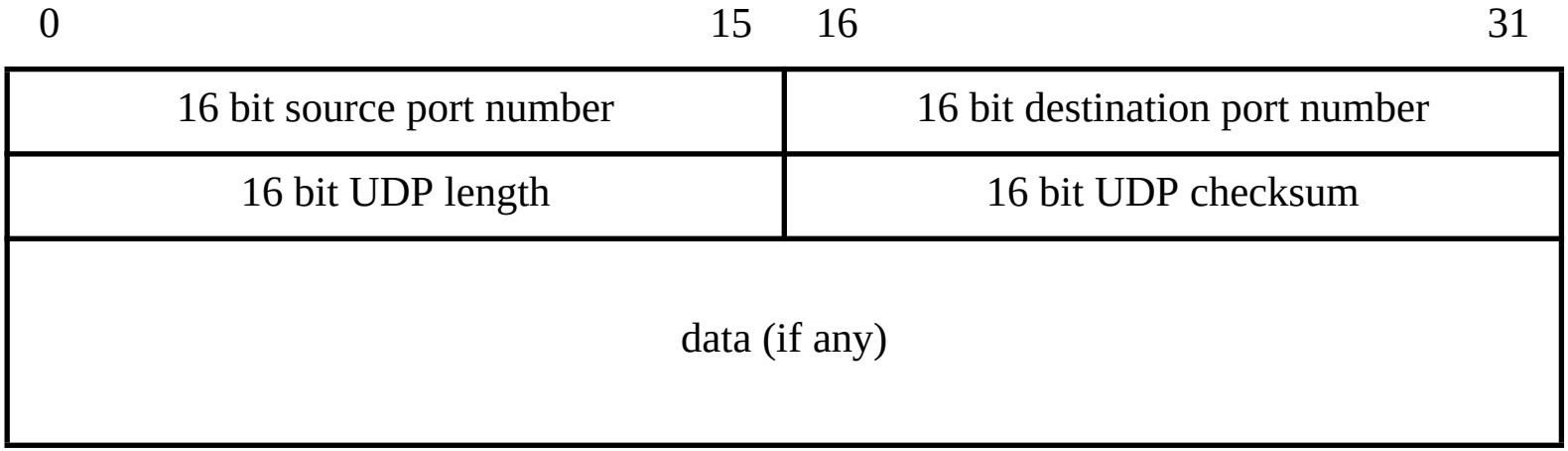
- User Datagram Protcol (UDP) <<< today's topic
  - Connectionless unreliable service
- Transmission Control Protocol (TCP)
  - Connection-oriented reliable stream service
- Stream Control Transmission Protocol (STCP)
  - a modern transmission protocol with many facilities which the user can chose from

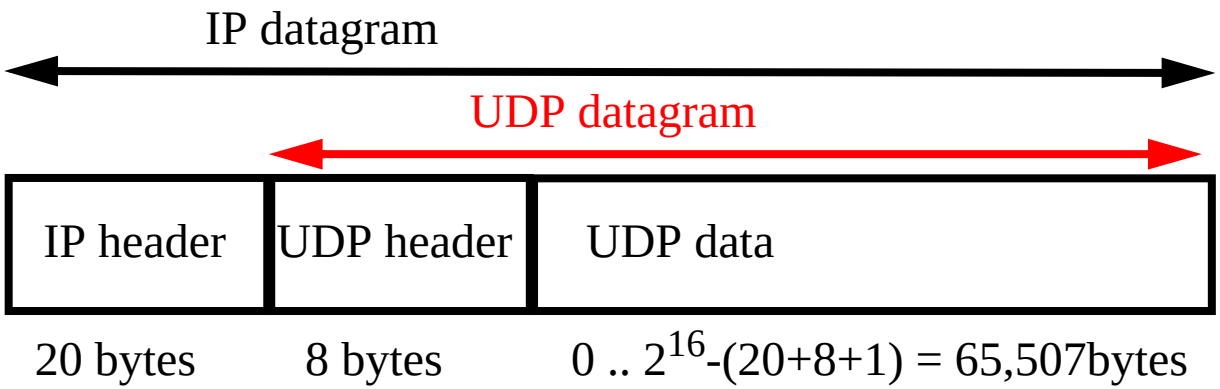
User Datagram Protocol (UDP)

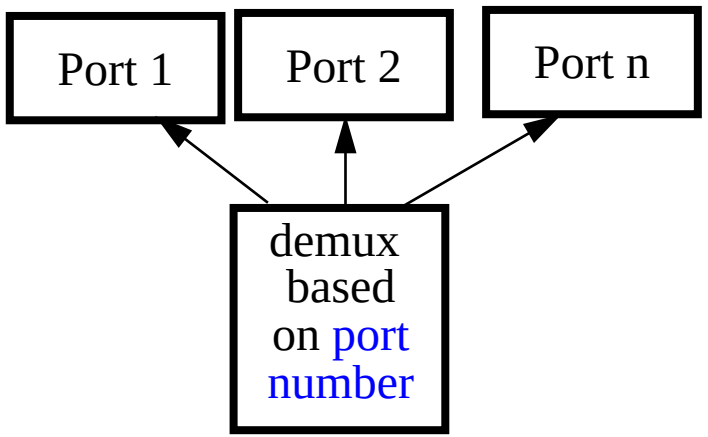
- Datagram-oriented transport layer protocol
- Provides connectionless unreliable service
- No reliability guarantee
- Checksum covers both header and data, end-to-end, but optional
  - if you care about your data you should be doing end-to-end checksums or using an even stronger error detection (e.g., MD5).
- An UDP datagram is silently discarded if checksum is in error.
  - No error message is generated
- Lots of UDP traffic is only sent locally
  - thus the reliability is comparable to the error rate on the local links. (see Stevens, Vol. 1, figure 11.5, pg. 147 for comparison of Ethernet, IP, UDP, and TCP checksum errors)
- Each output operation results in one UDP datagram, which causes one IP datagram to be sent
- Applications which use UDP: DNS, TFTP, BOOTP, DHCP, SNMP, NFS, VoIP, etc.
  - An advantage of UDP is that it is a base to build your own protocols on
  - Especially if you don't need reliability and in order delivery of lots of data

UDP Header

8 byte header + possible data

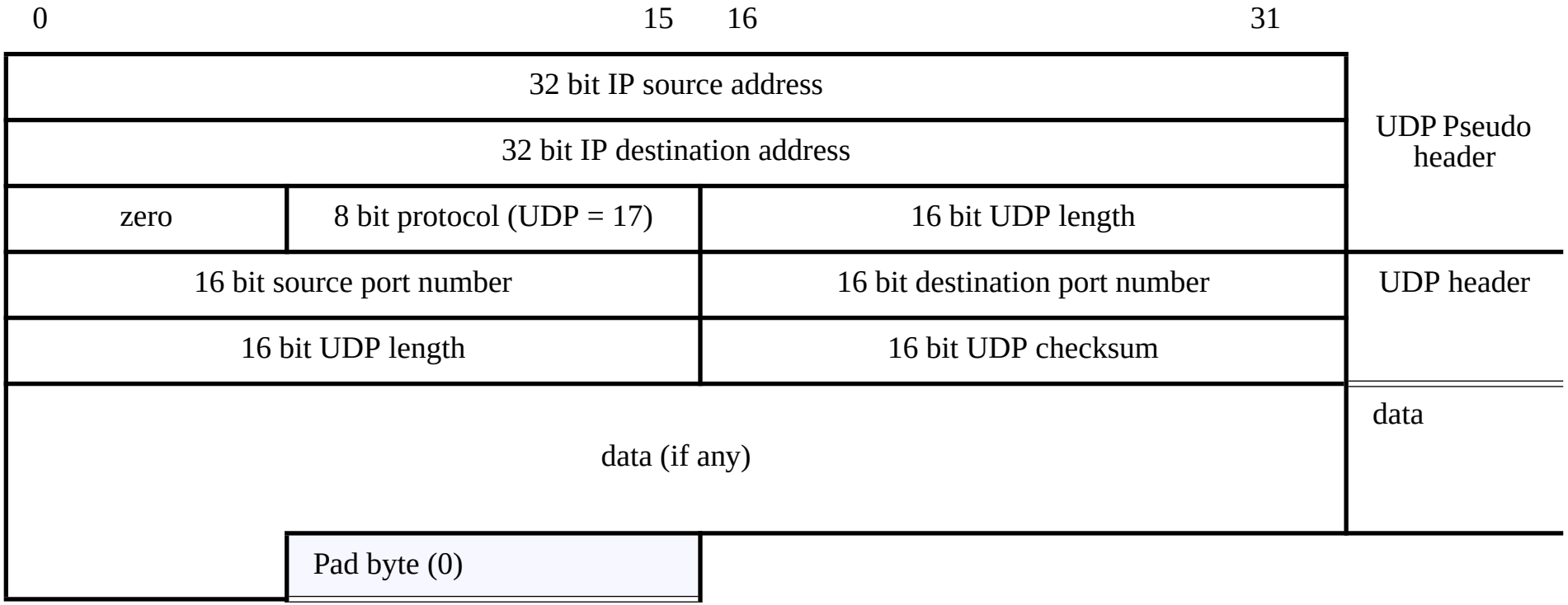






UDP Checksum and Pseudo-Header

- UDP checksum covers more info than is present in the UDP datagram alone: pseudo-header and pad byte (0) {to even number of 16 bit words}.
- Propose: to verify the UDP datagram reached its correct destination: right port number at the right IP address.
- Pseudo-header and pad byte are not transmitted with the UDP datagram, only used for checksum computation.



Reserved and Available UDP Port Numbers

	keyword	UNIX keyword	Description
0			reserved
7	ECHO	echo	Echo
9	DISCARD	discard	Discard == sink null
11	USERS	systat	Active users
13	DAYTIME	daytime	Daytime
15	-	netstat	Network status program
17	QUOTE	qotd	Quote of the day
19	CHARGEN	chargen	Character generator
37	TIME	time	Time server
39	RLP	rlp	Resource Location Protocol
42	NAMESERVER	name	Host Name Server
43	NICNAME	whois	Who is
53	DOMAIN	domain	Domain Name Server
67	BOOTPS	bootps	Bootstrap Protocol Server
68	BOOTPC	bootpc	Bootstrap Protocol Client
69	TFTP	tftp	Trivial File Transfer Protocol
88	KERBEROS	kerberos5	Kerberos v5 kdc
111	SUNRPC	sunrpc	SUN Remote Procedure Call (portmap)
123	NTP	ntp	Network Time Protocol
137	netbios_ns	netbios_ns	NetBIOS name service
138	netbios_dgm	netbios_dgm	NetBIOS Datagram Service
139	netbios_ssn	netbios_ssn	NetBIOS Session Service
161		snmp	Simple Network Management Protocol Agent
162		snmp-trap	Simple Network Management Protocol Traps
512		biff	mail notification
513		who	remote who and uptime
514		syslog	remote system logging
517		talk	conversation
518		ntalk	new talk, conversation
520		route	routing information protocol
525		timed	remote clock synchronization
533	netwall	netwall	Emergency broadcasting
750	kerberos	kerberos	Kerberos (server)
6000 + display number			X11 server
7000			X11 font server

Port numbers in three groups

Range	Purpose
0 .. 1023	System (Well-Known) Ports[a]
1024 .. 49151	User (Registered) Ports
49152 .. 65535	Dynamic and/or Private Ports

a. Roughly 300 well know port numbers remain unassigned and 38 reserved
Roughly 26k registered port numbers remain unassigned and 9 reserved

‘For the purpose of providing services to unknown callers, a service contact port is defined. This list specifies the port used by the server process as its contact port. The contact port is sometimes called the "well-known port".’

http://www.iana.org/assignments/port-numbers

Linux chooses the local port to use for TCP and UDP traffic from this range:

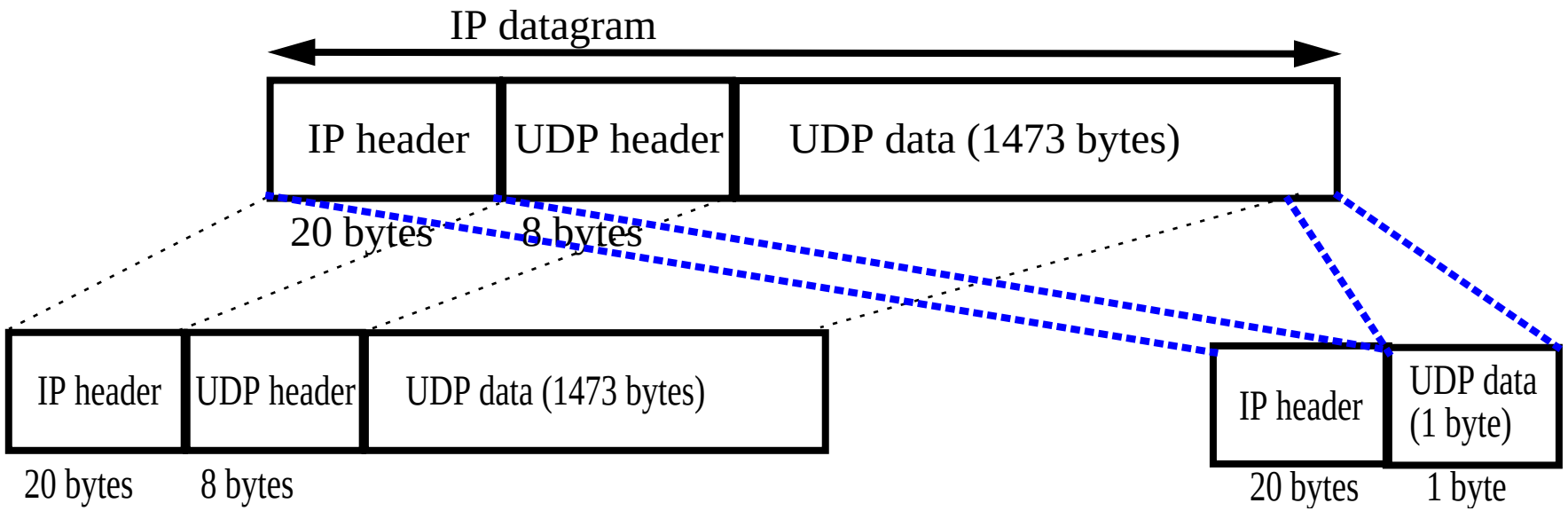
```
$ cat /proc/sys/net/ipv4/ip_local_port_range
  1024    29999
```

MTU and Datagram Fragmentation

If datagram size > MTU, perform fragmentation.

- At sending host or at intermediate router (IPv4).
- Reassembled only at final destination.

Example: 1501 (20 + 8 + 1473 data) on Ethernet (MTU=1500):



- Note there is no UDP header in the second fragment.
- Therefore, a frequent operation is to compute the path MTU before sending anything else. (see RFC 1191 for the table of common MTUs)

Fragmentation Required

If datagram size > MTU, DF (Don’t Fragment) in IP header is on, then the router sends ICMP Unreachable Error.

Of course this can be used to find Path MTU.

Interaction between UDP and ARP

With ARP cache empty, send a UDP datagram with 8192 bytes onto an Ethernet

- 8192 bytes > ethernet MTU, therefore 6 fragments are created by IP
- if ARP cache is empty, first fragment causes ARP request to be sent
- This leads to two timing questions:

1. Are the remaining fragments sent before the ARP reply is received?
2. What does ARP do with multiple packets to the same destination while waiting for a reply?

Example under BSD

```
Bsdi% arp -a      ARP cache is empty
Bsdi% sock -u -i -n1 -w8192 svr4 discard
```

10.0		arp who-has svr4 tell bsdi
20.001234	(0.0012)	arp who-has svr4 tell bsdi
30.001941	(0.0007)	arp who-has svr4 tell bsdi
40.002775	(0.0008)	arp who-has svr4 tell bsdi
50.003495	(0.0007)	arp who-has svr4 tell bsdi
60.004319	(0.0008)	arp who-has svr4 tell bsdi
70.008772	(0.0045)	arp reply svr4 is-at 0:0:c0:c2:9b:26
80.009911	(0.0011)	arp reply svr4 is-at 0:0:c0:c2:9b:26
90.011127	(0.0012)	bsdi > svr4: (frag 10863:800@7400)
100.011255	(0.0001)	arp reply svr4 is-at 0:0:c0:c2:9b:26
110.012562	(0.0013)	arp reply svr4 is-at 0:0:c0:c2:9b:26
120.013458	(0.0009)	arp reply svr4 is-at 0:0:c0:c2:9b:26
130.014526	(0.0011)	arp reply svr4 is-at 0:0:c0:c2:9b:26
140.015583	(0.0011)	arp reply svr4 is-at 0:0:c0:c2:9b:26

- on a BSDI system:
  - each of the additional (5) fragments caused an ARP request to be generated
    - this **violates** the Host Requirements RFC - which tries to prevent ARP flooding by limiting the maximum rate to 1 per second
  - when the ARP reply is received the **last** fragment is sent
    - Host Requirements RFC says that ARP should save at least one packet and this should be the latest packet
  - unexplained anomaly: the System Vr4 system sent 7 ARP replies back!
  - no ICMP “time exceeded during reassembly” message is sent
    - BSD derived systems - never generate this error!
      It does set the timer internally and discard the fragments, but never sends an ICMP error.
    - fragment 0 (which contains the UDP header) was not received - so there is no way to know which process sent the fragment; thus unless fragment 0 is received - you are not required to send an ICMP “time exceeded during reassembly” error.

Not just a fluke (i.e., a rare event)

- The same error occurs even if you don’t have fragmentation - simply sending multiple UDP datagrams rapidly when there is no ARP entry is sufficient!
- NFS sends UDP datagrams whose length just exceeds 8192 bytes
  - NFS will timeout and resend
  - however, there will always be this behavior - if the ARP cache has no entry for this destination!

Still a problem?

A UDP with 8192 payload to echo port as seen on SuSE 9.2 linux 2.6.8-24:

```
No.     Time        Source                Destination           Protocol Info
     37 3.020002    172.16.33.16          Broadcast             ARP      Who has 172.16.33.5?  Tell 172.16.33.16
     38 3.021385    172.16.33.5           172.16.33.16          ARP      172.16.33.5 is at 00:40:8c:24:37:f4
     39 3.021422    172.16.33.16          172.16.33.5           IP       Fragmented IP protocol (proto=UDP 0x11, off=4440)
     40 3.021452    172.16.33.16          172.16.33.5           IP       Fragmented IP protocol (proto=UDP 0x11, off=5920)
     41 3.021480    172.16.33.16          172.16.33.5           IP       Fragmented IP protocol (proto=UDP 0x11, off=7400)
```

3.021385-3.020002=.001383 sec. ⇒ 1.383ms for the ARP reply

All but the last 3 fragments are dropped! Including the initial echo request packet -- so in the fragments that do arrive you don’t know who they are for -- because the first fragment was lost!

With an even larger UDP packet

I removed the arp cache entry with: /sbin/arp -i eth1 -d 172.16.33.5

When sending 65500 bytes of UDP payload -- it looses many packets (in fact all but the last 3 fragments)!!!

```
No.     Time        Source                Destination           Protocol Info
     36 4.342158    172.16.33.16          Broadcast             ARP      Who has 172.16.33.5?  Tell 172.16.33.16
     37 4.342875    172.16.33.5           172.16.33.16          ARP      172.16.33.5 is at 00:40:8c:24:37:f4
     38 4.342906    172.16.33.16          172.16.33.5           IP       Fragmented IP protocol (proto=UDP 0x11, off=62160)
     39 4.342932    172.16.33.16          172.16.33.5           IP       Fragmented IP protocol (proto=UDP 0x11, off=63640)
     40 4.342986    172.16.33.16          172.16.33.5           IP       Fragmented IP protocol (proto=UDP 0x11, off=65120)
```

With the entry in the ARP cache get:

```
No.     Time        Source                Destination           Protocol Info
     35 5.118063    172.16.33.16          172.16.33.5           IP       Fragmented IP protocol (proto=UDP 0x11, off=1480)
     36 5.118095    172.16.33.16          172.16.33.5           IP       Fragmented IP protocol (proto=UDP 0x11, off=2960)
     37 5.118115    172.16.33.16          172.16.33.5           IP       Fragmented IP protocol (proto=UDP 0x11, off=4440)
     38 5.118214    172.16.33.16          172.16.33.5           IP       Fragmented IP protocol (proto=UDP 0x11, off=5920)
     39 5.118328    172.16.33.16          172.16.33.5           IP       Fragmented IP protocol (proto=UDP 0x11, off=7400)
     40 5.118450    172.16.33.16          172.16.33.5           IP       Fragmented IP protocol (proto=UDP 0x11, off=8880)
     41 5.118574    172.16.33.16          172.16.33.5           IP       Fragmented IP protocol (proto=UDP 0x11, off=10360)
     42 5.118695    172.16.33.16          172.16.33.5           IP       Fragmented IP protocol (proto=UDP 0x11, off=11840)
     43 5.118819    172.16.33.16          172.16.33.5           IP       Fragmented IP protocol (proto=UDP 0x11, off=13320)
               ….
     72 5.122385    172.16.33.16          172.16.33.5           IP       Fragmented IP protocol (proto=UDP 0x11, off=56240)
     73 5.122508    172.16.33.16          172.16.33.5           IP       Fragmented IP protocol (proto=UDP 0x11, off=57720)
     74 5.122631    172.16.33.16          172.16.33.5           IP       Fragmented IP protocol (proto=UDP 0x11, off=59200)
     75 5.122787    172.16.33.16          172.16.33.5           IP       Fragmented IP protocol (proto=UDP 0x11, off=60680)
     76 5.122877    172.16.33.16          172.16.33.5           IP       Fragmented IP protocol (proto=UDP 0x11, off=62160)
     77 5.122999    172.16.33.16          172.16.33.5           IP       Fragmented IP protocol (proto=UDP 0x11, off=63640)
     78 5.123122    172.16.33.16          172.16.33.5           IP       Fragmented IP protocol (proto=UDP 0x11, off=65120)
```

The initial UDP Echo request is still lost! The key parameter is `/proc/sys/net/ipv4/neigh/ethX/unres_qlen` where X is the interface (i.e., eth0, eth1, …) -- the default value is 3.

Maximum UDP Datagram size

- theoretical limit: 65,535 bytes - due to (IP's) 16-bit total length field
  - with 20 bytes of IP header + 8 bytes of UDP header ⇒ 65,507 bytes of user data
- two limits:
  - sockets API limites size of send and receive buffer; generally 8 kbytes, but you can call a routine to change this
  - TCP/IP implementation - Stevens found various limits to the sizes - even with loopback interface (see Stevens, Vol. 1, pg. 159)
- Hosts are required to handle at least 576 byte IP datagrams, thus lots of protocols limit themselves to 512 bytes or less of data:
  - DNS, TFTP, BOOTP, and SNMP

Datagram truncation

What if the application is not prepared to read the datagram of the size sent?

Implementation dependent:

- traditional Berkeley: silently truncate
- 4.3BSD and Reno: **can** notify the application that the data was truncated
- SVR4: excess data returned in subsequent reads - application is not told that this all comes from one datagram
- TLI: sets a flag that more data is available, subsequent reads return the rest of the datagram

Socket API

- int socket(int domain, int type, int protocol);
  - creates an endpoint description that you can use to send and receive network traffic
- int bind(int sockfd, struct sockaddr *my_addr, socklen_t addrlen);
  - binds a socket to the local address and port: my_address
- int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);
  - connect the local socket to a remote socket
- int listen(int s, int backlog);
  - indicates that socket is willing to accept connections & limites the queue of pending connections
- int accept(int s, struct sockaddr *addr, socklen_t *addrlen);
  - accepted the first connection request from the queue and connects it to the socket
- connection oriented sending and receiving:
  - ssize_t send(int s, const void *buf, size_t len, int flags);
  - ssize_t recv(int s, void *buf, size_t len, int flags);
- datagram sending and receiving:
  - ssize_t sendto(int s, const void *buf, size_t len, int flags, const struct sockaddr *to, socklen_t tolen);
  - ssize_t recvfrom(int s, void *buf, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen);
- int close(int fd) and int shutdown(int s, int how) - end it all!

Learning about Socket programming

For examples of using the socket API and networking programming see :

- Richard Stevens, *UNIX Network Programming, Volume 1, Second Edition: Networking APIs: Sockets and XTI*, Prentice Hall, 1998, ISBN 0-13-490012-X
  - source code *ftp://ftp.kohala.com/pub/rstevens/unpv12e.tar.gz*
  - errata list: *http://www.kohala.com/~rstevens/typos.unpv12e.txt*
- Brian "Beej" Hall, “Beej’s Guide to Network Programming: Using Internet Sockets”, 04/08/2004 07:22:02 PM

  http://www.ecst.csuchico.edu/~beej/guide/net/

Two addition socket functions for controlling various properties of sockets are:

- int getsockopt(int s, int level, int optname, void *optval, socklen_t *optlen);
- int setsockopt(int s, int level, int optname, const void *optval, socklen_t optlen);

For example, the options SO_SNDBUF and SO_RCVBUF - control the size of the sending buffer and the receiver buffer.

Simple UDP client

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#define bigBufferSize     8192
#define destination_host "172.16.33.5"

main(argc, argv)
int argc;
char **argv;
{ int client_socket_fd;           /* Socket to client, server */
  struct sockaddr_in server_addr; /* server's address */
  char bigBuffer[bigBufferSize];  /* buffer of data to send as payload */
  int sendto_flags=0;
                                  /* create a UDP socket */
  if ((client_socket_fd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1) {
    perror("Unable to open socket"); exit(1); };

                                  /* initialize the server address structure */
  memset( (char*)&server_addr, 0, sizeof(server_addr));
  server_addr.sin_family=AF_INET;
  server_addr.sin_port=htons(9);  /* 9 is the UDP port number for Discard */

  if (inet_aton(destination_host, (struct sockaddr*)&server_addr.sin_addr) == 0) {
    fprintf(stderr, "could not get an address for: %s", destination_host);exit(1);}

  if ((sendto(client_socket_fd, bigBuffer, bigBufferSize,
              sendto_flags, (struct sockaddr*)&server_addr, sizeof(server_addr))) == -1) {
    perror("Unable to send to socket"); close(client_socket_fd); exit(1);}

  close(client_socket_fd);        /* close the socket */
  exit(0);
}
```

UDP server design

Stevens, Vol, 1, pp. 162-167 discusses how to program a UDP server

You can often determine what IP address the request was sent to (i.e., the destination address):

- for example: thus ignoring datagrams sent to a broadcast address

You can limit a server to a given incoming IP address:

- thus limiting requests to a given interface

You can limit a server to a given foreign IP address and port:

- only accepting requests from a given foreign IP address and port #

Multiple recipients per port (for implementations with multicasting support)

- setting SO_REUSEADDR socket option ⇒ each process gets a copy of the incoming datagram

Note: limited size input queue to each UDP port, can result in silent discards without an ICMP message being sent back (since OS discarded, not the network!)

UDP listener example

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define bigBufferSize 8192
#define my_port       52000
#define destination_host "127.0.0.1"

main(argc, argv)
int argc;
char **argv;
{
  int client_socket_fd;                 /* Socket to client, server */
  struct sockaddr_in client_addr;       /* client's address */
  struct sockaddr_in other_addr;        /* other party's address */
  int other_addr_len;
  char bigBuffer[bigBufferSize];
  int sendto_flags=0;
                                        /* create a UDP socket */
  if ((client_socket_fd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1) {
    perror("Unable to open socket"); exit(1); };

  memset((char*)&client_addr, 0, sizeof(client_addr)); /* initialize address structure */
  client_addr.sin_family=AF_INET;
  client_addr.sin_port=htons(my_port);
  client_addr.sin_addr.s_addr = htonl(INADDR_ANY);

  if (bind(client_socket_fd, (struct sockaddr*)&client_addr, sizeof(client_addr))==-1) {
    close(client_socket_fd); exit(1); }

  if ((recvfrom(client_socket_fd, bigBuffer, bigBufferSize,
                sendto_flags, (struct sockaddr*)&other_addr, &other_addr_len)) == -1) {
    perror("Unable to receive from socket"); close(client_socket_fd); exit(1); }

  printf("Received packet from %s:%d\nData: %s\nString length=%d\n",
  inet_ntoa(other_addr.sin_addr), ntohs(other_addr.sin_port), bigBuffer, strlen(bigBuffer));

  close(client_socket_fd); exit(0);}
```

Changed the client

Changing the following:

```
#define destination_host "127.0.0.1"
#define my_port        52000

server_addr.sin_port=htons(my_port);
```

Adding some content to the bigBuffer:

```
sprintf(bigBuffer, "This is a simple test string to be sent to the
other party\n");
```

Sending only as much of the buffer as necessary:

```
if ((sendto(client_socket_fd, bigBuffer, strlen(bigBuffer),
             sendto_flags, (struct sockaddr*)&server_addr,
sizeof(server_addr))) == -1) {…}
```

Results in the listener outputting:

```
Received packet from 127.0.0.1:1260
Data: This is a simple test string to be sent to the other party

String length=59
```

Building a UDP packet from scratch

```
/*  simple example of building a UDP packet from scratch, based on the program:
    PingPong - 970621 by Willy TARREAU  <tarreau@aemiaif.ibp.fr>
      http://www.insecure.org/sploits/inetd.internal_udp_ports.DOS.attack.html
  As this program uses RAW sockets, you must be root to run it
*/

#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/udp.h>
#include <netdb.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <ctype.h>
#include <errno.h>

struct sockaddr addrfrom;
struct sockaddr addrto;
int s;
u_char outpack[65536];
struct iphdr *ip;
struct udphdr *udp;

main(int argc, char **argv) {
    struct sockaddr_in *from;
    struct sockaddr_in *to;
    struct protoent *proto;
    int i;
    char *src,*dest;
    int srcp, destp;
    int packetsize,datasize;

   if (argc!=5) {fprintf(stderr,"Usage: %s src_addr src_port dst_addr dst_port\n", argv[0]);
                 fprintf(stderr,"src_addr and dst_addr must be given as IP addresses (xxx.xxx.xxx.xxx)\n");
                 exit(2);}
     src=argv[1]; srcp=atoi(argv[2]); dest=argv[3]; destp=atoi(argv[4]);
```

```
if (!(proto = getprotobyname("raw"))) {perror("getprotobyname(raw)");exit(2);}

if ((s = socket(AF_INET, SOCK_RAW, proto->p_proto)) < 0) {perror("socket");exit(2);}

memset(&addrfrom, 0, sizeof(struct sockaddr));
from = (struct sockaddr_in *)&addrfrom;
from->sin_family = AF_INET;
from->sin_port=htons(srcp);
if (!inet_aton(src, &from->sin_addr)) {fprintf(stderr,"Incorrect address for 'from': %s\n",src);exit(2); }

memset(&addrto, 0, sizeof(struct sockaddr));
to = (struct sockaddr_in *)&addrto;
to->sin_family = AF_INET;
to->sin_port=htons(destp);
if (!inet_aton(dest, &to->sin_addr)) {fprintf(stderr,"Incorrect address for 'to': %s\n",dest);exit(2); }

packetsize=0;

/* build a UDP packet from scratch */

ip=(struct iphdr *)outpack;
ip->version=4;      /* IPv4 */
ip->ihl=5;          /* IP header length: 5 words  */
ip->tos=0;          /* no special type of service */
ip->id=0;           /* no ID */
ip->frag_off=0;     /* not a fragment - so there is no offset */
ip->ttl=0x40;       /* TTL = 64 */

if (!(proto = getprotobyname("udp"))) {perror("getprotobyname(udp)"); exit(2);}

ip->protocol=proto->p_proto;
ip->check=0;        /* null checksum, will be automatically computed by the kernel */
ip->saddr=from->sin_addr.s_addr; /* set source and destination addresses */
ip->daddr=to->sin_addr.s_addr;
/* end of ip header */
```

```
    packetsize+=ip->ihl<<2;
                                    /* udp header */
    udp=(struct udphdr *)((int)outpack + (int)(ip->ihl<<2));
    udp->source=htons(srcp);
    udp->dest=htons(destp);
    udp->check=0;                   /* ignore UDP checksum */
    packetsize+=sizeof(struct udphdr);
    /* end of udp header */

    /* add data to UDP payload if you want: */
    for (datasize=0;datasize<8;datasize++) {
        outpack[packetsize+datasize]='A'+datasize;
    }
    packetsize+=datasize;
    udp->len=htons(sizeof(struct udphdr)+datasize);
    ip->tot_len=htons(packetsize);

    if (sendto(s, (char *)outpack, packetsize, 0, &addrto, sizeof(struct sockaddr))==-1)
       {perror("sendto"); exit(2);}

    printf("packet sent !\n");
    close(s);
    exit(0);
}
```

ICMP Source Quench Error

Since UDP has no flow control, a node could receive datagrams faster than it can process them. In this situation the host **may** send an ICMP source quench.

Note: "**may** be generated" - it is not required to generate this error

Stevens (Vol. 1, pp. 160-161) gives the example of sending 100 1024-byte datagrams from a machine on an ethernet via a router and SLIP line to another machine:

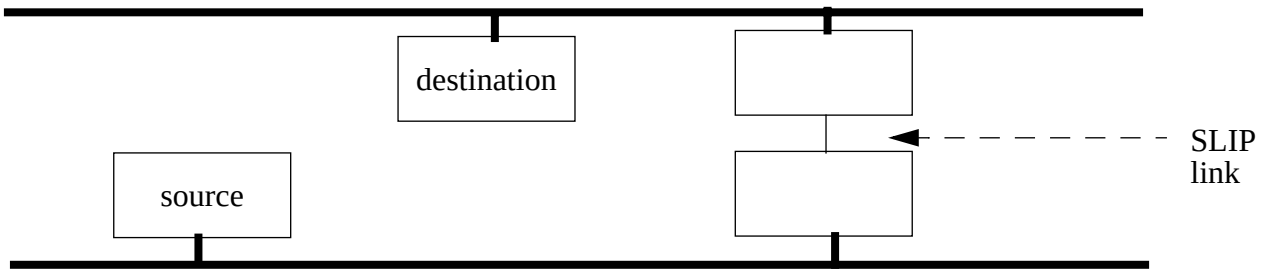


Figure 45: simplified from Stevens, Vol. 1, inside cover

- SLIP link is ~1000 times slower than the ethernet
- 26 datagrams are transmitted, then a source quench is sent for each successive datagram

- the router gets all 100 packets, before the first has been sent across the link!
  - the new Router Requirements RFC - says that routers should not generate source quench errors, since it just consumes network bandwidth and it is an ineffective and unfair fix for congestion
- In any case, the sending program never responded to the source quench errors!
  - BSD implementations ignore received source quenchs if the protocol is UDP
  - the program finished before the source quench was received!

Thus if you want reliability you have to build it in and do end-to-end flow control, error checking, and use (and thus wait for) acknowledgements.

No error control

Since UDP has no error control, the sender has to take responsibility for sending the datagram again if this datagram must be delivered.

But how does the send know if a datagram was successfully delivered?

- Unless the receiver sends a reply (or does some action due to receiving a given datagram) the sender will **not** know! [loss]
- Note that without some additional mechanism - the sender doesn't know if the datagram was delivered multiple times! [duplicates]

If you want reliability you have to build your own protocol on top of UDP to achieve it. This includes deciding on your own retransmission scheme, timeouts, etc.

BOOTP: Bootstrap Protocol (RFC 951)

Although you can figure out who you are, i.e., your IP address, via RARP - many machines want more information.

BOOTP requests and answer are sent via UDP (port 67 server; port 68 client)

- so it is easy to make a user space server
- the client (who wants the answer) need not have a full TCP/IP, it can simply send what **looks** like a UDP datagram with a BOOTP request[1].

Opcode (1=request, 2=reply)	hardware type (1=ethernet)	hardware address length (6 for ethernet)	hop count
transaction ID			
number of seconds		unused	
client IP address			
your IP address			
server IP address			
gateway IP address			
client hardware address (16 bytes of space)			
server hostname (64 bytes)			
Boot file name (128 bytes)			
Vendor specific information (64 bytes)			

1. see Stevens, Vol. 1, figure 16.2, pg. 216

BOOTP continued

When a request is sent as an IP datagram:

- if client does not know its IP address it uses 0.0.0.0
- if it does not know the server's address it uses 255.255.255.255
- if the client does not get a reply, it tries again in about 2 sec.

Vendor specific information (RFC 1497 and RFC1533)

- if this area is used the first 4 bytes are: IP address 99.130.83.99 this is called the "**magic cookie**"
- the rest of the area is a list of items, possibly including:
  - Pad (tag=0);
  - Subnet mask (tag=1);
  - Time offset (tag=2);
  - List of IP addresses of Gateways (tag=3);
  - Time server's IP address (tag=4);
  - Name Server (tag=5);
  - Domain Name Server (tag=6);
  - LOG server (tag=7); …
  - LPR server (tag=9); …
  - this Host's name (tag=12);
  - Boot file size (tag=13); …
  - Domain name (tag=15); …
  - End (tag=255)

DHCP: Dynamic Host Configuration Protocol (RFC 1531)

Extends the Vendor specific options area by 312 bytes.

This protocol is designed to make it easier to allocate (and reallocate) addresses for clients. DHCP defines:

- Requested IP Address - used in client request (DHCPDISCOVER) to request that a particular IP address (tag=50)
- IP Address Lease Time - used in a client request (DHCPDISCOVER or DHCPREQUEST) to request a lease time for the IP address. In a server reply (DHCPOFFER), specific lease time offered. (tag=51)
- Option Overload - used to indicate that the DHCP “sname” or “file” fields are being overloaded by using them to carry DHCP options. A DHCP server inserts this option if the returned parameters will exceed the usual space allotted for options, i.e., it uses the sname and file fields for another purpose! (tag=52)

- DHCP Message Type - the type of the DHCP message (tag=53)

Message Type	purpose
1	DHCPDISCOVER
2	DHCPOFFER
3	DHCPREQUEST
4	DHCPDECLINE
5	DHCPACK
6	DHCPNAK
7	DHCPRELEASE

- Server Identifier - used in DHCPOFFER and DHCPREQUEST (optionally in DHCPACK and DHCPNAK) messages. Servers include this in the DHCPOFFER to allow the client to distinguish **between** lease offers. DHCP clients indicate which of several lease offers is being accepted by including this in a DHCPREQUEST message. (tag=54)
- Parameter Request List - used by a DHCP client to request values for specified configuration parameters. The client **may** list options in order of preference. The DHCP server **must** try to insert the requested options in the order requested by the client. (tag=55)

- Message - used by a server to provide an error message to client in a DHCPNAK message in the event of a failure. A client may use this in a DHCPDECLINE message to indicate the reason **why** the client declined the offered parameters.(tag=56)
- Maximum DHCP Message Size - specifies the maximum length DHCP message that it is willing to accept. A client may use the maximum DHCP message size option in DHCPDISCOVER or DHCPREQUEST messages, but should not use the option in DHCPDECLINE messages. (tag=57)
- Renewal (T1) Time Value - specifies the time interval from address assignment until the client transitions to the RENEWING state. (tag=58)
- Rebinding (T2) Time Value - specifies the time interval from address assignment until the client transitions to the REBINDING state.(tag=59)
- Class-identifier - used by DHCP clients to optionally identify the type and configuration of a DHCP client. Vendors and sites may choose to define specific class identifiers to convey particular configuration or

other identification information about a client. Servers not equipped to interpret the class-specific information sent by a client **must** ignore it (although it may be reported). (tag=60)

- Client-identifier - used by DHCP clients to specify their unique identifier. DHCP servers use this value to index their database of address bindings. This value is expected to be unique for all clients in an administrative domain. (tag=61)

DHCP's importance

- allows reuse of address, which avoids having to tie up addresses for systems which are not currently connected to the Internet
- avoids user configuration of IP address (avoids mistakes and effort)
- allows recycling of an IP address when devices are scrapped
- …

How big a problem is manual configuration?

A large site (such as DuPont Co. - a large chemical company) has over 65,000 IP addressable devices; or consider what happens if each of the 815,000 Wal-Mart employees has an IP device

Address management software

Product	Vendor	URL
Network Registrar	Cisco	http://www.cisco.com
NetID	Nortel Networks	http://www.nortelnetworks.com
Meta IP 4.1	CheckPoint	http://www.metaip.checkpoint.com
QIP Enterprise 5.0	Lucent Technologies	http://qip.lucent.com

DHCP performance problems

Most implementations of DHCP do a duplicate address detection (DAAD) test after they have picked an address to assign.

An alternative approach to speed up the DHCP process does the duplicate address detection process in the background (in advance) so that you will have a set of recently tested addresses to hand out:

> Jon-Olov Vatn and Gerald Q. Maguire Jr., "The effect of using co-located care-of addresses on macro handover latency", Fourteenth Nordic Tele-traffic Seminar (NTS 14), August 18 - 20, 1998, Lyngby, Denmark.
>
> *http://www.it.kth.se/~vatn/research/nts14-coloc.pdf*

The result is that a DHCP request can be answered in less than 100ms.

Example of dhcpd.conf

```
### Managed by Linuxconf, you may edit by hand.
### Comments may not be fully preserved by linuxconf.
server-identifier dhcptest1;
default-lease-time 1000;
max-lease-time 2000;
option domain-name           "3ctechnologies.se";
option domain-name-servers   130.237.12.2;
option host-name             "s1.3ctechnologies.se";
option routers               130.237.12.2;
option subnet-mask           255.255.255.0;
subnet 130.237.12.0 netmask 255.255.255.0 {
          range  130.237.12.3 130.237.12.200;
          default-lease-time 1000;
          max-lease-time 2000;
}
subnet 130.237.11.0 netmask 255.255.255.0 {
           range  130.237.11.3 130.237.11.254;
           default-lease-time 1000;
           max-lease-time 2000;
}
```

DHCP and DNS

- There is no dynamic host name assignment yet.
- Interaction between DHCP and DNS is needed.

For example: once a host is assigned an IP address the DNS should be updated dynamically:

- If the host hasn't got a name: it should assign a name along the IP address assignment (no DNS update is needed).
- If the host has already a name: the DNS should be dynamically updated once the host has gotten a new IP address from DHCP.

The IETF's Dynamic Host Configuration (dhc) Working group *http://www.ietf.org/html.charters/dhc-charter.html* is working on addressing the issues concerning interaction between DHCP and DNS.

Trivial File Transfer Procotol (TFTP)

TFTP uses UDP (unlike FTP which uses TCP)

- simple and small
- requires only UDP, IP, and a device driver - easily fits in ROM
- a stop-and-wait protocol
- lost packets detected by timeout and retransmission
- Two operations:
  - Read Request (RRQ)
  - Write Request (RRQ) - for security reasons the file must already exist
- The TFTP server (“tftpd”) is generally run setrooted (i.e., it only has access to is own directory) and with a special **user** and **group** ID - since there is no password or other protection of the access to files via TFTP!
- TFTP request is sent to the well know port number (69/udp)
- TFTP server uses an unused ephemeral port for its replies
  - since a TFTP transfer can last for quite some time - it uses another port; thus freeing up the well known port for other requests

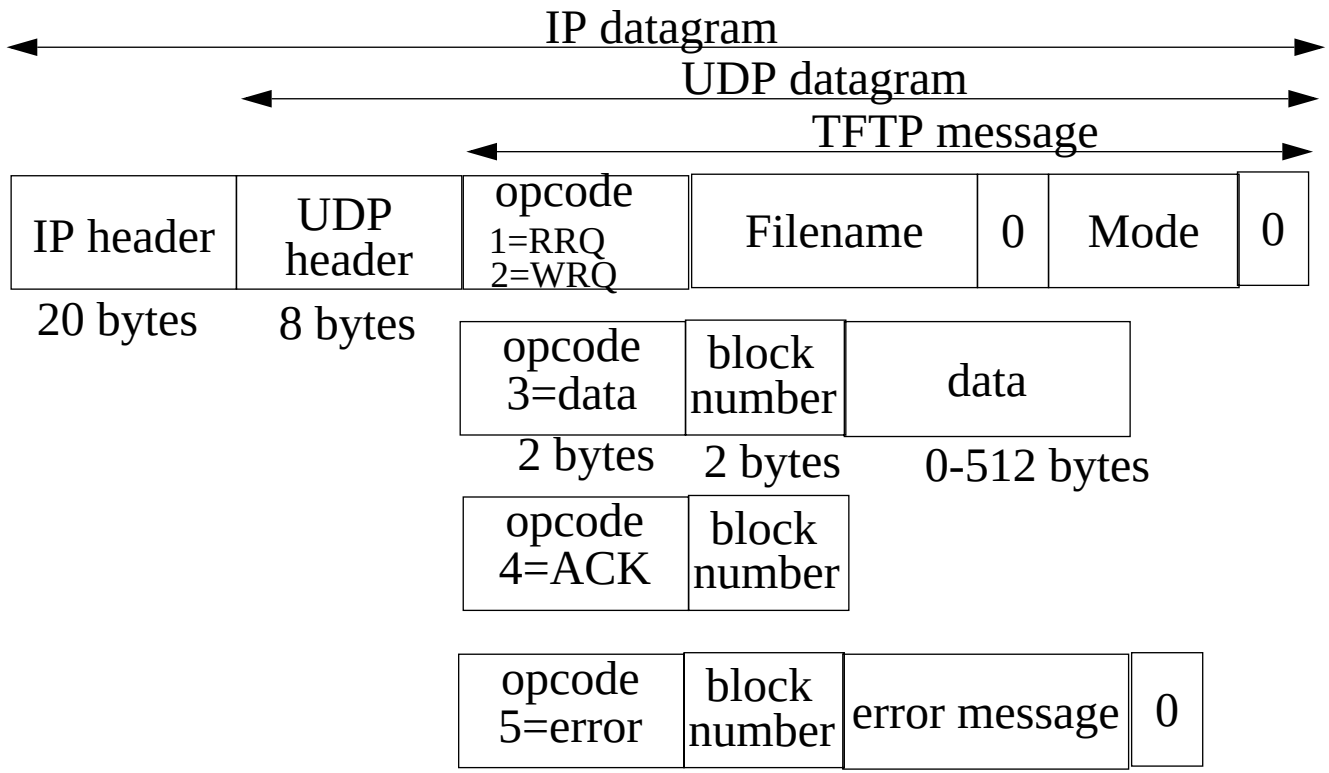


Figure 46: TFTP messages (see Stevens, Vol. 1, figure 15.1, pg. 210)

Filename and Mode (“netascii” or “octet”) are both N bytes sequences terminated by a null byte.

Widely used for bootstrapping diskless systems (such as X terminals) and for dumping the configuration of routers (this is where the write request is used)

Mapping names to IP addresses

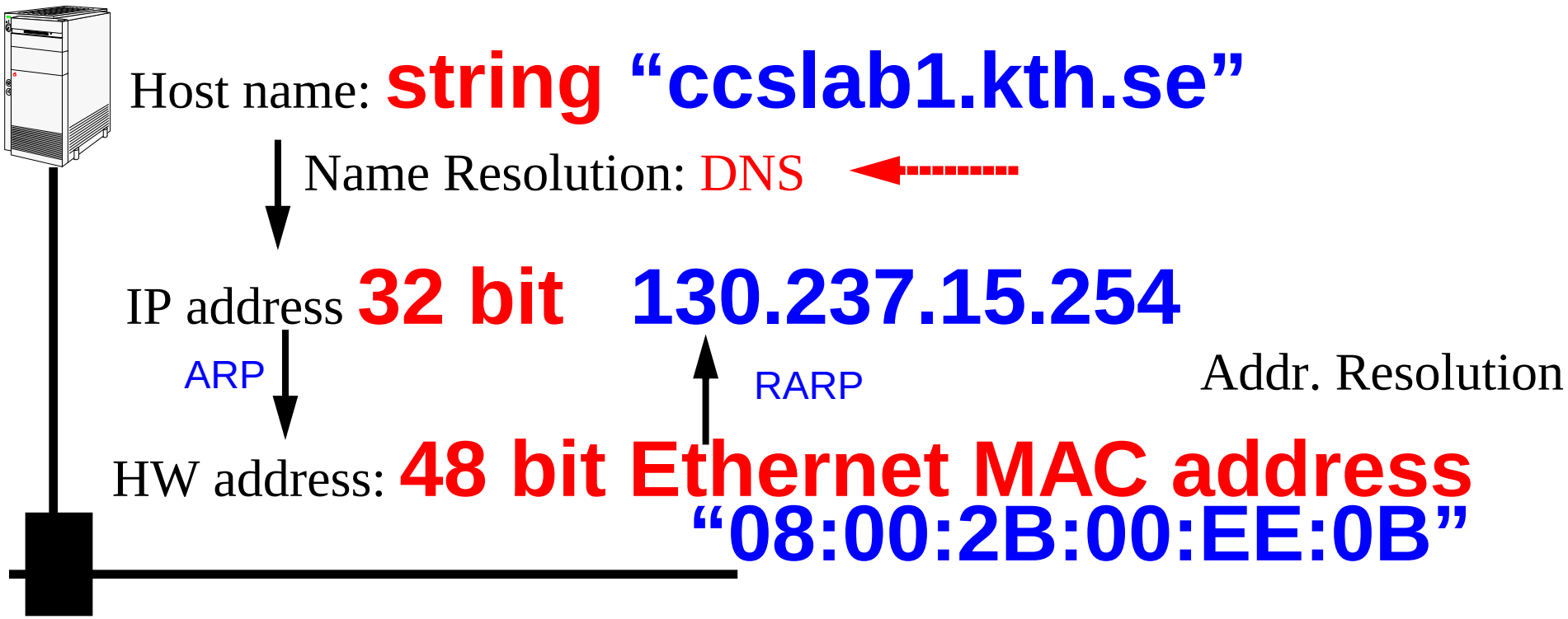


Figure 47: mapping between host names and IP address(es)

DNS: Domain Name Service (RFC 1034, RFC 1035)

- To make the network more user friendly
- Distributed database (with caching) providing:
  - hostname ⇒ IP address, IP address⇒ hostname
  - mailbox ⇒ mail server
  - …
- applications call a “resolver”
  - gethostbyname: hostname ⇒ IP address
  - gethostbyaddr: IP address⇒ hostname
- Resolver’s contact name servers (see “/etc/resolv.conf”)
- DNS names:
  - domain name: list of labels from a root, i.e., www.imit.kth.se
  - Fully Qualified Name (FQDN): a domain name ending in “.” - there are no further labels
  - leaves are managed locally through delegation of authority (to a zone) **not** centrally; this allows scaling
  - if a name server does not know the answer it asks other name servers
    - every name server **must** know how to contact a root server
- Uses UDP (for query) and TCP (zone transfer and large record query)

Zones

A zone is a subtree of the DNS tree which is managed separately.

Each zone must have multiple name servers:

- a primary name server for the zone
  - gets its data from disk files (or other stable store)
  - must know the IP address of one or more root servers
- one or more secondary name servers for the zone
  - get their data by doing a zone transfer from a primary
  - generally query their primary server every ~3 hours

To find a server you may have to walk the tree up to the root or possibly from the root down (but the later is not friendly).

DNS Message format

0 16 31

Identification	Parameters
Number of Questions	Number of Answers
Numer of authority	Number of Additional
Question section …	
Answer section …	
Additional Information section …	

Bit or Parameter field	Meaning
0	Operation: 0=Query, 1=Response
1-4	Query type: 0=standard, 1=Inverse
5	Set if answer is authoritative
6	Set if answer is truncate
7	Set if answer is desired
8	Set if answer is available
9-11	reserved
12-15	Response Type: 0=No error, 1=Format error in query, 2=Server failure, 3=Name does not exist

Internet's top level domains

(see Stevens, Vol. 1, figure 14.2, pg. 189)

Domain	Description
com	commercial organizations
edu	educational organizations
gov	other U.S. government organizations (see RFC 1811 for policies)
int	international organizations
mil	U.S. Military
net	networks
org	other organizations
arpa	special domain for address to name mappings, e.g., 5.215.237.130.in-addr.arpa
ae	United Arab Emerates
…	
se	Sweden
zw	Zimbabwe

Lots of interest in having subdomains of "com"

- companies registering product names, etc. - in some cases asking for 10s of addresses
- who gets to use a given name? problems with registered trade marks, who registered the name first, … [How much is a name worth?]

New top level domains

There is a proposed new set of top level domains and an increase in the number of entities which can assign domain names.

Generic Top Level Domains (gTLDs), November 16, 2000:

.aero	for the entire aviation community
.biz	for business purpose
.coop	for cooperatives
.info	unrestricted
.museum	for museums
.name	for personal names
.pro	for professionals

CORE (Council of Registrars) - operational organization composed of authorized Registrars for managing allocations under gTLDs.

WIPO provides arbitration concerning names:

http://arbiter.wipo.int/domains/gtld/newgtld.html

Domain registrars

Internet Corporation for Assigned Names and Numbers (ICANN) Accredited Registrars, the full list is at *http://www.icann.org/registrars/accredited-list.html*

Even more registrars are on their way to being accredited and operating!

Country Code Top-Level Domains (CCTLDs)

http://www.iana.org/cctld/cctld-whois.htm

For Sweden (SE) SE-DOM, the NIC is: http://www.nic-se.se/

	Administrative contact	**Technical contact**
Address	II-Stiftelsen Sehlstedtsgatan 7 SE-115 28 Stockholm Sweden	Network Information Centre Sweden NIC-SE Box 5774 SE-114 87 Stockholm Sweden
e-mail	ii-stiftelsen@iis.se	hostmaster@nic-se.se
phone	+46 8 56849050	+46 8 54585700
fax	+46 8 50618470	+46 8 54585729

The above is from http://www.iana.org/root-whois/se.htm

URL for registration services: http://www.iis.se/

Resource Records (RR)

See Stevens, Vol. 1, figure 14.2, pg. 201 (augmented with additional entires)

Record type	Description
A	an IP address. Defined in RFC 1035
AAAA	an IPv6 address. Defined in RFC 1886
PTR	pointer record in the in-addr.arpa format. Defined in RFC 1035.
CNAME	canonical name≡ alias (in the format of a domain name). Defined in RFC 1035.
HINFO	Host information. Defined in RFC 1035.
MX	Mail eXchange record. Defined in RFC 1035.
NS	authoritative Name Server (gives authoritative name server for this domain).Defined in RFC 1035.
TXT	other attributes. Defined in RFC 1035.
AFSDB	AFS Data Base location. Defined in RFC 1183.
ISDN	ISDN. Defined in RFC 1183.
KEY	Public key. Defined in RFC 2065.
KX	Key Exchanger. Defined in RFC 2230.
LOC	Location. Defined in RFC 1876.
MG	mail group member. Defined in RFC 1035.
MINFO	mailbox or mail list information. Defined in RFC 1035.
MR	mail rename domain name. Defined in RFC 1035.
NULL	null RR. Defined in RFC 1035.
NS	Name Server. Defined in RFC 1035.

See Stevens, Vol. 1, figure 14.2, pg. 201 (augmented with additional entires)

Record type	Description
NSAP	Network service access point address. Defined in RFC 1348. Redefined in RFC 1637. Redefined in RFC 1706.
NXT	Next. Defined in RFC 2065.
PX	Pointer to X.400/RFC822 information. Defined in RFC 1664.
RP	Responsible Person. Defined in RFC 1183.
RT	Route Through. Defined in RFC 1183.
SIG	Cryptographic signature. Defined in RFC 2065.
SOA	Start Of Authority. Defined in RFC 1035.
SRV	Server. DNS Server resource record -- RFC 2052, for use with DDNS.
TXT	Text. Defined in RFC 1035.
WKS	Well-Known Service. Defined in RFC 1035.
X25	X25. Defined in RFC 1183.

Note that an number of the RR types above are for experimental use.

Name of an organization:

```
ISI.EDU.        PTR     0.0.9.128.IN-ADDR.ARPA.
```

Network names

Conventions:

- it.kth.se includes all the computers in the KTH/SU IT-University
- kth.se includes all the computers at KTH
- …

As resource records:

```
> set querytype=any
> kth.se
…
Non-authoritative answer:
kth.se                  internet address = 130.237.72.201
kth.se

                        origin = kth.se
                        mail addr = hostmaster.kth.se
                        serial = 2002011500
                        refresh = 3600 (1H)
                        retry   = 600 (10M)
                        expire  = 604800 (1W)
                        minimum ttl = 86400 (1D)
```

kth.se	nameserver = kth.se
kth.se	nameserver = nic.lth.se
kth.se	nameserver = ns.kth.se

Authoritative answers can be found from:

kth.se	nameserver = kth.se
kth.se	nameserver = nic.lth.se
kth.se	nameserver = ns.kth.se
kth.se	internet address = 130.237.x.y
nic.lth.se	internet address = 130.235.z.w
ns.kth.se	internet address = 130.237.m.n

ARPANET.ARPA. PTR 0.0.0.10.IN-ADDR.ARPA.

isi-net.isi.edu. PTR 0.0.9.128.IN-ADDR.ARPA.

Example:

```
$ORIGIN it.kth.se.
@                       1D IN SOA       bbbb hostmaster (
                                                2002012001      ; serial
                                                8H              ; refresh
                                                2H              ; retry
                                                2W              ; expiry
                                                8H )            ; minimum

                1D      IN NS           ns.ele.kth.se.
                1D      IN NS           ns.kth.se.
                1D      IN MX           0 mail
                1D      IN A            130.237.x.y
                1D      IN AFSDB        1 xxxx
                1D      IN AFSDB        1 yyyy
                1D      IN AFSDB        1 zzzz
```

MX information

```
> set querytype=MX
> kth.se
…
kth.se                  preference = 0, mail exchanger = mail1.kth.se
kth.se                  nameserver = kth.se
kth.se                  nameserver = nic.lth.se
kth.se                  nameserver = ns.kth.se
mail1.kth.se            internet address = 130.237.32.62
kth.se                  internet address = 130.237.72.201
nic.lth.se              internet address = 130.235.20.3
ns.kth.se               internet address = 130.237.72.200
```

Another examine in MX RR format:

```
1D      IN MX       10 xxx.e.kth.se.
1D      IN MX       20 yyy.e.kth.se.
```

Host names and info

How to give your host a name?

see RFC 1178: Choosing a Name for Your Computer

Internet Addresses: A second address for your host?

- to have multiple addresses for you computer, see section on “ifconfig”

Hostinfo (HINFO)

```
> set querytype=HINFO
> kth.se
…
kth.se                          CPU = sun-4/60          OS = unix
```

Entry	owner	clas	TTL	RR type	value	comment
xxxx			1D	IN HINFO	"PC" "FLINUX"	; “CPU” “OS”
			1D	IN A	130.237.x.y	

Storing other attributes - TXT records

The general syntax is:

<owner> <class> <ttl> TXT “<attribute name>=<attribute value>”

Examples:

host.widgets.com IN TXT “printer=lpr5”

sam.widgets.com IN TXT “favorite drink=orange juice”

For more information see:

RFC 1464: *Using the Domain Name System To Store Arbitrary String Attributes*

Configuring DNS

- Configuring the BIND resolver
  - /etc/resolv.config
- Configuring the BIND nameserver (named)
  - /etc/named.boot or /etc/named.config
- Configuring the nameserver database files (zone files)
  - named.hosts the zone file that maps hostnames to IP addresses
  - named.rev the zone file that maps IP addresses to hostnames

Root servers

ROOT-SERVERS.NET		IP address(es)	Home ASN	Location(s)
A	Verisign Global Registry Services	198.41.0.4	19836	Herndon, VA, US
B	EP.Net	192.228.79.201 2001:478:65::53		Marina del Rey, CA, US
C	Cogent Communications	192.33.4.12	2149	Herndon VA; Los Angeles; New York City; Chicago
D	University of Maryland	128.8.10.90	27	College Park, MD, US
E	NASA Ames Research Centre	192.203.230.10	297	Mountain View, CA, US
F	Internet Systems Consortium (ISC)	192.5.5.241 2001:500::1035	3557	Ottawa; Palo Alto; San Jose CA;New York City; San Francisco;Madrid; Hong Kong; Los Angeles;Rome; Auckland; Sao Paulo;Beijing; Seoul; Moscow; Taipei;Dubai; Paris; Singapore; Brisbane;Toronto; Monterrey; Lisbon;Johannesburg; Tel Aviv; Jakarta;Munich; Osaka; Prague
G	US Department of Defence	192.112.36.4	568	Vienna, VA, US
H	US Army Research Lab	128.63.2.53 2001:500:1::803f:235	13	Aberdeen, MD, US
I	Autonomica/NORDUnet	192.36.148.17	29216	Stockholm; Helsinki; Milan;London; Geneva; Amsterdam;Oslo; Bangkok; Hong Kong;Brussels; Frankfurt; Ankara; Bucharest;Chicago; Washington DC;Tokyo; Kuala Lumpur;Palo Alto; Wellington
J	Verisign Global Registry Services	192.58.128.30	26415	Herndon, VA, US
K	Reseaux IP Europeens (RIPE)	193.0.14.129 2001:7fd::1	25152	London (UK); Amsterdam (NL);Frankfurt (DE); Athens (GR);Doha (QA); Milan (IT);Reykjavik (IS); Helsinki (FI);Geneva (CH); Poznan (PL);Budapest (HU)
L	IANA	198.32.64.12	20144	Los Angeles, CA, US
M	WIDE Project	202.12.27.33 2001:dc3::35	7500	Tokyo; Seoul (KR); Paris (FR)

see *http://www.root-servers.org/*

Load leveling [29]

For example, f.root-servers.net has a **single** IP address (192.5.5.241), but requests sent to 192.5.5.241 are routed to *different* nameservers

This routing can depend on:

- where the request is made from
- what the load on each of these names servers is

Note: this is transparent to the host which sent the request to F

ISC uses Hierarchical Anycast routing to do this, with some of the servers being:

- large, redundant, … installations serving the *global* internet
- small installations serving a *local* region
- 28 nodes as of Feb. 2005

F root nameserver nodes

	Site Code	Location	IPv4/IPv6	Node Type
Asia Pacific	AKL1	Auckland	IPv4	Local Node
	BNE1	Brisbane	IPv4	Local Node
	CGK1	Jakarta	IPv4	Local Node
	HKG1	Hong Kong	IPv4	Local Node
	KIX1	Osaka	IPv4 and IPv6	Local Node
	PEK1	Beijing	IPv4	Local Node
	SEL1	Seoul	IPv4 and IPv6	Local Node
	TPE1	Taipei	IPv4	Local Node
Americas	AKL1	São Paulo	IPv4	Local Node
	LAX1	Los Angeles	IPv4 and IPv6	Local Node
	LGA1	New York	IPv4 and IPv6	Local Node
	MTY1	Monterrey	IPv4	Local Node
	PAO1	Palo Alto	IPv4 and IPv6	Global Node
	SFO2	San Francisco	IPv4 and IPv6	Global Node
	SQL1	Redwood City	IPv4 and IPv6	Local Node
	SJC1	San Jose	IPv4	Local Node
	YOW1	Ottawa	IPv4 and IPv6	Local Node
	YYZ1	Toronto	IPv4	Local Node
Rest of World	CDG1	Paris	IPv4 and IPv6	Local Node
	DXB1	Dubai	IPv4	Local Node
	JNB1	Johannesburg	IPv4	Local Node
	LIS1	Lisbon	IPv4 and IPv6	Local Node
	MAD1	Madrid	IPv4	Local Node
	MUC1	Munich	IPv4 and IPv6	Local Node
	PRG1	Prague	IPv4	Local Node
	ROM1	Rome	IPv4	Local Node
	SVO1	Moscow	IPv4	Local Node
	TLV1	Tel Aviv	IPv4	Local Node

Where is f.root-servers.net ?

traceroute to f.root-servers.net (192.5.5.241), 30 hops max, 40 byte packets

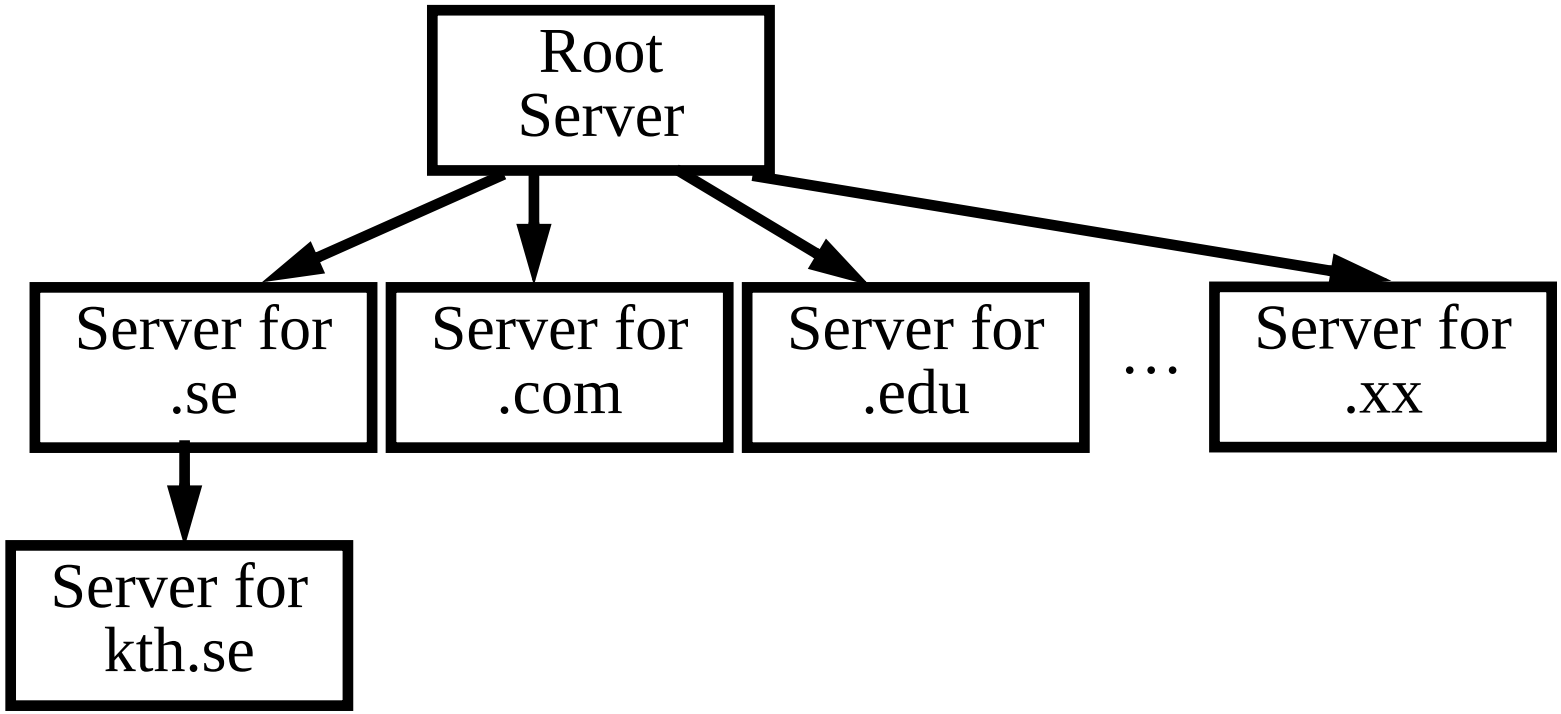
1	net212a.imit.kth.se (130.237.212.2)	1 ms	1 ms	0 ms
2	cn4-kf4-p2p.gw.kth.se (130.237.211.205)	1 ms	1 ms	1 ms
3	ea4-cn4-p2p.gw.kth.se (130.237.211.102)	1 ms	1 ms	1 ms
4	kth2-ea4-b.gw.kth.se (130.237.211.241)	1 ms	1 ms	1 ms
5	stockholm4-SRP2.sunet.se (130.242.85.66)	1 ms	1 ms	1 ms
6	130.242.82.49 (130.242.82.49)	1 ms	1 ms	1 ms
7	se-kth.nordu.net (193.10.252.177)	1 ms	2 ms	4 ms
8	so-1-0.hsa2.Stockholm1.Level3.net (213.242.69.21)	4 ms	5 ms	4 ms
9	so-4-1-0.mp2.Stockholm1.Level3.net (213.242.68.205)	14 ms	8 ms	8 ms
10	as-1-0.bbr2.London1.Level3.net (212.187.128.25)	36 ms	36 ms	36 ms
11	as-0-0.bbr1.NewYork1.Level3.net (4.68.128.106)	106 ms	104 ms	104 ms
12	ae-0-0.bbr2.SanJose1.Level3.net (64.159.1.130)	180 ms	247 ms	180 ms
13	so-14-0.hsa4.SanJose1.Level3.net (4.68.114.158)	*	178 ms	178 ms
14	ISC-Level3-fe.SanJose1.Level3.net (209.245.146.219)	178 ms	180 ms	178 ms
15	f.root-servers.net (192.5.5.241)	180 ms	178 ms	180 ms

Where is i.root-servers.net ?

traceroute to i.root-servers.net (192.36.148.17), 30 hops max, 40 byte packets

1	net212a.imit.kth.se (130.237.212.2)	1 ms	0 ms	0 ms
2	cn4-kf4-p2p.gw.kth.se (130.237.211.205)	1 ms	1 ms	4 ms
3	cn5-cn4-p2p.gw.kth.se (130.237.211.201)	4 ms	4 ms	4 ms
4	kth1-cn5-p2p.gw.kth.se (130.237.211.41)	4 ms	4 ms	4 ms
5	stockholm3-SRP2.sunet.se (130.242.85.65)	4 ms	4 ms	4 ms
6	ge-2-2.cyb-gw.sth.netnod.se (194.68.123.73)	2 ms	1 ms	1 ms
7	ge-2-1.icyb-gw.sth.netnod.se (192.36.144.190)	1 ms	1 ms	2 ms
8	srp-1-1.ibyb-gw.sth.netnod.se (192.36.144.235)	1 ms	1 ms	1 ms
9	ge-0-0.r1.sth.dnsnode.net (194.146.105.187)	1 ms	2 ms	2 ms
10	i.root-servers.net (192.36.148.17)	3 ms	3 ms	2 ms

Dynamic Domain Name System (DDNS)



Host Name	IP-address
host_a	130.237.x.1
host_b	130.237.x.2
host_c	130.237.x.3
host_d	130.237.x.4
mobile1	c/o address <<< we can update this dynamically

DDNS

RFC 2136: Dynamic Updates in the Domain Name System (DNS UPDATE)

- add or delete resource records

RFC 2052: A DNS RR for specifying the location of services (DNS SRV)

- When a SRV-cognizant web-browser wants to retrieve

  http://www.asdf.com/

  it does a lookup of

  http.tcp.www.asdf.com

RFC 2535: Domain Name System Security Extensions (DNSSec)

Summary

This lecture we have discussed:

- UDP
- BOOTP
- DHCP
- DNS, DDNS

References

[29] Joe Abley, f.root-servers.net, NZNOG 2005, February 2005, Hamilton, NZ

http://www.isc.org/pubs/pres/NZNOG/2005/F%20Root%20Server.pdf