

LTSP protocol review

- the LTSP client has no hard disk, no pen drive, no floppy disk and no cdrom
- it boots up using the network
- when the LTSP client boots up it uses standard network protocols
 - initially it needs to get an IP address, gateway IP address, DNS address and network address
 - it uses DHCP which provides all the above
 - DHCP can also tell it the operating system file it needs to download
 - and which tftp server to download it from
 - the DHCP server also allows options to be passed into the kernel

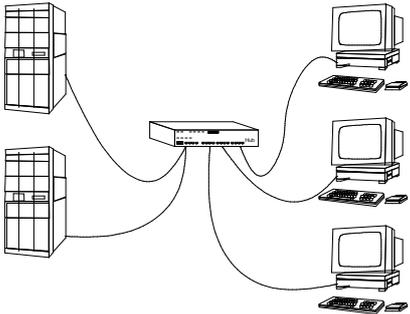
LTSP protocol review

- TFTP server hosts the linux kernel
- once the kernel is loaded into client it is executed and it examines the options
 - these indicate the NFS server from where the root / directory is to be mounted
 - see option `root-path` in the dhcp configuration example

LTSP protocol review

- it mounts / from the prescribed server and runs a program called `/sbin/init`
- this runs all the start up programs and eventually it runs the graphical login session
- this is a client for the protocol which connects to the server
 - this is specified in the `/etc/lts.conf` file (found in the mounted filesystem)

Dynamic IP Address Assignment

- 
 - statically assigning IP addresses to computers has limitations
 - computer moves from one subnet to another
 - or if physical number of computers exceed available IP addresses

BOOTP Limitations

- BOOTP provides a configuration setting for a client
 - IP address, operating system, subnet mask, dns server, gateway address
 - a set of static settings from a host identifier to parameters for that host
- a manager must enter parameters for each host on the server
 - BOOTP does not dynamically allocate IP addresses
- nowadays it is common place for laptops to be join/leave different subnet networks

Dynamic Host Configuration Protocol (DHCP)

- has many advantages over BOOTP
- clients can obtain all configuration details in a single message
 - IP address, subnet mask, gateway, operating system server IP address
 - IP addresses allocated dynamically
- when a client boots it broadcasts a DHCP message on the local network
 - client receives (possibly multiple DHCP server offers)
 - selects one server and verifies acceptance
- can build a highly reliable system by using multiple servers

DHCP IP address assignment

- different types of IP address assignment
 - *manual configuration*
 - eg. manager allocates specific addresses per Ethernet address
 - *automatic configuration*
 - DHCP server allocates a permanent IP address when a client boots
 - *dynamic configuration*
 - DHCP server *loans* IP address for a limited time
- DHCP servers can be configured to manage different clients with the three different IP address assignments

DHCP leased IP addresses

- DHCP specifies a minimum lease period of 1 hour
 - managers may specify default as 1 hour..oo
- client can release an IP address or request an extension to the lease

tftp protocol

- tftp code is simple, small and is ideal to be placed into ROM as part of a systems bootstrap process
- unlike ftp it uses UDP/IP rather than TCP/IP
- it implements its own reliable recovery mechanism (sliding window of size 1 otherwise known as idlerq)

tftp protocol

- operation is simple and the designers of tftp have fought to keep it this way over the 20+ years
- sending side (server) sends a file in 512 byte blocks
 - each block is sent
 - sender waits for an acknowledgement, before sending the next
- receiver sends an acknowledgement each time it receives a block

tftp protocol

- client
 - initially sends a packet requesting a file transfer
 - packet contains:

2 octet opcode	n octets	1 octet	n octets	1 octet
readreq (1)	filename	0	mode	0
 - trusted clients may also write a file to the server in which case the first two octets have the value 2
 - blocks are numbered starting at 1 and each block is 512 bytes
 - any block shorter than 512 bytes indicates the last block (eof)

tftp packet types

- client write request

2 octet opcode	n octets	1 octet	n octets	1 octet
writereq (2)	filename	0	mode	0
- data packet

2 octet opcode	2 octets	up to 512 octets
data (3)	block #	data octets...
- acknowledgement

2 octet opcode	2 octets
ack (4)	block #

tftp packet types

- error packet

2 octet opcode	2 octets	n octets	1 octet
error (5)	error code	error message	0

- tftp is symmetric

- each side implements a timeout and retransmission

- if sending side times out (ie it does not receive an ACK)

- it retransmits the last block

- if reading side does not receive a block within a time period it transmits an ACK

tftp configuration in GNU/Linux

- consists of installing the tftpd package and adding an entry to the inetd.conf file

```
tftp dgram udp wait nobody \
/usr/sbin/tcpd /usr/sbin/in.tftpd \
--tftpd-timeout 300 --retry-timeout 5 \
--mcast-port 1758 --mcast-addr 239.239.239.0-255 \
--mcast-ttl 1 --maxthread 100 --verbose=5 /tftpboot
```

- important fields are:

- /tftpboot which limits the tftp daemon to transmitting files which are contained below: /tftpboot
 - otherwise there could be a security problem
 - configuration details tell inetd that this tftp is a UDP protocol and that it can serve the files using the multicast feature of IP

Pseudo code for tftp (IdleRQ with symmetric retransmission)

```
transmitTFTPfile () {
    frameno := 0 ;
    while true do {
        m := createNextFrameFromFile(frameno)
        transmitFrameOverUDP(m, frameno)
        repeat
            event := waitForEvent() ;
            case event of

                ackArrived : (* correct ack number arrived *) |
                nackArrived: transmitFrameOverUDP(m, frameno) |
                timeout    : transmitFrameOverUDP(m, frameno)

            esac
        until ackArrived
        inc(frameno)
    }
}
```

tftp client pseudo code

```
receiveFile () {
    lastblock := 0
    newblock := 0 ;
    repeat
        repeat
            event := getFrameFromUDP(m) ;
            case event of

                data    : newblock := getBlockNoFromFrame(m) ;
                        transmitAckOverUDP(newblock) |
                timeout: transmitAckOverUDP(lastblock)

            esac
            until newblock==lastblock+1 ;
            lastblock = newblock ;
            storeFrameInMemory(m)
        until frame size < 512 bytes
    }
}
```

Tutorial: Idle RQ Protocol

- warning the code below in this tutorial has a deliberate bug in it

Tutorial: Idle RQ Protocol

- suppose you implement an Idle RQ protocol where the transmitter code is summarised as:

```
n = 1
loop
  sends a frame of data (n)
  event = waitforevent();
  case event of
    timeout: break;
    ack    : n = ack_number()+1 ; break;
    nack   : break
  end
end
end
```

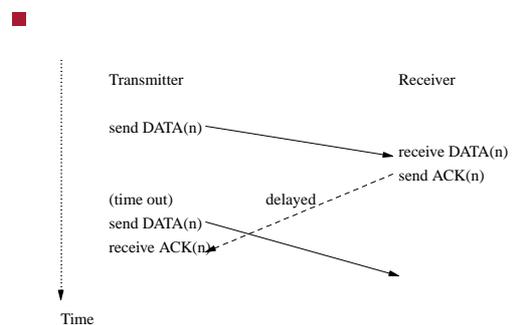
Tutorial: Idle RQ Protocol

- the receiver operates by:

```
receive process
loop
  event = waitforevent();
  case event of
    received frame: if not duplicate data frame(n)
                    then
                      give frame to higher layer(n)
                    fi
                    send_ack_number (n)
  end
end
end
```

Tutorial: Idle RQ Protocol

- on running the code and examining the packets sent across the network it is observed that the following packets are transmitted:



Tutorial: Idle RQ Protocol

- write down the next six frame numbers which are transmitted and complete the following diagram
- assume that only ACK(n) is delayed, causing one timeout.