

# Computer Networks '2018

Jürgen Schönwälder

Jacobs University Bremen

August 20, 2018



JACOBS  
UNIVERSITY



<http://cnds.eecs.jacobs-university.de/courses/cn-2018/>

# Computer Networks and Distributed Systems

- Introduction to Computer Science 1st Semester
- Programming in C I 1st Semester
- Algorithms and Data Structures 2nd Semester
- Programming in C II 2nd Semester
- Computer Architecture and Programming Languages 3rd Semester
- Operating Systems 3rd Semester
- Computer Networks 4th Semester
- Distributed Algorithms 6th Semester
- Project and Bachelor Thesis 5th/6th Semester

# Course Objectives

- Introduce fundamental data networking concepts
- Focus on widely deployed Internet protocols
- Prepare students for further studies in networking
- Combine theory with practical experiences
- Raise awareness of weaknesses of the Internet



# Course Content

1. Introduction
2. Fundamental Networking Concepts
3. Local Area Networks (IEEE 802)
4. Internet Network Layer (IPv4, IPv6)
5. Internet Routing (RIP, OSPF, BGP)
6. Internet Transport Layer (UDP, TCP)
7. Firewalls and Network Address Translators
8. Domain Name System (DNS)
9. Abstract Syntax Notation 1 (ASN.1)
10. External Data Representation (XDR)
11. Augmented Backus Naur Form (ABNF)
12. Electronic Mail (SMTP, IMAP)
13. Document Access and Transfer (HTTP, FTP)

# Reading Material

- A.S. Tanenbaum, "Computer Networks", 4th Edition, Prentice Hall, 2002
- W. Stallings, "Data and Computer Communications", 6th Edition, Prentice Hall, 2000
- C. Huitema, "Routing in the Internet", 2nd Edition, Prentice Hall, 1999
- D. Comer, "Internetworking with TCP/IP Volume 1: Principles Protocols, and Architecture", 4th Edition, Prentice Hall, 2000
- J.F. Kurose, K.W. Ross, "Computer Networking: A Top-Down Approach Featuring the Internet", 3rd Edition, Addison-Wesley 2004.

# Grading Scheme

- Final examination (40%)
  - Covers the whole lecture
  - Closed book (and closed computers / networks)
- Quizzes (30%)
  - Control your continued learning success
  - 6 quizzes with 10 pts each
  - 50 pts and above equals 30% of the overall grade
- Assignments (30%)
  - Learning by solving assignments
  - Implement network protocols
  - Gain some practical experience in a lab session
  - 6 assignments with 10 pts each
  - 50 pts and above equals 30% of the overall grade

# Part 1: Introduction

- 1 Internet Concepts and Design Principles
- 2 Structure and Growth of the Internet
- 3 Internet Programming with Sockets

# Section 1: Internet Concepts and Design Principles

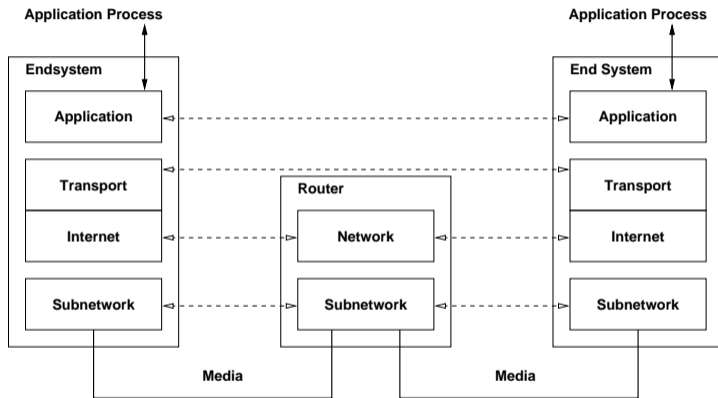
1 Internet Concepts and Design Principles

2 Structure and Growth of the Internet

3 Internet Programming with Sockets



# Internet Model



- Subnetworks only have to provide very basic services
- Even the Internet layer can be used as a subnetwork

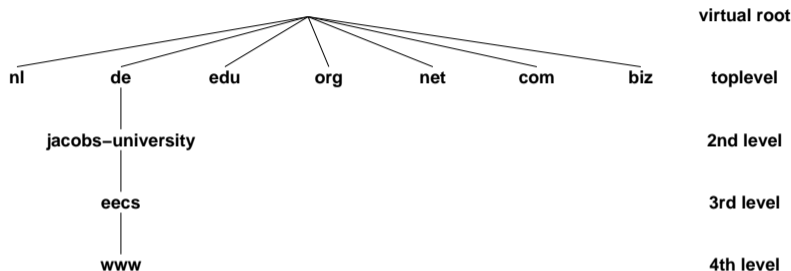
# Internet Design Principles

- Connectivity is its own reward
- All functions which require knowledge of the state of end-to-end communication should be realized at the endpoints (end-to-end argument)
- There is no central instance which controls the Internet and which is able to turn it off
- Addresses should uniquely identify endpoints
- Intermediate systems should be stateless wherever possible
- Implementations should be liberal in what they accept and stringent in what they generate
- Keep it simple (when in doubt during design, choose the simplest solution)

# Internet Addresses

- Four byte IPv4 addresses are typically written as four decimal numbers separated by dots where every decimal number represents one byte (dotted quad notation). A typical example is the IPv4 address 212.201.48.1
- Sixteen byte IPv6 addresses are typically written as a sequence of hexadecimal numbers separated by colons (:) where every hexadecimal number represents two bytes
- Leading zeros in IPv6 addresses can be omitted and two consecutive colons can represent a sequence of zeros
- The IPv6 address 1080:0:0:0:8:800:200c:417a can be written as 1080::8:800:200c:417a
- See RFC 5952 for the recommended representation of IPv6 addresses

# Internet Domain Names



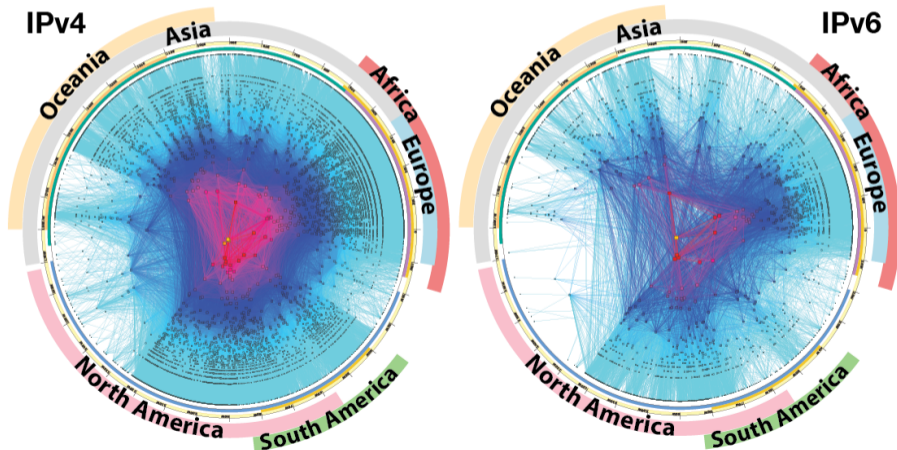
- The Domain Name System (DNS) provides a distributed hierarchical name space which supports the delegation of name assignments
- DNS name resolution translates DNS names into one or more Internet addresses

# Autonomous Systems

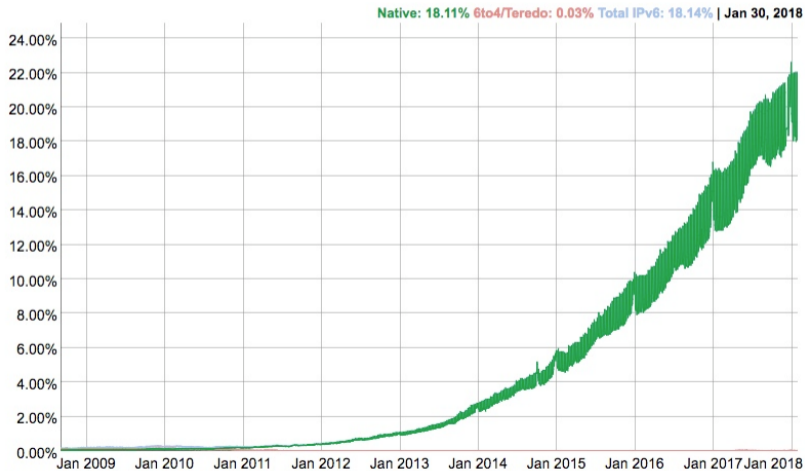
- The global Internet consists of a set of inter-connected autonomous systems
- An *autonomous system* (AS) is a set of routers and networks under the same administration
- Autonomous systems are historically identified by 16-bit numbers, called the AS numbers (meanwhile the number space has been enlarged to 32-bit numbers)
- IP packets are forwarded between autonomous systems over paths that are established by an *Exterior Gateway Protocol* (EGP)
- Within an autonomous system, IP packets are forwarded over paths that are established by an *Interior Gateway Protocol* (IGP)

## CAIDA's IPv4 vs IPv6 AS Core AS-level Internet Graph

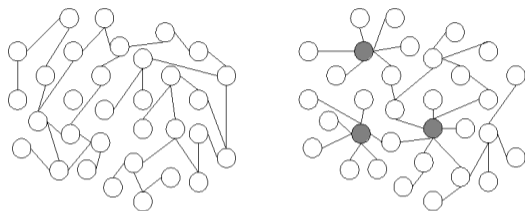
Archipelago July 2015



# Google IPv6 Adoption Statistics



# Internet – A Scale-free Network?



- Scale-free: The probability  $P(k)$  that a node in the network connects with  $k$  other nodes is roughly proportional to  $k^{-\gamma}$ .
- Examples: social networks, collaboration networks, protein-protein interaction networks, ...
- Properties: short paths, robust against random failures, sensitive to targeted attacks



# Section 2: Structure and Growth of the Internet

1 Internet Concepts and Design Principles

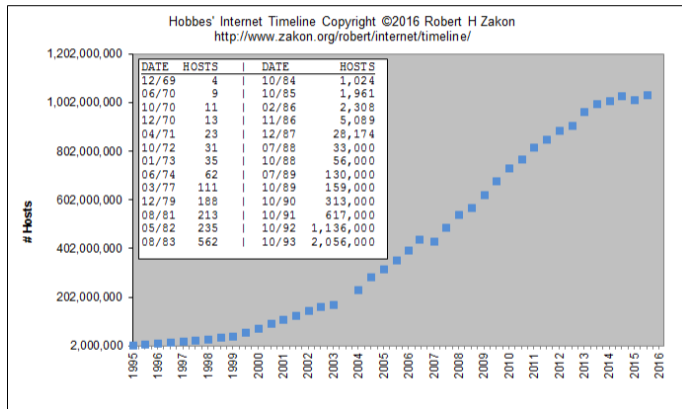
**2 Structure and Growth of the Internet**

3 Internet Programming with Sockets

# Evolution of Internet Communication

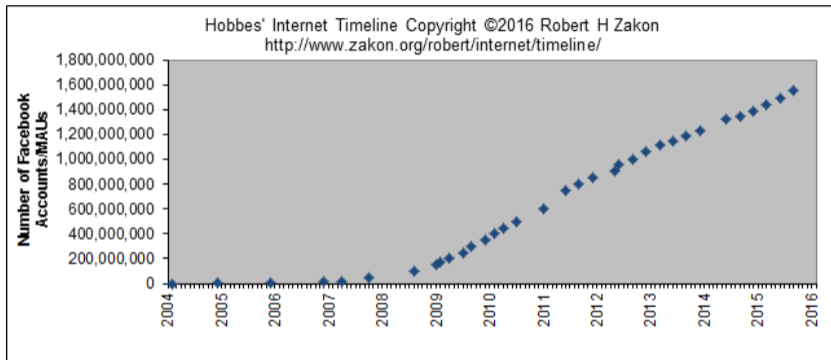
- 1970: private communication (email)
- 1980: discussion forums (usenet)
- 1985: software development and standardization (GNU)
- 1995: blogs, art, games, trading, searching (ebay, amazon)
- 1998: multimedia communication (rtp, sip, netflix)
- 2000: books and encyclopedia (wikipedia)
- 2005: social networks, video sharing (facebook, youtube)
- 2010: cloud computing, content delivery networks (akamai, amazon)
- 2015: ???

# Growth of the Internet



- How would you count the number of hosts on the Internet?

# Growth of Facebook

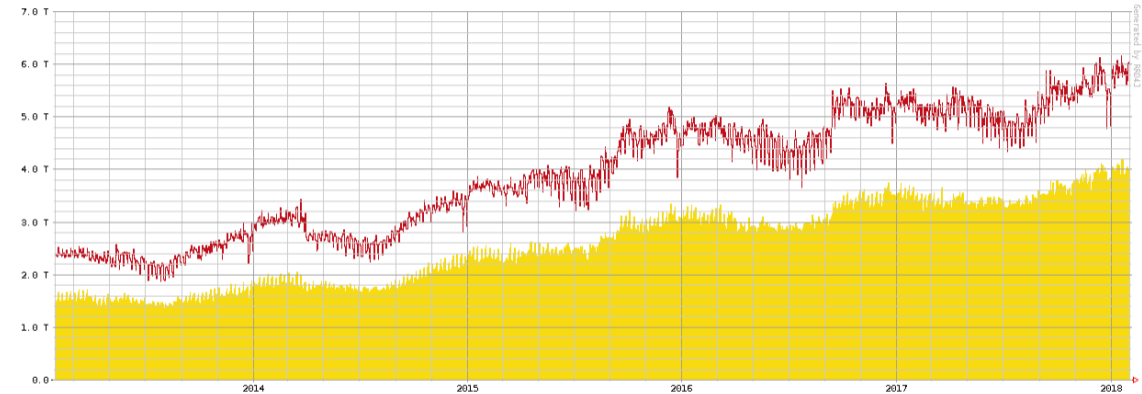


- This is pretty easy to count. . .
- But how many accounts are active, how many are fake accounts?

# Internet Exchange Points (Spring 2018)

- Internet Exchange Frankfurt/Germany (DE-CIX)
  - Connected networks:  $\approx 800$
  - Average throughput (1 year):  $\approx 4.0\text{ Tbps}$
  - Maximum throughput (1 year):  $\approx 6.0\text{ Tbps}$
  - Maximum transport capacity:  $\approx 8\text{ Tbps}$
  - Total optical backbone capacity:  $\approx 48\text{ Tbps}$
  - $3 \times 160\ 100$  Gigabit-Ethernet ports
  - <http://www.decix.de/>
- Amsterdam Internet Exchange (AMS-IX)
  - <http://www.ams-ix.net/>
- London Internet Exchange (LINX)
  - <https://www.linx.net/>

# DE-CIX Traffic (5 Years)



■ average traffic in bits per second  
■ peak traffic in bits per second  
Current 3893.4 G  
Averaged 2547.4 G  
Graph Peak 6160.8 G  
DE-CIX All-Time Peak 6160.78  
Created at 2018-02-01 00:22 UTC  
Copyright 2018 DE-CIX Management GmbH

# Networking Challenges

- Switching efficiency and energy efficiency
- Routing and fast convergence
- Security, trust, and key management
- Network measurements and network operations
- Ad-hoc networks and self-organizing networks
- Wireless sensor networks and the Internet of Things
- Delay and disruption tolerant networks
- High bandwidth and long delay networks
- Home networks and data center networks
- ...

# Section 3: Internet Programming with Sockets

- 1 Internet Concepts and Design Principles
- 2 Structure and Growth of the Internet
- 3 Internet Programming with Sockets**



# Internet Programming with Sockets

- Sockets are abstract communication endpoints with a rather small number of associated function calls
- The socket API consists of
  - address formats for various network protocol families
  - functions to create, name, connect, destroy sockets
  - functions to send and receive data
  - functions to convert human readable names to addresses and vice versa
  - functions to multiplex I/O on several sockets
- Sockets are the de-facto standard communication API provided by operating systems

# Socket Types

- Stream sockets (`SOCK_STREAM`) represent bidirectional reliable communication endpoints
- Datagram sockets (`SOCK_DGRAM`) represent bidirectional unreliable communication endpoints
- Raw sockets (`SOCK_RAW`) represent endpoints which can send/receive interface layer datagrams
- Reliable delivered message sockets (`SOCK_RDM`) are datagram sockets with reliable datagram delivery
- Sequenced packet sockets (`SOCK_SEQPACKET`) are stream sockets which retain data block boundaries

# IPv4 Socket Addresses

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

typedef ... sa_family_t;
typedef ... in_port_t;

struct in_addr {
    uint8_t  s_addr[4];      /* IPv4 address */
};

struct sockaddr_in {
    uint8_t   sin_len;      /* address length (BSD) */
    sa_family_t sin_family; /* address family */
    in_port_t sin_port;    /* transport layer port */
    struct in_addr sin_addr; /* IPv4 address */
};
```

# IPv6 Socket Addresses

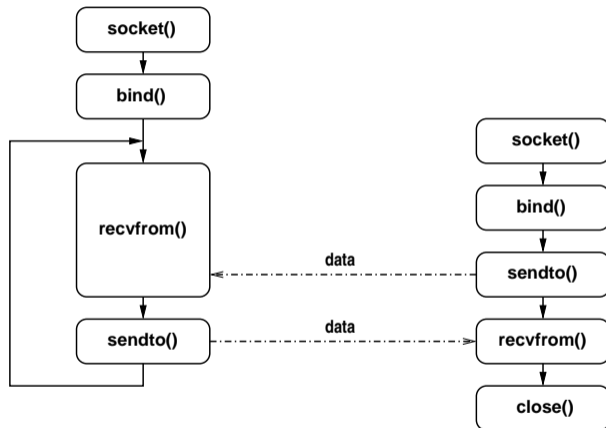
```
#include <sys/socket.h>
#include <netinet/in.h>

typedef ... sa_family_t;
typedef ... in_port_t;

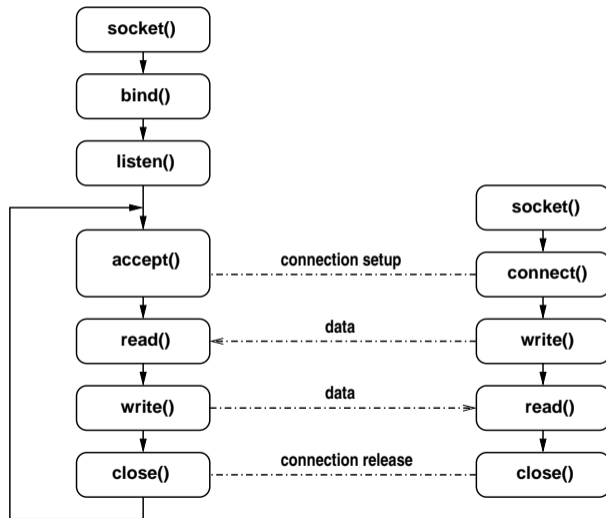
struct in6_addr {
    uint8_t  s6_addr[16];      /* IPv6 address */
};

struct sockaddr_in6 {
    uint8_t   sin6_len;        /* address length (BSD) */
    sa_family_t sin6_family;   /* address family */
    in_port_t sin6_port;      /* transport layer port */
    uint32_t  sin6_flowinfo;   /* flow information */
    struct in6_addr sin6_addr; /* IPv6 address */
    uint32_t  sin6_scope_id;   /* scope identifier */
};
```

# Connection-Less Communication



# Connection-Oriented Communication



# Socket API Summary

```
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>

#define SOCK_STREAM    ...
#define SOCK_DGRAM    ...
#define SOCK_RAW      ...
#define SOCK_RDM      ...
#define SOCK_SEQPACKET ...

#define AF_LOCAL ...
#define AF_INET ...
#define AF_INET6 ...

#define PF_LOCAL ...
#define PF_INET ...
#define PF_INET6 ...
```

# Socket API Summary

```
int socket(int domain, int type, int protocol);
int bind(int socket, struct sockaddr *addr,
         socklen_t addrlen);
int connect(int socket, struct sockaddr *addr,
           socklen_t addrlen);
int listen(int socket, int backlog);
int accept(int socket, struct sockaddr *addr,
          socklen_t *addrlen);

ssize_t write(int socket, void *buf, size_t count);
int send(int socket, void *msg, size_t len, int flags);
int sendto(int socket, void *msg, size_t len, int flags,
           struct sockaddr *addr, socklen_t addrlen);

ssize_t read(int socket, void *buf, size_t count);
int recv(int socket, void *buf, size_t len, int flags);
int recvfrom(int socket, void *buf, size_t len, int flags,
            struct sockaddr *addr, socklen_t *addrlen);
```



# Socket API Summary

```
int shutdown(int socket, int how);
int close(int socket);

int getsockopt(int socket, int level, int optname,
               void *optval, socklen_t *optlen);
int setsockopt(int socket, int level, int optname,
               void *optval, socklen_t optlen);
int getsockname(int socket, struct sockaddr *addr,
                 socklen_t *addrlen);
int getpeername(int socket, struct sockaddr *addr,
                 socklen_t *addrlen);
```

- All API functions operate on abstract socket addresses
- Not all functions make equally sense for all socket types

# Mapping Names to Addresses

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

#define AI_PASSIVE      ...
#define AI_CANONNAME   ...
#define AI_NUMERICHOST ...

struct addrinfo {
    int          ai_flags;
    int          ai_family;
    int          ai_socktype;
    int          ai_protocol;
    size_t       ai_addrlen;
    struct sockaddr *ai_addr;
    char         *ai_canonname;
    struct addrinfo *ai_next;
};
```

# Mapping Names to Addresses

```
int getaddrinfo(const char *node,  
               const char *service,  
               const struct addrinfo *hints,  
               struct addrinfo **res);  
void freeaddrinfo(struct addrinfo *res);  
const char *gai_strerror(int errcode);
```

- Many books still document the old name and address mapping functions
  - gethostbyname()
  - gethostbyaddr()
  - getservbyname()
  - getservbyaddr()

which are IPv4 specific and should not be used anymore

# Mapping Addresses to Names

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

#define NI_NOFQDN      ...
#define NI_NUMERICHOST ...
#define NI_NAMEREQD   ...
#define NI_NUMERICSERV ...
#define NI_NUMERICSCOPE ...
#define NI_DGRAM      ...

int getnameinfo(const struct sockaddr *sa,
                socklen_t salen,
                char *host, size_t hostlen,
                char *serv, size_t servlen,
                int flags);
const char *gai_strerror(int errcode);
```

# Multiplexing

```
#include <sys/select.h>


typedef ... fd_set;

FD_ZERO(fd_set *set);
FD_SET(int fd, fd_set *set);
FD_CLR(int fd, fd_set *set);
FD_ISSET(int fd, fd_set *set);

int select(int n, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
int pselect(int n, fd_set *readfds, fd_set *writefds,
            fd_set *exceptfds, struct timespec *timeout,
            sigset_t sigmask);
```

- `select()` works with arbitrary file descriptors
- `select()` frequently used to implement the main loop of event-driven programs

# References

-  B. Carpenter.  
Architectural Principles of the Internet.  
RFC 1958, IAB, June 1996.
-  B. Carpenter.  
Internet Transparency.  
RFC 2775, IBM, February 2000.
-  R. Gilligan, S. Thomson, J. Bound, J. McCann, and W. Stevens.  
Basic Socket Interface Extensions for IPv6.  
RFC 3493, Intransa Inc., Cisco, Hewlett-Packard, February 2003.
-  R. Atkinson and S. Floyd.  
IAB Concerns and Recommendations Regarding Internet Research and Evolution.  
RFC 3869, Internet Architecture Board, August 2004.
-  S. Kawamura and M. Kawashima.  
A Recommendation for IPv6 Address Text Representation.  
RFC 5952, NEC BIGLOBE, Ltd., NEC AccessTechnica, Ltd., August 2010.

# Part 2: Fundamental Concepts

- 4 Classification and Terminology
- 5 Communication Channels and Transmission Media
- 6 Media Access Control
- 7 Transmission Error Detection
- 8 Sequence Numbers, Acknowledgements, Timer
- 9 Flow Control and Congestion Control
- 10 Layering and the OSI Reference Model

# Section 4: Classification and Terminology

- 4 Classification and Terminology
- 5 Communication Channels and Transmission Media
- 6 Media Access Control
- 7 Transmission Error Detection
- 8 Sequence Numbers, Acknowledgements, Timer
- 9 Flow Control and Congestion Control
- 10 Layering and the OSI Reference Model



# Network Classifications

- Distance
  - Local area network, wide area network, personal area network, ...
- Topology
  - Star, ring, bus, line, tree, mesh, ...
- Transmission media
  - Wireless network, optical network, ...
- Purpose
  - Industrial control network, media distribution network, cloud network, access network, aggregation network, backbone network, vehicular network, ...
- Ownership
  - Home networks, national research networks, enterprise networks, government networks, community networks, ...

# Communication Modes

- Unicast — Single sender and a single receiver (1:1)
- Multicast — Single sender and multiple receivers (1:n)
- Concast — Multiple senders and a single receiver (m:1)
- Multipeer — Multiple senders and multiple receivers (m:n)
  
- Anycast — Single sender and nearest receiver out of a group of receivers
- Broadcast — Single sender and all receivers attached to a network
- Geocast — Single sender and multiple receivers in a certain geographic area

## Definition (communication protocol)

A *communication protocol* is a set of rules and message formats that govern the communication between communicating peers. A protocol defines

- the set of valid messages (syntax of messages) and
  - the meaning of each message (semantics of messages).
- 
- A protocol is necessary for any function that requires cooperation between communicating peers
  - A protocol implements ideally a well-defined service
  - It is often desirable to layer new protocols on already existing protocols in order reuse existing services

# Circuit vs. Packet Switching

- Circuit switching:
  - Communication starts by creating a (virtual) circuit between sender and receiver
  - Data is forwarded along the (virtual) circuit
  - Communication ends by removing the (virtual) circuit
  - Example: Traditional telecommunication networks
- Packet switching:
  - Data is carried in packets
  - Every packet carries information identifying the destination
  - Every packet is routed independently of other packets to its destination
  - Example: Internet

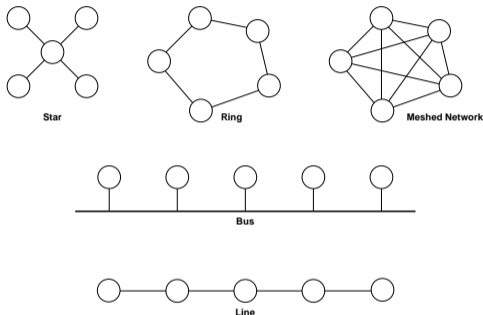
# Connection-oriented vs. Connection-less Services

- Connection-oriented:
  - Usage of a service starts by creating a connection
  - Data is exchanged within the context of a connection
  - Service usage ends by terminating the connection
  - State may be associated with connections (stateful)
  - Example: Fetching a web page on the Internet
- Connection-less:
  - Service can be used immediately
  - Usually no state maintained (stateless)
  - Example: Internet name lookups

# Data vs. Control vs. Management Plane

- Data Plane:
  - Concerned with the forwarding of data
  - Acting in the resolution of milliseconds to microseconds
  - Often implemented in hardware to achieve high data rates
- Control Plane:
  - Concerned with telling the data plane how to forward data
  - Acting in the resolution of seconds or sub-seconds
  - Traditionally implemented as part of routers and switches
  - Recent move to separate the control plane from the data plane
- Management Plane:
  - Concerned with the configuration and monitoring of data and control planes
  - Acting in the resolution of minutes or even much slower
  - May involve humans in decision and control processes

# Topologies (1/3)



- The *topology* of a network describes the way in which nodes attached to the network are interconnected

# Topologies (2/3)

- Star:
  - Nodes are directly connected to a central node
  - Simple routing
  - Good availability if central node is highly reliable
- Ring:
  - Nodes are connected to form a ring
  - Simple routing
  - Limited reliability, failures require reconfiguration
- Meshed Network:
  - Nodes are directly connected to all other nodes
  - Very good reliability
  - Requires  $\frac{n(n-1)}{2}$  links for  $n$  nodes



# Topologies (3/3)

- Bus:
  - Nodes are attached to a shared medium
  - Simple routing
  - Good availability if the bus is highly reliable
- Line:
  - Nodes are connected to form a line
  - Average reliability
  - Network position influences transmission delays
- Line topologies are especially important in the case  $n = 2$  (point to point links)

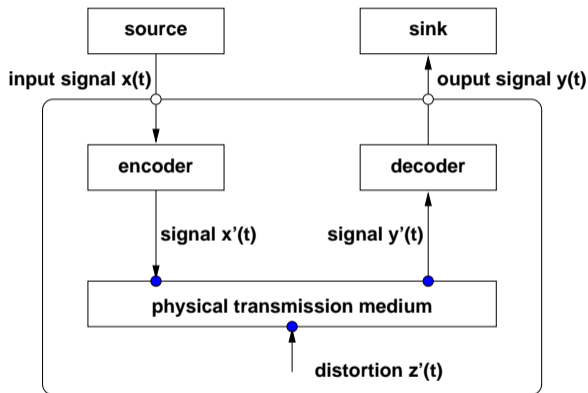
# Structured Cabling

- Networks in office buildings are typically hierarchically structured:
  - Every floor has a (potentially complex) network segment
  - The floor network segments are connected by a backbone network
  - Multiple buildings are interconnected by connecting the backbone networks of the buildings
- Cabling infrastructure in the buildings should be usable for multiple purposes (telephone network, data communication network)
- Typical lifetimes:
  - Network rooms and cable ways (20-40 years)
  - Fibre wires (about 15 years)
  - Copper wires (about 8 years)
  - Cabling should survive 3 generations of active network components

# Section 5: Communication Channels and Transmission Media

- 4 Classification and Terminology
- 5 Communication Channels and Transmission Media**
- 6 Media Access Control
- 7 Transmission Error Detection
- 8 Sequence Numbers, Acknowledgements, Timer
- 9 Flow Control and Congestion Control
- 10 Layering and the OSI Reference Model

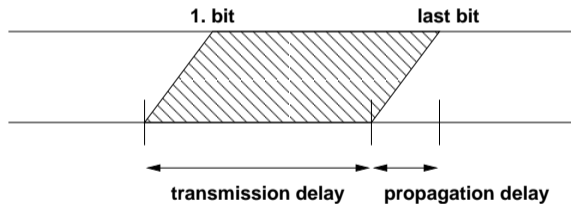
# Communication Channel Model



- Signals are in general modified during transmission, leading to transmission errors.

# Channel Characteristics

- The *data rate* (bit rate) describes the data volume that can be transmitted per time interval (e.g., 100 Mbit/s)
- The *bit time* is the time needed to transmit a single bit (e.g., 1 microsecond for 1 Mbit/s)



- The *delay* is the time needed to transmit a message from the source to the sink. It consists of the *propagation delay* and the *transmission delay*
- The *bit error rate* is the probability of a bit being changed during transmission

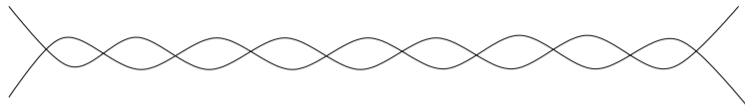
# Transmission Media Overview

- Copper wires:
  - Simple wires
  - Twisted pair
  - Coaxial cables
- Optical wires:
  - Fibre (multimode and single-mode)
- Air:
  - Radio waves
  - Micro waves
  - Infrared waves
  - Light waves

# Simple Electrical Wires

- Simple two-wire open lines are the simplest transmission medium
- Adequate for connecting equipment up to 50 m apart using moderate bit rates
- The signal is typically a voltage or current level relative to some ground level
- Simple wires can easily experience crosstalk caused by capacitive coupling
- The open structure makes wires susceptible to pick-up noise signals from other electrical signal sources

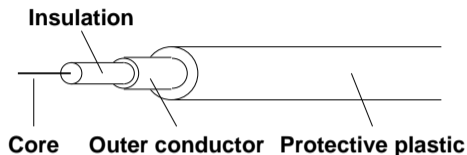
# Twisted Pairs



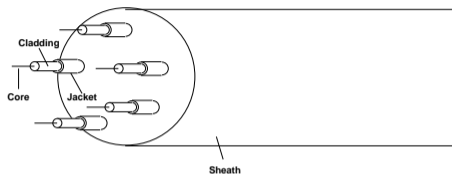
- A twisted pair consists of two insulated copper wires
- Twisting the wires in a helical form cancels out waves
- Unshielded twisted pair (UTP) of category 3 was the standard cabling up to 1988
- UTP category 5 and above are now widely used for wiring (less crosstalk, better signals over longer distances)
- Shielded twisted pair (STP) cables have an additional shield further reducing noise



# Coaxial Cable

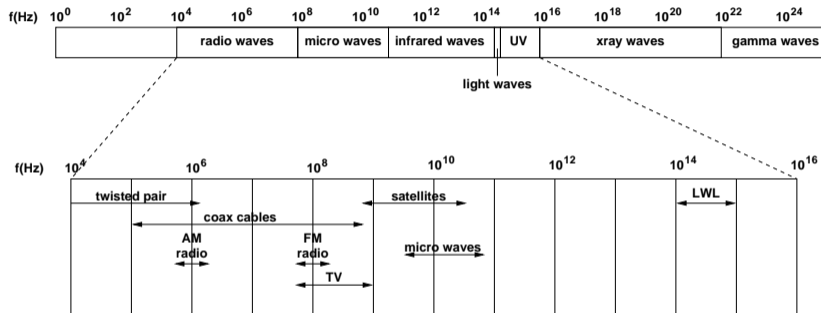


- Coax cables are shielded (less noise) and suffer less from attenuation
- Data rates of 500 mbps over several kilometers with a error probability of  $10^{-7}$  achievable
- Widely used for cable television broadcast networks (which in some countries are heavily used for data communication today)



- The glass core is surrounded by a glass cladding with a lower index of refraction to keep the light in the core
- Multimode fiber have a thick core (20-50 micrometer) and propagate light using continued refraction
- Single-mode fiber have a thin core (2-10 micrometer) which guides the light through the fiber
- High data rates, low error probability, thin, lightweight, immune to electromagnetic interference

# Electromagnetic Spectrum



- Usage of most frequencies is controlled by legislation
- The Industrial/Scientific/Medical (ISM) band (2400-2484 MHz) can be used without special licenses

# Transmission Impairments (1/2)

- *Attenuation:*
  - The strength of a signal falls off with distance over any transmission medium
  - For guided media, attenuation is generally an exponential function of the distance
  - For unguided media, attenuation is a more complex function of distance and the makeup of the atmosphere
- *Delay distortion:*
  - Delay distortion occurs because the velocity of propagation of a signal through a guided medium varies with frequency
  - Various frequency components of a signal will arrive at the receiver at different times

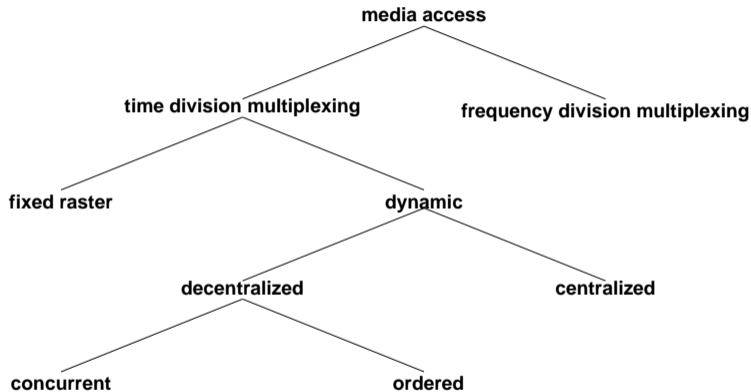
# Transmission Impairments (2/2)

- Noise
  - Thermal noise (white noise) is due to thermal agitation of electrons and is a function of temperature
  - Intermodulation noise can occur if signals at different frequencies share the same transmission medium
  - Crosstalk is an unwanted coupling between signal paths
  - Impulse noise consists of irregular pulses or noise spikes of short duration and of relatively high amplitude

# Section 6: Media Access Control

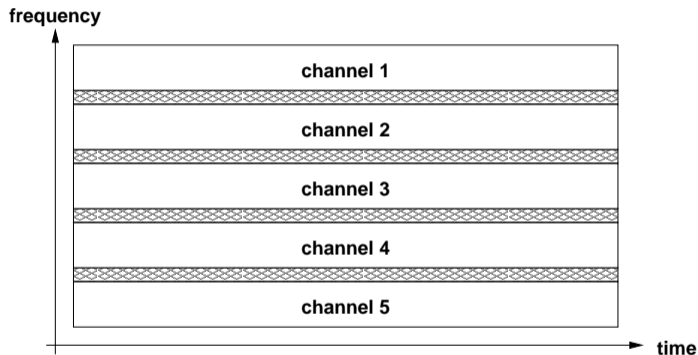
- 4 Classification and Terminology
- 5 Communication Channels and Transmission Media
- 6 Media Access Control**
- 7 Transmission Error Detection
- 8 Sequence Numbers, Acknowledgements, Timer
- 9 Flow Control and Congestion Control
- 10 Layering and the OSI Reference Model

# Media Access Control Overview



- Shared transmission media require coordinated access to the medium (media access control)

# Frequency Division Multiplexing (FDM)



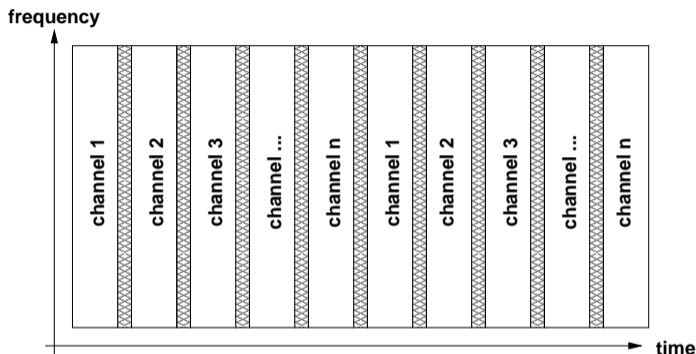
- Signals are carried simultaneously on the same medium by allocating to each signal a different frequency band



# Wavelength Division Multiplexing (WDM)

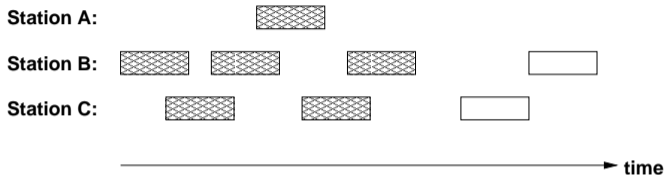
- Optical fibers carry multiple wavelength at the same time
- WDM can achieve very high data rates over a single optical fiber
- Dense WDM (DWDM) is a variation where the wavelengths are spaced close together, which results in an even larger number of channels.
- Theoretically, there is room for 1250 channels, each running at 10 Gbps, on a single fiber (= 12.5 Tbps).
- A single cable often bundles a number of fibers and for deployment or reasons, fibres are sometimes even bundled with power cables.

# Time Division Multiplexing (TDM)



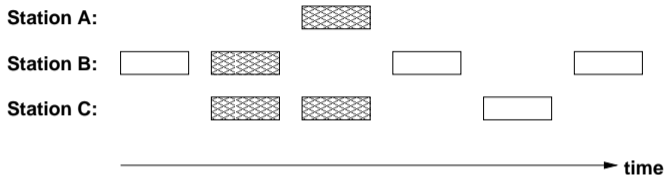
- Signals from a given sources are assigned to specific time slots
- Time slot assignment might be fixed (synchronous TDM) or dynamic (statistical TDM)

# Pure Aloha



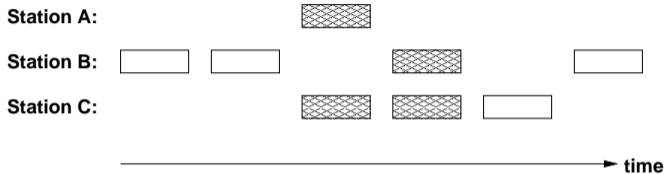
- Developed in the 1970s at the University of Hawaii
- Sender sends data as soon as data becomes available
- Collisions are detected by listening to the signal
- Retransmit after a random pause after a collision
- Not very efficient ( $\approx 18\%$  of the channel capacity)

# Slotted Aloha



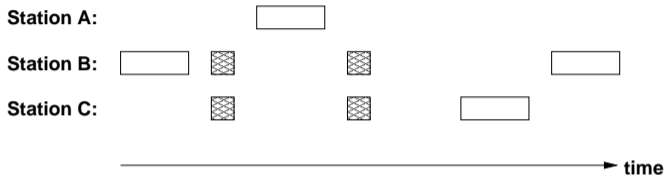
- Senders do not send immediately but wait for the beginning of a time slot
- Time slots may be advertised by short control signals
- Collisions only happen at the start of a transmission
- Avoids sequences of partially overlaying data blocks
- Slightly more efficient ( $\approx 37\%$  of the channel capacity)

# Carrier Sense Multiple Access (CSMA)



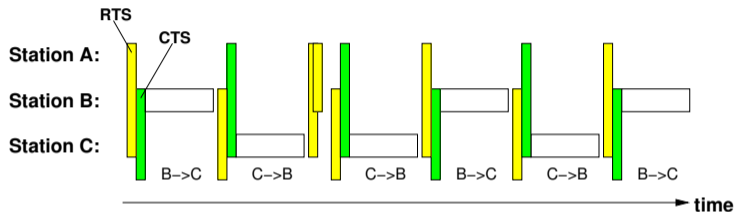
- Sense the media whether it is unused before starting a transmission
- Collisions are still possible (but less likely)
- 1-persistent CSMA: sender sends with probability 1
- p-persistent CSMA: sender sends with probability  $p$
- non-persistent CSMA: sender waits for a random time period before it retries if the media is busy

# CSMA with Collision Detection (CSMA-CD)



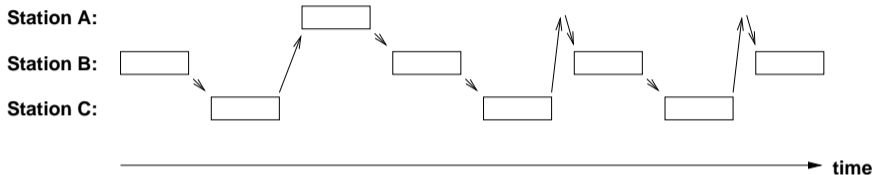
- Terminate the transmission as soon as a collision has been detected (and retry after some random delay)
- Let  $\tau$  be the propagation delay between two stations with maximum distance
- Senders can be sure that they successfully acquired the medium after  $2\tau$  time units
- Used by the classic Ethernet developed at Xerox Parc

# Multiple Access with Collision Avoidance (MACA)



- A station which is ready to send first sends a short RTS (ready to send) message to the receiver
- The receiver responds with a short CTS (clear to send) message
- Stations who receive RTS or CTS must stay quiet
- Solves the *hidden station* and *exposed station* problem

# Token Passing



- A token is a special bit pattern circulating between stations - only the station holding the token is allowed to send data
- Token mechanisms naturally match physical ring topologies - logical rings may be created on other physical topologies
- Care must be taken to handle lost or duplicate token



# Section 7: Transmission Error Detection

- 4 Classification and Terminology
- 5 Communication Channels and Transmission Media
- 6 Media Access Control
- 7 Transmission Error Detection**
- 8 Sequence Numbers, Acknowledgements, Timer
- 9 Flow Control and Congestion Control
- 10 Layering and the OSI Reference Model

# Transmission Error Detection

- Data transmission often leads to transmission errors that affect one or more bits
- Simple parity bits can be added to code words to detect bit errors
- Parity bit schemes are not very strong in detecting errors which affect multiple bits
- Computation of error check codes must be efficient (in hardware and/or software)

# Cyclic Redundancy Check (CRC)

- A bit sequence (bit block)  $b_n b_{n-1} \dots b_1 b_0$  is represented as a polynomial  $B(x) = b_n x^n + b_{n-1} x^{n-1} + \dots + b_1 x + b_0$
- Arithmetic operations:

$$0 + 0 = 1 + 1 = 0 \quad 1 + 0 = 0 + 1 = 1$$

$$1 \cdot 1 = 1 \quad 0 \cdot 0 = 0 \cdot 1 = 1 \cdot 0 = 0$$

- A generator polynomial  $G(x) = g_r x^r + \dots g_1 x + g_0$  with  $g_r = 1$  and  $g_0 = 1$  is agreed upon between the sender and the receiver
- The sender transmits  $U(x) = x^r \cdot B(x) + t(x)$  with

$$t(x) = (x^r \cdot B(x)) \bmod G(x)$$

# Cyclic Redundancy Check (CRC)

- The receiver tests whether the polynomial corresponding to the received bit sequence can be divided by  $G(x)$  without a remainder
- Efficient hardware implementation possible using XOR gates and shift registers
- Only errors divisible by  $G(x)$  will go undetected
- Example:
  - Generator polynomial  $G(x) = x^3 + x^2 + 1$   
(corresponds to the bit sequence 1101)
  - Message  $M = 1001\ 1010$   
(corresponds to the polynomial  $B(x) = x^7 + x^4 + x^3 + x$ )

# CRC Computation

1001 1010 000 : 1101

1101

----

100 1

110 1

-----

10 00

11 01

-----

1 011

1 101

-----

1100

1101

-----

1 000

1 101

-----

101

=> transmitted bit sequence 1001 1010 101

# CRC Verification

1001 1010 101 : 1101

1101

----

100 1

110 1

-----

10 00

11 01

-----

1 011

1 101

-----

1100

1101

-----

1 101

1 101

-----

0    =>    remainder 0, assume no transmission error

# Choosing Generator Polynomials

- $G(x)$  detects all single-bit errors if  $G(x)$  has more than one non-zero term
- $G(x)$  detects all double-bit errors, as long as  $G(x)$  has a factor with three terms
- $G(x)$  detects any odd number of errors, as long as  $G(x)$  contains the factor  $(x + 1)$
- $G(x)$  detects any burst errors for which the length of the burst is less than or equal to  $r$
- $G(x)$  detects a fraction of error bursts of length  $r + 1$ ; the fraction equals to  $1 - 2^{-(r-1)}$
- $G(x)$  detects a fraction of error bursts of length greater than  $r + 1$ ; the fraction equals to  $1 - 2^{-r}$

# Well-known Generator Polynomials

- The HEC polynomial  $G(x) = x^8 + x^2 + x + 1$  is used by the ATM cell header
- The CRC-16 polynomial  $G(x) = x^{16} + x^{15} + x^2 + 1$  detects all single and double bit errors, all errors with an odd number of bits, all burst errors with 16 or less bits and more than 99% of all burst errors with 17 or more bits
- The CRC-CCITT polynomial  $G(x) = x^{16} + x^{15} + x^5 + 1$  is used by the HDLC protocol
- The CRC-32 polynomial  $G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$  is used by the IEEE 802 standards



# Internet Checksum

```
uint16_t
checksum(uint16_t *buf, int count)
{
    uint32_t sum = 0;
    while (count-- > 0) {
        sum += *buf++;
        if (sum > 0xffff) {
            sum &= 0xffff;
            sum++;
        }
    }
    return ~((sum & 0xffff) >> 1);
}
```

# Internet Checksum Computation

```
data[] = dead cafe face (hexadecimal)
```

```

    0000
+   dead (data[0])
-----
    dead
+   cafe (data[1])
-----
   1a9ab
+   '--->1
-----
    a9ac
+   face (data[2])
-----
   1a47a
+   '--->1
-----
    a47b   complement
-----> 5b84 (checksum)

verification:
    0000
+ '   dead (data[0])
-----
    dead
+ '   cafe (data[1])
-----
    a9ac
+ '   face (data[2])
-----
    a47b
+ '   5b84 (checksum)
-----
    ffff (test passed)
```

# Internet Checksum Properties

- Summation is commutative and associative
- Computation independent of the byte order
- Computation can be parallelized on processors with word sizes larger than 16 bit
- Individual data fields can be modified without having to recompute the whole checksum
- Can be integrated into copy loop
- Often implemented in assembler or special hardware
- For details, see RFC 1071, RFC 1141, and RFC 1624

# Section 8: Sequence Numbers, Acknowledgements, Timer

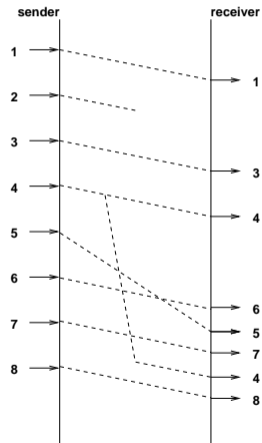
- 4 Classification and Terminology
- 5 Communication Channels and Transmission Media
- 6 Media Access Control
- 7 Transmission Error Detection
- 8 Sequence Numbers, Acknowledgements, Timer**
- 9 Flow Control and Congestion Control
- 10 Layering and the OSI Reference Model

# Errors Affecting Complete Data Frames

- Despite bit errors, the following transmission errors can occur
  - Loss of complete data frames
  - Duplication of complete data frames
  - Receipt of data frames that were never sent
  - Reordering of data frames during transmission
- In addition, the sender must adapt its speed to the speed of the receiver (*end-to-end flow control*)
- Finally, the sender must react to congestion situations in the network (*congestion control*)

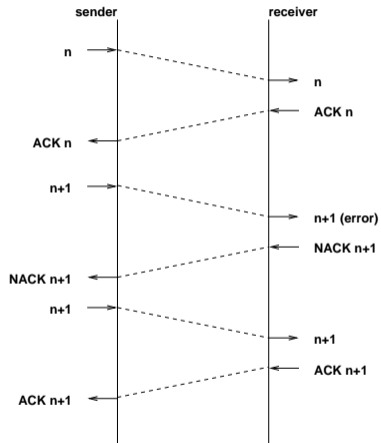
# Sequence Numbers

- The sender assigns growing sequence numbers to all data frames
- A receiver can detect reordered or duplicated frames
- Loss of a frame can be determined if a missing frame cannot travel in the network anymore
- Sequence numbers can grow quickly on fast networks



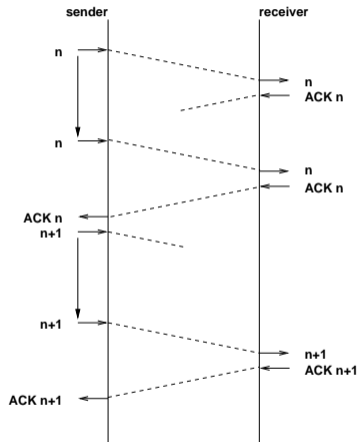
# Acknowledgements

- Retransmit to handle errors
- A positive acknowledgement (ACK) is sent to inform the sender that the transmission of a frame was successful
- A negative acknowledgement (NACK) is sent to inform the sender that the transmission of a frame was unsuccessful
- Stop-and-wait protocol: a frame is only transmitted if the previous frame was been acknowledged



# Timers

- Timer can be used to detect the loss of frames or acknowledgments
- A sender can use a timer to retransmit a frame if no acknowledgment has been received in time
- A receiver can use a timer to retransmit acknowledgments
- Problem: Timers must adapt to the current delay in the network



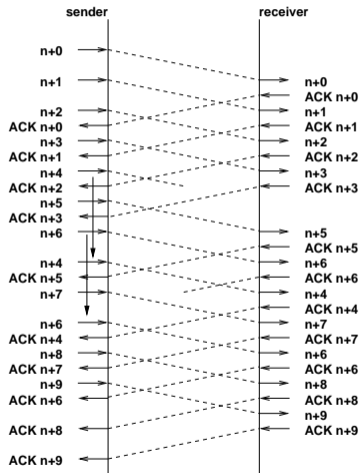


# Section 9: Flow Control and Congestion Control

- 4 Classification and Terminology
- 5 Communication Channels and Transmission Media
- 6 Media Access Control
- 7 Transmission Error Detection
- 8 Sequence Numbers, Acknowledgements, Timer
- 9 Flow Control and Congestion Control**
- 10 Layering and the OSI Reference Model

# Flow Control

- Allow the sender to send multiple frames before waiting for acknowledgments
- Improves efficiency and overall delay
- Sender must not overflow the receiver
- The stream of frames should be smooth and not bursty
- Speed of the receiver can vary over time



# Sliding Window Flow Control

- Sender and receiver agree on a window of the sequence number space
- The sender may only transmit frames whose sequence number falls into the sender's window
- Upon receipt of an acknowledgement, the sender's window is moved
- The receiver only accepts frames whose sequence numbers fall into the receiver's window
- Frames with increasing sequence number are delivered and the receiver window is moved.
- The size of the window controls the speed of the sender and must match the buffer capacity of the receiver

# Sliding Window Implementation

- Implementation on the sender side:
  - SWS (send window size)
  - LAR (last ack received)
  - LFS (last frame send)
  - Invariant:  $LFS - LAR + 1 \leq SWS$
- Implementation on the receiver side:
  - RWS (receiver window size)
  - LFA (last frame acceptable)
  - NFE (next frame expected)
  - Invariant:  $LFA - NFE + 1 \leq RWS$

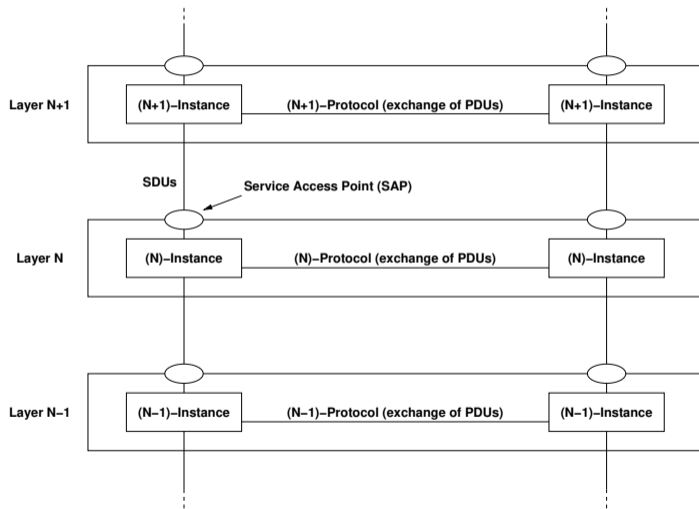
# Congestion Control

- Flow control is used to adapt the speed of the sender to the speed of the receiver
- Congestion control is used to adapt the speed of the sender to the speed of the network
- Principles:
  - Sender and receiver reserve bandwidth and puffer capacity in the network
  - Intermediate systems drop frames under congestion and signal the event to the senders involved
  - Intermediate systems send control messages (choke packets) when congestion builds up to slow down senders

# Section 10: Layering and the OSI Reference Model

- 4 Classification and Terminology
- 5 Communication Channels and Transmission Media
- 6 Media Access Control
- 7 Transmission Error Detection
- 8 Sequence Numbers, Acknowledgements, Timer
- 9 Flow Control and Congestion Control
- 10 Layering and the OSI Reference Model**

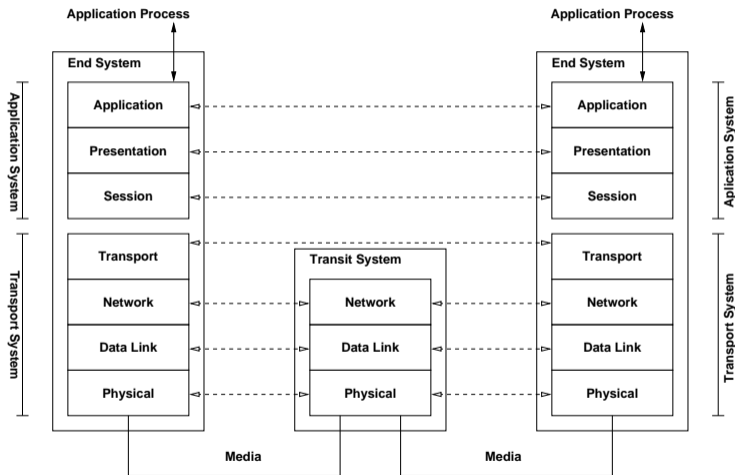
# Layering Overview



- Principles:
  - A layer provides a well defined service
  - A layer (service) is accessed via a service access point (SAP)
  - Multiple different protocols may implement the same service
  - Protocol data units (PDUs) are exchanged between peer entities
  - Service data units (SDUs) are exchanged between layers (services)
  - Every service access point (SAP) needs an addressing mechanism
- Advantages:
  - Information hiding and reuse
  - Independent evolution of layers
- Disadvantages:
  - Layering hinders certain performance optimizations
  - Tension between information-hiding (abstraction) and performance



# OSI Reference Model Overview



# Physical and Data Link Layer

- Physical Layer:
  - Transmission of an unstructured bit stream
  - Standards for cables, connectors and sockets
  - Encoding of binary values (voltages, frequencies)
  - Synchronization between sender and receiver
- Data Link Layer:
  - Transmission of bit sequences in so called frames
  - Data transfer between directly connected systems
  - Detection and correction of transmission errors
  - Flow control between senders and receivers
  - Realization usually in hardware




# Network and Transport Layer

- Network Layer:
  - Determination of paths through a complex network
  - Multiplexing of end-to-end connections over an intermediate systems
  - Error detection / correction between network nodes
  - Flow and congestion control between end systems
  - Transmission of datagrams or packets in packet switched networks
- Transport Layer:
  - Reliable end-to-end communication channels
  - Connection-oriented and connection-less services
  - End-to-end error detection and correction
  - End-to-end flow and congestion control

# Session, Presentation and Application Layer

- Session Layer:
  - Synchronization and coordination of communicating processes
  - Interaction control (check points)
- Presentation Layer:
  - Harmonization of different data representations
  - Serialization of complex data structures
  - Data compression
- Application Layer:
  - Fundamental application oriented services
  - Terminal emulation, name and directory services, data base access, network management, electronic messaging systems, process and machine control, ...

# References

-  J. F. Kurose and K. W. Ross.  
*Computer Networking: A Top-Down Approach Featuring the Internet.*  
Addison-Wesley, 3 edition, 2004.
-  A. S. Tanenbaum.  
*Computer Networks.*  
Prentice Hall, 4 edition, 2002.
-  J. Stone and C. Partridge.  
When The CRC and TCP Checksum Disagree.  
In *Proc. SIGCOMM 2000*, pages 309–319, Stockholm, August 2000. ACM.

# Part 3: Local Area Networks (IEEE 802)

- 11 Local Area Networks Overview
- 12 Ethernet (IEEE 802.3)
- 13 Bridges (IEEE 802.1)
- 14 Virtual Local Area Networks (IEEE 802.1Q)
- 15 Port Access Control (IEEE 802.1X)
- 16 Wireless LAN (IEEE 802.11)
- 17 Logical Link Control (IEEE 802.2)

# Section 11: Local Area Networks Overview

## 11 Local Area Networks Overview

12 Ethernet (IEEE 802.3)

13 Bridges (IEEE 802.1)

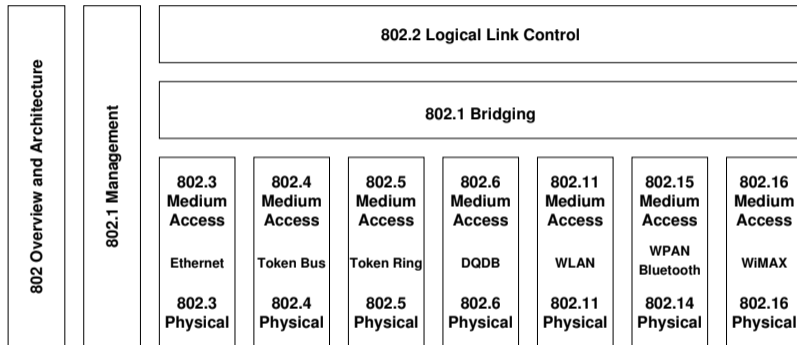
14 Virtual Local Area Networks (IEEE 802.1Q)

15 Port Access Control (IEEE 802.1X)

16 Wireless LAN (IEEE 802.11)

17 Logical Link Control (IEEE 802.2)

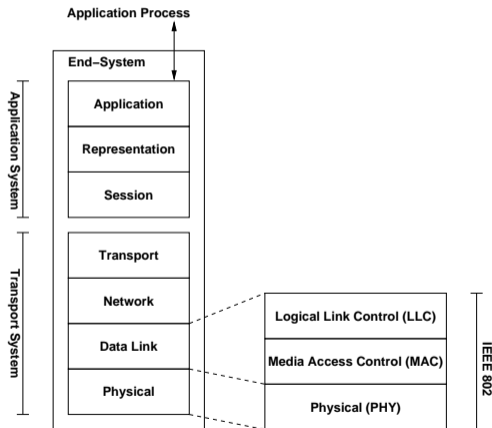
# IEEE 802 Overview



- IEEE 802 standards are developed since the early 1980s
- Dominating technology in local area networks (LANs)



# IEEE 802 Layers in the OSI Model



- The Logical Link Control layer provides a common service interface for all IEEE 802 protocols
- The Medium Access Control layer defines the method used to access the transmission media used
- The Physical layer defines the physical properties for the various transmission media that can be used with a certain IEEE 802.x protocol

- IEEE 802 addresses (sometimes called MAC addresses or meanwhile also EUI-48 addresses) are 6 octets (48 bit) long
- The common notation is a sequence of hexadecimal numbers with the bytes separated from each other using colons or hyphens (00:D0:59:5C:03:8A or 00-D0-59-5C-03-8A)
- The highest bit indicates whether it is a *unicast* address (0) or a *multicast* address (1). The second highest bit indicates whether it is a *local* (1) or a *global* (0) address
- The *broadcast* address, which represents all stations within a broadcast domain, is FF-FF-FF-FF-FF-FF
- Globally unique addresses are created by vendors who apply for a number space delegation by the IEEE

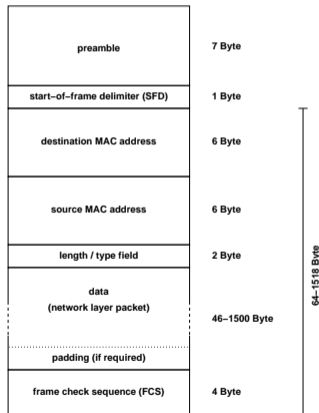
# Section 12: Ethernet (IEEE 802.3)

- 11 Local Area Networks Overview
- 12 Ethernet (IEEE 802.3)**
- 13 Bridges (IEEE 802.1)
- 14 Virtual Local Area Networks (IEEE 802.1Q)
- 15 Port Access Control (IEEE 802.1X)
- 16 Wireless LAN (IEEE 802.11)
- 17 Logical Link Control (IEEE 802.2)

Year	Achievement
1976	Original Ethernet paper published
1990	10 Mbps Ethernet over twisted pair (10BaseT)
1995	100 Mbps Ethernet
1998	1 Gbps Ethernet
2002	10 Gbps Ethernet
2010	100 Gbps Ethernet
2017	400 Gbps Ethernet (predicted)

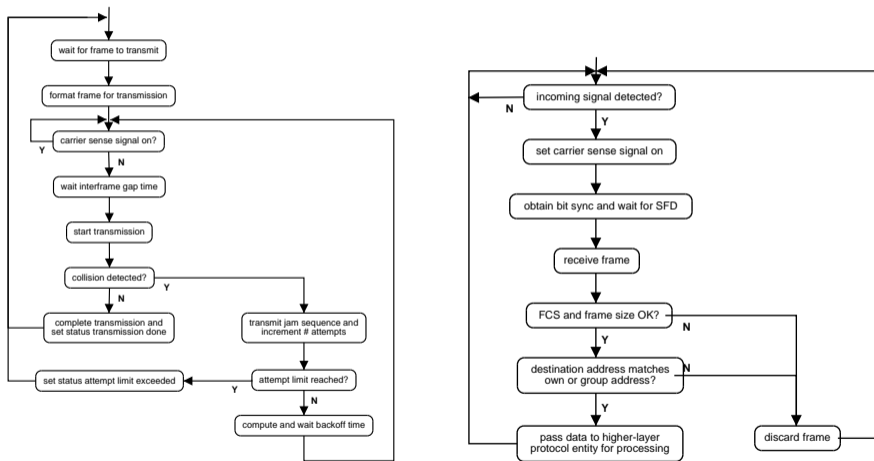
- Link aggregation allows to “bundle” links, e.g., four 10 Gbps links can be bundled to perform like a single 40 Gbps link

# IEEE 802.3 Frame Format



- Classic Ethernet used CSMA/CD and a shared bus
- Today's Ethernet uses a star topology with full duplex links
- Jumbo frames with sizes up to 9000 bytes can be used on dedicated links to improve throughput
- Interface cards capable to segment large chunks of data (e.g., 64k) into a sequence of frames (large segment offload, LSO) improve throughput on the sending side

# Transmitting and Receiving IEEE 802.3 Frames (CSMA/CD)



# Ethernet Media Types

Name	Medium	Maximum Length
10Base2	coax, $\varnothing=0.25$ in	200 m
10Base5	coax, $\varnothing=0.5$ in	500 m
10BaseT	twisted pair	100 m
10BaseF	fiber optic	2000 m
100BaseT4	twisted pair	100 m
100BaseTX	twisted pair	100 m
100BaseFX	fiber optic	412 m
1000BaseLX	fiber optic	500 / 550 / 5000 m
1000BaseSX	fiber optic	220-275 / 550 m
1000BaseCX	coax	25 m
1000BaseT	twisted pair	100 m

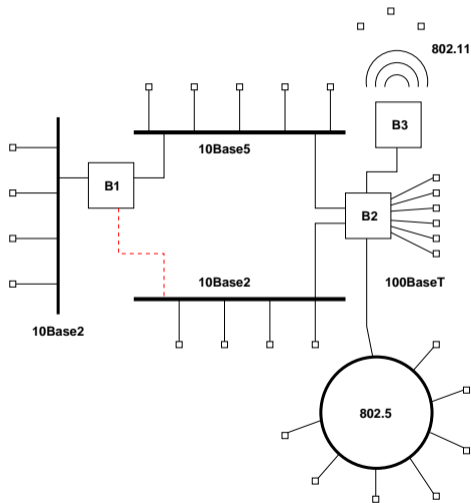
Name	Medium	Maximum Length
10GBase-SR	fiber optic	26 / 82 m
10GBase-LR	fiber optic	10 km
10GBase-ER	fiber optic	40 km
10GBase-T	twisted pair	55 / 100 m
40GBASE-KR4	backplane	1m
40GBASE-CR4	copper cable	7m
40GBASE-SR4	fiber optic	100 / 125 m
40GBASE-LR4	fiber optic	10 km
40GBASE-FR	fiber optic	2km
100GBASE-CR10	copper cable	7m
100GBASE-SR10	fiber optic	100/ 125 m
100GBASE-LR4	fiber optic	10 km
100GBASE-ER4	fiber optic	40 km

# Section 13: Bridges (IEEE 802.1)

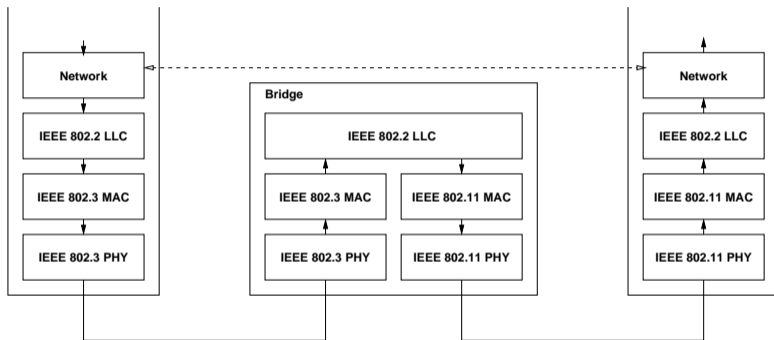
- 11 Local Area Networks Overview
- 12 Ethernet (IEEE 802.3)
- 13 Bridges (IEEE 802.1)**
- 14 Virtual Local Area Networks (IEEE 802.1Q)
- 15 Port Access Control (IEEE 802.1X)
- 16 Wireless LAN (IEEE 802.11)
- 17 Logical Link Control (IEEE 802.2)



# Bridged IEEE 802 Networks

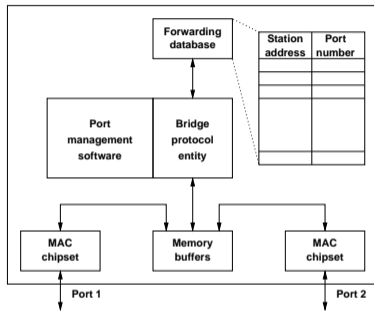


# IEEE 802 Bridges



- *Source Routing Bridges:* Sender routes the frame through the bridged network
- *Transparent Bridges:* Bridges are transparent to senders and receivers

# Transparent Bridges (IEEE 802.1D)



- Lookup an entry with a matching destination address in the forwarding database and forward the frame to the associated port.
- If no matching entry exists, forward the frame to all outgoing ports except the port from which the frame was received (flooding).

# Backward Learning and Flooding

1. The forwarding database is initially empty.
  2. Whenever a frame is received, add an entry to the forwarding database (if it does not yet exist) using the frame's source address and the incoming port number.
  3. Reinitialize the timer attached to the forwarding base entry for the received frame.
  4. Lookup the destination address in the forwarding database.
  5. If found, forward the frame to the identified port. Otherwise, send the frame to all ports (except the one from which it was received).
  6. Periodically remove entries from the forwarding table whose timer has expired.
- Aging of unused entries reduces forwarding table size and allows bridges to adapt to topology changes.
  - Backward learning and flooding requires a cycle free topology.

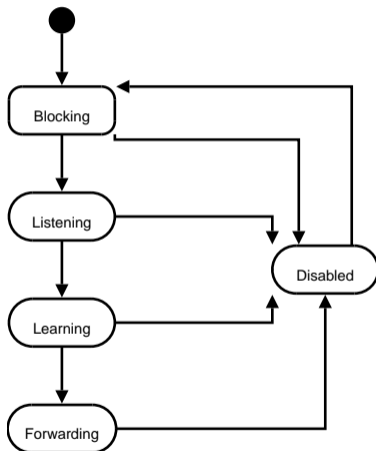
# Spanning Tree

1. The root of the spanning tree is selected (root bridge). The root bridge is the bridge with the highest priority and the smallest bridge address.
2. The costs for all possible paths from the root bridge to the various ports on the bridges is computed (root path cost). Every bridge determines which root port is used to reach the root bridge at the lowest costs.
3. The designated bridge is determined for each segment. The designated bridge of a segment is the bridge which connects the segment to the root bridge with the lowest costs on its root port. The ports used to reach designated bridges are called designated ports.
4. All ports are blocked which are not designated ports or root ports. The resulting active topology is a spanning tree.

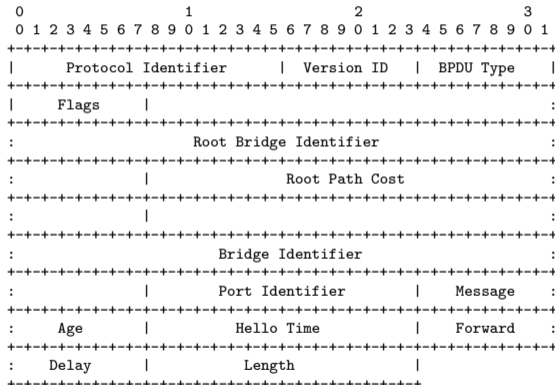
# Port States

- Blocking: A port in the blocking state does not participate in frame forwarding.
- Listening: A port in the transitional listening state has been selected by the spanning tree protocol to participate in frame forwarding.
- Learning: A port in the transitional learning state is preparing to participate in frame forwarding.
- Forwarding: A port in the forwarding state forwards frames.
- Disabled: A port in the disabled state does not participate in frame forwarding or the operation of spanning tree protocol.

# Port State Transitions



# Bridge Protocol PDUs



- BPDUs are sent periodically over all ports (including blocked ports) to a special multicast address
- BPDUs are usually encapsulated in an LLC header (see below)
- Failure to transmit or deliver BPDUs may result in bridging errors



# Broadcast Domains

- A bridged LAN defines a single *broadcast domain*:
  - All frames sent to the broadcast address are forwarded on all links in the bridged networks.
  - Broadcast traffic can take a significant portion of the available bandwidth.
  - Devices running not well-behaving applications can cause *broadcast storms*.
  - Bridges may flood frames if the MAC address cannot be found in the forwarding table.
- It is desirable to reduce the size of broadcast domains in order to separate traffic in a large bridged LAN.
- Do not confuse a broadcast domain with a collision domain, i.e., segments on which media access collisions can occur.

# Section 14: Virtual Local Area Networks (IEEE 802.1Q)

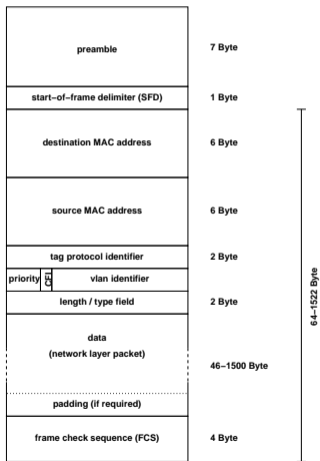
- 11 Local Area Networks Overview
- 12 Ethernet (IEEE 802.3)
- 13 Bridges (IEEE 802.1)
- 14 Virtual Local Area Networks (IEEE 802.1Q)**
- 15 Port Access Control (IEEE 802.1X)
- 16 Wireless LAN (IEEE 802.11)
- 17 Logical Link Control (IEEE 802.2)

# IEEE 802.1Q Virtual LANs

- VLANs provide a separation of logical LAN topologies from physical LAN topologies
- VLANs are identified by a VLAN identifier (1..4094)
- VLANs allow to separate the traffic on an IEEE 802 network
- A station only receives frames belonging to that VLANs it is a member of
- VLANs can reduce the network load:
  1. VLANs often cover only a certain part of the underlying physical topology
  2. Frames that are targeted to all stations (broadcasts) will only be delivered to the stations connected to the VLAN
- Stations can be a member of multiple VLANs simultaneously (important for shared servers)

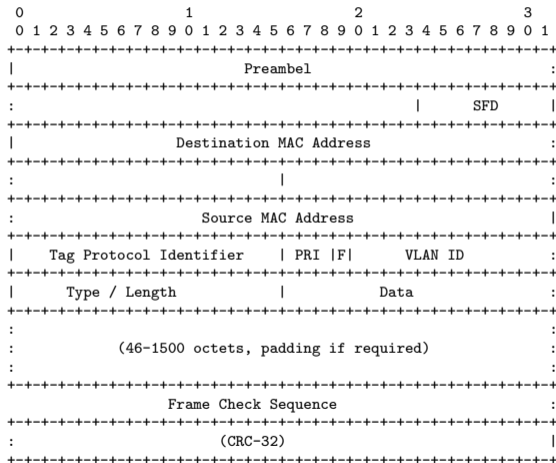
Virtual LANs are a common way to separate traffic on large local area networks. VLAN technology appeared in the 1990s.

# IEEE 802.1Q Tagged Frames



- The tag protocol identifier indicates a tagged frame
- Tagged 802.3 frames are 4 bytes longer than original 802.3 frames
- Tagged frames should only appear on links that are VLAN aware

# IEEE 802.3 Tagged Frame Format



# VLAN Membership

- Bridge ports can be assigned to VLANs in different ways:
  - Ports are administratively assigned to VLANs (port-based VLANs)
  - MAC addresses are administratively assigned to VLANs (MAC address-based VLANs)
  - Frames are assigned to VLANs based on the payload contained in the frames (protocol-based VLANs)
  - Members of a certain multicast group are assigned to VLAN (multicast group VLANs)
- The Generic Attribute Registration Protocol (GARP) can (among other things) propagate VLAN membership information.

# IEEE 802.1 Q-in-Q Tagged Frames (IEEE 802.1ad)

- With two tags, a theoretical limit of  $4096 \cdot 4096 = 16777216$  different tags can be achieved (larger tag space)
- A tag stack allows bridges to easily modify the tags since bridges can easily “push” or “pop” tags
- A tag stack creates a mechanism for ISPs to encapsulate customer single-tagged 802.1Q traffic with a single outer tag; the outer tag is used to identify traffic from different customers
- QinQ frames are convenient means of constructing Layer 2 tunnels, or applying Quality of service (QoS) policies, etc.
- 802.1ad is upward compatible with 802.1Q and although 802.1ad is limited to two tags, there is no ceiling on the standard allowing for future growth
- Double tagging is relatively easy to add to existing products



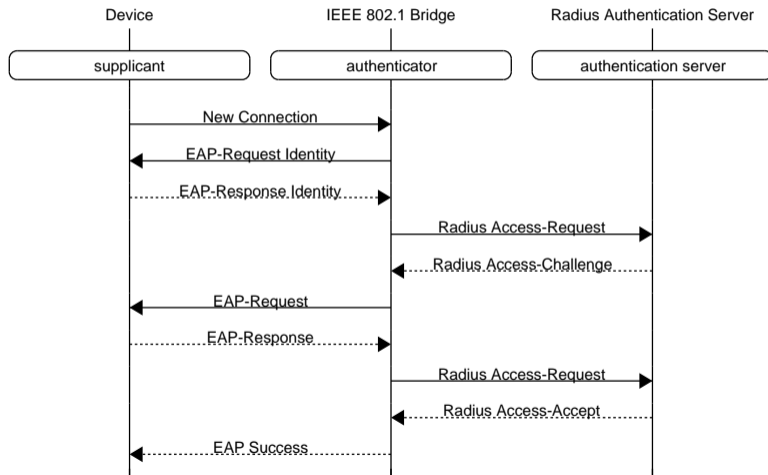
# Section 15: Port Access Control (IEEE 802.1X)

- 11 Local Area Networks Overview
- 12 Ethernet (IEEE 802.3)
- 13 Bridges (IEEE 802.1)
- 14 Virtual Local Area Networks (IEEE 802.1Q)
- 15 Port Access Control (IEEE 802.1X)**
- 16 Wireless LAN (IEEE 802.11)
- 17 Logical Link Control (IEEE 802.2)

# IEEE 802.1X Port Access Control

- Port-based network access control grants access to a switch port based on the identity of the connected machine.
- The components involved in 802.1X:
  - The *supplicant* runs on a machine connecting to a bridge and provides authentication information.
  - The *authenticator* runs on a bridge and enforces authentication decisions.
  - The *authentication server* is a (logically) centralized component which provides authentication decisions (usually via RADIUS).
- The authentication exchange uses the Extensible Authentication Protocol (EAP).
- IEEE 802.1X is becoming increasingly popular as a roaming solution for IEEE 802.11 wireless networks.

# IEEE 802.1X Sequence Diagram



# Section 16: Wireless LAN (IEEE 802.11)

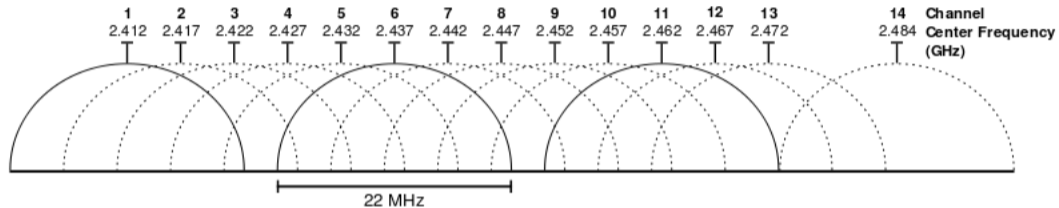
- 11 Local Area Networks Overview
- 12 Ethernet (IEEE 802.3)
- 13 Bridges (IEEE 802.1)
- 14 Virtual Local Area Networks (IEEE 802.1Q)
- 15 Port Access Control (IEEE 802.1X)
- 16 Wireless LAN (IEEE 802.11)**
- 17 Logical Link Control (IEEE 802.2)

# IEEE 802.11 Wireless LAN

Protocol	Released	Frequency	Data Rate	Indoor	Outdoor
802.11a	1999	5 GHz	54 Mbps	35 <i>m</i>	120 <i>m</i>
802.11b	1999	2.4 GHz	11 Mbps	38 <i>m</i>	140 <i>m</i>
802.11g	2003	2.4 GHz	54 Mbps	38 <i>m</i>	140 <i>m</i>
802.11n	2009	2.4/5 GHz	248 Mbps	70 <i>m</i>	250 <i>m</i>
802.11ac	2014	5 GHz	600 Mbps	70 <i>m</i>	250 <i>m</i>

- Very widely used wireless local area network (WLAN).
- As a consequence, very cheap equipment (base stations, interface cards).
- Wired equivalent privacy (WEP) was a disaster (at least for those who believe a wire is secure).
- Recommended is WPA-2 (Wifi Protected Access), in particular in combination with 802.1X and EAP-TLS.

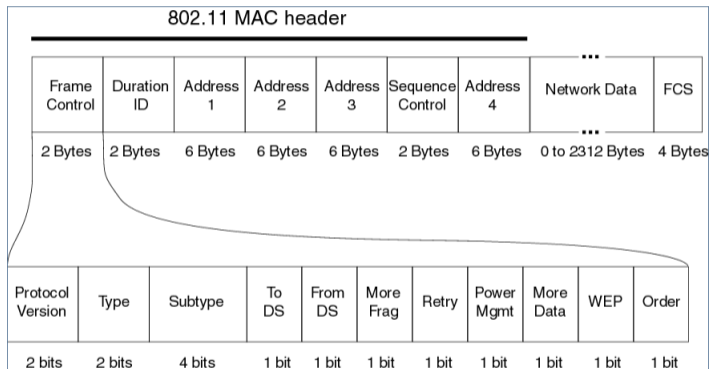
# IEEE 802.11 2.4 GHz Channels



# IEEE 802.11 Frame Types

- Data Frames: Carrying “useful” payloads
- Control Frames: Facilitate the exchange of data frames
  - Ready-to-send (RTS) and Clear-to-send (CTS) frames
  - Acknowledgement (ACK) frames
- Management Frames: Maintenance of the network
  - Beacon frames
  - Authentication / deauthentication frames
  - Association / deassociation frames
  - Probe request / response frames
  - Reassociation request / response frames

# IEEE 802.11 Frame Format

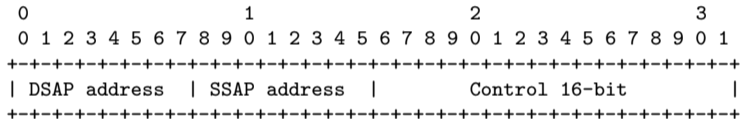
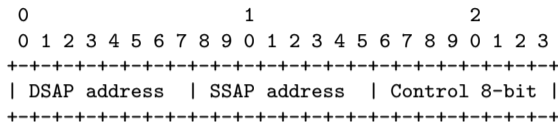




# Section 17: Logical Link Control (IEEE 802.2)

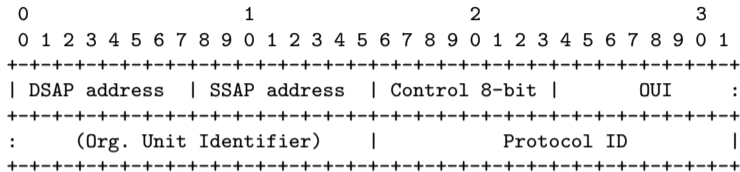
- 11 Local Area Networks Overview
- 12 Ethernet (IEEE 802.3)
- 13 Bridges (IEEE 802.1)
- 14 Virtual Local Area Networks (IEEE 802.1Q)
- 15 Port Access Control (IEEE 802.1X)
- 16 Wireless LAN (IEEE 802.11)
- 17 Logical Link Control (IEEE 802.2)**

# IEEE Logical Link Control Header



- The LLC header follows the MAC frame header (but not always used)
- DSAP = Destination Service Access Point
- SSAP = Source Service Access Point
- Control = Various control bits, may indicate subsequent header

# IEEE Logical Link Control SNAP Header (8-bit Control)



- The LLC header may be followed by the Subnetwork Access Protocol (SNAP) header (if indicated by the Control field)
- The Organizational Unit Identifier (OUI) identifies an organization defining Protocol ID values
- The Protocol ID identifies the protocol in the following payload
- If the OUI is 0x000000, the Protocol ID contains an Ethernet type value

# References



R. M. Metcalfe and D. R. Boggs.

Ethernet: Distributed packet switching for local computer networks.  
*Communications of the ACM*, 19(5):395–404, July 1976.



R. Perlman.

An Algorithm for Distributed Computation of a Spanning Tree in an Extended LAN.  
*SIGCOMM Computer Communication Review*, 15(4), September 1985.



A. F. Benner, P. K. Pepeljugoski, and R. J. Recio.

A Roadmap to 100G Ethernet at the Enterprise Data Center.  
*IEEE Applications and Practice*, 45(3), November 2007.



L. D. Nardis and M.-G. Di Benedetto.

Overview of the IEEE 802.15.4/4a standards for low data rate Wireless Personal Data Networks.  
In *Proc. of the 4th IEEE Workshop on Positioning, Navigation and Communication 2007 (WPNC'07)*, Hannover, March 2007. IEEE.

# Part 4: Internet Network Layer

18 Concepts and Terminology

19 Internet Protocol Version 6

20 Internet Protocol Version 4

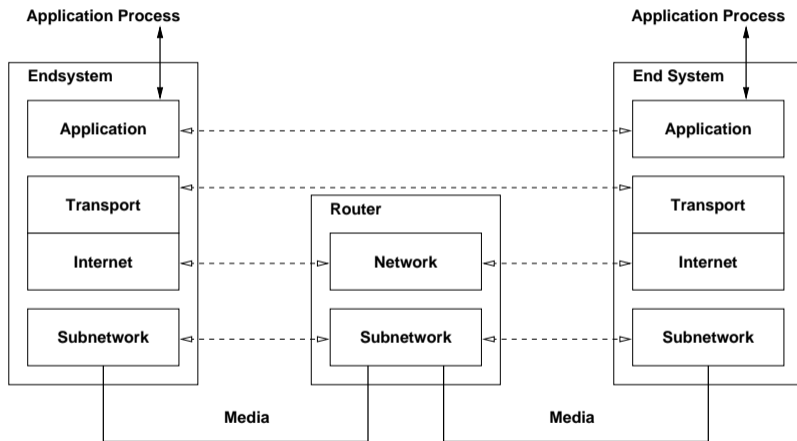
# Section 18: Concepts and Terminology

18 Concepts and Terminology

19 Internet Protocol Version 6

20 Internet Protocol Version 4

# Internet Reference Model



# Terminology (1/2)

- A *node* is a device which implements an Internet Protocol (such as IPv4 or IPv6).
- A *router* is a node that forwards IP packets not addressed to itself.
- A *host* is any node which is not a router.
- A *link* is a communication channel below the IP layer which allows nodes to communicate with each other (e.g., an Ethernet).
- The *neighbors* is the set of all nodes attached to the same link.
- An *interface* is a node's attachment to a link.
- An *IP address* identifies an interface or a set of interfaces.



## Terminology (2/2)

- An *IP prefix* is the initial part of an IP address identifying an IP network. The IP prefix is commonly defined by the number of the initial bits of an IP address that are identifying an IP network, the so called *prefix length*.
- An *interface identifier* is the portion of an IP address that identifies an interface in a certain IP network.
- An *IP packet* is a bit sequence consisting of an IP header and the payload.
- The *link MTU* is the maximum transmission unit, i.e., maximum packet size in octets, that can be conveyed over a link.
- The *path MTU* is the the minimum link MTU of all the links in a path between a source node and a destination node.

# Internet Address / Prefix Assignment

- Manual: A network administrator assigns an IP prefix manually to an interface.
- System: A networking stack automatically assigns a prefix to an interface (e.g., 127.0.0.1/8 or ::1/128 for a loopback interface).
- Stateless automatic configuration: A networking stack automatically calculates and assigns an IP prefix (e.g., deriving an interface identifier from a lower-layer address and combining it with a learned prefix).
- Stateful automatic configuration: A networking stack obtains an prefix from a service providing IP addresses on request (e.g., DHCP).
- Temporary addresses: A networking stack generates temporary addresses from a know prefix in order to enhance privacy.

# Jacobs University's IP Networks

- Jacobs University currently uses the global IPv4 address blocks 212.201.44.0/22 and 212.201.48.0/23. How many IPv4 addresses can be used in these two address spaces?
- 212.201.44.0/22:  $2^{32-22} - 2 = 2^{10} - 2 = 1022$   
212.201.48.0/23:  $2^{32-23} - 2 = 2^9 - 2 = 510$
- Jacobs University currently uses the global IPv6 address block 2001:638:709::/48. How many IPv6 addresses can be used?
- 2001:638:709::/48:  $2^{128-48} - 2 = 2^{80} - 2$  which is 1208925819614629174706174.
- If you equally distribute the addresses over the campus area ( $30 \cdot 10^4 m^2$ ), what is the space covered per address?

# Internet Network Layer Protocols

- IPv6:
  - The *Internet Protocol version 6* (IPv6) provides for transmitting datagrams from sources to destinations using 16 byte IPv6 addresses
  - The *Internet Control Message Protocol version 6* (ICMPv6) is used for IPv6 error reporting, testing, auto-configuration and address resolution
- IPv4:
  - The *Internet Protocol version 4* (IPv4) provides for transmitting datagrams from sources to destinations using 4 byte IPv4 addresses
  - The *Internet Control Message Protocol version 4* (ICMPv4) is used for IPv4 error reporting and testing
  - The *Address Resolution Protocol* (ARP) maps IPv4 addresses to IEEE 802 addresses

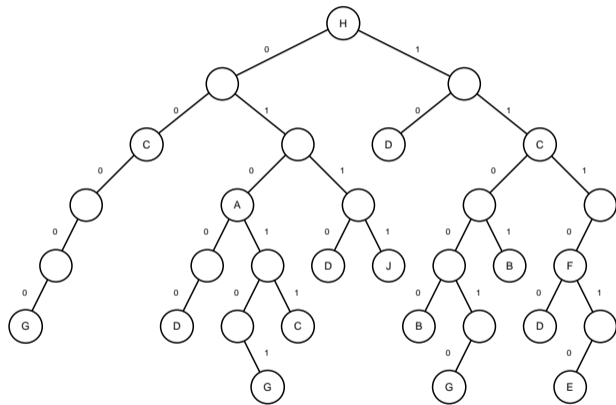
# IP Forwarding

- IP addresses can be divided into a part which identifies a network (the network prefix) and a part which identifies an interface of a node within that network (the interface identifier).
- The *forwarding table* realizes a mapping of the network prefix to the next node (next hop) closer to the destination and the local interface used to reach the next node.
- For every IP packet, the entry in the forwarding table with the longest matching prefix for the destination address has to be found (longest prefix match).
- A default forwarding table entry (if it exists) uses a zero-length prefix, that is either 0.0.0.0/0 (IPv4) or ::/0 (IPv6).

# IP Forwarding Table Management

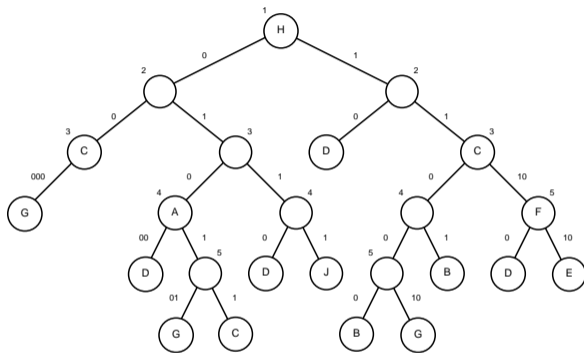
- Entries of the IP forwarding table may be created by different entities:
  - Manual: A network administrator creates entries in the IP forwarding table manually.
  - System: A networking stack automatically creates forwarding entries (e.g., when assigning a prefix to a network interface).
  - Automatic Configuration Protocols: Protocols discovering valid prefixes or obtaining IP addresses and prefixes dynamically from a pool may create suitable IP forwarding table entries.
  - Routing Protocols: Distributed routing protocols create and maintain one or more routing tables that these routing tables feed data into the IP forwarding table.
- Some implementations support multiple forwarding tables that can be selected by certain packet properties.

# Longest Prefix Match: Binary Trie



- A binary trie is the representation of the binary prefixes in a tree.

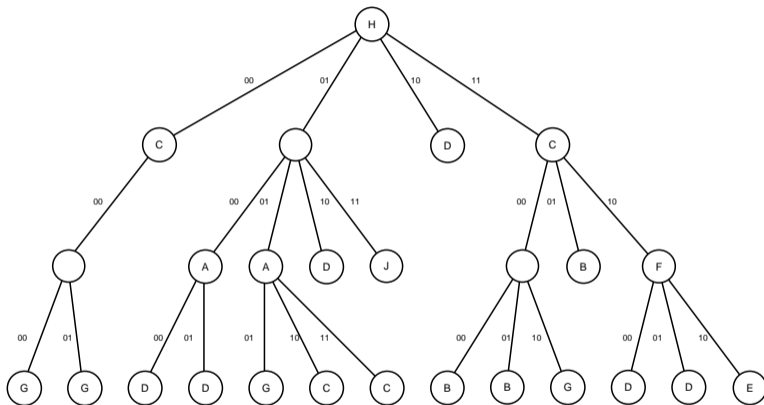
# Longest Prefix Match: Path Compressed Trie



- A path compressed trie is obtained by collapsing all one-way branch nodes.
- The number attached to nodes indicates the next (absolute) bit to inspect.
- While walking down the tree, you verify in each step that the prefix still matches the prefix stored at each node.



# Longest Prefix Match: Two-bit stride multibit Trie



- A two-bit multibit trie reduces the number of memory accesses.

# Section 19: Internet Protocol Version 6

18 Concepts and Terminology

**19 Internet Protocol Version 6**

20 Internet Protocol Version 4

# IPv6 Interface Identifier

- Interface identifiers in IPv6 unicast addresses are used to uniquely identify interfaces on a link.
- For unicast addresses, except those that start with binary 000, interface identifiers are generally required to be 64 bits long.
- Combination of the interface identifier with a network prefix leads to an IPv6 address.
- Link local unicast addresses have the prefix fe80::/10.
- Interface identifier may be obtained from an IEEE 802 MAC address using a modified EUI-64 format, but this has privacy issues.
- Alternatively, it is possible to use temporary interface identifiers that keep changing.

# Modified EUI-64 Format

```
|0          1|1          3|3          4|
|0          5|6          1|2          7|
+-----+-----+-----+
|cccccc0gccccccc|ccccccccmmmmmmmm|mmmmmmmmmmmmmmmm|
+-----+-----+-----+
```

```
|0          1|1          3|3          4|4          6|
|0          5|6          1|2          7|8          3|
+-----+-----+-----+-----+
|cccccc1gccccccc|cccccccc11111111|11111110mmmmmmmm|mmmmmmmmmmmmmmmm|
+-----+-----+-----+-----+
```

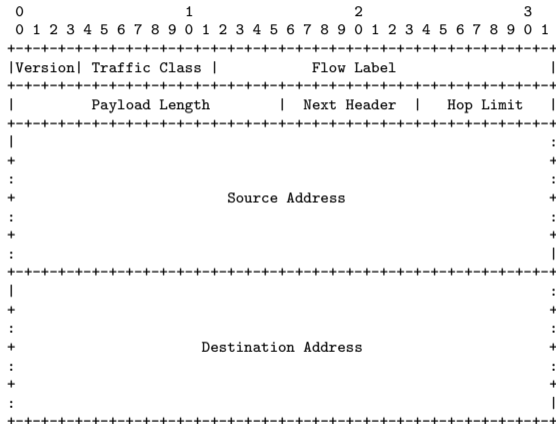
- Modified EUI-64 format can be obtained from IEEE 802 MAC addresses.
- Warning: Ipv6 addresses derived from MAC addresses can be used to track mobile nodes used in different networks.
- Solution: Temporary addresses with interface identifiers based on time-varying random bit strings and relatively short lifetimes.

# IPv6 Multicast Addresses

Address	Description
ff02::1	All nodes on the local link.
ff02::2	All routers on the local link.
ff02::3	All hosts on the local link.
ff02::1:2	All DHCP servers and relay agents on a local link.
ff02::fb	All multicast DNS servers on a local link.

- IPv6 multicast addresses use the prefix ff00::/8.
- The addresses listed above are some of the well-known multicast addresses.
- Applications can, of course, allocate additional multicast addresses.

# IPv6 Packet Format (RFC 8200)



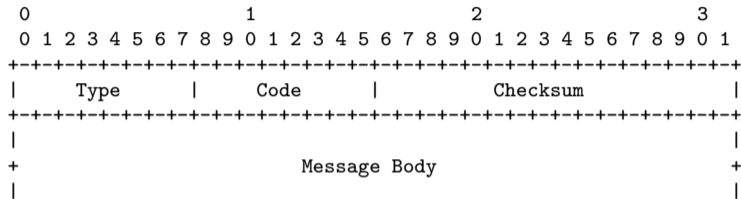
# IPv6 Extension Header

- All IPv6 header extensions and options are carried in a header daisy chain.
  - Hop-by-Hop Options Extension Header (HO)
  - Destination Options Extension Header (DO)
  - Routing Extension Header (RH)
  - Fragment Extension Header (FH)
  - Authentication Extension Header (AH)
  - Encapsulating Security Payload Extension Header (ESP)
- Link MTUs must be at least 1280 octets and only the sender is allowed to fragment packets.

- IPv6 packets are forwarded using the longest prefix match algorithm.
- IPv6 addresses have relatively long prefixes, which allows network operators to achieve better address aggregation, which reduces the number of forwarding table entries needed in the backbone infrastructure.
- Due to the length of the prefixes, it is crucial to use a longest prefix match algorithm whose complexity does not depend on the number of entries in the forwarding table or the average prefix length.
- Due to better aggregation possibilities, IPv6 forwarding tables can be expected to be shorter than IPv4 forwarding tables



# IPv6 Error Handling (ICMPv6) (RFC 4443)



- The Internet Control Message Protocol Version 6 (ICMPv6) is used
  - to report error situations,
  - to run diagnostic tests,
  - to auto-configure IPv6 nodes, and
  - to supports the resolution of IPv6 addresses to link-layer addresses.

# IPv6 Neighbor Discovery (RFC 4861)

- Discovery of the routers attached to a link.
- Discovery of the prefixes used on a link.
- Discovery of parameters such as the link MTU or the hop limit for outgoing packets.
- Automatic configuration of IPv6 addresses.
- Resolution of IPv6 addresses to link-layer addresses.
- Determination of next-hop addresses for IPv6 destination addresses.
- Detection of unreachable nodes which are attached to the same link.
- Detection of conflicts during address generation.
- Discovery of better alternatives to forward packets.

# IPv6 over IEEE 802.3 (RFC 2464)

- Frames containing IPv6 packets are identified by the value 0x86dd in the IEEE 802.3 type field.
- The link MTU is 1500 bytes, which corresponds to the IEEE 802.3 maximum frame size of 1500 byte.
- The mapping of IPv6 addresses to IEEE 802.3 addresses is table driven. Entries in so called address translation tables can be either statically configured or dynamically learned using the neighbor discovery protocol.
- IPv6 over IEEE 802.3 does not use IEEE LLC encapsulation.

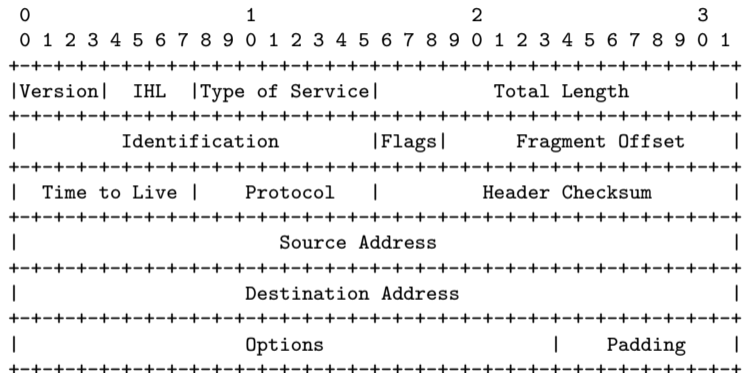
# Section 20: Internet Protocol Version 4

18 Concepts and Terminology

19 Internet Protocol Version 6

**20 Internet Protocol Version 4**

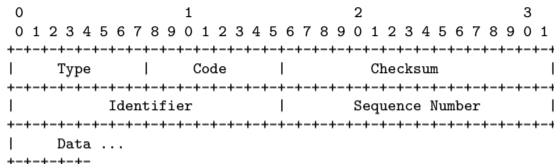
# IPv4 Packet Format (RFC 791)



# IPv4 Error Handling (ICMPv4)

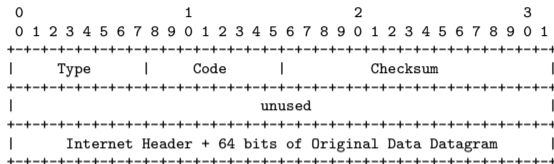
- The Internet Control Message Protocol (ICMP) is used to inform nodes about problems encountered while forwarding IP packets.
  - Echo Request/Reply messages are used to test connectivity.
  - Unreachable Destination messages are used to report why a destination is not reachable.
  - Redirect messages are used to inform the sender of a better (shorter) path.
- Can be used to trace routes to hosts:
  - Send messages with increasing TTL starting with one and interpret the ICMP response message.
  - Pack additional data into the request to measure latency.
- ICMPv4 is an integral part of IPv4 (even though it is a different protocol).

# ICMPv4 Echo Request/Reply



- The ICMP echo request message (type = 8, code = 0) asks the destination node to return an echo reply message (type = 0, code = 0).
- The Identifier and Sequence Number fields are used to correlate incoming replies with outstanding requests.
- The data field may contain any additional data.

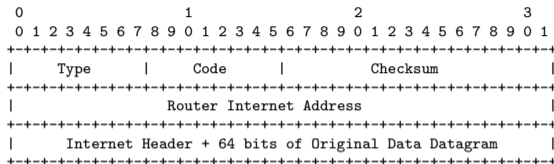
# ICMPv4 Unreachable Destinations



- The Type field has the value 3 for all unreachable destination messages.
- The Code field indicates why a certain destination is not reachable.
- The data field contains the beginning of the packet which caused the ICMP unreachable destination message.



# ICMPv4 Redirect



- The Type field has the value 5 for redirect messages.
- The Code field indicates which type of packets should be redirected.
- The Router Internet Address field identifies the IP router to which packets should be redirected.
- The data field contains the beginning of the packet which caused the ICMP redirect message.

# IPv4 Fragmentation

- IPv4 packets that do not fit the outgoing link MTU will get fragmented into smaller packets that fit the link MTU.
  - The Identification field contains the same value for all fragments of an IPv4 packet.
  - The Fragment Offset field contains the relative position of a fragment of an IPv4 packet (counted in 64-bit words).
  - The flag More Fragments (MF) is set if more fragments follow.
- The Don't Fragment (DF) flag can be set to indicate that a packet should not be fragmented.
- IPv4 allows fragments to be further fragmented without intermediate reassembly.

# Fragmentation Considered Harmful

- The receiver must buffer fragments until all fragments have been received. However, it is not useful to keep fragments in a buffer indefinitely. Hence, the TTL field of all buffered packets will be decremented once per second and fragments are dropped when the TTL field becomes zero.
- The loss of a fragment causes in most cases the sender to resend the original IP packet which in most cases gets fragmented as well. Hence, the probability of transmitting a large IP packet successfully goes quickly down if the loss rate of the network goes up.
- Since the Identification field identifies fragments that belong together and the number space is limited, one cannot fragment an arbitrary large number of packets.

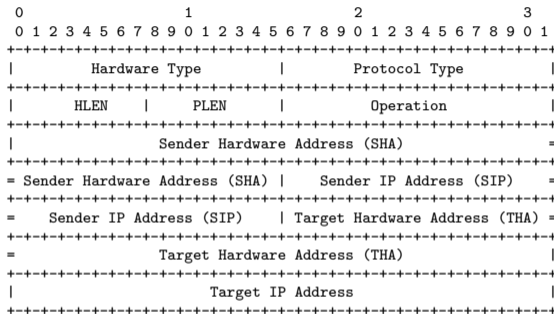
# MTU Path Discovery (RFC 1191)

- The sender sends IPv4 packets with the DF flag set.
- A router which has to fragment a packet with the DF flag turned on drops the packet and sends an ICMP message back to the sender which also includes the local maximum link MTU.
- Upon receiving the ICMP message, the sender adapts his estimate of the path MTU and retries.
- Since the path MTU can change dynamically (since the path can change), a once learned path MTU should be verified and adjusted periodically.
- Not all routers send necessarily the local link MTU. In this cases, the sender tries typical MTU values, which is usually faster than doing a binary search.

# IPv4 over IEEE 802.3 (RFC 894)

- IPv4 packets are identified by the value 0x800 in the IEEE 802.3 type field.
- According to the maximum length of IEEE 802.3 frames, the maximum link MTU is 1500 byte.
- The mapping of IPv4 addresses to IEEE 802.3 addresses is table driven. Entries in so called mapping tables (sometimes also called address translation tables) can either be statically configured or dynamically learned.
- Note that the RFC 894 approach does not provide an assurance that the mapping is actually correct...

# IPv4 Address Translation (RFC 826)



- The Address Resolution Protocol (ARP) resolved IPv4 addresses to link-layer addresses of neighboring nodes.

# ARP and RARP

- The `Hardware Type` field identifies the address type used on the link-layer (the value 1 is used for IEEE 802.3 MAC addresses).
- The `Protocol Type` field identifies the network layer address type (the value 0x800 is used for IPv4).
- ARP/RARP packets use the 802.3 type value 0x806.
- The `Operation` field contains the message type: ARP Request (1), ARP Response (2), RARP Request (3), RARP Response (4).
- The sender fills, depending on the request type, either the `Target IP Address` field (ARP) or the `Target Hardware Address` field (RARP).
- The responding node swaps the `Sender/Target` fields and fills the empty fields with the requested information.

# DHCP Version 4 (DHCPv4)

- The Dynamic Host Configuration Protocol (DHCP) allows nodes to retrieve configuration parameters from a central configuration server.
- A binding is a collection of configuration parameters, including at least an IP address, associated with or bound to a DHCP client.
- Bindings are managed by DHCP servers.
- Bindings are typically valid only for a limited lifetime.
- See RFC 2131 for the details and the message formats.
- See RFC 3118 for security aspects due to lack of authentication.



# DHCPv4 Message Types

- The DHCPDISCOVER message is a broadcast message which is sent by DHCP clients to locate DHCP servers.
- The DHCPOFFER message is sent from a DHCP server to offer a client a set of configuration parameters.
- The DHCPREQUEST is sent from the client to a DHCP server as a response to a previous DHCPOFFER message, to verify a previously allocated binding or to extend the lease of a binding.
- The DHCPACK message is sent by a DHCP server with some additional parameters to the client as a positive acknowledgement to a DHCPREQUEST.
- The DHCPNAK message is sent by a DHCP server to indicate that the client's notion of a configuration binding is incorrect.

## DHCPv4 Message Types (cont.)

- The DHCPDECLINE message is sent by a DHCP client to indicate that parameters are already in use.
- The DHCPRELEASE message is sent by a DHCP client to inform the DHCP server that configuration parameters are no longer used.
- The DHCPINFORM message is sent from the DHCP client to inform the DHCP server that only local configuration parameters are needed.

# References I



C. Kent and J. Mogul.

Fragmentation Considered Harmful.

In *Proc. SIGCOMM '87 Workshop on Frontiers in Computer Communications Technology*, August 1987.



R.P. Draves, C. King, S. Venkatachary, and B.N. Zill.

Constructing Optimal IP Routing Tables.

In *Proc. 18th IEEE INFOCOM 1999*, pages 88–97, New York, March 1999.



M. A. Ruiz-Sánchez, E. W. Biersack, and W. Dabbous.

Survey and Taxonomy of IP Address Lookup Algorithms.

*IEEE Network*, 15(2):8–23, March 2001.



D. C. Plummer.

An Ethernet Address Resolution Protocol.

RFC 826, MIT, November 1982.



C. Hornig.

A Standard for the Transmission of IP Datagrams over Ethernet Networks.

RFC 894, Symbolics Cambridge Research Center, April 1984.



J. Mogul and S. Deering.

Path MTU Discovery.

RFC 1191, DECWRL, Stanford University, November 1990.

# References II



R. Droms.

Dynamic Host Configuration Protocol.  
RFC 2131, Bucknell University, March 1997.



R. Droms and W. Arbaugh.

Authentication for DHCP Messages.  
RFC 3118, Cisco Systems, University of Maryland, June 2001.



S. Deering and R. Hinden.

Internet Protocol, Version 6 (IPv6) Specification.  
RFC 2460, Cisco, Nokia, December 1998.



T. Narten, E. Nordmark, W. Simpson, and H. Soliman.

Neighbor Discovery for IP version 6 (IPv6).  
RFC 4861, IBM, Sun Microsystems, Daydreamer, Elevate Technologies, September 2007.



A. Conta, S. Deering, and M. Gupta.

Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification.  
RFC 4443, Transwitch, Cisco Systems, Tropos Networks, March 2006.



M. Crawford.

Transmission of IPv6 Packets over Ethernet Networks.  
RFC 2464, Fermilab, December 1998.

# References III



R. Droms, J. Bound, B. Volz, T. Lemon, C. Perkins, and M. Carney.

Dynamic Host Configuration Protocol for IPv6 (DHCPv6).

RFC 3315, Cisco, Hewlett Packard, Ericsson, Nominum, Nokia Research Center, Sun Microsystems, July 2003.



IANA.

Special-Use IPv4 Addresses.

RFC 3330, Internet Assigned Numbers Authority, September 2002.



M. Blanchet.

Special-Use IPv6 Addresses.

RFC 5156, Viagenie, April 2008.



T. Narten, R. Draves, and S. Krishnan.

Privacy Extensions for Stateless Address Autoconfiguration in IPv6.

RFC 4941, IBM Corporation, Microsoft Research, Ericsson Research, September 2007.

# Part 5: Internet Routing

- 21 Distance Vector Routing (RIP)
- 22 Link State Routing (OSPF)
- 23 Path Vector Policy Routing (BGP)

# Section 21: Distance Vector Routing (RIP)

21 Distance Vector Routing (RIP)

22 Link State Routing (OSPF)

23 Path Vector Policy Routing (BGP)

# Bellman-Ford

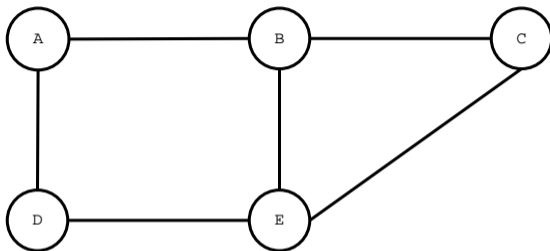
- Let  $G = (V, E)$  be a graph with the vertices  $V$  and the edges  $E$  with  $n = |V|$  and  $m = |E|$ .
- Let  $D$  be an  $n \times n$  distance matrix in which  $D(i, j)$  denotes the distance from node  $i \in V$  to the node  $j \in V$ .
- Let  $H$  be an  $n \times n$  matrix in which  $H(i, j) \in E$  denotes the edge on which node  $i \in V$  forwards a message to node  $j \in V$ .
- Let  $M$  be a vector with the link metrics,  $S$  a vector with the start node of the links and  $D$  a vector with the end nodes of the links.



## Bellman-Ford (cont.)

1. Set  $D(i, j) = \infty$  for  $i \neq j$  and  $D(i, j) = 0$  for  $i = j$ .
2. For all edges  $l \in E$  and for all nodes  $k \in V$ : Set  $i = S[l]$  and  $j = D[l]$  and  $d = M[l] + D(j, k)$ .
3. If  $d < D(i, k)$ , set  $D(i, k) = d$  and  $H(i, k) = l$ .
4. Repeat from step 2 if at least one  $D(i, k)$  has changed. Otherwise, stop.

# Bellman-Ford Example (Round 0)



A	Dest.	Link	Cost
	A	local	0
C	Dest.	Link	Cost
	C	local	0
E	Dest.	Link	Cost
	E	local	0

B	Dest.	Link	Cost
	B	local	0
D	Dest.	Link	Cost
	D	local	0

# Bellman-Ford Example (Round 1)

A	Dest.	Link	Cost
	A	local	0
	B	A-B	1
	D	A-D	1

C	Dest.	Link	Cost
	B	B-C	1
	C	local	0
	E	C-E	1

E	Dest.	Link	Cost
	B	B-E	1
	C	C-E	1
	D	D-E	1
	E	local	0

B	Dest.	Link	Cost
	A	A-B	1
	B	local	0
	C	B-C	1
	E	B-E	1

D	Dest.	Link	Cost
	A	A-D	1
	D	local	0
	E	D-E	1

# Bellman-Ford Example (Round 2)

A	Dest.	Link	Cost
	A	local	0
	B	A-B	1
	C	A-B	2
	D	A-D	1
	E	A-B	2

B	Dest.	Link	Cost
	A	A-B	1
	B	local	0
	C	B-C	1
	D	A-B	2
	E	B-E	1

C	Dest.	Link	Cost
	A	B-C	2
	B	B-C	1
	C	local	0
	D	C-E	2
	E	C-E	1

D	Dest.	Link	Cost
	A	A-D	1
	B	A-D	2
	C	D-E	2
	D	local	0
	E	D-E	1

E	Dest.	Link	Cost
	A	B-E	2
	B	B-E	1

# Count-to-Infinity

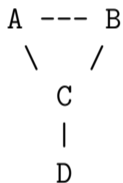
- Consider the following topology:

A --- B --- C

- After some distance vector exchanges, C has learned that he can reach A by sending packets via B.
- When the link between A and B breaks, B will learn from C that he can still reach A at a higher cost (count of hops) by sending a packet to C.
- This information will now be propagated to C, C will update the hop count and subsequently announce a more expensive not existing route to B.
- This counting continues until the costs reach infinity.

# Split Horizon

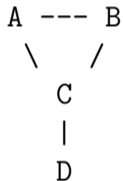
- Idea: Nodes never announce the reachability of a network to neighbors from which they have learned that a network is reachable.
- Does not solve the count-to-infinity problem in all cases:



- If the link between C and D breaks, B will not announce to C that it can reach D via C and A will not announce to C that it can reach D via C (split horizon).
- But after the next round of distance vector exchanges, A will announce to C that it can reach D via B and B will announce to C that it can reach D via A.

# Split Horizon with Poisoned Reverse

- Idea: Nodes announce the unreachability of a network to neighbors from which they have learned that a network is reachable.
- Does not solve the count-to-infinity problem in all cases:



- C now actively announces infinity for the destination D to A and B.
- However, since the exchange of the distance vectors is not synchronized to a global clock, the following exchange can happen:

# Split Horizon with Poisoned Reverse

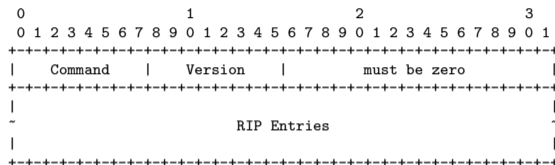
1. C first announces infinity for the destination D to A and B.
2. A and B now update their local state with the metric infinity for the destination D directly via C. The other stale information to reach D via the other directly connected node is not updated.
3. A and B now send their distance vectors. A and B now learn that they can not reach D via the directly connected nodes. However, C learns that it can reach D via either A or B.
4. C now sends its distance vector which contains false information to A and B and the count-to-infinity process starts.



# Routing Information Protocol (RIP)

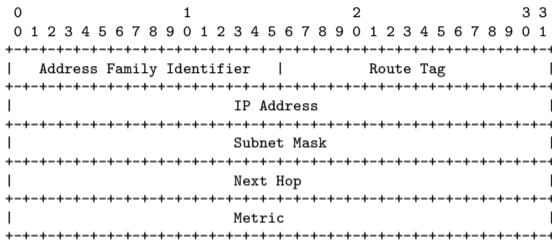
- The Routing Information Protocol version 2 (RIP-2) defined in RFC 2453 is based on the Bellman-Ford algorithm.
- RIP defines infinity to be 16 hops. Hence, RIP can only be used in networks where the longest paths (the network diameter) is smaller than 16 hops.
- RIP-2 runs over the User Datagram Protocol (UDP) and uses the well-known port number 520.

# RIP Message Format



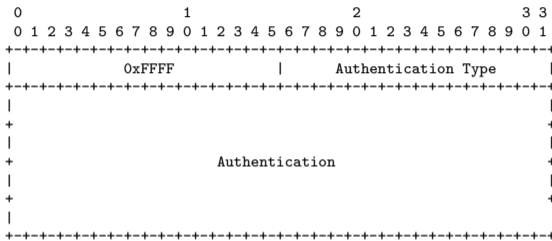
- The Command field indicates, whether the message is a request or a response.
- The Version field contains the protocol version number.
- The RIP Entries field contains a list of so called fixed size RIP Entries.

# RIP Message Format (cont.)



- The Address Family Identifier field identifies an address family.
- The Route Tag field marks entries which contain external routes (established by an EGP).

# RIP Message Format (cont.)



- Special RIP entry to support authentication.
- Originally only trivial cleartext password authentication.
- RFC 2082 defines authentication based on MD5 using a special RIP message trailer.

# Section 22: Link State Routing (OSPF)

21 Distance Vector Routing (RIP)

**22 Link State Routing (OSPF)**

23 Path Vector Policy Routing (BGP)

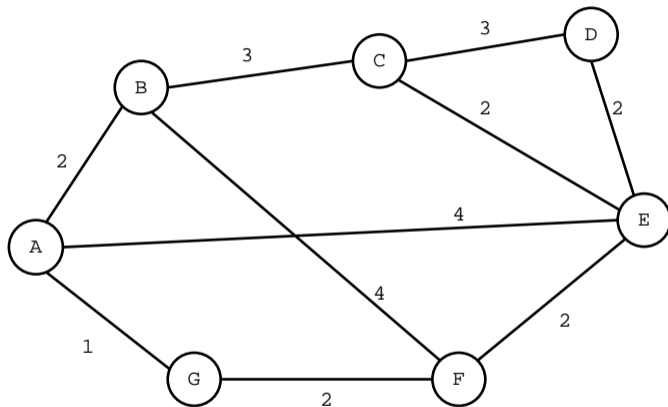
# Dijkstra's Shortest Path Algorithm

1. All nodes are initially tentatively labeled with infinite costs indicating that the costs are not yet known.
2. The costs of the root node are set to 0 and the root node is marked as the current node.
3. The current node's cost label is marked permanent.
4. For each node  $A$  adjacent to the current node  $C$ , the costs for reaching  $A$  are calculated as the costs of  $C$  plus the costs for the link from  $C$  to  $A$ . If the sum is less than  $A$ 's cost label, the label is updated with the new cost and the name of the current node.

## Dijkstra's Shortest Path Algorithm (cont.)

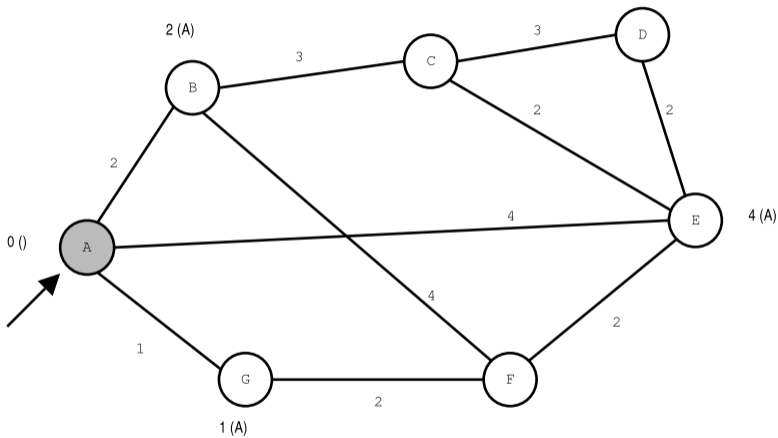
5. If there are still nodes with tentative cost labels, a node with the smallest costs is selected as the new current node. Goto step 3 if a new current node was selected.
6. The shortest paths to a destination node is given by following the labels from the destination node towards the root.

# Dijkstra Example

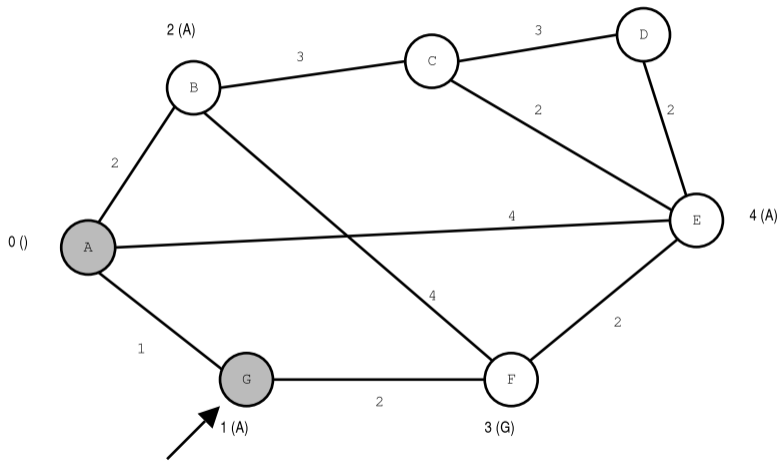




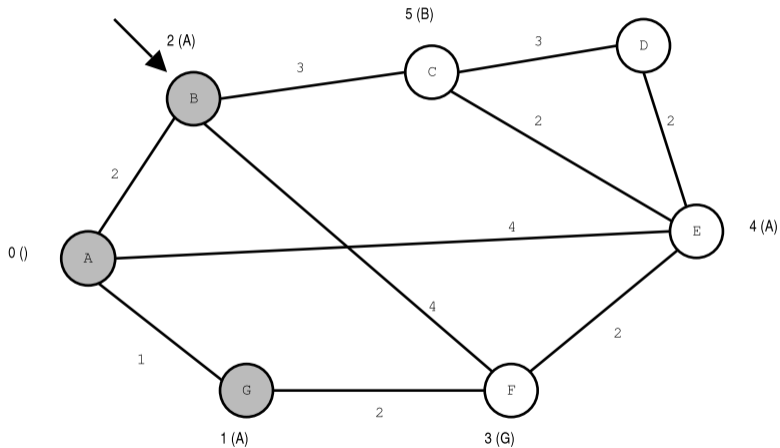
# Dijkstra Example (cont.)



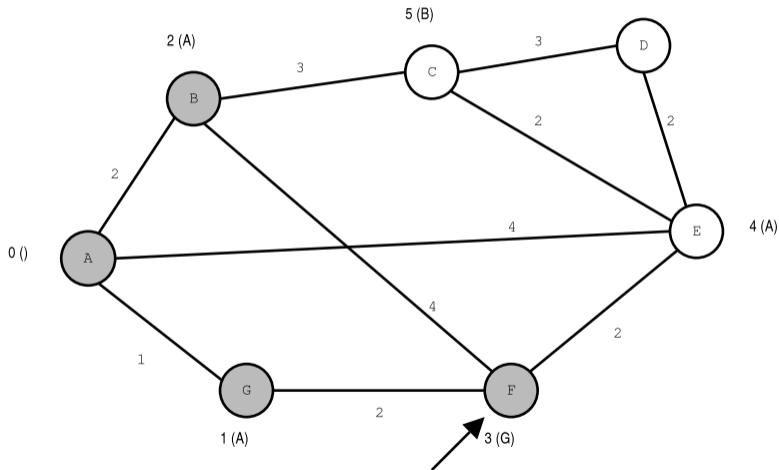
# Dijkstra Example (cont.)



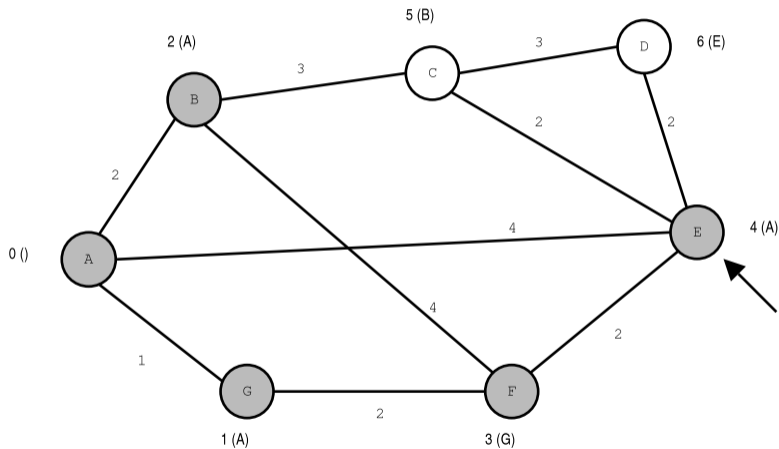
# Dijkstra Example (cont.)



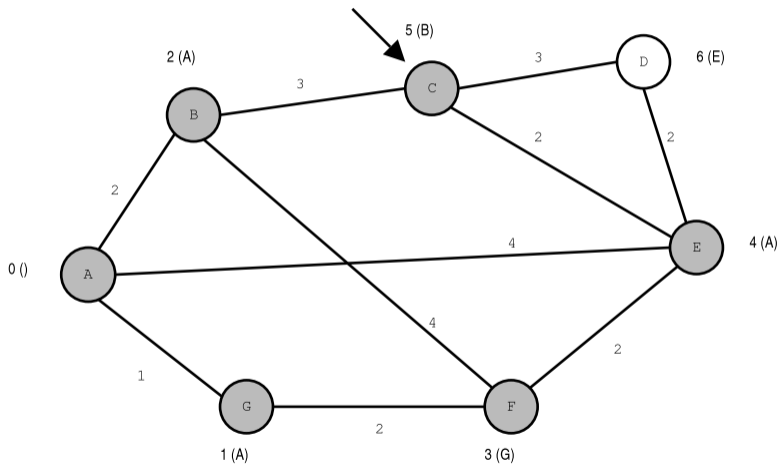
# Dijkstra Example (cont.)



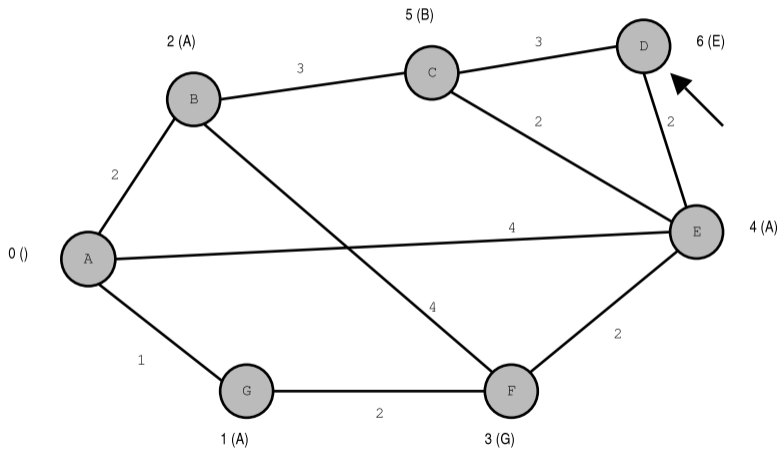
# Dijkstra Example (cont.)



# Dijkstra Example (cont.)



# Dijkstra Example (cont.)



# Open Shortest Path First (OSPF)

- The Open Shortest Path First (OSPF) protocol defined in RFC 2328 is a link state routing protocol.
- Every node independently computes the shortest paths to all the other nodes by using Dijkstra's shortest path algorithm.
- The link state information is distributed by flooding.
- OSPF introduces the concept of areas in order to control the flooding and computational processes.
- An OSPF area is a group of a set of networks within an autonomous system.
- The internal topology of an OSPF area is invisible for other OSPF areas. The routing within an area (intra-area routing) is constrained to that area.



- The OSPF areas are inter-connected via the OSPF backbone area (OSPF area 0). A path from a source node within one area to a destination node in another area has three segments (inter-area routing):
  1. An intra-area path from the source to a so called area border router.
  2. A path in the backbone area from the area border of the source area to the area border router of the destination area.
  3. An intra-area path from the area border router of the destination area to the destination node.

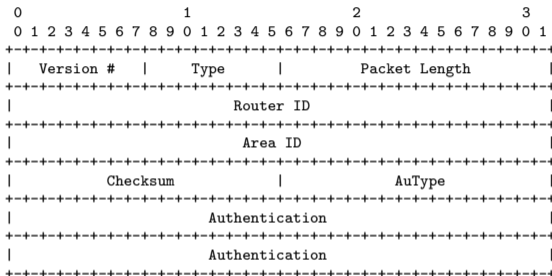
# OSPF Router Classification

- OSPF routers are classified according to their location in the OSPF topology:
  1. *Internal Router*: A router where all interfaces belong to the same OSPF area.
  2. *Area Border Router*: A router which connects multiple OSPF areas. An area border router has to be able to run the basic OSPF algorithm for all areas it is connected to.
  3. *Backbone Router*: A router that has an interface to the backbone area. Every area border router is automatically a backbone router.
  4. *AS Boundary Router*: A router that exchanges routing information with routers belonging to other autonomous systems.

# OSPF Stub Areas

- *Stub Areas* are OSPF areas with a single area border router.
- The routing in stub areas can be simplified by using default forwarding table entries which significantly reduces the overhead.

# OSPF Message Header



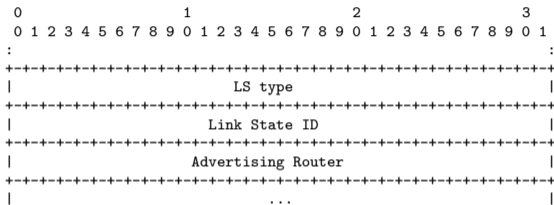
# OSPF Hello Message

```
0          1          2          3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
:
+-----+
|                               |
|           Network Mask       |
+-----+
|   HelloInterval   |   Options   |   Rtr Pri   |
+-----+
|   RouterDeadInterval   |
+-----+
|           Designated Router   |
+-----+
|   Backup Designated Router   |
+-----+
|           Neighbor           |
+-----+
|           ...               |
+-----+
```

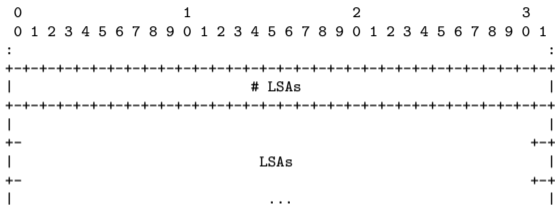
# OSPF Database Description Message

```
0                               1                               2                               3
0 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
:
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Interface MTU           |   Options   |0|0|0|0|0|0|I|M|MS|
+-----+-----+-----+-----+-----+-----+-----+
|                               DD sequence number                               |
+-----+-----+-----+-----+-----+-----+-----+
|                               |
+-                               +-
|                               |
+-                               +-
|           An LSA Header           |
+-                               +-
|                               |
+-                               +-
|                               |
+-----+-----+-----+-----+-----+-----+-----+
|                               ...                               |
```

# OSPF Link State Request Message

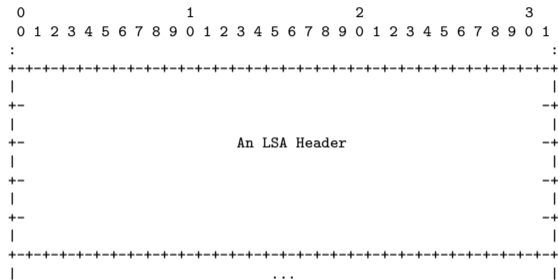


# OSPF Link State Update Message

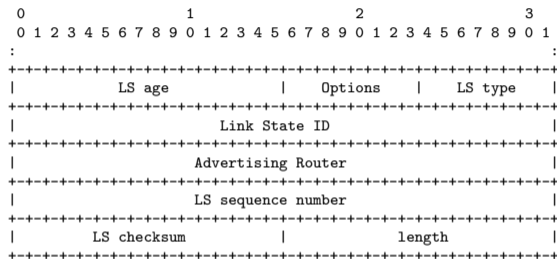




# OSPF Link State Ack. Message



# OSPF Link State Advertisement Header



# Section 23: Path Vector Policy Routing (BGP)

21 Distance Vector Routing (RIP)

22 Link State Routing (OSPF)

23 Path Vector Policy Routing (BGP)

# Border Gateway Protocol (RFC 1771)

- The Border Gateway Protocol version 4 (BGP-4) exchanges reachability information between autonomous systems.
- BGP-4 peers construct AS connectivity graphs to
  - detect and prune routing loops and
  - enforce policy decisions.
- BGP peers generally advertise only routes that should be seen from the outside (advertising policy).
- The final decision which set of announced paths is actually used remains a local policy decision.
- BGP-4 runs over a reliable transport (TCP) and uses the well-known port 179.

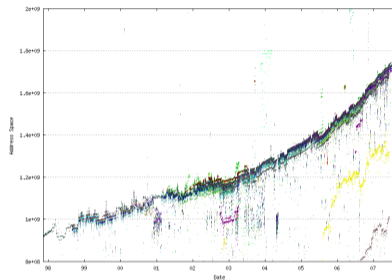
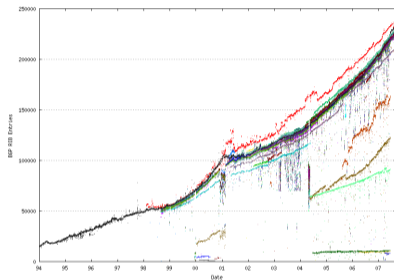
# AS Categories (RFC 1772)

- *Stub AS*:
  - A Stub AS has only a single peering relationship to one other AS.
  - A Stub AS only carries local traffic.
- *Multihomed AS*:
  - A Multihomed AS has peering relationships with more than one other AS, but refuses to carry transit traffic.
- *Transit AS*:
  - A Transit AS has peering relationships with more than one other AS, and is designed (under certain policy restrictions) to carry both transit and local traffic.

# Routing Policies

- Policies are provided to BGP in the form of configuration information and determined by the AS administration.
- Examples:
  1. A multihomed AS can refuse to act as a transit AS for other AS's. (It does so by only advertising routes to destinations internal to the AS.)
  2. A multihomed AS can become a transit AS for a subset of adjacent AS's, i.e., some, but not all, AS's can use the multihomed AS as a transit AS. (It does so by advertising its routing information to this set of AS's.)
  3. An AS can favor or disfavor the use of certain AS's for carrying transit traffic from itself.
- Routing Policy Specification Language (RFC 2622)

# BGP Routing Table Statistics



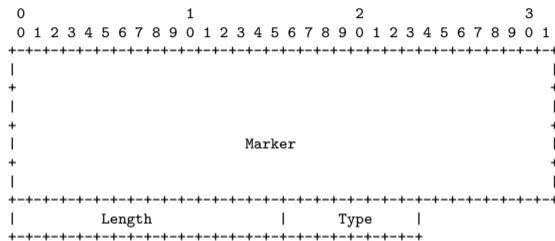
- <http://bgp.potaroo.net/>
- See also: G. Huston, "The BGP Routing Table", Internet Protocol Journal, March 2001

# BGP-4 Phases and Messages

- Once the transport connection has been established, BGP-4 basically goes through three phases:
  1. The BGP4 peers exchange OPEN messages to open and confirm connection parameters
  2. The BGP4 peers exchange initially the entire BGP routing table. Incremental updates are sent as the routing tables change. Uses BGP UPDATE messages.
  3. The BGP4 peers exchange so called KEEPALIVE messages periodically to ensure that the connection and the BGP-4 peers are alive.
- Errors lead to a NOTIFICATION message and subsequent close of the transport connection.

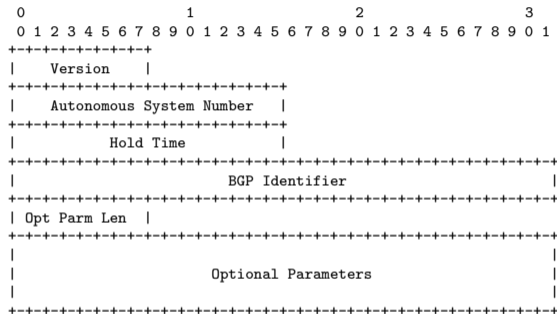


# BGP-4 Message Header



- The Marker is used for authentication and synchronization.
- The Type field indicates the message type and the Length field its length.

# BGP-4 Open Message



# BGP-4 Open Message

- The Version field contains the protocol version number.
- The Autonomous System Number field contains the 16-bit AS number of the sender.
- The Hold Time field specifies the maximum time that the receiver should wait for a response from the sender.
- The BGP Identifier field contains a 32-bit value which uniquely identifies the sender.
- The Opt Parm Len field contains the total length of the Optional Parameters field or zero if no optional parameters are present.
- The Optional Parameters field contains a list of parameters. Each parameter is encoded using a tag-length-value (TLV) triple.

# BGP-4 Update Message

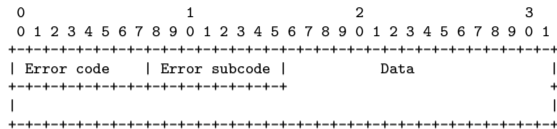
Unfeasible Routes Length (2 octets)
Withdrawn Routes (variable)
Total Path Attribute Length (2 octets)
Path Attributes (variable)
Network Layer Reachability Information (variable)

- The UPDATE message consists of two parts:
  1. The list of unfeasible routes that are being withdrawn.
  2. The feasible route to advertise.
- The Unfeasible Routes Length field indicates the total length of the Withdrawn Routes field in bytes.

# BGP-4 Update Message

- The `Withdrawn Routes` field contains a list of IPv4 address prefixes that are being withdrawn from service.
- The `Total Path Attribute Length` field indicates the total length of the `Path Attributes` field in bytes.
- The `Path Attributes` field contains a list of path attributes conveying information such as
  - the origin of the path information,
  - the sequence of AS path segments,
  - the IPv4 address of the next hop border router, or
  - the local preference assigned by a BGP4 speaker.
- The `Network Layer Reachability Information` field contains a list of IPv4 prefixes that are reachable via the path described in the path attributes fields.

# BGP-4 Notification Message



- NOTIFICATION messages are used to report errors.
- The transport connection is closed immediately after sending a NOTIFICATION.
- Six error codes plus 20 sub-codes.

# BGP-4 Keep Alive Message

- BGP-4 peers periodically exchange KEEPALIVE messages.
- A KEEPALIVE message consists of the standard BGP-4 header with no additional data.
- KEEPALIVE messages are needed to verify that shared state information is still present.
- If a BGP-4 peer does not receive a message within the Hold Time, then the peer will assume that there is a communication problem and tear down the connection.

- BGP communities are 32 bit values used to convey user-defined information
- A community is a group of destinations which share some common property
- Some well-known communities, e.g.:
  - NO\_EXPORT
  - NO\_ADVERTISE
- Most take the form AS:nn (written as 701:120) where the meaning of nn (encoded in the last 16 bits) depends on the source AS (encoded in the first 16 bits)
- Mostly used for special treatment of routes



# Internal BGP (iBGP)

- Use of BGP to distribute routing information within an AS.
- Requires to setup BGP sessions between all routers within an AS.
- Route Reflectors can be used to reduce the number of internal BGP sessions:
  - The Route Reflector collects all routing information and distributes it to all internal BGP routers.
  - Scales with  $O(n)$  instead of  $O(n^2)$  internal BGP sessions.
- BGP Confederations are in essence internal sub-ASes that do full mesh iBGP with a few BGP sessions interconnecting the sub-ASes.

# BGP Route Selection (cbgp)

1. Ignore if next-hop is unreachable
2. Prefer locally originated networks
3. Prefer highest Local-Pref
4. Prefer shortest AS-Path
5. Prefer lowest Origin
6. Prefer lowest Multi Exit Discriminator (metric)
7. Prefer eBGP over iBGP
8. Prefer nearest next-hop
9. Prefer lowest Router-ID or Originator-ID
10. Prefer shortest Cluster-ID-List
11. Prefer lowest neighbor address

# BGP's and Count-to-Infinity

- BGP does not suffer from the count-to-infinity problem of distance vector protocols:
  - The AS path information allows to detect loops.
- However, BGP iteratively explores longer and longer (loop free) paths.

# Multiprotocol BGP

- Extension to BGP-4 that makes it possible to distribute routing information for additional address families
- Announced as a capability in the open message
- Information for new protocol put into new path attributes
- Used to support IPv6, multicast, VPNs, ...

# References



G. Huston.  
The BGP Routing Table.  
*The Internet Protocol Journal*, 4(1), March 2001.



R. Chandra and P. Traina.  
BGP Communities Attribute.  
RFC 1997, Cisco Systems, August 1996.



Y. Rekhter, T. Li, and S. Hares.  
A Border Gateway Protocol 4 (BGP-4).  
RFC 4271, Juniper Networks, NextHop Technologies, January 2006.



I. van Beijnum.  
*BGP*.  
O'Reilly, September 2002.



B. Quoitin, C. Pelsser, L. Swinnen, O. Bonaventure, and S. Uhlig.  
Interdomain Traffic Engineering with BGP.  
*IEEE Communications Magazine*, 41(5):122–128, May 2003.



R. Mahajan, D. Wetherall, and T. Anderson.  
Understanding BGP Misconfiguration.  
In *Proc. SIGCOMM 2002*. ACM, August 2002.



J. Li, M. Guidero, Z. Wu, E. Purpus, and T. Ehrenkrantz.  
BGP Routing Dynamics Revisited.  
*SIGCOMM Computer Communication Review*, 37(2), April 2007.

# Part 6: Internet Transport Layer (UDP, TCP)

24 Transport Layer Overview

25 User Datagram Protocol (UDP)

26 Transmission Control Protocol (TCP)

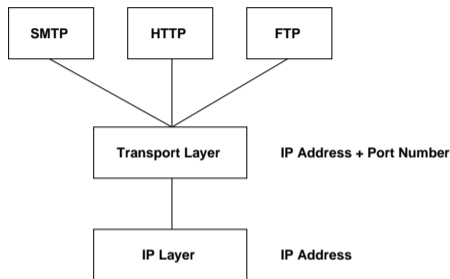
# Section 24: Transport Layer Overview

24 Transport Layer Overview

25 User Datagram Protocol (UDP)

26 Transmission Control Protocol (TCP)

# Internet Transport Layer



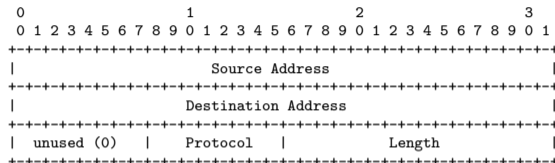
- Network layer addresses identify interfaces on nodes (node-to-node significance).
- Transport layer addresses identify communicating application processes (end-to-end significance).
- 16-bit port numbers enable the multiplexing and demultiplexing of packets at the transport layer.



# Internet Transport Layer Protocols Overview

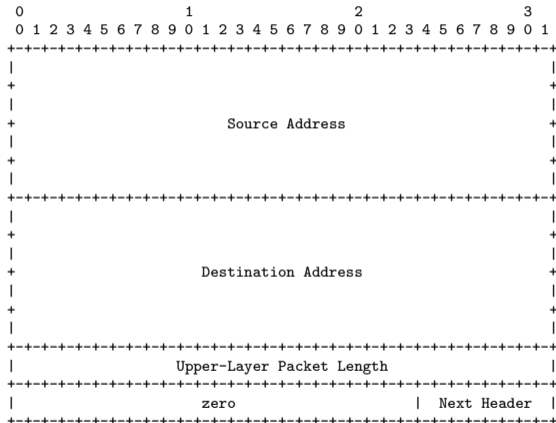
- The *User Datagram Protocol* (UDP) provides a simple unreliable best-effort datagram service.
- The *Transmission Control Protocol* (TCP) provides a bidirectional, connection-oriented and reliable data stream.
- The *Stream Control Transmission Protocol* (SCTP) provides a reliable transport service supporting sequenced delivery of messages within multiple streams, maintaining application protocol message boundaries (application protocol framing).
- The *Datagram Congestion Control Protocol* (DCCP) provides a congestion controlled, unreliable flow of datagrams suitable for use by applications such as streaming media.

# IPv4 Pseudo Header



- Pseudo headers are used during checksum computation.
- A pseudo header excludes header fields that are modified by routers.

# IPv6 Pseudo Header



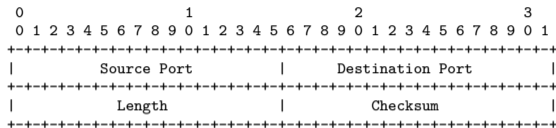
# Section 25: User Datagram Protocol (UDP)

24 Transport Layer Overview

**25 User Datagram Protocol (UDP)**

26 Transmission Control Protocol (TCP)

# User Datagram Protocol (UDP)



- UDP (RFC 768) provides an unreliable datagram transport service.
- The UDP header simply extends the IP header with source and destination port numbers and a checksum.
- UDP adds multiplexing services to the best effort packet delivery services provided by the IP layer.
- UDP datagrams can be multicasted to a group of receivers.

# User Datagram Protocol (UDP)

- The `Source Port` field contains the port number used by the sending application layer process.
- The `Destination Port` field contains the port number used by the receiving application layer process.
- The `Length` field contains the length of the UDP datagram including the UDP header counted in bytes.
- The `Checksum` field contains the Internet checksum computed over the pseudo header, the UDP header and the payload contained in the UDP packet.

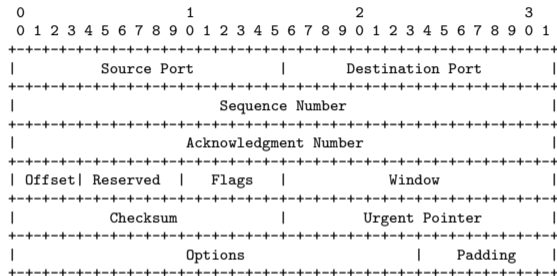
# Section 26: Transmission Control Protocol (TCP)

24 Transport Layer Overview

25 User Datagram Protocol (UDP)

26 Transmission Control Protocol (TCP)

# Transmission Control Protocol (TCP)



- TCP (RFC 793) provides a bidirectional connection-oriented and reliable data stream over an unreliable connection-less network protocol.



# Transmission Control Protocol (TCP)

- The `Source Port` field contains the port number used by the sending application layer process.
- The `Destination Port` field contains the port number used by the receiving application layer process.
- The `Sequence Number` field contains the sequence number of the first data byte in the segment. During connection establishment, this field is used to establish the initial sequence number.
- The `Acknowledgment Number` field contains the next sequence number which the sender of the acknowledgement expects.
- The `Offset` field contains the length of the TCP header including any options, counted in 32-bit words.

# Transmission Control Protocol (TCP)

- The `Flags` field contains a set of binary flags:

---

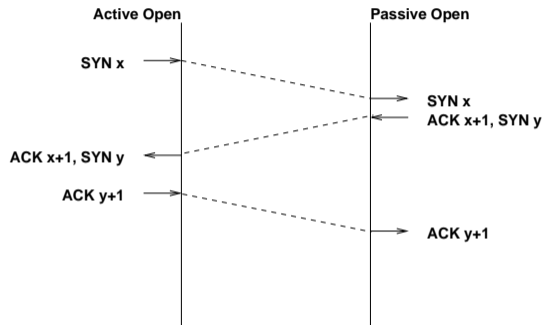
Flag	Description
URG	Indicates that the Urgent Pointer field is significant
ACK	Indicates that the Acknowledgment Number field is significant
PSH	Data should be pushed to the application as quickly as possible
RST	Reset of the connection
SYN	Synchronization of sequence numbers
FIN	No more data from the sender

---

# Transmission Control Protocol (TCP)

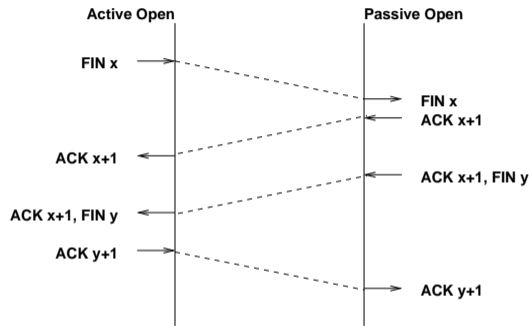
- The `Window` field indicates the number of data bytes which the sender of the segment is willing to receive.
- The `Checksum` field contains the Internet checksum computed over the pseudo header, the TCP header and the data contained in the TCP segment.
- The `Urgent Pointer` field points, relative to the actual segment number, to important data if the `URG` flag is set.
- The `Options` field can contain additional options.

# TCP Connection Establishment



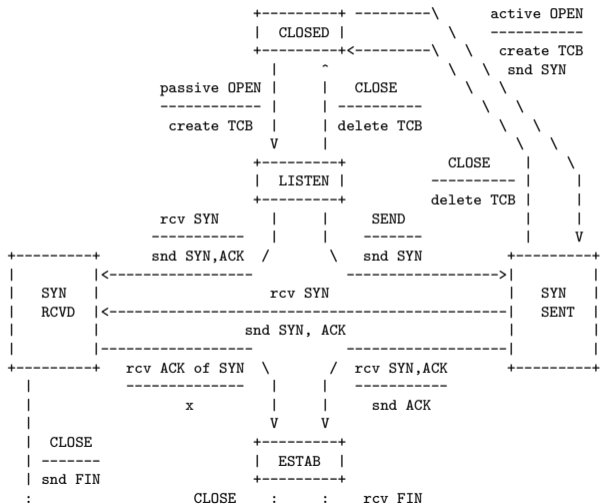
- Handshake protocol establishes TCP connection parameters and announces options.
- Guarantees correct connection establishment, even if TCP packets are lost or duplicated.

# TCP Connection Tear-down

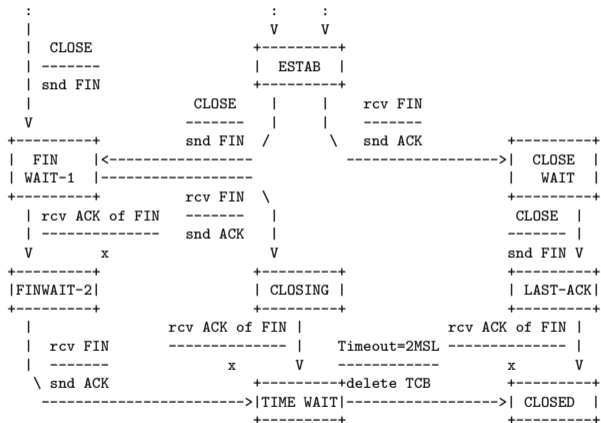


- TCP provides initially a bidirectional data stream.
- A TCP connection is terminated when both unidirectional connections have been closed. (It is possible to close only one half of a connection.)

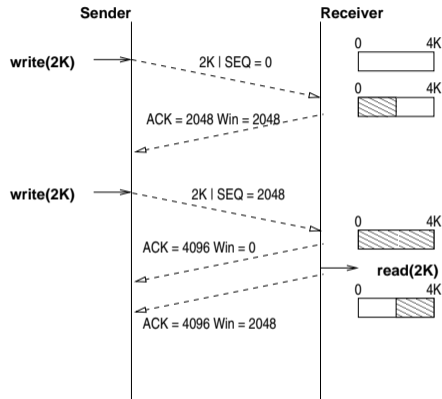
# TCP State Machine (Part #1)



# TCP State Machine (Part #2)



# TCP Flow Control



- Both TCP engines advertise their buffer sizes during connection establishment.
- The available space left in the receiving buffer is advertised as part of the acknowledgements.



# TCP Flow Control Optimizations

- Nagle's Algorithm
  - When data comes into the sender one byte at a time, just send the first byte and buffer all the rest until the byte in flight has been acknowledgement.
  - This algorithm provides noticeable improvements especially for interactive traffic where a quickly typing user is connected over a rather slow network.
- Clark's Algorithm
  - The receiver should not send a window update until it can handle the maximum segment size it advertised when the connection was established or until its buffer is half empty.
  - Prevents the receiver from sending a very small window updates (such as a single byte).

# TCP Congestion Control

- TCP's congestion control introduces the concept of a congestion window (*cwnd*) which defines how much data can be in transit.
- The congestion window is maintained by a TCP sender in addition to the flow control receiver window (*rwnd*), which is advertised by the receiver.
- The sender uses these two windows to limit the data that is sent to the network and not yet received (*flightsize*) to the minimum of the receiver and the congestion window:

$$flightsize \leq \min(cwin, rwin)$$

- The key problem to be solved is the dynamic estimation of the congestion window.

# TCP Congestion Control (cont.)

- The initial window (IW) is usually initialized using the following formula:

$$IW = \min(4 \cdot SMSS, \max(2 \cdot SMSS, 4380\text{bytes}))$$

*SMSS* is the sender maximum segment size, the size of the largest segment that the sender can transmit (excluding TCP/IP headers and options).

- During slow start, the congestion window *cwnd* increases by at most *SMSS* bytes for every received acknowledgement that acknowledges data. Slow start ends when *cwnd* exceeds *ssthresh* or when congestion is observed.
- Note that this algorithm leads to an exponential increase if there are multiple segments acknowledged in the *cwnd*.

# TCP Congestion Control (cont.)

- During congestion avoidance, *cwnd* is incremented by one full-sized segment per round-trip time (RTT). Congestion avoidance continues until congestion is detected. One formula commonly used to update *cwnd* during congestion avoidance is given by the following equation:

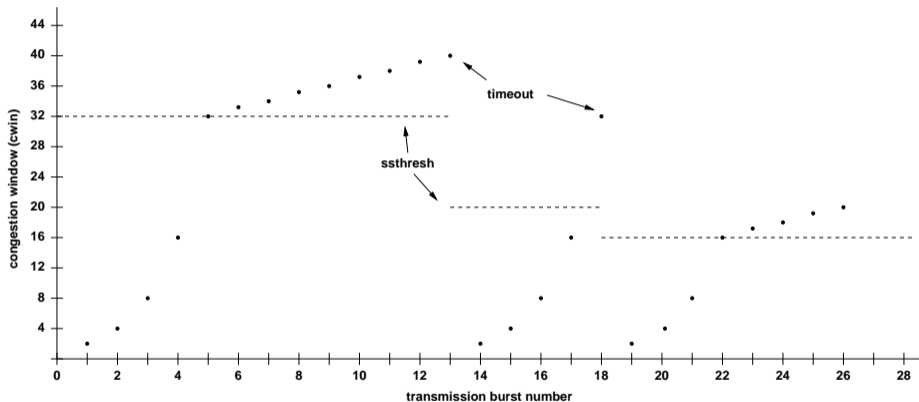
$$cwnd = cwnd + (SMSS * SMSS / cwnd)$$

This adjustment is executed on every incoming non-duplicate ACK.

- When congestion is noticed (the retransmission timer expires), then *cwnd* is reset to one full-sized segment and the slow start threshold *ssthresh* is updated as follows:

$$ssthresh = \max(\text{flightsize}/2, 2 \cdot SMSS)$$

# TCP Congestion Control (cont.)



- Congestion control with an initial window size of 2K.

# Retransmission Timer

- The retransmission timer controls when a segment is resent if no acknowledgement has been received.
- The retransmission timer  $RTT$  needs to adapt to round-trip time changes.
- General idea:
  - Measure the current round-trip time
  - Measure the variation of the round-trip time
  - Use the estimated round-trip time plus the measured variation to calculate the retransmit timeout
  - Do not update the estimators if a segment needs to be retransmitted (Karn's algorithm).

# Retransmission Timer

- If an acknowledgement is received for a segment before the associated retransmission timer expires:

$$RTT = \alpha \cdot RTT + (1 - \alpha)M$$

$M$  is the measured round-trip time;  $\alpha$  is typically  $\frac{7}{8}$ .

- The standard deviation is estimated using:

$$D = \alpha \cdot D + (1 - \alpha)|RTT - M|$$

$\alpha$  is a smoothing factor.

- The retransmission timeout  $RTO$  is determined as follows:

$$RTO = RTT + 4 \cdot D$$

The factor 4 has been chosen empirically.

# Fast Retransmit / Fast Recovery

- TCP receivers should send an immediate duplicate acknowledgement when an out-of-order segment arrives.
- The arrival of four identical acknowledgements without the arrival of any other intervening packets is an indication that a segment has been lost.
- The sender performs a fast retransmission of what appears to be the missing segment, without waiting for the retransmission timer to expire.
- Upon a fast retransmission, the sender does not exercise the normal congestion reaction with a full slow start since acknowledgements are still flowing.
- See RFC 2581 section 3.1 for details.



# Karn's Algorithm

- The dynamic estimation of the  $RTT$  has a problem if a timeout occurs and the segment is retransmitted.
- A subsequent acknowledgement might acknowledge the receipt of the first packet which contained that segment or any of the retransmissions.
- Karn suggested that the  $RTT$  estimation is not updated for any segments which were retransmitted and that the  $RTO$  is doubled on each failure until the segment gets through.
- The doubling of the  $RTO$  leads to an exponential back-off for each consecutive attempt.

# Selected TCP Options

- Maximum Segment Size (MSS):
  - Communicates the maximum receive segment size of the sender of this option during connection establishment
- Window Scale (WS):
  - The number carried in the 16-bit Window field of a TCP header is scaled (shifted) by a certain constant to enable windows larger than  $2^{16}$  octets
- TimeStamps (TS):
  - Timestamps exchanged in every TCP header to deal with the 32-bit sequence number space limitation (and to enhance round-trip time measurements)
- Selective Acknowledgment (SACK):
  - Indicate which blocks of the sequence number space are missing and which blocks are not (to improve cumulative acknowledgements)

# Explicit Congestion Notification

- Idea: Routers signal congestion by setting some special bits in the IP header.
  - The ECN bits are located in the Type-of-Service field of an IPv4 packet or the Traffic-Class field of an IPv6 packet.
  - TCP sets the ECN-Echo flag in the TCP header to indicate that a TCP endpoint has received an ECN marked packet.
  - TCP sets the Congestion-Window-Reduced (CWR) flag in the TCP header to acknowledge the receipt of and reaction to the ECN-Echo flag.
- ⇒ ECN uses the ECT and CE flags in the IP header for signaling between routers and connection endpoints, and uses the ECN-Echo and CWR flags in the TCP header for TCP-endpoint to TCP-endpoint signaling.

# TCP Performance

- Goal: Simple analytic model for steady state TCP behavior.
- We only consider congestion avoidance (no slow start).
- $W(t)$  denotes the congestion window size at time  $t$ .
- In steady state,  $W(t)$  increases to a maximum value  $W$  where it experiences congestion. As a reaction, the sender sets the congestion window to  $\frac{1}{2}W$ .
- The time interval needed to go from  $\frac{1}{2}W$  to  $W$  is  $T$  and we can send a window size of packets every  $RTT$ .
- Hence, the number  $N$  of packets is:

$$N = \frac{1}{2} \frac{T}{RTT} \left( \frac{W}{2} + W \right)$$

# TCP Performance (cont.)

- The time  $T$  between two packet losses equals  $T = RTT \cdot W/2$  since the window increases linearly.
- By substituting  $T$  and equating the total number of packets transferred with the packet loss probability, we get

$$\frac{W}{4} \cdot \left( \frac{W}{2} + W \right) = \frac{1}{p} \iff W = \sqrt{\frac{8}{3p}}$$

where  $p$  is the packet loss probability (thus  $\frac{1}{p}$  packets are transmitted between each packet loss).

- The average sending rate  $\bar{X}(p)$ , that is the number of packets transmitted during each period, then becomes:

$$\bar{X}(p) = \frac{1/p}{RTT \cdot W/2} = \frac{1}{RTT} \sqrt{\frac{3}{2p}}$$

# Part 7: Firewalls and Network Address Translators

27 Middleboxes

28 Firewalls

29 Network Address Translators

30 NAT Traversal (STUN)

# Section 27: Middleboxes

27 Middleboxes

28 Firewalls

29 Network Address Translators

30 NAT Traversal (STUN)

## Definition (RFC 3234)

A middlebox is any intermediary device performing functions other than the normal, standard functions of an IP router on the datagram path between a source host and destination host [?].

- A middlebox is not necessarily a physical box — it is usually just a function implemented in some other box.
- Middleboxes challenge the End-to-End principle and the hourglass model of the Internet architecture.
- Middleboxes are popular (whether we like this or not).



# Concerns about Middleboxes

- Protocols designed without consideration of middleboxes may fail, predictably or unpredictably, in the presence of middleboxes.
- Middleboxes introduce new failure modes; rerouting of IP packets around crashed routers is no longer the only case to consider.
- Configuration is no longer limited to the two ends of a session; middleboxes may also require configuration and management.
- Diagnosis of failures and misconfigurations is more complex.

# Types of Middleboxes

- *Network Address Translators (NAT)*: A function that dynamically assigns a globally unique address to a host that doesn't have one, without that host's knowledge.
- *NAT with Protocol Translator (NAT-PT)*: A function that performs NAT between an IPv6 host and an IPv4 network, additionally translating the entire IP header between IPv6 and IPv4 formats.
- *IP Tunnel Endpoints*: Tunnel endpoints, including virtual private network endpoints, use basic IP services to set up tunnels with their peer tunnel endpoints which might be anywhere in the Internet.
- *Transport Relays*: A middlebox which translates between two transport layer instances.

## Types of Middleboxes (cont.)

- *Packet classifiers, markers and schedulers*: Packet classifiers classify packets flowing through them according to policy and either select them for special treatment or mark them, in particular for differentiated services.
- *TCP performance enhancing proxies*: “TCP spoofer” are middleboxes that modify the timing or action of the TCP protocol in flight for the purposes of enhancing performance.
- *Load balancers that divert/munge packets*: Techniques that divert packets from their intended IP destination, or make that destination ambiguous.
- *IP Firewalls*: A function that screens and rejects packets based purely on fields in the IP and Transport headers.
- *Application Firewalls*: Application-level firewalls act as a protocol end point and relay

## Types of Middleboxes (cont.)

- *Application-level gateways (ALGs)*: ALGs translate IP addresses in application layer protocols and typically complement IP firewalls.
- *Transcoders*: Functions performing some type of on-the-fly conversion of application level data.
- *Proxies*: An intermediary program which acts as both a server and a client for the purpose of making requests on behalf of other clients.
- *Caches*: Caches are functions typically intended to optimise response times.
- *Anonymisers*: Functions that hide the IP address of the data sender or receiver. Although the implementation may be distinct, this is in practice very similar to a NAT plus ALG.
- ...

# Section 28: Firewalls

27 Middleboxes

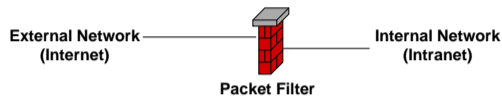
**28** Firewalls

29 Network Address Translators

30 NAT Traversal (STUN)

- A firewall is a system (or a set of systems) that enforce access control policies between two (or more) networks.
  - *Conservative firewalls* allow known desired traffic and reject everything else.
  - *Optimistic firewalls* reject known unwanted traffic and allow the rest.
  - Firewalls typically consist of packet filters, transport gateways and application level gateways.
  - Firewalls not only protect the “inside” from the “outside”, but also the “outside” from the “inside”.
- ⇒ There are many ways to circumvent firewalls if internal and external hosts cooperate.

# Firewall Architectures



- The simplest architecture is a packet filter which is typically implemented within a router that connects the internal network with the external network.
- Sometimes called a “screening router”.

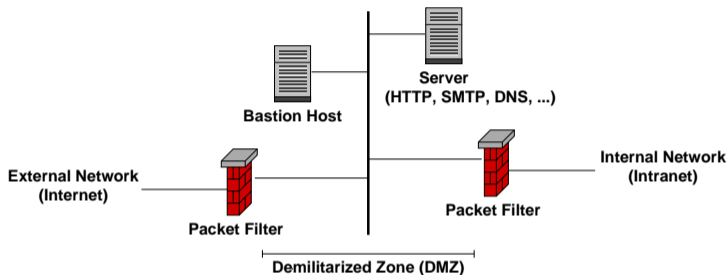
# Firewall Architectures



- A bastion host is a multihomed host connected to the internal and external network which does not forward IP datagrams but instead provides suitable gateways.
- Effectively prevents any direct communication between hosts on the internal network with hosts on the external network.

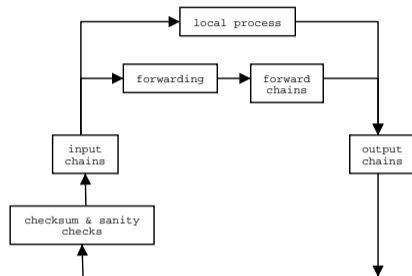


# Firewall Architectures



- The most common architecture consists of two packet filters which create a demilitarized zone (DMZ).
- Externally visible servers and gateways are located in the DMZ.

# Packetfilter Example: Linux ipchains



- Input chains are lists of filter rules applied to all incoming IP packets.
- Output chains are lists of filter rules applied to all outgoing IP packet.
- Forward chains are lists of filter rules applied to all forwarded IP packets.

# Section 29: Network Address Translators

27 Middleboxes

28 Firewalls

29 Network Address Translators

30 NAT Traversal (STUN)

# Network Address Translators

- *Basic Network Address Translation (NAT)*:  
Translates private IP addresses into public IP addresses.
- *Network Address Port Translation (NAPT)*:  
Translates transport endpoint identifiers. NAPT allows to share a single public address among many private addresses (masquerading).
- *Bi-directional NAT (Two-Way NAT)*:  
Translates outbound and inbound and uses DNS-ALGs to facilitate bi-directional name to address mappings.
- *Twice NAT*:  
A variation of a NAT which modifies both the source and destination addresses of a datagram. Used to join overlapping address domains.

# Network Address Port Translation Example

Ext. IP	Ext. Port	Int. IP	Int. Port
212.201.44.241	12345	10.50.1.1	1234
212.201.44.241	54321	10.50.1.2	1234
212.201.44.241	15243	10.50.1.1	4321



**External Network**

**Internal Network**

# Full Cone NAT

- A full cone NAT is a NAT where all requests from the same internal IP address and port are mapped to the same external IP address and port.
- Any external host can send a packet to the internal host, by sending a packet to the mapped external address.

# Restricted Cone NAT

- A restricted cone NAT is a NAT where all requests from the same internal IP address and port are mapped to the same external IP address and port.
- Unlike a full cone NAT, an external host (with IP address  $X$ ) can send a packet to the internal host only if the internal host had previously sent a packet to IP address  $X$ .

# Port Restricted Cone NAT

- A port restricted cone NAT is like a restricted cone NAT, but the restriction includes port numbers.
- Specifically, an external host can send a packet, with source IP address  $X$  and source port  $P$ , to the internal host only if the internal host had previously sent a packet to IP address  $X$  and port  $P$ .



# Symmetric NAT

- A symmetric NAT is a NAT where all requests from the same internal IP address and port, to a specific destination IP address and port, are mapped to the same external IP address and port.
- If the same host sends a packet with the same source address and port, but to a different destination, a different mapping is used.
- Only the external host that receives a packet can send a UDP packet back to the internal host.

# Section 30: NAT Traversal (STUN)

27 Middleboxes

28 Firewalls

29 Network Address Translators

**30 NAT Traversal (STUN)**

# STUN (RFC 5389)

- Session Traversal Utilities for NAT (STUN)
- Client / server protocol used for NAT discovery:
  1. The client sends a request to a STUN server
  2. The server returns a response containing the IP address seen by the server (i.e., a mapped address)
  3. The client compares its IP address with the IP address returned by the server; if they are different, the client is behind a NAT and learns its mapped address
- RFC 5780 details a number of tests that can be performed using STUN to determine the behaviour of a NAT.

# References



B. Carpenter and S. Brim.  
Middleboxes: Taxonomy and Issues.  
RFC 3234, IBM Zurich Research Laboratory, February 2002.



P. Srisuresh and M. Holdrege.  
IP Network Address Translator (NAT) Terminology and Considerations.  
RFC 2663, Lucent Technologies, August 1999.



B. Carpenter.  
Internet Transparency.  
RFC 2775, IBM, February 2000.



N. Freed.  
Behavior of and Requirements for Internet Firewalls.  
RFC 2979, Sun, October 2000.



J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy.  
STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs).  
RFC 3489, dynamicsoft, Microsoft, Cisco, March 2003.



E. D. Zwicky, S. Cooper, and D. B. Chapman.  
*Building Internet Firewalls*.  
O'Reilly, 2 edition, 2000.

# Part 8: Domain Name System (DNS)

31 Overview and Features

32 Resource Records

33 Message Formats

34 Security and Dynamic Updates

35 Creative Usage

# Section 31: Overview and Features

31 Overview and Features

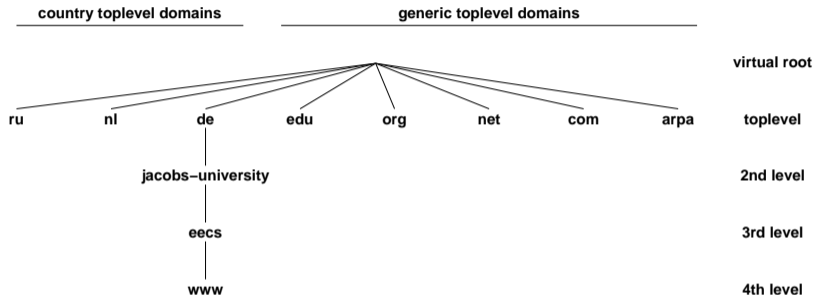
32 Resource Records

33 Message Formats

34 Security and Dynamic Updates

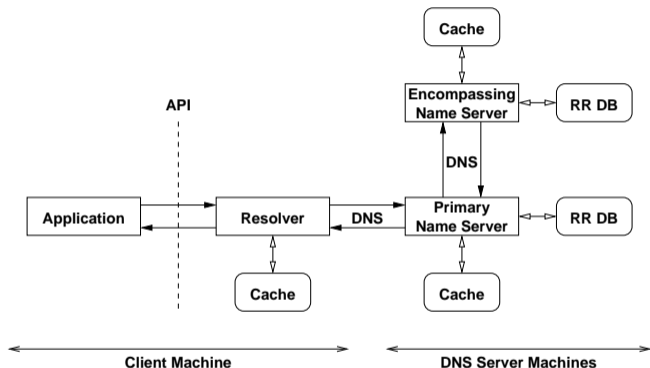
35 Creative Usage

# Domain Name System (DNS)



- The Domain Name System (DNS) provides a global infrastructure to map human friendly domain names into addresses (and other data).
- The DNS is a critical resource since most Internet users depend on name resolution services provided by the DNS.

# Resolver and Name Resolution



- The resolver is typically tightly integrated into the operating system (or more precisely standard libraries).



# DNS Characteristics

- Hierarchical name space with a virtual root.
- Administration of the name space can be delegated along the path starting from the virtual root.
- A DNS server knows a part (a zone) of the global name space and its position within the global name space.
- Name resolution queries can in principle be sent to arbitrary DNS servers. However, it is good practice to use a local DNS server as the primary DNS server.
- Recursive queries cause the queried DNS server to contact other DNS servers as needed in order to obtain a response to the query.
- The original DNS protocol does not provide sufficient security. There is usually no reason to trust DNS responses.

# DNS Labels and Names

- The names (labels) on a certain level of the tree must be unique and may not exceed 63 byte in length. The character set for the labels is historically 7-bit ASCII. Comparisons are done in a case-insensitive manner.
- Labels must begin with a letter and end with a letter or decimal digit. The characters between the first and last character must be letters, digits or hyphens.
- Labels can be concatenated with dots to form paths within the name space. Absolute paths, ending at the virtual root node, end with a trailing dot. All other paths which do not end with a trailing dot are relative paths.
- The overall length of a domain name is limited to 255 bytes.

# DNS Internationalization

- Recent efforts did result in proposals for Internationalized Domain Names in Applications (IDNA) (RFC 5890, RFC 5891, RFC 3492).
- The basic idea is to support internationalized character sets within applications.
- For backward compatibility reasons, internationalized character sets are encoded into 7-bit ASCII representations (ASCII Compatible Encoding, ACE).
- ACE labels are recognized by a so called ACE prefix. The ACE prefix for IDNA is `xn--`.
- A label which contains an encoded internationalized name might for example be the value `xn--de-jg4avhby1noc0d`.

# Section 32: Resource Records

31 Overview and Features

**32 Resource Records**

33 Message Formats

34 Security and Dynamic Updates

35 Creative Usage

# Resource Records

- Resource Records (RRs) hold typed information for a given name.
- Resource records have the following components:
  - The *owner* is the domain name which identifies a resource record.
  - The *type* indicates the kind of information that is stored in a resource record.
  - The *class* indicates the protocol specific name space, normally IN for the Internet.
  - The *time to life* (TTL) defines how many seconds information from a resource record can be stored in a local cache.
  - The data format (RDATA) of a resource records depends on the type of the resource record.

# Resource Record Types

Type	Description
A	IPv4 address
AAAA	IPv6 address
CNAME	Alias for another name (canonical name)
HINFO	Identification of the CPU and the operating system (host info)
TXT	Some arbitrary (ASCII) text
MX	List of mail server (mail exchanger)
NS	Identification of an authoritative server for a domain
PTR	Pointer to another part of the name space
SOA	Start and parameters of a zone (start of zone of authority)
RRSIG	Resource record signature
DNSKEY	Public key associated with a name
DS	Delegation signer resource record
NSEC	Next secure resource resource
SRV	Service record (generalization of the MX record)

# Section 33: Message Formats

31 Overview and Features

32 Resource Records

**33 Message Formats**

34 Security and Dynamic Updates

35 Creative Usage

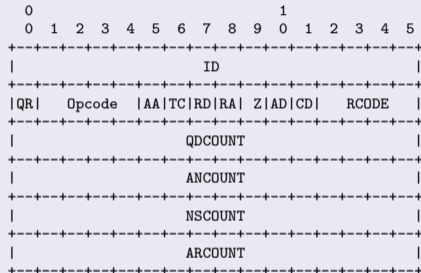
# DNS Message Formats

- A DNS message starts with a protocol header. It indicates which of the following four parts is present and whether the message is a query or a response.
- The header is followed by a list of questions.
- The list of questions is followed by a list of answers (resource records).
- The list of answers is followed by a list of pointers to authorities (also in the form of resource records).
- The list of pointers to authorities is followed by a list of additional information (also in the form of resource records). This list may contain for example A resource records for names in a response to an MX query.



# DNS Message Header

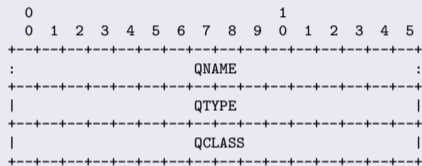
## Header Format



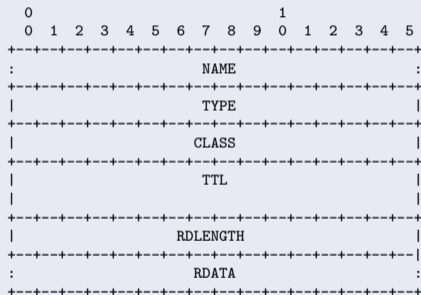
- Simple DNS queries usually use UDP as a transport.
- For larger data transfers (e.g., zone transfers), DNS may utilize TCP.

# DNS Message Formats

## DNS Query Format



## DNS Response Format



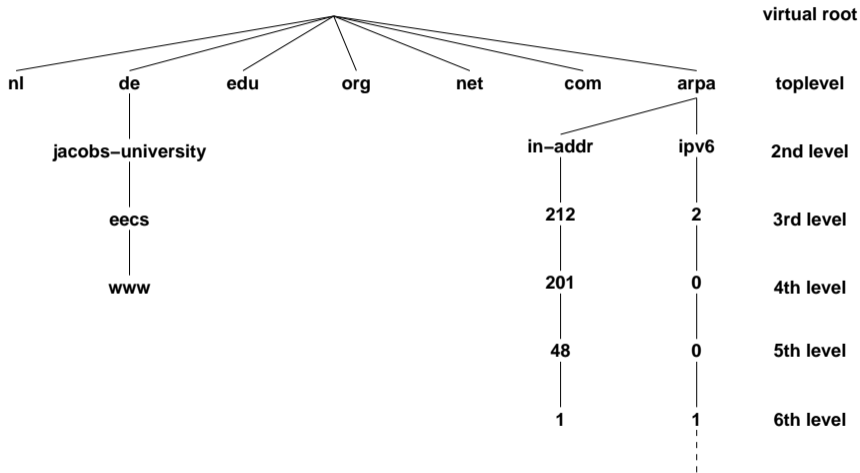
# Resource Record Formats

- An A resource record contains an IPv4 address encoded in 4 bytes in network byte order.
- An AAAA resource record contains an IPv6 address encoded in 16 bytes in network byte order.
- A CNAME resource record contains a character string preceded by the length of the string encoded in the first byte.
- A HINFO resource record contains two character strings, each prefixed with a length byte. The first character string describes the CPU and the second string the operating system.
- A MX resource record contains a 16-bit preference number followed by a character string prefixed with a length bytes which contains the DNS name of a mail exchanger.

# Resource Record Formats

- A NS resource record contains a character string prefixed by a length byte which contains the name of an authoritative DNS server.
- A PTR resource record contains a character string prefixed with a length byte which contains the name of another DNS server. PTR records are used to map IP addresses to names (so called reverse lookups). For an IPv4 address of the form  $d_1.d_2.d_3.d_4$ , a PTR resource record is created for the pseudo domain name  $d_4.d_3.d_2.d_1.in - addr.arpa$ . For an IPv6 address of the form  $h_1h_2h_3h_4 : \dots : h_{13}h_{14}h_{15}h_{16}$ , a PTR resource record is created for the pseudo domain name  $h_{16}.h_{15}.h_{14}.h_{13}.\dots.h_4.h_3.h_2.h_1.ip6.arpa$

# DNS Reverse Trees



# Resource Record Formats

- A SOA resource record contains two character strings, each prefixed by a length byte, and five 32-bit numbers:
  - Name of the DNS server responsible for a zone.
  - Email address of the administrator responsible for the management of the zone.
  - Serial number (SERIAL) (must be incremented whenever the zone database changes).
  - Time which may elapse before cached zone information must be updated (REFRESH).
  - Time after which to retry a failed refresh (RETRY).
  - Time interval after which zone information is considered not current anymore (EXPIRE).
  - Minimum lifetime for resource records (MINIMUM).

# Section 34: Security and Dynamic Updates

31 Overview and Features

32 Resource Records

33 Message Formats

**34 Security and Dynamic Updates**

35 Creative Usage

- DNS security (DNSSEC) provides data integrity and authentication to security aware resolvers and applications through the use of cryptographic digital signatures.
- The Resource Record Signature (RRSIG) resource record stores digital signatures.
- The DNS Public Key (DNSKEY) resource record can be used to store public keys in the DNS.
- The Delegation Signer (DS) resource record simplifies some of the administrative tasks involved in signing delegations across organizational boundaries.
- The Next Secure (NSEC) resource record allows a security-aware resolver to authenticate a negative reply for either name or type non-existence.



# Dynamic DNS Updates

- RFC 2136 / RFC 3007 define a mechanism which allows to dynamically update RRs on name server.
- This is especially useful in environments which use dynamic IP address assignments.
- The payload of a DNS update message contains
  - the zone section,
  - the prerequisite section (supporting conditional updates),
  - the update section, and
  - an additional data section.
- The `nsupdate` command line utility can be used to make manual updates. Some DHCP servers perform automatic updates when they hand out an IP address.

# Section 35: Creative Usage

31 Overview and Features

32 Resource Records

33 Message Formats

34 Security and Dynamic Updates

**35 Creative Usage**

# DNS and Anycasting

- DNS servers often make use of IP anycasting in order to improve availability and performance
  - Several DNS service instances with the same IP address are deployed
  - The IP routing system determines to which service instance a specific DNS request is routed
- Many of the DNS root servers ([a-m].root-servers.org) use anycasts; the number of DNS service instances was reported to be above 600 in October 2016
- Anycasting works well for a simple stateless request / response protocol like DNS, see RFC 7094 and RFC 4786 for further details on IP anycasts

# DNS and Service Load Balancing

- Content Delivery Networks (CDN) sometimes use short lived DNS answers to direct requests to servers close to the requester.
- An underlying assumption is that the recursive resolver used by a host is located close to the host (in terms of network topology).
- This assumption is not generally true, for example, if hosts use generic recursive resolvers like Google's public DNS resolver (8.8.8.8 or 2001:4860:4860::8888), see also <https://www.xkcd.com/1361/>.
- DNS extensions have been defined to allow a recursive resolver to indicate a client subnet in a DNS request so that DNS servers can provide responses that match the location of the host, see RFC 7871 for further details.

# Kaminsky DNS Attack

- Cache poisoning attack ('2008):
  - Cause applications to generate queries for non-existing names such as `aaa.example.net`, `aab.example.net`, etc.
  - Send fake responses quickly, trying to guess the 16-bit query ID number.
  - In the fake responses, include additional records that overwrite A records for lets say `example.net`.
- Counter measure:
  - Updated DNS libraries use random port numbers.
  - An attacker has to guess a 16-bit ID number and in addition the 16-bit port number.
- The real solution is DNSSEC ...

# DNS as DDoS Amplifier

- DNS queries with a spoofed source address can be used to direct responses from open resolvers to a certain attack target; the DNS resolver thus helps to hide (to some extent) the source of the attack.
- Since DNS responses are typically larger than DNS queries, a DNS resolver also acts as an amplifier, turning, for example, 100Mbps query traffic into 1Gbps attack traffic.
- DNS security makes amplification significantly more effective if cryptographic algorithms are used that require relatively long keys.
- It has been shown that elliptic curve algorithms tend to be way more space efficient than traditional RSA algorithms.

# DNS Blacklists

- DNS Blacklists store information about bad behaving hosts.
- Originally used to publish information about sites that originated unsolicited email (spam).
- If the IP address 192.0.2.99 is found guilty to emit spam, a DNS Blacklists at bad.example.com will add the following DNS records:

```
99.2.0.192.bad.example.com    IN  A      127.0.0.2
99.2.0.192.bad.example.com    IN  TXT    "Spam received."
```
- A mail server receiving a connection from 192.0.2.99 may lookup the A record of 99.2.0.192.bad.example.com and if it has the value 127.0.0.2 decline to serve the client.
- For more details, see RFC 5782.

- Kensuke Fukuda has analyzed the DNS traffic generated by middleboxes when they perform reverse lookups on IP addresses that they see (so called DNS backscatter).
- Geoff Huston placed advertisements on web pages that include one-time valid DNS names to drive certain measurements and they found that these one-time DNS names sometimes enjoy additional lookups from very different locations in the network weeks later (which he called stalking).
- Since DNS traffic is often not filtered, people have created many different techniques and tools to tunnel IP traffic over DNS.
- It has been reported that DNS has seen some usage as a command and control channel for malware and botnets.



# References I



P. Mockapetris.

Domain Names - Concepts and Facilities.

RFC 1034, ISI, November 1987.



P. Mockapetris.

Domain Names - Implementation and Specification.

RFC 1035, ISI, November 1987.



J. Klensin.

Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework.

RFC 5890, August 2010.



J. Klensin.

Internationalized Domain Names in Applications (IDNA): Protocol.

RFC 5891, August 2010.



A. Costello.

Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA).

RFC 3492, UC Berkeley, March 2003.



P. Vixie, S. Thomson, Y. Rekhter, and J. Bound.

Dynamic Updates in the Domain Name System (DNS UPDATE).

RFC 2136, ISC, Bellcore, Cisco, DEC, April 1997.

# References II



B. Wellington.

Secure Domain Name System (DNS) Dynamic Update.

RFC 3007, Nominum, November 2000.



R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose.

DNS Security Introduction and Requirements.

RFC 4033, Telematica Instituut, ISC, VeriSign, Colorado State University, NIST, March 2005.



R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose.

Resource Records for the DNS Security Extensions.

RFC 4034, Telematica Instituut, ISC, VeriSign, Colorado State University, NIST, March 2005.



R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose.

Protocol Modifications for the DNS Security Extensions.

RFC 4035, Telematica Instituut, ISC, VeriSign, Colorado State University, NIST, March 2005.



J. Levine.

DNS Blacklists and Whitelists.

RFC 5782, Taughannock Networks, February 2010.



D. Schneider.

Fresh Phish.

*IEEE Spectrum*, 45(10):29–32, October 2008.

# Part 9: Augmented Backus Naur Form (ABNF)

36 Basics, Rule Names, Terminal Symbols

37 Operators

38 Core Definitions

39 ABNF in ABNF

# Section 36: Basics, Rule Names, Terminal Symbols

36 Basics, Rule Names, Terminal Symbols

37 Operators

38 Core Definitions

39 ABNF in ABNF

- The Augmented Backus Naur Form (ABNF) defined in RFC 5234 can be used to formally specify textual protocol messages.
- An ABNF definition consists of a set of rules (sometimes also called productions).
- Every rule has a name followed by an assignment operator followed by an expression consisting of terminal symbols and operators.
- The end of a rule is marked by the end of the line or by a comment. Comments start with the comment symbol ; (semicolon) and continue to the end of the line.
- ABNF does not define a module concept or an import/export mechanism.

# Rule Names and Terminal Symbols

- The name of a rule must start with an alphabetic character followed by a combination of alphabets, digits and hyphens. The case of a rule name is not significant.
- Terminal symbols are non-negative numbers. The basis of these numbers can be binary (b), decimal (d) or hexadecimal (x). Multiple values can be concatenated by using the dot . as a value concatenation operator. It is also possible to define ranges of consecutive values by using the hyphen - as a value range operator.
- Terminal symbols can also be defined by using literal text strings containing US ASCII characters enclosed in double quotes. Note that these literal text strings are case-insensitive.

# Simple ABNF Examples

```
CR      = %d13           ; ASCII carriage return code in decimal
CRLF   = %d13.10        ; ASCII carriage return and linefeed code sequence
DIGIT  = %x30-39        ; ASCII digits (0 - 9)
ABA    = "aba"          ; ASCII string "aba" or "ABA" or "Aba" or ...
abba   = %x61.62.62.61 ; ASCII string "abba"
```

# ABFN Case-Sensitive String Support

- RFC 7405 adds support for case-sensitive strings.
  - `%s` = case-sensitive string
  - `%i` = case-insensitive string

- Examples:

```
r1 = "aBc"
```

```
r2 = %i"aBc"
```

```
r3 = %s"aBc"
```

The rules `r1` and `r2` are equivalent and they will both match “abc”, “Abc”, “aBc”, “abC”, “ABc”, “aBC”, “AbC”, and “ABC”. The rule `r3` matches only “aBc”.



# Section 37: Operators

36 Basics, Rule Names, Terminal Symbols

**37 Operators**

38 Core Definitions

39 ABNF in ABNF

# ABFN Operators

- Concatenation
  - Concatenation operator symbol is the empty word
  - Example: `abba = %x61 %x62 %x62 %x61`
- Alternatives
  - Alternatives operator symbol is the forward slash /
  - Example: `aorb = %x61 / %x62`
  - Incremental alternatives assignment operator `=/` can be used for long lists of alternatives
- Grouping
  - Expressions can be grouped using parenthesis
  - A grouped expression is treated as a single element
  - Example: `abba = %x61 (%x62 %x62) %x61`

- Repetitions
  - The repetitions operator has the format  $n*m$  where  $n$  and  $m$  are optional decimal values
  - The value of  $n$  indicates the minimum number of repetitions (defaults to 0 if not present)
  - The value  $m$  indicates the maximum number of repetitions (defaults to infinity if not present)
  - The format  $*$  indicates 0 or more repetitions
  - Example: `abba = %x61 2 %x62 %x61`
- Optional
  - Square brackets enclose an optional element
  - Example: `[ab]` ; equivalent to `*1(ab)`

# Section 38: Core Definitions

36 Basics, Rule Names, Terminal Symbols

37 Operators

**38 Core Definitions**

39 ABNF in ABNF

# ABNF Core Definitions

ALPHA	=	%x41-5A / %x61-7A	; A-Z / a-z
BIT	=	"0" / "1"	
CHAR	=	%x01-7F	; any 7-bit US-ASCII character, ; excluding NUL
CR	=	%x0D	; carriage return
CRLF	=	CR LF	; Internet standard newline
CTL	=	%x00-1F / %x7F	; controls
DIGIT	=	%x30-39	; 0-9
DQUOTE	=	%x22	; " (Double Quote)
HEXDIG	=	DIGIT / "A" / "B" / "C" / "D" / "E" / "F"	
HTAB	=	%x09	; horizontal tab
LF	=	%x0A	; linefeed
LWSP	=	*(WSP / CRLF WSP)	; linear white space (past newline)
OCTET	=	%x00-FF	; 8 bits of data
SP	=	%x20	; space
VCHAR	=	%x21-7E	; visible (printing) characters
WSP	=	SP / HTAB	; White space

# Section 39: ABNF in ABNF

36 Basics, Rule Names, Terminal Symbols

37 Operators

38 Core Definitions

**39 ABNF in ABNF**

# ABNF in ABNF

```
rulelist      = 1*( rule / (*c-wsp c-nl) )

rule         = rulename defined-as elements c-nl
              ; continues if next line starts with white space

rulename     = ALPHA *(ALPHA / DIGIT / "-")

defined-as   = *c-wsp ("=" / "=/") *c-wsp
              ; basic rules definition and incremental alternatives

elements     = alternation *c-wsp

c-wsp       = WSP / (c-nl WSP)

c-nl        = comment / CRLF
              ; comment or newline

comment     = ";" *(WSP / VCHAR) CRLF
```

# ABNF in ABNF

```
alternation    = concatenation
               >(*c-wsp "/" *c-wsp concatenation)

concatenation  = repetition *(1*c-wsp repetition)

repetition    = [repeat] element

repeat        = 1*DIGIT / (*DIGIT "*" *DIGIT)

element       = rulename / group / option /
                char-val / num-val / prose-val

group         = "(" *c-wsp alternation *c-wsp ")"

option        = "[" *c-wsp alternation *c-wsp "]"
```



# ABNF in ABNF

char-val = DQUOTE \*(%x20-21 / %x23-7E) DQUOTE  
; quoted string of SP and VCHAR without DQUOTE

num-val = "%" (bin-val / dec-val / hex-val)

bin-val = "b" 1\*BIT [ 1\*("." 1\*BIT) / ("-" 1\*BIT) ]  
; series of concatenated bit values  
; or single ONEOF range

dec-val = "d" 1\*DIGIT [ 1\*("." 1\*DIGIT) / ("-" 1\*DIGIT) ]

hex-val = "x" 1\*HEXDIG [ 1\*("." 1\*HEXDIG) / ("-" 1\*HEXDIG) ]

prose-val = "<" \*(%x20-3D / %x3F-7E) ">"  
; bracketed string of SP and VCHAR without angles  
; prose description, to be used as last resort

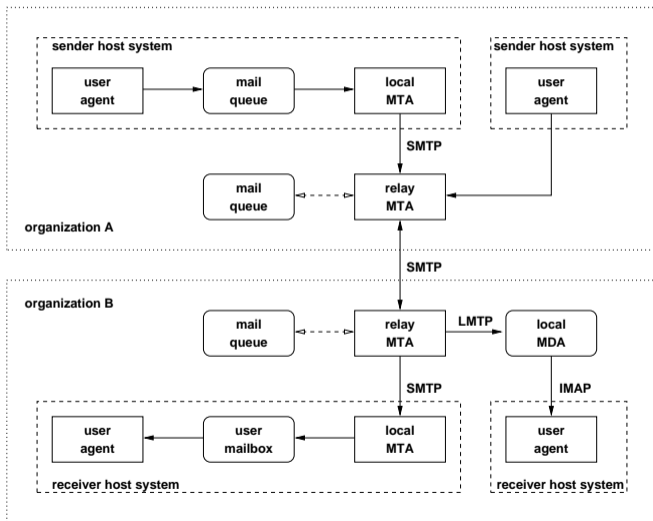
# Part 10: Electronic Mail (SMTP, IMAP)

- 40 Components and Terminology
- 41 Simple Mail Transfer Protocol (SMTP)
- 42 Multipurpose Internet Mail Extensions (MIME)
- 43 Internet Message Access Protocol (IMAP)
- 44 Filtering of Messages (SIEVE)
- 45 DomainKeys Identified Mail (DKIM)

# Section 40: Components and Terminology

- 40 Components and Terminology
- 41 Simple Mail Transfer Protocol (SMTP)
- 42 Multipurpose Internet Mail Extensions (MIME)
- 43 Internet Message Access Protocol (IMAP)
- 44 Filtering of Messages (SIEVE)
- 45 DomainKeys Identified Mail (DKIM)

# Components Involved in Electronic Mail



- Mail User Agent (MUA) - the source or targets of electronic mail
- Mail Transfer Agent (MTA) - server and clients providing mail transport service
- Mail Delivery Agent (MDA) - delivers mail messages to the receiver's mail box
- Store-and-Forward Principle - mail messages are stored and then forwarded to another system; responsibility for a message is transferred once it is stored again
- Envelop vs. Header vs. Body - mail messages consist of a header and a body; the transfer of mail message is controlled by the envelop (which might be different from the header)

# Section 41: Simple Mail Transfer Protocol (SMTP)

- 40 Components and Terminology
- 41 Simple Mail Transfer Protocol (SMTP)**
- 42 Multipurpose Internet Mail Extensions (MIME)
- 43 Internet Message Access Protocol (IMAP)
- 44 Filtering of Messages (SIEVE)
- 45 DomainKeys Identified Mail (DKIM)

# Simple Mail Transfer Protocol (SMTP)

- Defined in RFC 5321 (originally RFC 821)
- Textual client/server protocol running over TCP (default port 25)
- Small set of commands to be executed by an SMTP server
- Supports multiple mail transactions over a single transport layer connection
- Server responds with structured response codes
- Message formats specified in ABNF

# SMTP Commands

Command	Description
HELO	Identify clients to a SMTP server (HELLO)
EHLO	Extended identification (EXTENDED HELLO)
MAIL	Initiate a mail transaction (MAIL)
RCPT	Identify an individual recipient (RECIPIENT)
DATA	Transfer of mail message (DATA)
RSET	Aborting current mail transaction (RESET)
VRFY	Verify an email address (VERIFY)
EXPN	Expand a mailing list address (EXPAND)
HELP	Provide help about SMTP commands (HELP)
NOOP	No operation, has no effect (NOOP)
QUIT	Ask server to close connection (QUIT)



# SMTP in ABNF (excerpt)

```
helo = "HELO" SP Domain CRLF
ehlo = "EHLO" SP Domain CRLF
mail = "MAIL FROM:" ("<>" / Reverse-Path) [SP Mail-Parameters] CRLF
rcpt = "RCPT TO:" ("
```

# Theory of 3 Digit Reply Codes

- The first digit denotes whether the response is good, bad or incomplete.
  - 1yz Positive Preliminary reply
  - 2yz Positive Completion reply
  - 3yz Positive Intermediate reply
  - 4yz Transient Negative Completion reply
  - 5yz Permanent Negative Completion reply
- The second digit encodes responses in specific categories.
- The third digit gives a finer gradation of meaning in each category specified by the second digit.

# Internet Message Format

- The format of Internet messages is defined in RFC 5322.
- Most important ABNF productions and messages fields:

```
fields          = *(trace *resent-field) *regular-field
```

```
resent-field    = resent-date / resent-from / resent-sender
```

```
resent-field    =/ resent-to / resent-cc / resent-bcc
```

```
resent-field    =/ resent-msg-id
```

```
regular-field   = orig-date / from / sender
```

```
regular-field   =/ reply-to / to / cc / bcc
```

```
regular-field   =/ message-id / in-reply-to / references
```

```
regular-field   =/ subject / comments / keywords
```

- Note that fields such as to or cc may be different from the actual addresses used by SMTP commands (the envelope).

# Originator Fields

- The `From:` field specifies the author(s) of the message.

```
from = "From:" mailbox-list CRLF
```

- The `Sender:` field specifies the mailbox of the sender in cases where the actual sender is not the author (e.g., a secretary).

```
sender = "Sender:" mailbox CRLF
```

- The `Reply-To:` field indicates the mailbox(es) to which the author of the message suggests that replies be sent.

```
reply-to = "Reply-To:" address-list CRLF
```

# Destination Address Fields

- The `To:` field contains the address(es) of the primary recipient(s) of the message.

`to = "To:" address-list CRLF`

- The `Cc:` field (Carbon Copy) contains the addresses of others who are to receive the message, though the content of the message may not be directed at them.

`cc = "Cc:" address-list CRLF`

- The `Bcc:` field (Blind Carbon Copy) contains addresses of recipients of the message whose addresses are not to be revealed to other recipients of the message.

`bcc = "Bcc:" (address-list / [CFWS]) CRLF`

# Identification and Origination Date Fields

- The Message-ID: field provides a unique message identifier that refers to a particular version of a particular message.

```
message-id = "Message-ID:" msg-id CRLF
```

- The In-Reply-To: field will contain the contents of the Message-ID: field of the message to which this one is a reply.

```
in-reply-to = "In-Reply-To:" 1*msg-id CRLF
```

- The References: field will contain the contents of the parent's References: field (if any) followed by the contents of the parent's Message-ID: field (if any).

```
references = "References:" 1*msg-id CRLF
```

# Informational Fields

- The Subject: field contains a short string identifying the topic of the message.

```
subject = "Subject:" unstructured CRLF
```

- The Comments: field contains any additional comments on the text of the body of the message.

```
comments = "Comments:" unstructured CRLF
```

- The Keywords: field contains a comma-separated list of important words and phrases that might be useful for the recipient.

```
keywords = "Keywords:" phrase *(", " phrase) CRLF
```

# Trace Fields

- The `Received:` field contains a (possibly empty) list of name/value pairs followed by a semicolon and a date-time specification. The first item of the name/value pair is defined by item-name, and the second item is either an addr-spec, an atom, a domain, or a msg-id.

```
received = "Received:" name-val-list ";" date-time CRLF
```

- The `Return-Path:` field contains an email address to which messages indicating non-delivery or other mail system failures are to be sent.

```
return = "Return-Path:" path CRLF
```

- A message may have multiple `received` fields and the `return` field is optional

```
trace = [return] 1*received
```



# Resend Fields

- Resent fields are used to identify a message as having been reintroduced into the transport system by a user.
- Resent fields make the message appear to the final recipient as if it were sent directly by the original sender, with all of the original fields remaining the same.
- Each set of resent fields correspond to a particular resending event.

```
resent-date = "Resent-Date:" date-time CRLF
resent-from = "Resent-From:" mailbox-list CRLF
resent-sender = "Resent-Sender:" mailbox CRLF
resent-to = "Resent-To:" address-list CRLF
resent-cc = "Resent-Cc:" address-list CRLF
resent-bcc = "Resent-Bcc:" (address-list / [CFWS]) CRLF
resent-msg-id = "Resent-Message-ID:" msg-id CRLF
```

# Internet Message Example

Date: Tue, 1 Apr 1997 09:06:31 -0800 (PST)  
From: coyote@desert.example.org  
To: roadrunner@acme.example.com  
Subject: I have a present for you

Look, I'm sorry about the whole anvil thing, and I really didn't mean to try and drop it on you from the top of the cliff. I want to try to make it up to you. I've got some great birdseed over here at my place--top of the line stuff--and if you come by, I'll have it all wrapped up for you. I'm really sorry for all the problems I've caused for you over the years, but I know we can work this out.

--

Wile E. Coyote "Super Genius" coyote@desert.example.org

# Section 42: Multipurpose Internet Mail Extensions (MIME)

- 40 Components and Terminology
- 41 Simple Mail Transfer Protocol (SMTP)
- 42 Multipurpose Internet Mail Extensions (MIME)**
- 43 Internet Message Access Protocol (IMAP)
- 44 Filtering of Messages (SIEVE)
- 45 DomainKeys Identified Mail (DKIM)

# Multipurpose Internet Mail Extensions

- The Multipurpose Internet Mail Extensions (MIME) defines conventions to
  - support multiple different character sets;
  - support different media types;
  - support messages containing multiple parts;
  - encode content compatible with RFC 5321 and RFC 5322.
- The set of media types and identified characters sets is extensible.
- MIME is widely implemented and not only used for Internet mail messages.

# MIME Header Fields

- The `MIME-Version:` field declares the version of the Internet message body format standard in use.

```
version = "MIME-Version:" 1*DIGIT "." 1*DIGIT CRLF
```

- The `Content-Type:` field specifies the media type and subtype of data in the body.

```
content = "Content-Type:" type "/" subtype *(";" parameter)
```

- The `Content-Transfer-Encoding:` field specifies the encoding transformation that was applied to the body and the domain of the result.

```
encoding = "Content-Transfer-Encoding:" mechanism
```

# MIME Header Fields

- The optional Content-ID: field allows one body to make reference to another.

```
id = "Content-ID:" msg-id
```

- The optional Content-Description: field associates some descriptive information with a given body.

```
description = "Content-Description" *text
```

# MIME Media Types

- Five discrete top-level media types (RFC 2046):
  - text
  - image
  - audio
  - video
  - application
- Two composite top-level media types (RFC 2046):
  - multipart
  - message
- Some media types have additional parameters (e.g., the character set).
- The Internet Assigned Numbers Authority (IANA) maintains a list of registered media types and subtypes.

# MIME Boundaries

- Multipart documents consists of several entities which are separated by a boundary delimiter line.
- After its boundary delimiter line, each body part then consists of a header area, a blank line, and a body area.
- The boundary is established through a parameter of the Content-Type: header field of the message.

```
Content-Type: multipart/mixed; boundary="gc0pJq0M:08jU534c0p"
```

- The boundary delimiter consists of two dashes followed by the established boundary followed by optional white space and the end of the line. The last boundary delimiter contains two hyphens following the boundary.

```
--gc0pJq0M:08jU534c0p
```

- The boundary delimiter must chosen so as to guarantee that there is no clash with the content.



# MIME Example

```
From: Nathaniel Borenstein <nsb@bellcore.com>  
To: Ned Freed <ned@innosoft.com>  
Date: Sun, 21 Mar 1993 23:56:48 -0800 (PST)  
Subject: Sample message  
MIME-Version: 1.0  
Content-type: multipart/mixed; boundary="simple boundary"
```

This is the preamble. It is to be ignored.

--simple boundary

This is implicitly typed plain US-ASCII text.  
It does NOT end with a linebreak.

--simple boundary

Content-type: text/plain; charset=us-ascii

This is explicitly typed plain US-ASCII text ending with a linebreak.

--simple boundary--

This is the epilogue. It is also to be ignored.

# Base64 Encoding

- Idea: Represent three input bytes (24 bit) using four characters taken from a 6-bit alphabet.
- The resulting character sequence is broken into lines such that no line is longer than 76 characters if the base64 encoding is used with MIME.
- If the input text has a length which is not a multiple of three, then the special character = is appended to indicate the number of fill bytes.
- Base64 encoded data is difficult to read by humans without tools (which however are trivial to write).

# Base64 Character Set

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

# Base64 Example

```
CkRhdGU6IFR1ZSwgMSBBcHIgMTk5NyAwOTowNjozMScAtMDgwMCAoUFNlUkQpGcm9t
OiBjb3lvdGVAZGVzZXJ0LmV4YW1wbGUub3JnClRvOiByb2FkcjVubmVvYQGFjbWUu
ZXhhbXBsZS5jb20KU3ViamVjdDogSSBoYXZlIGVzZGh1c2VudCBmb3IgeW91CgpM
b29rLCBjJ20gc29ycnkgYWJvdXQgdGh1IHdob2x1IGFudmlsIHRoaW5nLCBhbmQg
SSByZWZsbnkKZGlkaW50IG1lYW4gdG8gdHJ5IGFuZCBkcm9wIG10IG9uIHlvdSBm
cm9tIHRoZSB0b3Agb2YgdGh1CmNsaWZmLiAgSSB3YW50IHRvIHRyeSB0byBtYWtl
IG10IHVwIHRvIHlvdS4gIEkndmUgZ290IHNvbWUKZ3JlYXQgYmlyZHNlZWQgb3Zl
ciBoZXJlIGF0IG15IHBSYWNlLS10b3Agb2YgdGh1IGxpbmUKc3R1ZmYtLWZlZCBp
ZiB5b3UgY29tZSBieSwgSSdsbCB0YXZlIG10IGFsbCB3cmFwcGVkIHVwCmZvciB5
b3UuICBjJ20gc29ycnkgYWNvcnJ5IGZvciBhbGwgdGh1IHByb2JsZW1zIEkndmUg
Y2F1c2Vkc29ycnkgYWNvcnJ5IGZvciB5b3UgY29tZSBieSwgSSdsbCB0YXZlIG10
IHdvcmsgdGhpcyBvdXQuCi0tCltpbGUgRS4gQ295b3RlICAgI1N1cGVyIEdlbmll
cyIgeW91Cm9tIHRoZSB0b3Agb2YgdGh1c2VudCBmb3IgeW91Cm9tIHRoZSB0b3Ag
b29rLCBjJ20gc29ycnkgYWNvcnJ5IGZvciBhbGwgdGh1IHByb2JsZW1zIEkndmUg
```

- What does this text mean? (Note that this example uses a non-standard line length to fit on a slide.)

# Quoted-Printable Encoding

- Idea: Escape all characters which are not printable characters in the US-ASCII character set.
- The escape character is = followed by a two digit hexadecimal representation of the octet's value (in uppercase).
- Relatively complex rules ensure that
  - line breaks in the original input are preserved (so called hard line breaks);
  - line breaks are added to ensure that encoded lines be no more than 76 characters long (so called soft line breaks).
- The encoding is intended for data that largely consists of printable characters in the US-ASCII character set.

# Section 43: Internet Message Access Protocol (IMAP)

- 40 Components and Terminology
- 41 Simple Mail Transfer Protocol (SMTP)
- 42 Multipurpose Internet Mail Extensions (MIME)
- 43 Internet Message Access Protocol (IMAP)**
- 44 Filtering of Messages (SIEVE)
- 45 DomainKeys Identified Mail (DKIM)

# Internet Message Access Protocol (IMAP)

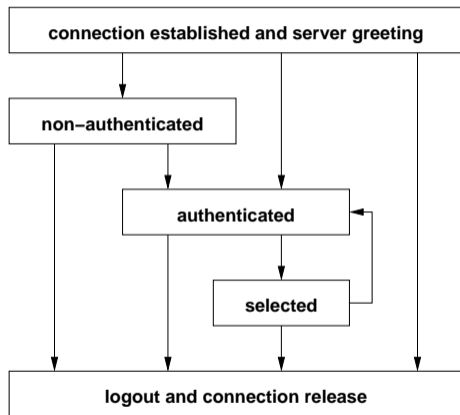
- The Internet Message Access Protocol (IMAP) defined in RFC 3501 allows a client to access and manipulate electronic mail messages stored on a server.
- Messages are stored in mailboxes (message folders) which are identified by a mailbox name.
- IMAP runs over TCP with the well-known port number 143.
- IMAP users are typically authenticated using passwords (transmitted in cleartext over TCP).
- It is strongly suggested to use the Transport Layer Security (TLS) option to encrypt authentication exchanges and message transfers.

# IMAP Message Identification

- Messages in a mailbox are identified by numbers:
  - The *unique identifier* identifies a message in a mailbox independent of its position. Unique identifiers should remain persistent across IMAP sessions.
  - The *message sequence number* is the relative position of a message in a mailbox. The first position in a mailbox is 1.
- Persistency of the unique identifier can only be achieved to a certain extent and implementation therefore must cope with non-persistent unique identifiers.



# IMAP States



- The state determines the set of applicable commands.

# IMAP Commands

- Commands applicable in all states:

CAPABILITY	Reports the server's capabilities
NOOP	Empty command (may be used to trigger status updates)
LOGOUT	Terminate the IMAP session

- Commands applicable in the *non-authenticated* state:

AUTHENTICATE	Indicates an authentication mechanism to the server
LOGIN	Trivial authentication with a cleartext password
STARTTLS	Start TLS negotiation to protect the channel

- Note that the completion of the STARTTLS command can change the capabilities announced by the server.

- Commands applicable in the *authenticated* state:

SELECT	Select an existing mailbox (read-write)
EXAMINE	Select an existing mailbox (read-only)
CREATE	Create a new mailbox
DELETE	Delete an existing mailbox
RENAME	Rename an existing mailbox
SUBSCRIBE	Add a mailbox to the server's list of active mailboxes
UNSUBSCRIBE	Remove a mailbox from the server's list of active mailboxes
LIST	List all existing mailboxes
LSUB	List all active mailboxes
STATUS	Retrieve the status of a mailbox
APPEND	Append a message to a mailbox

- The SELECT and EXAMINE commands cause a transition into the *selected* state.

- Commands applicable in the *selected* state:
  - CHECK Create a copy of the current mailbox (checkpoint)
  - CLOSE Close the current mailbox and leave state
  - EXPUNGE Expunge all messages marked as deleted
  - SEARCH Search for messages matching given criteria
  - FETCH Fetch data of a message in the current mailbox
  - STORE Store data as a message in the current mailbox
  - COPY Copy a message to the end of the specified mailbox
  - UID Execute a command using unique identifiers
- The CLOSE command causes a transition back into the *authenticated* state.

# IMAP Tagging

- IMAP supports asynchronous, concurrent operations. A client can send multiple commands which the server will execute asynchronously.
- A client tags commands such that responses returned by the server can be related to previously sent commands.
- Server responses
  - that do not indicate command completion are prefixed with the token \*;
  - that request additional information to complete a command are prefixed with the token +;
  - that communicate the successful or unsuccessful completion of a command are prefixed with the tag.

# IMAP Commands in ABNF

```
tag          = 1*<any ATOM_CHAR except "+">

command      = tag SPACE (command_any / command_auth /
                        command_nonauth / command_select) CRLF
              ;; Modal based on state

command_any  = "CAPABILITY" / "LOGOUT" / "NOOP" / x_command
              ;; Valid in all states

command_auth = append / create / delete / examine / list / lsub /
              rename / select / status / subscribe / unsubscribe
              ;; Valid only in Authenticated or Selected state

command_nonauth = login / authenticate
                 ;; Valid only when in Non-Authenticated state

command_select = "CHECK" / "CLOSE" / "EXPUNGE" /
                 copy / fetch / store / uid / search
                 ;; Valid only when in Selected state
```

# IMAP Responses in ABNF

```
response      = *(continue_req / response_data) response_done
continue_req  = "+" SPACE (resp_text / base64)
response_data = "*" SPACE (resp_cond_state / resp_cond_bye /
                           mailbox_data / message_data / capability_data) CRLF
response_done = response_tagged / response_fatal
response_fatal = "*" SPACE resp_cond_bye CRLF
               ;; Server closes connection immediately
response_tagged = tag SPACE resp_cond_state CRLF
```

# Section 44: Filtering of Messages (SIEVE)

- 40 Components and Terminology
- 41 Simple Mail Transfer Protocol (SMTP)
- 42 Multipurpose Internet Mail Extensions (MIME)
- 43 Internet Message Access Protocol (IMAP)
- 44 Filtering of Messages (SIEVE)**
- 45 DomainKeys Identified Mail (DKIM)



# SIEVE Filtering Language

- The SIEVE language defined in RFC 5228 can be used to filter messages at time of final delivery.
- The language can be implemented either on the mail client or the mail server.
- SIEVE is extensible, simple, and independent of the access protocol, mail architecture, and operating system.
- The SIEVE language can be safely executed on servers (such as IMAP servers) as it has no variables, loops, or ability to shell out to external programs.

- SIEVE scripts are sequences of commands.
  - An *action command* is an identifier followed by zero or more arguments, terminated by a semicolon. Action commands do not take tests or blocks as arguments.
  - A *control command* is similar, but it takes a test as an argument, and ends with a block instead of a semicolon.
  - A *test command* is used as part of a control command. It is used to specify whether or not the block of code given to the control command is executed.
- The empty SIEVE script keeps the message (implicit keep).

# SIEVE in ABNF

```
argument = string-list / number / tag
arguments = *argument [test / test-list]
```

```
block = "{" commands "}"
```

```
command = identifier arguments ( ";" / block )
commands = *command
```

```
start = commands
```

```
string = quoted-string / multi-line
string-list = "[" string *("," string) "]" / string
;; if there is only a single string, the brackets are optional
```

```
test = identifier arguments
test-list = "(" test *("," test) ")"
```

# SIEVE Example

```
# Declare any optional features or extension used by the script
require ["fileinto", "reject"];

# Reject any large messages (note that the four leading dots get
# "stuffed" to three)

if size :over 1M {
    reject text:
    Please do not send me large attachments.
    Put your file on a server and send me the URL.
    Thank you.
    .... Fred
    .
    ;
        stop;
}
```

# SIEVE Example (cont.)

```
# Move messages from IETF filter discussion list to filter folder
if header :is "Sender" "owner-ietf-mta-filters@imc.org" {
    fileinto "filter"; # move to "filter" folder
    stop;
}
#
# Keep all messages to or from people in my company
#
elsif address :domain :is ["From", "To"] "example.com" {
    keep;          # keep in "In" folder
    stop;
}
```

# SIEVE Example (cont.)

```
# Try and catch unsolicited email.  If a message is not to me,
# or it contains a subject known to be spam, file it away.
#
if anyof (not address :all :contains
          ["To", "Cc", "Bcc"] "me@example.com",
          header :matches "subject"
          ["*make*money*fast*", "*university*dipl*mas*"]) {
  # If message header does not contain my address,
  # it's from a list.
  fileinto "spam";  # move to "spam" folder
} else {
  # Move all other (non-company) mail to "personal" folder.
  fileinto "personal";
}
```

# Section 45: DomainKeys Identified Mail (DKIM)

- 40 Components and Terminology
- 41 Simple Mail Transfer Protocol (SMTP)
- 42 Multipurpose Internet Mail Extensions (MIME)
- 43 Internet Message Access Protocol (IMAP)
- 44 Filtering of Messages (SIEVE)
- 45 DomainKeys Identified Mail (DKIM)**

# DomainKeys Identified Mail (DKIM)

## Goals

Intended to allow good senders to prove that they did send a particular message, and to prevent forgers from masquerading as good senders (if those senders sign all outgoing mail).

## Non Goals

DKIM does not protect of content of messages (encryption), provides no protection after delivery and no protection against replay.

## History

- Yahoo! introduced DomainKeys
- Cisco introduced Identified Internet Mail
- Both proposals got merged and resulted in DKIM



# DKIM Features

- DKIM separates the identity of the signer of a message from the identity of the purported authors of a message.
- Message signatures are carried as message header fields and not as part of the message body.
- Signature verification failure does not force rejection of the message.
- DKIM keys can be fetched via DNS queries but other key fetching protocols are possible as well.
- No trusted third party and public key infrastructure required.
- Designed to support incremental deployment.

# Example Signature

```
DKIM-Signature: a=rsa-sha256; d=example.net; s=brisbane;  
  c=simple; q=dns/txt; i=@eng.example.net;  
  t=1117574938; x=1118006938;  
  h=from:to:subject:date;  
  z=From:foo@example.net|To:joe@example.com|  
    Subject:demo=20run|Date:July=205,=202005=203:44:08=20PM=20-0700;  
  bh=MTIzNDU2Nzg5MDEyMzQ1Njc4OTAxMjMONTY3ODkwMTI=;  
  b=dzdVv0fAKCdLXdJ0c9G2q8LoXSlEniSbav+yuU4zGeeruD00lszZ  
    VoG4ZHRNiYzR
```

- The DKIM-Signature header contains several key/value pairs holding information about the signer, the crypto algorithm and parameters used, and the signature itself.
- The bh contains the body hash of the canonicalized body of the message.
- The b contains the actual DKIM signature.

# DKIM Keys in DNS TXT Records

- DKIM keys can be stored in DNS TXT records.
- For the signature domain `d=example.net` and the signer `s=brisbane`, a DNS query is sent to retrieve the TXT record under `brisbane._domainkey.example.net`.
  - The security of this lookup and hence the trustworthiness of the key depends on the security of the DNS.
- + Easy to deploy without public key infrastructures and very scalable key lookup.

# Example: Step #1

From: Joe SixPack <joe@football.example.com>  
To: Suzie Q <suzie@shopping.example.net>  
Subject: Is dinner ready?  
Date: Fri, 11 Jul 2003 21:00:37 -0700 (PDT)  
Message-ID: <20030712040037.46341.5F8J@football.example.com>

Hi.

We lost the game. Are you hungry yet?

Joe.

# Example: Step #2

```
DKIM-Signature: v=1; a=rsa-sha256; s=brisbane; d=example.com;  
c=simple/simple; q=dns/txt; i=joe@football.example.com;  
h=Received : From : To : Subject : Date : Message-ID;  
bh=2jUSOH9NhtVGCQWnr9BrIAPreKQj06Sn7XIkfJV0zv8=;  
b=AuUoFEfDxDkH1LXSZEpZj79LICEps6eda7W3deTVF0k4yAUoqOB  
4nujc7YopdG5dWLSdNg6xNAZp0Pr+kHxt1IrE+NahM6L/LbvaHut  
KVdkLLkpVaVVQPzeRDI009S02I15Lu7rDNH6mZckBdrIx0orEtZV  
4bmp/YzhwvcubU4=;
```

```
Received: from client1.football.example.com [192.0.2.1]  
by submitserver.example.com with SUBMISSION;  
Fri, 11 Jul 2003 21:01:54 -0700 (PDT)  
From: Joe SixPack <joe@football.example.com>  
To: Suzie Q <suzie@shopping.example.net>  
Subject: Is dinner ready?  
Date: Fri, 11 Jul 2003 21:00:37 -0700 (PDT)  
Message-ID: <20030712040037.46341.5F8J@football.example.com>
```

Hi.

We lost the game. Are you hungry yet?

# Example: Step #3

```
X-Authentication-Results: shopping.example.net
  header.from=joe@football.example.com; dkim=pass
Received: from mout23.football.example.com (192.168.1.1)
  by shopping.example.net with SMTP;
  Fri, 11 Jul 2003 21:01:59 -0700 (PDT)
DKIM-Signature: v=1; a=rsa-sha256; s=brisbane; d=example.com;
  c=simple/simple; q=dns/txt; i=joe@football.example.com;
  h=Received : From : To : Subject : Date : Message-ID;
  bh=2jUSOH9NhtVGCQWnr9BrIAPreKQj06Sn7XIkfJV0zv8=;
  b=AuUoFEfDxTDkH1LXSZEpZj79LICEps6eda7W3deTVF0k4yAUoqOB
  4nujc7YopdG5dWLSdNg6xNAZpOPr+kHxt1IrE+NahM6L/LbvaHut
  KVdkLLkpVaVVPzeRDI009S02I15Lu7rDNH6mZckBdrIxOorEtZV
  4bmp/YzhwvcubU4=;
Received: from client1.football.example.com [192.0.2.1]
  by submitserver.example.com with SUBMISSION;
  Fri, 11 Jul 2003 21:01:54 -0700 (PDT)
From: Joe SixPack <joe@football.example.com>
To: Suzie Q <suzie@shopping.example.net>
Subject: Is dinner ready?
Date: Fri, 11 Jul 2003 21:00:37 -0700 (PDT)
Message-ID: <20030712040037.46341.5F8J@football.example.com>
```

Hi.

We lost the game. Are you hungry yet?

Joe.

# References I



R. Housley.

Cryptographic Message Syntax.

RFC 5652, Vigil Security, September 2009.



J. Klensin.

Simple Mail Transfer Protocol.

RFC 5321, October 2008.



P. Resnick.

Internet Message Format.

RFC 5322, QUALCOMM Incorporated, October 2008.



N. Freed and N. Borenstein.

Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies.

RFC 2045, Innosoft, First Virtual, November 1996.



N. Freed and N. Borenstein.

Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types.

RFC 2046, Innosoft, First Virtual, November 1996.



M. Crispin.

Internet Message Access Protocol -Version 4rev1.

RFC 3501, University of Washington, March 2003.

# References II



P. Guenther and T. Showalter.

Sieve: A Mail Filtering Language.

RFC 5228, Sendmail Inc., Mirapoint Inc., January 2008.



W. Segmuller and B. Leiba.

Sieve Email Filtering: Relational Extensions.

RFC 5231, IBM T.J. Watson Research Center, January 2008.



C. Daboo.

Sieve Email Filtering: Spamtest and Virustest Extensions.

RFC 5235, January 2008.



J. Degener.

Sieve Extension: Copying Without Side Effects.

RFC 3894, Sendmail, October 2004.



S. Josefsson.

The Base16, Base32, and Base64 Data Encodings.

RFC 4648, SJD, October 2006.



J. Callas, L. Donnerhacker, H. Finney, D. Shaw, and R. Thayer.

OpenPGP Message Format.

RFC 4880, PGP Corporation, IKS GmbH, November 2007.



# References III



B. Ramsdell.

Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification.  
RFC 3851, Sendmail, Inc., July 2004.



E. Allman, J. Callas, M. Delany, M. Libbey, J. Fenton, and M. Thomas.

DomainKeys Identified Mail (DKIM) Signatures.

RFC 4871, Sendmail Inc., PGP Corporation, Yahoo! Inc., Cisco Systems Inc., May 2007.



W. Stallings.

Comprehensive Internet E-Mail Security.

*The Internet Protocol Journal*, 19(3), November 2016.

# Part 11: Hypertext Transfer Protocol (HTTP)

46 Overview

47 URLs, URNs, URIs, IRIs

48 HTTP 1.1 Methods

49 HTTP 1.1 Features

50 HTTP 2.0

# Section 46: Overview

46 Overview

47 URLs, URNs, URIs, IRIs

48 HTTP 1.1 Methods

49 HTTP 1.1 Features

50 HTTP 2.0

# Hypertext Transfer Protocol (HTTP)

- The Hypertext Transfer Protocol (HTTP) version 1.1 was originally defined in 1997 in RFC 2068 and later revised in 1999 in RFC 2616 is one of the core building blocks of the World Wide Web. The latest revision is published in RFCs 7230-7235.
- HTTP is primarily used to exchange documents between clients (browsers) and servers.
- The HTTP protocol runs on top of TCP and it uses the well known port number 80. It uses MIME conventions to distinguish different media types.
- RFC 2817 describes how to use TLS with HTTP 1.1.
- The Hypertext Transfer Protocol (HTTP) version 2.0 was published in 2015 in RFC 7540. It supports multiple streams multiplexed over a single connection and it enables servers to push data to clients.

# Section 47: URLs, URNs, URIs, IRIs

46 Overview

**47** URLs, URNs, URIs, IRIs

48 HTTP 1.1 Methods

49 HTTP 1.1 Features

50 HTTP 2.0

# URL, URI, URN, IRI, ...

- *Uniform Resource Identifier (URI)*

A URI is a sequence of characters from the US-ASCII for identifying an abstract or physical resource.

- *Uniform Resource Locator (URL)*

The term URL refers to the subset of URIs that provide a means of locating the resource by describing its primary access mechanism (e.g., its network "location").

- *Uniform Resource Name (URN)*

The term URN refer to the subset of URIs that identify resources by means of a globally unique and persistent name.

- *Internationalized Resource Identifier (IRI)*

An IRI is a sequence of characters from the Universal Character Set for identifying an abstract or physical resource.

# URI Syntax in ABNF

URI = scheme ":" hier-part [ "?" query ] [ "#" fragment ]

hier-part = "//" authority path-abempty  
/ path-absolute  
/ path-rootless  
/ path-empty

URI-reference = URI / relative-ref

absolute-URI = scheme ":" hier-part [ "?" query ]

relative-ref = relative-part [ "?" query ] [ "#" fragment ]

relative-part = "//" authority path-abempty  
/ path-absolute  
/ path-noscheme  
/ path-empty

scheme = ALPHA \*( ALPHA / DIGIT / "+" / "-" / "." )

# URI Syntax in ABNF

```
authority    = [ userinfo "@" ] host [ ":" port ]
userinfo     = *( unreserved / pct-encoded / sub-delims / ":" )
host         = IP-literal / IPv4address / reg-name
port        = *DIGIT

IP-literal   = "[" ( IPv6address / IPvFuture  ) "]"

IPvFuture    = "v" 1*HEXDIG "." 1*( unreserved / sub-delims / ":" )

IPv6address  =
    6( h16 ":" ) ls32
  /
    "::" 5( h16 ":" ) ls32
  / [
    h16 ] "::" 4( h16 ":" ) ls32
  / [ *1( h16 ":" ) h16 ] "::" 3( h16 ":" ) ls32
  / [ *2( h16 ":" ) h16 ] "::" 2( h16 ":" ) ls32
  / [ *3( h16 ":" ) h16 ] "::"   h16 ":"   ls32
  / [ *4( h16 ":" ) h16 ] "::"
  / [ *5( h16 ":" ) h16 ] "::"
  / [ *6( h16 ":" ) h16 ] "::"
```



# URI Syntax in ABNF

```
h16           = 1*4HEXDIG
ls32          = ( h16 ":" h16 ) / IPv4address

IPv4address   = dec-octet "." dec-octet "." dec-octet "." dec-octet

dec-octet     = DIGIT           ; 0-9
               / %x31-39 DIGIT ; 10-99
               / "1" 2DIGIT    ; 100-199
               / "2" %x30-34 DIGIT ; 200-249
               / "25" %x30-35   ; 250-255

reg-name      = *( unreserved / pct-encoded / sub-delims )

path          = path-abempty    ; begins with "/" or is empty
               / path-absolute  ; begins with "/" but not "//"
               / path-noscheme  ; begins with a non-colon segment
               / path-rootless  ; begins with a segment
               / path-empty     ; zero characters
```

# URI Syntax in ABNF

```
path-abempty  = *( "/" segment )
path-absolute = "/" [ segment-nz *( "/" segment ) ]
path-noscheme = segment-nz-nc *( "/" segment )
path-rootless = segment-nz *( "/" segment )
path-empty    = 0<pchar>

segment       = *pchar
segment-nz    = 1*pchar
segment-nz-nc = 1*( unreserved / pct-encoded / sub-delims / "@" )
pchar         = unreserved / pct-encoded / sub-delims / ":" / "@"

query         = *( pchar / "/" / "?" )
fragment     = *( pchar / "/" / "?" )

pct-encoded   = "%" HEXDIG HEXDIG
unreserved    = ALPHA / DIGIT / "-" / "." / "_" / "~"
reserved      = gen-delims / sub-delims
gen-delims    = ":" / "/" / "?" / "#" / "[" / "]" / "@"
sub-delims    = "!" / "$" / "&" / ">" / "(" / ")"
              / "*" / "+" / "," / ";" / "="
```

# Section 48: HTTP 1.1 Methods

46 Overview

47 URLs, URNs, URIs, IRIs

**48 HTTP 1.1 Methods**

49 HTTP 1.1 Features

50 HTTP 2.0

# HTTP 1.1 Methods

Method	Description
GET	Retrieve a resource identified by a URI
HEAD	Retrieve meta-information of a resource identified by a URI
POST	Annotate an existing resource by passing information to a resource
PUT	Store information under the supplied URI (may create a new resource)
DELETE	Delete the resource identified by the URI
OPTIONS	Request information about methods supported for a URI
TRACE	Application-layer loopback of request messages for testing purposes
CONNECT	Initiate a tunnel such as a TLS or SSL tunnel

- A Client invokes a *method* on a *resource* by sending a *request message*
- A server returns a *response message*, which may include a *representation* of the accessed *resource*

# Safe and idempotent methods

- Safe methods:
  - Safe methods are intended only for information retrieval and should not change the state of the server
  - Safe methods should not have side effects, beyond relatively harmless effects such as logging
  - The methods GET, HEAD, OPTIONS and TRACE are defined to be safe
- Idempotent methods:
  - Idempotent methods can be executed multiple times without producing results that are different from a single execution of the method
  - The methods PUT and DELETE are idempotent.
  - Safe methods should be idempotent as well

# Method Properties

Method	Req Body	Res Body	Safe	Idempotent	Cacheable
GET	No	Yes	Yes	Yes	Yes
HEAD	No	No	Yes	Yes	Yes
POST	Yes	Yes	No	No	Yes
PUT	Yes	Yes	No	Yes	No
DELETE	No	Yes	No	Yes	No
OPTIONS	Opt	Yes	Yes	Yes	No
CONNECT	Yes	Yes	No	No	No
TRACE	Yes	Yes	Yes	Yes	No

- Supporting caches well is a fundamental goal of HTTP

# HTTP 1.1 ABNF

Message = Request / Response

Request = Request-Line \*(message-header CRLF) CRLF [ message-body ]

Response = Status-Line \*(message-header CRLF) CRLF [ message-body ]

Request-Line = Method SP Request-URI SP HTTP-Version CRLF

Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF

Method = "OPTIONS" / "GET" / "HEAD" / "POST"

Method =/ "PUT" / "DELETE" / "TRACE" / "CONNECT"

Method =/ Extension-Method

Extension-Method = token

# HTTP 1.1 ABNF (cont.)

```
Request-URI    = "*" | absoluteURI | abs_path | authority
HTTP-Version   = "HTTP" "/" 1*DIGIT "." 1*DIGIT
Status-Code    = 3DIGIT
Reason-Phrase  = *<TEXT, excluding CR, LF>
```

```
message-header = field-name ":" [ field-value ]
field-name     = token
field-value    = *( field-content / LWS )
field-content  = <the OCTETs making up the field-value
                and consisting of either *TEXT or combinations
                of token, separators, and quoted-string>
```



# Section 49: HTTP 1.1 Features

46 Overview

47 URLs, URNs, URIs, IRIs

48 HTTP 1.1 Methods

**49 HTTP 1.1 Features**

50 HTTP 2.0

# Persistent Connections and Pipelining

- A client can establish a persistent connection to a server and use it to send multiple Request messages.
- HTTP relies on the MIME Content-Length header field to detect the end of a message body (document).
- Early HTTP versions allowed only a single Request/Response exchange over a single TCP connection, which is of course rather expensive.
- HTTP 1.1 also allows clients to make multiple requests without waiting for each response (pipelining), which can significantly reduce latency.

# Chunked Transfer Encoding

- Supports streaming of data where the content length is initially not known
- Uses the Transfer-Encoding HTTP header in place of the Content-Length header
- Content is send in small chunks.
- The size of each chunk is sent right before the chunk itself.
- The data transfer is terminated by a final chunk of length zero.

# Caching and Proxies

- Probably the most interesting and also most complex part of HTTP is its support for proxies and caching.
- Proxies are entities that exist between the client and the server and which basically relay requests and responses.
- Some proxies and clients maintain caches where copies of documents are stored in local storage space to speedup future accesses to these cached documents.
- The HTTP protocol allows a client to interrogate the server to determine whether the document has changed or not.
- Not all problems related to HTTP proxies and caches have been solved. A good list of issues can be found in RFC 3143.

# Negotiation

- Negotiation can be used to select different document formats, different transfer encodings, different languages or different character sets.
- Server-driven negotiation begins with a request from a client (a browser). The client indicates a list of its preferences. The server then decides how to best respond to the request.
- Client-driven negotiation requires two requests. The client first asks the server what is available and then decides which concrete request to send to the server.
- In most cases, server-driven negotiation is used since this is much more efficient.

# Negotiation Example

- The following is an example of typical negotiation header lines:

```
Accept: text/xml, application/xml, text/html;q=0.9, text/plain;q=0.8
```

```
Accept-Language: de, en;q=0.5
```

```
Accept-Encoding: gzip, deflate, compress;q=0.9
```

```
Accept-Charset: ISO-8859-1, utf8;q=0.66, *;q=0.33
```

- The first line indicates that the client accepts text/xml, application/xml, and text/plain and that it prefers text/html over text/plain.
- The last line indicates that the client prefers ISO-8859-1 encoding (with preference 1), UTF8 encoding with preference 0.66 and any other encoding with preference 0.33.

# Conditional Requests

- Clients can make conditional requests by including headers that qualify the conditions under which the request should be honored.
- Conditional requests can be used to avoid unnecessary requests (e.g., to validate cached data).
- Example:  
`If-Modified-Since: Wed, 26 Nov 2003 23:21:08 +0100`
- The server checks whether the document was changed after the date indicated by the header line and only process the request if this is the case.

# Entity Tags

- An entity tag (ETag) is an opaque identifier assigned by a web server to a specific version of a resource found at a URL.
- If the resource content at that URL ever changes, a new and different ETag is assigned.
- ETags can be quickly compared to determine whether two versions of a resource are the same.
- A client can send a conditional request using the `If-None-Match` header.
- Example:  
`If-None-Match: "686897696a7c876b7e"`
- ETags can be used like cookies for tracking users. . .



# Other Features and Extensions

- Patch Method (RFC 5789)
  - A method to apply partial modifications to a resource.
- Delta Encoding (RFC 3229)
  - A mechanism to request only the document changes relative to a specific version.
- Web Distributed Authoring (RFC 2518, RFC 3253)
  - An extension to the HTTP/1.1 protocol that allows clients to perform remote web content authoring operations.
- HTTP as a Substrate (RFC 3205)
  - HTTP is sometimes used as a substrate for other application protocols (e.g., the Internet Printing Protocol).
  - There are some pitfalls with this approach as documented in RFC 3205.

# Section 50: HTTP 2.0

46 Overview

47 URLs, URNs, URIs, IRIs

48 HTTP 1.1 Methods

49 HTTP 1.1 Features

**50 HTTP 2.0**

- HTTP/1.0
  - separate TCP connection for every resource request
  - published as RFC 1945 in May 1996
- HTTP/1.1
  - persistent connections and pipelining
  - conditional requests
  - published in RFC 2068 in Jan 1997, revised as RFC 2616 in Jun 1999
  - modularized version published as RFCs 7230-7235 in Jun 2014
- HTTP/2
  - header compression (binary encoding)
  - multiplexing (avoiding head-of-line blocking)
  - server push into client caches
  - published as RFC 7540 in May 2015

# References I



R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee.  
**Hypertext Transfer Protocol – HTTP/1.1.**  
RFC 2616, UC Irvine, Compaq/W3C, Compaq, W3C/MIT, Xerox, Microsoft, W3C/MIT, June 1999.



R. Khare and S. Lawrence.  
**Upgrading to TLS Within HTTP/1.1.**  
RFC 2817, 4K Associates / UC Irvine, Agranat Systems, May 2001.



E. Rescorla.  
**HTTP Over TLS.**  
RFC 2818, RTFM, May 2000.



D. Robinson and K. Coar.  
**The Common Gateway Interface (CGI) Version 1.1.**  
RFC 3875, Apache Software Foundation, October 2004.



J. Mogul, B. Krishnamurthy, F. Douglass, A. Feldmann, Y. Goland, A. van Hoff, and D. Hellerstein.  
**Delta encoding in HTTP.**  
RFC 3229, Compaq WRL, AT&T, Univ. of Saarbruecken, Marimba, ERS/USDA, January 2002.



Y. Goland, E. Whitehead, A. Faizi, S. Carter, and D. Jensen.  
**HTTP Extensions for Distributed Authoring – WEBDAV.**  
RFC 2518, Microsoft, UC Irvine, Netscape, Novell, February 1999.

# References II

-  G. Clemm, J. Amsden, T. Ellison, C. Kaler, and J. Whitehead.  
Versioning Extensions to WebDAV (Web Distributed Authoring and Versioning).  
RFC 3253, Rational Software, IBM, Microsoft, U.C. Santa Cruz, March 2002.
-  K. Moore.  
On the use of HTTP as a Substrate.  
RFC 3205, University of Tennessee, February 2002.
-  R. Fielding and J. Reschke.  
Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing.  
RFC 7230, Adobe, greenbytes, June 2014.
-  R. Fielding and J. Reschke.  
Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content.  
RFC 7231, Adobe, greenbytes, June 2014.
-  R. Fielding and J. Reschke.  
Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests.  
RFC 7232, Adobe, greenbytes, June 2014.
-  R. Fielding, Y. Lafon, and J. Reschke.  
Hypertext Transfer Protocol (HTTP/1.1): Range Requests.  
RFC 7233, Adobe, W3C, greenbytes, June 2014.

# References III



R. Fielding, M. Nottingham, and J. Reschke.

Hypertext Transfer Protocol (HTTP/1.1): Caching.

RFC 7234, Adobe, Akamai, greenbytes, June 2014.



R. Fielding and J. Reschke.

Hypertext Transfer Protocol (HTTP/1.1): Authentication.

RFC 7235, Adobe, greenbytes, June 2014.



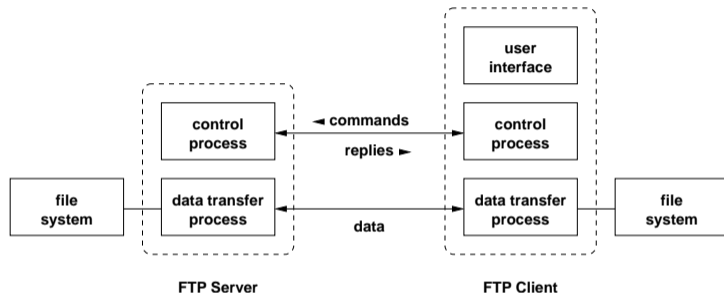
M. Belshe, R. Peon, and M. Thomson.

Hypertext Transfer Protocol Version 2 (HTTP/2).

RFC 7540, BitGo, Google, Mozilla, May 2015.

# Part 12: File Transfer Protocol

# File Transfer Protocol (FTP)



- The FTP protocol defined in RFC 959 uses a separate TCP connection for each data transfer.
- This approach sidesteps any problems about marking the end of files.



# File Transfer Protocol (FTP)

- The control connection uses a text-based line-oriented protocol which is similar to SMTP.
- The client sends commands which are processed by the server. The server sends responses using three digit response codes.
- If the data transfer connection is initiated by the server, then the client's port number must be conveyed first to the server.
- If the data transfer connection is initiated by the client, then the well-known port number 20 is used. The well-known port number for the control connection is 21.

# File Transfer Protocol (FTP)

- FTP allows to resume a data transmission that did not complete using a special restart mechanism.
- FTP can be used to initiate a data transfer between two remote systems. However, this feature of FTP can result in some interesting security problems (see RFC 2577 for more details).
- RFC 2228 defines security extensions for FTP (authentication, integrity and confidentiality).
- RFC 4217 defines how to run FTP over TLS.

# FTP Commands

```
USER <SP> <username> <CRLF>
PASS <SP> <password> <CRLF>
ACCT <SP> <account-information> <CRLF>
CWD <SP> <pathname> <CRLF>
CDUP <CRLF>
SMNT <SP> <pathname> <CRLF>
QUIT <CRLF>
REIN <CRLF>
PORT <SP> <host-port> <CRLF>
PASV <CRLF>
TYPE <SP> <type-code> <CRLF>
STRU <SP> <structure-code> <CRLF>
MODE <SP> <mode-code> <CRLF>
RETR <SP> <pathname> <CRLF>
```

# FTP Commands

```
STOR <SP> <pathname> <CRLF>
STOU <CRLF>
APPE <SP> <pathname> <CRLF>
ALLO <SP> <decimal-integer>
      [<SP> R <SP> <decimal-integer>] <CRLF>
REST <SP> <marker> <CRLF>
RNFR <SP> <pathname> <CRLF>
RNTO <SP> <pathname> <CRLF>
ABOR <CRLF>
DELE <SP> <pathname> <CRLF>
RMD  <SP> <pathname> <CRLF>
MKD  <SP> <pathname> <CRLF>
PWD  <CRLF>
```

# FTP Commands

```
LIST [<SP> <pathname>] <CRLF>  
NLST [<SP> <pathname>] <CRLF>  
SITE <SP> <string> <CRLF>  
SYST <CRLF>  
STAT [<SP> <pathname>] <CRLF>  
HELP [<SP> <string>] <CRLF>  
NOOP <CRLF>
```

# FTP Command Parameters

```
<username> ::= <string>
<password> ::= <string>
<account-information> ::= <string>
<string> ::= <char> | <char><string>
<char> ::= any of the 128 ASCII characters
           except <CR> and <LF>
<marker> ::= <pr-string>
<pr-string> ::= <pr-char> | <pr-char><pr-string>
<pr-char> ::= printable characters, any ASCII
            code 33 through 126
<byte-size> ::= <number>
<host-port> ::= <host-number>,<port-number>
```

# FTP Command Parameters

```
<host-number> ::= <number>,<number>,<number>,<number>
<port-number> ::= <number>,<number>
<number>      ::= any decimal integer 1 through 255
<form-code>  ::= N | T | C
<type-code>  ::= A [<sp> <form-code>]
               | E [<sp> <form-code>]
               | I
               | L <sp> <byte-size>
<structure-code> ::= F | R | P
<mode-code>  ::= S | B | C
<pathname>   ::= <string>
<decimal-integer> ::= any decimal integer
```

# References



J. Postel and J. Reynolds.  
**File Transfer Protocol (FTP).**  
RFC 959, ISI, October 1985.



M. Horowitz and S. Lunt.  
**FTP Security Extensions.**  
RFC 2228, Cygnus Solutions, Bellcore, October 1997.



M. Allman, S. Ostermann, and C. Metz.  
**FTP Extensions for IPv6 and NATs.**  
RFC 2428, NASA Lewis/Sterling Software, Ohio University, The Inner Net, September 1998.



P. Ford-Hutchinson.  
**Securing FTP with TLS.**  
RFC 4217, IBM UK Ltd, October 2005.

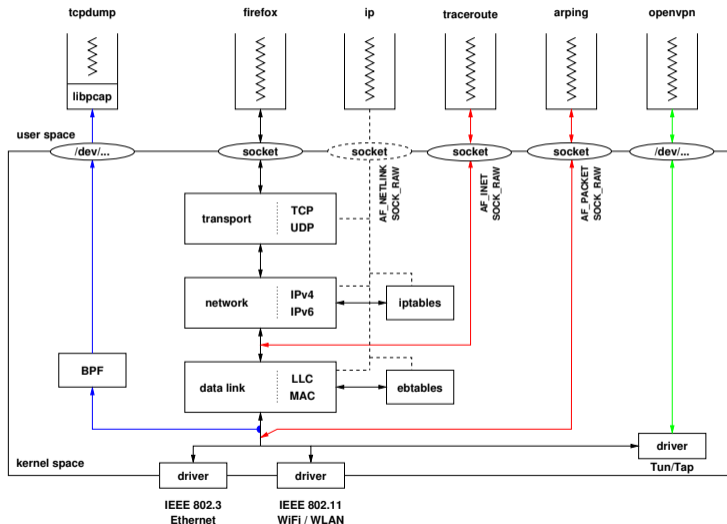


## 51 Packet Capturing

# Section 51: Packet Capturing

## 51 Packet Capturing

# Linux Network Stack Overview



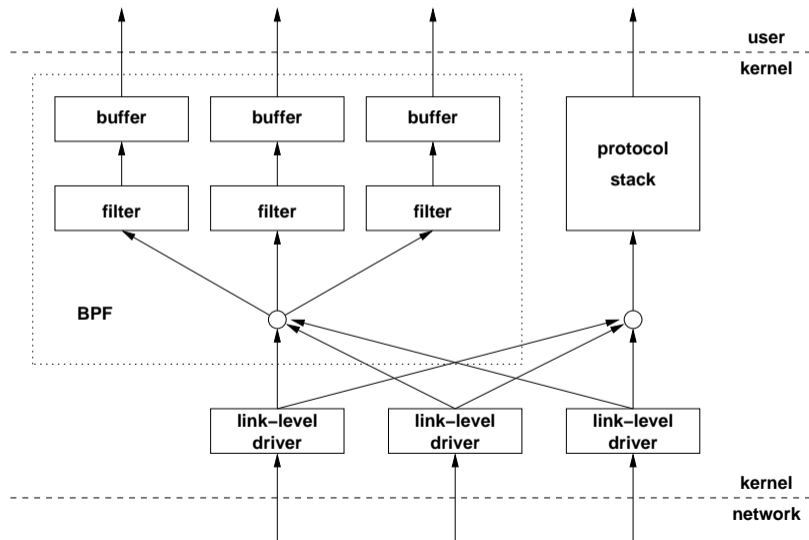
# Motivation and Requirements

- Network protocol analyzers like `wireshark`, `tcpdump` or `ngrep` running in user space want to receive (raw) packets as they are received by the device drivers in the kernel.
  - User space tools like `iftop`, `ntop`, `yaf`, `softflowd`, `vnstat` want to receive (raw) packets in order to produce statistics.
  - To achieve a solution with good performance,
    - captured packets should be filtered or aggregated as early as possible,
    - copying of packets should be avoided as much as possible since copying data is an expensive operation, and
    - the number of system calls and associated context switches should be minimized
- ⇒ Discard unimportant packets (frames) as early as possible

# BSD Packet Filter (BPF)

- The BSD Packet Filter is based on a simple but powerful control-flow machine, which is implemented in the kernel.
- A BPF program is generated by user space applications and sent to the kernel for execution.
- Human readable filter expressions are translated into BPF programs using a compiler.
- Packets that pass a BPF filter are first collected within the kernel and later moved to user space applications in batches.
- The BPF machine has the following components:
  - An accumulator for all calculations
  - An index register (x) allowing access to data relative to a certain position
  - Memory for storing intermediate results
- All registers and memory locations of the BPF are 32-bit wide.

# BPF within a Unix Kernel



# BPF Example #1

- Select all Ethernet frames which contain IPv4 packets:

```
(000) ldh   [12]                ; load ethernet type field
(001) jeq   #0x800             jt 2  jf 3 ; compare with 0x800
(002) ret   #96                ; return snaplen
(003) ret   #0                 ; filter failed
```

## BPF Example #2

- Select all Ethernet frames which contain IPv4 packets which do not originate from the networks 128.3.112/24 and 128.3.254/24 (ip and not src net 128.3.112/24 and not src net 128.3.254/24):

```
(000) ldh   [12]           ; load ethernet type field
(001) jeq   #0x800        jt 2   jf 7   ; compare with 0x800
(002) ld    [26]          ; load ipv4 address field
(003) and   #0xffffffff00 ; mask the network part
(004) jeq   #0x80037000   jt 7   jf 5   ; compare with 128.3.112.0
(005) jeq   #0x8003fe00   jt 7   jf 6   ; compare with 128.3.254.0
(006) ret   #96           ; return snaplen
(007) ret   #0            ; filter failed
```



- The `libpcap` C library provides the following functionality:
  - portable API that hides the differences of packet filter implementations in different operating systems.
  - A compiler which translates human readable filter expressions into BPF programs.
  - An interpreter for BPF programs which can be used to filter (previously captured) packets in user space.
  - Functions for writing captured packets to files and for reading previously captured packets from files.

# References



S. McCanne and V. Jacobson.

The BSD Packet Filter: A New Architecture for User-level Packet Capture.

In *Proc. Usenix Winter Conference*, January 1993.