Computer Networks '2016

Jürgen Schönwälder

JACOBS UNIVERSITY

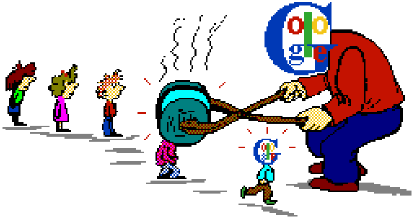
February 26, 2017

http://cnds.eecs.jacobs-university.de/courses/cn-2016/

Part: Preface

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

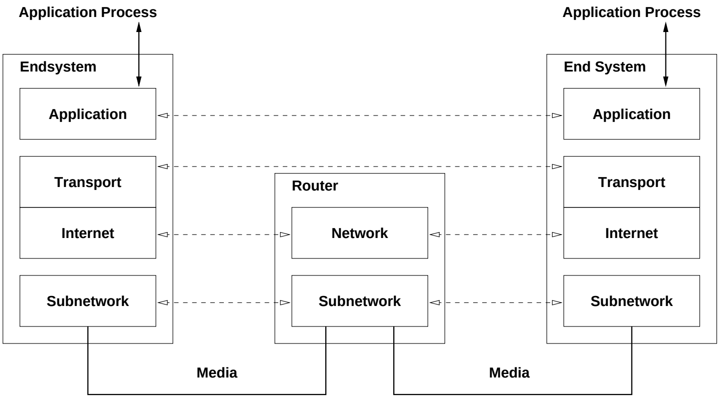
Reasons for not taking this course

- You do not have the time required for this course
- You lack some of the required background
- You just want to know how to configure your system(s)
- You find the topics covered by this course boring
- You are unable to do some programming in C/Unix
- You dislike reading book chapters or specifications
- You hate programming exercises and plugging cables

Part: Introduction

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

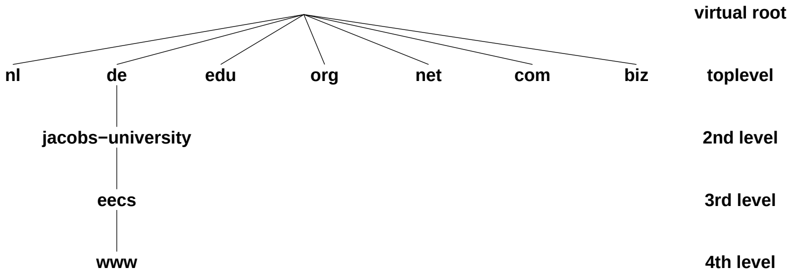
Internet Design Principles

- 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 nulls in IPv6 addresses can be omitted and two consecutive colons can represent a sequence of nulls
- 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

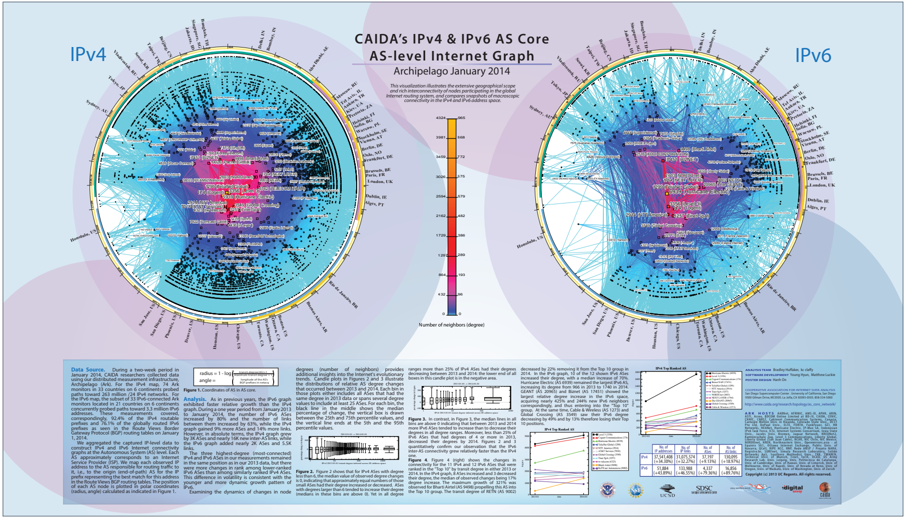


- The Domain Name System (DNS) provides a distributed hierarchical name space which supports the delegation of name assignments
- DNS name resolution translates DNS names into one or more Internet addresses

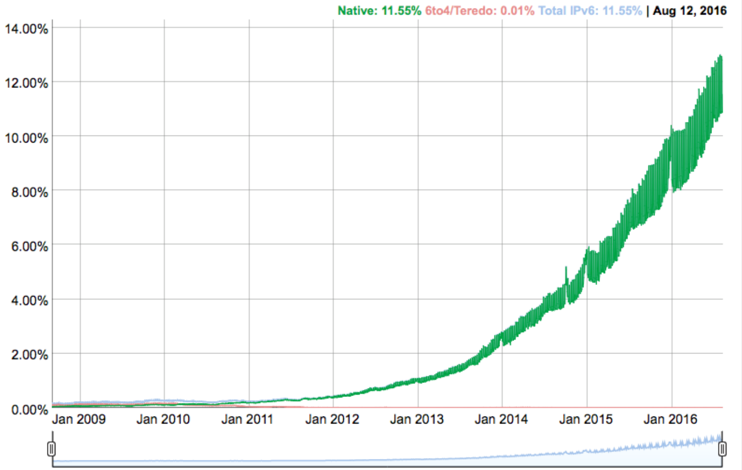
Autonomous Systems

- 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)

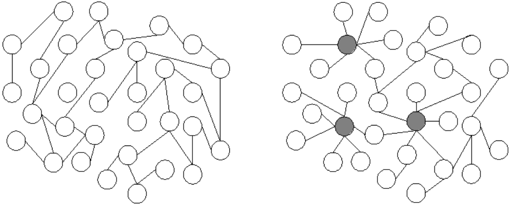
Autonomous Systems



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

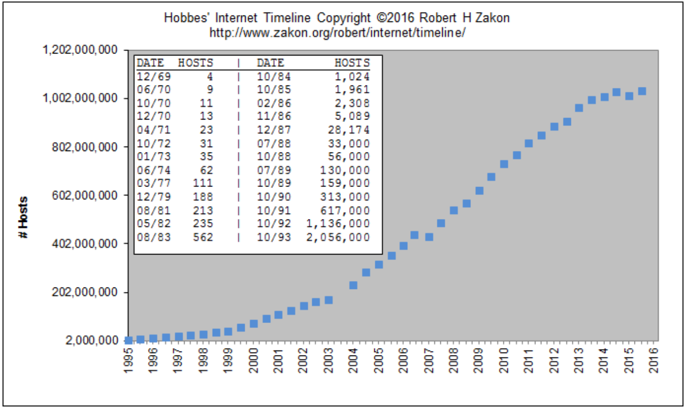
Structure and Growth of the Internet

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: ???

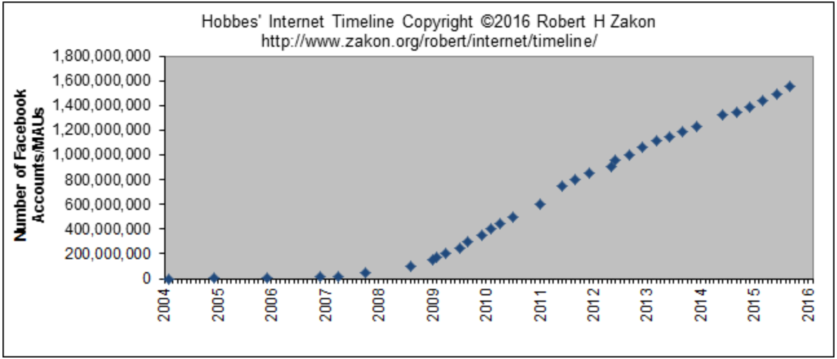
Growth of the Internet



DATE	HOSTS	DATE	HOSTS
12/69	4	10/84	1,024
06/70	9	10/85	1,961
10/70	11	02/86	2,308
12/70	13	11/86	5,089
04/71	23	12/87	28,174
10/72	31	07/88	33,000
01/73	35	10/88	56,000
06/74	62	07/89	130,000
03/77	111	10/89	159,000
12/79	188	10/90	313,000
08/81	213	10/91	617,000
05/82	235	10/92	1,136,000
08/83	562	10/93	2,056,000

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

Growth of Facebook

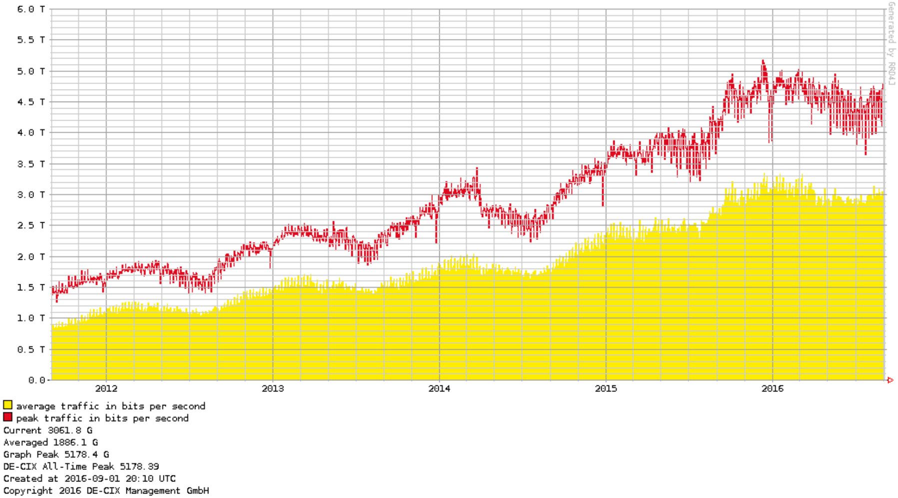


- This is pretty easy to count. . .
- But how many accounts are active, how many are fake accounts?

Internet Exchange Points (Fall 2016)

- Internet Exchange Frankfurt/Germany (DE-CIX)
  - Connected networks: ≈ 650
  - Average throughput (1 year): $\approx 2.9\,Tbps$
  - Maximum throughput (1 year): $\approx 5.2\,Tbps$
  - Maximum capacity: $\approx 14\,Tbps$
  - 170+ 1 Gigabit-Ethernet ports
  - 900+ 10 Gigabit-Ethernet ports
  - 100+ 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)



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
- ...

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

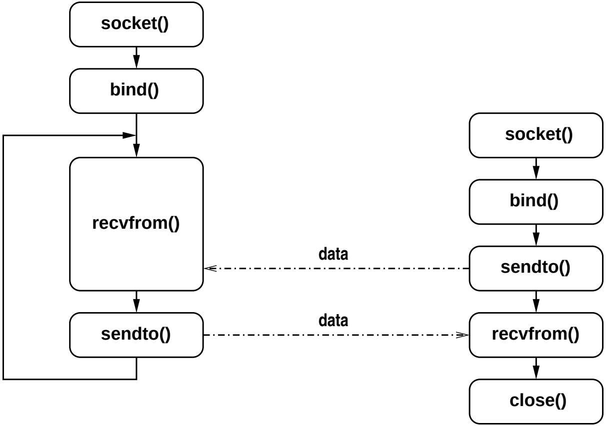
IPv4 Socket Addresses

```
#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 */
};
```

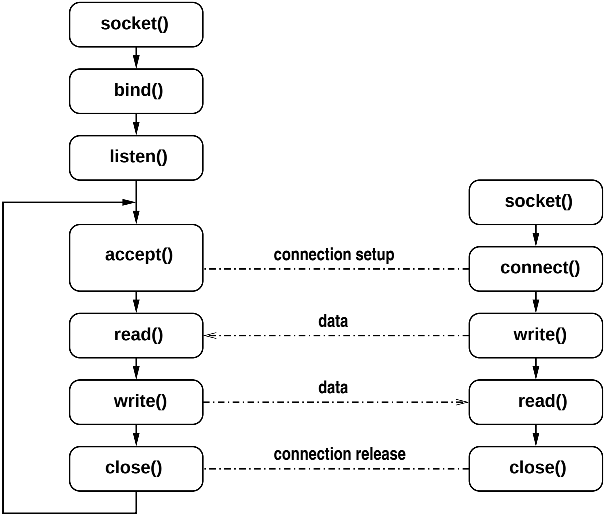
IPv6 Socket Addresses

```
#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 SCOK_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.

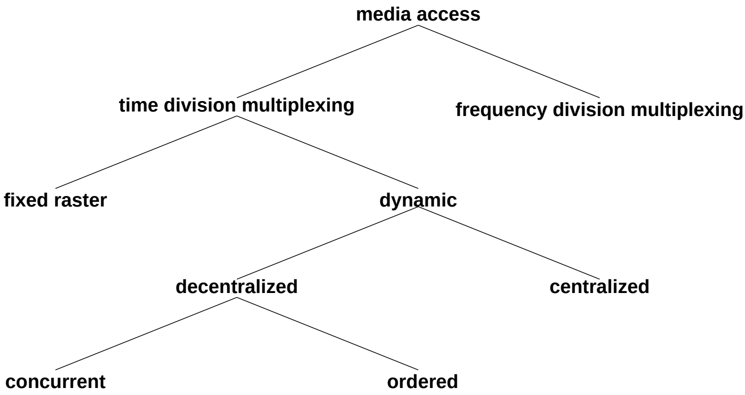
R. Gilligan, S. Thomson, J. Bound, J. McCann, and W. Stevens.
Basic Socket Interface Extensions for IPv6.
RFC 3493, Intransa Inc., Cisco, Hewlett-Packard, February 2003.

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.

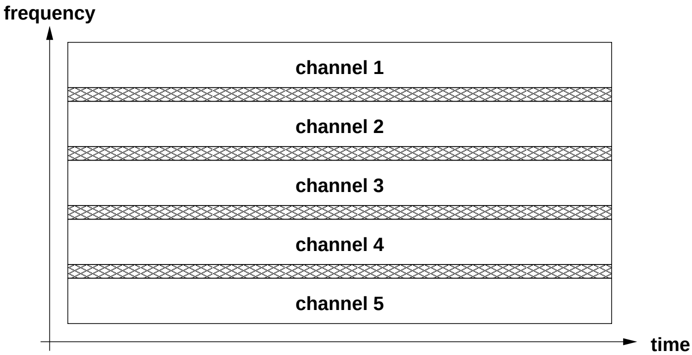
Part: Fundamental Network Concepts

Media Access



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

Frequency Division Multiplexing (FDM)

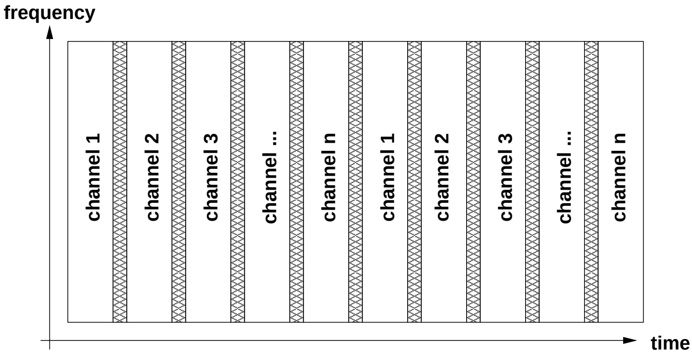


- Signals are carried simultaneously on the same medium by allocating to each signal a different frequency band

Wavelength Division Multiplexing (WDM)

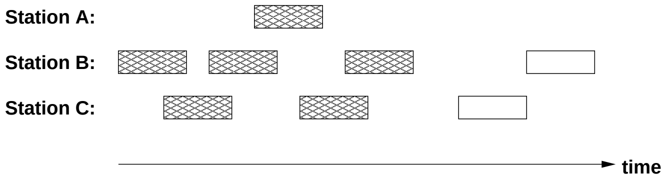
- Optical fibers carry multiple wavelength at the same time
- WDM can achieve very high data rates over a single optical fiber
- Dense WDM (DWDM) is a variation where the wavelengths are spaced close together, which results in an even larger number of channels.
- Theoretically, there is room for 1250 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)



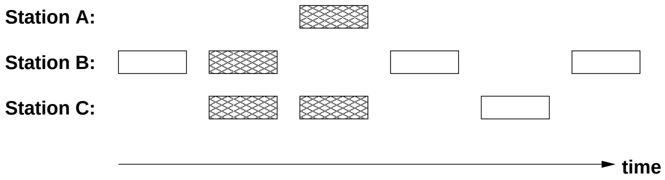
- Signals from a given sources are assigned to specific time slots
- Time slot assignment might be fixed (synchronous TDM) or dynamic (statistical TDM)

Pure Aloha



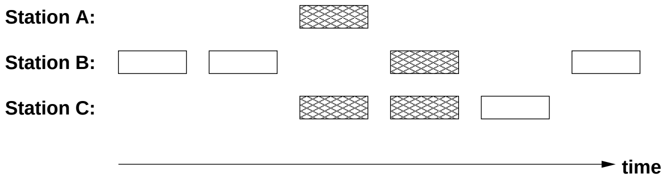
- Developed in the 1970s at the University of Hawaii
- Sender sends data as soon as data becomes available
- Collisions are detected by listening to the signal
- Retransmit after a random pause after a collision
- Not very efficient ($\approx$ 18 % of the channel capacity)

Slotted Aloha



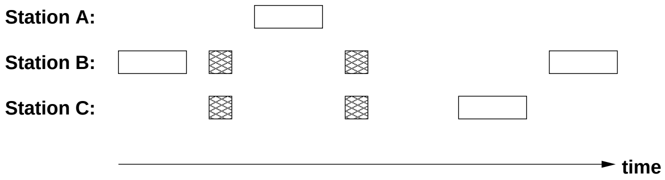
- Senders do not send immediately but wait for the beginning of a time slot
- Time slots may be advertised by short control signals
- Collisions only happen at the start of a transmission
- Avoids sequences of partially overlaying data blocks
- Slightly more efficient (≈ 37 % of the channel capacity)

Carrier Sense Multiple Access (CSMA)



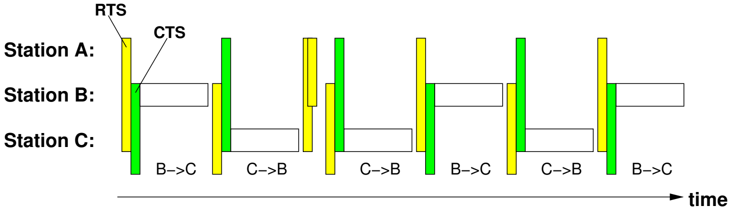
- Sense the media whether it is unused before starting a transmission
- Collisions are still possible (but less likely)
- 1-persistent CSMA: sender sends with probability 1
- p-persistent CSMA: sender sends with probability p
- non-persistent CSMA: sender waits for a random time period before it retries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

CSMA with Collision Detection (CSMA-CD)



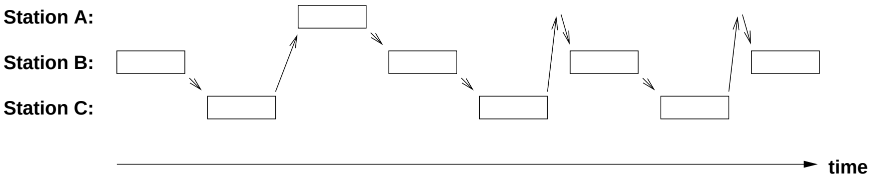
- Terminate the transmission as soon as a collision has been detected (and retry after some random delay)
- Let τ be the propagation delay between two stations with maximum distance
- Senders can be sure that they successfully acquired the medium after 2τ time units
- Used by the classic Ethernet developed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



- 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

Transmission Error Detection

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) $b_n b_{n-1} \ldots b_1 b_0$ is represented as a polynomial $B(x) = b_n x^n + b_{n-1} x^{n-1} + \ldots + b_1 x + b_0$
- 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 $G(x) = g_r x^r + \ldots g_1 x + g_0$ 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--) {
        sum += *buf++;
        if (sum & 0xffff0000) {
            sum &= 0xffff;
            sum++;
        }
    }
    return ~(sum & 0xffff);
}
```

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

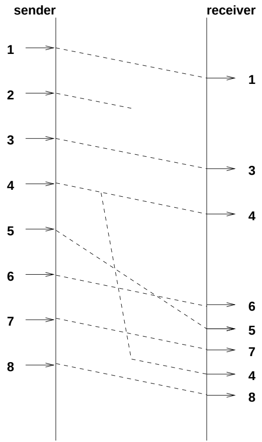
Further Error Situations

- 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, Acknowledgements, Timer

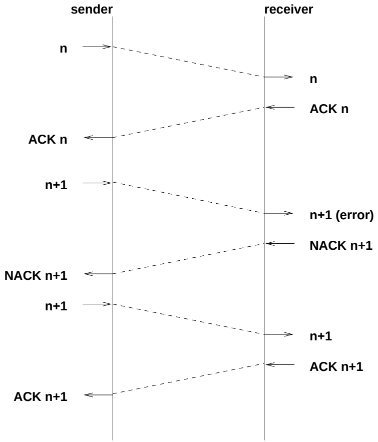
Sequence Numbers

- 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



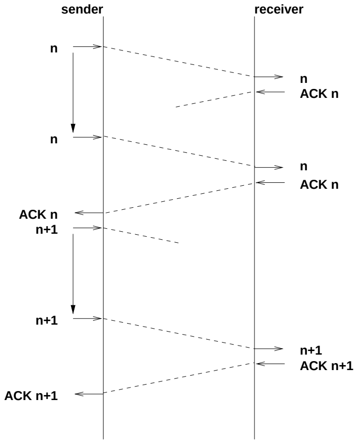
Acknowledgements

- 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



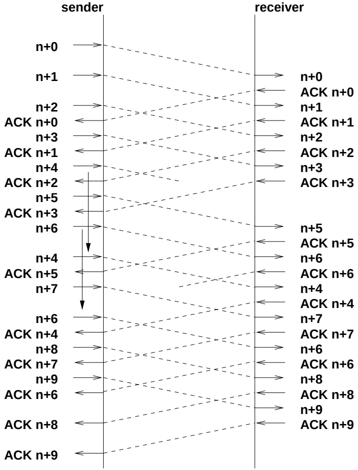
Timers

- 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



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

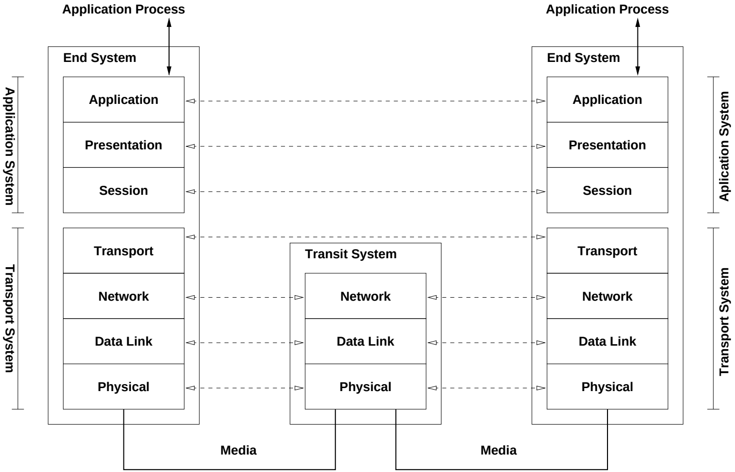
- Implementation on the sender side:
  - `SWS` (send window size)
  - `LAR` (last ack received)
  - `LFS` (last frame send)
  - Invariant: `LFS` - `LAR` $+ 1 \leq$ `SWS`
- Implementation on the receiver side:
  - `RWS` (receiver window size)
  - `LFA` (last frame acceptable)
  - `NFE` (next frame expected)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

OSI Reference Model

OSI Reference Model



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 and Presentation 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 emulationen, 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.

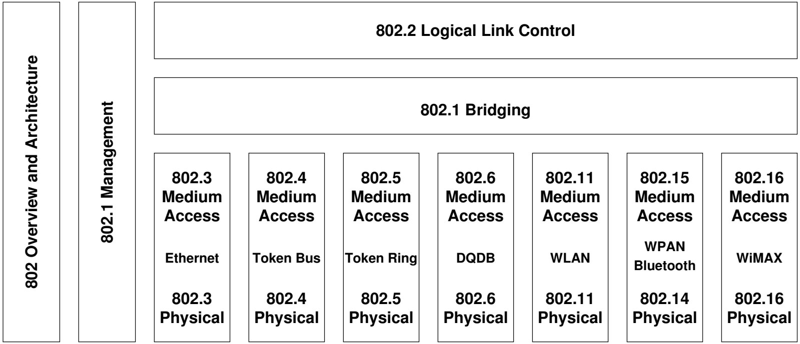
A. S. Tanenbaum.
Computer Networks.
Prentice Hall, 4 edition, 2002.

J. Stone and C. Partridge.
When The CRC and TCP Checksum Disagree.
In *Proc. SIGCOMM 2000*, pages 309–319, Stockholm, August 2000. ACM.

Part: Local Area Networks (IEEE 802)

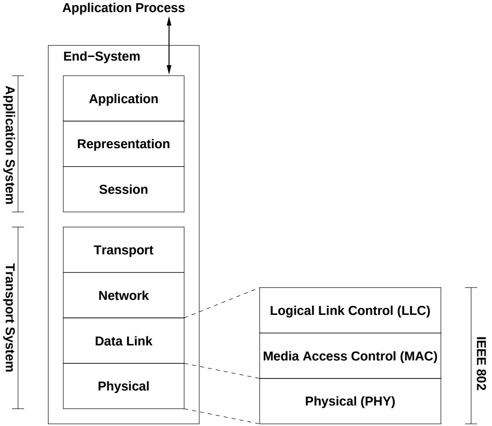
Local Area Networks Overview

IEEE 802 Overview



- IEEE 802 standards are developed since the mid 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

- 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

IEEE 802.3 (Ethernet)

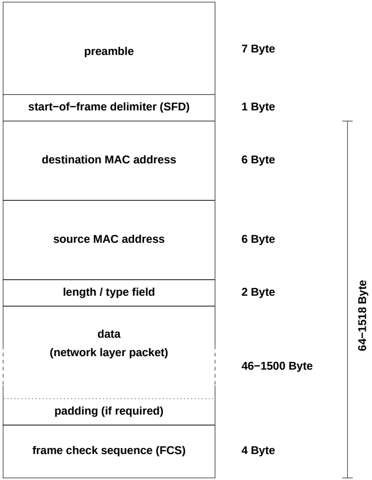
IEEE 802.3 (Ethernet)

- Evolution of Ethernet Technology:

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
2015	1 Tbps Ethernet (predicted)
2020	100 Tbps Ethernet (predicted)

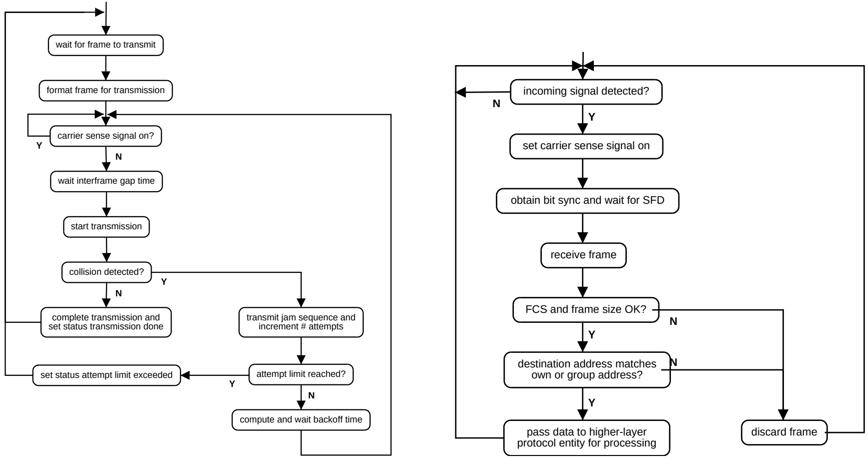
- Link aggregation allows to "bundle" links, e.g., four 10 Gbps links can be bundled to perform like a single 40 Gbps link

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)



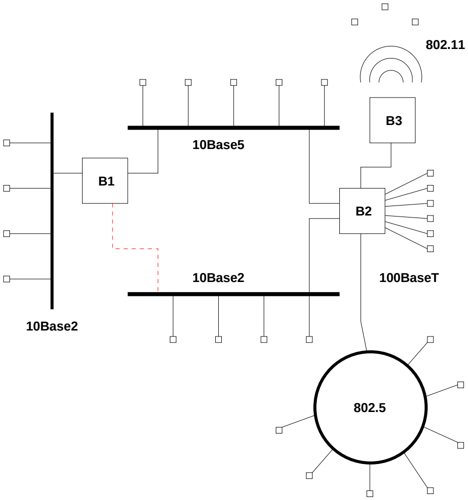
Ethernet Media Types

Name	Medium	Maximum Length
10Base2	coax, ø=0.25 in	200 m
10Base5	coax, ø=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

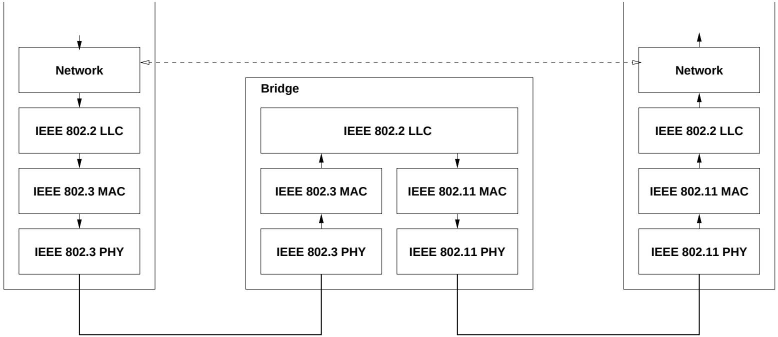
Ethernet Media Types (cont.)

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

Bridged IEEE 802 Networks

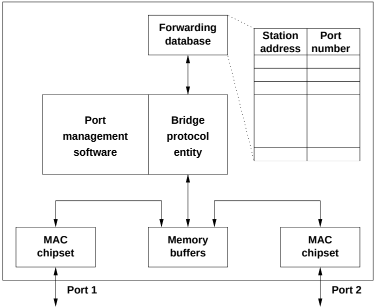


IEEE 802 Bridges



- *Source Routing Bridges*: Sender has to route 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

- Algorithm:
  - The forwarding database is initially empty.
  - 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.
  - Reinitialize the timer attached to the forwarding base entry for the received frame.
- Aging of unsused entries reduces forwarding table size and allows bridges to adapt to topology changes.
- Backward learning requires an cycle free topology.

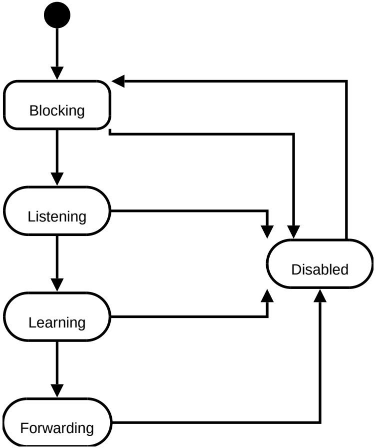
Spanning Tree

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 fame 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



Broadcast Domains

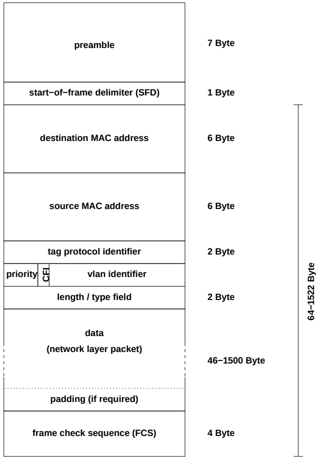
- A bridged LAN defines a single *broadcast domain*:
  - All frames send to the broadcast address are forwarded on all links in the bridged networks.
  - Broadcast traffic can take a significant portion of the available bandwidth.
  - Devices running not well-behaving applications can cause *broadcast 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.

IEEE 802.1Q Virtual LANs

IEEE 802.1Q Virtual LANs

- VLANs allow to separate the traffic on an IEEE 802 network
- A station connected to a certain VLAN only sees frames belonging to that VLAN
- VLANs can reduce the network load; frames that are targeted to all stations (broadcasts) will only be delivered to the stations connected to the VLAN.
- Stations (especially server) can be a member of multiple VLANs simultaneously
- Separation of logical LAN topologies from physical LAN topologies
- VLANs are identified by a VLAN identifier (1..4094)

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

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

IEEE 802.1X Port Access Control

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.

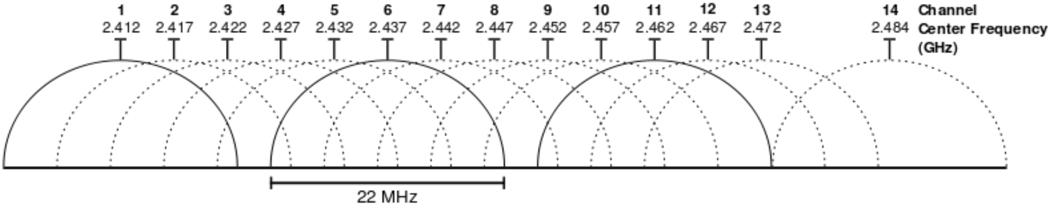
IEEE 802.11 Wireless LAN

IEEE 802.11 Wireless LAN

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

- 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.

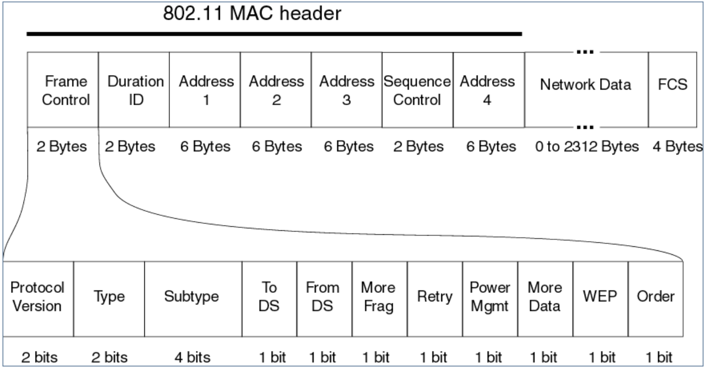
IEEE 802.11 2.4 GHz Channels



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



IEEE 802.15.4 (LowPAN)

- Low power (battery operated) personal area networks (LowPAN)
- Operates in 868-868.8 MHz (Europe), 902-928 MHz (North America), 2400-2483.5 MHz (worldwide) frequency bands.
- Application areas: Home automation, surveillance systems, logistics, ...
- Full function devices (FFDs) serve as coordinator of a personal area network (PAN).
- Reduced function devices (RFDs) are extremely simple and can be battery powered with long lifetimes.
- IEEE 802.15.4a provides localization support.

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.

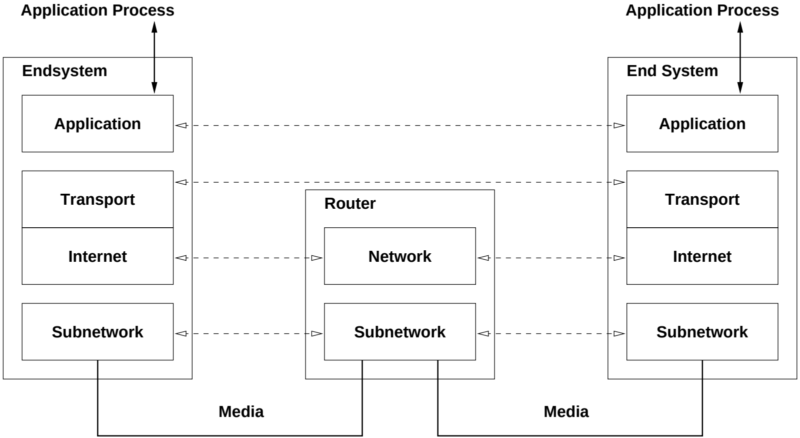
R. Perlman.
An Algorithm for Distributed Computation of a Spanning Tree in an Extended LAN.
SIGCOMM Computer Communication Review, 15(4), September 1985.

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: Internet Network Layer

Internet Model



Terminology

- 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.

Terminology (cont.)

- An *interface* is a node's attachement to a link.
- An *IP address* identifies an interface or a set of interfaces.
- 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 Network Layer Protocols

- 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
- 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

Reserved IPv4 Address Blocks

0.0.0.0/8	"This" Network	[RFC1700]
10.0.0.0/8	Private-Use Networks	[RFC1918]
14.0.0.0/8	Public-Data Networks	[RFC1700]
24.0.0.0/8	Cable Television Networks	[RFC3330]
39.0.0.0/8	Class A Subnet Experiment	[RFC1797]
127.0.0.0/8	Loopback	[RFC1700]
128.0.0.0/16	Reserved by IANA	[RFC3330]
169.254.0.0/16	Link Local	[RFC3330]
172.16.0.0/12	Private-Use Networks	[RFC1918]
191.255.0.0/16	Reserved by IANA	[RFC3330]
192.0.0.0/24	Reserved by IANA	[RFC3330]
192.0.2.0/24	Test-Net / Documentation	[RFC3330]
192.88.99.0/24	6to4 Relay Anycast	[RFC3068]
192.168.0.0/16	Private-Use Networks	[RFC1918]
198.18.0.0/15	Network Interconnect / Device Benchmark Testing	[RFC2544]
223.255.255.0/24	Reserved by IANA	[RFC3330]
224.0.0.0/4	Multicast	[RFC3171]
240.0.0.0/4	Reserved for Future Use	[RFC1700]

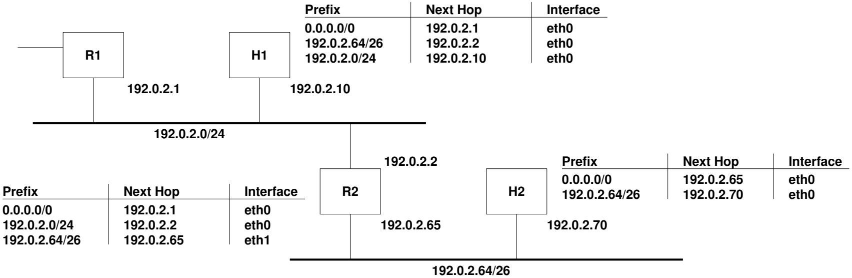
IPv4 Packet Format

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

IPv4 Forwarding

- IPv4 addresses can be devided into a part which identifies a network (netid) and a part which identifies an interface of a node within that network (hostid).
- The number of bits of an IPv4 address which identifies the network is called the *address prefix*.
- The address prefix is commonly written as a decimal number, appended to the usual IPv4 address notation by using a slash (/) as a separator (e.g., 192.0.2.0/24).
- The *forwarding table* realizes a mapping of the network prefix to the next node (next hop) and the local interface used to reach the next node.
- For every IP packet, the entry in the forwarding table has to be found with longest matching network address prefix (longest prefix match).

IPv4 Forwarding Example



H1:

Prefix	Next Hop	Interface
0.0.0.0/0	192.0.2.1	eth0
192.0.2.64/26	192.0.2.2	eth0
192.0.2.0/24	192.0.2.10	eth0

R2:

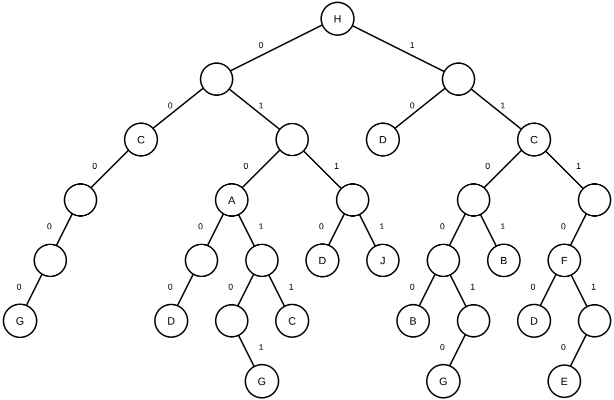
Prefix	Next Hop	Interface
0.0.0.0/0	192.0.2.1	eth0
192.0.2.0/24	192.0.2.2	eth0
192.0.2.64/26	192.0.2.65	eth1

H2:

Prefix	Next Hop	Interface
0.0.0.0/0	192.0.2.65	eth0
192.0.2.64/26	192.0.2.70	eth0

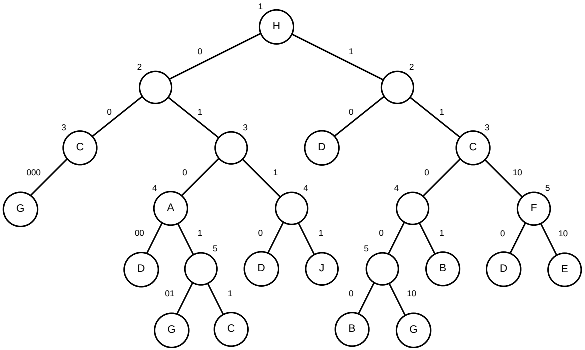
- Default forwarding table entries use the prefix 0.0.0.0/0.
- This example identifies directly reachable hosts by using the IP address of a local interface as the next hop.
- Other representations use an explicit direct/indirect flag.

Longest Prefix Match: Binary Trie



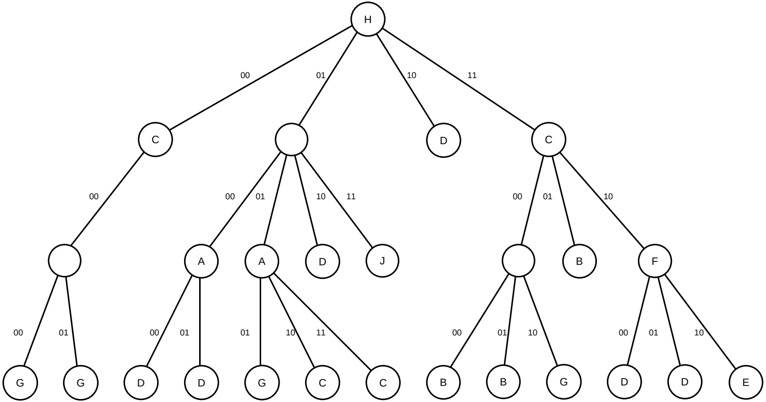
- A binary trie is the representation of the binary prefixes in a tree.

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.

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

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Identifier          |        Sequence Number        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Data ...
+-+-+-+-+-
```

- 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

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             unused                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Internet Header + 64 bits of Original Data Datagram      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- 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.

Unreachable Destination Codes

0	Net Unreachable
1	Host Unreachable
2	Protocol Unreachable
3	Port Unreachable
4	Fragmentation Needed and Don't Fragment was Set
5	Source Route Failed
6	Destination Network Unknown
7	Destination Host Unknown
8	Source Host Isolated
9	Communication with Destination Network is Administratively Prohibited
10	Communication with Destination Host is Administratively Prohibited
11	Destination Network Unreachable for Type of Service
12	Destination Host Unreachable for Type of Service
13	Communication Administratively Prohibited
14	Host Precedence Violation
15	Precedence cutoff in effect

ICMPv4 Redirect

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Router Internet Address                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Internet Header + 64 bits of Original Data Datagram      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- 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.

Redirect Codes

0	Redirect datagrams for the Network.
1	Redirect datagrams for the Host.
2	Redirect datagrams for the Type of Service and Network.
3	Redirect datagrams for the Type of Service and Host.

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 sent in the payload of IEEE 802.3 frames according to the specification in 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)

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Hardware Type         |         Protocol Type         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      HLEN     |     PLEN      |           Operation           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Sender Hardware Address (SHA)                 =
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
= Sender Hardware Address (SHA) |    Sender IP Address (SIP)    =
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
=    Sender IP Address (SIP)    | Target Hardware Address (THA) =
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
=                 Target Hardware Address (THA)                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Target IP Address                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- 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.

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 Protocol Version 6

IPv6 Interface Identifier

- Interface identifiers in IPv6 unicast addresses are used to uniquely indentify interfaces on a link.
- For all unicast addresses, except those that start with binary 000, interface identifiers are required to be 64 bits long and to be constructed in modified EUI-64 format.
- Combination of the interface identifier with a network prefix leads to an IPv6 address.
- Link local unicast addresses have the prefix fe80::/10.
- Ongoing work on the construction of globally unique local addresses using cryptographic hash functions.

Modified EUI-64 Format

```
|0              1|1             3|3              4|
|0              5|6             1|2              7|
+----------------+----------------+----------------+
|cccccc0gcccccccc|ccccccccmmmmmmmm|mmmmmmmmmmmmmmmm|
+----------------+----------------+----------------+
```

```
|0              1|1             3|3              4|4              6|
|0              5|6             1|2              7|8              3|
+----------------+----------------+----------------+----------------+
|cccccc1gcccccccc|cccccccc11111111|11111110mmmmmmmm|mmmmmmmmmmmmmmmm|
+----------------+----------------+----------------+----------------+
```

- Modified EUI-64 format can be obtained from IEEE 802 MAC addresses.
- Warning: These IPv6 addresses constructed out of 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 Packet Format (RFC 2460)

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version| Traffic Class |           Flow Label                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Payload Length        |  Next Header  |   Hop Limit   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                         Source Address                        +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                      Destination Address                      +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

IPv6 Extension Header

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

IPv6 Forwarding

- IPv6 packets are forwarded using the longest prefix match algorithm which is used in the IPv4 network.
- IPv6 addresses have much longer prefixes which allows to do better address aggregation in order to reduce the number of forwarding table entries.
- Due to the length of the prefixes, it is even more crucial to use an algorithm whose complexity does not dependent on the number of entries in the forwarding table or the average prefix length.

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.
- ICMPv6 integrates the functions of ICMPv4 and ARP and adds a simple bootstrapping (auto-configuration) mechanism.
- For more complex node configuration, DHCPv6 (RFC 3315) can be used.

IPv6 over IEEE 802.3 (RFC 2464)

- IPv6 packets can be encapsulated into IEEE 802.3 frames:
  - 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 mapping tables (sometimes also called address translation tables) can either be statically configured or dynamically learned using neighbor discovery.

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.

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.

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

References II

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.

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.

References III

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: Internet Routing

Internet Routing Overview

Internet Routing Protocols

- The *Routing Information Protocol* (RIP) is a simple distance vector IGP routing protocol based on the distributed Bellman-Ford shortest path algorithm
- The *Open Shortest Path First* (OSPF) IGP routing protocol floods link state information so that every node can compute shortest paths using Dijkstra's shortest path algorithm
- The *Intermediate System to Intermediate System* (IS-IS) routing protocol is another link state routing protocol adopted from the ISO/OSI standards
- The *Border Gateway Protocol* (BGP) EGP routing protocol propagates reachability information between ASs, which is used to make policy-based routing decisions

Distance Vector Routing (RIP)

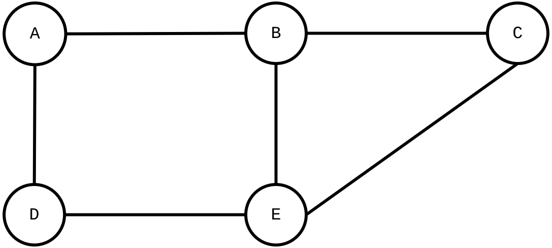
Bellman-Ford

- 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

B	Dest.	Link	Cost
	B	local	0

C	Dest.	Link	Cost
	C	local	0

D	Dest.	Link	Cost
	D	local	0

E	Dest.	Link	Cost
	E	local	0

Bellman-Ford Example (Round 1)

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

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

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

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

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

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
	C	C-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:

```
A --- B
 \   /
   C
   |
   D
```

- 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:

```
A --- B
 \   /
   C
   |
   D
```

- 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

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Command    |    Version    |          must be zero         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                          RIP Entries                          ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- 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.)

```
 0                   1                   2                   3 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Address Family Identifier   |          Route Tag            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          IP Address                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Subnet Mask                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Next Hop                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Metric                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- 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.)

```
 0                   1                   2                   3 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             0xFFFF            |      Authentication Type      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                        Authentication                         +
|                                                               |
+                                                               +
|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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

Link State Routing (OSPF)

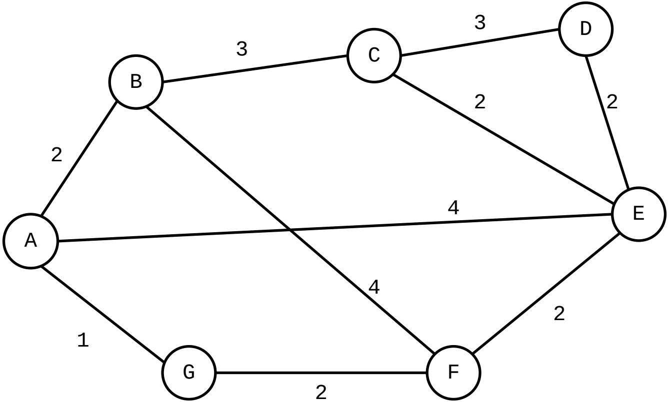
Dijkstra's Shortest Path Algorithm

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.

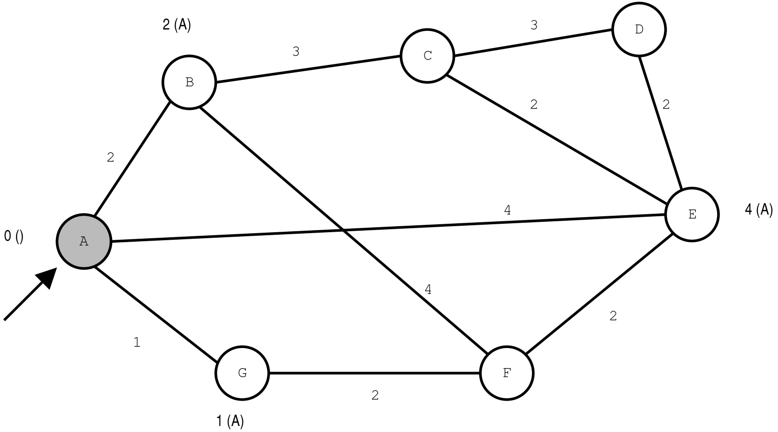
Dijkstra's Shortest Path Algorithm (cont.)

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.

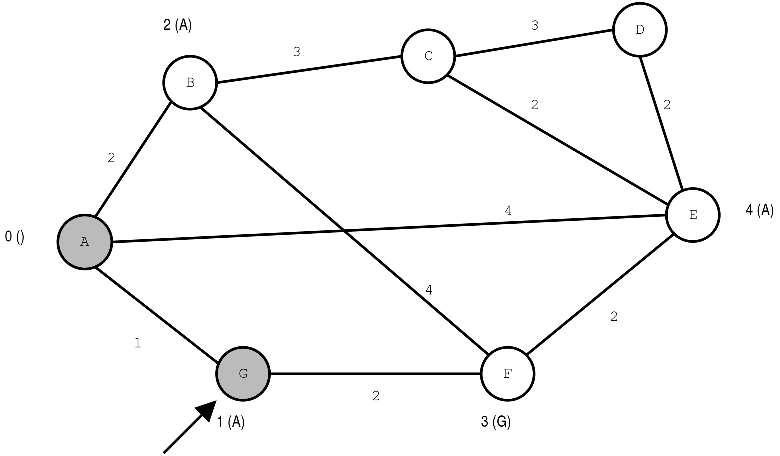
Dijkstra Example



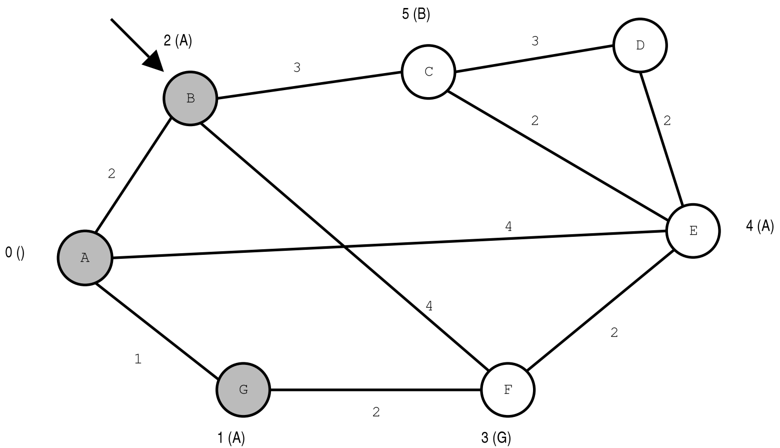
Dijkstra Example (cont.)



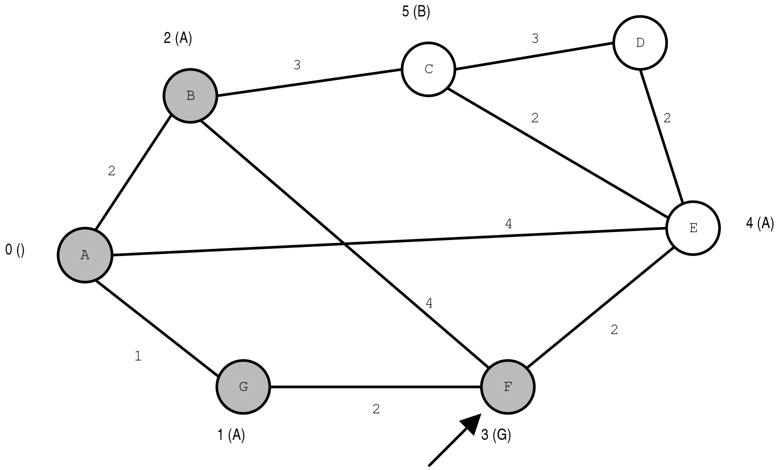
Dijkstra Example (cont.)



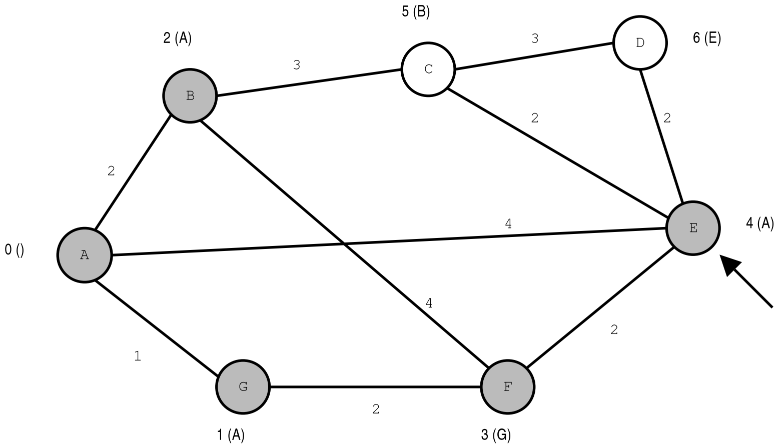
Dijkstra Example (cont.)



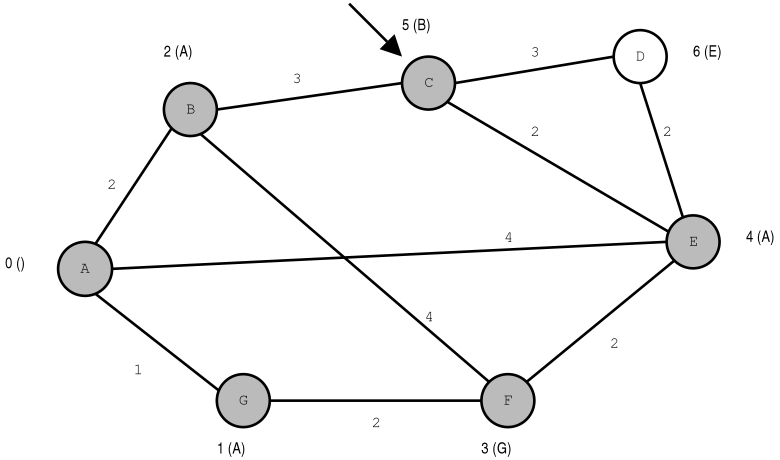
Dijkstra Example (cont.)



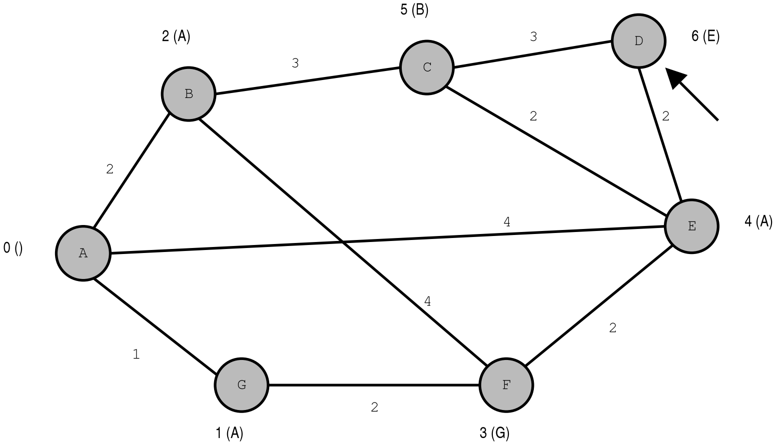
Dijkstra Example (cont.)



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.

OSPF Areas

- 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

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Version #   |     Type      |         Packet Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Router ID                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Area ID                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |             AuType            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Authentication                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Authentication                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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 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|I|M|MS
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     DD sequence number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-                                                             -+
|                                                               |
+-                      An LSA Header                          -+
|                                                               |
+-                                                             -+
|                                                               |
+-                                                             -+
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              ...                              |
```

OSPF Link State Request 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
:                                                               :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          LS type                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Link State ID                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Advertising Router                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              ...                              |
```

OSPF Link State Update 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
:                                                               :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            # LSAs                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-                                                            +-+
|                             LSAs                              |
+-                                                            +-+
|                              ...                              |
```

OSPF Link State Ack. 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
:                                                               :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-                                                             -+
|                                                               |
+-                         An LSA Header                       -+
|                                                               |
+-                                                             -+
|                                                               |
+-                                                             -+
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              ...                              |
```

OSPF Link State Advertisement Header

```
 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
:                                                               :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            LS age             |    Options    |    LS type    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Link State ID                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Advertising Router                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     LS sequence number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         LS checksum           |             length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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.

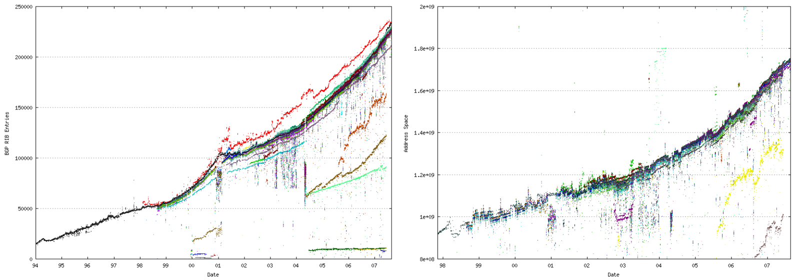
AS Categories (RFC 1772)

- *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

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                                                               +
|                           Marker                              |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Length               |      Type     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- 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

```
 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
+-+-+-+-+-+-+-+-+
|    Version    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Autonomous System Number  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Hold Time           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         BGP Identifier                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Opt Parm Len  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                       Optional Parameters                     |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Error code    | Error subcode |           Data                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- 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

- 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 collets 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.

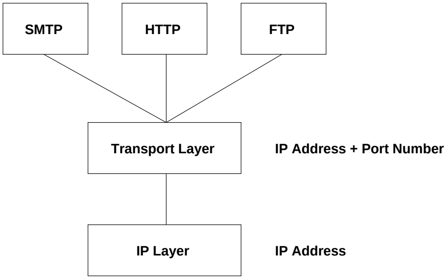
B. Quoitin, C. Pelsser, L. Swinnen, O. Bonaventure, and S. Uhlig.
Interdomain Traffic Engineering with BGP.
IEEE Communications Magazine, 41(5):122–128, May 2003.

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. Ehrenkranz.
BGP Routing Dynamics Revisited.
SIGCOMM Computer Communication Review, 37(2), April 2007.

Part: Internet Transport Layer (UDP, 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.

Pseudo Header

IPv4 Pseudo Header

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  unused (0)   |    Protocol   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- Pseudo headers are used during checksum computation.
- Excludes header fields that are modified by routers.

IPv6 Pseudo Header

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                         Source Address                        +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                      Destination Address                      +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Upper-Layer Packet Length                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      zero                     |  Next Header  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

User Datagram Protocol (UDP)

User Datagram Protocol (UDP)

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Length             |           Checksum            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- UDP (RFC 768) provides an unreliable datagram transport service.

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.

Transmission Control Protocol (TCP)

Transmission Control Protocol (TCP)

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Offset| Reserved  |   Flags   |            Window             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- 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.

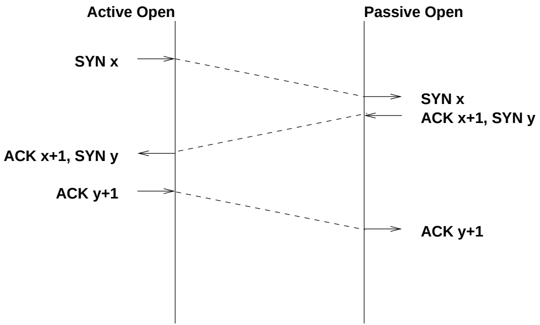
Transmission Control Protocol (TCP)

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

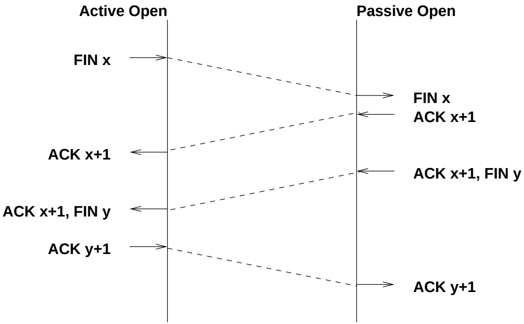
- The `Window` field indicates the number of data bytes which the sender of the segment is willing to receive.
- The `Checksum` field contains the Internet checksum computed over the pseudo header, the TCP header and the data contained in the TCP segment.
- The `Urgent Pointer` field points, relative 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.

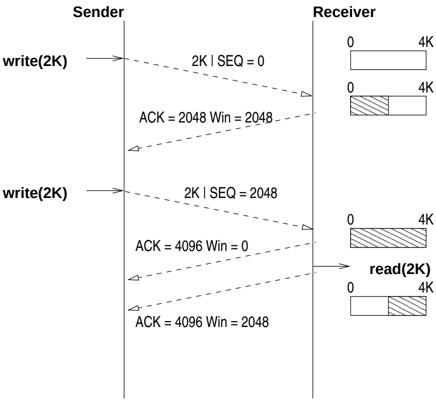
TCP State Machine (Part #1)

```
                             +---------+ ---------\      active OPEN
                             |  CLOSED |            \    -----------
                             +---------+<---------\   \   create TCB
                               |     ^              \   \  snd SYN
                  passive OPEN |     |   CLOSE        \   \
                  ------------ |     | ----------       \   \
                   create TCB  |     | delete TCB         \   \
                               V     |                      \   \
                             +---------+            CLOSE    |    \
                             |  LISTEN |          ---------- |     |
                             +---------+          delete TCB |     |
                  rcv SYN      |     |     SEND              |     |
                 -----------   |     |    -------            |     V
+---------+      snd SYN,ACK  /       \   snd SYN          +---------+
|         |<-----------------           ------------------>|         |
|   SYN   |                    rcv SYN                     |   SYN   |
|   RCVD  |<-----------------------------------------------|   SENT  |
|         |                  snd SYN, ACK                  |         |
|         |------------------           -------------------|         |
+---------+   rcv ACK of SYN  \       /  rcv SYN,ACK       +---------+
  |           --------------   |     |   -----------
  |                  x         |     |     snd ACK
  |                            V     V
  |  CLOSE                   +---------+
  | -------                  |  ESTAB  |
  | snd FIN                  +---------+
  :                   CLOSE    :     :    rcv FIN
```

TCP State Machine (Part #2)

```
  :                            :     :
  |                            V     V
  |  CLOSE                   +---------+
  | -------                  |  ESTAB  |
  | snd FIN                  +---------+
  |                   CLOSE    |     |    rcv FIN
  |                  -------   |     |    -------
  V                  snd FIN  /       \   snd ACK           +---------+
|  FIN    |<-----------------           ------------------->|  CLOSE  |
| WAIT-1  |------------------                               |   WAIT  |
+---------+          rcv FIN  \                             +---------+
  | rcv ACK of FIN   -------   |                             CLOSE  |
  | --------------   snd ACK   |                            ------- |
  V        x                   V                            snd FIN V
+---------+                  +---------+                    +---------+
|FINWAIT-2|                  | CLOSING |                    | LAST-ACK|
+---------+                  +---------+                    +---------+
  |                rcv ACK of FIN |                  rcv ACK of FIN |
  |  rcv FIN       -------------- |    Timeout=2MSL  -------------- |
  |  -------              x       V    ------------         x       V
   \ snd ACK                 +---------+delete TCB          +---------+
    ------------------------>|TIME WAIT|------------------->| CLOSED  |
                             +---------+                    +---------+
```

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.

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 (flight size) 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:

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

  SMSS is the sender maximum segment size, the size of the largest sement 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:

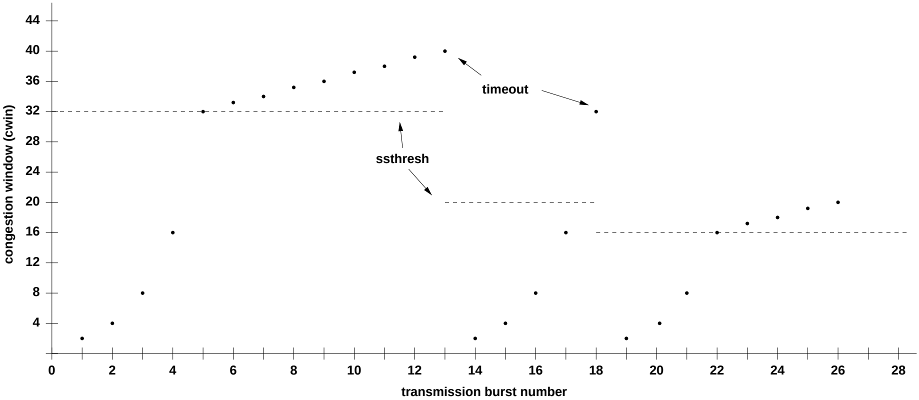
$$cwnd = cwnd + (SMSS * SMSS / cwnd)$$

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

- When congestion is noticed (the retransmission timer expires), then cwnd is reset to 1 full-sized segment and the slow start threshold *ssthresh* is updated as follows:

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

TCP Congestion Control (cont.)



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

Retransmission Timer

- The retransmission timer controls when a segment is resend 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; α is typicall $\frac{7}{8}$.
- The standard deviation is estimated using:

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

  α is a smoothing factor.
- The retransmission timeout RTO is determined as follows:

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

  The factor 4 has been choosen 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.

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.

$\Longrightarrow$ 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

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

  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:

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

TCP Performance (cont.)

- Example:
  - $RTT = 200ms, \; p = 0.05$
  - $\bar{X}(p) \approx 27.4pps$
  - With 1500 byte segments, the data rate becomes $\approx 32Kbps$
- Given this formula, it becomes clear that high data rates (say 10Gbps or 1 Tbps) require an extremely and unrealistic small packet error rate.
- TCP extensions to address this problem can be found in RFC 3649 [**?**].

References

J. Postel.
User Datagram Protocol.
RFC 768, ISI, August 1980.

J. Postel.
Transmission Control Protocol.
RFC 793, ISI, September 1981.

M. Allman, V. Paxson, and W. Stevens.
TCP Congestion Control.
RFC 2581, NASA Glenn/Sterling Software, ACIRI/ICSI, April 1999.

K. Ramakrishnan, S. Floyd, and D. Black.
The Addition of Explicit Congestion Notification (ECN) to IP.
RFC 3168, TeraOptic Networks, ACIRI, EMC, September 2001.

M. Hassan and R. Jain.
High Performance TCP/IP Networking.
Prentice Hall, 2004.

S. Floyd.
HighSpeed TCP for Large Congestion Windows.
RFC 3649, ICSI, December 2003.

Part: Firewalls and Network Address Translators

Middleboxes

Middleboxes

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.
- ...

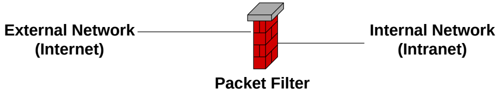
Firewalls

Firewalls

- 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".

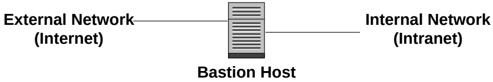
⇒ There are many ways to circumvent firewalls if internal and external hosts cooperate.

Firewall Architectures



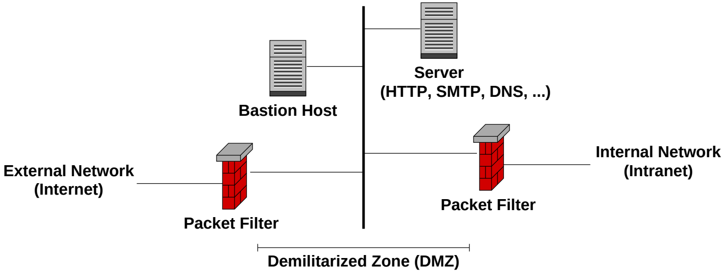
- The simplest architecture is a packet filter which is typically implemented within a router that connects the internal network with the external network.
- Sometimes called a “screening router”.

Firewall Architectures



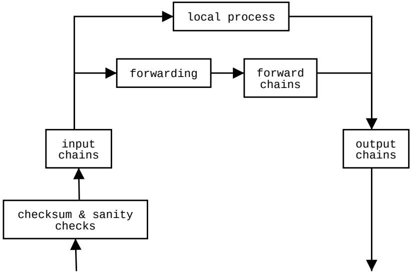
- A bastion host is a multihomed host connected to the internal and external network which does not forward IP datagrams but instead provides suitable gateways.
- Effectively prevents any direct communication between hosts on the internal network with hosts on the external network.

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 fiter rules applied to all forwarded IP packets.

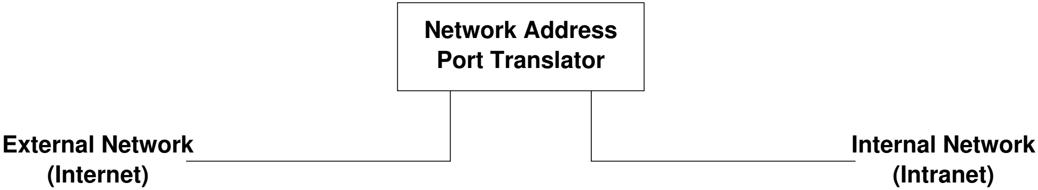
Network Address Translators

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



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.

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: Security at the Network and Transport Layer

Terminology (Cryptography)

- *Cryptology* subsumes cryptography and cryptanalysis:
  - *Cryptography* is the art of secret writing.
  - *Cryptanalysis* is the art of breaking ciphers.
- *Encryption* is the process of converting *plaintext* into an unreadable form, termed *ciphertext*.
- *Decryption* is the reverse process, recovering the plaintext back from the ciphertext.
- A *cipher* is an algorithm for encryption and decryption.
- A *key* is some secret piece of information used as a parameter of a cipher and customises the algorithm used to produce ciphertext.

Cryptosystem

Definition

A cryptosystem is a quintuple (M, C, K, E_k, D_k), where

- M is a cleartext space,
- C is a chiffretext space,
- K is a key space,
- $E_k : M \to C$ is an encryption transformation with $k \in K$, and
- $D_k : C \to M$ is a decryption transformation with $k \in K$.

For a given k and all $m \in M$, the following holds:

$$D_k(E_k(m)) = m$$

Cryptosystem Requirements

- The transformations E_k and D_k must be efficient to compute.
- It must be easy to find a key $k \in K$ and the functions E_k and D_k.
- The security of the system rests on the secrecy of the key and not on the secrecy of the algorithms.
- For a given $c \in C$, it is difficult to systematically compute
  - D_k even if $m \in M$ with $E_k(m) = c$ is known
  - a cleartext $m \in M$ such that $E_k(m) = c$.
- For a given $c \in C$, it is difficult to systematically determine
  - E_k even if $m \in M$ with $E_k(m) = c$ is known
  - $c' \in C$ with $c' \neq c$ such that $D_k(c')$ is a valid cleartext in M.

Symmetric vs. Asymmetric Cryptosystems

Symmetric Cryptosystems

- Both (all) parties share the same key and the key needs to be kept secret.
- Examples: AES, DES (outdated), Twofish, Serpent, IDEA, ...

Asymmetric Cryptosystems

- Each party has a pair of keys: one key is public and used for encryption while the other key is private and used for decryption.
- Examples: RSA, DSA, ElGamal, ...

Cryptographic Hash Functions

Definition

A cryptographic hash function H is a hash function which meets the following requirements:

1. The hash function is efficient to compute for arbitrary cleartexts m.
2. Given h it should be hard to find any m such that $h = H(m)$.
3. Given an input m_1, it should be hard to find another input $m_2 \neq m_1$ such that $H(m_1) = H(m_2)$.
4. It should be hard to find two different messages m_1 and m_2 such that $H(m_1) = H(m_2)$.

Cryptographic hash functions are used to compute a fixed size fingerprint (message digest) of a variable length clear text.

Digital Signatures

- Digital signatures are used to prove the authenticity of a message (or document) and its integrity.
  - The receiver can verify the claimed identity of the sender.
  - The sender can not deny that it did sent the message.
  - The receiver can not tamper the message itself.
- Digitally signing a message (or document) means that
  - the sender puts a signature into a message (or document) that can be verified and
  - that we can be sure that the signature cannot be faked (e.g., copied from some other message)
- Digital signatures are often implemented by signing a cryptographic hash of the original message (or document) since this is usually less computationally expensive

Diffie-Hellman

- Initialization:
  - Define a prime number p and a primitive root g of p with $g < p$. The numbers p and g can be made public.
- Exchange:
  - A picks $x_A \in \mathbb{Z}_p$ and computes $y_A = g^{x_A} \bmod p$. x_A is kept secret while y_A is sent to B.
  - B picks $x_B \in \mathbb{Z}_p$ and computes $y_B = g^{x_B} \bmod p$. x_B is kept secret while y_B is sent to A.
  - A computes:

$$K_{AB} = y_B^{x_A} \bmod p = (g^{x_B} \bmod p)^{x_A} \bmod p = g^{x_A x_B} \bmod p$$

  - B computes:

$$K_{AB} = y_A^{x_B} \bmod p = (g^{x_A} \bmod p)^{x_B} \bmod p = g^{x_A x_B} \bmod p$$

  - A and B now own a shared key K_{AB}.

Diffie-Hellman (cont.)

- A number g is a primitive root of p if the sequence $g^1 \bmod p, g^2 \bmod p, \ldots, g^{p-1} \bmod p$ produces the numbers $1, \ldots, p$ in any permutation.
- p should be choosen such that $(p-1)/2$ is prime as well.
- p should have a length of at least 512 bits.
- An attacker can play "man in the middle" (MIM) by claiming B's identity to A and A's identity to B.

Diffie-Hellman Example

- In this unrealistic example, we pick $n = 47$ and $g = 3$.
- A picks $x_A = 8$ and computes $y_A = 3^8 \bmod 47 = 28$.
- B picks $X_B = 10$ and computers $y_B = 3^{10} \bmod 47 = 17$.
- A sends $(47, 3, 28)$ while B sends $(47, 3, 17)$.
- A computes $K_{AB} = 17^{10} \bmod 47 = 4$.
- B computes $K_{AB} = 28^{10} \bmod 47 = 4$.

Internet Protocol Security (IPsec)

IPsec in a Nutshell

- Goal: Secure packets at the IP network layer
- Modes:
  - Transport mode:
    The IPsec header is inserted just after the IP header
  - Tunnel mode:
    A complete IP packet is encapsulated in a new IP packet
- A Security Association (SA) is a simplex association between two IPsec endpoints
- For duplex unicast communication, two SAs must be established

IPsec Headers

- The IP Authentication Header (AH) offers integrity and data origin authentication, with optional (at the discretion of the receiver) anti-replay features
- The Encapsulating Security Payload (ESP) protocol offers the same set of services, and also offers confidentiality
- Both AH and ESP offer access control, enforced through the distribution of cryptographic keys and the management of traffic flows as dictated by the Security Policy Database (SPD)
- The index into the SPD is called the SPI and typically used to identify SAs

IPsec AH (RFC 4302)

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Header   |  Payload Len  |          RESERVED             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Security Parameters Index (SPI)               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Sequence Number Field                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                Integrity Check Value-ICV (variable)           |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

IPsec ESP (RFC 4303)

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ ----
|               Security Parameters Index (SPI)                 | ^Int.
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |Cov-
|                      Sequence Number                          | |ered
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ | ----
|                    Payload Data* (variable)                   | |   ^
~                                                               ~ |   |
|                                                               | |Conf.
+               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |Cov-
|               |     Padding (0-255 bytes)                     | |ered*
+-+-+-+-+-+-+-+-+               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |   |
|                               |  Pad Length   | Next Header   | v   v
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ ------
|         Integrity Check Value-ICV   (variable)                |
~                                                               ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

IPsec Keying

- Manual keying (not really recommended)
- Automatic keying with the Internet Key Exchange (IKEv2) protocol (currently version 2).
  - IKEv2 itself uses the Diffie-Hellman algorithm plus certificates.
  - IKEv2 is a simplified version IKEv1 (which happened to be too complicated).
- No widespread deployment of an automatic keying infrastructure and as a consequence transport-mode IPsec is not widely used on the open Internet.
- Tunnel-mode IPsec has seen larger deployment in order to build secure tunnels into enterprise networks (also known as Virtual Private Networks, VPNs).

Manual Keying Example

```
#!/usr/sbin/setkey -f

# Clear the sa database and the spd database
flush;
spdflush;

# Traffic going from 212.201.49.188 to 10.70.17.11 needs an AH signed
# using HMAC-SHA1 using secret 12345678901234567890
add 212.201.49.188 10.70.17.11 ah 15700 -A hmac-sha1 "12345678901234567890";

# Traffic going from 212.201.49.188 to 10.70.17.11 needs encryption
# using 3des-cbc with key 123456789012123456789012'
add 212.201.49.188 10.70.17.11 esp 15701 -E 3des-cbc "123456789012123456789012";

# Traffic going out to 10.70.17.11 must be encrypted and be wrapped
# in an AH authentication header.
spdadd 212.201.49.188 10.70.17.11 any -P out ipsec
    esp/transport//require
    ah/transport//require;
```

Manual Keying Example

```
#!/usr/sbin/setkey -f

# Clear the sa database and the spd database
flush;
spdflush;

# Traffic going from 212.201.49.188 to 10.70.17.11 needs an AH signed
# using HMAC-SHA1 using secret 12345678901234567890
add 212.201.49.188 10.70.17.11 ah 15700 -A hmac-sha1 "12345678901234567890";

# Traffic going from 212.201.49.188 to 10.70.17.11 needs encryption
# using 3des-cbc with key 123456789012123456789012'
add 212.201.49.188 10.70.17.11 esp 15701 -E 3des-cbc "123456789012123456789012";

# Traffic coming in from 212.201.49.188 must be encrypted and wrapped
# in an AH authentication header.
spdadd 212.201.49.188 10.70.17.11 any -P in ipsec
    esp/transport//require
    ah/transport//require;
```

Transport Layer Security (TLS)

Transport Layer Security

- Transport Layer Security (TLS), formerly known as Secure Socket Layer (SSL), was created by Netscape in order to secure data transfers on the Web.
- As a user-space implementation, TLS can be shipped with applications and does not require operating system support.
- TLS uses X.509 certificates to authenticate servers and clients (although client authentication is often not used at the TLS layer).
- X.509 certificates more or less require a public key infrastructure including revocation server.
- TLS is widely used to secure application protocols running over TCP (e.g., http, smtp, ftp, telnet, imap, ...)

X.509 Certificate ASN.1 Definition

```
Certificate  ::=  SEQUENCE  {
     tbsCertificate       TBSCertificate,
     signatureAlgorithm   AlgorithmIdentifier,
     signatureValue       BIT STRING  }

TBSCertificate  ::=  SEQUENCE  {
     version         [0]  EXPLICIT Version DEFAULT v1,
     serialNumber         CertificateSerialNumber,
     signature            AlgorithmIdentifier,
     issuer               Name,
     validity             Validity,
     subject              Name,
     subjectPublicKeyInfo SubjectPublicKeyInfo,
     issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
                          -- If present, version MUST be v2 or v3
     subjectUniqueID [2]  IMPLICIT UniqueIdentifier OPTIONAL,
                          -- If present, version MUST be v2 or v3
     extensions      [3]  EXPLICIT Extensions OPTIONAL
                          -- If present, version MUST be v3
     }
```

X.509 Certificate ASN.1 Definition

```
Version  ::=  INTEGER  {  v1(0), v2(1), v3(2)  }

CertificateSerialNumber  ::=  INTEGER

Validity ::= SEQUENCE {
     notBefore      Time,
     notAfter       Time }

Time ::= CHOICE {
     utcTime        UTCTime,
     generalTime    GeneralizedTime }

UniqueIdentifier  ::=  BIT STRING

SubjectPublicKeyInfo  ::=  SEQUENCE  {
     algorithm            AlgorithmIdentifier,
     subjectPublicKey     BIT STRING  }

Extensions  ::=  SEQUENCE SIZE (1..MAX) OF Extension

Extension  ::=  SEQUENCE  {
     extnID      OBJECT IDENTIFIER,
     critical    BOOLEAN DEFAULT FALSE,
     extnValue   OCTET STRING
                 -- contains the DER encoding of an ASN.1 value
                 -- corresponding to the extension type identified
                 -- by extnID
     }
```

X.509 Subject Alternative Name Extension

```
id-ce-subjectAltName OBJECT IDENTIFIER ::=  { id-ce 17 }

SubjectAltName ::= GeneralNames

GeneralNames ::= SEQUENCE SIZE (1..MAX) OF GeneralName

GeneralName ::= CHOICE {
     otherName                       [0]     OtherName,
     rfc822Name                      [1]     IA5String,
     dNSName                         [2]     IA5String,
     x400Address                     [3]     ORAddress,
     directoryName                   [4]     Name,
     ediPartyName                    [5]     EDIPartyName,
     uniformResourceIdentifier       [6]     IA5String,
     iPAddress                       [7]     OCTET STRING,
     registeredID                    [8]     OBJECT IDENTIFIER }

OtherName ::= SEQUENCE {
     type-id    OBJECT IDENTIFIER,
     value      [0] EXPLICIT ANY DEFINED BY type-id }

EDIPartyName ::= SEQUENCE {
     nameAssigner            [0]     DirectoryString OPTIONAL,
     partyName               [1]     DirectoryString }
```

TLS Record Protocol

Record Protocol

The record protocol takes messages to be transmitted, fragments the data into manageable blocks, optionally compresses the data, adds a message authentication code, and encrypts and transmits the result. Received data is decrypted, verified, decompressed, reassembled, and then delivered to higher-level clients.

- The record layer is used by the handshake protocol, the change cipher spec protocol, the alert protocol, and the application data protocol.
- The fragmentation and reassembly provided does not preserve application message boundaries.

TLS Handshake Protocol

Handshake Protocol

- Exchange messages to agree on algorithms, exchange random numbers, and check for session resumption.
- Exchange the necessary cryptographic parameters to allow the client and server to agree on a premaster secret.
- Exchange certificates and cryptographic information to allow the client and server to authenticate themselves.
- Generate a master secret from the premaster secret and the exchanged random numbers.
- Provide security parameters to the record layer.
- Allow client and server to verify that the peer has calculated the same security parameters and that the handshake completed without tampering by an attacker.

TLS Change Cipher Spec Protocol

Change Cipher Spec Protocol

The change cipher spec protocol is used to signal transitions in ciphering strategies.

- The protocol consists of a single ChangeCipherSpec message.
- This message is sent by both the client and the server to notify the receiving party that subsequent records will be protected under the newly negotiated CipherSpec and keys.

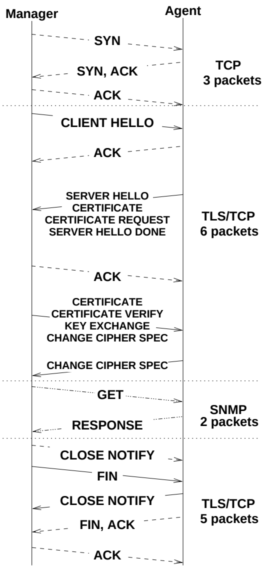
TLS Alert Protocol

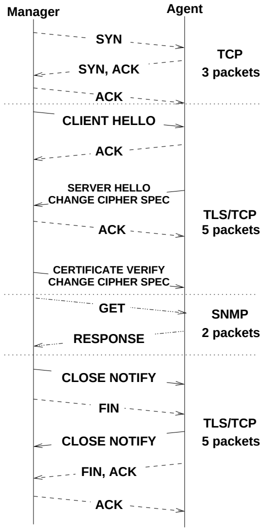
Alert Protocol

The alert protocol is used to signal exceptions (warnings, errors) that occured during the processing of TLS protocol messages. An alert has an alert level and an alert description (enumeration).

- The alert protocol is used to properly close a TLS connection by exchanging `close_notify` alert messages.
- The closure exchange allows to detect truncation attacks.

Details of an SNMP GET Operation over TLS





Secure Shell (SSH)

- SSH provides a secure connection through which user authentication and several inner protocols can be run.
- The general architecture of SSH is defined in RFC 4251.
- SSH was initially developed by Tatu Ylonen at the Helsinki University of Technology in 1995, who later founded SSH Communications Security.
- SSH was quickly adopted as a replacement for insecure remote login protocols such as telnet or rlogin/rsh.
- Several commercial and open source implementations are available running on almost all platforms.
- SSH is a Proposed Standard protocol of the IETF since 2006.

SSH Protocol Overview

SSH Protocol Layers

1. The **Transport Layer Protocol** provides server authentication, confidentiality, and integrity with perfect forward secrecy
2. The **User Authentication Protocol** authenticates the client-side user to the server
3. The **Connection Protocol** multiplexes the encrypted data stream into several logical channels

$\Rightarrow$ SSH authentication is not symmetric!

$\Rightarrow$ The SSH protocol is designed for clarity, not necessarily for efficiency (shows its academic roots)

SSH Terminology

Host Key

Every machine must have a public/private host key pair. Host Keys are often identified by their fingerprint.

User Key

Users may have their own public/private key pairs.

User Password

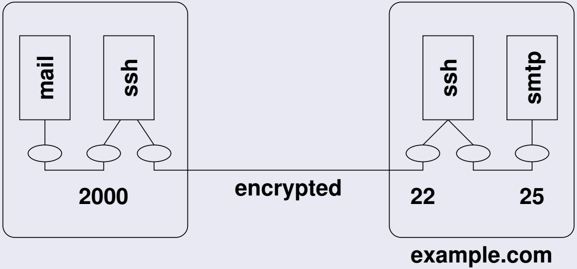
Accounts may have passwords to authenticate users.

Passphrase

The storage of a user's private key may be protected by a passphrase.

SSH Features: TCP Forwarding

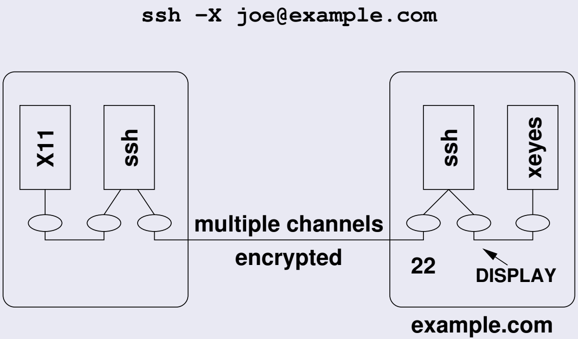
```
ssh -f joe@example.com -L 2000:example.com:25 -N
```



TCP Forwarding

TCP forwarding allows users to tunnel unencrypted traffic through an encrypted SSH connection.

SSH Features: X11 Forwarding

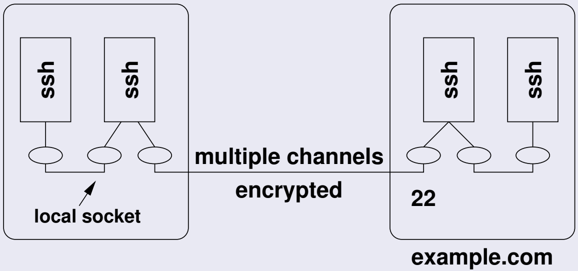


X11 Forwarding

X11 forwarding is a special application of TCP forwarding allowing X11 clients on remote machines to access the local X11 server (managing the display and the keyboard/mouse).

SSH Features: Connection Sharing

`ssh joe@example.com`

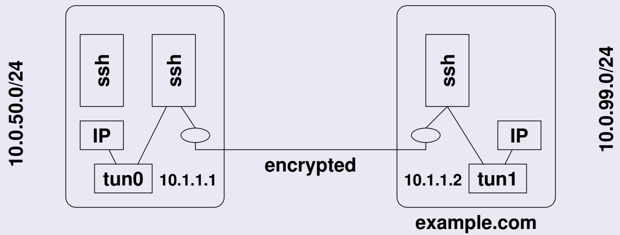


Connection Sharing

New SSH connections hook as a new channel into an existing SSH connection, reducing session startup times (speeding up shell features such as tab expansion).

SSH Features: IP Tunneling

```
ssh -f -w 0:1 example.com
```



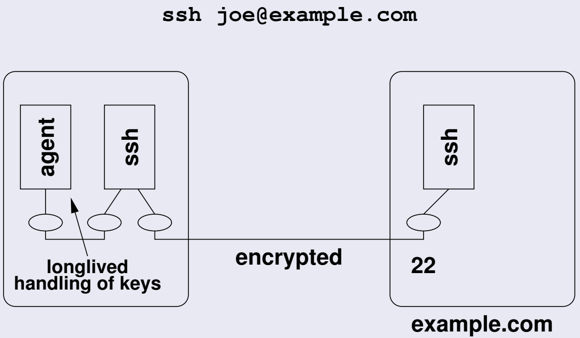

```
ifconfig tun0 10.1.1.1 10.1.1.2 \
         netmask 255.255.255.255
route add 10.0.99.0/24 10.1.1.2
```

```
ifconfig tun0 10.1.1.2 10.1.1.1 \
         netmask 255.255.255.255
route add 10.0.50.0/24 10.1.1.1
```

IP Tunneling

Tunnel IP packets over an SSH connection by inserting tunnel interfaces into the kernels and by configuring IP forwarding.

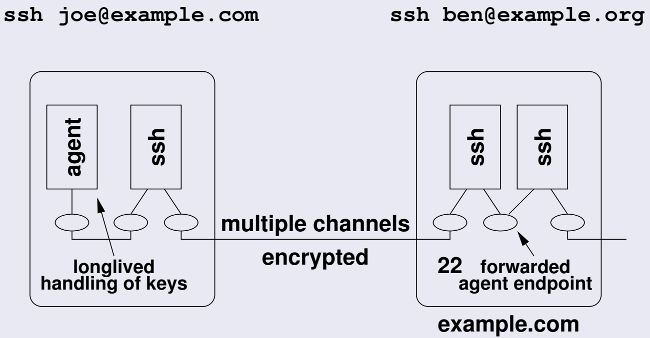
SSH Features: SSH Agent



SSH Agent

Maintains client credentials during a login session so that credentials can be reused by different SSH invocations without further user interaction.

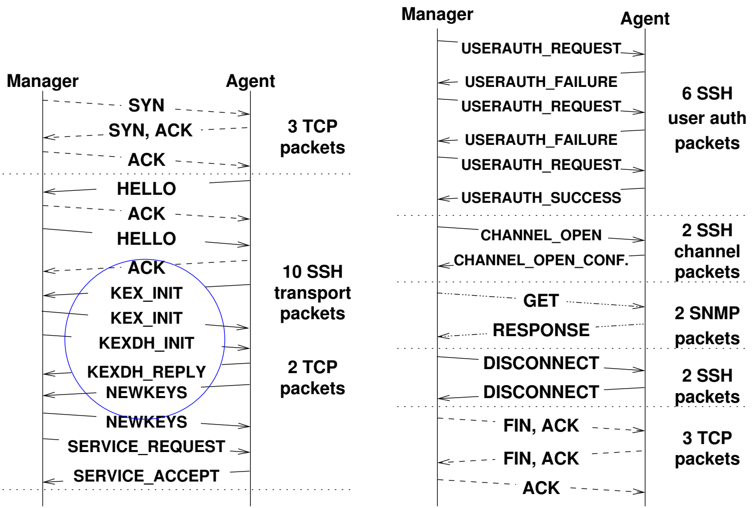
SSH Features: SSH Agent Forwarding



SSH Agent Forwarding

An SSH server emulates an SSH Agent and forwards requests to the SSH Agent of its client, creating a chain of SSH Agent delegations.

Details of an SNMP GET Operation over SSH



SSH Transport Protocol

- Transport Protocol (RFC 4253) provides
  - strong encryption,
  - server authentication,
  - integrity protection, and
  - optionally compression.
- SSH transport protocol typically runs over TCP
- 3DES (required), AES128 (recommended)
- HMAC-SHA1 (recommended)
- Automatic key re-exchange, usually after 1 GB of data have been transferred or after 1 hour has passed, whichever is sooner.

SSH Key Exchange

- The SSH host key keyex identifies a server by its hostname or IP address and possibly port number.
- Other keyex mechanisms use different naming schemes for a host.
- Different key exchange algorithms
  - Diffie-Hellman style key exchange
  - GSS-API style key exchange
- Different Host key algorithms
  - Host key used to authenticate key exchange
  - SSH RSA and DSA keys
  - X.509 (under development)

SSH User Authentication

- Executes after transport protocol initialization (key exchange) to authenticate client.
- Authentication methods:
  - Password (classic password authentication)
  - Interactive (challenge response authentication)
  - Host-based (uses host key for user authentication)
  - Public key (usually DSA or RSA keypairs)
  - GSS-API (Kerberos / NETLM authentication)
  - X.509 (under development)
- Authentication is client-driven.

SSH Connection Protocol

- Allows opening of multiple independent channels.
- Channels may be multiplexed in a single SSH connection.
- Channel requests are used to relay out-of-band channel specific data (e.g., window resizing information).
- Channels commonly used for TCP forwarding.

OpenSSH Implementation

Privilege Separation

Privilege separation is a technique in which a program is divided into parts which are limited to the specific privileges they require in order to perform a specific task.

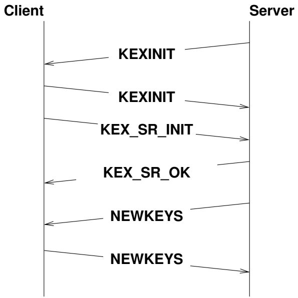
- OpenSSH is using two processes: one running with special privileges and one running under normal user privileges
- The process with special privileges carries out all operations requiring special permissions.
- The process with normal user privileges performs the bulk of the computation not requiring special rights.
- Bugs in the code running with normal user privileges do not give special access rights to an attacker.

Performance of Short Lived Sessions

Protocol	Time (meat) [ms]			Time (turtle) [ms]			Data [bytes]	Packets
	min	avg	max	min	avg	max		
v1/CSM/UDP/nn	0.24	0.25	0.29	0.85	0.95	1.43	292	2
v1/CSM/TCP/nn	0.39	0.40	0.43	1.27	1.38	1.72	1012	10
v2/CSM/UDP/nn	0.24	0.25	0.30	0.85	0.96	1.50	292	2
v2/CSM/TCP/nn	0.46	0.48	0.58	1.28	1.46	2.40	1012	10
v3/USM/UDP/nn	0.48	0.48	0.54	1.75	1.84	1.95	718	4
v3/USM/TCP/nn	0.63	0.64	0.69	2.22	2.46	9.59	1490	12
v3/USM/UDP/an	0.50	0.63	0.87	1.79	1.89	2.34	742	4
v3/USM/TCP/an	0.65	0.66	0.70	2.21	2.31	2.48	1514	12
v3/USM/UDP/ap	0.51	0.52	0.59	1.88	2.05	4.17	763	4
v3/USM/TCP/ap	0.66	0.68	0.71	2.31	2.42	2.60	1535	12
v3/TSM/SSH/ap	**13.49**	**13.73**	**14.20**	**107.35**	**110.45**	**144.33**	**5310**	**31**
v3/TSM/TLS/ap	**11.01**	**11.15**	**12.57**	**67.44**	**68.70**	**86.59**	**4107**	**16**
v3/TSM/DTLS/ap	10.89	11.05	12.00	67.68	69.96	155.10	3457	8
v3/TSM/TLSsr/ap	**2.23**	**2.27**	**2.45**	**5.47**	**5.72**	**6.28**	**1457**	**15**

- SSH (TLS/DTLS) transports behave like a DoS attack for short-lived SNMP sessions (e.g., shell scripts)
- TLS's session resumption mechanism cures the problem
- How can we do session resumption with SSH?

Session Resumption Key Exchange



- Server maintains session state for recently closed sessions
- Client and server perform session resumption by using of a session resumption key exchange algorithm
- SSH's algorithm negotiation feature handles this nicely

Session Resumption with Server Side State

Algorithm (Server Side State)

C: Client sends the session identifier and a MAC computed over the session keys to the server in a `SSH2_MSG_KEXSR_INIT` message

S: Server looks up the cached session and verifies the MAC

- If successful, it returns an `SSH2_MSG_KEX_SR_OK` message, followed by a standard `SSH2_MSG_NEWKEYS` exchange
- On failure, `SSH2_MSG_KEX_SR_ERROR` is sent and key exchange proceeds with another key exchange algorithm, or fails

\+ Simple design and easy to implement

− Server has to maintain session state (scalability)

Session Resumption with Client Side State

Algorithm (Client Side State)

S: After key (re)negotiation, the server sends an encrypted ticket in a `SSH2_MSG_KEX_SR_TICKET` message

C: The client sends the encrypted ticket and a MAC computed over the session identifier to the server in a `SSH2_MSG_KEXSR_INIT` message

S: The server decrypts the ticket and verifies the MAC

- If successful, it returns an `SSH2_MSG_KEX_SR_OK` message, followed by a standard `SSH2_MSG_NEWKEYS` exchange.
- On failure, `SSH2_MSG_KEX_SR_ERROR` is sent and key exchange proceeds with another key exchange algorithm, or fails.

+ Server side state reduced to a key for encrypting tickets

TicketContent Data Structure

```
struct TicketEnc {
    char* name;
    u_char* key;
    u_char* iv;
};

struct TicketMac {
    char* name;
    u_char* key;
};
```

```
struct TicketContent {
    u_char* session_id;
    u_int session_id_len;
    TicketEnc tenc_ctos;
    TicketEnc tenc_stoc;
    TicketMac tmac_ctos;
    TicketMac tmac_stoc;
    char* tcomp_ctos;
    char* tcomp_stoc;
    int hostkey_type;
    char* client_version_string;
    char* server_version_string;
};
```

- SSH allows to use different algorithms in each direction!

Ticket Data Structure

```
struct Ticket {
    u_int seq_nr;
    u_char* id;
    u_char* enc_ticket;
    u_int enc_ticket_len;
    int64_t time_stamp;
};
```

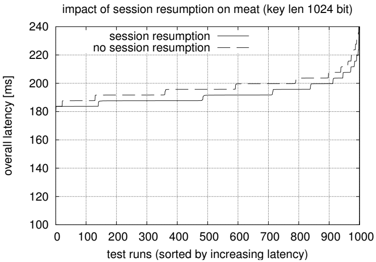
- Contains the encrypted `TicketContent` data structure in `enc_ticket`
- The `id` uniquely identifiers a ticket
- The `seq_nr` and `time_stamp` fields can be used to quickly discard outdated tickets
- Encryption key and its IV are generated at server start-up

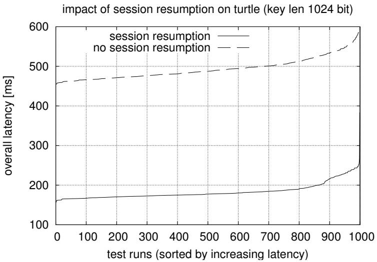
Performance Evaluation

Name	CPUs	RAM	Ethernet	Kernel
`meat`	2 Xeon 3 GHz	2 GB	1 Gbps	2.6.16.29
`veggie`	2 Xeon 3 GHz	1 GB	1 Gbps	2.6.16.29
`turtle`	1 Ultra Sparc IIi	128 MB	100 Mbps	2.6.20

- SSH client: `veggie` / SSH server: `meat` and `turtle`
- Measuring overall execution time of "`ssh $host exit`"
- Used HMAC-MD5 hash function and AES-128 encryption
- Hosts and the network were idle during the experiments
- 1000 experiments, results sorted by the measured latency
- Absolute numbers irrelevant, look at relative numbers

Session Resumption Performance (key length 1024)

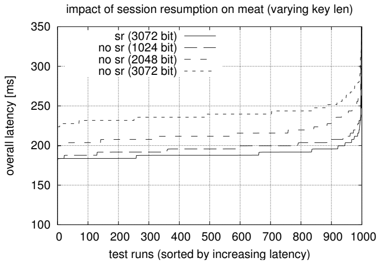




- With a key length of 1024 bits, the performance gain on an idle fast machine is observable but small
- With the same key length, the performance gain on a small idle machine is significant (factor 4)

$\Rightarrow$ Session resumption is particularly useful for processing power constrained low-end consumer /enterprise products

Impact of the Key Length on the Performance



- Session resumption performance is largely independent of the key length
- With increasing key length, the performance gain increases also on fast idle machines

⇒ Even on a fast processors, the performance gain is significant if you need long keys to achieve strong security

References I

S. Kent and K. Seo.
Security Architecture for the Internet Protocol.
RFC 4301, BBN Technologies, December 2005.

S. Kent.
IP Authentication Header.
RFC 4302, BBN Technologies, December 2005.

S. Kent.
IP Encapsulating Security Payload (ESP).
RFC 4303, BBN Technologies, December 2005.

T. Ylonen and C. Lonvick.
The Secure Shell (SSH) Protocol Architecture.
RFC 4251, SSH Communications Security Corp, Cisco Systems, January 2006.

T. Ylonen and C. Lonvick.
The Secure Shell (SSH) Authentication Protocol.
RFC 4252, SSH Communications Security Corp, Cisco Systems, January 2006.

T. Ylonen and C. Lonvick.
The Secure Shell (SSH) Transport Layer Protocol.
RFC 4253, SSH Communications Security Corp, Cisco Systems, January 2006.

References II

T. Ylonen and C. Lonvick.
The Secure Shell (SSH) Connection Protocol.
RFC 4254, SSH Communications Security Corp, Cisco Systems, January 2006.

J. Schönwälder, G. Chulkov, E. Asgarov, and M. Cretu.
Session Resumption for the Secure Shell Protocol.
In *Proc. 11th IFIP/IEEE International Symposium on Integrated Network Management (IM 2009)*, pages 157–163, May 2009.

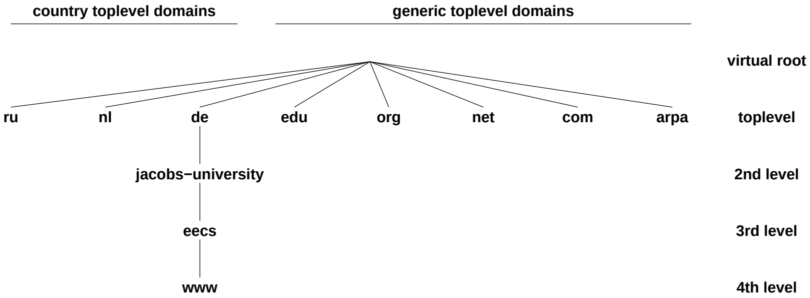
T. Dierks and E. Rescorla.
The Transport Layer Security (TLS) Protocol Version 1.2.
RFC 5246, Independent, RTFM, August 2008.

E. Rescorla and N. Modadugu.
Datagram Transport Layer Security.
RFC 4347, RTFM, Stanford University, April 2006.

D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk.
Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.
RFC 5280, NIST, Microsoft, Trinity College Dublin, Entrust, Vigil Security, May 2008.

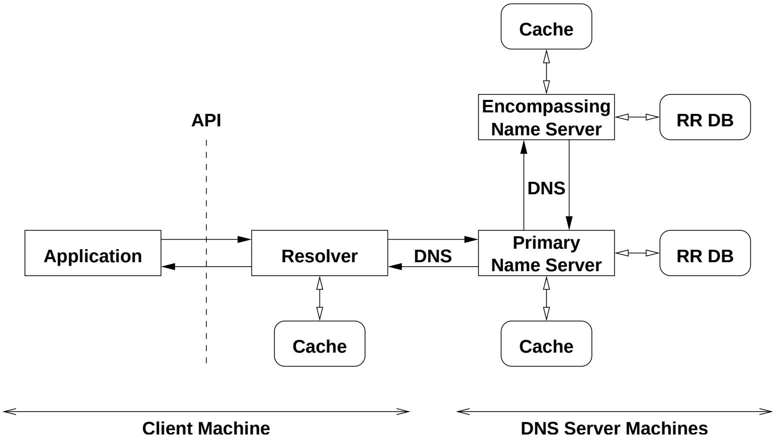
Part: Domain Name System (DNS)

Domain Name System (DNS)



- The Domain Name System (DNS) [RFC 1034, RFC 1035] provides a global infrastructure to map human friendly domain names into IP addresses and vice versa.
- 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`.

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)

Message Formats

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

```
  0                   1
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      ID                       |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|QR|   Opcode  |AA|TC|RD|RA| Z|AD|CD|   RCODE   |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    QDCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ANCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    NSCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ARCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

- 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

```
  0                       1
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
:                     QNAME                     :
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                     QTYPE                     |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                     QCLASS                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

DNS Response Format

```
  0                       1
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
:                     NAME                      :
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                     TYPE                      |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                     CLASS                     |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                     TTL                       |
|                                               |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                   RDLENGTH                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--|
:                     RDATA                     :
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

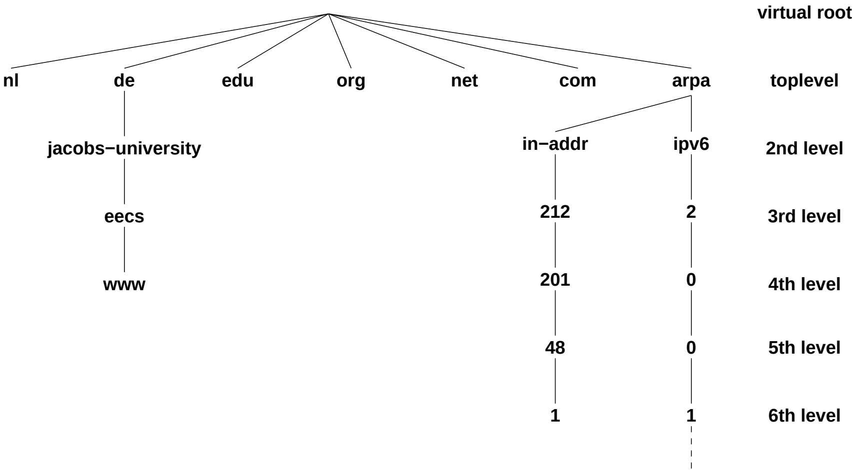
Resource Record Formats

- 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 $h_1h_2h_3h_4 : \ldots : h_{13}h_{14}h_{15}h_{16}$, a PTR resource record is created for the pseudo domain name $h_{16}.h_{15}.h_{14}.h_{13}.\ldots.h_4.h_3.h_2.h_1.ip6.arpa$

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).

Security and Dynamic Updates

DNS Security

- 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.

Creative Usage

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 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 RFC5782.

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.

B. Wellington.
Secure Domain Name System (DNS) Dynamic Update.
RFC 3007, Nominum, November 2000.

References II

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: Augmented Backus Naur Form (ABNF)

Basics, Rule Names, Terminal Symbols

ABNF Basics

- 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 alphabetics, 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         ; ASCCI digits (0 - 9)

ABA   = "aba"           ; ASCII string "aba" or "ABA" or "Aba" or ...

abba  = %x61.62.62.61   ; ASCII string "abba"
```

Operators

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`

ABFN Operators

- 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)`

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
```

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
```

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` match only “aBc”.

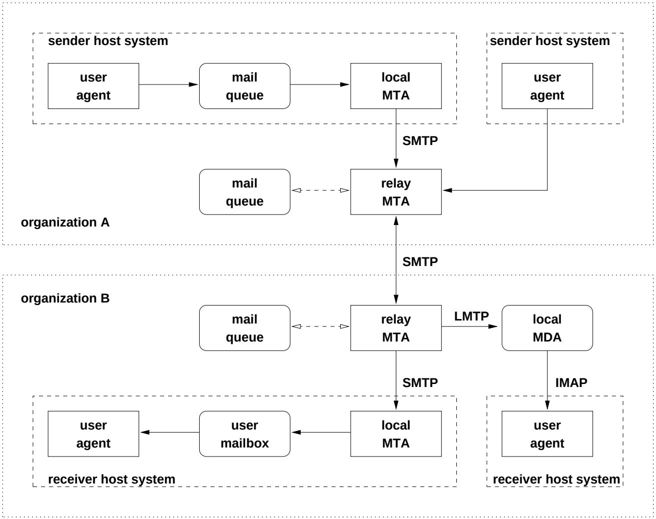
References

D. Crocker and P. Overell.
Augmented BNF for Syntax Specifications: ABNF.
RFC 5234, Brandenburg InternetWorking, THUS plc., January 2008.

P. Kyzivat.
Case-Sensitive String Support in ABNF.
RFC 7405, December 2014.

Part: Electronic Mail (SMTP, IMAP)

Components Involved in Electronic Mail



Terminology

- 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)

Simple Mail Transfer Protocol (SMTP)

Simple Mail Transfer Protocol (SMTP)

- Defined in RFC 5321 (originally RFC 821)
- Textual client/server protocol running over TCP
- 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
- Fully specified in ABNF

SMTP Commands

HELO	Indentify clients to a SMTP server (HELLO)
EHLO	Extended identification (EXTENDED HELLO)
MAIL	Inititate a mail transaction (MAIL)
RCPT	Identity 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:" ("<Postmaster@" domain ">" / "<Postmaster>" /
                   Forward-Path) [SP Rcpt-Parameters CRLF
data = "DATA" CRLF
rset = "RSET" CRLF
vrfy = "VRFY" SP String CRLF
expn = "EXPN" SP String CRLF
help = "HELP" [ SP String ] CRLF
noop = "NOOP" [ SP String ] CRLF
quit = "QUIT" CRLF
```

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

resend-field   =  resent-date / resent-from / resent-sender
resend-field   =/ resent-to / resent-cc / resent-bcc
resend-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.

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
```

Multipurpose Internet Mail Extensions (MIME)

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


```
CkRhdGU6IFR1ZSwgMSBBcHIgMTk5NyAwOTowNjozMSAtMDgwMCAoUFNUKQpGcm9t
OiBjb3lvdGVAZGVzZXJ0LmV4YW1wbGUub3JnClRvOiByb2FkcnVubmVyQGFjbWUu
ZXhhbXBsZS5jb20KU3ViamVjdDogSSBoYXZlIGEgcHJlc2VudCBmb3IgeW91CgpM
b29rLCBJJ20gc29ycnkgYWJvdXQgdGhlIHdob2xlIGFudmlsIHRoaW5nLCBhbmQg
SSByZWFsbHkKZGlkbid0IG1lYW4gdG8gdHJ5IGFuZCBkcm9wIGl0IG9uIHlvdSBm
cm9tIHRoZSB0b3Agb2YgdGhlCmNsaWZmLiAgSSB3YW50IHRvIHRyeSB0byBtYWtl
IGl0IHVwIHRvIHlvdS4gIEkndmUgZ290IHNvbWUKZ3JlYXQgYmlyZHNlZWQgb3Zl
ciBoZXJlIGF0IG15IHBsYWNlLS10b3Agb2YgdGhlIGxpbmUKc3R1ZmYtLWFuZCBp
ZiB5b3UgY29tZSBieSwgSSdsbCBoYXZlIGl0IGFsbCB3cmFwcGVkIHVwCmZvciB5
b3UuICBJJ20gcmVhbGx5IHNvcnJ5IGZvciBhbGwgdGhlIHByb2JsZW1zIEkndmUg
Y2F1c2VkCmZvciB5b3Ugb3ZlciB0aGUgeWVhcnMsIGJ1dCBJIGtub3cgd2UgY2Fu
IHdvcmsgdGhpcyBvdXQuCi0tCldpbGUgRS4gQ295b3RlICAgIlN1cGVyIEdlbml1
cyIgICBjb3lvdGVAZGVzZXJ0LmV4YW1wbGUub3JnCg==
```


- 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.

Internet Message Access Protocol (IMAP)

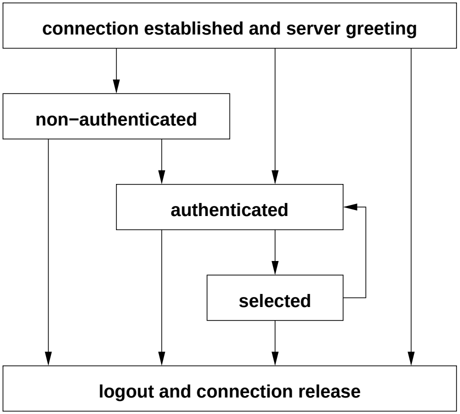
Internet Message Access Protocol (IMAP)

- 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 extend 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 negotation to protect the channel

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

IMAP Commands

- 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.

IMAP Commands

- 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 en 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 for unsuccessful completion of a comment 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
```

Filtering of Messages (SIEVE)

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 of 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

- 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;
}
```

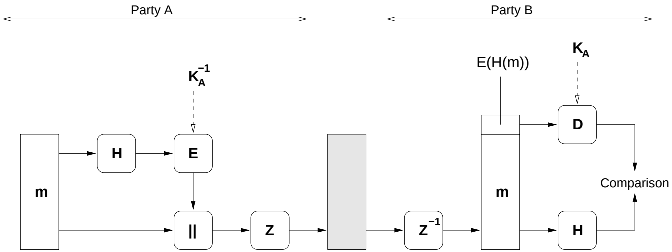
SIEVE Example (cont.)

```
# 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";
}
```

Pretty Good Privacy (PGP)

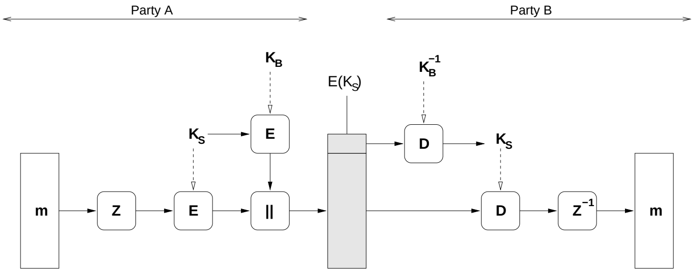
- PGP was developed by Philip Zimmerman in 1991 and is rather famous because PGP also demonstrated why patent laws and export laws in a globalized world need new interpretations.
- There are nowadays several independent PGP implementations.
- The underlying PGP specification is now called open PGP (RFC 4880).
- The competitor to PGP is S/MIME.

PGP Signatures



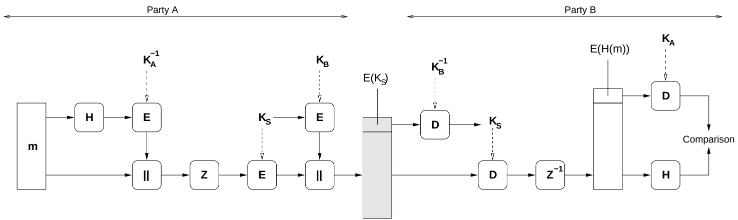
- A computes $c = Z(E_{K_A^{-1}}(H(m))||m)$
- B computes $Z^{-1}(c)$, splits the message and checks the signature by computing $D_{K_A}(E_{K_A^{-1}}(H(m)))$ and then checking the hash $H(m)$.
- PGP uses the hash-function MD5, the public-key algorithm RSA and zlib compression.

PGP Confidentiality



- A encrypts the message using the key K_s generated by the sender and appended to the encrypted message.
- The key K_s is protected by encrypting it with the public key K_B.
- Symmetric encryption is fast while public-key algorithm make it easier to exchange keys.

PGP Signatures and Confidentiality



- Signature and confidentiality can be combined as shown above.
- PGP uses in addition Radix-64 encoding to ensure that the messages can be represented in the ASCII code.
- PGP supports segmentation/reassembly functions for very large messages.

PGP Key Management

- Keys are maintained in so called key rings (one for public keys and one for private keys).
- Key generation utilizes various sources of random information (`/dev/random` if available) and symmetric encryption algorithms to generate good key material.
- So called "key signing parties" are used to sign keys of others and to establish a "web of trust" in order to avoid centralized certification authorities.

PGP Private Key Ring

Timestamp	Key ID	Public Key	Encrypted Private Key	User ID
⋮	⋮	⋮	⋮	⋮
T_i	$K_i \bmod 2^{64}$	K_i	$E_{H(P_i)}(K_i^{-1})$	User_i
⋮	⋮	⋮	⋮	⋮

- Private keys are encrypted using $E_{H(P_i)}()$, which is a symmetric encryption function using a key which is derived from a hash value computed over a user supplied passphrase.
- The Key ID is taken from the last 64 bits of the key K_i.

PGP Public Key Ring

Timestamp	Key ID	Public Key	Owner Trust	User ID	Signatures	Sig. Trust(s)
⋮	⋮	⋮	⋮	⋮	⋮	⋮
T_i	$K_i \bmod 2^{64}$	K_i	otrust_i	User_i	...	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮

- Keys in the public key ring can be signed by multiple parties. Every signature has an associated trust level:
  1. undefined trust
  2. usually not trusted
  3. usually trusted
  4. always trusted
- Computing a trust level for new keys which are signed by others (trusting others when they sign keys).

Cryptographic Message Syntax

- The Cryptographic Message Syntax (CMS) is a syntax used to digitally sign, digest, authenticate, or encrypt arbitrary message content.
- CMS supports different content types:
  - Data: plain data, typically encapsulated in the content types explained below
  - SignedData: content of any type with zero or more signatures
  - EnvelopedData: encrypted content of any type and encrypted content-encryption keys for one or more recipients
  - DigestedData: content of any type and a message digest of the content
  - EncryptedData: encrypted content of any type
  - AuthenticatedData: content of any type, a message authentication code (MAC), and encrypted authentication keys for one or more recipients
- CMS is defined using ASN.1 [**?**].

Secure MIME

- The `application/pkcs7-mime` MIME type is used to carry CMS content types including
  - EnvelopedData
  - SignedData
  - CompressedData
- The signature is usually carried together with the message in a multipart/signed message.

S/MIME Signed Message Example

```
Content-Type: multipart/signed;
   protocol="application/pkcs7-signature";
   micalg=sha1; boundary=boundary42

--boundary42
Content-Type: text/plain

This is a clear-signed message.

--boundary42
Content-Type: application/pkcs7-signature; name=smime.p7s
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7s

ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6
4VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj
n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
7GhIGfHfYT64VQbnj756

--boundary42--
```

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@eng.example.net|To:joe@example.com|
     Subject:demo=20run|Date:July=205,=202005=203:44:08=20PM=20-0700;
   bh=MTIzNDU2Nzg5MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTI=;
   b=dzdVyOfAKCdLXdJOc9G2q8LoXSlEniSbav+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=2jUSOH9NhtVGCQWNr9BrIAPreKQjO6Sn7XIkfJVOzv8=;
      b=AuUoFEfDxTDkHlLXSZEpZj79LICEps6eda7W3deTVFOk4yAUoqOB
        4nujc7YopdG5dWLSdNg6xNAZpOPr+kHxt1IrE+NahM6L/LbvaHut
        KVdkLLkpVaVVQPzeRDI009SO2Il5Lu7rDNH6mZckBdrIx0orEtZV
        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.
```

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=2jUSOH9NhtVGCQWNr9BrIAPreKQjO6Sn7XIkfJVOzv8=;
      b=AuUoFEfDxTDkHlLXSZEpZj79LICEps6eda7W3deTVFOk4yAUoqOB
        4nujc7YopdG5dWLSdNg6xNAZpOPr+kHxt1IrE+NahM6L/LbvaHut
        KVdkLLkpVaVVQPzeRDI009SO2Il5Lu7rDNH6mZckBdrIx0orEtZV
        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.

P. Guenther and T. Showalter.
Sieve: A Mail Filtering Language.
RFC 5228, Sendmail Inc., Mirapoint Inc., January 2008.

References II

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. Donnerhacke, H. Finney, D. Shaw, and R. Thayer.
OpenPGP Message Format.
RFC 4880, PGP Corporation, IKS GmbH, November 2007.

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.

References III

W. Stallings.
Comprehensive Internet E-Mail Security.
The Internet Protocol Journal, 19(3), November 2016.

Part: Hypertext Transfer Protocol (HTTP)

Overview

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.

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 ] "::"              ls32
              / [ *5( h16 ":" ) 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    = "!" / "$" / "&" / "'" / "(" / ")"
              / "*" / "+" / "," / ";" / "="
```

HTTP 1.1 Methods

- `OPTIONS`
  - Request information about the communication options available
- `GET`
  - Retrieve whatever information is identified by a URI
- `HEAD`
  - Retrieve only the meta-information which is identified by a URI
- `POST`
  - Annotate an existing resource or pass data to a data-handling process

HTTP 1.1 Methods (cont.)

- `PUT`
  - Store information under the supplied URI
- `DELETE`
  - Delete the resource identified by the URI
- `TRACE`
  - Application-layer loopback of request messages for testing purposes
- `CONNECT`
  - Initiate a tunnel such as a TLS or SSL tunnel

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>
```

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

- Negotation 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

- 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., IPP or SOAP).
  - There are some pitfalls with this approach as documented in RFC 3205.

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. Douglis, 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.

R. Fielding, M. Nottingham, and J. Reschke.
Hypertext Transfer Protocol (HTTP/1.1): Caching.
RFC 7234, Adobe, Akamai, greenbytes, June 2014.

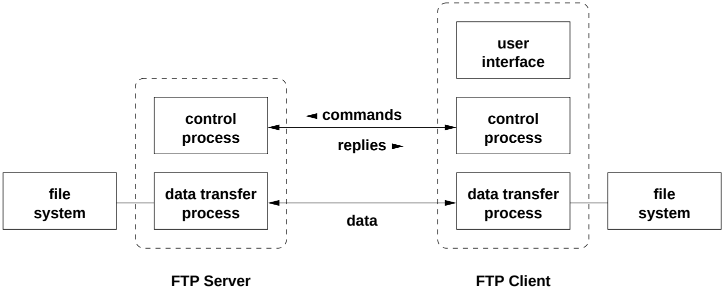
References III

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: 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.

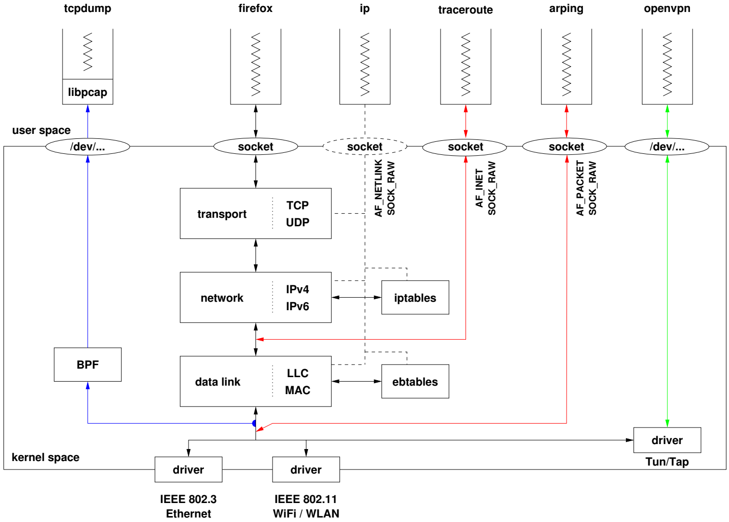
M. Allman, S. Ostermann, and C. Metz.
FTP Extensions for IPv6 and NATs.
RFC 2428, NASA Lewis/Sterling Software, Ohio University, The Inner Net, September 1998.

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

Part: Appendix

54 Packet Capturing

Linux Network Stack Overview



Motivation and Requirements

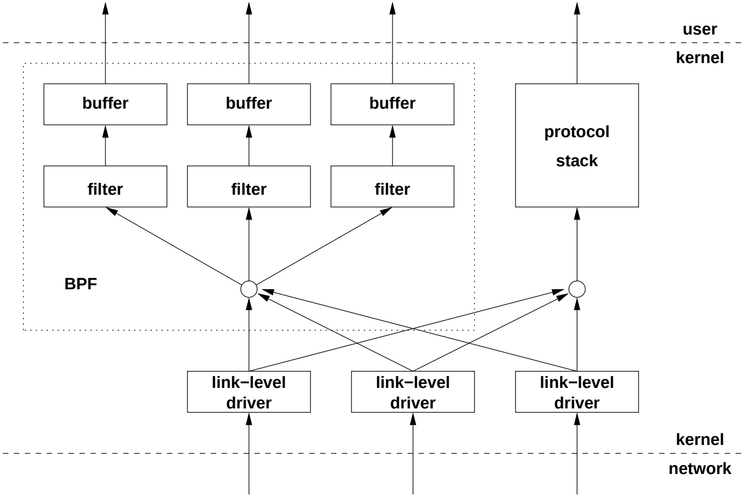
- Network protocol analyzers like `wireshark`, `tcpdump` or `ngrep` running in user space want to receive (raw) packets as they are received by the device drivers in the kernel.
- User space tools like `iftop`, `ntop`, `yaf`, `softflowd`, `vnstat` want to receive (raw) packets in order to produce statistics.
- 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

$\Rightarrow$ 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 are not 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   #0xffffff00                ; 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
```

`libpcap`

- 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.