ELEC50004: Software Systems (Baig)

Clemen Kok — Imperial College London

March 29, 2023

Contents

1 Computer Networks

1.1 Introduction

Circuit Switching. A fixed path is maintained for the duration of a call. Three phases: Circuit establishment (route established by configuring switches), data transfer and circuit disconnection. Overhead in call set-up. Once circuit set-up no extra overhead. An established circuit has a constant cost per unit time. It doesn't matter if data is being transferred or not - the circuit still uses resources. The problem is that if any link or switch on the circuit fails, the entire circuit fails.

Packet Switching. Data is sent as small packets of bits. Each packet is routed independently. Sender injects packets into network and intermediate nodes forward packets to other nodes. Each packet must contain routing information (Source and Destination) but packets may follow different rotues from source to destination. It is possible that packets may arrive out of order or lost. No fixed allocation of resources; users are charged for amount of data sent through and not by amount of time needed by communcation.

Multiplexing. Circuit switching may be implemented by reserving an entire physical channel between sender and receiver. However, a channel could also be a fixed frequency band or fixed time slot - such that multiple sender receiver pairs may share the same channel but use different frequency bands / time slots on it. Frequency Division Multiplexing and Time Division Multiplexing.

Connection-oriented and Connection-less Communication. Connection-oriented: Maintains connection between sender and receiver pair across multiple transmissions. Connection-less: Each transmission is considered independent of the previous, between a potentially different pair of sender and receiver.

Switching Mechanism	Connection-oriented	Connection-less
Circuit	Y	N
Packet	Y	Y

Explanation. Connections in packet switched networks may be maintained in software at the sender and receiver. They are end-to-end connections. Some protocols are connection-oriented (SMTP, FTP, TCP etc.), whilst some are connectionless (HTTP, UDP, IP, DHCP etc.) - depending on whether they maintain end-to-end connections or not. Furthermore, transmissions throughout the duration of connection may take different paths between sender and receiver - unlike circuit switching, where the paths remain fixed.

OSI Layers. Each layer attaches its own header to the outgoing packet. A header may be thought of as a 'bag' around the outgong packet. Bags must be removed in the opposite order to which they were put around the packet. Each layer's header contains information useful to the protocol of the layer. It's not computers that communicate, but rather the application processes (e.g. tabs in a browser running multiple application process).

IPV4 Addresses. A 32-bit IP address identifies each node on the internet (actually each network interface) - 2^{32} unique addresses. Divided into 4 groups of 8 bits. Some IP Addresses are special:

1. 255.255.255.255 [Broadcast Address].
2. 127.0.0.1 [Loopback/Local Host Address].
3. 192.168.0.0 - 192.168.255.255 [Local IP Addresses].

Port Number. An application process is identified by a 16-bit port number: 0-65536. An application process may 'bind' to a port number to reserve it for itself and identify itself for communication over the network. The OS ensures that port numbers are unique; no two processes can bind to the same port number. Numbers 0-1023 are reserved for well known applications and privileged services: FTP (20), SSH (22), DNS (53), Web Server (80), Telnet (23), SMTP (25).

L5 (Application)/L4 (Transport) Communication. The application and transport layer is only implemented on end systems, and not on intermediate nodes like routers and switches. These layers do all the bookkeeping at end points and establish a logical E2E communication to which the network and layers below is transparent. i.e. - as if communication is direct between the two host's L5/L4.

Socket. A socket is a software interface between an application process on L5 and the protocol on L4. Each application process on L5 has to open its own socket into L4. Thus, each socket is identified by two things: (1) Source IP and (2) Source application process' port number.

L4 (Transport)/L3 (Network) Communication. The transport layer uses port numbers to ensure that correct packets go to the correct application processes through their respective sockets. The network layer uses IP addresses to communicate between nodes.

1.2 Application Layer Protocols

1.2.1 Hypertext Transfer Protocol

Socket Life Cycle. (1) Create: OS allocates socket resources and returns a handle. (2) Bind: Socket connected to specific network address and port. (3) Connection: Establish connection to remote end-point. (4) Transfer: Data sent

TCP	UDP
Reliable transport	Unreliable
Flow Control (sender won't overwhelm receiver)	No Flow Control
Congestion Control (throttle sender when network overloaded)	No Congestion Control
Connection-Oriented (Setup required between client and server processes)	Connection-less

and received using socket. (5) Close: Socket closed, and any connections torn down and released. Not all sockets pass through all stages, and not all sockets establish a connection (only TCP).

Application Layer Protocols - HTTP. HTTP is an application message format (for request and response messages). When used by a web appliaction, HTTP packets are sent through TCP socketsm because TCP is reliable and web pages need to be fetched reliably, without data loss. It is meant for the client-server application architecture; HTTP request messages are sent from client to server. Response messages are received by client from server. Client/server model = client can request content from a powerful server.

HTTP Request Methods:

1. GET: Retrieve Information.
2. POST: Submit data to server.
3. PUT: Save object at location.
4. DELETE: Delete object at location.
5. HEAD: Retrieve resource information.

HTTP Response Codes:

1. 200 OK: Request succeeded, requested object later in this message.
2. 301 MOVED PERMANENTLY. Requested object moved, new location specified later in the message.
3. 400 BAD REQUEST. Requested message not found by server.
4. 505 HTTP VERSION NOT SUPPORTED.

What happens when you type a URL in the address bar? We type http://www.eie2.com/softwaresys.html and hit enter. A HTTP Request Message of the following form is generated: GET/softwaresyst.html HTTP/1.1

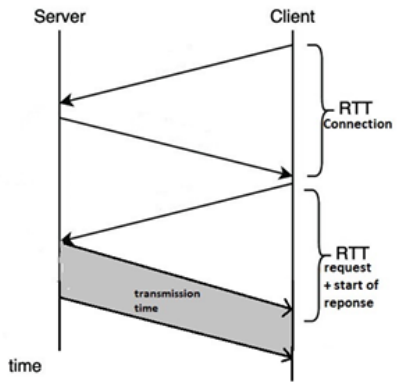
Host: www.eie2.com. The browser opens a TCP socket to connect it to the URL, but it cannot do that without knowing the IP Address corresponding to the URL (Port Number 80). The browser uses DNS (application layer protocol) to get the IP corresponding to www.eie2.com. In this process, DNS itself uses a UDP socket to connect to DNS servers. With the TCP socket opened, the HTTP GET Message is passed through the socket to the server, and a HTTP response is received from the server through the same socket.

Persistent and Non-persistent HTTP. HTTP/1.0 - Non-persistent HTTP. One object fetched per connection/socket. HTTP/1.1 and above - Persistent HTTP. Multiple objects sent over a single persistent HTTP connection. Only one handshake needed, but more burden on the server when averaged across all clients.

HTTP Connection and Request-Response. Round Trip Time (RTT) is a measure of the latency of the network. It is the time between initiating a request and receiving a response.

HTTP Connection Overheads. 1 RTT is used to initiate a TCP connection. It is also used for an HTTP request and for the first few bytes to HTTP response to return. Total time to serve and fetch one object is 2RTT + Transmission Time.

Figure 1: HTTP/1.0

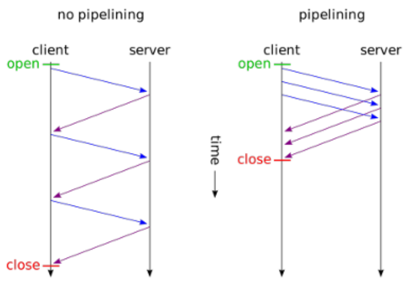


Retrieving 10 JPEG images - *Non-Persistent Case.* Base file + 10 JPEG images requires 11 * 2 RTT + transmission time = 22 RTT + transmission time.

Retrieving 10 JPEG images - *Persistent Case.* Base file + 10 JPEG images requires 1 RTT for connection + 11*(1 RTT (Req-Res)) + Transmission Time = 12 RTT + Transmission Time. Difference can be attributed to not having to re-establish TCP connection every time an image is fetched.

HTTP Optimisations - Pipelining. Pipelining allows multiple HTTP requests to be sent over a single TCP connection without waiting for the corresponding TCP responses. Requires requests to be queued on server side. Effect of Pipelining in previous example: 1 RTT for connection + δ * 1 RTT for all requests and first response + transmission time = 2 RTT + transmission time. A strict request-response order is not enforced. Requests are queued up on the server side and served in FIFO order.

Figure 2: HTTP Pipeline



Head of Line (HoL) Blocking. When the request at the head of the queue is heavy and requires disproportionately more time than the other requests, which need to wait. HTTP/2.0 solves this by *interleaving* request and response messages on the same TCP connection through breaking messages into small frames of fixed length. This allows the transmission of smaller objects to be completed faster and significantly reduces user-perceived delay compared to not using interleaving.

Interleaving Example. To understand this, consider the example of a Web page consisting of one large video clip and, say, 8 smaller objects. Thus, with pipelining, the server will receive 9 concurrent requests from any browser wanting to see this Web page. For each of these requests, the server needs to send 9 competing HTTP response messages to the browser. Suppose all frames are of fixed length, the video clip consists of 1000 frames, and each of the smaller objects consists of two frames. With frame interleaving, after sending one frame from the video clip, the first frame of each of the smaller objects is sent. Then after sending the second frame of the video clip, the last frame of each of the small objects is sent. Thus, the transmission of the smaller objects is complete by the end of the 18th frame [a large part of the page is rendered for the user very quickly]. If interleaving were not used, the transmission of the smaller objects would be completed only after 1016 frames.

Parallel Connections. Parallel TCP connections can also be opened with the server for block downloads. The source can start multiple threads, each with a different port number, and each of which can open its own TCP connection. They naturally solve the HoL Blocking problem but consume a lot of time and space for maintenance.

Optimising the Web: Web Caching. Origin servers may be located at a great distance from the end user. We can cache information inside the local network or a nearby ISP. This reduces traffic on an institutions access link. Caching refers to storing data for reuse. We cache when the user visits a page for the first time. When the user requests the same page, the cache will serve the copy, keeping the origin server from getting overloaded.

Conditional GET. The web cache tests if its copy of an object is updated. The Conditional GET only gets an object if it was modified after a certain date.

Cookies. HTTP is stateless, but some applications require statefulness. HTTP does not maintain client information on the server side. Cookies are used to maintain the state between transactions. They are stored on the client side and hvae IDs. The information is also stored on the server side in a backend database.

Network Protocol. A network protocol specifies a format for its Protocol Data Unit (PDU) - request and response messages in HTTP. Protocols use a specific model for implementation (TCP/IP model for the Internet). Protocols can be classified according to different networking characteristics - HTTP for example is a connectionless, stateless, pull-based, in-band protocol.

1.2.2 Domain Name Service

DNS. DNS is a distributed database with a client-server architecture. It is implemented in a hierarchy of many name servers. The DNS protocol specifies how client applications communicate with name servers. It specifies how name servers communicate with each other.

DNS as a Distributed, Hierarchical Database. If a client wants the IP Address of amazon.com, the client queries the root server to find the .com DNS server. The client when queries the .com Top Level Domain server to get the authoritative amazon server. The client then queries the authoritative amazon server to get the IP address for amazon.com. Authoritative DNS servers are an organisation's own DNS server(s), providing the authoritative hostname to IP mappings for the organisation's named hosts. A local DNS finds the root server address by consulting the root hints file in the server.

Optimisation: Local DNS Server. A local server at one's ISP or organisation network caches name-IP mappings. The local DNS server either returns a reply from its local cache of recent name-to-address translation pairs or forwards the request into the DNS hierarchy for resolution.

DNS Database Resource Records (RR). RR is a 4-tuple: (name, value, type, time-to-live (ttl)). TTL indicates the time after which the record will be removed from the database. RR Types:

1. If the type is A, the name is the hostname and the value is the IP address.

It is used to map a host name to an IP address. The final mapping of a host name to IP address is through a TYPE A RR being returned in a DNS response.

2. If the type is NS, the name contains the domain (foo.com) and the value is the hostname of the authoritative Name Server for the domain. **It is used to map a host name to another host name and to relay DNS queries from one host to another host.**

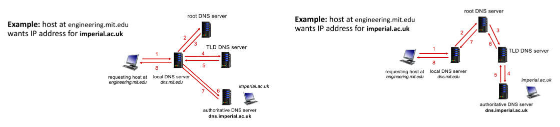
3. If the type is CNAME, the name contains the alias name for some "canonical" (actual) name. For example, www.ibm.com is actually servereast.backup2.ibm.com. The value is the canonical name. **Used for alias to canonical name mapping.**

4. If the type is MX, the data is like CNAME records but this is meant for mail servers.**Used for host name to corresponding mail server mapping.**

DNS Protocol Messages. DNS query and reply messages both have the same format. The first 12 bytes is the header section, which has a number of fields. The first field is a 16-bit number that identifies the query. This identifier is copied into the reply message to a query, allowing the client to match received replies with sent queries. Flags: Query/Reply - indicates whether a message is query (0) or reply (1). Authoritative - set in a reply message when a DNS server is an authoritative server for a queried name. Recursion - set when a client desires that the DNS server should perform recursion when it doesn't have the record. Recrsion-Available: set in a reply if the DNS server supports recursion. The question section contains information about the query being made. This includes a name field that contains the name being queried and a type field indicating the type of question being asked about the name. In a reply from the DNS server, the answer section contains the RRs for the name originally queried. A reply can return multiple RRs in the answer, since a hostname can have multiple IP addresses (e.g. in replicated web servers). The authority section contains records of other authoritative servers. The additional section contains other helpful records. For example, the answer field in a reply to an MX query contains the RR providing the canonical hostname of a mail server.

DNS Name Resolution. Recursive queries put the burden of name resolution on the contracted name server, and cause heavy loads at uppre levels of the hierarchy. Usually most DNS servers will not enable recursive querying. Recursive queries are used instead within the network of primary and secondary DNS servers.

Figure 3: DNS Name Resolution (Iterative vs Recursive)


Example: host at engineering.mit.edu wants IP address for imperial.ac.uk
root DNS server
TLD DNS server
requesting host at engineering.mit.edu
local DNS server dns.mit.edu
imperial.ac.uk
authoritative DNS server dns.imperial.ac.uk
1
2
3
4
5
6
7
8
Example: host at engineering.mit.edu wants IP address for imperial.ac.uk
root DNS server
TLD DNS server
requesting host at engineering.mit.edu
local DNS server dns.mit.edu
imperial.ac.uk
authoritative DNS server dns.imperial.ac.uk
1
2
3
4
5
6
7
8

Other DNS Services. Host Aliasing: Usually, a host has one canonical name and more than one aliases. Load Distribution: A website might use several replicated web servers. In this case, a set of IP addresses corresponds to the same host alias name. When clients make a DNS query for an alias name mapped to a set of addresses, the server responds with the entire set of IP addresses but rotates the ordering of the addresses within each reply.

1.3 Peer-to-Peer Applications and Protocols

1.3.1 BitTorrent

P2P Architectures. In a P2P Environment, there is no always-on server. End systems directly communicate. Peers are intermittently connected and change IP Addresses.

File Distribution. In the P2P Architecture, each peer can assist the server in distributing the file. In particular, when a peer receives some file data, it can use its own upload capacity to redistribute data to other peers.

File Distribution - *Client-Server Case.* Server transmission: The server must sequentially send/upload N file copies. Client download: Each client must download the file copy. Thus, the time to distribute F to N Clients using the Client-Server approach is $D_{c-s} \geq max(N * F/u_s, F/d_{min})$, which increases linearly in N.

.

File Distribution - *P2P Case.* Server transmission: Must upload one copy. Client download: Each client must download a file copy. Client uploads: All clients together provide an aggregate upload rate of Σu_i. Thus the tme to distribute F to N clients using P2P is $D_{P2P} \geq max(N * F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)$. This increases linearly in N but so does Σu_i as each peer brings service capacity.

BitTorrent. In BT, a large file is divided into 256kb chunks. Peers in the torrent can send and receive file chunks among each other. Torrents refer to a group of peers exchanging chunks of a file, while the Tracker tracks peers participating in the torrent.

Key Steps:

1. When a peer P joins the torrent network N, it registers itself with the tracker T, and periodically informs T that it is still in the torrent. Initially, T sends a list of IPs of S $\subseteq$ N peers.

2. P tries to establish concurrent TCP connections with each node in S. P succeeds in establishing connections with K $\subseteq$ S peers. The peers in K are P's 'neighbouring peers'. K is a dynamic set; neighbours come and go.

3. P periodically requests neighbours to send 'chunk lists', and from the options available, P requests for chunks it does not currently have.

4. P now has to decide which chunks to request first and which neighbours to send chunks to first. P requests on the basis of 'rarest chunk first'; the chunk with the fewest copies among her neighbours.

5. P has to continually keep track of the neighbours that are transmitting to it at the fastest rate - these are called unchoked neighbours - and sends chunks to them first. This is called the *tit-for-tat* approach. It enables P to find better trading partners and get the file faster, leading to a higher upload rate.

6. *Element of Randomization.* Every 30 seconds, P optimistically unchokes a new random partner, a 'probing peer'. If two peers are satisfied with trading, they put each other in their top peers list. Hence, peers capable of uploading at compatible rates find each other. Randomization allows new entrants to find providers.

7. Neighbours other than the top peers and the probing peer are choked. They do not receive chunks from P.

8. Once the peer has an entire file, they may selfishly leave or altruistically remain in the torrent.

1.3.2 Distributed Hash Tables

Introduction. The goal of pure P2P is to get rid of any cental server in the system. This goal is not entirely achievable, but the role of the server can be minimised to the extent that any 'peer-like' machine can act as the server. The first steps towards this goal would be to create a distributed database for the data that was earlier stored on a centralized server, thereby reducing the weight of the server.

.

DHT in P2P Setting. A wants to download a copy of the latest "Linux" distribution. This copy is available with B and C. The DHT therefore has entries "Linux", IP_B and "Linux", IP_C. These entries are available with D in the DHT. Thus, the part of the table with the IPs for "Linux" - specifically h("Linux"), where h is a hash function, is with D.

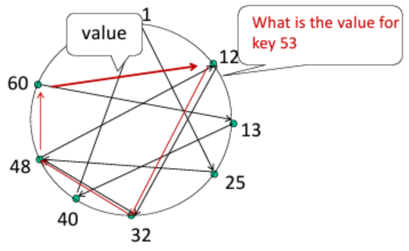
Naive Solution. Let every peer in the network store a list of IPs of all other peers. A also has this list. So, she queries all N peers for "Linux", and finds that D has the DHT entry for "Linux". She queries the DHT at D, and accesses the IPs of B and C and asks them to transmit a copy.

DHT Insertion Algorithm. Assign every peer an identifier x $\in$ 1...2^n. Any number in this range can be stored in n bits. We define a hash function h, where the key is 1...2^n, and make h available to every peer. Consider 8 peers - 1,12,13,25,32,40,48,60, where we let n = 6. A wishes to insert the entry ("Network Resources", 132.22.254.1) into the DHT. She would hash the key using h; assume that h("Network Resources") = 53. If peer 53 exists, the entry is inserted there. Else, the entry is inserted into the successor of 53. A similar algorithm is used for lookup. To find peer 53, we use a recursive lookup - explained more below.

DHT - Explained simply. A Distributed Hash Table (DHT) is like a big library where each book is stored in a different room. Each room has a unique number and each book has a number too. When you need a book, you ask the librarian for the room number and go directly to that room to get the book you need. In a DHT, each computer stores a portion of the data and has a unique identifier, and when you need data, you can quickly locate it by querying the computer responsible for storing it. This allows for efficient storage and retrieval of data across a network of computers without the need for a centralized server.

Circular DHT. In a circular DHT, each peer is only aware of its immediate successor and predecessor. To resolve a query, we query each immediate successor until we get the peer; this takes O(N) messages to resolve. Thus we use *shortcuts*. Each peer keeps track of the IP addresses of its predecesssor, successor and shortcuts. This reduces the number of messages and gives O(logN).

Figure 4: Circular DHT with Shortcuts



Peer Churn. Each peer periodically checks its two successors to check aliveness. If the immediate successor leaves, we choose the next successor as the new immediate successor.

BitTorrent Tracker is a DHT. BitTorrent uses the Kademlia DHT to create a distributed tracker. A newly arriving peer queries the DHT with a "torrent identifier", and can determine the "tracking peer" - the peer that has the list of peers in that torrent. The arriving peer can now query this tracking peer for the list of other peers in the torrent.

1.4 Transport Layer Protocols

1.4.1 Introduction

Functionality: Sender Side. The sender receives an application-layer message and breaks it up into segments. It determines the segments' header fields values, attaches headers to segments and passes these segments to the network layer (IP). A segment is an application layer message (whole/fragment with identifier) and transport layer header. The network layer protocols exchange datagrams (segment + network layer header with IP addresses). The link layer protocols exchange frames (network layer datagram + link layer header). The physical layer protocols exchange raw bits over the channel.

Functionality: Receiver Side. The receiver receives segments from the network layer. It checks header values (including port number), and extracts the application layer message. It the demultiplexes the message up to the application via a socket.

UDP and TCP. UDP and TCP are the two major transfer layer protocols. Both provide the minimal multiplexing and demultiplexing services, as well as integrity checking with error detection fields in their segments' headers. TCP also includes additional services such as reliable data transfer, flow control, sequence numbers, acknowledgements and timers. It also includes congestion control, preveting swamping.

1.4.2 Multiplexing

Multiplexing. Multiplexing is the process of gathering data from the application layer through multiple sockets and passing it as a stream of segments to the network layer. It is a many-to-one operation and happens at the sender.

Demultiplexing. Demultiplexing is the process of delivering data, arriving at the transport-layer from th enetwork layer to multiple sockets on the application layer. It is a many-to-one operation. Demultiplexing receives IP datagrams on the network layer. How it works: The destination receives IP datagrams. Each datagram has a source IP address and a destination IP address. Each datagram alos carries one transport-layer segment, and has a source and a destination port number. The transport layer uses these port numbers and possibly IP addresses to direct segment to the appropriate sockets.

Why Port Numbers are not sufficient for connection-oriented demultiplexing. We also need the source IP address. When segments arrive at a web server, they have the same destination port number and may also have the same source port number. Thus, to demultiplex properly, we will also need the source IP address. TCP uses the 4-tuple (IP Destination, Port Destination, IP Source and Port Source) to identify the exact connection socket to forward a packet to.

1.4.3 Integrity Checking

Checksum. The checksum is a compressed summary of the state of the segment before it was transmitted. If the bits are flipped, the data integrity is compromised. The checksum will change to reflect that.

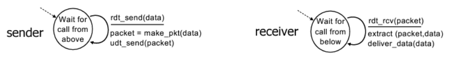
UDP Checksum. The sender treats the contents of a segment (including the header) as a sequence of 16-bit words. The checksum is computed by adding all the 16-bit words of the segment and computing the 1's complement of the sum. We then put the checksum value in the UDP checksum field. The receiver then adds all 16-bit words of the segment, including the checksum. In UDP, if the checksum is invalid, the packet is dropped. In TCP, retransmission occurs. The checksum is not the most reliable in terms of checking data integrity; bit flips may occur in such a way that the checksum is not changed (e.g. last tw obits of the two numbers being added). However, stronger checks of integrity occur on the link layer using hash functions. Note: When adding numbers, a carryout from the MSB needs to be added to the result.

1.4.4 Reliable Data Transfer

Why does data get lost / arrive out of order? IP is only a 'best effort' service. Packets may be lost due to network congestion, such as when routers' queues overflow. Packets may get corrupted and dropped at the receiver or at an intermediate router. Packets may be undeliverable due to network hardware issues. Packets may be dropped or corrputed due to software bugs. Due to variable delays on different channels, packets may also arrive out of order.

Reliable Data Transfer (RDT) Protocol 1.0. Reliable Transfer over a reliable channel - the underlying channel in this situation is perfectly reliable: No bit errors and no loss of packets. The FSMs for sender and receiver are trivial in this case; the sender sends data into the underlying channel and the receiver reads data from the underlying channel.

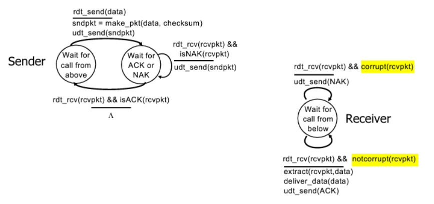
Figure 5: RDT 1.0 - Perfect Channel



RDT 2.0 - Channel with Bit Errors. ACKs - the receiver explicitly tells the sender that the packet has been received without error. NAKs - the receiver explicitly tells the sender that the received packet had errors. The sender retransmits packets on the receipt of a NAK. The simplifying assumption of the protocols is Stop and Wait - the sender sends a packet and waits for a receiver's response to it before sending the next packet. But what happens if the ACK/NAK is corrupted? The sender would not know what happened at the receiver. If the sender retransmits the packet, there could be a possible duplicate reception at the receiver. Duplicates: The sender retransmits the current packet if the ACK/NAK is corrupted. The sender adds a sequence number to each packet. The receiver discards a duplicated packet. If the sender receives a corrupted ACK or NAK, they cannot be sure if the packet reached the receiver or not.

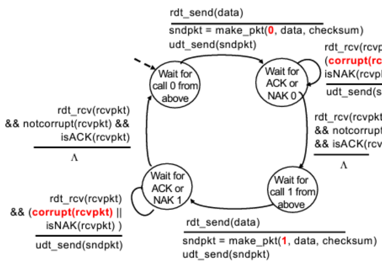
RDT 2.1 - Sender handling garbled ACK/NAKs. We add a sequence number to the packet. Two numbers (0,1) will suffice. The four states instead

Figure 6: RDT 2.0 - ACKs and NAKs Introduced



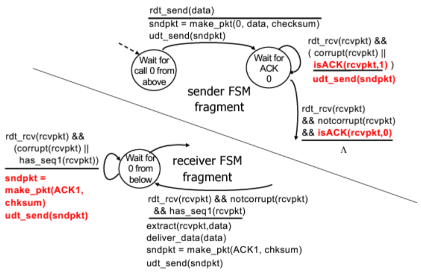
of two distinguishes alternating packets with sequence numbers 0 and 1. In a stop and wait protocol, that is sufficient to unambiguously discard a duplicate packet - one having the same sequence number as the previous packet - on the receiver's side if it was successfully accepted previously.

Figure 7: RDT 2.1 - Sequence Numbers to overcome duplication


rdt_send(data)
sndpkt = make_pkt(0, data, checksum)
udt_send(sndpkt)
Wait for call 0 from above
Wait for ACK or NAK 0
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isNAK(rcvpkt))
udt_send(sndpkt)
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt)
Λ
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt)
Λ
Wait for ACK or NAK 1
Wait for call 1 from above
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isNAK(rcvpkt))
udt_send(sndpkt)
rdt_send(data)
sndpkt = make_pkt(1, data, checksum)
udt_send(sndpkt)

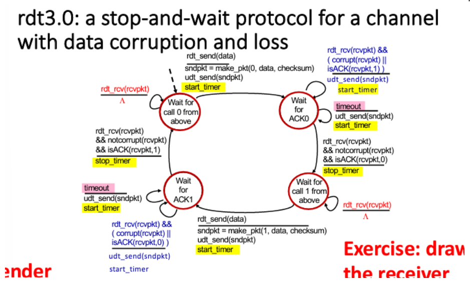
RDT 2.2 - NAK-free Protocol. The same functionality as RDT 2.1 can be implemented using ACKs only. Instead of a NAK, the receiver re-sends an ACK for the last packet received successfully using the previous sequence number. A duplicate ACK at the sender results in the same action as a NAK - retransmit current packet. TCP uses this approach to be NAK-free.

Figure 8: RDT 2.2 - NAK-free RDT


rdt_send(data)
sndpkt = make_pkt(0, data, checksum)
udt_send(sndpkt)
Wait for call 0 from above
Wait for ACK 0
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isACK(rcvpkt,1))
udt_send(sndpkt)
sender FSM fragment
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,0)
Λ
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || has_seq1(rcvpkt))
sndpkt = make_pkt(ACK1, chksum)
udt_send(sndpkt)
Wait for 0 from below
receiver FSM fragment
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && has_seq1(rcvpkt)
extract(rcvpkt,data)
deliver_data(data)
sndpkt = make_pkt(ACK1, chksum)
udt_send(sndpkt)

RDT 3.0 - Channels with Errors and Loss. We now assume that the underlying channel can also lose packets (both data and ACKs). We still continue with Stop-and-Wait. In this approach, the sender waits for a *reasonable* amount of time for an ACK. If no ACK is received in this time, the packet is retransmitted. If a packet or ACK is delayed beyond timeout but not lost, retransmission will cause duplication (but sequence numbers already handle this).

Figure 9: RDT 3.0 - Reasonable wait to deal with packet loss


rdt3.0: a stop-and-wait protocol for a channel with data corruption and loss
rdt_send(data)
sndpkt = make_pkt(0, data, checksum)
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isACK(rcvpkt,1))
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt)
Λ
Wait for call 0 from above
Wait for ACK0
timeout
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,1)
stop_timer
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,0)
stop_timer
Wait for ACK1
Wait for call 1 from above
timeout
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt)
Λ
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isACK(rcvpkt,0))
udt_send(sndpkt)
start_timer
rdt_send(data)
sndpkt = make_pkt(1, data, checksum)
udt_send(sndpkt)
start_timer
Exercise: draw the receiver
ender

Stop-and-wait is not realistic. Pipelining is necessary for a real transport layer protocol. It significantly improves channel utility.

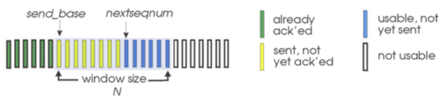
1.4.5 Go-Back-N and Selective Repeat

Introduction. Other than ACKs and Timers, these techniques require longer ranges of sequence numbers, and buffers on sender and receiver sides. The buffering mechanism also helps solve the problem of out of order delivery. Out of order delivery is impossible to solve with stop-and-wait. TCP uses aspects of both of these approaches.

Go-Back-N Sender. In GBN, the sender is allowed to transmit multiple packets (when available) without waiting for an acknowledgement, but is constrained to have no more than some maximum allowable number N of unacknowledged packets in the pipeline (thus a ”window” of up to N consecutive transmitted but unACKed packets). The base is the sequence number of the oldest unacknowledged packet. The `nextseqnum` is the smallest unused sequence number. Sequence numbers in the interval [0, base-1] correspond to packets that have already been transmitted and acknowledged. The interval [base, `nextseqnum`-1] corresponds to packets that have been sent but not yet acknowledged. Sequence numbers in the interval [`nextseqnum`, base+N-1] can be used for packets that can be sent immediately, should data arrive from the upper layer. Sequence numbers greater than or equal to base+N cannot be used until an unacknowledged packet currently in the pipeline (the packet with the sequence number base) has been acknowledged. *Imposing a limit N on the window size allows TCP to implement flow control and congestion control.*

Go-Back-N Receiver. The receiver maintains a rcv_base pointing to the

Figure 10: GBN Sender



next expected sequence number. On receipt of the next expected sequence number, we move rcv_base to the next expected sequence number, and send ACK to rcv_base-1. On receipt of an out-of-order packet, we don't move rcv_base, put the packet in the buffer and re-ACK rcv_base-1. The packet with the highest in-order sequence number is the one that has the highest sequence number among all packets received so far with no gaps in between. If out-of-order packets are not buffered, a delayed earlier packet would require retransmissions of all out-of-order packets which may be prevented by buffering out-of-order packets so that if the earlier packet is only slightly delayed and an ACK can be sent for the highest in order packet in that case, the sender does not have to retransmit. On the other hand, if the earlier packet is more than slightly delayed, the sender is going to get timed out and resend all the later packets anyway.

Figure 11: GBN Receiver

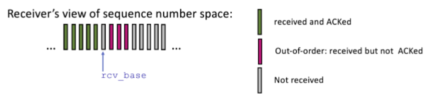
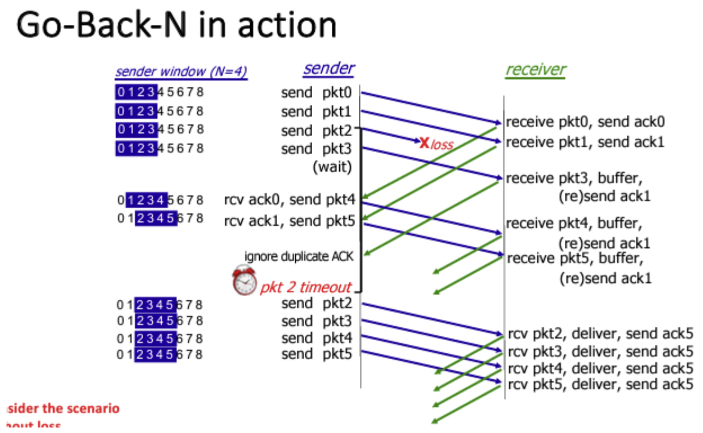
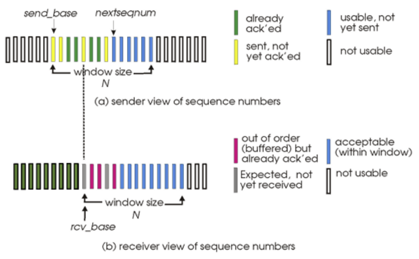

Receiver's view of sequence number space:
rcv_base
received and ACKed
Out-of-order: received but not ACKed
Not received

Figure 12: GBN In Action



Selective Repeat. An important mechanism of GBN is cumulative acknowledgement. An alternative mechanism is keeping track of each packet individually. The receiver individually acknowledges all correctly received packets. It buffers packets as needed for eventual in-order delivery to the upper layer. The sender times-out/retransmits individually for unACKed packets; the sender also maintains a timer for each unACKed packet.

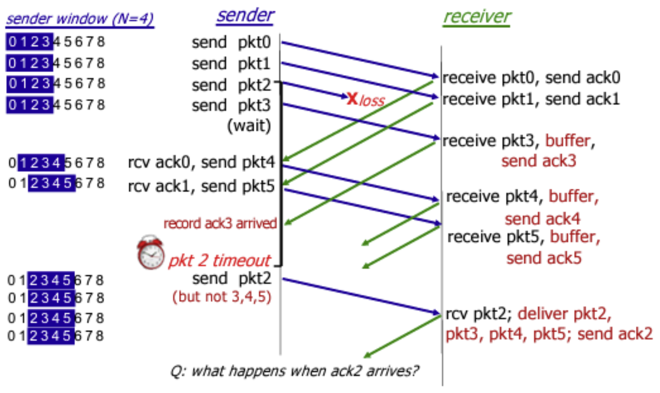
Figure 13: Selective Repeat: Sender and Receiver Side Windows


send_base
nextseqnum
already ack'ed
sent, not yet ack'ed
usable, not yet sent
not usable
window size N
(a) sender view of sequence numbers
out of order (buffered) but already ack'ed
Expected, not yet received
acceptable (within window)
not usable
window size N
rcv_base
(b) receiver view of sequence numbers

Selective Repeat Sender. When data is received from above - if the next available sequence number is within window size, send the packet. Timeout: Resend packet n only, and restart its timer. ACK(n) in [send_base, send_base+N]: Mark packet n as received. If n is the smallest unACKed packet, advance the window base to the next unACKed sequence number.

Selective Repeat Receiver. Packet n in [rcvbase, rcvbase+N-1] is received: Send ACK(n). If n is out-of-order, buffer it. If it is in-order, deliver the packet and advance window to next not-yet-received packet. If packet n is in [rcvbase-N, rcvbase-1], ACK(n). Else, ignore.

Figure 14: SR In Action


sender window (N=4)
sender
receiver
send pkt0
send pkt1
send pkt2
send pkt3
(wait)
X loss
receive pkt0, send ack0
receive pkt1, send ack1
receive pkt3, buffer, send ack3
rcv ack0, send pkt4
rcv ack1, send pkt5
receive pkt4, buffer, send ack4
receive pkt5, buffer, send ack5
record ack3 arrived
pkt 2 timeout
send pkt2 (but not 3,4,5)
rcv pkt2; deliver pkt2, pkt3, pkt4, pkt5; send ack2
Q: what happens when ack2 arrives?

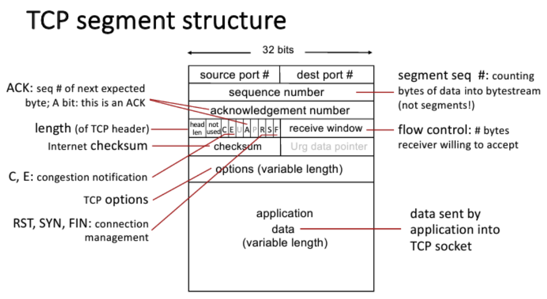
GBN vs SR. The main difference between GBN and SR is that GBN retransmits a group of packets when a packet is lost, while SR retransmits only the lost packet. GBN is more efficient when consecutive packets are lost, while SR is more efficient when only a few packets are lost.

1.4.6 Transmission Control Protocol

TCP. TCP is Point-to-Point. There is only one sender and one receiver. It is reliable and has an in-order byte stream. There are no message boundaries. It is full-duplex with bi-directional data flow in the same connection. Pipelining

- TCP congestion and flow control set the window size. TCP is connection-oriented with handshaking (exchange of control messages) initialising the sender and receiver state before data exchange. TCP is flow controlled and the sender will not overwhelm the receiver.

Figure 15: TCP Segment Structure


TCP segment structure
32 bits
source port #
dest port #
sequence number
acknowledgement number
head len
not used
C
E
U
A
P
R
S
F
receive window
checksum
Urg data pointer
options (variable length)
application data (variable length)
ACK: seq # of next expected byte; A bit: this is an ACK
length (of TCP header)
Internet checksum
C, E: congestion notification
TCP options
RST, SYN, FIN: connection management
segment seq #: counting bytes of data into bytestream (not segments!)
flow control: # bytes receiver willing to accept
data sent by application into TCP socket

TCP Sequence Numbers and ACKs. Sequence Number - byte stream "number" of first byte in segment's data. Acknowledgements - Sequence Number of the next byte expected from the other side. Cumulative ACK. It is up to the implementor on how the receiver handles out-of-order segments.

TCP Sender. When data is received from an application, we create a segment with a sequence number. The sequence number is the byte-stream number of the first data byte in the segment. We start the timer if it is not already running (base packet). The timer is for the oldest unACKed segment (like GBN). On timeout, we retransmit the oldest unACKed segment - so only one packet is retransmitted at a time. We then restart the timer. On ACK received, we acknowledge previously unACKed segments with lower sequence numbers in-order; we slide the window accordingly and start timer if there are still unACKed segments. TCP can be thought of as SR with accumulative ACK, or GBN with only single packet retransmission (the oldest unACKed packet). If the TCP receiver is buffering out-of-order packets, a cumulative ACK can cause the receiver to prevent several retransmissions from happening. Else, these retransmissions would happen one by one as packet timers expire. The TCP timer is always running for the oldest unACKed packet (like GBN).

TCP 3-way handshake. Say, both the client and server create a TCP socket. The client connects to the server sending a SYN message with a sequence number x (the SYN message is a TCP segment with the SYN bit set in the header). The server is waiting for a connection, and receives the SYN message and enters the SYN received state (not the established state) and sends a SYN ACK message back. Finally, the client sends an ACK message to the server, and when the server receives this it enters the ESTABlished state. This is when the application process would see the return from the wait on the socket accept() call.

Figure 16: TCP Retransmission

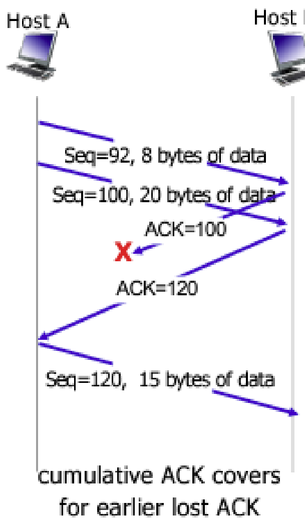
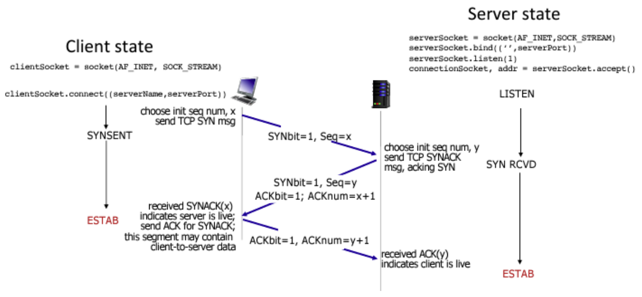


Figure 17: TCP 3-way handshake


```
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
```

```
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
connectionSocket, addr = serverSocket.accept()
```
Server state
Client state
LISTEN
choose init seq num, x send TCP SYN msg
SYNSENT
SYNbit=1, Seq=x
choose init seq num, y send TCP SYNACK msg, acking SYN
SYN RCVD
SYNbit=1, Seq=y ACKbit=1; ACKnum=x+1
received SYNACK(x) indicates server is live; send ACK for SYNACK; this segment may contain client-to-server data
ESTAB
ACKbit=1, ACKnum=y+1
received ACK(y) indicates client is live
ESTAB

Closing TCP connection. The client and server both close their side of the connection. The Initiator send a TCP segment with FIN bit = 1 and responds to the received FIN with ACK.

1.5 Network Layer Protocols

1.5.1 Introduction

Introduction. The network layer delivers datagrams from source to the destination host. Each datagram contains a transport layer segment. The network layer is not an end-to-end service like the transport layer. IP Datagrams are stored and forwarded by routers; datagrams are mapped from input to output ports using a forwarding table. This prcocess is called forwarding and constitutes the *data plane* of the network layer. Distributed pathfinding algorithms are implemented into routers, to do forwarding based on those algorithms. Routing algorithms fill up the forwarding tables. This operation is called routing and constitutes the *control plane* of the network layer.

Dynamic Host Configuration Protocol. Client-server based application layer protocol. The network's access point (border router) implements the DHCP server. Each computer connecting to the network is a DHCP client. Each connecting computer dynamically obtains an IP address from the DHCP server as it joins the network. Dynamic allocation allows the reuse of addresses (only hold addresses while connected/on). Typically, the DHCP server will be co-located in the router, serving all subnets to which that router is attached. DHCP is an application layer protocol. The network's ability to broadcast is crucial to its function. DHCP can return more than just the allocated IP address on the subnet. It can return the address of the first-hop router for the client, the name and IP address of the DNS server and the network mask (indicating network vs host portion of the address).

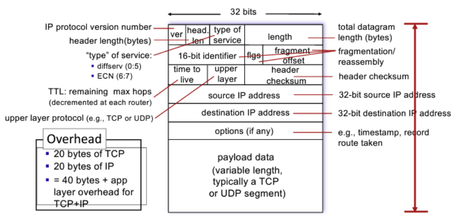
DHCP Overview:

1. Host broadcasts DHCP Discover message.
2. DHCP Server responds with DHCP Offer message. [The above two steps can be skipped if a client remembers and wishes to reuse a previously allocated network address]
3. Host requests an IP address with a DHCP request message.
4. DHCP Server sends back the address with a DHCP ACK message.

1.5.2 IP Addressing

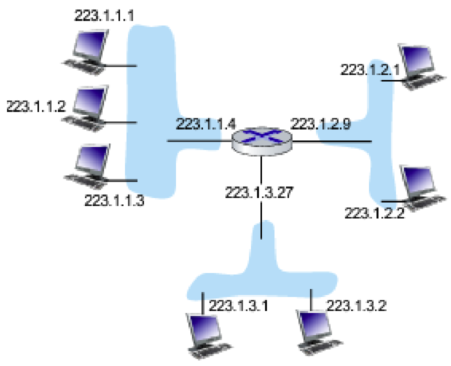
Interfaces. An IP Address is a 32-bit identifier associated with each host or router interface. An interface is the connection between a host/router and the physical link. Routers typically have multiple interfaces. Hosts typically have one or two interfaces (Wired Ethernet, Wireless 802.11).

Figure 18: IP Datagram



Subnets. A subnet is a device interface that can physically reach each other without passing through an intervening router. IP Addresses have a hierarchical structure; the *Network ID* identifies the subnet: all devices in the same subnet share these higher order bits. The *Host ID* identifies the host within the subnet: the remaining lower order bits after the Network ID.

Figure 19: 3 Subnet Network



Classful Routing (1981-1993). IP Addresses were divided into classes A, B and C with fixed lengths for the Network ID - A [1-126.x.y.z], B [128-191.x.y.z] and C [192-223.x.y.z]. 4 most significant bits were

Classless Interdomain Routing (CIDR). Introduced in 1993, CIDR allowed variable sized subnets. The Network ID can be of arbitrary length. The address in CIDR is written as a.b.c.d/x, where x is the number of bits in the subnet portion of the address. E.g. - 200.21.16.0/23 represents **11001000 00010111 0010000**0 00000000, where the first 23 zeroes represent the Network ID and the last 9 zeroes represent the Host ID.

Subnet Mask. The subnet mask is a 32-bit number which, when bitwise ANDed with an IP Address zeroes all but the Network ID bits. For an a.b.c.d/x address, the subnet is x 1's followed by 32-x 0's. For example, the subnet mask 11111111 11111111 11111110 00000000 is /23.

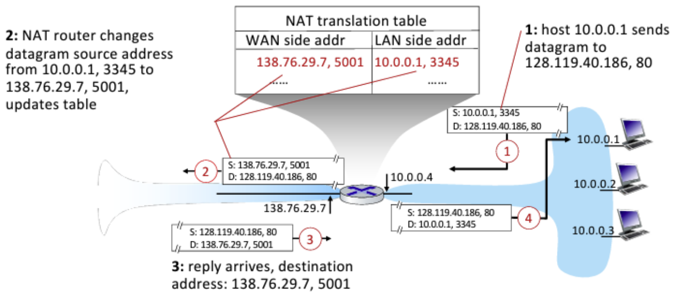
Subnetting Example 1. An ISP owns a block of addresses of the form 141.165.34.64/26. To create four subnets with each subnet receiving the same number of IP addresses, we can create the following subnets: 141.165.34.64/28, 141.165.34.80/28, 141.165.34.96/28, 141.165.34.112/28.

Subnetting Example 2. An ISP owns a block of addresses of the form 141.165.34.64/26. To create four subnets with each subnet with subnet 1, 2, 3 and 4 requiring 20, 12 ,8 and 4 addresses, we can create the following subnets: 141.165.34.64/27, 141.165.34.96/28, 141.165.34.112/29 and 141.165.34.120/29. *Thus, the more free trailing "zeroes" in the IP, the more addresses that subnet has.*

Network Address Translation (NAT). All devices in the local network have 32-bit addresses in a "private" IP address space (10/8, 172.16/12, 192.168/16 prefixes) that can only be used in the local network. Just one IP address is therefore needed from the provider ISP for all devices. The NAT Router does the following: (1) In outgoing datagrams, it replaces the Source IP Address and Port Number to NAT IP Address and a new Port Number. (2)

Remote clients/servers will respond using (NAT IP Address, New Port Number) as destination address. (3) In the NAT Translation Table, every (Source IP Addres, Port Number) to (NAT IP Address, New Port Number) mapping is maintained. (4) In incoming datagrams, replace (NAT IP Address, New Port Number) in destination fields of every incoming datagram with corresponding (Source IP Address, Port Number) stored in NAT Table.

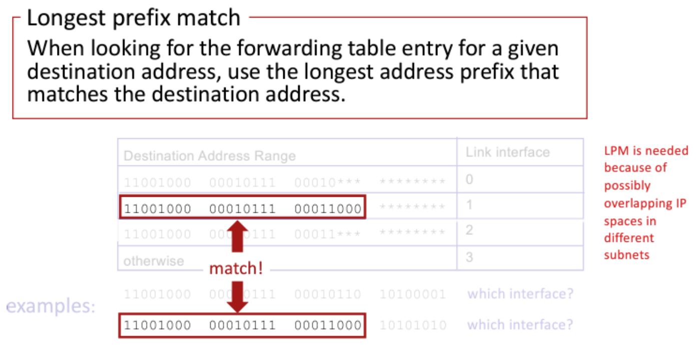
Figure 20: Network Address Translation



1.5.3 Forwarding (Data Plane)

Longest Prefix Matching. Longest Prefix Matching is used to ensure that the packet is forwarded to the correct next-hop router by matching the longest prefix in the destination IP address with an entry in the routing table. The routing table contains a list of prefixes, each associated with a specific next-hop router. The prefix with the longest match to the destination IP address is selected as the best match, and the packet is forwarded to the corresponding next-hop router. It is needed only because of possibly overlapping IP spaces in different subnets.

Figure 21: Longest Prefix Matching



Longest prefix match

When looking for the forwarding table entry for a given destination address, use the longest address prefix that matches the destination address.

Destination Address Range	Link interface
11001000 00010111 00010*** ********	0
11001000 00010111 00011000 ********	1
11001000 00010111 00011*** ********	2
otherwise	3

match!

examples:

11001000 00010111 00010110 10100001	which interface?
11001000 00010111 00011000 10101010	which interface?

LPM is needed because of possibly overlapping IP spaces in different subnets

1.5.4 Routing (Control Plane)

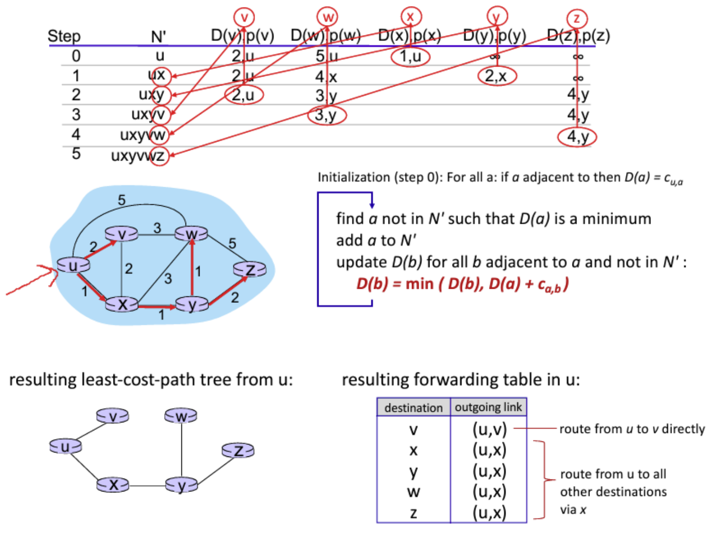
Forwarding vs Routing. *Forwarding* is the actual act of moving packets from one interface to another on a networking device, while *Routing* is the process of selecting the best path for those packets to take through the network.

Graph Abstraction and Link Costs. Costs are defined by a network operator and may reflect certain policies. Costs could always be 1; or inversely related to the bandwidth; or inversely related to congestion. Cost could be a function defined by the operator.

Link State Algorithms. Link State Algorithms are centralised greedy algorithms at each router. Each router maintains complete topology of the network and the link costs. The best paths are determined at each router by processing the full topology.

Djikstra's Link-State Routing Algorithm. Assume Centralised: The network topology and link costs are known to all nodes. This is accomplished via *Link State Broadcast.* All nodes have the same information. This algorithm computes the least cost paths from one node (source) to all other nodes (single source shortest path). It gives the forwarding table for that node. This algorithm is *iterative*, meaning that after k iterations, it learns the least cost path to k destinations. Notation: $c_{w,z}$ is the cost of a direct link connecting w and z; it is ∞ for non-direct neighbours. D(v) is the current estimate of cost of the least-cost-path from source to destination v. p(v) is the predecessor node along path from source to v. N' is the set of nodes whose least-cost-path is definitively known.

Figure 22: Djikstra Link State Algorithm


v
w
x
y
z
5
2
3
5
2
3
1
1
1
2
u
v
w
x
y
z
resulting least-cost-path tree from u:
route from u to v directly
route from u to all other destinations via x

Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					

Initialization (step 0): For all a: if *a* adjacent to then *D(a)* = $c_{u,a}$

```
find a not in N' such that D(a) is a minimum
add a to N'
update D(b) for all b adjacent to a and not in N' :
    D(b) = min ( D(b), D(a) + $c_{a,b}$ )
```

resulting forwarding table in u:

destination	outgoing link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
z	(u,x)

Algorithm 1 Djikstra Link State Routing Algorithm

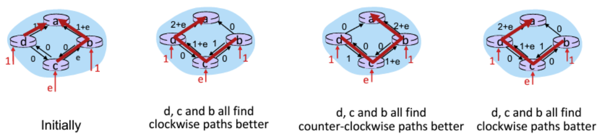
```
1: Initialization: $N' = \{u\}$ /* compute least cost path from $u$ to all other nodes */
2: for all nodes $v$ do
3:   if $v$ adjacent to $u$/* $u$ initially knows direct-path-cost only to direct neighbors */ then
4:     $D(v) = c_{u,v}$ /* but may not be minimum cost! */
5:   else
6:     $D(v) = \infty$
7:   end if
8: end for
9: repeat
10:   /* Find $w$ not in $N'$ such that $D(w)$ is a minimum, add $w$ to $N'$, and update $D(v)$ for all $v$ adjacent to $w$ and not in $N'$ */
11:   find $w$ not in $N'$ such that $D(w)$ is a minimum
12:   $N' = N' \cup \{w\}$
13:   for all nodes $v$ adjacent to $w$ and not in $N'$ do
14:     $D(v) = \min(D(v), D(w) + c_{w,v})$ /* new least-path-cost to $v$ is either old least-cost-path to $v$ or known least-cost-path to $w$ plus direct-cost from $w$ to $v$ */
15:   end for
16: until all nodes in $N'$
```

Algorithmic Complexity. For $|N|$ Nodes and $|E|$ Edges: $O((|E|+|N|)lg|N|)$ [Binary Heap, Worst Case], $O(|E|+|N|lg|N|)$ [Fibonacci Heap, Average Case].

Message Passing Complexity. Each router must broadcast its link state information to other n routers. Efficient broadcast algorithms can disseminate link state information of one router to all other routers in O($|N$) link-crossings. Each router's message crosses O($|N|$) links, so the overall message complexity is O($|N|^2$).

Route Oscillations. When link costs depend on dynamic factors such as traffic volume, frequent route oscillations are possible. In the example below, when routing to destination a, traffic enters at d, c and b with rates 1, e(<1) and 1. The link costs are directional and volume-dependent.

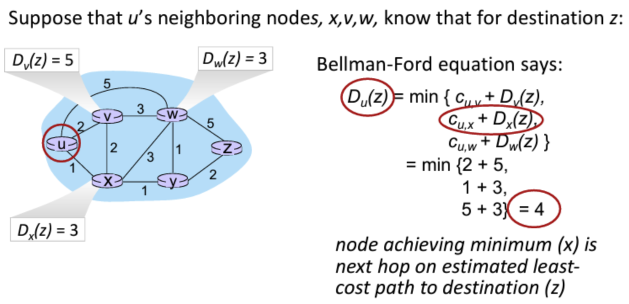
Figure 23: Route Oscillations


a
d
b
c
1
1+e
0
0
0
e
1
1
e
Initially
2+e
1+e
d, c and b all find clockwise paths better
2+e
d, c and b all find counter-clockwise paths better
d, c and b all find clockwise paths batter

Distance Vector Algorithms. Distance Vector Algorithms are distributed dynamic programming algorithms at each router. The routers do not maintain global topology, and instead they only exchange next-hop information with their neighbours. Each router iteratively computes the best neighbour to forward an incoming datagram to.

Bellman-Ford Equation (Dynamic Programming). Let D_x(y) be the cost of the least-cost path from x to y. Thus, D_x(y) $= min_v$ $\{c_{x,v} + D_v(y)\}$, where min_v is the minimum taken over all neighbours v of x.

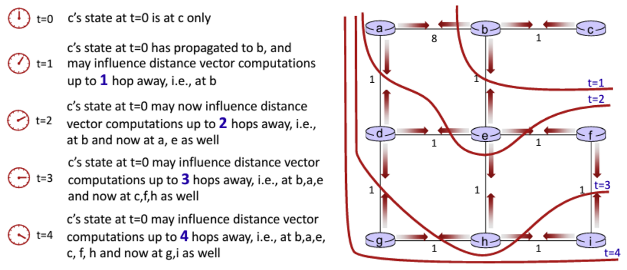
Figure 24: Distance Vector


Suppose that u's neighboring nodes, x,v,w, know that for destination z:
$D_v(z) = 5$
$D_w(z) = 3$
$D_x(z) = 3$
Bellman-Ford equation says:
$D_u(z) = \min \{ c_{u,v} + D_v(z), c_{u,x} + D_x(z), c_{u,w} + D_w(z) \}$
$= \min \{2 + 5, 1 + 3, 5 + 3\} = 4$
node achieving minimum (x) is next hop on estimated least-cost path to destination (z)

Algorithm Characteristics. Each node waits for a change in the local link cost, or a message from its neighbour. It then recomputes the dv estimates using the dv received from the neighbour. If the dv to any destination has changed, it notifies the neighbours. DV is iterative and asynchronous. It is also distributed and self-stopping; neighbours notify neighbours when necssary and when no notifications are received, no actions are taken. Each router maintains a table of distances (or metrics) to each destination network.

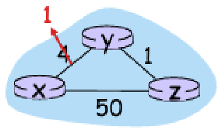
State Information Diffusion. Iterative Communication; computation steps diffuse information through the network.

Figure 25: State Information Diffusion


t=0 c's state at t=0 is at c only
t=1 c's state at t=0 has propagated to b, and may influence distance vector computations up to 1 hop away, i.e., at b
t=2 c's state at t=0 may now influence distance vector computations up to 2 hops away, i.e., at b and now at a, e as well
t=3 c's state at t=0 may influence distance vector computations up to 3 hops away, i.e., at b,a,e and now at c,f,h as well
t=4 c's state at t=0 may influence distance vector computations up to 4 hops away, i.e., at b,a,e, c, f, h and now at g,i as well

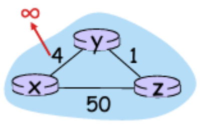
Good news travels slowly. Link cost changes: Node detects local link cost change. It then updates routing info and recalculates the local DV. If the DV changes, we notify neighbours. For example: at t_0, y detects a link cost change, updates its DV and informs neighbours. At t_1, z receives updates from y, updates its table, and computes the new least cost to x, and sends its neighbours its DV. At t_2, y receives z's update. y's least costs do not change, so y does not send a message to z.

Figure 26: Good news travels slowly



Bad news travels very slowly. In this example, y sees that the direct link to x has gone down, but the current z DV has a better (cost 5) path to x. Y computes the new cost 6 via z, then notifies z of the new cost of 6 to x. z learns that the path to x via y has the new cost 6, so z computes that the new cost to x will be 7 via y, and notifies y of the new cost of 7 to x. y learns that path to x via z has new cost 7, so y computes "my new cost to x will be 8 via y", notifies z of new cost of 8 to x. z learns that path to x via y has new cost 8, so z computes "my new cost to x will be 9 via y", notifies y of new cost of 9 to x... z learns that path to x via y has new cost 51, so z computes "my new cost to x will be 50, directly to x", notifies y of new cost of 50 to x. y learns that path to x via z still has cost 51. No change at y. (still goes through z, but path at z has changed to the correct one).

Figure 27: Bad news travels very slowly



Link State vs Distance Vector. Distance vector routing is a type of routing protocol in which routers communicate with *their neighboring routers* to share information about the best path to a destination network. Link state routing is a type of routing protocol in which routers communicate with *all other routers* in the network to share information about the state of their own links.

1.5.5 Internet Routing

Autonomous System (AS). An AS is a collection of one or more associated Internet Protocol (IP) prefixes (i.e. subnetworks) with a clearly defined routing policy that governs how the AS exchanges routing information with

Link State	Distance Vector
n routers, O(n^2) messages sent	Exchange between neighbours; convergence time varies
O(n^2) algorithm, may have oscillations	Convergence time varies, may have routing loops
Router can advertise incorrect link cost and each router computes only its own table	Each router's table used by others: error propagates through the network

other autonomous systems. Therefore, an AS will be usually owned by a single organization or ISP.

Intra-AS Routing. Routing within AS. All routers in an AS must run the same intra-domain protocol. Routers in different AS can run different intra-domain routing protocols. Gateway routers at the edges of individual AS link(s) to gateway routers of other AS.

Inter-AS Routing. Routing among different AS. Gateways perform inter-domain routing as well as intra-domain routing.

Interconnected AS. Forwarding table configured by intra and inter-AS routing algorithms. Intra-AS routing determines entries for destinations within AS. Inter-AS and Intra-AS determines entries for external destinations.

Intradomain Forwarding. Suppose router in AS1 receives a datagram destined outside of AS1. The source router should forward packet to gateway router in AS1; but which router to forward to? Inter-domain routing must learn what destinations reachable through AS2 & AS3. This reachability information must be propagated to all routers in AS1 for intra-domain routing.

The Internet. Intra-AS (Routing within ISP): OSPF (Open Shortest Path First); Link-State Routing. Inter-AS (Routing between ISPs): BGP (Border Gateway Protocol).

1.6 Link Layer Protocols

1.6.1 Introduction

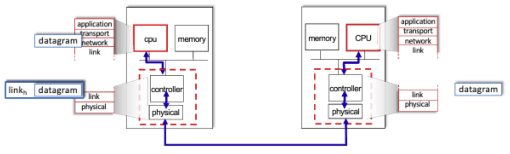
The Link Layer. The Link Layer does not care about origin or ultimate destination. Its only job is to get you from the start of a link to the end of a link uncorrupted and secure. Frames make up the Link Layer PDUs, which encapsulate network layer datagrams.

Services. Framing: Encapsulating the network layer datagram into a frame, adding a header and trailer. Addressing: Add "MAC" Addresses in the frame headers to identify source and destination (different from IP Address). Medium Access: Handle link access if it's a shared medium. Error Detection: Errors caused by signal attenuation and noise. Receiver detects errors, signals retransmission, or drops frame. Much more powerful mechanism on Link Layer than Network/Transport. Error Correction: Receiver identifies and corrects bit error without retransmission. Flow Control: Pacing transmission speeds between adjacent sending and receiving nodes. Reliable Delivery: Seldom used on links with low BER, but more on wireless links with high error rates.

MAC vs IP. Why both MAC and IP addresses? "Back in time, when computer networks were created: some computers would communicate with each other to share something. To do that, they needed to know whowas talking and whowas being talked to. So, instead of giving each computer a name, we gave them an ID. This ID is called the MAC address, and uniquely identifies each computer." (In fact, each network card, but back in that time you could think of one MAC address for each computer). There wasn't a unique specification on how computers would talk to each other: many protocols appeared: TCP/IP, IPX/SPX, and so on. Each protocol would specify things as they thought appropriate. For example, IPX/SPX would address each computer using the MAC address and some more information. But the TCP/IP protocol was designed a little bit different: they decided that having a virtual address, made of 4 bytes (0.0.0.0 to 255.255.255.255) was enough, and was even easier to manage. It didn't matter if your network cards all had similar MAC addresses to be logically grouped, they would be grouped using the IP address: for example, all IP addressed that began with 10.0.x.x. would part of the engineering group, and those of 10.1.x.x. are the printers, etc. IP addresses thus became the addresses used to route data between computers (and subnets). However, network devices still need to know to which network card a message is going to, so they, translate the IP addresses to MAC address. *Why not eliminate MAC Addresses and retain IP: All devices have a MAC Address that is unchanging, but IP Addresses are dynamically assigned and may change.*

Link Layer Implementation. The Link Layer is implemented partly in software and hardware. The software part is implemented in the OS and runs on the host CPU. The hardware part (and on-chip firmware) of the Link Layer is implemented on the Network Interface Card (NIC). The NIC is a Ethernet or WiFi Adapter which implements the link layer and physical layer functionalities. It attaches into the host's system buses through extension slots.

Figure 28: Communicating Interfaces



Sending side:

- Encapsulates datagram in frame
- Adds error checking bits, reliable data transfer, flow control, etc.

Receiving side:

- Looks for errors, reliable data transfer, flow control, etc.
- Extracts datagram, passes to upper layer at receiving side

Addressing. Datagrams from the network layer contain IP Addresses. Each datagram must first be enclosed in a frame before being transmitted. Frames contain MAC Addresses - 48-bits represented by 12 hexadecimal digits. No two NICs have the same MAC address. IEEE allocates the chunk of $2^2 4$ addresses by first fixing the 24 bits of a MAC address and letting the company create unique combinations of the last 24 bits for each adapter.

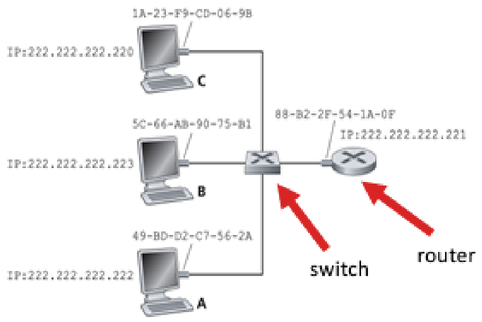
Network Layer Datagrams and Link Layer Frames. Having different addressing on the Link Layer decouples it from the Network Layer, allowing innovation on the network layer. Link Layer frames are also specific to Link Layer protocols, which provide physical layer specifications. The layer control and physical medium characteristics are closely linked.

IP-based routes are made of MAC-based hops. An IP datagram from source to destination retains the same source and destination addresses across the entire path. The source and destination MAC addresses change at every hop. An intervening router makes forwarding decisions based on the destination IP address, determining an outgoing interface. Once the interface is determined, the appropriate MAC address are used to make the hop.

Routers forward packets based on IP addresses, but using MAC addresses. A datagram with the destination IP 10.0.1.1 arrives at the router. The router uses Longest Prefix Matching to determine 10.0.1.254 as the Next-Hop Router's IP. The Network layer at the router tells the Link Layer to construct a frame with the MAC of 10.0.1.254 as the destination address, with its own MAC as the source. This requires IP to MAC translation.

IP to MAC Address translation is also required at the hosts in a subnet. Each NIC on a subnet has a unique 48-bit MAC address and a locally unique 32-bit IP address. Translation between the two is needed.

Figure 29: If C wants to send a datagram to A, it needs to know the MAC address corresponding to A's IP.



1.6.2 Address Resolution Protocol

Introduction. How do we determine a destination's MAC address, knowing its IP address? Each IP node (host, router) has an ARP table. ARP Tables contain IP to MAC address mappings - [IP, MAC, TTL]. TTL is the time after which address mapping will be forgotten (typically 20 minutes).

ARP in Action. For example, if Node A wishes to send a datagram to Node B. Scenario 1: B's IP-to-MAC mapping exists in A's ARP table. The frame is created and transmitted. Scenario 2: B's IP-to-MAC mapping does not exist in A's ARP table. The ARP protocol is triggered.

ARP Steps

1. A broadcasts an ARP query, containing B's IP Address. The destination MAC address is FF-FF-FF-FF-FF-FF; all nodes on the LAN receive the ARP query.
2. B replies to A with ARP response, giving its MAC address.
3. A receives B's reply, adding B's entry into its local ARP table.

Routing to another subnet. A creates an IP datagram with IP source A and destination B. A creates a link-layer frame containing A-to-B IP datagram. R's (Router) MAC address is the frame's destination. The frame is sent from A to R. The datagram is removed and passed up to IP. R determines the outgoing interface and passes the datagram with IP source A and destination B to link layer. R creates a link layer frame containing the A-to-B IP datagram. The frame destination address is B's MAC address. It transmits the link-layer frame. B receives the frame and extracts IP datagram destination B. B passes datagram up protocol stack to IP.

1.6.3 Switches

Introduction. We need switches to create non-trivial subnet topologies. It is transparent to other devices on the subnet. If C wishes to send a frame to A, the frame "passes through" the switch. The switch looks at the destination MAC address and forwards the frame to the appropriate outgoing interface. Just like a router, a switch needs a forwarding table, but the mapping is done using MAC addresses. *Switch vs Hub* - A hub practically extends the same channel (cable), but a switch creates multiple distinct logical channels.

Forwarding Tables. Each entry in a switch's forwarding table maps a destination MAC address to an outgoing switch interface.

Self-Learning. A frame comes in through interface x; the switch consults its table for the destination MAC of the frame. If a row exists for that destinatioon

MAC address, the frame is simply forwarded to the corresponding interface, say y. If no row exists for the destination MAC address, the switch simply forwards that frame to all its interfaces (except x, where the frame came from). The switch also looks up the table for the row for the source MAC address as well: (1) If an entry does not exist for that MAC address, it adds a row for that MAC address, sets the interface value to x and updates the time entry to the current time. (2) If an entry exists for that MAC address with interface entry x, the switch simply updates the time in that row to the current time. Each row is deleted after a certain TTL. Thus, for an empty forwarding table, if B->C and C->A, only B and C's MAC addresses get registered.

Topologies. A bus topology uses a single coaxial trunk cable that runs between all the nodes on the network. When a new node is added a t-connector is used to connect the new node to the cable. A star topology is designed with each node (file server, workstations, and peripherals) connected directly to a central network hub or switch. each node (file server, workstations, and peripherals) connected directly to a central network hub or switch.

1.6.4 MAC Sub-layer: Multiple Access Protocols

Link Types. Links may be classified as (1) Point-to-point, consisting of a single sender and receiver, and (2) Broadcast, having multiple sending and receiving nodes all connected to a single, shared broadcast channel.

Multiple access protocols. Single shared broadcast channel means that two or more simultaneous transmissions by nodes will cause interference. Collision is detected by a node that receives two or more signals at the same time. A multiple access protocol includes a distributed algorithm which determines how nodes can share the channel (which node can transmit), and communication between nodes regarding channel-sharing, which itself uses the same channel (so no out-of-band channel for coordination).

An ideal protocol. Given a broadcast link of rate R bps, an ideal protocol should have the following characteristics: (1) When only one node wants to transmit, it should get a throughput of R bps. (2) When M nodes want to transmit simultaneously, each should experience an average throughput of R/M bps. (3) The protocol should be fully decentalised to avoid a single point of failure. (4) The protocol should be simple and therefore inexpensive to implement.

Protocol Types. Channel Partitioning - dividing the channel into smaller "sub-channels" (based on timeslots, frequency or code). Each sub-channel should be allocated to a node for exclusive use, making collision impossible. Random access - don't divide the channel (so collisions can happen). Nodes may send data randomly when required. A mechanism is in place to recover from collisions. Taking Turns - nodes may take turns on the same channel(s). Nodes with more to send may be given longer turns.

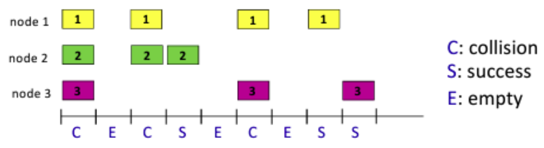
Time Division Multiple Access (TDMA). Access channel in rounds (time). Each node gets a fixed length slot in each rounds. Unused slots go idle. Example: in 6-node LAN, 1, 3 and 4 have packets to send; 2, 5 and 6 will go idle. TDMA has the drawback that a node is limited to an average rate of R/N bps even when it is the only node with packets to send. A second drawback is that a node must always wait for its turn in the transmission sequence—again, even when it is the only node with a frame to send.

Frequency Division Multiple Access (FDMA). Channel spectrum divided into frequency bands. Each node is assigned a fixed frequency band. Unused transmission time in a frequency band goes idle. The drawback of TDMA and FDMA is that there is idle channel time, reducing channel utility.

Random Access Protocols. When a node needs to send a packet, it transmits at full channel data rate R. There is no a-priori coordination done among nodes (truly random). Two or more transmitting nodes can cause a collision. A Random Access Protocol needs to specify how to detect and recover from collisions.

Slotted ALOHA. Assumptions: All frames have the same size. Time is divided into equal sized slots. Nodes start to transmit only at the beginning of a slot. Nodes are synchronized with slot timings (so they know when they start transmitting). If two or more nodes transmit at the start of a slot, collision occurs (and all nodes detect this collision). Operation: When a node obtains a fresh frame to transmit, it transmits it at the start of the next slot. If no collision occurs, the node is ready for further transmission into the next slot. If collision occurs, the node retransmits the frame in each subsequent slot with probability p until success (usually same p for all nodes).

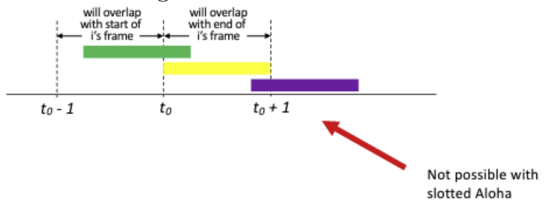
Figure 30: Slotted ALOHA.



Slotted ALOHA - Analysis. Pros: A single active node can continuously transmit at full rate on the channel. The protocol is highly decentralised; nodes only need to sync with slot times. It is simple to implement. Cons: Idle slots occur due to probabilistic nature of retransmissions. Nodes need to be able to detect collision in less than the time to transmit a packet. Clock synchronization is needed. Efficiency: suppose that there are N nodes with many frames to send, and each retransmits with probability p. The probability of a given node successfully transmitting in a given slot is $p(1-p)^{N-1}$. The probability of any node transmitting in a given slot is $Np(1-p)^{N-1}$. To maximise efficiency, we find p* that maximises $Np*(1-p*)^{N-1}$. For many nodes, we take limit of $Np*(1-p*)^{N-1}$ as N goes to infinity, which gives the maximum efficiency of 1/e or /37. Thus, at best, the channel is used for useful transmissions 37% of the time.

Pure ALOHA. Pure ALOHA is unslotted. No synchronisation with slots is required. When a frame first arrives at a node, it is transmitted immediately. Collision probability increases with no synchronisation. Frames sent at t_0 collide with other frames sent in [t_0-1, t_0+1]. So, frames collide not only with frames sent at the exact same time, but also with frames sent before or after it.

Figure 31: Pure ALOHA.


will overlap with start of i's frame
will overlap with end of i's frame
t_0 - 1
t_0
t_0 + 1
Not possible with slotted Aloha

Pure ALOHA - Analysis. P(Success by given node) = P(Node transmits)*P(No other node transmits in [t_0-1,t_0])*P(No other node transmits in [t_0-1,t_0]) = p$(1-p)^{2(N-1)}$. Choosing optimum p and letting N go to infinity , P(success) = .18 (worse than slotted ALOHA).

Carrier Sense Multiple Access (CSMA). Simple CSMA: Listen before transmit. If the channel is sensed as idle, transmit entire frame. If the channel is sensed as busy, defer transmission (by a random period of time). CSMA/CA (CSMA with Collision Detection): Collisions detected within short time. Colliding transmissions aborted, reducing channel wastage. Easy in wired but difficult in wireless networks.

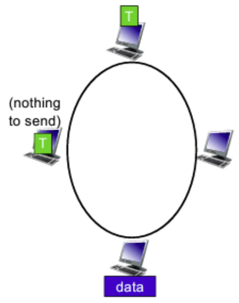
Ethernet CSMA. (1) NIC receives datagram from network layer, creating a frame. (2) NIC senses channel: if the channel is idle, NIC starts frame transmission. If the channel is busy, the NIC waits until the channel is idle then transmits. (3) If NIC transmits entire frame without collision, NIC is done with that frame. (4) If NIC detects another transmission while sending, it aborts transmission, sending a jam signal. (5) After aborting, NIC enters binary (exponential) backoff; after the mth collision, NIC chooses K at random from 0, 1, 2...2^m-1. NIC waits and returns to step 2. More collisions mean longer backoff interval. A jam signal also notifies all stations on the network; they must wait a short period of time before attempting to transmit again.

CSMA/CD Efficiency. t_prop = Maximum propagation delay between 2 nodes in LAN. t_trans = time to transmit maximum size frame. Efficiency = $\frac{1}{1+5t_{prop}/t_{trans}}$. Efficiency goes to 1 as t_prop goes to 0 and as t_prop goes to infinity. The performance is better than ALOHA and it is simple, cheap and decentralised.

Taking Turns Protocols. Polling: A master node invites other nodes to transmit in turn. It is typically used with ”dumb” devices. The concerns are polling overhead, latency and there being a single point of failure. The master node polls each of the nodes in a round-robin fashion. In particular, the master node first sends a message to node 1, saying that it (node 1) can transmit up to some maximum number of frames. After node 1 transmits some frames, the master node tells node 2 it (node 2) can transmit up to the maximum number of frames. (The master node can determine when a node has finished sending its frames by observing the lack of a signal on the channel.) The procedure continues in this manner, with the master node polling each of the nodes in a cyclic manner. The polling protocol eliminates the collisions and empty slots that plague random access protocols. This allows polling to achieve a much higher efficiency.

Token Passing. Control token passed from one node to next sequentially. Concerns: Token overhead, latency and single point of failure if any node crashes in between. When a node receives a token, it holds onto the token only if it has some frames to transmit; otherwise, it immediately forwards the token to the next node. If a node does have frames to transmit when it receives the token, it sends up to a maximum number of frames and then forwards the token to the next node. Token passing is decentralized (no master node) and highly efficient.

Figure 32: Token Ring.



2 Databases

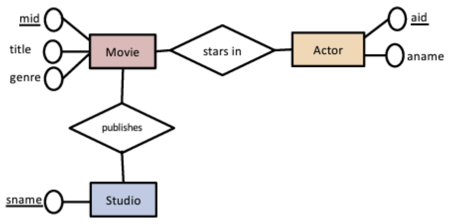
2.1 Entity-Relationship Modelling

Entity Relationship Model (ERM). An ERM is a conceptual model of data that is about identifying the entities which make up the data and the relationships between these entities. Components include: Entities (Things in the system), Relationships (How they relate to each other) and Attributes (Properties of entity/relationship).

Component	Description	Represented by
Entity Set	Entities (same set of properties/noun)	Rectangle
Relationship Set	How two or more entity sets relate (verb)	Diamond
Attribute	Property (Primary Keys - underlined)	Circle

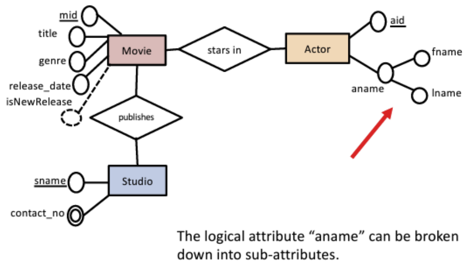
Primary Key. A Primary Key is an attribute which uniquely identifies an entity in an entity set. There must never exist two entities with the same key. Primary keys are shown as underlined attributes. They may be *natural* or *surrogate*. Natural keys are attributes that arise from the application data. Surrogate keys are invented attributes only for use within the database. Primary keys may be composed of multiple attributes (e.g. `firstname, lastname, dateofbirth`). We usually want to minimise the number of attributes. Note: in the example below, movie names repeat and hence we need a surrogate key, which is an internally generated unique key.

Figure 33: Simple ERD.



Complex Attributes. Computed Attributes: Properties calculated from other attributes. Multi-valued Attributes: Sets or Lists of multiple values. Composite Attributes: Logical properties that break down into sub-attributes.

Figure 34: ERD with complex attributes.

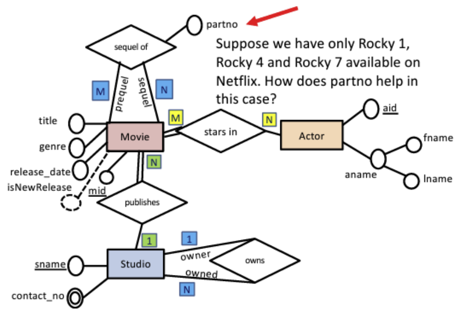


Participation Constraints. Total Participation - every movie must be starred in by an actor. Partial Participation - not every actor needs to star in a movie. Represented by double lines.

Cardinality Constraints. One-To-Many / Many-To-One: A studio may publish any number of movies, but a movie is published by a single studio. Many-to-Many: An actor may star in any number of movies, and a movie may be starred in by any number of actors.

Self-Relationships. Some binary relationships can involve the same entity set twice. A movie may be a sequel of another movie etc. Relationships can also have their own attributes (Relationship Set Attributes).

Figure 35: ERD with more complex relationships.



Ternary Relationships. Some relationships involve more than two entity sets. E.g. *studios* may promote *movies* on Netflix featuring their *actors*. These may also be replaced with binary relationships by being re-drawn as a set of binary relations, using a *weak entity*. A weak entity is one that does not have its own primary key.

Figure 36: ERD with Ternary Relationships

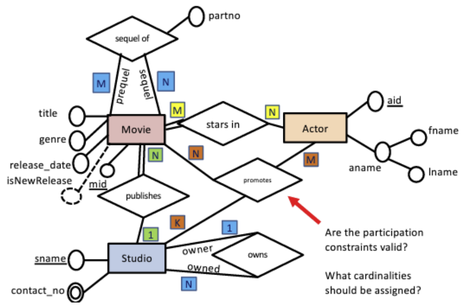
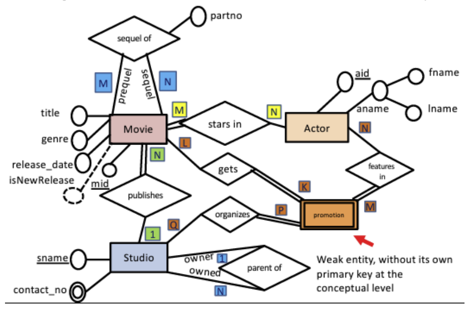
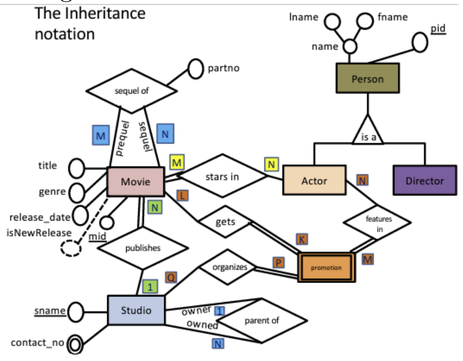


Figure 37: ERD with Weak Entity



Inheritance. Modelled as a binary 'is a' relationship with full participation on the child side and a 1-1 cardinality.

Figure 38: ERD with Inheritance



2.2 Relational Model

Overview. An approach to managing data where all data is represented in terms of tuples (rows) in relations (tables). Relationships are implemented by joining together different tables based on common attributes. We can apply further filters afterwards to produce specific information. The basic components: every field in the table has a unique name. Each field stores a single item of data. When implemented as a physical model, each field has a particular data type - e.g. Boolean, Integer etc.. Furthermore, each field will also have validation rules (e.g. field cannot be NULL).

Basic Components. Every table must have one or more fields which uniquely identify a record in the table - this is called the primary key. Two tables may be joined through key fields. The key field of one table must appear as the foreign key of the other table. *Referential Integrity*: Referential integrity refers to the relationship between tables. Because each table in a database must have a primary key, this primary key can appear in other table because of its relationship to data within those tables. Referential integrity is the logical dependency of a foreign key on a primary key. Moreover, when a table B has a foreign key to another table A, referential integrity requires that any foreign key value in table B must refer to an existing record in table A.

Relational Schema. The name of a relation and its set of attributes together is called the schema of that relation. E.g. Movies(title, year, length, genre) means that the table movies contains rows each of which have 4 attributes namely title, year, length and genre. The schemas of the entire set of relations of a database make up the relational database schema.

Migration from ERM to RM:

1. Strong entity sets become relations.

2. Weak entities become relations, and borrow keys from all entities they participate in relationships with.
3. For 1-to-many relationships, we migrate the key from the 1-side to the many-side.
4. For 1-to-many relationships with partial participation on the many-side, we create a separate relation and migrate keys into it from both sides.
5. For many-to-many relationships, we create new relations and migrate keys from both sides.
6. Relationship attributes go to appropriate relations.
7. Multi-valued attributes get their own relations, because every field in a relation must be atomic.

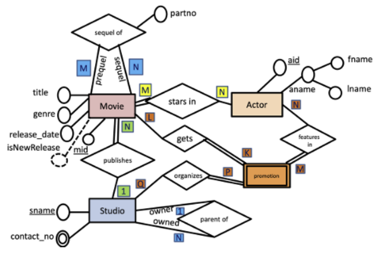


Figure 39: Migration from ERM to RM

Movie(mid, title, genre, release_date, sname**)**
Actor(aid, fname, lname**)**
Studio(sname**)**
Owns(sname, psname)
Promotion(mid, aid, sname)
Sequel_of(mid, prid, partno**)**
Stars_in(mid, aid)
Contact(sname, contactno)

Relational Model Refinement. Redundancies: Large tables are likely to contain redundancies. Insertion Anomaly: Values of attributes are not known at insertion times and need to be assigned NULL values. E.g. when a new employee is added to the table `EMP_DEPT`, they may not have a department assigned to them; thus some values will be NULL. Deletion Anomaly: Deleting one part of the information delets another. If some employees leave the organisation, we could lose department information, for example. Updation anomaly: If we update the value of one attribute of a particular department, then we must update the tuples of all employees who work in that department. E.g. if we change the department manager, we will need to update all instances of their SSN in the `Dmgr_ssn` column.

Guidelines for good relational schema. Do not combine attributes from multiple entity types and relationship types into a single relation. Design the relation schema so that no insertion, deletion, or updation anomalies are present in the relations. As far as possible, avoid placing attributes in a relation whose values may frequently be NULL. Minimise redundant copying of attribute values in relations.

Normalisation. Normalisation can be considered a process of analysing the given relational schema based on their *functional dependencies* and *primary keys* to minimise redundancy and minimise anomalies. Relations can be taken to the 1st, 2nd, 3rd and BCNF normal forms, which are at increasing stages of refinement.

Functional Dependencies. Let X and Y be two subsets of attributes of a relation R (set of all attributes in the relation). A functional dependency, denoted by X→Y specifies the following constraint: for any two tuples t_1 and t_2 in R, if $t_1[X] = t_2[X] \rightarrow t_1[Y] = t_2$[Y]. In other words, given the values of attributes in X, we uniquely determine the values of attributes in Y. *Corollary*: Let P be the set of primary key attributes. Then, P → R is true by definition.

Types of Functional Dependencies. Full - A functional dependency is full if removal of any attribute A from X means that the dependency does not hold anymore. Partial - A functional dependency is partial if after the removal of some attribute A from X the dependency still holds.

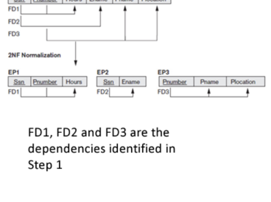
Additional Terms and Concepts. If a relation has more than one key, each is called the *candidate* key. One of the candidate keys is arbitrarily designed to be the primary key. The other non-primary keys are called *secondary* keys. An attribute of a relation R is called a *prime* attribute of R if it is a member of some candidate key of R. Attributes that are not prime are called *non-prime* attributes.

First Normal Form (1NF). 1NF requires that a relation have no composite attributes or multivalued attributes. In other words, each attribute value in the table must be atomic. We've already ensured this by implementing multivalued attributes as separate tables.

Second Normal Form (2NF). 2NF requires that the relation is in 1NF. Furthermore, every non-prime attribute in R is fully functionally dependent on the primary key of R.

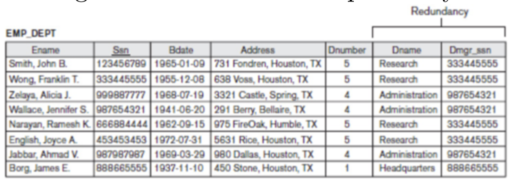
1NF to 2NF Normalisation. (1) Identify all functional dependncies whose left sides (determinants) are proper subsets of the primary key, and right sides are non-prime attributes. (2) Break the table accordingly. *Informally, the process may be thought of as splitting the table based on the pieces of the primary key that can fully determine those pieces of the table.*

Figure 40: 2NF Normalisation



Transistive Dependencies. Even in 2NF, tables can still have redundancies. Once `Dnumber` is determined by `SSN`, the `Dname` and `Dmgr_ssn` values are always the same as they are determined by `Dname`. This creates a transistive dependency. The functional dependency X → Y in a relation R is a transitive dependency if there is a set of attributes Z that is neither a candidate key nor a subset of any key of R, and both X → Z and Z → Y hold.

Figure 41: Transistive Dependency

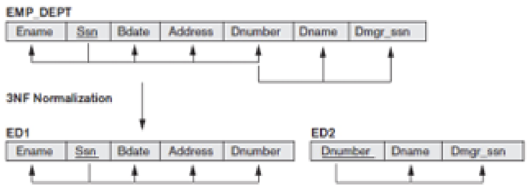

Redundancy

EMP_DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 FireOak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555

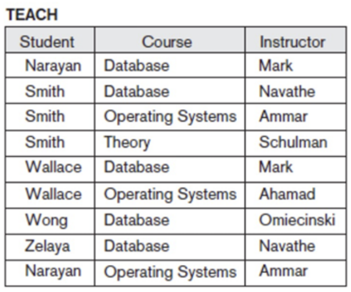
Third Normal Form (3NF). 3NF requires that the relation is in 2NF, and that no nonprime attribute in the relation is transistively dependent on the primary key. More precisely, no nonprime attributes Z exist such that for the primary key X, X → Z and Z → X. To convert from 2NF to 3NF - if there are nonprime attributes Z, Y in the relation, such that for the primary key X, X → Z and Z → Y: (1) Split the relation on Z and (2) Repeat the process on resultant tables.

Figure 42: 2NF to 3NF Normalisation



Boyce-Codd Normal Form (BCNF). 3NF can be further improved to a stricter normal form called BCNF. The table below is in 3NF but still seems to have redundancies. BCNF requires that the relation be in 3NF, and that all determinants (LHS of dependencies) are candidate keys, or possible primary keys. The table below is in 3NF and not BCNF because of the dependencies: {Student, Course} → Instructor and Instructor → Course (Instructor is not a candidate key, but a determinant in one of the dependencies). To normalise from 3NF to BCNF - if there is a non-prime attribute Z and a prime attribute X, such that Z → X exists, split the table on Z.

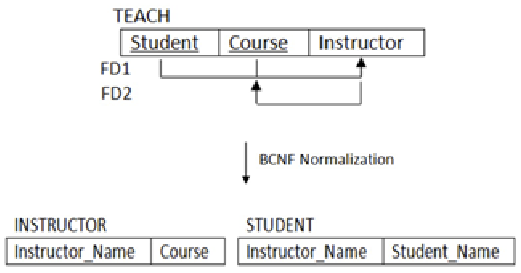
Figure 43: 3NF Table



TEACH

Student	Course	Instructor
Narayan	Database	Mark
Smith	Database	Navathe
Smith	Operating Systems	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating Systems	Ahamad
Wong	Database	Omiecinski
Zelaya	Database	Navathe
Narayan	Operating Systems	Ammar

Figure 44: 3NF to BCNF Normalisation



Conclusion. The 2NF and 3NF forms removed redundancies in the non-prime attributes. If a nonprime attribute is repeating for a part of the primary key, going to 2NF removes that redundancy. Then, if a nonprime attribute is repeating with another non-prime attribute, going to 3NF removes that redundancy. What is left is redundancy in prime attributes, which is taken care of by going to BCNF.

2.3 Relational Algebra

We need a set of operators to allow us to combine and filter the information in various tables, to implement high-level functions of the system. We use a set of mathematical operators - relational algebra operators.

Operators. Set - Union, Intersection, Difference, Cartesian Product. Database - Select, Project, Join, Aggregate, Division, Rename.

Union Operator. Written as (r + s), (r U s). Relations must be union compatible and duplicate rows are eliminated.

Intersection Operator. Written as (r ∩ s). Relations must be intersection compatible.

Difference Operator. Written as (r - s). Relations must be difference compatible. This includes rows that are in r but not in s.

Projection Operator. Extracts certain columns from the table.

Selection Operator. The selection operator extracts certain rows from the table and discards the others. Retrieved tuples must satisfy a given *filtering condition.*

Cross Join. Written as (r x s). Concatenates rows from two relations, making all possible combinations of rows.

Inner Join. The join operation, denoted by (r $\bowtie_{COND}$ s), is used to combine related tuples from two relations. Here, COND is the matching condition. The example below demonstrates the operation (r $\bowtie_{C=D}$ s). We can think of this as a cross join with a filter.

Natural Join. Denoted by (r * s). It combines tuples of two relations using an implicit condition, i.e. tables are related by columns that have the same names and domains. A Natural Join is simply an inner join in which the columns in the condition are chosen by default to be *the ones with the same names.* The reason for having a separate join operator is simply because often, the primary key of one table and the focus of another have the same name, hence a natural join simplifies the syntax.

Left Outer Join. Denoted by (r $=\bowtie_{COND}$ s). It combines tuples of two relations and keeps in the result of every tuple from the left table, but only those from the right table that meet the join condition.

Aggregation. Syntax: $_{<GroupingAttribute}F_{<FunctionList>}(RelationName)$. The function list contains simple mathematical functions (MAX, MIN, AVG, SUM, COUNT). A column name may be given as a grouping attribute to fragment the function outputs into groups.

Rename. Syntax: ρ(c1, c2...ck) (E). Rename the columns of E to c1, c2...ck respectively. Often used with aggregation as we need to be able to unambiguously refer to the columns that store the results of the aggregate functions.

RDBMS. A database is a structured collection of data that is managed by a DBMS (Database Management System). Some of the most powerful and widely used database management systems are relational: RDBMS. They provide a high-level query language called SQL, which is based on relational algebra.

ACID. All modern RDBMs ensure ACID properties.

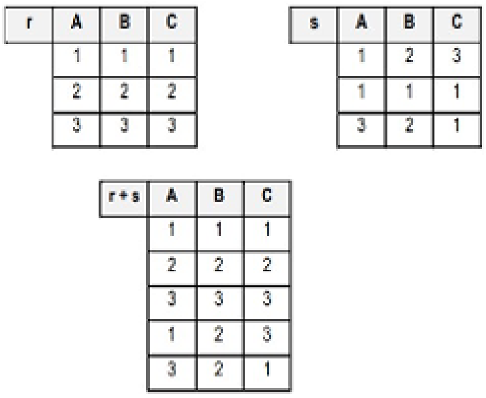
1. Atomicity - transactions, which consist of multiple statements, are treated as atomic; either the whole thing succeeds, or whole thing fails.
2. Consistency - transactions can only bring the database from one valid state to another. A state is valid if it follows all the specified rules including constraints.

3. Isolation - ensures that concurrent execution of transactions leaves the database in the exact same state that would have been obtained if the transactions were executed sequentially.

4. Durability - database comits are final and durable even if the system fails after a commit.

Author's Note. I've decided not to include content from the SQL chapters as SQL expertise is better obtained through practice.

Figure 45: Union

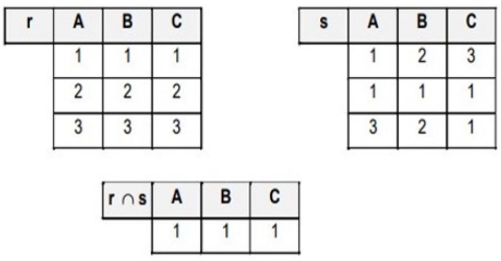


r	A	B	C
	1	1	1
	2	2	2
	3	3	3

s	A	B	C
	1	2	3
	1	1	1
	3	2	1

r+s	A	B	C
	1	1	1
	2	2	2
	3	3	3
	1	2	3
	3	2	1

Figure 46: Intersection

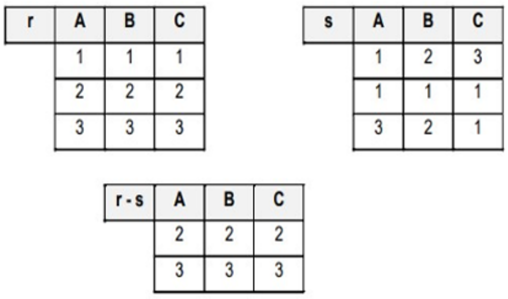


r	A	B	C
	1	1	1
	2	2	2
	3	3	3

s	A	B	C
	1	2	3
	1	1	1
	3	2	1

r ∩ s	A	B	C
	1	1	1

Figure 47: Difference



r	A	B	C
	1	1	1
	2	2	2
	3	3	3

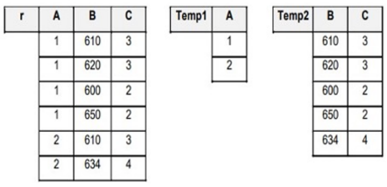
s	A	B	C
	1	2	3
	1	1	1
	3	2	1

r - s	A	B	C
	2	2	2
	3	3	3

Figure 48: Projection

$Temp1 = \prod_{A}(r)$

$Temp2 = \prod_{B,C}(r)$



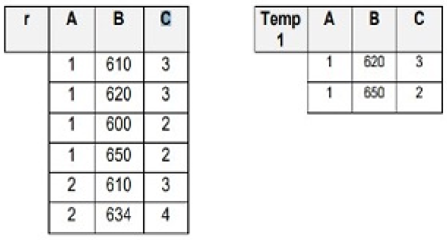
r	A	B	C
	1	610	3
	1	620	3
	1	600	2
	1	650	2
	2	610	3
	2	634	4

Temp1	A
	1
	2

Temp2	B	C
	610	3
	620	3
	600	2
	650	2
	634	4

Figure 49: Selection

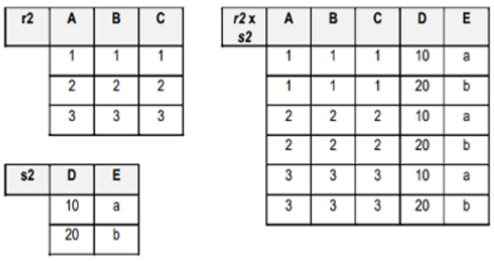
$Temp1 = \sigma_{(B >=620) \text{ and } (c<4)} \ (r)$



r	A	B	C
	1	610	3
	1	620	3
	1	600	2
	1	650	2
	2	610	3
	2	634	4

Temp 1	A	B	C
	1	620	3
	1	650	2

Figure 50: Cross Join

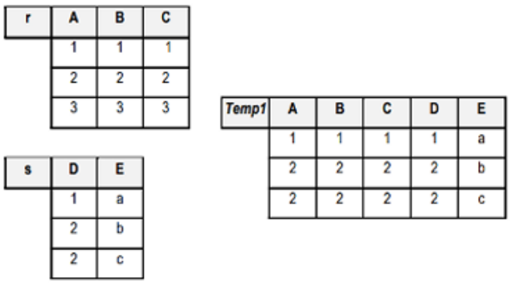


r2	A	B	C
	1	1	1
	2	2	2
	3	3	3

s2	D	E
	10	a
	20	b

r2 x s2	A	B	C	D	E
	1	1	1	10	a
	1	1	1	20	b
	2	2	2	10	a
	2	2	2	20	b
	3	3	3	10	a
	3	3	3	20	b

Figure 51: Inner Join

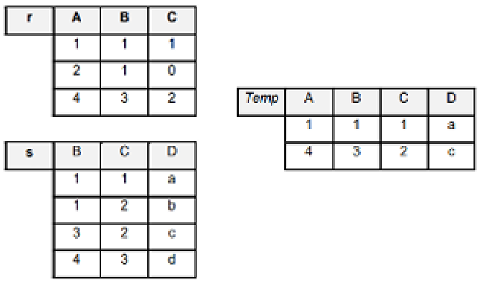


r	A	B	C
	1	1	1
	2	2	2
	3	3	3

s	D	E
	1	a
	2	b
	2	c

Temp1	A	B	C	D	E
	1	1	1	1	a
	2	2	2	2	b
	2	2	2	2	c

Figure 52: Natural Join

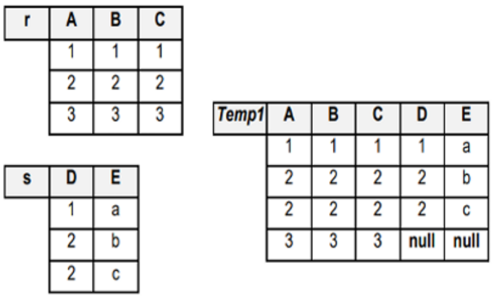


r	A	B	C
	1	1	1
	2	1	0
	4	3	2

s	B	C	D
	1	1	a
	1	2	b
	3	2	c
	4	3	d

Temp	A	B	C	D
	1	1	1	a
	4	3	2	c

Figure 53: Left Outer Join



r	A	B	C
	1	1	1
	2	2	2
	3	3	3

s	D	E
	1	a
	2	b
	2	c

Temp1	A	B	C	D	E
	1	1	1	1	a
	2	2	2	2	b
	2	2	2	2	c
	3	3	3	null	null