Justin Ciocoi

Oct. 14, 2023

CSCI 379: Computer Networking

Midterm Review

Chapter 2: Application Layer

- In chapter 2, a variety of key application-layer concepts are defined, including:
  - *Network services* required by applications
  - *Clients and Servers*
  - *Processes*
  - *Transport-Layer Interfaces*

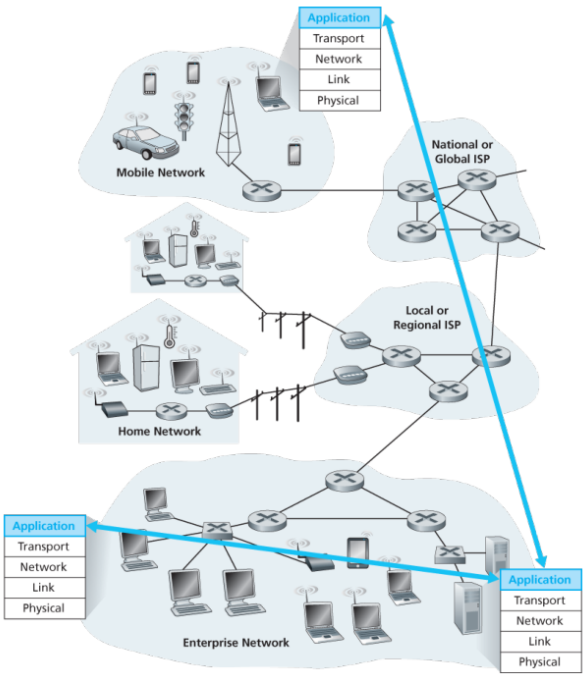


Figure 2.1 Communication for a network application takes place between end systems at the application layer

2.1: Principles of Network Applications

- **The Client-Server Architecture**
    - In the client-server model, there is an *always-on* server that can be accessed by a client at the client's convenience
    - The *server* has a *unique, fixed* ***IP Address***
    - Many of the most popular web services operate using this architecture
- **P2P Architecture**
    - In this model, there is no reliance on a central server which can be accessed, and instead network tasks are achieved through intermittently connected hosts called ***peers***
    - This model is used in things like torrenting, where a file housed on another end system of the same variety is downloaded, and in internet telephony applications like Skype and VOIP
    - The P2P model provides a much better level of scalability, as it is *self-scalable* since an additional user will also provide additional resources and thus the resource pool *should* grow at a rate proportional to resource use

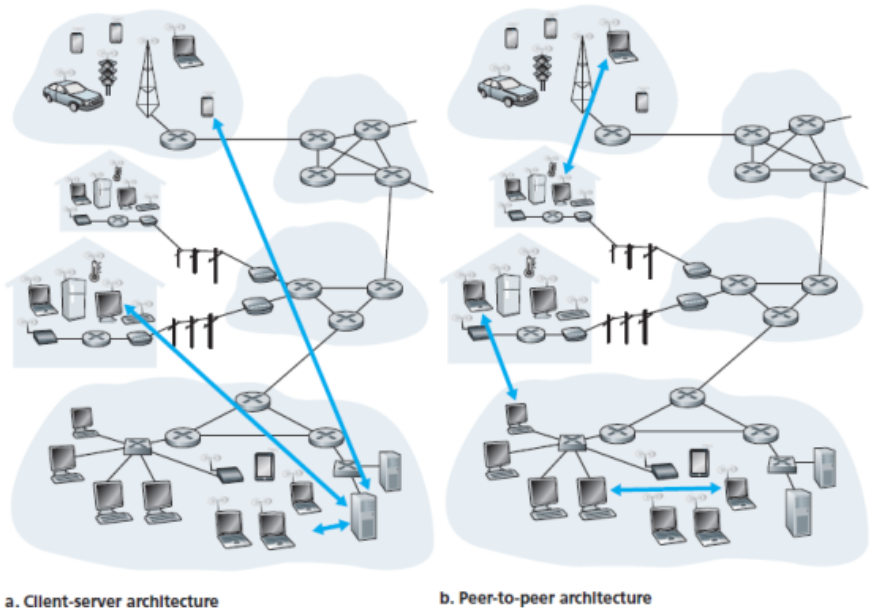


Figure 2.2 (a) Client-server architecture; (b) P2P architecture

- **Inter-Process Communication**
    - When two processes on *different* end systems want to communicate, they must do so over the network

- To do this, a process on an end system will *send and receive* messages through a software interface called ***socket***, also called a *port*
- A socket is the interface between the application layer and the transport layer within a host
- It can also be referred to as the *Application-Programming Interface*, or ***API***
- A destination process can be identified using an IP address to identify the end system, and then a socket, or port number specifying the destination process on said end system

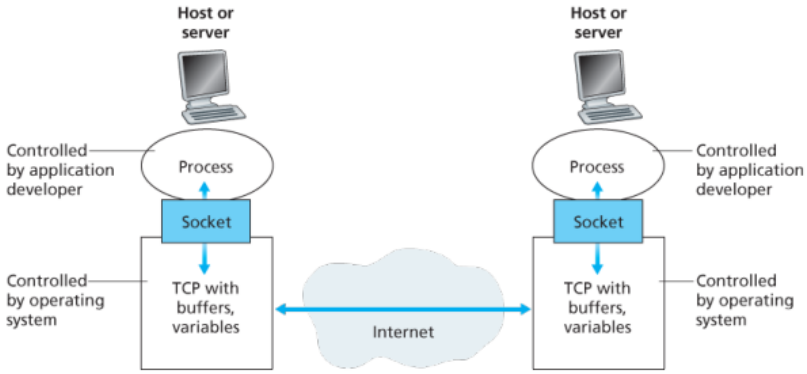


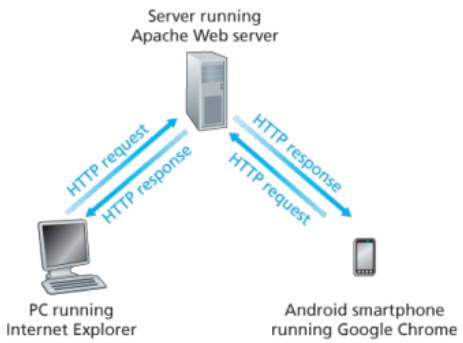
Figure 2.3 Application processes, sockets, and underlying transport protocol

- **Properties of Transport Services**
  - *Reliable Data Transfer*
    - There is always the potential for packets to get lost when traveling across a network
    - Therefore, it can be advantageous for a transport protocol to provide a guarantee of data delivery, which we can call reliable data transfer
    - Certain applications are *loss-tolerant* which means they can tolerate the loss of some packets
      - This includes things like music or video streaming, where a lost packet can manifest in buffering or bit-rate lowering
    - Other applications, like document downloading or email require a reliable data transfer
  - *Throughput*
  - *Timing*
  - *Security*

- **Transport Services Provided by the Internet**
  - *TCP Services*
    - *Connection-Oriented Service*, which refers to the three way handshake involved in TCP, initiating a connection until it is told to be severed
    - *Reliable Data Transfer*, which refers to TCP's guarantee of data delivery
  - *UDP Services*
    - UDP is a *best-effort, connectionless* protocol

2.2: The Web and HTTP

- **Hyper Text Transfer Protocol (HTTP)**
  - HTTP is implemented in both a client-side and server-side program, which run on different end systems and communicate using HTTP messages
  - HTTP uses TCP as its underlying protocol rather than UDP
  - Once HTTP sends the message into a socket, TCP takes over and therefore HTTP does not have to worry about doing the job of reliable data transfer
  - HTTP is a *stateless protocol,* which means the server does not store any state information about the client



- **Persistent and Non-Persistent Connections**
  - HTTP with non persistent connections will establish a new connection for each HTTP message that is being sent back and forth between client and server
  - This can be inefficient due to the need for a TCP handshake for each web object, whereas a persistent connection can send multiple objects over the same TCP

connection

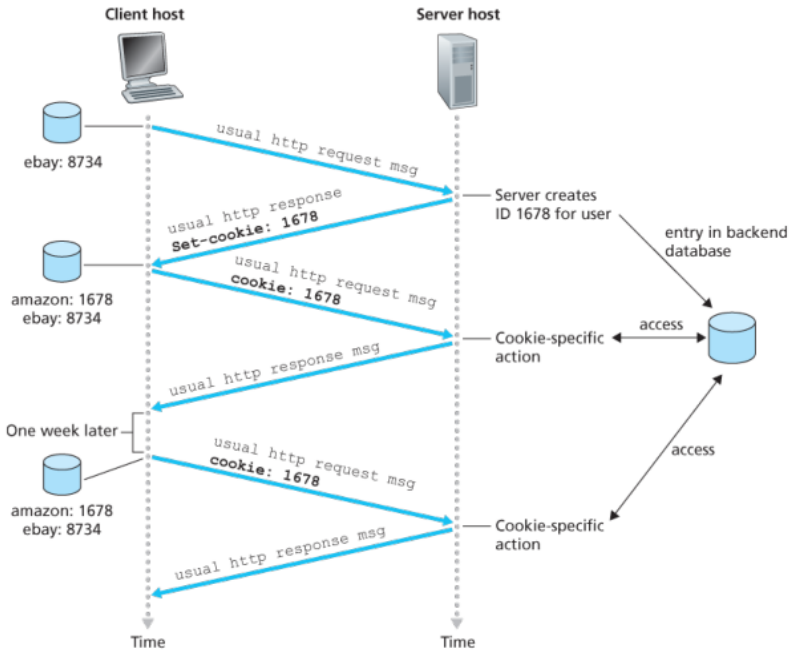
    - A persistent connection will be closed after a variable *time-out* time during which the connection in question is not used

- **HTTP Message Format**
    - *HTTP Request Message*
- ```
  GET /somedir/page.html HTTP/1.1
  Host: www.someschool.edu
  Connection: close
  User-agent: Mozilla/5.0
  Accept-language: fr
  ```
    - The first line includes the *Method*, in this case `GET`, the *File URL*, and the *HTTP Version*
    - The second line specifies the *host IP Address*
    - The third line identifies whether it is a *persistent or non-persistent connection*
    - The fourth line specifies *the program operating on the client side*
    - The last line indicates *language preference* for the response message
- *HTTP Response Message*
    - The response message is very similar to the request message, but it contains a status code and message corresponding to the site's status
    - ```
      HTTP/1.1 200 OK
      Connection: close
      Date: Tue, 18 Aug 2015 15:44:04 GMT
      Server: Apache/2.2.3 (CentOS)
      Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT
      Content-Length: 6821
      Content-Type: text/html

      (data data data data data ...)
      ```
    - Common status codes include
        - 200 OK
        - 301 Moved Permanently
        - 400 Bad Request
        - 404 Not Found

        - 505 HTTP Version Not Supported

- **Cookies**
    - Since HTTP is stateless, it does not directly store any information about the user on the server side
    - However, for many sites, having the site remember prior user interaction is clearly beneficial, and therefore a system is in place to facilitate this
    - *Cookies* use a unique identification number for each user that is sent in an HTTP request messaged which is indexed in a backend server database that is used to store information corresponding to that same unique identifier



- **Web Caching**
    - A *web cache* or *proxy server* is an intermediate server where HTTP information is cached for future use
    - When an HTTP request goes out, it is first sent to the proxy server
        - If the proxy server has the desired information, it can return the information and bypass the origin server entirely, leading to more efficient navigation of the web
    - The proxy server will then forward the request to an origin server to service the user
    -

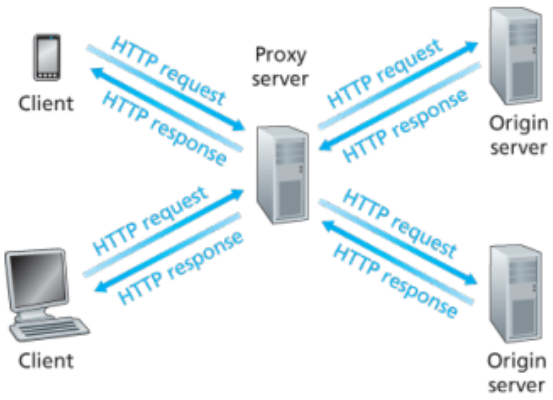


Figure 2.11 Clients requesting objects through a Web cache

- *The Conditional GET*
  - The conditional `GET` command allows a proxy server to check if a file has been updated on an origin server since the last cache of that file
  - It allows proxy servers to check if the information they have is the most up to date or not and respond to users with the most recent version of the file

2.3: Electronic Mail in the Internet

- The internet mail system has three main components
  - *User agents*
  - *Mail servers*
  - *Simple Mail Transfer Protocol (SMTP)*

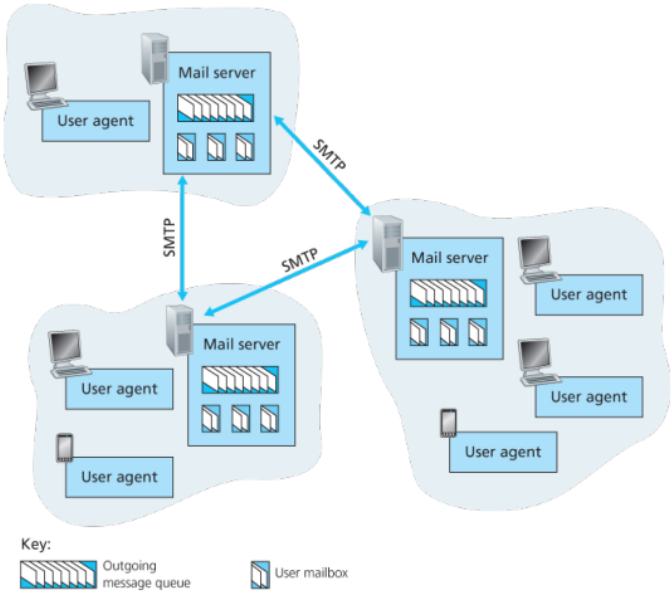


Figure 2.14 A high-level view of the Internet e-mail system

- **Comparison to HTTP**
  - HTTP is a *pull protocol*, whereas SMTP is a *push protocol*
  - Also, SMTP imposes the restriction of 7-bit ASCII encoding in the body and header of all messages, whereas HTTP does not

2.4: DNS - The Internet's Directory Service

- Hosts can be identified in different ways including a hostname, like *google.com* or an IP address like *121.7.106.83*
- For people, remembering phonetically and mnemonically easy names, like *google .com* is easier than remembering IP addresses, but for routers, the hierarchical structure of IP addresses is highly beneficial
- The *DNS* refers to two things
  - A distributed database implemented in a hierarchy of DNS servers spread across the world
  - An application-layer protocol that allows hosts to query this distributed database
- Usually, DNS servers are UNIX machines running the Berkeley Internet Name Domain (BIND) software
- DNS runs over UDP and uses port 53

- DNS is often used by other application-layer protocols, like HTTP and SMTP, to translate user-supplied hostnames to their corresponding IP addresses
- DNS provides aliasing for hosts and mail servers allowing for easier user access
- DNS also provides load distribution among multiple replicated servers which can help improve overall performance of the DNS
- **The Hierarchical Structure of DNS Servers**
-

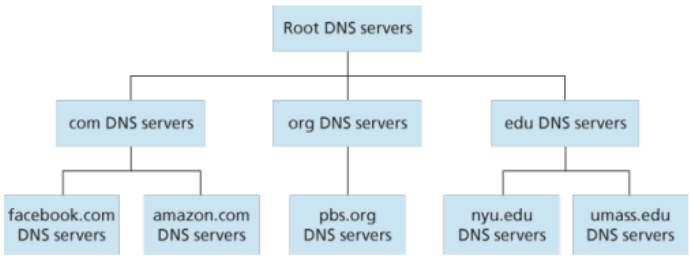


Figure 2.17 Portion of the hierarchy of DNS servers

  - **Root DNS servers** contain the IP addresses of the top-level domain (TLD) servers
    - There are about 400 of these servers worldwide
  - **Top-Level Domain Servers** contain the IP addresses of authoritative DNS Servers
    - Each top level domain (.com, .org, .net, .uk, .fr) has its own TLD server, or cluster of servers
  - **Authoritative DNS Servers** which contain the IP addresses of any publicly accessible files from an organization
    - Each organization has its own DNS server or pays a third-party to provide the service for them
- **Path That a DNS Request Takes**
  - A host sends a request to a Local DNS server, which sends it to a root DNS server which returns the IP address of the corresponding top-level domain server
  - The TLD server responds with the IP address of the authoritative DNS which sends back the file to the local DNS server
  - Finally, the local DNS server returns the file to the requesting host)

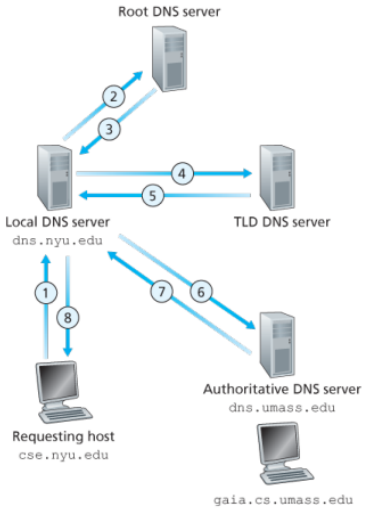


Figure 2.19 Interaction of the various DNS servers

- **DNS Caching**
  - The above method can be inefficient since every level of the DNS hierarchy must be contacted for every DNS request
  - To remedy this inefficiency, we can use *DNS Caching*
  - With DNS caching, any level of the DNS hierarchy can cache information and bypass contacting the higher levels of the hierarchy
  - This can be seen below

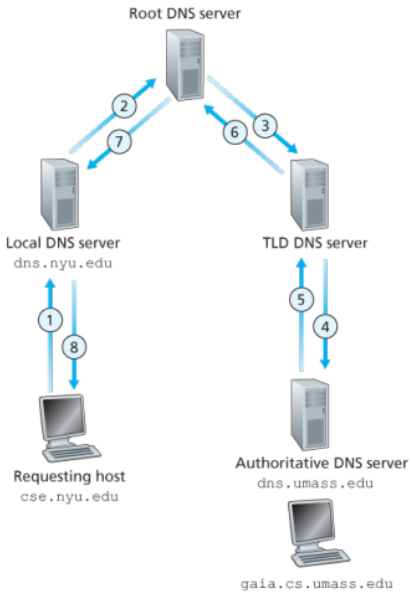
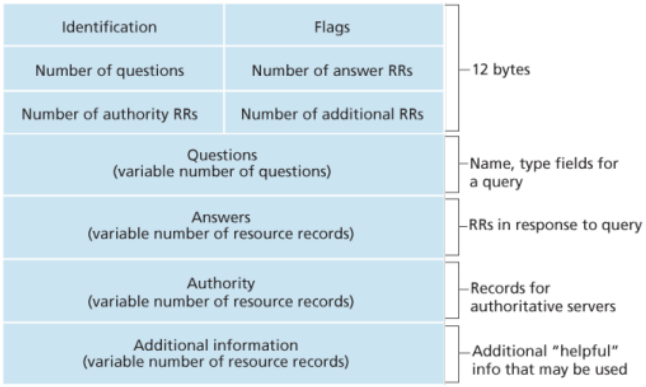


Figure 2.20 Recursive queries in DNS

- **DNS Message Format**



- **Entering Records into the DNS Database**
  - To enter data into the DNS database, an IP address and corresponding domain name must be registered
  - A *registrar* is a commercial entity that verifies the uniqueness of the domain name and enters it into the DNS database

Chapter 3: Transport Layer

3.1: Introduction and Transport-Layer Services

- A transport-layer protocol provides for *logical communication* between application processes running on different hosts
- These protocols are implemented in end systems, but do not need to be implemented in network routers

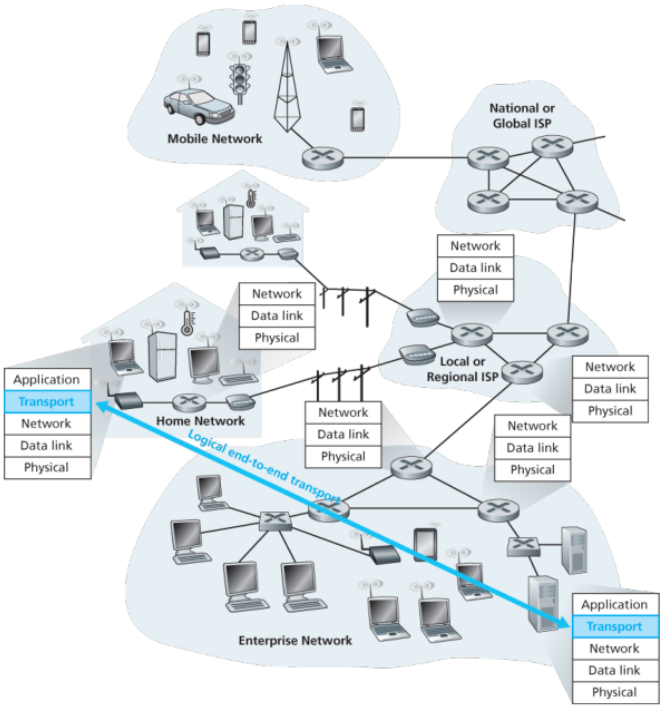


Figure 3.1 The transport layer provides logical rather than physical communication between application processes

3.2: Multiplexing and Demultiplexing

- **Multiplexing**
  - When a segment is sent from the application layer into the network layer, it is multiplexed such that the network layer can understand it
- **Demultiplexing**
  - Similarly, a segment coming from the transport layer must be demultiplexed before it enters the application-layer

Justin Ciocoi

Oct. 4, 2023

CSCI 379

Computer Networking

Transport Layer

Transport vs. Network Layer

- The network layer performs logical communication between hosts whereas the transport layer performs logical communication between processes which are reliant on network layer services
- **Internet Transport Layer Protocols**
  - TCP
    - Reliable
    - In order
    - Congestion controlled
    - Flow controlled
    - Connection handshake
  - UDP
    - No-frills
    - Extension of "best-effort" IP
    - UDP provides no delay or bandwidth guarantees

Multiplexing and Demultiplexing

- When data is sent from a sender, additional data is added on in the form of a header
  - This is referred to as *multiplexing*
- When the data is received by the receiver, the information present in the header will be used to *demultiplex* the data, or ensure it reaches the correct destination

User Datagram Protocol (UDP)

- UDP is often described as *bare-bones, no-frills, or best effort*, which indicates the lesser quality of the services it provides when compared to TCP

- When using UDP, segments could be
    - Lost
    - Delivered out of order to the recipient
- UDP is a **connectionless protocol**
    - There is no handshake between sender and receiver as there is with TCP
    - Each UDP segment is handled independently of all the other UDP segments
- *When* is UDP most commonly used?
    - Streaming multimedia applications, since they are loss tolerant (can afford to lose some segments) and rate sensitive (are heavily impacted by TCPs rate-limiting congestion control)
    - DNS
    - SNMP (Simple Network Management Protocol, which is a protocol used to monitor and manage network connected devices)
- Reliable transfer over UDP
    - Reliable transfer can be achieved over UDP by implementation at the application level
- UDP segment header
    - Source port number (16-bit)
    - Destination port number (16-bit)
    - Length of UDP segment, including header (16-bit)
    - Checksum (16-bit)
    - The UDP header has a fixed size of *64 bits* (8 bytes)

UDP Checksum

- Treat the segment contents as a string of 16-bit integers
- The checksum will be the ones complement sum of segment contents
- Sender puts the checksum in header

- Receiver calculates its own checksum and compares it to the sent checksum
    - If they match, no error is detected (there could still be errors)
    - If they do not, errors are detected
- Example for two 16-bit integers
    - $$1110011001100110 + 1101010101010101 = (1)1011101110111011 = 1011101110111100$$
    - The above is the sum, and the checksum is the ones complement of the sum, or 0100010001000011

Reliable Data Transfer (TCP)

- If the channel is error/loss free
    - Sender sends
    - Receiver receives
    - No feedback is needed
- If the channel is unreliable in the sender to receiver direction
    - Sender sends
    - Receiver sends acknowledgement (ACK)
- If the channel is unreliable in both directions
    - Sender sends
    - Receiver sends ACK
    - Both or one *can* be lost
    - If data or ACK is lost is lost, sender is stuck forever
        - Thus, you should use timers to make the sender re-transmit data
- Sequence numbers can also be used in order for the receiver to be able to differentiate duplicate packets
- Take a 1Gbps link, 15 ms prop. delay, and an 8000-bit packet:

- $D_{trans} = \frac{L}{R} = \frac{8000bits}{10^9 bits/sec} = 8$
- U_{sender}: utilization, or the fraction of time that the sender is busy sending
  - $U_{sender} = \frac{L/R}{RTT+L/R} = \frac{0.008}{30.008} = 0.00027 \approx 0.03\%$

Go-Back-N

- The sender can have up to N unacknowledged packets in the pipeline
- The receiver will only send a *cumulative acknowledgement* and won't acknowledge if there is a gap
- The sender has a **timer** for the oldest unacknowledged packer
  - When the timer expires, all unacknowledged packets will be retransmitted

Selective Repeat

- The sender can have up to N unacknowledged packets in the pipeline
- The receiver will send *individual acknowledgments* for each packet
- Similar timer system as the Go-Back-N protocol, but it operates on a packet-by-packet basis

TCP Overview

- Connection-oriented
- Reliable
- Point-to-point
- Pipelined
- Full duplex
  - Bidirectional simultaneous data transfer
- Flow Controlled
- TCP **segment structure**
  - Source and Destination Ports
    - 16-bits each

    - Sequence number
        - 32-bits
    - Acknowledgement number
        - 32-bits
    - Other fields like checksum, receive window (`rwnd`) and Urgent data pointer are used
    - Options field and application data field follow the above and are of variable length
    - **TCP Header** is **20 bytes** long
- The TCP specifications do not specify how to deal with handling out of order segments, and this is thus up to the implementor

TCP Timeouts

- How does TCP set the **timeout value**, or the amount of time that should pass before packet retransmission
    - It should be *longer* than the **RTT**
    - If its too *short* premature timeouts will occur and cause unnecessary retransmissions
    - If its too *long* the connection will react slowly to segment loss
- How does TCP estimate RTT?
    - A variable `SampleRTT` is the time between segment transmission and acknowledgement receipt
        - Retransmissions are ignored
    - The `SampleRTT` will vary, and a smoother Estimated RTT is desired
        - Therefore, TCP will average *several recent measurements* to Estimate the value of RTT
    - $EstimatedRTT = (1 - \alpha) * EstimatedRTT + \alpha * SampleRTT$
        - A typical value for α is 0.125
    - A safety margin is added as a buffer to the `EstimatedRTT`
        - $DevRTT = (1 - \beta) * DevRTT + \beta * |SampleRTT - EstimatedRTT|$

        - Typically, $\beta = 0.25$
    - So, $TimeoutInterval = EstimatedRTT + 4 * DevRTT$

TCP Sender Events

- When data is received from the app
    - Creates a segment with a sequence number
    - Starts the timer if it is not already running
- Upon timeout
    - Retransmit segment that caused timeout
    - Restart timer
- Upon reception of acknowledgement
    - Update what is known to be acknowledged
    - Start timer for any still unacknowledged segments
- Retransmission Scenarios
    - When the acknowledgement is lost
        - packet is retransmitted after timeout is reached
    - When a premature timeout occurs
        - Receiver receives duplicate packets and sends duplicate acknowledgements

Justin Ciocoi

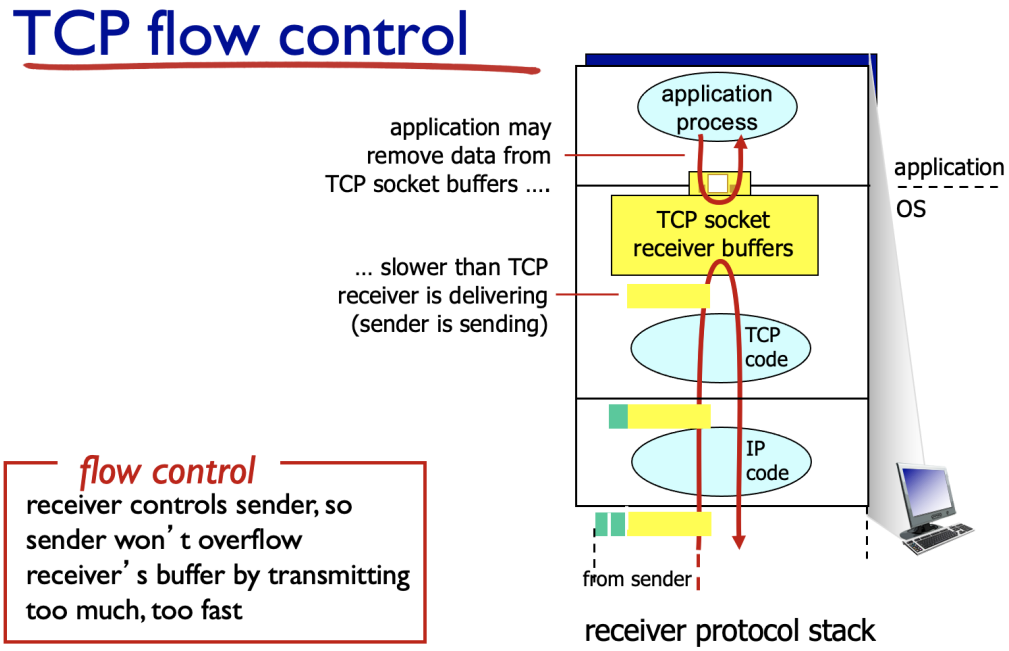
Oct. 10, 2023

CSCI 379 Class Notes

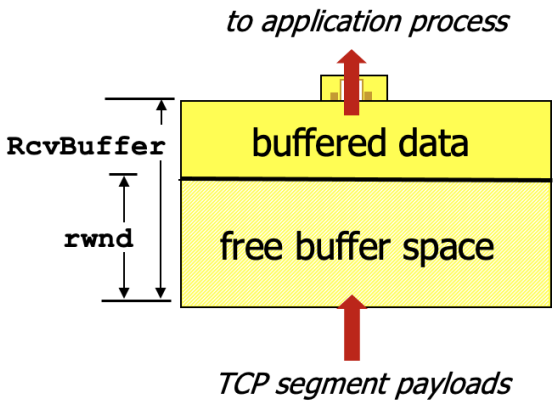
TCP Transmission

Flow Control

- *Cumulative Acknowledgements* are used to pipeline packets and lessen the overhead of TCP connections by allowing for fewer transmissions to be sent back and forth
- *Flow Control* is a way for a receiver to tell a sender that its buffer is close to filling up
    - This happens to prevent the buffer filling up and causing packet loss
- This is achieved using the *receive window* field in the TCP segment header



-

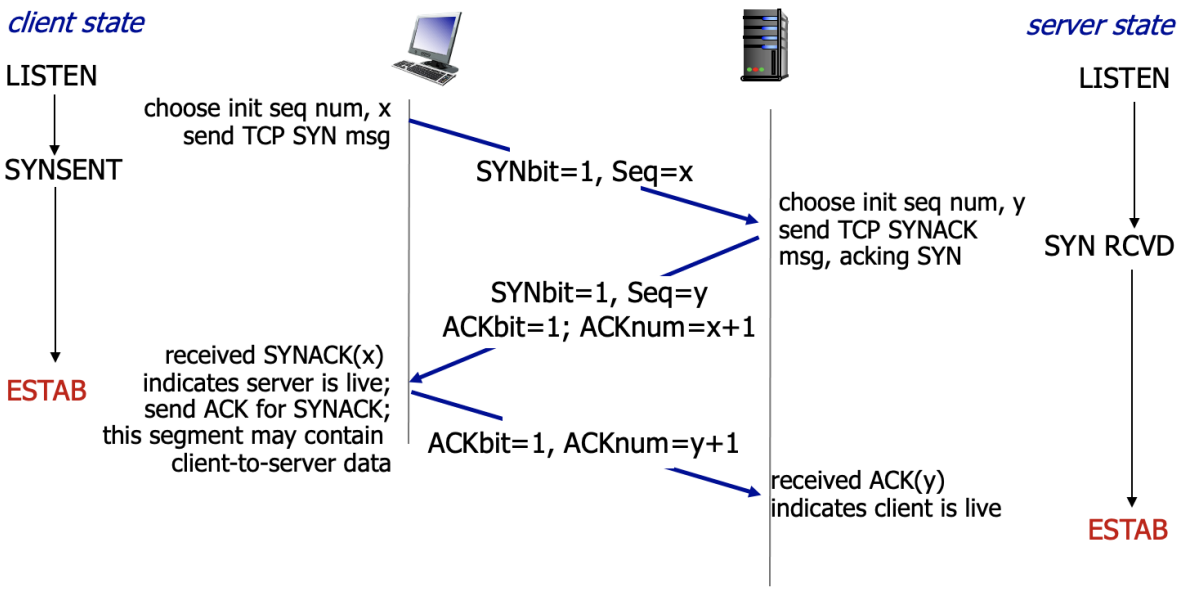


receiver-side buffering

-
- The receiver informs sender of free space using the `rwnd` value in the TCP header
  - `RcvBuffer` size is set via socket options (typical default is 4096 bytes)
  - Many OSes auto-adjust `RcvBuffer`
- Sender limits size of sent data to the value oof the receivers `rwnd` variable
- Prevents buffer overflow

Connection Management

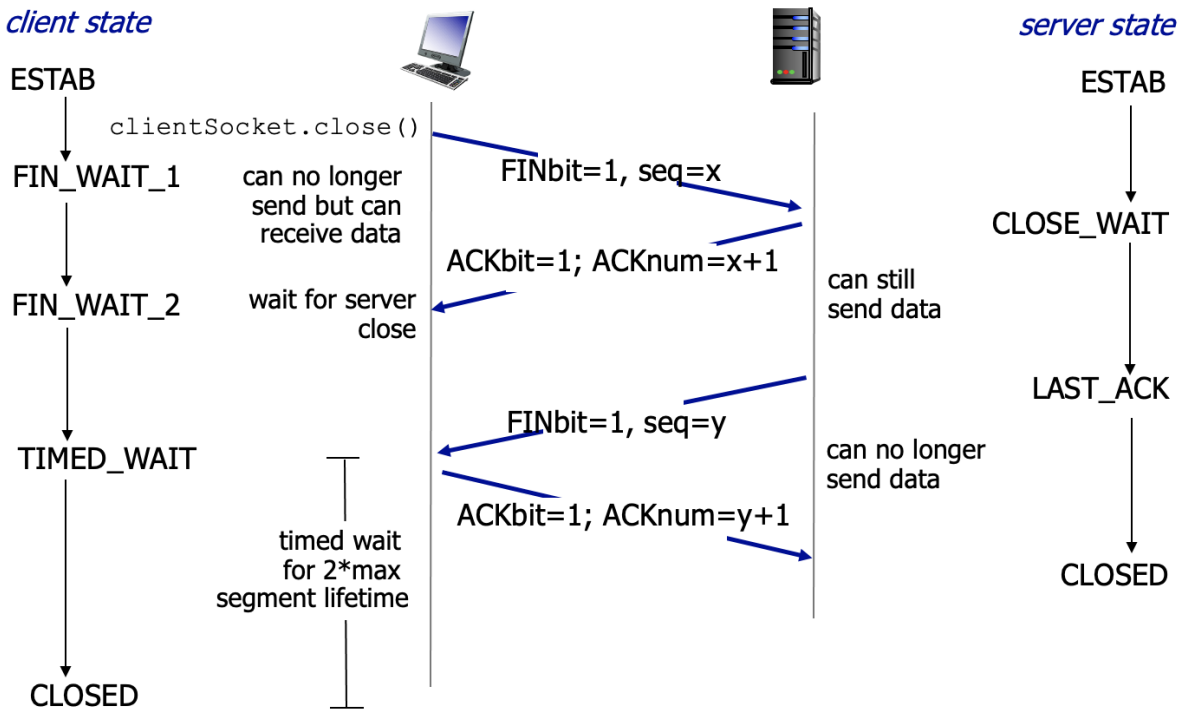
- **The TCP Three-Way Handshake**



  -

- **Closing a connection**
  - A TCP connection is closed using the `FIN` flag in the TCP header and setting it to 1

- When receiver receives a FIN flag, it will acknowledge it, and then after some time it will send its own `FIN` flag
  - During this time, the receiver is still capable of transmitting data over the connection



Congestion Control

- In a connection, lost packets and long delays are often the result of *congestion* in the network
- This is a very important issue in the realm of networking, given that most networks are susceptible to congestion at times

Justin Ciocoi

Oct 4, 2023

CSCI 379 Textbook Notes

Computer Networking

Textbook Notes

Section 3.7: TCP Congestion Control

Overview of TCP Congestion Control

- TCP Provides a reliable transport service between two processes which are running on different hosts
    - A congestion-control mechanism is also of paramount importance to the functioning of TCP
        - The approach that TCP takes in order to control congestion is to limit the rate of sent traffic as a function of perceived network congestion
        - This brings up important questions about TCP
            - *How* does TCP limit the rate of traffic that is being sent out
            - *How* does TCP *perceive* the level of congestion present on a network
            - *Which* algorithm should be used to reduce traffic
    - The TCP congestion-control mechanism operating at the *sender* keeps track of an additional variable, `cwnd`, or the congestion window, which imposes a constraint on the rate at which the TCP sender can send traffic into the network
        - This is how rate limiting is achieved under the TCP protocol
    - Next, we must consider how the TCP protocol is able to perceive the congestion on the bath between sender and recipient
        - Congestion can be perceived, roughly speaking, through the responses associated with dropped datagrams and *loss events*
    - TCP is said to be **self-clocking**

        - This means that when the protocol does not receive duplicate ACKs
        - The rate of arrival of these responses will indicate to TCP at which rate to increase its congestion window
            - The faster the rate at which ACKs are received, the higher the perceived bandwidth under TCP and thus the higher the rate limit goes

Guiding Principles of TCP Congestion Control

- A **lost segment** implies **congestion**
    - Thus, the TCP sender's rate should be **decreased** when a segment is lost
- An **acknowledged segment** indicates successful network delivery
    - When the sender receives an ACK for a previously unacknowledged segment, its rate will be **increased**
- The two previous principles imply TCP's mechanism of **bandwidth probing**
    - Roughly speaking, bandwidth probing can be explained as TCP's increasing of the data rate until a loss event occurs
    - This can be thought of as TCP continuously checking for the fastest achievable speed by slowly increasing the rate at which data is transmitted
- Slow Start
    - When a TCP connection is established, the value of `cwnd` is initialized as a small value of 1 MSS
        - If MSS = 500 bytes and RTT=200 msec
        - $InitialSendingRate = \frac{MSS}{RTT} \approx 20kbps$
- TCP's congestion control mechanism can be referred to as an **additive-increase, multiplicative-decrease (AIMD)** form of congestion control
    - The `cwnd` variable is increased linearly by TCP until the point at which a loss event occurs, where it will be decreased by a function of the rate at the previous loss event

TCP Fairness

- TCP can be thought of as a *fair* protocol

- In a situation with multiple connections to the same recipient, each connection is granted the same amount of bandwidth
- For this reason, many multimedia applications choose to run over **UDP** as losing the occasional packet during video or audio streaming is not as hurtful to the user experience as throttling speeds to achieve a *fair* connection environment

Explicit Congestion Notification (ECN): Network-Assisted Congestion Control

- Through more recent developments and implementations of TCP, the protocol can now make use of **ECNs** in order to allow the network layer to explicitly signal network congestion to both a sender and a receiver
- In theory, this provides a connection that can more optimally utilize the available bandwidth of the network

Justin Ciocoi

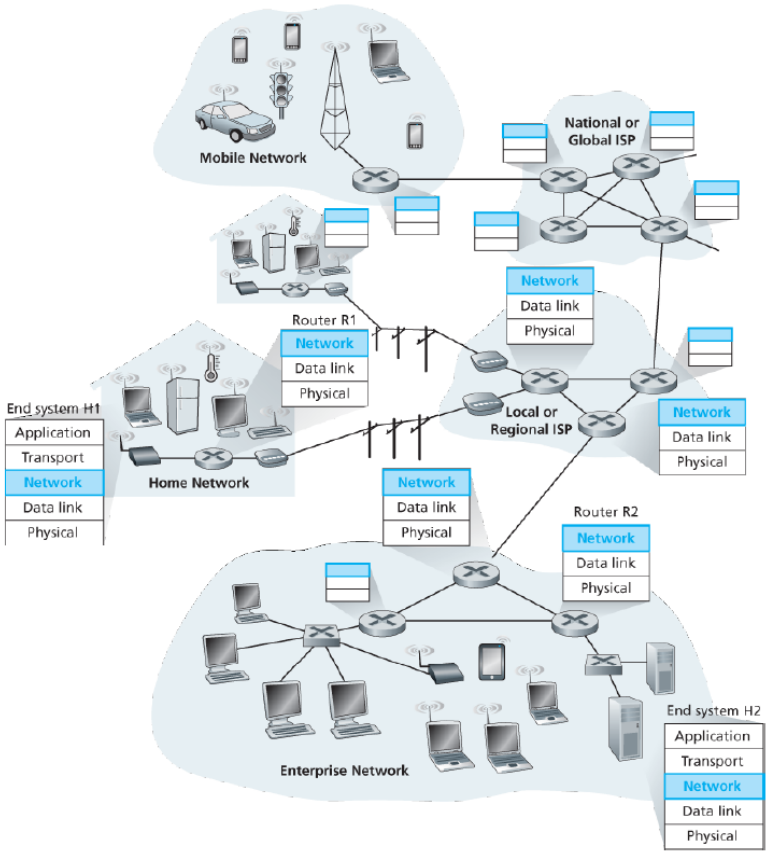
Oct. 6, 2023

CSCI 379 Textbook Notes

Computer Networking

The Network Layer

4.1: Overview of the Network Layer



- Based on the above diagram, let us assume host **H1** is sending data to host **H2**
- The following will be the steps taken by the network to transmit that data
    - The network layer in **H1** takes segments from the transport layer in **H1** and encapsulates them into datagrams
    - **H1** then sends these datagrams to its nearby router, **R1**

    - On the receiving end, host **H2**'s network layer receives the datagrams from *its* nearby router, **R2**
    - **H2** then extracts the transport layer segments from the datagrams and delivers the segments to the transport layer in **H2**
- **4.1.1: Forwarding and Routing: The Data and Control Planes**
    - The network's layer primary role is to move packets from a sending host to a receiving host
    - Two important network layer functions in the facilitation of this movement can be identified
        - **Forwarding**
            - When a packet arrives at a router's *input* link, the router must move that packet to the appropriate *output* link
            - A packet might also be *blocked* if the origin is known to be malicious or if the destination is forbidden
            - Alternatively, data can also be *duplicated* and sent over multiple outgoing links
        - **Routing**
            - The network layer is responsible for determining the route that data will take from sender to receiver
            - *Routing algorithms* determine the path that the packet in the earlier example would take from host **H1** to host **H2**
        - Often, routing and *forwarding* are used interchangeably in writings about the network layer, but more specifically, forwarding refers to a *local* process and routing refers to a *network-wide* process

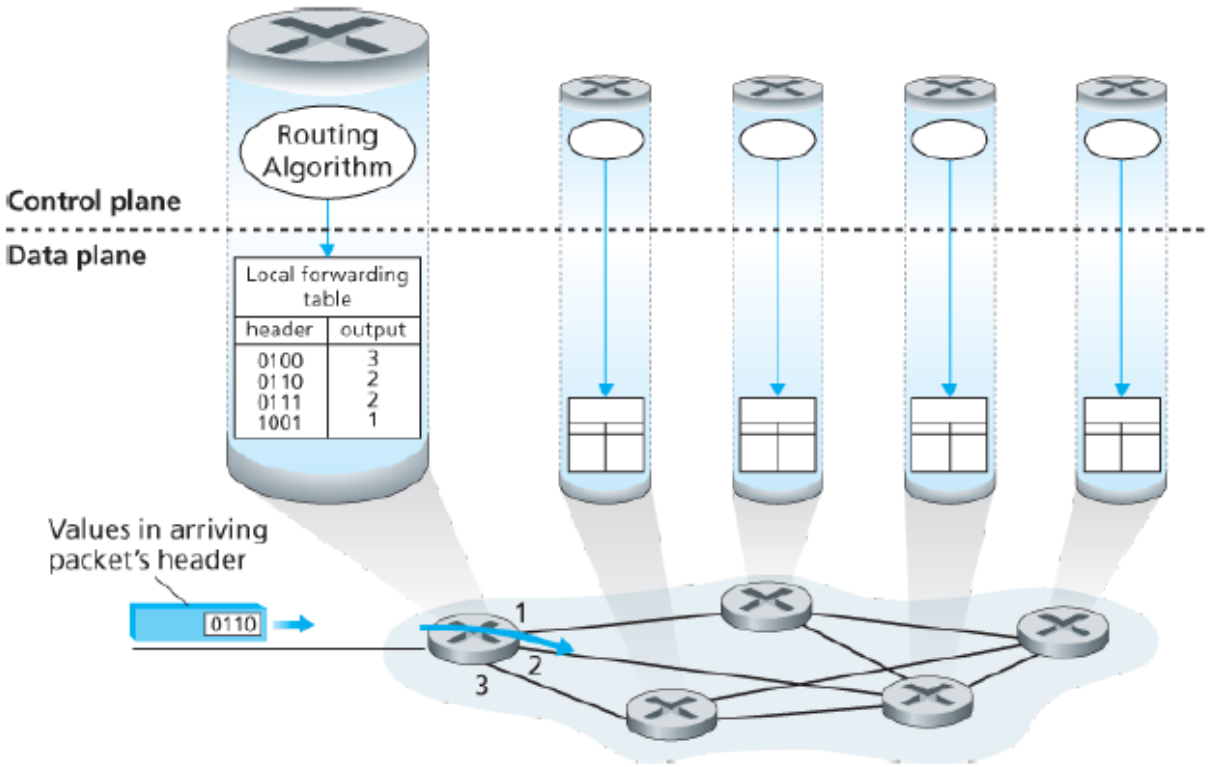


Figure 4.2 Routing algorithms determine values in forward tables

-
  - The above depicts a *forwarding table*
  - A router will look at certain values arriving in the header of a packet, and based on those values will forward it to the appropriate outgoing link
  - In this diagram, a packet containing the value `0110` arrives at the router, and the forwarding table tells the router to send that packet to output link 2
  - The composition of forwarding tables is done on a network-wide scale by using network routing algorithms to allow each and every router to compute adequate values for their forwarding tables
- **4.1.2 Network Service Model**
  - So, having discussed the basics of network layer functionality, we can now consider the different services that the network layer is capable of providing
    - *Guaranteed Delivery*, meaning that when a packet is sent, it is guaranteed to eventually arrive at its destination
    - Guaranteed Delivery with *Bounded Delay*, meaning guaranteed delivery within a specified window of time

- *In-Order Packet Delivery*, meaning the packets will arrive at their destination in the same order in which they were sent
- *Guaranteed Minimal Bandwidth*
- *Security*, since the network layer can encrypt datagrams at the source and decrypt them at the destination, which provides confidentiality to transport layer segments

- It is important to note that this is just a *partial* list of potential network layer functionality, and in reality there are many more variations

4.2: What's Inside a Router?

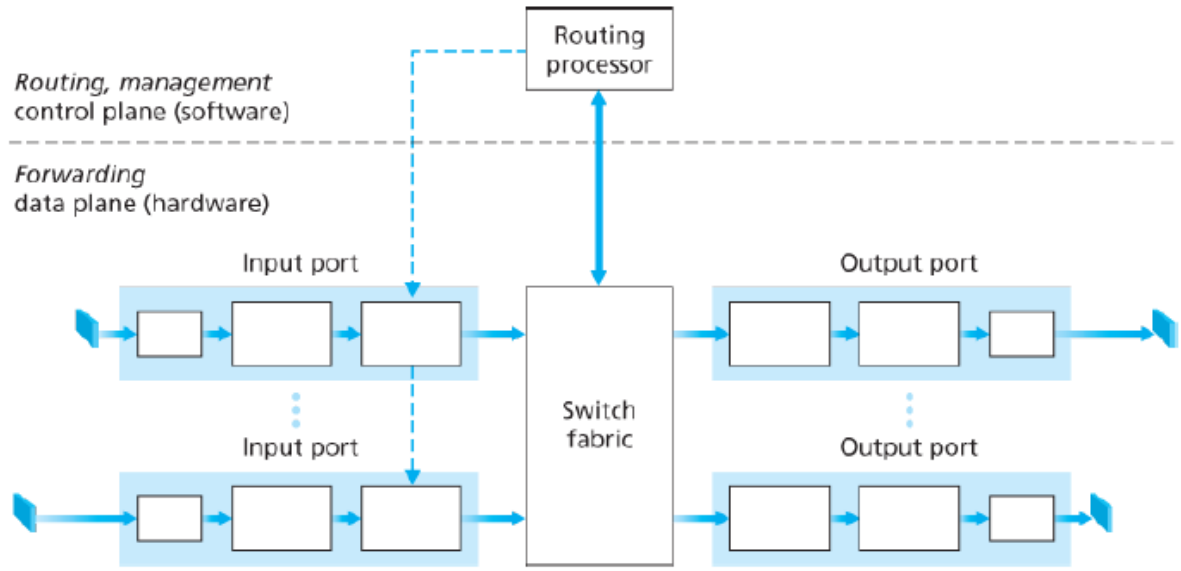


Figure 4.4 Router architecture

-

- From this high-level diagram of the inner workings of a router four main components can be identified

  - ***Input Ports***, which perform several key functions, including

    - Performing the *physical layer* function of terminating an incoming physical link at a router

    - They also perform *link layer* functions needed to interoperate with the link layer on the other side of the incoming link

    - Most importantly, a *lookup function* is also performed in which the forwarding table is consulted to determine the appropriate output link to which to send a packet through the switch fabric

        - ***Switching Fabric***, which is a fabric that connects a router's input ports to its output ports
        - ***Output Ports***
            - It performs functions very similar to the input port but in an outgoing direction
            - When a link is *bidirectional*, an output port will usually be paired with the input port for that link on the same line card
        - ***Routing Processor***, which performs *control plane* functions such as
            - Executing routing protocols
            - Maintaining routing tables
            - Computing a router's forwarding table
    - These components are almost *always* implemented in hardware
    - The data plane operates at the nanosecond time scale, whereas the control plane operates at the millisecond or second time scale, and is thus usually implemented in software
    - Let us consider two types of information that could be used for forwarding of information over a network
        - *Destination-Based Forwarding*, where a router forwards information based on the most efficient possible route to its specified destination
        - *Generalized Forwarding*, where a router uses a set of predetermined rules to direct traffic regardless of a packet's specified destination

- **4.2.1: Input Port Processing and Destination-Based Forwarding**

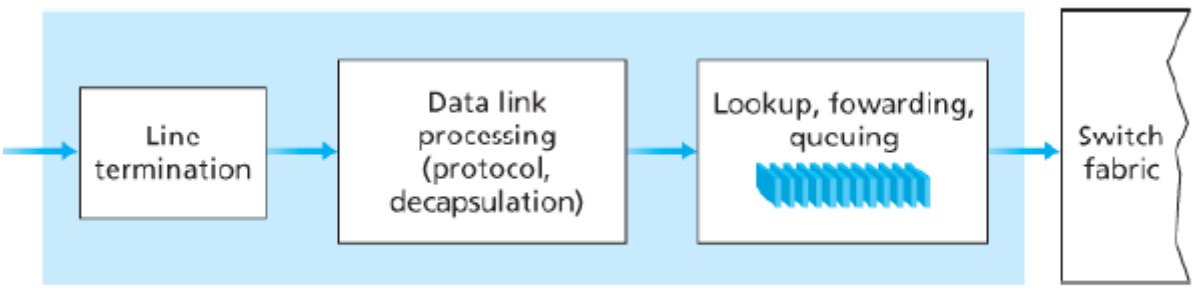
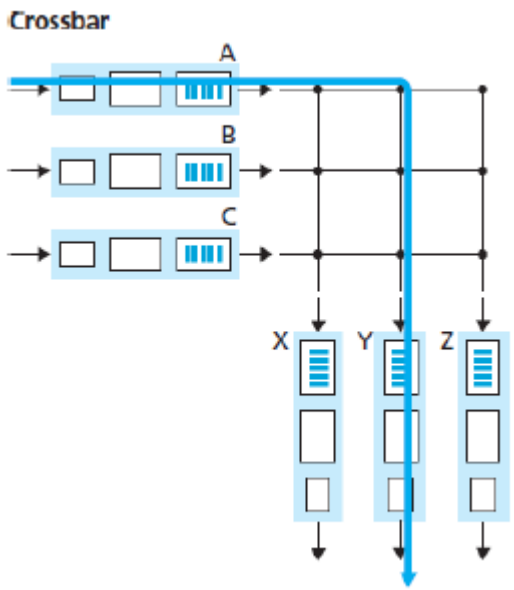


Figure 4.5 Input port processing

-

    - Here we can see a more detailed view of the input port processing described earlier and shown in Figure 4.4
    - The forwarding table for an input link will either be computed by the routing processor or is received from a remote *SDN controller*
    - The forwarding table is copied from the routing processor to each input line card such that forwarding decisions can be made locally at each input port without the need to consult the routing processor
    - Using a forwarding table, the operation of lookup seems fairly simple, but at gigabit transmission rates, these operations must be performed in nanoseconds
        - To achieve this, not only is lookup performed in hardware, but fast lookup algorithms are used
    - Once the lookup has determined the appropriate output port for a packet, the packet can then be sent to the switching fabric
    - Packets can be temporarily blocked from entering a busy switching fabric and queued to enter at a later time
    - Apart from lookup, input processing is also made up of
        - Physical and link layer processing
        - Packet version number, checksum, and time-to-live bust be checked and the latter two rewritten
        - Counters used in management of networks must be updated
- **4.2.2: Switching**
    - The switching fabric is at the heart of the router, and is the thing that actually forwards packets
    - There are a number of different ways in which switching can be accomplished
        - *Switching via memory*
            - This operates similar to a traditional computer system
            - The routing processor is tasked with switching and input and output ports function as traditional I/O devices in a traditional OS

        - Headers are copied into memory such that the CPU (routing processor) is able to facilitate switching
        - Because only one memory read/write can be done at a time over the shared system bus, two packets cannot be switched simultaneously under a memory switching model
        - In some modern routers, this processing is done on individual line cards which thus resemble shared-memory multiprocessing systems
    - *Switching via a bus*
        - In this approach, the routing processor is not involved as the input port pre-pends a switch-internal header to the packet indicating the appropriate output port
        - All output ports will receive this packet, but only the one matching the packet's header will keep it
        - Because all packets must travel over a shared system bus, switching speed is limited to the bus speed
    - *Switching via an interconnection network*
        - Using a more sophisticated system such as this is a way to overcome the limitations presented by the previous two models



        -

- **4.2.3: Output Port Processing**

- Output port processing takes packets that have been stored in the output port's memory and transmits them over the output link, which includes
  - *Selecting* and *de-queueing* packets for transmission
  - Performing the necessary physical and link layer operations needed in transmission
- The below illustrates a basic view of output port processing

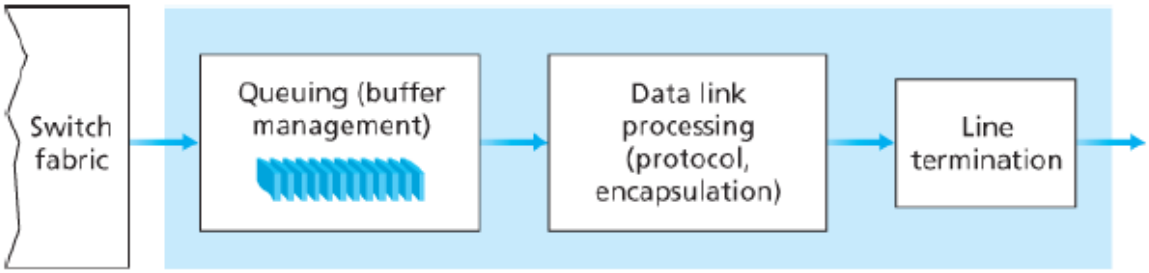


Figure 4.7 Output port processing

-

- **4.2.4: Where Does Queueing Occur?**
  - Considering the models discussed in input and output processing, it is clear that packet queues could form in either side of a router's links
  - When packet queues become large enough, they can eventually exhaust the memory of a router and cause *packet loss*
  - Assume R_{line} is the identical transmission rate of N input links and N output links, and R_{switch} is the switching fabric transfer rate or the rate at which packets can be moved from input to output port
  - If $R_{switch} \geq N * R_{line}$, then only negligible queueing will occur
  - However, if $R_{switch} < N * R_{line}$, then packet queueing will occur at the *input ports*
    - In order to illustrate the consequences of this type of queueing, we must assume that all link speeds are identical, that input and output speeds are the same, and that packets are serviced by the switching fabric in a first-come first-served manner
    - In this model, multiple packets can be transferred in parallel, as long as their desired output link is different
    - However, packets destined for the same output link must wait

- In an input queued switch, however, it is possible for a packet to block the progression of another at the head of the switching fabric despite there being no competition for the blocked packet's desired output port
- This phenomenon, called *head-of-the-line (HOL) blocking* and is illustrated in the below diagram

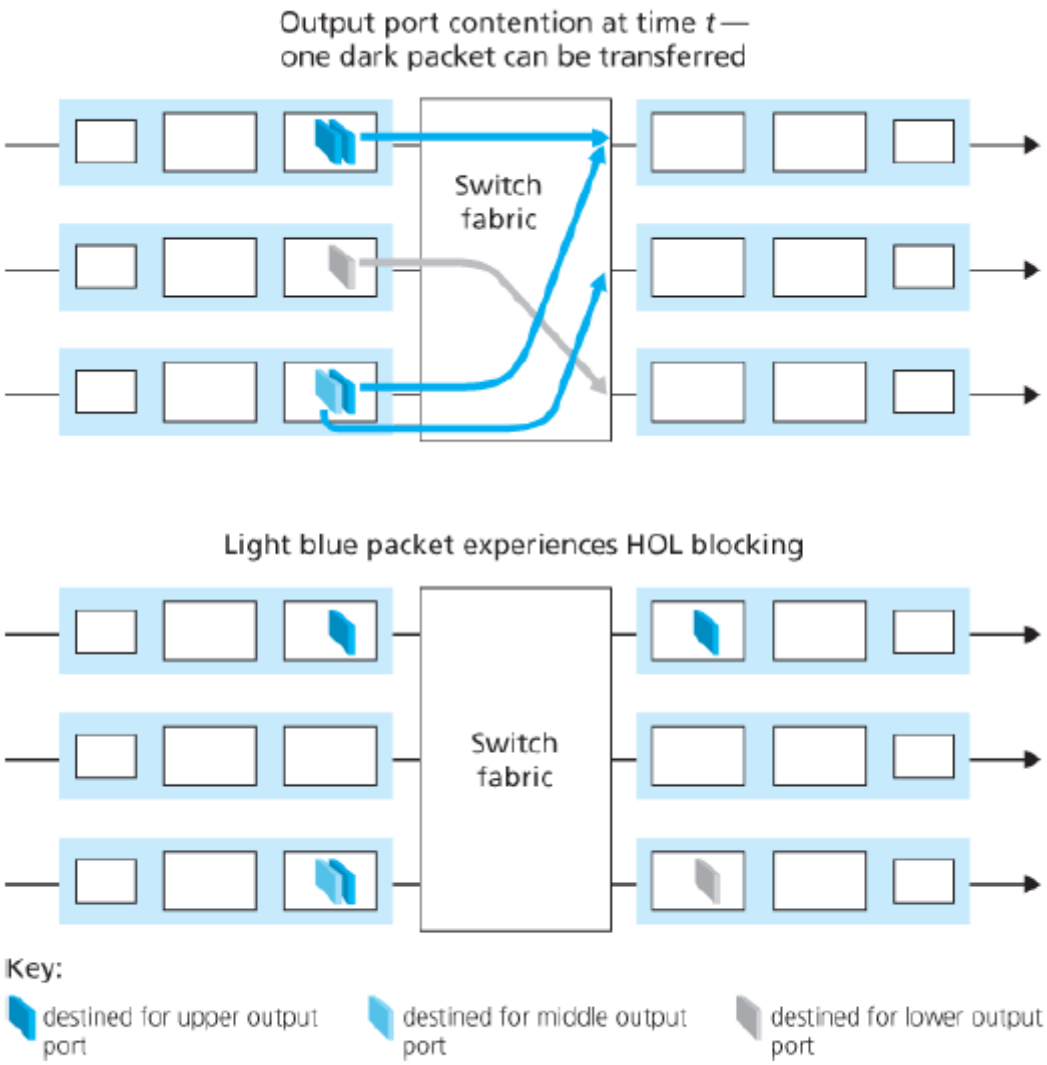


Figure 4.8 HOL blocking at and input-queued switch

-

- Even if $R_{switch} \geq N * R_{line}$, since multiple packets can arrive at an output port in the time it takes for one packet to be transmitted, queuing can clearly occur at output links as well
  - If there is not sufficient memory to buffer an incoming packet, a decision must be made to either
    - *drop* the arriving packet

        - *remove* one or more already-queued packets to make room for the new packet
    - Sometimes packets are dropped before a buffer is totally full to indicate congestion on a network
    - There are a number of proactive packet dropping and marking policies known as *active queue management (AQM) algorithms*
    - According to recent theoretical and experimental efforts, the amount of buffering needed in an output port is $B = RTT * \frac{C}{N}$ where
        - C is the link capacity
        - N is the number of TCP flows passing through a link
        - RTT is the average round trip time

- **4.2.5: Packet Scheduling**
    - When discussing packet scheduling, as with CPU scheduling, there are a number of different ways in which it might be implemented, and these various ways have their own benefits and detriments
    - *First-in-First-out (FIFO)*

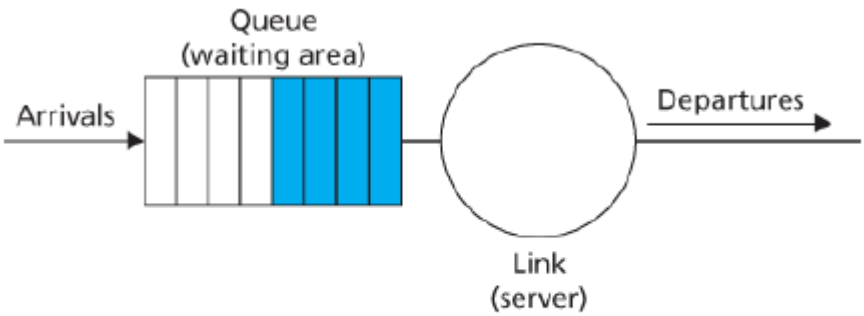


Figure 4.10 FIFO queueing abstraction

        -
        - In this structure, packets are serviced in the order in which they arrive at the router's output link

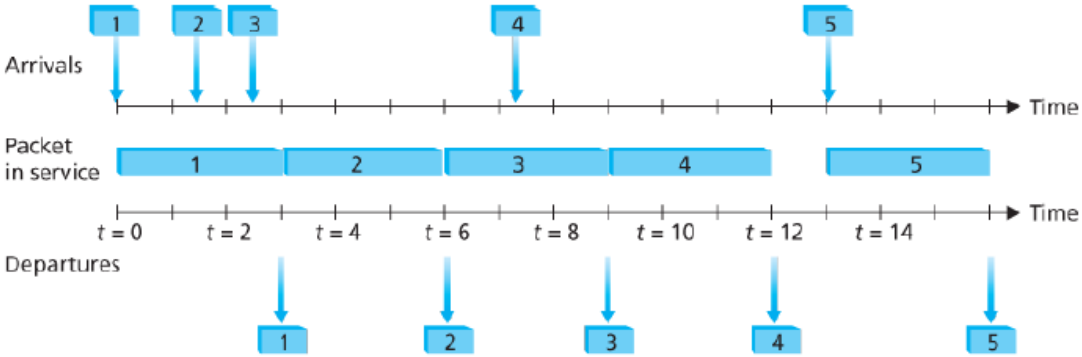


Figure 4.11 The FIFO queue in operation

- *Priority Queue*
  - In a priority queuing structure, packets are assigned a priority class upon arriving at the output link
  - Typically, each priority class has its own queue
  - There is *preemptive* and *nonpreemptive* priority queuing where preemptive queuing allows a high priority packet to interrupt the transmission of a lower priority packet
  - The below are basic representations of priority queueing

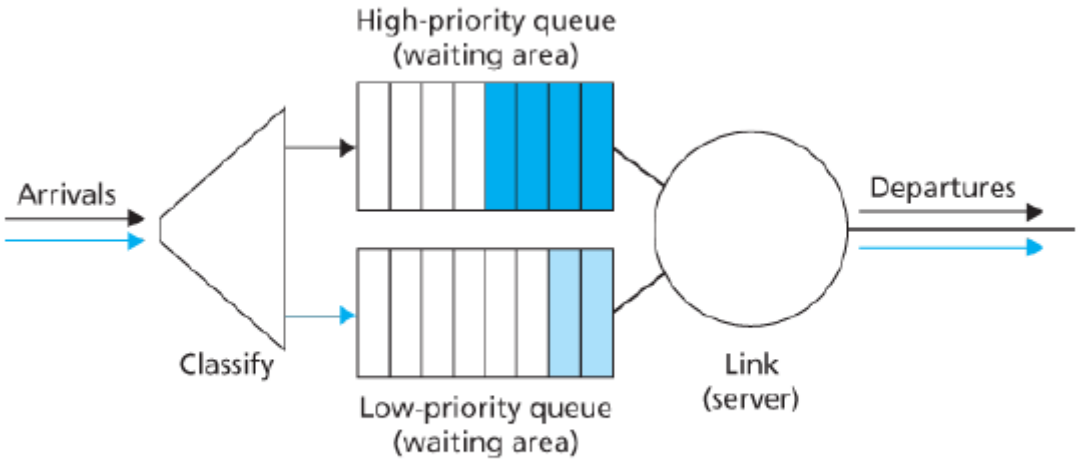


Figure 4.12 The priority queueing model

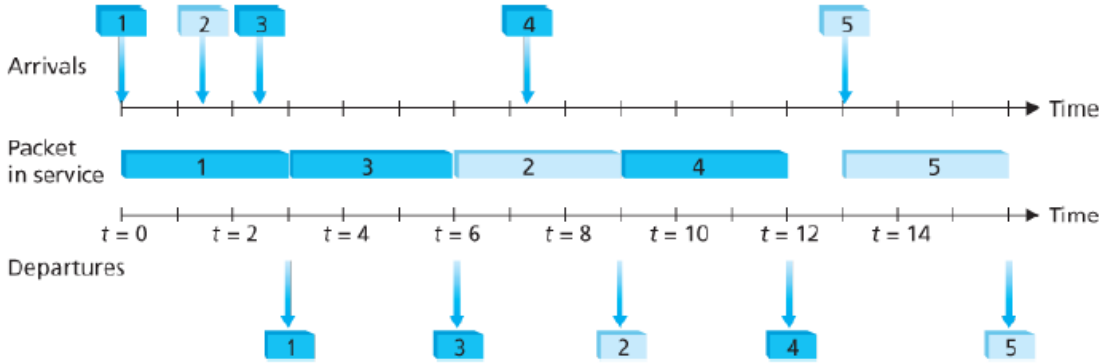


Figure 4.13 The priority queue in operation

- *Round Robin and Weighted Fair Queueing (WFQ)*
  - Similar to priority queueing, packets are separated into classes as they arrive at on output link
  - However, rather than there being a strict service priority, the scheduler will *alternate* service between the classes
  - A *work-conserving queueing* discipline will never allow the link to remain idle whenever there are packets of any class that are queued for transmission
    - If it finds no queued packets in the current class, it will move to another and begin to check for queued packets
  - Below is an example of *two-class* round robin queueing

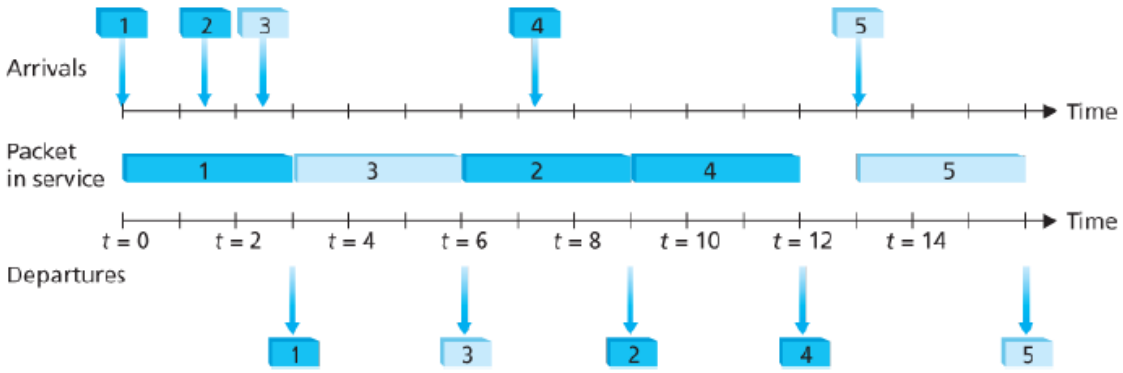


Figure 4.14 The two-class robin queue in operation

  - A generalized form of round robin queueing is the weighted fair queueing discipline
  - In weighted fair queueing, the different classes will be cycled between, but the time intervals during the which a class, i, will be serviced is determined by a

weight, w_i

- A class, i, will then be guaranteed to receive $\frac{w_i}{\sum w_j}$ fraction of service where $\sum w_j$ is the sum taken over all classes that have packets queued for transmission

- Therefore, for a link with transmission rate, R, under WFQ, a class i will always achieve a throughput of at least $R * \frac{w_i}{\sum w_j}$
- Below is a representation of weighted fair queuing

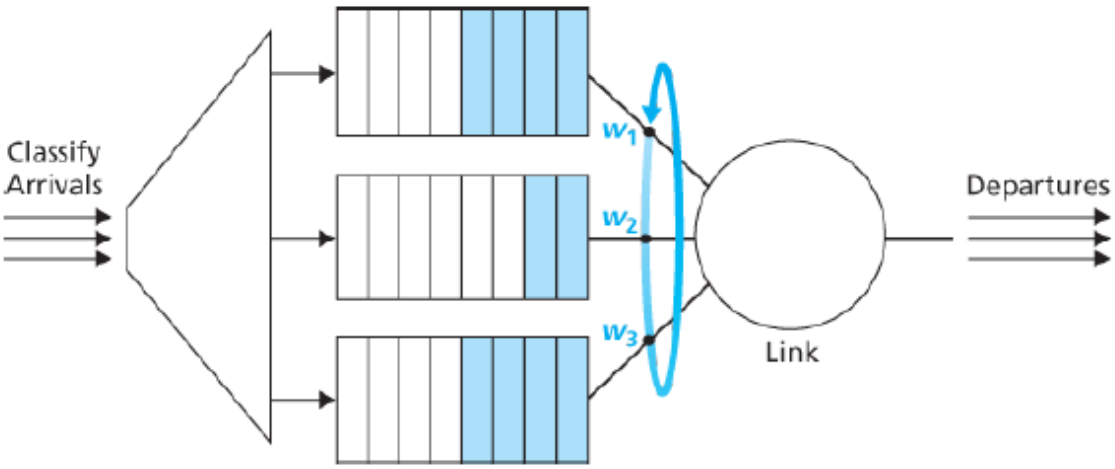


Figure 4.15 Weighted fair queueing

-

Justin Ciocoi

Oct. 10, 2023

CSCI 379 Textbook Notes

Computer Networking

4.3: The Internet Protocol (IP): IPv4, Addressing, IPv6, and More

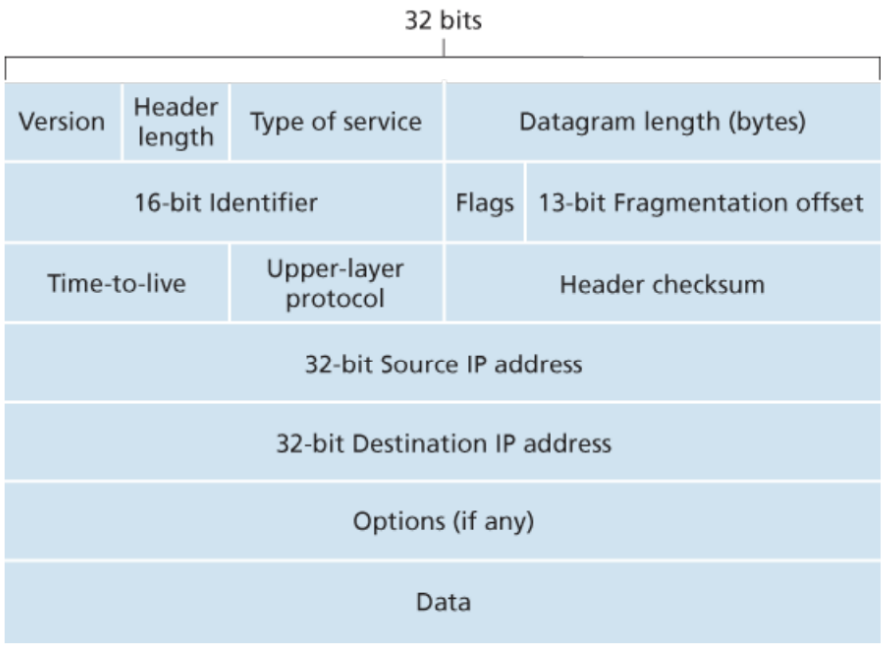
- **4.3.1: IPv4 Datagram Format**
  - 

Figure 4.16 IPv4 datagram format

  - As shown in the above, the key fields in an IPv4 datagram are
    - **Version number** (4-bit) which specifies the IP protocol version and tells the router how to interpret the remainder of the datagram
    - **Header Length** (4-bit) tells the router how long the header is and therefore where the payload actually begins
    - **Type of Service (TOS)** allows different types of IP datagrams to be distinguished from each other
    - **Datagram Length** (16-bit)

        - **Identifier, flags, fragmentation offset**
            - These only exist in IPv4 and not in IPv6
        - **Time-to-live** which is a field used to ensure that datagrams do not continue to circulate forever
        - **Protocol** specifies the transport-layer protocol to which the data from the datagram should be passed
        - **Header checksum** which helps in detecting bit errors during transmission
        - **Source and Destination IP Address** which specify the source and destination IPs of the datagram
        - **Options** which allow the IP header o be extended and used rarely to save the overhead used in the average IP datagram
        - **Payload** which is the actual data contained within the datagram
    - Some protocols can carry big datagrams whereas others only smaller ones
    - The maximum amount of data that a link-layer frame can carry is called the *maximum transmission unit (MTU)*
- **4.3.2: IPv4 Datagram Fragmentation**
    - If a router receives a datagram that has a size larger than its corresponding output link's MTU, it must fragment the payload into two or more smaller datagrams
        - Each smaller datagram is referred to as a *fragment*
    - When fragmented datagrams arrive, they must be reassembled before they reach the transport layer at the destination
    - This is where the identifier, flags, and fragmentation fields of the IPv4 datagram are used
    -

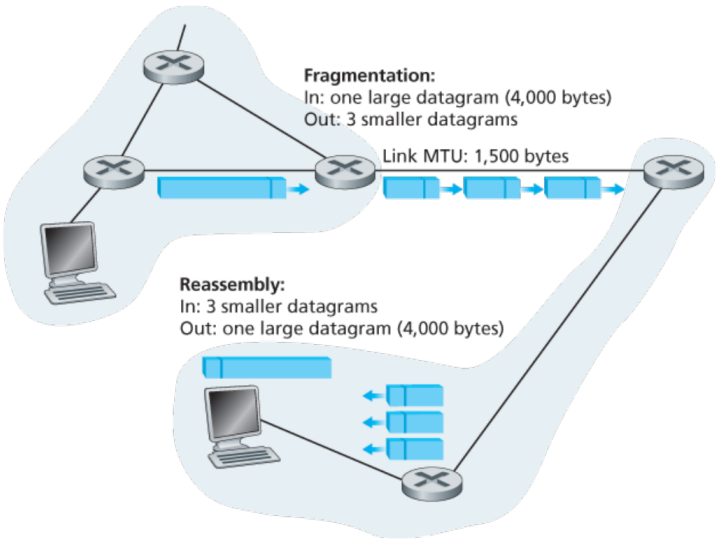


Figure 4.17 IP fragmentation and reassembly

- **4.3.3: IPv4 Addressing**
  - The boundary between a host and its physical link to the network is called an *interface*
  - IP addresses, rather than correlating to hosts directly, correlate more directly to the interfaces within those hosts
  - In IPv4, IP addresses are *32 bits, or 4 bytes*, long
  - A network which connects host interfaces to a router interfaces is known as a *subnet (also called IP network, or just network)*
  - A *subnet mask* is a prefix of bits that is the same across all devices connected to the same subnet
    - The length of a subnet mask. can vary
  - *To determine the subnets, detach each interface from its host or router, creating islands of isolated networks. Each network here is a* ***subnet***
  - The Internet's address assignment strategy is known as *classless inter-domain routing (CIDR)*, pronounced "cider"
  - CIDR generalizes 32 bit IP addresses in the form $a.b.c.d/x$, where x indicates the number of bits in the first part of the address
  - x is generally referred t as the *network prefix*
  -

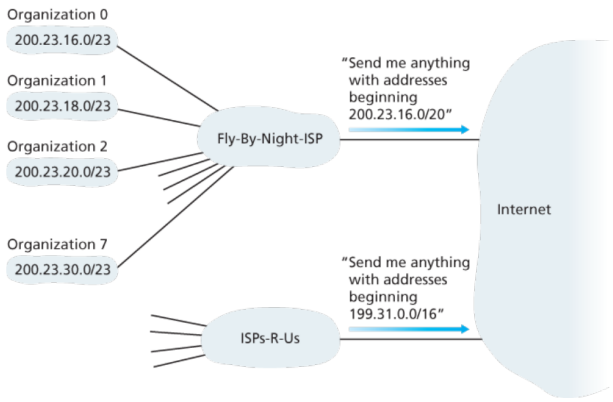


Figure 4.21 Hierarchical addressing and route aggregation

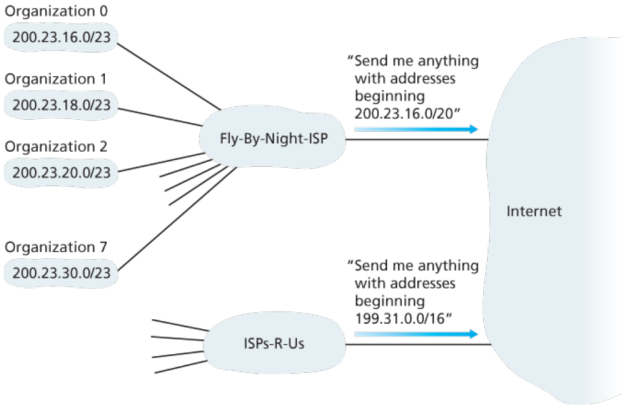


Figure 4.21 Hierarchical addressing and route aggregation

  - The above illustrates what would happen why a hierarchical structure for IPv4 addressing as it allows more dynamic reallocation of IP subnets

- **Dynamic Host Configuration Protocol**
  - After a block of IP addresses is obtained by an organization from an ISP, the host addresses must also be configured
  - While this *can* be done manually, it is most often done through the use of *DHCP*
  - DHCP can be configured to give a temporary IP address each time a network connection is initiated or to give the same IP address to each device
  - DHCP automated much of the process of connecting a host system to a network, and is thus often referred to as a *plug-and-play* or *zero-conf (zero configuration)* protocol
  - **DHCP Server Discovery**
    - When a new device connects to the network, it must first find a DHCP server using a *DHCP discover message*, which a client sends within a UDP packet to port 67
    - The message is sent with source IP 0.0.0.0 and destination IP 255.255.255.255 and is broadcast to all nodes on the subnet

- **DHCP Server Offers**
  - A DHCP server will respond to the client with a *DHCP offer message* that is also broadcast to all nodes on the subnet
  - Since their may be multiple DHCP servers on the network, a client may be able to choose between multiple offers
  - The DHCP server will assign the IP address, the network mask, and an *IP address lease time*, the amount of time for which an IP address will remain valid
- **DHCP Request**
  - After choosing an offer, the client will send a *DHCP request message*
- **DHCP ACK**
  - After receiving a DHCP request message, the DHP server will respond with a *DHCP ACK message* which confirms the parameters requested by the client

- **4.3.4: Network Address Translation**
  - Given everything above, we know that every IP-capable device needs an IP address
  - However let's consider some issues with the systems described earlier in this section
    - Given an already allocated block of addresses, what happens when an organization requires more addresses than are allocated within the block
      - This is especially concerning as more and more internet connected devices are in use in almost every workplace
    - Especially if two contiguous address blocks are already assigned, then an organization might be forced to have non-sequential blocks of IP addresses - something that would complicate network management
  - The solution to this problem is *network address translation (NAT)*
  -

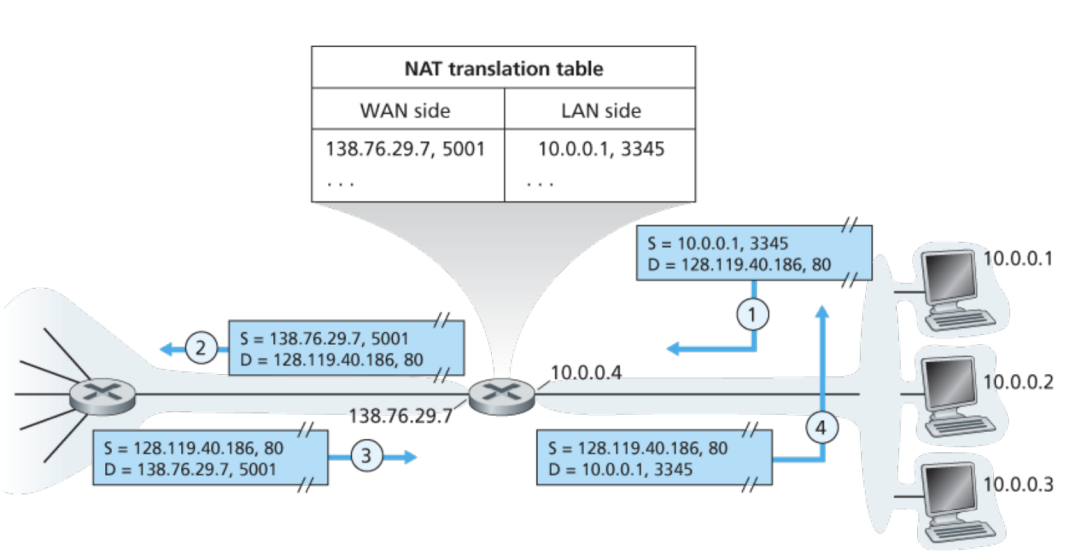


Figure 4.25 Network address translation

- The NAT-enabled router clearly functions as a router, but to the outside world, it *looks* like it is a *single* device with a *single* IP address
- In the above figure, all traffic leaving the home router has a *source address* of 138.76.29.7 and all traffic entering the home router has a *destination address* 138.76.29.7
- If all arriving traffic has an identical destination IP address, how does the router know which internal host should be receiving the data in question
    - A *NAT translation table* will be used at the NAT router and port numbers will be included alongside IPs in this table
- A NAT translation table indexes source ports assigned to outgoing traffic to the port numbers used on the WAN, and the table is then used to translate the public IP and port to host IP
- *Focus on Security*
    - Knowing a range of IP addresses belonging to an organization could enable an attacker to conduct various types of malicious packet attacks
    - Two popular defense mechanisms to these types of attacks are
        - *Firewalls*
        - *Intrusion Detection Systems (IDSs)*
    - For example, a firewall could be in place to prevent all ICMP echo request packets from coming through, thereby preventing port scanning or network mapping

attacks

    - An IDS uses an already known list of packet signatures that are part of attacks, and if a match between this database and entering packets is found, an alert is created
    - An *Intrusion Prevention System (IPS)* is similar but actually blocks the traffic rather than just creating an alert

- **4.3.5: IPv6**
    - In the early 1990s, researchers started to realize that the 32-bit IPv4 address space was beginning to run out of available addresses with new nodes and subnets being added at an exponentially growing rate
    - To respond to this, a new IP protocol, IPv6, was created
    - In addition to increasing the size of the address space, designers of IPv6 took the chance to tweak and augment various other aspects of the IPv4 protocol based on the operational experience they now had with it
    - In February 2011, IANA allocated out the last available IPv4 addresses within their pool
- **IPv6 Datagram Format**
    -

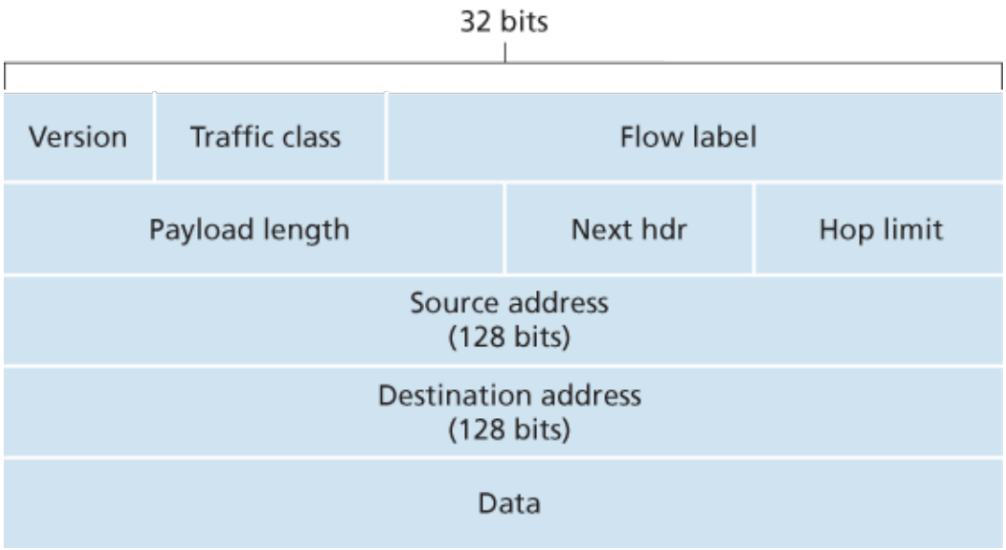


Figure 4.26 IPv6 datagram format

    - The key differences between IPv4 and IPv6 are:
        - **Expanded Addressing Capabilities**

        - IPv6 increases the size of the IP address from 32 bits to 128 bits (with IPv6 every grain of sand on earth could be assigned an IP address)
        - IPv6 introduces *any-cast addresses* which allow a datagram to be delivered to any one of a group of specified hosts
            - This could be useful, for example, for downloading a document from the closest, and therefore fastest, server hosting that document
    - **Streamlined 40-byte Header**
        - A number of IPv4 fields have been dropped or made optional, which results in faster processing of IP datagrams by a router
    - **Flow Labeling**
        - IPv6 can differentiate between "types" of *flow*
        - For example, a real time service would be treated as a *flow* since a continuous transfer of information would bee necessary to maintain the experience
- As shown in the above diagram, the key fields of the IPv6 datagram are
    - **Version**, which is a 4-bit field identifying the IP version number
    - **Traffic Class**
        - Can be used to prioritize certain datagrams *within* a flow
        - Can be used to prioritize datagrams from certain applications over other applications
    - **Flow Label**, a 20-bit field used to identify a flow of datagrams
    - **Payload Length**, which is a 16-bit value representing the length of the IPv6 datagram not including the 40-byte header
    - **Next Header**, which identifies the protocol to which the data will be delivered (i.e. UDP or TCP)
    - **Hop Limit**, The upper limit on the number of times a datagram can be transmitted by a router; a datagram is discarded when the limit is surpassed
    - **Source and Destination IP Addresses**

        - **Data**
    - As we can see here, there are several fields present in the IPv4 datagram that are missing here
        - **Fragmentation/Reassembly**
            - Under IPv6, fragmentation and reassembly can be done *only* at the source or the destination, and not by intermediate routers
            - If a datagram is too large, a router will simply drop it and send back a "Packet Too Big" ICMP error message to the sender, after which the sender should respond using a smaller datagram size
            - By removing fragmentation and reassembly functionality from the routers, considerably speeds up IP forwarding within the network
        - **Header Checksum**
            - Since protocols like TCP, UDP, and Ethernet already perform checksumming, the designers of IPv6 probably felt it was largely redundant to include it again in the IP protocol
        - **Options**
            - While not present in the header, the *options* field still can be one of the possible headers pointed to from within the original IPv6 header
            - Since there is no options field, we can have a fixed length IPv6 header
- **Transitioning from IPv4 to IPv6**
    - How will the widespread public internet, which is based on IPv4, be transitioned to IPv6?
    - One approach would be to declare a flag day - a given time and date where all internet machines would be turned off and upgraded from IPv4 to IPv6
        - Given the size of the modern internet, such an approach is unthinkable
    - The approach that has been most widely adopted in practice is one that involves *tunneling*

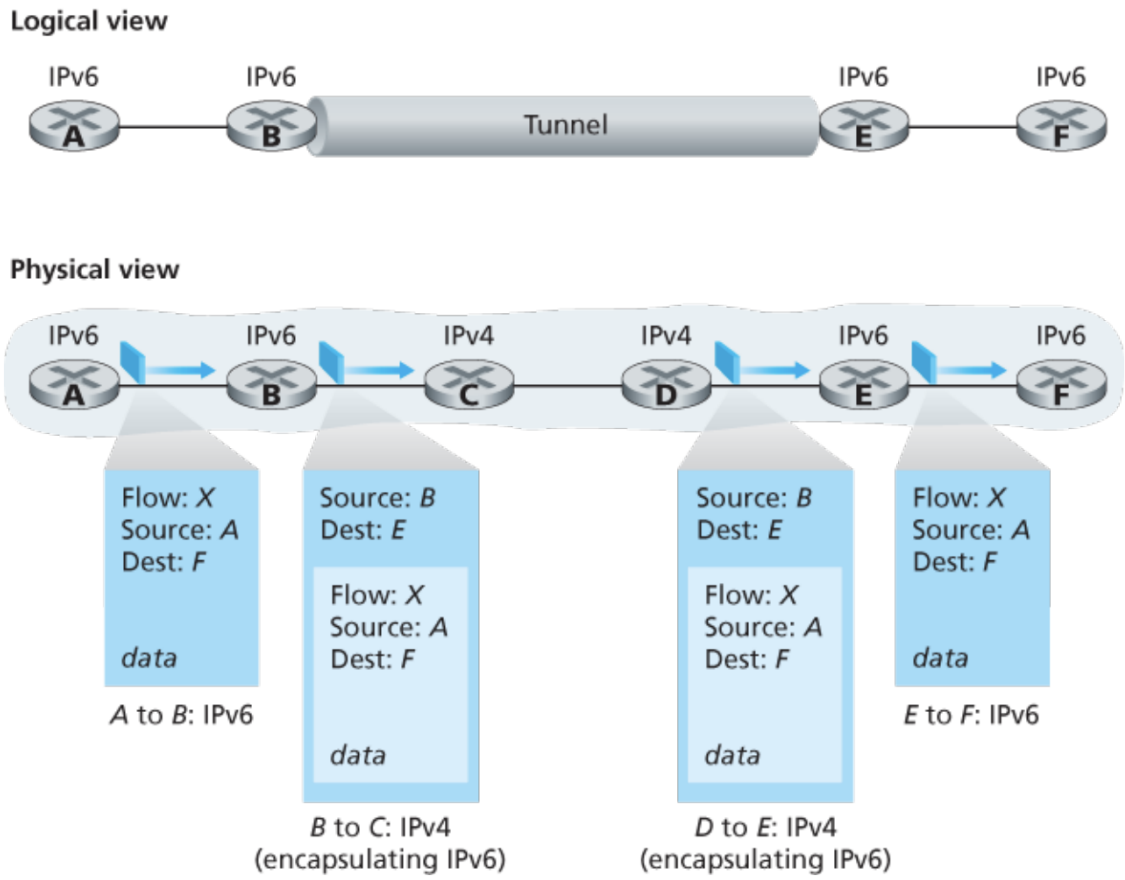


Figure 4.27 Tunneling

- Assume for example two IPv6 hosts are connected by a series of IPv4 routers
- If host A wants to send an IPv6 datagram to host B, it can encapsulate the *entirety* of the IPv6 datagram, including header, in an IPv4 datagram and send it that way
- This way, the IPv4 routers can act as a *tunnel* for IPv6 data while not understanding it, as only the receiving host must be able to decipher the data contained inside

- One important lesson to take from this is the difficulty that comes with changing protocols at any level, and the desire to design and create protocols that are more scalable and adaptable to deal with the future