Computer Networks

Spring 2021

Jean-Claude Graf

September 20, 2021

Contents

1 Introduction

1.1 Components

- Are composed of three basic components
  - ◇ **End-Systems:**
    - ∗ Send and receive data
    - ∗ Represent the users of a networks
    - ∗ Huge diversity of such devices
  - ◇ **Switches/Routers:** Forward data to the destination
  - ◇ **Links:** Interconnect switches/routers and end-systems
- **Last Mile:** Link which connects user to their ISP
  - ◇ There are different types of last mile
  - ◇ **Digital Subscriber Line (DSL)**
    - ∗ Used telephone lines
      - \+ Cheap because lines were already there
    - ∗ Composed of:
      - ○ Upstream data
      - ○ Downstream data
      - ○ 2-way phone channel
    - ∗ Asynchronous
  - ◇ **Cable Access Technology (CATV)**
    - ∗ Uses TV cable
    - ∗ Coaxial copper cable for house connection
    - ∗ Fiber cable for household junction to ISP
    - ∗ Composed of:
      - ○ Downstream data
      - ○ Upstream data
    - ∗ Asynchronous
    - ∗ Many households share the same connection
      - \- Vulnerable to hacks
  - ◇ **Enterprise Access**
    - ∗ Relatively directly connected to ISP
    - ∗ Local switches and aggregated switches
  - ◇ **Ethernet**
    - ∗ Most widely used local area network technology
    - ∗ Twisted pair copper
    - ∗ Symmetric (same speed in both direction)
  - ◇ **Undersea Cable**
    - ∗ Different hardening depending on the environment
  - ◇ **Other**
    - ∗ Many other technologies
    - ∗ Satellites, balloons etc.
- **Internet:** Network of networks
  - ◇ Enormously complex
  - ◇ We consider a abstraction
  - ◇ A network is graph with nodes and edged:
    - ∗ **Node:**
      - ○ **Capacity:** How much data can they transfer at a given time?
      - ○ **Degree:** How are they connected to other nodes?

        * **Edges:**
            - **Directed:** Can data travel in both directions?
            - **Capacity:** How much data can they transfer at a given time?
            - **Latency:** How much time does it take to transfer data?

1.2 Resources Sharing

- Links are shared among users
- **Oversubscribe:** Assumed that not all user will use it at the same time
- Terminology
    - ◇ **P:** Peak rate
    - ◇ **A:** Average rate
    - ◇ **A/P:** Level of utilisation in %
- Two main principles
- **Circuit Switching/Reservation**
    - ◇ Reserve the bandwidth P for all switches on the path in advance
    - ◇ Only use $A/P\%$
    - ◇ Good when P/A small
    - ◇ Multiplexed at flow-level
    - ◇ Two different methods
    - ◇ **Time-base:** Periodic timeslots are reserved
    - ◇ **Frequency-base:** Specific frequencies are reserved
    - ◇ Both methods can be combined
    - \+ Predictable performance
    - \- Does not routes around failed switches
        - * Setup of new path required
    - \- Long setup and take down of the link
        - * Not suited for short communication
    - ◇ Used in phone networks
- **Packet Switching/On-Demand**
    - ◇ Send data when needed
    - ◇ Good when P/A large
    - ◇ Multiplexed at packet-level
    - \+ Efficient use of resource
    - \+ Routes around failed switches
    - \- Unpredictable performance
    - \- Packets can clash
        - * I.e. arrive at a switch/router at the same time
        - * Requires buffers
    - \- Network can “overflow” (congestion)
    - ◇ Used by the internet

1.3 Network Layer

- Network is structured in 5 layer
- Each layer has its protocols
- Each layer builds on a service provided by the lower layer
- Each layer provides a service to the upper layer
- OSI model has 7 layers
- **Physical Layer:**

    - ◇ Provides physical transfer of bits
    - ◇ Moves *bits*
- **Link Layer:**
    - ◇ Provides local best-effort delivery
    - ◇ Moves *frames*
- **Network Layer:**
    - ◇ Provides global best-effort delivery
    - ◇ Moves *packet*
- **Transport Layer:**
    - ◇ Provides end-to-end delivery (reliable or not)
    - ◇ Transports *segments*
- **Application Layer:**
    - ◇ Provides network access
    - ◇ Exchanges *messages*
- Each layer adds its own header to the message before transmitting it to the lower level
- Different devices implement different layers
    - ◇ Host implements all layers
    - ◇ Router implements up to L3
    - ◇ Switch implements up to L2

1.4 End-To-End Principle

- Network is unreliable, but try to make it as reliable as possible
    - ◇ Else we have too many end-to-end retries
- End-to-end check **always** necessary
- **Fate-Sharing Principle:** Acceptable to loose state but only if the entity is lost too
    - ◇ Store state always co-local to entities

1.5 Characteristics and Performance

- Network is characterized by delay, loss and throughput
- **Delay**
    - ◇ Total delay is sum of:
        - ∗ **Transmission Delay**
            - ○ Time to push all bits onto the link
            - ○ Transmission delay [sec] $= \frac{\text{packet size [\# bits]}}{\text{line bandwith [\# bits/sec]}}$
            - ○ Due to link properties
        - ∗ **Propagation Delay**
            - ○ Time for a bit to travel through the link
            - ○ Propagation delay [sec] $= \frac{\text{link length [m]}}{\text{propagation speed [m/sec]}}$
            - ○ Propagation speed is a fraction of c (speed of light)
            - ○ Due to link properties
        - ∗ **Processing Delay**
            - ○ Time to process a request
            - ○ Due to traffic and switch internals
            - ○ Tiny
                - ▷ Can be neglected
        - ∗ **Queuing Delay**
            - ○ Time a packet waits in the buffer for transmission
            - ○ Varies from packet to packet

        - ▷ Hard to predict
        - ▷ Analysis requires statistical measures
      - ○ Depends on:
        - ▷ Arrival rate of new pack
        - ▷ Transmission rate of outgoing link
        - ▷ Traffic burstiness
      - ○ **Analysis**
        - ▷ **a [packets/sec]:** average packet arrival rate
        - ▷ **R [bit/sec]:** transmission rate on outgoing link
        - ▷ **L [bit]:** fixed packet length
        - ▷ **La [bit/sec]:** average bits arrival rate
        - ▷ **La/R:** traffic intensity
        - ▷ Queuing delay is exponential with increasing traffic intensity
        - ▷ Diverges at 1
        - ▷ Increases fast if traffic is bursty
        - ▷ Goal: operate far from critical point
      - ○ Due to traffic and switch internals
    - ◇ Cannot predict which delay dominates
- **Loss**
  - ◇ Queue can hold at most N packets in their buffer
    - ∗ $N + 1$ packet gets dropped
  - ◇ Upper bound $= N \cdot L/R$
  - ◇ Loss may be acceptable or not
    - ∗ Can resend redundant information as $A, B, A \oplus B$
- **Throughput**
  - ◇ Rate at which a host receives data
  - ◇ Average throughput [# bits/sec] $= \frac{\text{data size [\# bits]}}{\text{transfer time [sec]}}$
  - ◇ End-to-end throughput is equal to the throughput of the bottleneck
    - ∗ I.e. the min of the throughputs of all links

2 Application Layer

- Top-most layer
- Used by user-applications as well as core internet applications

2.1 Domain Name Service (DNS)

- Provides core functionality for the internet
- Human identify hosts using hostnames, while internet uses IPs
- DNS provides hostnames to IP mapping
- It is not one-to-one mapping:
  - ◇ **Load Balancing:** One hostname maps to multiple IP
  - ◇ **Reuse hardware:** Multiple hostnames map to same IP
    - ∗ E.g. host multiple websites on same machine
- **History**
  - ◇ One list of all mappings
  - ◇ Manually shared and updated
  - ◇ Placed in `/etc/hosts`
  - \- Not scalable
  - \- Hard to manage
  - \- Inconsistent
  - \- List no always available for download

+ Scalability
+ Availability
  - ◇ Domains are independent of each other
+ Extensible
  - ◇ Can extend any part without interference of other parts

2.1.1 Structure

- DNS uses three intertwined hierarchies
  - ◇ **Naming Structure**
    - ∗ **Root:**
      - ∘ Imaginary dot after the TLD in the URL
    - ∗ **Top Level Domain (TLD):** Sit at the top
    - ∗ **Domain:** Subtree of the TLD
    - ∗ **Subdomain:** Subtree of domain
      - ∘ Can be arbitrarily nested
    - ∗ A name is a leaf-to-root path
  - ◇ **Management**
    - ∗ Root is managed by the IANA
      - ∘ Association of 13 companies
    - ∗ TLDs are managed by private or federal organisations
    - ∗ Domains are managed by private organisations
      - ∘ I.e. Hosting companies and ISPs
    - \+ Name collision is trivially avoided
  - ◇ **Infrastructure**
    - ∗ 13 root servers
      - ∘ Named a through m
      - ∘ Each has numerous mirrors

        - Distributed all around the world
        - Two k root server are located in CH
      - TLD are managed professionally
      - Domains are managed by ISPs or locally
- **BGP Anycast**
  - Allows sharing of same IP for multiple servers
  - Determines "fastest" path
    - Not necessarily the shortest
  - Used by the roots servers for load balancing
  - Allows DNS system to scale
  - Done for e.g. root server
- For the system to work, we need
  - Each DNS server knows the address of the root server
  - Each root servers knows the address of all TLD servers
  - Each DNS server knows the address of all its children
- **DNS Server**
  - Each domain must have at least two DNS server
    - Ensure availability
    - Load-balancing
  - Stores *Resource Records* as tuples `(name, value, type, TTL)`
    - **Name:** The variable which we want to map
    - **Value:** Desired value
    - **Type:** One of:

Record Type	Name	Value
A	hostname	IP address
NS	domain	DNS server name
MX	domain	Mail server name
CNAME	alias	canonical name
PTR	IP address	corresponding hostname

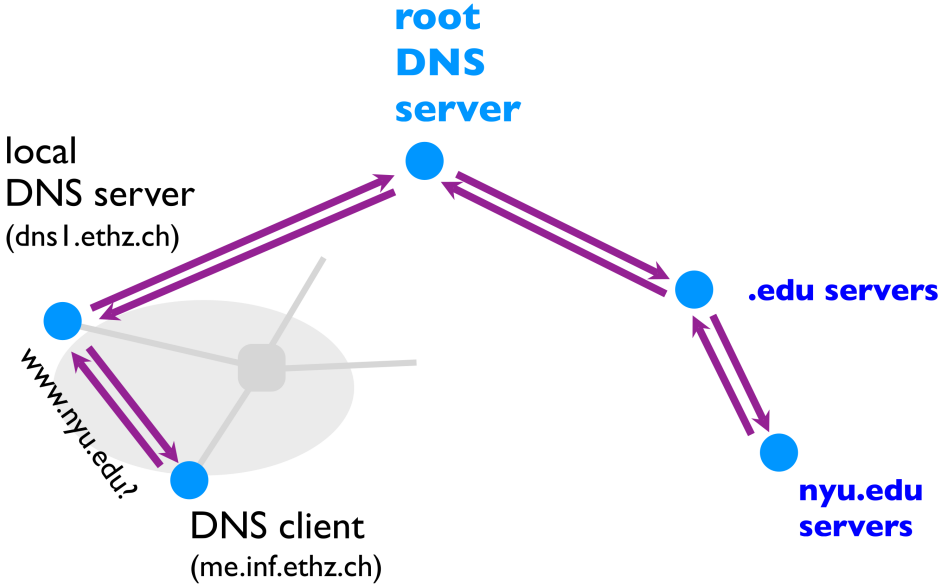
    - **Time-to-Life (TTL):** How long the mapping is valid
      - Used for caching

2.1.2 DNS Query

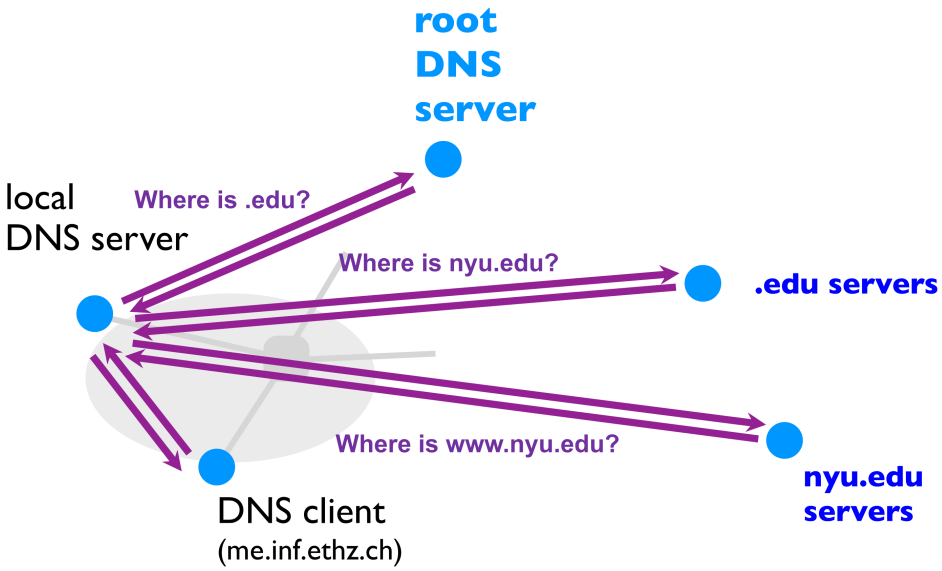
- Used UDP on port 53
  - Unreliable; may not get an answer
- **Main Steps**
  - Browser wants to access a domain name
  - OS triggers resolution process by sending request to local DNS server
  - DNS systems translates domain
- **Local DNS Server**
  - Normally closed to the endhost
  - Can be either:
    - **Locally:** Running on host
    - **Enterprise:** Somewhere in the network
    - **ISP:** Running by the ISP
    - **External:** Run by a third-party company
      - Wrong geographical local
      - PGB Anycast does not give fastest server
      - Insecure

- ◇ Endhosts can choose a DNS server by:
  - ∗ Hardcoding it to `/etc/resolv.conf`
  - ∗ Rely on dynamic allocation by `DHCP`

- **Query Type**
  - ◇ **Recursive Query**
    - ∗ Client offloads task of resolving to the next server
    - ∗ Not done in practice
      - ○ Except for external resolvers apparently
    - ∗ Steps:
      - ○ DNS client asks DNS server for domain IP
      - ○ DNS server queries root server
      - ○ Root server queries TLD server
      - ○ TLD server queries domain server
      - ○ Domain server returns domain IP to TLD server
      - ○ TLD server returns domain IP to root server
      - ○ Root server returns domain IP to DNS server
      - ○ DNS server return domain IP to DNS client



  - ◇ **Iterative Query**
    - ∗ Each server only returns address to next server
    - ∗ Steps
      - ○ DNS client asks DNS server for IP
      - ○ DNS server queries root server
      - ○ Root server returns TLD server IP to DNS server
      - ○ DNS server queries TLD server at given IP
      - ○ TLD server returns domain server IP to DNS server
      - ○ DNS server queries domain server at given IP
      - ○ Domain server returns domain IP to DNS server
      - ○ DNS server return IP to DNS client



* Somehow we sometimes say that CNS client is recursively (makes only one request) while the DNS server is iteratively (makes many requests)

2.1.3 Caching

- Most DNS queries requests the same site
- Cache result of former queries
- DNS records is cached for TTL seconds
- Hit rate of about 75%

2.1.4 Problems

- DNS is vital for the internet
  - ◇ Server not accessible when DNS server not available
- DNS servers can see the sites one visits
  - ◇ Sell data
- Vulnerable to different security breaches

2.2 Web

- Distributed database of pages
- Composed of:
  - ◇ **Infrastructure:** Client/Browser, servers, proxies
  - ◇ **Content:** Objects organized in web sites
  - ◇ **Implementation:** URL, HTTP

2.2.1 URL

- **Uniform Resource Locator (URL):** address to a internet source
- `protocol://hostname[:port]/directory_path/resource`
  - ◇ **Protocol:** e.g. HTTP(S), FTP, SMTP etc.
  - ◇ **Hostname:** DNS name or IP address
  - ◇ **Port:** Optional and defaults to protocol standard
    - * 80 for HTTP, 443 for HTTPS
  - ◇ **Directory_path/Resource:** Identify resource on destination

2.2.2 HTTP

- **Hypertext Transfer Protocol (HTTP)**

+ Simple
- Properties
  - ◇ Synchronous Request-Response: make request and wait
  - ◇ Layered over a bidirectional byte stream: Build on TCP (most of the times)
  - ◇ Text-Based: Human readable
  - ◇ State-Less: No info about past client requests
- **Request**
  - ◇ Message is composed of header and body
  - ◇ Following three header fields are required
  - ◇ **Method:**
    - ∗ **GET:** request resource
    - ∗ **HEAD:** request headers only
      - ○ E.g. used by bots
    - ∗ **POST:** send data to the server
  - ◇ **URL:** Relative (to server) address of resource
  - ◇ **Version:** HTTP version to use
  - ◇ Can have many other optional header fields
    - ∗ Header has a variable length
- **Response**
  - ◇ Message is composed of header and body
  - ◇ Following three header fields are required
  - ◇ **Version:** HTTP version to use
  - ◇ **Status:** Status code
    - ∗ **1XX:** Informational
    - ∗ **2XX:** Success
    - ∗ **3XX:** Redirection
    - ∗ **4XX:** Client Error
    - ∗ **5XX:** Server Error
  - ◇ **Phrase:** Status message
    - ∗ Associated with status code
  - ◇ Can have many other optional header fields
    - ∗ Header has a variable length
- **Cookies**
  - ◇ **Stateless:** Each request is treated independently
    - \+ Server-side scalability
    - \+ Failure handling is trivial
    - \- Some application need state
  - ◇ **Cookie:** Small textfile saved on the client side
    - ∗ Sent along with each request
    - ∗ Server can request us to update the content
- **Web Page**
  - ◇ Web pages consist of man different resources
  - ◇ All resources have to be fetched
  - ◇ Resources have to be compile according to their dependency
    - ∗ Prevent race conditions
    - ∗ Browser has to be conservative
    - ∗ Model as DAG

        - **Node:** Task (fetch or compile)
        - **Edge:** Must-happen-before Relationship
        - **Weight:** Time predeceasing task takes
      - Loading time is equal the critical path (the longest path) length
      - Find using either algorithm:
        - Top-Sort
        - Recursively find the longest path for each node
      - Costs $O(n + v)$
      - Speed-up task on critical path will shorten or change the critical path
    - Loading time is ambiguously defined
      - I.e. not clear when page has finished loading
- **Speed-up**
  - Many techniques to speed-up the loading
  - **Simplify and Restructure Web Page**
    - Prevent cross dependency
    - Better compression
    - Inline code
    - Mark resources as asynchronous
    - **Minify:** Make code as compact as possible
  - **Faster End-Devices**
    - Shorter rendering time
  - **Better Network Infrastructure**
    - Decrease fetching time
    - Diminishing Returns
  - **Decrease RTT**
    - Great improvement
    - Hard to achieve in practice
  - **Simplify Network Protocols**
    - Naive HTTP opens one TCP connection per object
      - Fetch n objects costs $2n$ RTT
    - Use M parallel connection
      - Fetch n objects costs $2n/M$ RTT
      - Burden to server
      - Bandwidth contention
    - Use persistent connection
      - Fetch n objects costs $n + 1$ RTT
      - Default in HTTP/1.1
      + Reduce connection setup and tear down cost
      + Allow for more accurate RTT estimation
      + Allow TCP congestion window to increase
    - Use parallel requests/responses
      - Fetch n objects costs 2 RTT
        - Assume that the packets are not too large
      - Pack multiple requests into one large
  - **Caching**
    - Same, unchanged resources are fetched over and over
    + Saves time
    + Decreases server load
    + Decreases network load

        * Significant portion of objects are uncachable
            - Dynamic data (e.g. weather)
            - Scripts (where the result depends on some parameters)
            - Cookies (result may depend on sent cookie)
            - SSL encrypted content
            - Advertising
        * Clients send in the request the time they have last fetched the object. The server sends the object only when it was changed in the mean time.
        * Caches are at:
            - **Client:** Browser Cache
            - **Close to Client:** By ISP, CDN or enterprise network
            - **Close to Destination:** Reverse proxy
    ◇ **Replication**
        * See CDN section below

2.3 Internet Video

- Requirement: Good quality without rebuffering
- Different viewers have different setups and therefore have different requirements
- **End-End Workflow:**
    ◇ Encode video in multiple qualities and resolutions
        * Resulting in encodings of different bitrate
    ◇ Replicate content using CDN
    ◇ Player picks appropriate quality
        * Manifest file gives information about available bitrates
        * Clients can pick different qualities for different chunks
            - Each chunk is about 1 to 3 seconds long
        * **Capacity Estimation:** Estimate bandwidth and choose video with smaller bitrate than estimated bandwidth
        * **Buffer-Based Adaption:** Choose bitrate based on buffered duration
            - Need to differentiate startup phase
        * Both algorithms can be combined
- **Bitrate** of a video is the size of video divided by the length of the video

2.4 Content Delivery Network / Content Distribution Network (CDN)

- Not an application itself but used by web and video
- Spread popular content among data centres around the world
- Interconnect CDN server
- Direct clients to CDN instead of original server
    ◇ Done using either:
        * **DNS-Based**
            - Returns different IP addresses for same domain
            - Based on geo-location
            - Requires rewriting of URL in the source files
            - May not result in good experience when using external DNS resolvers
        * **BGP Anycast**
            - Always give same IP
            - BGP directs traffic to “closest” CDN server
            + Easy

- Less flexible
- Less control

• Two types of caching:
  ◇ **Pull Caching:** Cache whatever clients request
  ◇ **Pull Replication:** Cache popular content before user request it

+ Load balancing
+ Better reliability
  ◇ No single source of failure
+ Content is closer to the client
  ◇ Reduce latency
+ Speedup uncachable data

3 Transport Layer

- Link between network and application layer
  - ◇ Network is best-effort
    - ∗ Packages can get:
      - ○ Lost
      - ○ Corrupted
      - ○ Reordered
      - ○ Delayed
      - ○ Duplicated
  - ◇ Applications should be restricted to app-specific functionality
  - ◇ Transport layer is key
- Requirements
  - ◇ Data delivery to the correct application
    - ∗ Demultiplexing required
  - ◇ File or byte-stream abstraction for applications
  - ◇ Reliability
  - ◇ No overload of receiver or network

3.1 Reliable Transport

- Key properties
  - ◇ **Correctness:** Input is equal output
    - ∗ **Formal Definition:** A packet is always resent if the previous packet was lost or corrupted. A packet may be resent at other times.
  - ◇ **Timeliness:** Minimize time between input and output
  - ◇ **Efficiency:** Minimize use of bandwidth
  - ◇ **Fairness:** Respect other players

3.1.1 Protocol

- Incrementally build up a reliable transmission protocol
- **Lossy Channel**
  - ◇ **Goal:** Lossless transmission
  - ◇
```
# Sender side
for word in list:
    send_packet(word)
    set_timer()

    upon timer going off:
        if no ACK received:
            send_packet(word)
            reset_timer()
    upon ACK:
        pass

# Receiver Side
receive_packet(p)
if check(p.payload) == p.checksum:
    send_ack()
```

```
        # Send word to application if not already done
        if word not delivered:
            deliver_word(word)

    else:
        pass
```

  - ◇ Timer determines when packet is lost
  - ◇ **Timeout:** Difficult to set
    - ∗ **Short:** Unnecessary retransmissions
    - ∗ **Long:** Slow transmission
      - ○ Long wait for missing packets
    - ∗ Tradeoff between timeliness and efficiency
  - \+ Is correct
  - \- Sends 1 packet per RTT
- **Improve Timeliness**
  - ◇ **Goal:** Improve timeliness and efficiency
  - ◇ Send multiple packets at the same time
  - ◇ Add sequence number to each packet
  - ◇ Add buffers to sender and receiver:
    - ∗ **Sender:** Store packets send but not acknowledged
      - ○ Delete once acknowledged
    - ∗ **Receiver:** Store out-of-sequence packets received
      - ○ Delete once in order and delivered
  - \+ Sends multiple packets per RTT
  - \- Sender can easily overwhelm receiver
    - ∗ By sending too many packets
- **Flow and Congestion Control**
  - ◇ **Goal:** Prevent overwhelmingness of receiver and of network
  - ◇ **Sliding Window:** Restrict the sender and receiver buffer to a certain size
  - ◇ **Sending Window:** List of sequence numbers the sender is allowed to send
    - ∗ Move forward iff the lowest packet number in the windows was acknowledged
    - ∗ **Congestion Control:** Prevent overwhelming network
  - ◇ **Receiving Window:** List of sequence numbers the receiver is allowed to receive
    - ∗ If lowest packet number in the window is received:
      - ○ Send acknowledgement for that packet
      - ○ Move sending window forward
    - ∗ **Flow Control:** Prevent overwhelming receiver
  - ◇ `size(sending window) <= size(receiving windows)`
  - ◇ Window size is exchanged during handshake
  - ◇ **Bandwidth-Delay-Product:** Window size should be Bandwidth $\cdot$ RTT
    - ∗ Send so many packets that we stop sending right before receiving the first acknowledgement
    - ∗ Maximizes timeliness
- **Improve ACKs**
  - ◇ Efficiency is fully determined by the acknowledgement and how to handle losses
  - ◇ Different algorithms
  - ◇ **Individual ACKs**

        * Acknowledge every single packet received
        * Losses are indicated implicitly by the missing ACK for that number
        * Resend missing packet after k subsequent ACKs of other packets
        + **Known Fate:** We have feedback for every single packet
        + Simple algorithm
        + Not sensitive to reordering
        - Loss of a ACK packet causes retransmission of that packet
    - **Cumulative ACKs**
        * Acknowledge each packet but send the highest sequence number for which all previous packets were received
        - Not obvious which packets were lost
            - Duplicate ACKs (same sequence number acknowledged) is a sign of loss
            - We know some packets were delivered but not which one
            - We do not know which packet to retransmit
    - **Full Information ACKs**
        * Combine individual and commutative ACKs
            - I.e. say up to which were all received in-order and which were received out of order
        + Complete information
        - Overhead
            - When large of gaps in the receiver window
            - Lower efficiency
- **Fairness**
    - How is fairness defined?
    - Fairness and efficiency do not always play along
    - Give users with small demand what they want and distribute the rest evenly
    - **Max-Min Fair Allocation:** maximize lowest demand, then second lowest, etc.
        * Algorithm
            1. Start with all flows at 0
            2. Increase all flows till there is a new bottleneck
            3. Hold the rate of the flows which are bottlenecked
            4. Repeat from step 2 for the remaining flows
        * We do not know the available bandwidth in practice
        * Approximate it by repeatedly increasing window size and decreasing if upon a loss
            - Loss is sign of congestion
- **Regard corruption, reordering, delays, duplicates**
    - **Corruption**
        * Rely on checksum
        * Treat corrupted packets as lost
    - **Reordering**
        * Creates duplicate ACKs
        * Depends on applies ACKing mechanism
        * Only a problem for commutative ACKs
            - Sender may think that the packet was lost and resends
    - **Delays**
        * Creates timeouts
        * Cannot be fixed
    - **Duplication:**

        - May me detected by the receiver
        - Created duplicate ACKs
        - Depends on applied ACKing machanism
        - Only a problem for commutative ACKs
            - Sender may think that the packet was lost and resends

3.1.2 Example Protocols

- **Go-Back-N (GBN)**
    - Sliding window protocol
    - Uses cumulative ACKs
    - **Goal:** Receiver should be as simple as possible
    - **Receiver**
        - Delivers packets in-order to the application
        - Uses commutative ACKss
    - **Sender**
        - Uses a single timer to detect loss
            - Is reset at each ACK
        - Upon timeout, resent the whole window starting with the lost one
- **Selective Repeat (SR)**
    - Sliding window protocol
    - Uses individual ACKs
    - Tries to avoid unnecessary retransmissions
    - **Receiver**
        - Uses individual ACKs
    - **Sender**
        - Uses a per-packet timer to detect loss
        - Upon timeout, resent the lost packet

3.2 Internet's Transport

- Two main protocols

Property	TCP	UDP
Data delivery to the correct application	Demultiplexing using port	Demultiplexing using port
File or byte-stream abstraction for applications	Segmentation and reassembly	NO
Reliability	ACKs and checksums	(optionally) checksums
No overload of receiver	Flow control via receiver window	NO
No overload of network	Congestion control via sender window	NO

- Do not (necessarily) provide
    - Delay and/or bandwidth guarantees
        - Would required changes at IP level
    - Sessions that survive change-of-IP-address

3.2.1 User Datagram Protocol (UDP)

- Unreliable, connection less, out-of-order data transfer protocol

- Simple extension of IP
- **Header**
  - ◇ 4 header fields
  - ◇ **SRC Port:** Port of destination
  - ◇ **DST Port:** Port of source
  - ◇ **Checksum:** Checksum calculated over data and pseudoheader
  - ◇ **Length:** Size of the data section
- **Data Section:**
  - ◇ Variable length

\+ No delay of connection establishment
\+ No delay for ordering and reliable delivery
\+ Small packet overhead
  - ◇ Due to small header

\+ Better scalability
  - ◇ Due to no state, buffer and timer

\+ Send data right away
  - ◇ No flow and congestion control
  - ◇ Just send it

- **Halve-Duplex:** One-way channel

3.2.2 Transmission Control Protocol (TCP)

- Connection-oriented, reliable, in-order byte-stream transport protocol
- Connection / session is a single bytestream
  - ◇ State is stored in host and not network
- Design
  - ◇ **ACKs:**
    - ∗ Sequence number for each byte of the payload
    - ∗ Cumulative ACKs
  - ◇ **Checksum:** On data and pseudoheader
  - ◇ **Timer:**
    - ∗ Resent on timeout
    - ∗ Exponential backoff
  - ◇ **Fast Retransmit:**
    - ∗ Retransmit packet on 3 successive duplicate ACKs
    - ∗ Assume packet is lost without waiting for timer
  - ◇ **Flow control**
    - ∗ Sliding window of size no larger than receiver wants
  - ◇ **Congestion control**
    - ∗ Dynamic adaption to network paths's capacity
- **Full Duplex:** Two-way channel

Structure

- **Header**
  - ◇ $\geq$ 20 Bytes
    - ∗ At least 5 lines
    - ∗ Each line is 4 bytes long
  - ◇ **SRC Port:** Port of destination
  - ◇ **DST Port:** Port of source
  - ◇ **Sequence Number:** Starting byte offset of the data sent in this segment

- ◇ **Acknowledgment:** Next expected byte
- ◇ **HdrLen:** Number of lines in header (for optional options)
- ◇ **Must Be Zero:** has always to be 0
- ◇ **Flags:** One or multiple of *SYN*, *ACK*, *FIN*, *RST*, *PSH*, *URG*
- ◇ **Advertised Window:** Window size of the receiver
- ◇ **Checksum:** Checksum calculated over data and pseudoheader
- ◇ **Urgent Pointer:** Mark data urgent
- ◇ **Options:**
  - ∗ Additional options
  - ∗ Variable size

- **Data Section**
  - ◇ Called TCP segment
  - ◇ Contains multiple consecutive bytes of the byte stream
  - ◇ Variable length $\leq$ MSS
    - ∗ **Maximum Transmission Unit (MTU):** Maximal size of a IP packet
    - ∗ TCP packet consists of $\geq$ 20 Bytes header + segment
    - ∗ **Maximum Segment Size (MSS)** = MTU − (IP header) − (TCP header)
  - ◇ Take $\leq n$ bytes from the byte stream and put them in one segment
- Packet send, when one of:
  - ◇ Segment full
  - ◇ Timer times out
    - ∗ Different timer from the ACK timer
- **Sequence Number and ACKing**
  - ◇ **Initial Sequence Number (ISN):** Number of the first byte to send
    - ∗ Chosen at random
      - ◦ For security reason
      - ◦ Prevent confusion of packets when reusing ports
  - ◇ Sliding window starts at ISN + i
  - ◇ Receiver ACKs last +1 byte received in order
    - ∗ I.e. the next byte it expects
    - ∗ May send duplicate ACKs
- **Flow Control**
  - ◇ **Advertised Windows W:** Number of bytes the sender can send starting at the next expected byte
  - ◇ `size(receiver window) >= W`
  - ◇ At most W bytes can be in flight
  - ◇ Transfer speed = $\min\{B, \frac{W}{\text{RTT}}\}$
    - ∗ **Windows Size W** in bytes and constant
    - ∗ **RTT** in seconds and constant
    - ∗ **Bandwidth B** in bytes/second

Connection

- **Establishment**
  - ◇ **3-Way Handshake:**
    - ∗ A picks ISN at random
    - ∗ A sends packet
      - ◦ Set sequence number to A's chosen ISN
      - ◦ Set flag to *SYN* (synchronize sequence number)
    - ∗ B picks ISN at random upon receiving packet from A

        - B sends packet
            - Set sequence number to B's chosen ISN
            - Set acknowledged to A's ISN +1
            - Set flag to SYN and ACK
        - A sends packet
            - Set sequence number to A's ISN
            - Set acknowledged to B's ISN +1
            - Set flag to ACK
        - Channel set up finished
    - Loss of SYN
        - Sender resends SYN upon expiration of timer
        - Hard to set initial timer value
            - Unclear what is the distance
            - Default is 3 seconds
- **Tear Down**
    - **Normal Termination - One Side at a Time**
        - A send packet
            - Sets flag to FIN (finish)
        - B send packet
            - Sets flat to ACK
        - Connection is now **half-closed**
        - A can still receive packets from B
        - B send packet
            - Set flag to FIN
        - A sends packet
            - Sets flat to ACK
        - Connection is now **closed**
    - **Normal Termination - Both Together**
        - A send packet
            - Sets flag to FIN (finish)
        - B send packet
            - Sets flat to ACK and FIN
        - A sends packet
            - Sets flat to ACK
        - Connection is now **closed**
    - **Abrupt Termination**
        - Happens upon e.g. a crash
        - A send packet
            - Sets flag to RST (reset)
        - Data in flight is lost
        - If B continues to send data, A resends the RST
            - Can happen because RST is transmitted unreliably (B does not ACK the RST)

Timeouts and Retransmissions

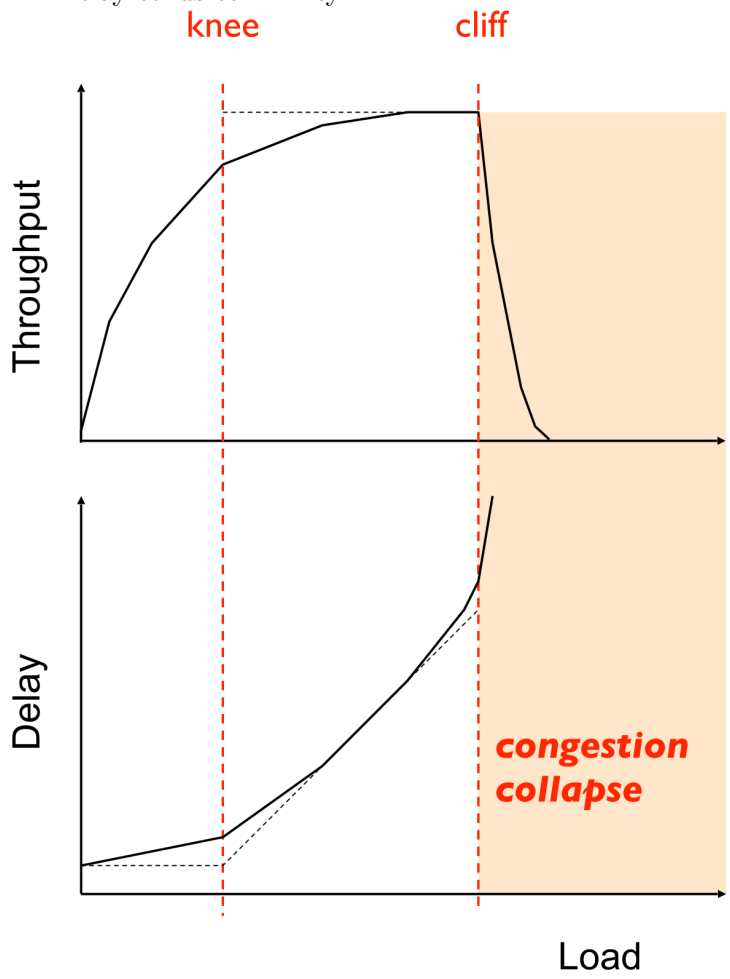
- Timer is set for last ACKed packet
    - I.e. if n is ACKed, we set the timer for n
- Timer is reset whenever new data is ACKed
    - Duplicate ACKs do not reset the timer

- Upon timeout the next expected packet is resent
- Setting the timeout is hard
  - ◇ **Short:** Duplicate packets
    - ∗ ACK cannot arrive in time
  - ◇ **Long:** Inefficient
    - ∗ Wait long when packet is lost
- We have to estimate RTT to set an appropriate timeout
  - ◇ Different algorithms
  - ◇ **Exponential Averaging**
    - ∗ **Idea:** Measure RTT in regular intervals and combine it with previous measurements
    - ∗ SampleRTT = AckRcvdTime − SendPacketTime
    - ∗ EstimatedRTT = $\alpha \cdot$ EstimatedRTT + $(1 - \alpha) \cdot$ SampleRTT
      - ◦ Initiated to 0
    - Hard to distinguish real ACKs and ACKs of retransmissions
  - ◇ **Karn/Patrige Algorithm**
    - ∗ Consider only original transmissions and not retransmissions
    - ∗ Use $\alpha = \frac{7}{8} = 0.875$
    - ∗ **Retransmission Timeout (RTO)** $= 2 \cdot$ EstimatedRTT
    - ∗ **RTO Expires:** Set RTO $= 2 \cdot$ RTO
    - ∗ **New SampleRTT recorded:** Reset RTO
      - ◦ RTO $= 2 \cdot$ EstimatedRTT
    - Can happen that RTT Sample > RTO
  - ◇ **Jacobson/Karels Improvement**
    - ∗ **Deviation** = |SampleRTT − EstimatedRTT|
    - ∗ **EstimatedDeviation:** Exponential average of Deviation
    - ∗ **RTO** = EstimatedRTT $+ 4 \cdot$ EstimatedDeviation
  - ◇ These algorithms are not used in practice
    - ∗ RTO is set to 200ms by default
    - ∗ Rather rely on duplicate ACKs
      - ◦ Resend after 3 duplicate ACKs

Congestion Control

- Was not part of the original TCP version
  - ◇ Sending rate only limited by flow control
  - ◇ In 1986 the internet was pretty much unusable tue to congestion
- Congestion is inevitable
- Increase networks load $\implies$ decrease of useful work done
- **Congestion Collapse**
  - ◇ Negative-Feedback Cycle:
    - ∗ Sudden increase of RTT $\implies$
    - ∗ RTT exceeds meximum retransmission interval $\implies$
    - ∗ Host begin to retransmit packets $\implies$
    - ∗ Host send same packet multiple times
  - ◇ Characterized by
    - ∗ **Knee:** With increasing load:
      - ◦ Throughput only increases slowly
      - ◦ Delay increases quickly
    - ∗ **Cliff:** With increasing load:

- Throughput decreases quickly
- Delay tends to infinity



- Solves the following:
  - ◇ **Bandwidth Estimation:** How to adjust the bandwidth of a single flow to the bottleneck bandwidth?
  - ◇ **Bandwidth Adaption:** How to adjust the bandwidth of a single flow to variation of the bottleneck bandwidth?
  - ◇ **Fairness:** How to share bandwidth "fairly" among flows?
- **Sender Window**
  - ◇ **Receiving Window RWND:** How many bytes can be sent without overflowing receiver buffer
    - ∗ Solves flow control problem
  - ◇ **Congestion Window CWND:** How many bytes can be sent without overflowing the network?
    - ∗ Solves congestion control problem
  - ◇ **Sender Window** $= \min\{\text{CWND}, \text{RWND}\}$
- **Congestion Detection**
  - ◇ There are multiple approaches
  - ◇ **Packet Delay:** Measure packet delay
    - Signal may be noisy
    - Delay varies considerably
    - Hard and risky

- ◇ **Packet Loss:** Measure packet loss
  - \+ Fail-safe signal
  - \+ TCP already monitors packet loss
    - ○ **Duplicate ACKs:** Mild congestion
    - ○ **Timeout:** Severe congestion

- **React to Congestion**
  - ◇ Gently increase window size when not congested, rapidly decrease when congested
  - ◇ **Slow Start**
    - ∗ **Goal:** Get a quick estimate of the available bandwidth
    - ∗ Slowly start and the increase exponentially
    - ∗ **Initial:** CWND = 1
    - ∗ **Upon ACK:** CWND = CWND + 1
    - \- Full window may get lost
  - ◇ **Fairness**
    - ∗ Two identical flows should end up with the same bandwidth
    - ∗ The sum of both flow should be close to the available bandwidth
    - ∗ **Trajectory Plot:** We want the bandwidth to be on the intersection of the efficiency and fairness line
  - ◇ **Additive Increase Multiplicative Decrease (AIMD)**
    - ∗ **Increase:** Add a constant to the point
      - ○ $(x, y) \to (\alpha + x, \alpha + y)$
      - ○ In TCP $\alpha = 1/\text{CWND}$
      - ○ $\implies$ Increment at most by 1 per RTT
    - ∗ **Decrease:** Take a fraction of the point
      - ○ In TCP set CWND = 1
    - ∗ Converges to fairness and efficiency
    - ∗ Fluctuates around optimum
  - ◇ **TCP Congestion Control**
    - ∗ **Slow-Start Threshold (ssthresh):** Threshold which determines switching between slow-start and AIMD
      - ○ **Initial:** ssthresh = ∞
      - ○ **On Timeout:** ssthresh = cwnd/2
    - ∗ **Fast Recovery**
      - ○ Reset CWND back to 0 is wasteful
      - ○ Try to react before network gets congested and we get a timeout
      - ○ Switch to AIMD
      - ○ Done after 3 consecutive duplicate ACKs
    - ∗ Full code:

```
Initially:
    cwnd = 1
    sshthresh = infinity

New ACK received:
    if (cwnd < sshthresh):
        # Slow Start
        cwnd = cwnd + 1
    else:
        # AIMD
        cwnd = cwnd + 1 / cwnd
```

```
        dup_ack = 0

    Timeout:
        # Multiplicative decrease
        sshthresh = cwnd / 2
        cwnd = 1

    Duplicate ACKs received:
        dup_ack ++
        if (dup_ack >= 3):
            # Fast recovery
            sshthresh = cwnd / 2
            cwnd = sshthresh
```

3.2.3 QUIC

- Deployed in 2015 by Google
- Has been standardized
- Many companies deploy it
- Built in many browsers
- Belongs to transport, security and application layer
  - ◇ Builds on top of UDP and is used by HTTP/2 slim
- Takes TCP header and puts everything into the UDP date section
- Has own congestion control etc.
- **Improvements**
  - ◇ **Head Offline Blocking:** One stream fails, stops other stream
    - ∗ Sliding window can not continue
    - ∗ Happens with HTTP/2
    - ∗ QUIC has multiple streams to prevent that
  - ◇ **Handshake**
    - ∗ TCP handshake is expensive
      - ○ TCP has 3-way handshake + 2-way for TLS
      - ○ Total of 3 RTT or so
    - ∗ QUIC needs only one initial handshake
      - ○ Combines transport and crypto handshake
    - ∗ **O-RTT:** Transport cookie prevents need for handshake if we were connected not long ago
      - ○ Cookies sent together with the request
      - ○ **Replay Attacks:**
        - ▷ Server reboot causes the server to not have a knowledge about recently connected clients
        - ▷ Clients have to resent request
        - ▷ Could cause multiple transactions
        - ▷ Nonce (random number) prevents this in TCP
          - · This is somehow a challenge
          - · A new one is sent which each request
  - ◇ **Change of IP**
    - ∗ TCP requires source IP to identify a connection
    - ∗ UDP does not do that
    - ∗ QUIC adds connectionID to identify a connection

        - Initialized randomly at handshake
    - ◇ **Flow Control**
        - ∗ Performed at two levels
        - ∗ **Connection Level:** Prevent stream from over-running the receivers buffer
        - ∗ **Stream Level:** Prevent individual streams from dominating the connection
    - ◇ **Evolve TCP**
        - ∗ Not really possible
        - ∗ Dropping certain package attributes looks suspicious and such packets get dropped
            - Network is conservative

3.3 Socket

- Application to transport interface
- End point of a "connection"
- Two main type of sockets
    - ◇ **UDP:** `SOCK_DGRAM`
    - ◇ **TCP:** `SOCK_STREAM`
- **Port**
    - ◇ Determine right destination application (demultiplexing) for a packet
    - ◇ 16 bit long
    - ◇ Source port and destination port are carried in the transport header
    - ◇ Different address space for TCP and UDP
    - ◇ **Well Known Ports:** $0 - 1023$
        - ∗ Bound to certain services
    - ◇ **Ephemeral Ports:** $1024 - 65535$
        - ∗ Assigned at random during connection
- OS provides mapping:
    - ◇ **UDP:** Destination port, destination IP $\leftrightarrow$ socket
    - ◇ **TCP:** Destination port, destination IP, source port, source IP $\leftrightarrow$ socket
        - ∗ Source information needed to identify connection
- Protocol, destination port, destination IP uniquely identify a socket

4 Algorithms

4.1 Algorithms

- **Max-Flow Min-Cut**
  - ◇ **Max-Flow Size:** How much traffic can a network carry
  - ◇ **Min-Cut Size:** How reliable is a network
  - ◇ Ford-Fulkerson
  - ◇ **Multi-commodity-flow (MCF):** Many different sources with different destinations
    - ∗ Requires specialized algorithms
- **Matching**
  - ◇ Analogous to finding circuits in networks
    - ∗ E.g. inside a switch of optical fibers
  - ◇ Can solve by max-flow
- **Shortest-Path**
  - ◇ Find optimal path in network

4.2 Linear Programming

- Technique for optimizing/solving linear objective functions
  - ◇ Given Linear equality and inequality constraints
- Use it as black box
- Very simple rules
- Modelling of problem is not unique
- Solved in polynomial time
  - ◇ But not always nice polynomials
- **Integer Linear Programming (ILP):** Constants are required to be integers
  - ◇ Is often NP-Hard
- Specific algorithms are often better
- Need to specify:
  - ◇ **Variables:** How does our setup look like
    - ∗ Real number
  - ◇ **Objective:** What is our goal
    - ∗ Max/min of a linear Combination of variables
  - ◇ **Constrains:** What constraints do we have
    - ∗ linear combination of variables
    - ∗ must be negative or zero
      - ○ Can always adopt it to follow that
- **Ex. Max-Flow**
  - ◇ **Variables:**
    - ∗ **f:** flow $s \to t$
    - ∗ $\mathbf{f_{uv}}$**:** flow on edge $u \to v$
  - ◇ **Objective:** Maximize f
  - ◇ **Constraints:**
    - ∗ **Capacity:** $f_{uv} \leq c_{uv}$
    - ∗ **End-Pts:**
      - ○ $\sum_{s \to \mathbf{v}} f_{sv} = f$
      - ○ $\sum_{\mathbf{v} \to t} f_{vt} = f$
    - ∗ **Conservation:** $\sum_{\mathbf{w} \to u} f_{wu} = \sum_{u \to \mathbf{v}} f_{uv} \quad \forall u \setminus \{s, t\}$
    - ∗ **Positive Flow:** $f_{uv} \geq 0$

- **Ex. Multiple Commodities Flow (MCF):**
  - ◇ Move commodities from different sources to different destinations
  - ◇ There is not specific algorithms for this problem
  - ◇ LP is best solution
  - ◇ Simplification: Each source x sends to only one destination $\text{dest}(x)$
    - ∗ Removed by identifying flows not only by source but also destination
  - ◇ **Variables:**
    - ∗ $\mathbf{f_x}$**:** flows
    - ∗ $\mathbf{f_{x,uv}}$**:** flow f_x on edge $u \to v$
  - ◇ **Objective:** Maximize $\sum_x f_x$
  - ◇ **Constraints:**
    - ∗ **Capacity:** $\sum_x f_{x,uv} \leq c_{uv}$
    - ∗ **End-Pts:**
      - ○ $\sum_{x \to \mathbf{v}} f_{x,xv} = f$
      - ○ $\sum_{\mathbf{v} \to \text{dest}(x)} f_{x,v\ \text{dest}(x)} = f$
    - ∗ **Conservation (flows do not mix):** $\sum_{\mathbf{w} \to u} f_{x,wu} = \sum_{u \to \mathbf{v}} f_{x,uv} \quad \forall u \backslash \{x, \text{dest}(x)\}$
    - ∗ **Positive Flow:** $f_{x,uv} \geq 0$
- **Ex. Shortest Path**
  - ◇ Minimize $s - t$ path length
  - ◇ **Variables:**
    - ∗ $\mathbf{x_{uv}}$**:** Indicator variable, is edge $u \to v$ on SP?
  - ◇ **Objective:** minimize $\sum_{\mathbf{u} \to \mathbf{v}} x_{uv} \cdot w_{uv}$
  - ◇ **Constraints:**
    - ∗ **Path Connected:**

$$\sum_{\mathbf{u} \to v} x_{uv} - \sum_{v \to \mathbf{w}} x_{vw} = \begin{cases} 0 & \text{if } v \notin \{s, t\} \\ 1 & \text{if } v = t \\ -1 & \text{if } v = s \end{cases}$$

    - ∗ **If we want the shortest path:** $x_{uv} \in \{0, 1\}$
    - ∗ **If we want the only the length of the shortest path:** $x_{uv} \in [0, 1]$
      - ○ I.e. it is okay if the flow is divided among multiple shortest paths (of equal length)

4.3 Randomness

- **Load Distribution**
  - ◇ n servers and one or multiple load balancers
  - ◇ How to distribute client requests?
  - ◇ Trivial Approaches
    - ∗ **Round Robin**
      - ○ Direct i-th request to $(i \mod n)$-th server
      - - Non trivial for multiple load balancers
      - - Does not work with sessions
    - ∗ **Least Loaded**
      - ○ Send requests to the least loaded server
      - - Overhead to determine least loaded
      - - Non trivial for multiple load balancers
      - - Does not work with sessions
  - ◇ **Random**

    * Select one server at random for each request
    * Equivalent to m balls into n bins problem
        - $P[i\text{-th and } j\text{-th ball go into same bin}] = P[X_{i,j}] = \frac{1}{n}$
        - $\mathbb{E}_{\text{Number of collisions}} = \mathbb{E}\left[\sum_{i,j \in 1,\dots,m, i \neq j} X_{i,j}\right] = \frac{1}{n}\binom{m}{2}$
        - $\mathbb{E}_{\text{Minimum load on bin } i} = \mathcal{O}(\frac{\ln n}{\ln \ln n})$
            ▷ For $m = n$
    * Improved Variant
        - Select two at random and take less loaded
        - $\mathbb{E}_{\text{Minimum load on bin } i} = \mathcal{O}(\frac{\ln \ln n}{\ln 2})$
        - Logarithmic improvement
        - Selecting more than 2 servers has diminishing returns
            ▷ 3 to 4 is still reasonable
            ▷ $\mathbb{E}_{\text{Minimum load on bin } i} = \mathcal{O}(\frac{\ln \ln n}{\ln k})$
    - Does not work with sessions
  ◇ **Random with Persist Sessions**
    * Send same sessions to same server
    * `server = Hash (user-id) % n`
    - Hash function is deterministic
        - Not quite random but still good enough
    * Hash can be calculated also over other header fields
        - e.g. `hash(src_ip, dest_ip, src_port, dest_port, TCP)`
        - Used for load balancing in data centers
    - Send all request to new server when one crashes
  ◇ **Random with Persistent Sessions and Consistent Hashing**
    * Do not send all session to new server on crash of one server
    * Hash the server IDs to the same space as the client requests
    * Map each client request to the "next" server in the hash
    - Huge load increase on successor server of failed server
        - Fixed by assigning a single server multiple hashes
- **Membership Testing and Counting**
  ◇ Used for
    * If object is in my CDN cache
    * If a TLS certificate has been revoked
    * If I have seen this packet before
    * How many packets of a flow have I forwarded (from router perspective)
  ◇ **Yes/No Hash Table**
    * We ignore collisions
    - Contains false positive
        - No problem as long as rate if low
        - No problem for many application
  ◇ **Hash Table with Bloom Filters**
    * We want to represent n object
    * We have m bits of memory in table T
    * We have k hash functions $h_1, \dots h_k$
        - With range $\{1, 2, \dots, m\}$
    * For each new input, we apply all hash functions and set the value to 1 in the table
    * $P[T[i] = 0 \text{ after } n \text{ insertions}] = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-kn/m}$
    * Value is member, if $T[i] = 1$ for all results of the hash functions

+ Cannot have false negative
- Can have false positive
  - P[Match is false positive] $= (1 - e^{-kn/m})^k$
  - $k \approx \frac{m}{n} \ln 2$ minimizes the probability
    - Minimal P[Match is false positive] $= 2^{-\frac{m}{n} \ln 2}$
      - Which is very small

◇ **Ex. Cache on Second Request**
* Most objects are only accessed once
* But when accessed more than once, it is very likely that it is accessed again
* First access: Add to bloom filter
* Second access (if present in bloom filter): Add to cache
* **Filter fills up:** Maintain primary and secondary filters
  - Once primary reaches threshold use secondary too
  - Once secondary reaches threshold, make secondary to primary and clear old primary (I guess)
  - Causes false negative

◇ **Counting Bloom Filters**
* Allow other cache filters to see what other cache servers cache
  - We need deletion to remove entries we no longer have
  - Count how many packets we have moved
- Can give false negative

• **Distributed Decision-Making**
◇ Selfish decision making can lead to bad performance for everyone
◇ **Braess's Paradox**
* We have start node S and end node E and two distinct $S \to T$ paths:
  - $S \xrightarrow{1} A, A \xrightarrow{x} T$
  - $S \xrightarrow{x} B, B \xrightarrow{1} T$
* Optimum latency for everyone is 1.5
* We add edge $B \xrightarrow{0} A$
* Everyone travels S, B, A, S
* Latency for everyone is 2

◇ **Hotelling's Law**
* Coop and Migros are always next to each other

5 Network Layer

5.1 General

- Builds on link layer and provides service to transport layer
- Transports packets
- Challenges
  - ◇ Scale to a global internet
  - ◇ Heterogeneity
  - ◇ Bandwidth control
  - ◇ Economics
- **Routing:** Figure out which path to take
- **Forwarding:** Send packet on its way
- **Network Service Models**
  - ◇ Describes what service does it provide to the transport layer
  - ◇ **Store-And-Forward Packet Switching:**
    - ∗ Both are implemented with store-and-forward packet switching
    - ∗ Routers receive packages, and store it (if necessary) in a FIFO quere before forwarding it
      - ○ Have a buffer per port (slightly simplified view)
  - ◇ Provide two different service models
  - ◇ **Datagrams (/Connection Less Service)**
    - ∗ Packets contain a destination address
    - ∗ Destination address is used to forward each packet from router to router
      - ○ Possible over different paths
    - ∗ Like postal letters
    - ∗ Prime type used today
    - ∗ Used for IP protocol
    - ∗ **Forwarding Table:**
      - ○ Used by each router
      - ○ Keyed by address
      - ○ Gives next hop for each destination address
      - ○ Is dynamic
        - ▷ Changes when new links are created and old are removed
  - ◇ **Virtual Circuits (Connection-Oriented Service)**
    - ∗ Uses circuit switching (in a virtual sense)
      - ○ There is no bandwidth reservation
      - ○ But statistical sharing of link
    - ∗ Like a telephone call
    - ∗ Three phases
      - ○ Connection establishment
        - ▷ Circuit is set up
        - ▷ Path is chosen
      - ○ Data transfer
        - ▷ Circuit is used
        - ▷ Packets are forwarded
      - ○ Connection teardown
        - ▷ Circuit is deleted
        - ▷ Circuit information is removed from all routers
    - ∗ Each packet has a shorter label

        - Used to identify the circuit and not destination
      - **Forwarding Table**
        - Used by each router
        - Keyed by the circuit identifier
          - Last router name, and label
        - Gives next router name and new packet label
      - **Multi-Protocol Label Switching (MPLS)**
        - Virtual-Circuit like
        - Used by many ISPs in their backbone
        - Adds MPLS fields upon entering their network
        - Removed MPLS field upon leaving their network
        - Takes up some digits of the IP
        + Allows forwarding on routes not possible using standard IP protocol
        + Potential increased switching speed
          - No longest-prefix matching required
        - Dated
        - Unflexible
        - Hard to setup
    - Datagrams vs Virtual Circuits

*	Setup phase	**Not needed**	Required
	Router State	**Per destination**	Per connection
	Addresses	Packet carries full address	**Packet carries short label**
	Routing	Per packet	**Per circuit**
	Failures	**Easier to mask**	Difficult to mask
	Quality of Service	Difficult to add	**Easier to add**

- **Internetworking**
  - Connection different networks together
  - Networks are different in:
    - Service model
    - Addressing
    - QoS
    - Packet size
    - Security
  - Hides the differences with a common protocol
  - **Internet Protocol (IP):**
    - Lowest common denominator
    - Asks for little
    - Provides little

5.2 IPv4

- **IP Datagram**
  - Each row is 32 bits
  - **Version:** 4 or 6
  - **IHL (Internet Header Length):** Length of header
  - **Type of Service:** distinguish different types of datagrams
    - real-time, non-real time etc.
    - For QoS
  - **Datagram length (bytes):** total length of datagram in bytes
    - At most 2^{16}

- ◇ **Identification:** TODO: What is identification for?
- ◇ **Flags:** Used for fragmentation
- ◇ **Fragment Offset:** Used for fragmentation
- ◇ **TTL:** Maximal desired number of hops
  - ∗ Decreased by one at every hop
- ◇ **Protocol:** TCP or UDP
  - ∗ Only considered at final destination
- ◇ **Header Checksum:** Checksum over header
- ◇ **Source Address:** 32 bit source
- ◇ **Destination Address:** 32 bit destination
- ◇ **Options:** Optional fields to set

• **IP Addresses and Prefixes**
- ◇ 32 bits long
- ◇ **Dotted Nottation:** Split into four parts of 8 bits length
  - ∗ $a.b.c.d$
- ◇ **Prefixes:** Addresses are allocated in blocks
  - ∗ Blocks are called prefixes
  - ∗ All addresses in the same L-bit prefix have the same top L bits
  - ∗ There are 2^{32-L} addresses in an L-bit block
  - ∗ The remaining represents the hosts on the network
  - ∗ Written as `IP Address / Prefix Length`
    - ○ IP address is the first address in the block
  - ∗ **More specific:** longer prefix
  - ∗ **Less Specific:** shorter prefix
  - ∗ Fist and last address in of prefix cannot be used
    - ○ **Network Identifier:** first address in prefix
      - ▷ Hosts bits are all zero
    - ○ **Broadcast Address:** last address in prefix
      - ▷ Host bits are all one
  - ∗ **Network Mask:** Together with the address give the prefix
    - ○ Bitwise and of mask and address give prefix/network identifier
      - ▷ Network bits are one
      - ▷ Host bits are zero
- ◇ **Public Addresses**
  - ∗ Assigned by IANA to different RIRs (regional internet registries)
  - ∗ RIRs issue ranges to ISP
  - ∗ ISP issue to customers
  - ∗ Buy IP4 is difficult
    - ○ Rund out of addresses
    - ○ Expensive
    - ○ May have been be misused before
    - ○ Geolocation
- ◇ **Private Addresses**
  - ∗ Only valid within a private network
  - ∗ Three prefixes are available
    - ○ 10.0.0.0/8
    - ○ 172.16.0.0/12
    - ○ 192.168.0.0/16
  - ∗ Need public IP and NAT to connect to global internet

- **IP Forwarding**
  - ◇ All IP addresses on one network have the same prefix
  - ◇ Next hop is determined using forwarding table
    - ∗ Indexed using IP prefix
    - ∗ Prefixes may overlap
  - ◇ **Longest Matching Prefix Forwarding Rule:**
    - ∗ Algorithm
      - ○ For each packet find all matching prefixes
      - ○ Find the longest prefix from that set
      - ○ Forward the packet to the next hop router for that prefix
    - ∗ **Compress Forwarding Table:** Remove redundant forwarding rules
      - ○ I.e. rules which are overwritten by more specific ones
    - ∗ Can provide default behaviour
      - ○ With less specific prefixes
    - ∗ Can provide special behaviour
      - ○ With more specific prefixes
    - ∗ Hierarchical addresses allow compact tables
    - ∗ Find longest matching prefix is more complex than table lookup
  - - Size of forwarding tables is keep growing
  - ◇ **Host Forwarding**
    - ∗ Host has a lookup table with two entries:
      - ○ **Network Prefix:** Send all traffic directly to the host
      - ○ **Default Route:** 0.0.0.0/0
        - ▷ Matches all addresses
        - ▷ Any more specific address is captured first
  - ◇ At each forward, the router must
    - ∗ Decrement TTL
    - ∗ Check and recalculate header checksum
    - ∗ Fragment larger packets if link network MTU is limited
    - ∗ Send congestion signals
    - ∗ Generate error messages
    - ∗ Handle options specified in the IP header

5.3 IP Helper Protocols

- IP **requires** help from other protocols
- There area many of then
- Often involve broadcast
- **Dynamic Host Configuration Protocol (DHCP)**
  - ◇ **Problem:** How does a host get
    - ∗ Its IP address
    - ∗ Routers IP address
    - ∗ Network prefix
    - ∗ **Default gateway**: Local router address
    - ∗ DNS Server
    - ∗ etc.
  - ◇ On connection, only the link MAC address is known
  - ◇ Introduced in 1993
  - ◇ Is a client - server application
  - ◇ Part of the network (router)

  - Uses UDP on port 67, 68
  - Steps
    - **Broadcast Message:** Client sends `DISCOVER` message to broadcast address
      - Message is received by all hosts on network
      - `255.255.255.255` for IP and `ff:ff:ff:ff:ff:ff` for MAC TODO: Why needed?
    - DHCP server send `OFFER` and offers certain options
    - Client send `REQUEST` to accept an offer
    - Server sends `ACK` an leases IP to client
  - Renewing lease only requires `REQUEST` followed by `ACK`
- Does not provide any security
  - Malicious host can act as a DHCP server
- **Address Resolution Protocol (ARP)**
  - Node needs link layer address to send frame but only has destination IP address
    - Provides IP to MAC mapping
  - Sits on the link layer
  - Only works if node and target are on the same link TODO: What does that mean?
  - Steps
    - Nodes sends a `REQUEST` broadcast message with an IP address
    - Target sees broadcast and notices its IP address
    - Target sends `REPLY` containing its MAC address
    - Node caches the MAC address in its ARP cache
  - Done using a table
  - Asks nodes addressed with their IP address to identify themselves

5.4 Packet Size

- **Maximum Transmission Unit (MTU)**
  - Maximal allowed packet size
- Different networks have different MTUs
- Goal: send packets as large as possible
  - Less header overhead
  - Fever packets required

- Hard to figure out appropriate size to send packets

- Two methods to solve this problem
  - **Fragmentation**
    - If a packet is too large for e given link, the router splits the packet into multiple smaller ones
      - Break data into pieces
      - Copy main IP header into the headers of all pieces
      - Adjust the `length` in the header
      - Set `fragment offset` in the header to indicate the position in the packet
      - Set `MF` (more fragments) `flags` in IP header on all pieces except the last
        - If set it means a fragment is following the current one
    - Receiver reassembles the fragmented packets
      - `Identification` links pieces together
      - `MF` tells than all pieces are together

    - Repeated fragmentation is possible $\implies$ messy
    - Fragmentation is undesirable
      - More work for routers

- Fragment loss $\implies$ packet loss $\implies$ retransmission of whole packet
- Security vulnerability

* Not used today anymore

◇ **Path MTU Discovery**

* Host sends largest packets
  - `DF` flag (do not fragment) set
  - Hosts use heuristisch on which MTUs to try
    ▷ There are protocols but they are not used in practice
* Routers send ICMP and tell their max largest size in case the original packet was too large for this router
* Hosts adjusts packet size to fit the MTU of the path

\- Path may change and break the transmission

\+ Avoids fragmentation

* Used today

5.5 Internet Control Message Protocol (ICMP)

- Sits on top of IP
- Is carried in an IP packet
- Used for:
  ◇ **Error:** Router sends ICMP when it encounters an error during forwarding
  ◇ **Testing:** TODO: Not sure what can be tested
- **Format**
  ◇ Header contains type, code and checksum
  ◇ Payload is often the problematic IP packet
  * For the host to identify the problem
  ◇ Types:

*	Dest. Unreachable	3/0 or 1	Lack of connectivity
	Dest. Unreachable	3/4	Path MTU Discovery
	Time Exceeded (Transit)	11/0	Traceroute
	Echo Request or Reply	8 or 0/0	Ping

- **Traceroute:**
  ◇ Allows host to find every router hop
  ◇ Sends probe packets with increasing TTL
  * Start with 1 and increase to N
  ◇ When TTL is 0 for a router, it reports that using ICMP
  * We can identify the router

\- Routers can identify probe and behave differently

\- Has not security measurements

5.6 Network Address Translation (NAT)

- Routers are supposed to look at only the IP header
- **Middlebox:** Device which should not, but look at transmission protocol header too
  ◇ Allows for new functionality
  * NAT Box
  * Firewall
  * Intrusion detection

\+ Possible rapid deployment path when there is not other option TODO: What does it mean?

  + Control over many hosts
  - Break end-to-end connectivity
    * Strange side-effects
  - Poor vantage point for many tasks
  - Cause internet ossification
    * Almost impossible to deploy new transport protocols
- **NAT Box**
  ◇ Done by a middlebox
  ◇ Connects an internal network to an external network
    * Does multiplexing
      ○ Many local hosts - single external IP
  ◇ Translates addresses and ports
- **How NAT works**
  ◇ Keeps a internal/external table
    * Contains internal to external IP and Port mapping
      ○ Port mapping is required that incoming packet get to the right host (in case there are multiple hosts with the same port)
  ◇ **Internal → external:** When an internal hosts establishes a TCP connection to the outside
    * Lookup in table
    * Create new entry if none exists
    * Rewrite source IP/Port of IP header
  ◇ **External → internal:**
    * Lookup in table
    * Rewrite address and port
    * If no entry in table the connection is blocked
      ○ Need to configure entry manually
- Connectivity broken
  ◇ Only works for initial outgoing traffix
- Problematic with UDP
  ◇ Since there is no connection
  ◇ Need special randezvous server for both parties to meet and both create internal to external mappings
+ Multiplexing
+ Flexible in address space
+ Useful functionality
  ◇ Firewall
  ◇ Privacy (hides network structure)

5.7 IPv6

- Replacement for IPv4
- IPv4 address space is exhausted
- Proposed in 1994
- 128 bits addresses
- Denoted in 8 groups of 4 hex digits (16 bits)
- Omit leading zeros within each group
- Omit one group of all zeros
  ◇ Not more than one, else original address cannot be reconstructed
- Not backwards compatible

- Specific routers required
- Header
  - Simpler
  - 32 bits per line
  - **Version:** 6
  - **Traffic Class:** Former type of service
  - **Flow label:** Allow router to identify a flow
    - Packets of same flow stay together (same path)
  - **Payload Length:** Former total length
  - **Next hdr (header):** Former protocol
  - **Hop Limit:** Old TTL
  - **Source Address:** 128 bits
  - **Destination Address:** 128 bits
  - **Flow:** Allows router to identify a flow
  - Removed
    - **IHL:**
      - **Streamlined Header Processing:** No option fields
        - Header has fixed 40 bytes size
        - Allows for faster processing
    - Fragmentation related fields were removed
      - Host must ensure that MTU is not exceeded
    - Header checksum is removed
      - Rely on transport layer checksum
- Hierarchical organisation
  - **Routing Prefix:** First $\geq$ 48 bit
    - Identifies ISP
  - **Subnet ID:** Middle $\leq$ 16 bits
    - Subnet or ISP customer
  - **Interface identifier:** Last 64 bits
    - Can be based on MAC address
- Address types
  - A single host can have multiple addresses for the same interface TODO: What for?
  - Local link address `fe80::/10`
    - Every hosts has it
    - Automatically setup
    - Based on MAC address
  - **Global Unique Address (GUA):** currently `2000::/3`
    - globally reachable
    - The *normal* IPv6 address
  - **Unique Local Address (ULA):** `fc00::/7`
    - For local deployments
    - Can be NATed to GUA
  - **Loopback Address:** `::1/128`
  - **Unspecified Address:** `::0/128`
  - etc.
- **Neighbor Discovery Protocol (NDP):**
  - Provides ARP and DHCP functionality
  - Hosts can be configured through stateless address autoconfiguration (SLAAC) bases on NDP

        - Hosts automatically set their link-local address
        - *Router solicitation* and *router advertisement* provide DHCP-like functionality
        - Host perform *duplicate address detection* based on *neighbor solicitation* and *neighbor advertisement*
        - *Neighbor solicitation* and *neighbor advertisement* replace ARP

      TODO: Not sure what this all means
- NAT is possible but not needed
- Many routers still block incoming traffic
  - Manual configuration necessary
- Transition
  - Many approaches proposed
  - **Multiple stacks:** Router can talk both
  - **Translators:** Map between both addresses
  - **Tunnels:** Encapsulate packet in an other (IPv6 over IPv4, IPv4 over IPv6 etc.)
    - Hard to know where to start and end a tunnel
  - **Happy Eyeballs:** Favour IPv6 over IPv4
    - Should dual-stack hosts use IPv4 or IPv6?
    - Issue A (IPv4) and AAAA (IPv6) DNS query simultaneously
    - If AAAA is received, use IPv6 immediately
    - If A is received, wait for some milliseconds for AAAA
      - A is often faster since it is cached
    - If first connection attempt to IPv6 (or IPv4) is unsuccessful try with IPv4 (or IPv6)

TODO: Possibly split into data and control plane

5.8 Routing

- Allocates network bandwidth
- Needs to adapting to failures
- A Quick comparison of different mechanisms
Mechanism	Timescale	Adaption
Load-sensitive routing	Seconds	Traffic hotspots
Routing	Minutes	Equipment failures
Traffic Engineering	Hours	Network load
Network Provisioning	Months	Network customers
- Delivery models
  - **Unicast:** One to one
  - **Broadcast:** One to many (all)
  - **Multicast:** One to few (not all)
  - **Anycast:** Many to many
    - One source always to one destination
- Goals of routing protocols
  - **Correctness:** Find paths that work
  - **Efficient paths:** Given path is minimal for some metric
  - **Fair paths:** Does not starve any nodes
  - **Fast convergence:** Recovers quickly after changes
  - **Scalability:** Works well as networks grows large
- Setting is decentralized and distributed:
  - All nodes are alike
    - No controller

  - ◇ Nodes only know what they learn by exchanging messages with neighbours
  - ◇ Nodes operate concurrently
  - ◇ Maybe node/link/message failures
- Apply different methods for different parts/typologies of the network

5.8.1 Network Typologies

- **Initial Idea:** Each computer is directly accessible via its own IP
  - \- Not scalable as internet grows
- Multiple techniques to scale routing
- **IP Prefix**
  - ◇ **Idea:** Bundle a bunch of devices together
  - ◇ Hosts attach to routers as IP prefixes
    - ∗ Multiple hosts are grouped in one prefix
  - ◇ Router advertises prefixes for hosts
  - ◇ Router addresses are in the /32 prefix
    - ∗ They are the only node to contact since they forward the traffic
- **Hierarchical Routing**
  - ◇ **Idea:** Route to region before routing to more specific prefix
  - ◇ Builds on top of IP prefixes by bundling many prefixes
  - ◇ Outside a regions, a single nodes has only one paths to all hosts within a different region
    - \+ Reduces forwarding table size as it hides region-internal structure
  - ◇ Different nodes of one region may use different paths to a certain region
  - \- May lead to longer paths
- **IP Prefix Aggregation (and Subnets)**
  - ◇ **Idea:** Allow more flexibility in changing the IP prefix size
  - ◇ **Subnet**
    - ∗ Internally split one less specific prefix into multiple more specific prefixes
    - ∗ One prefix is advertised to the internet
    - ∗ Done by e.g. a company to split different departments
  - ◇ **Aggregate**
    - ∗ Externally join multiple more specific prefixes into one less specific prefix
    - ∗ One prefix is advertised to the internet
    - ∗ Done by e.g. ISP
  - ◇ We need to take care to not *hide* some prefixes
  - \+ Reduces the number of prefixes to advertise to the rest of the network
  - \+ Compressed forwarding table

5.8.2 Shorted Path Routing

- Naive approach to find routing between two nodes
- Find shortest path by some metric
  - ◇ **Latency:** Avoid circuitous paths
  - ◇ **Bandwidth:** Avoid small pipes
  - ◇ **Money:** Avoid expensive links
  - ◇ **Hops:** To reduce switching
  - ◇ Ideal is a combination of all
- We ignore workload and only consider topology
  - ◇ Workload is only considered by sophisticated routing protocols

- **Optimality Property:** Subpaths of shortest paths are also shortest paths
- **Source Tree:** All shortest paths from this source to all nodes
- **Dijkstra's shortest path algorithm**
  - ◇ Mark all nodes tentative
  - ◇ Set distance for source to 0
  - ◇ Set distance to ∞ for all nodes
  - ◇ While tentative nodes remain:
    - ∗ Find node N with lowest distance
    - ∗ Add link to N to the shortest path
    - ∗ Potentially relax distances of neighbours of N
  - \+ Finds shortest path in order of increasing distances from source
  - ◇ Runtime depends on cost of extracent min-cost node
  - \- Gives complete source tree
    - ∗ Not required in internet
    - ∗ Requires complete topology
- **Equal-Cost Multipath (ECMP)**
  - ◇ Extension to shortest path routing
  - ◇ Allow multiple shortest paths from node to destination to be used at once
  - ◇ Source tree gets a DAG
  - ◇ Found using modified Dijkstra
  - ◇ Different paths have different latencies
  - ◇ Try to route a flow on the same path
    - ∗ Else we get jitter
  - \+ Better reliability
  - \+ Better performance

5.8.3 Intra-Domain Routing

- Routing within an AS
- Setting
  - ◇ Nodes know only the cost to their neighbours
  - ◇ Nodes communicate only with their neighbours
  - ◇ All nodes run the same algorithms
  - ◇ Nodes and links may fail
  - ◇ Messages may be lost
- Two main approaches
- **Distance Vector Routing (DV)**
  - ◇ Solves shortest path problem
  - ◇ Early approach
    - ∗ Used by ARPANET and RIP
    - ∗ Rarely used anymore
  - ◇ Each node maintains a vector of distances to all other nodes
  - ◇ Steps
    - ∗ Initialize vector with 0 cost to self and ∞ to all others
    - ∗ Periodically send vector to neighbours
    - ∗ Add locally stored "distance to neighbour", to all values heard from that neighbour
      - ○ Do that for all vectors received
        - ▷ I.e. for all nodes
    - ∗ Update distance vector by selecting minimal distance for each destination

- ◇ News travels one hop per exchange
- ◇ After n exchanges every node know the shortest path of up to n hops
- \+ Simple
- \+ Works well
- \+ Good news travels quickly
- \- Bad news travels slowly
  - ∗ Slow adoption on failures
- \- **Count to infinity problem:** After certain nodes becomes unreachable, the other nodes hear from each other that there is still a connection
  - ∗ Some solutions exists
    - ○ Do not work in all conditions
  - ∗ **Split Horizon:** Omit routers learned from neighbours A in updates sent to A
  - ∗ **Poisoned Reverse:** Set metric $= \infty$ for routers learned from A in updates sent to A
- ◇ **Routing Information Protocol (RIP)**
  - ∗ Introduces in 1988
  - ∗ DV protocol
  - ∗ Used hop-count as metric
  - ∗ Supported networks with longest distance of 15 hops
  - ∗ Vector is send out every 30 sec
  - ∗ Timeout of 180 sec to detect failure of nodes
  - ∗ Included split horizon and poison reverse
  - ∗ Builds on UDP

• **Link-State Routing (LS)**
- ◇ Replaced DV
- ◇ **Flooding**
  - ∗ Required for link-state routing to work
  - ∗ Sends message to every host in the network
  - ∗ Similar to broadcast but less controlled
  - ∗ Each node:
    - ○ sends incoming message on to all other neighbours
    - ○ remembers forwarded messages to prevent resending
      - ▷ Implemented using sequence number
      - ▷ Each source node has a own sequence number space
  - ∗ **Stop-and-wait**
    - ○ Receiver acknowledges receipt else it is resent
    - ○ Makes flooding reliable
    - ○ Only one packet is send at the time
  - ∗ Each link carries the message at least once
  - \+ Simple
  - \- Inefficient
    - ○ One node may receive the same message multiple times
- ◇ Working
  - ∗ Two phases

1) Modes flood topology in the form of **link state packets (LSP)**
   - ○ LSP contains:
     - ▷ Nodes name
     - ▷ Distances to all adjacent nodes
     - ▷ Sequence number to identify the flood

- Each nodes learns full topology by combining all LSPs

2) Each node computes its own forwarding table
   - Using Dijkstra or equivalent
   - All nodes optimize on same metric
   - Neighbours agree on shared link's cost
     - Can by dynamic
   - Link cost should be larger than zero

* Topology changes
  - Update LSP and flood to network
  - Node re-compute routes
  - Covers multiple scenarios
    - **Link Failure**
      - Both nodes notice and update their LPS
    - **Node failure**
      - Adjacent nodes notice failure of link and update their LPS
      - Failed node got disconnected
    - **Adding Link or Node**
      - All adjacent nodes update their LPS
      - Potential new node sends LPS

- Certain nodes may ignore flooded LSPs due to:
  - Sequence number reach max or gets corrupted
  - Node may crash and loose sequence number
  - Network partitions and heals again

◇ Can be fixed by including an age on LSP and forget old information

- More computation than DV

\+ Better dynamics than DV

◇ **IS-IS and OSPF**

* LS protocols
* Widely used in enterprise
* **Intermediate System to Intermediate System (IS-IS)**
  - Introduced in 1987
  - Very similar to OSPF
  - Network can be separated into areas
* **Open Shortest Path First (OSPF)**
  - Introduces in 1989
  - Loosely based on IS-IS
  - Splits AS's network into different areas
    + Scalability
  - **Internal routers:** Any router inside an area which is **not** connected to the backbone
    - Only run the LS algorithm with routers belonging to the same area
  - **Area border routers:** Any backbone router connecting to an area
    - Belong to the backbone and the area
    - Run the LS algorithm for all connected areas
    - Condenses topological information of their connected areas and distributes it to the backbone
  - **Backbone routers:** Any router which is part of the backbone
  - **AS border router:** Any router connected to a different AS
    - Can be a internal or backbone router

        - ▷ Advertise external routing information throughout the AS
      - ○ Areas are identified by 32-bit number
        - ▷ Formatted as IP addresses
      - ○ Areas topology is hidden to other areas
      - ○ All networks are exclusively connected via the backbone
        - ▷ Has IP 0.0.0.0
      - ○ Algorithm
        - ▷ Each router keeps a LS database storing the areas topology
        - ▷ Routers send LS advertisements (LSA)
          - · Send as IP packets
          - · Contain originating router ID and its interface with respective cost
          - · Area border routers and AS boundary routers distribute additional LSAs containing information of their connected area or AS
        - ▷ LS database in a collection of all LSAs
          - · Should be identical for all routers in the same area
      - + Support ECMP
      - + Extensible
    - * **IS-IS vs OSPF**

	OSPF	IS-IS
Purpose	Designed for IP traffic	Neutral towards level-3 protocols
Encapsulation	Runs on top of IP	Runs directly on layer 2
Area boundaries	On routers	On links
	Router belong to mult. area	Router belong to single area
Backbone area	Yes	No
IPv6 support	Yes	Implicitly

- **DV vs LS**

Goal	DV	LS
Correctness	Distributed Bellman-Ford	Replicated Dijkstra
Efficient paths	Approx. with shortest paths	Approx. with shortest paths
Fair paths	Approx. with shortest paths	Approx. with shortest paths
Fast convergence	Slow - many exchanges	**Fast - flood and compute**
Scalability	**Excellent - storage/computation**	Moderate: storage/computation

  - ◇ Tradeoff between scalability and speed of convergence

5.8.4 Inter-Domain Routing

- Routing between multiple ASes
- **Autonomous System (ASes):** Independently administered network
  - ◇ Define their topology
  - ◇ Decide on the protocols to use
- Challenges
  - ◇ **Scalability:** Keep networks reliable while # prefixes and # networks is growing
  - ◇ **Privacy:** Companies do not want that their network topology gets public
  - ◇ **Policy Enforcement:** We want to follow certain policies
- Intra-Domain Routing protocols do not fulfil all requirements
  - ◇ **LS:** Does not fulfil any
  - ◇ **DV:** Only privacy is fulfilled
- **Border Gateway Protocol (BGP)**
  - ◇ Announces prefixes which certain ASes can reach directly or indirectly

- ◇ **Idea:** Advertise entire AS-level path instead of distances
- ◇ **Path-vector routing**
  - ∗ Construct AS-paths as vectors
  - ∗ Advertise AS-path
  - ∗ ASes prepend themselves to the path when the path propagates announcements
  - ∗ Loops are detected and omitted by routers
- ◇ **Policies**
  - ∗ Money is key
  - ∗ **Business Relationship**
    - ○ 2 ASes only connect if they have a business relationships
    - ○ Two main relationships
    - ○ **Customer/Provider**
      - ▷ Provider provides service to customer
      - ▷ Customer pays provider
      - ▷ Different payment methods
        - · **95-th Percentile**
          - · Most popular for ISPs/companies
          - · Measure traffic every 5 minutes in Mbps
          - · Order data
          - · Pay according to the 95-th percentile
        - · **By Usage:**
          - · Pay for what you need
        - · **Flatrate:**
          - · Mostly for consumers
    - ○ **Peer/Peer**
      - ▷ Both have a common interest
      - ▷ Not money is paid
  - ∗ **Routing Policies**
    - ○ Key rules

      1) Providers transit traffic for their customers
      2) Peers do not transit traffic between each other
      3) Customers do not transit between their providers
         - · No valley routing
    - ○ Traffic is only forwarded if one makes money by doing so
    - ○ **Selection:** Which path should an AS use?
      - ▷ Controls *outbound traffic*
      - ▷ AS prefer sending to:

        1) **Customers:** Get money
        2) **Peers:** Use for free
        3) **Providers:** Pay yourself
    - ○ **Export:** Which paths should an AS advertise?
      - ▷ Defines allowed *inbound traffic*
      - ▷

		send to customer	peer	provider
from	customer	✓	✓	✓
	peer	✓	-	-
	provider	✓	-	-

      - ▷ Controlled through advertisements
      - ▷ To prevent partition of the network, Tier-1 ASes must be connected through

a full-mesh of peers

- ◇ **Protocol**
  - ∗ Simple protocol
  - ∗ Two sessions flavours
    - ○ **external BGP (eBGP)**
      - ▷ Connect border routers in different ASes
      - ▷ Used to learn routes to external destinations
      - ▷ Used to announce internally disseminated routes
      - ▷ TCP based BGP session
    - ○ **internal BGP (iBGP)**
      - ▷ Connect routers in the same AS
      - ▷ Used to disseminate externally-learned routes internally
  - ∗ Contains four basic message types
    - ○ Are carried in an IP packet
    - ○ Composed of IP prefix and attributes
    - ○ **OPEN**
      - ▷ Establish TCP-based BGP session
    - ○ **NOTIFICATION**
      - ▷ Report unusual conditions
    - ○ **UPDATE**
      - ▷ Inform neighbour about:
        - · change in best route
        - · new best route
        - · removal of best route
      - ▷ Four different attributes:
        - · Describe route properties
        - · Used in route selection
        - · Can be:
          - · **Local:** Only seed by iBGP
          - · **Global:** Seen by iBGP and eBGP
        - · **NEXT-HOP**
          - · Global
          - · Indicates where to send the traffic next (indicates the egress point)
          - · Set when the route enters an AS
            - · Not changed within the AS
        - · **AS-PATH**
          - · Global
          - · Lists all the ASes a route has traversed
          - · AS prepends itself
          - · Prevents loops
          - · Controls inbound/outbound traffic
        - · **LOCAL-PREF**
          - · Local
          - · Set at the border
          - · Represents how *preferred* a route is
          - · Routers select the route with the highest value (if they are able to choose)
          - · May lead to longer intra-domain routing distances
        - · **MED**

            - Multi-Exit Discriminator
            - Global
            - Set by the announcer
            - Indicates the relative proximity of a prefix with respect to the announcer
            - Closer proximity $\implies$ lower MED $\implies$ preferred
            - If often ignored by the sender
            - Can only influence inbound traffic from a single AS and not prefer traffic from one AS over another
        - **KEEPALIVE**
            - Inform neighbour that the connection is alive
    - Route selection
        - **Single Path Protocol:** BGP selects a single route
            - Optimisation not possible
        - Selects routes according to: (select next lower if we have a tie)
            - higher LOCAL-PREF
            - shorter AS-PATH length
            - lower MED
            - select route learned from eBGP over iBGP
            - lower iBGP metric to the next-hop
                - Based on some intra-domain routing protocol
            - smaller egress IP address (breaks tie)
        - **Hot Potato Routing:** Dump traffic as soon as possible to someone else
            - ASes are selfish
            - Leads to asymmetric routing
                - Traffic often does not flow on same path in both directions
    - ASes enforce their policy using:
        - **Import Rules:** Enforced using input filters
            - Assign LOCAL-PREF for different route types
        - **Export Rules:** Enforced using input and output filters
            - Assigned tags to routes and use only routes with certain tags
- **Problems**
    - **Reachability**
        - Is not guaranteed
        - Connection may exists but it is not allowed to be used due to policies
    - **Security**
        - No security measurements in place
        - ASes can advertise any prefixes
        - ASes can arbitrary modify route content like AS-PATH
        - ASes can forward traffic along different paths than advertised
    - **Convergence**
        - Route may never converges due to policy osculation
        - We get route updates all the time
        - Only happens if there are nonsensical cycles
        - Caused by arbitrary policies by the ASes
        - **Gao-Rexford Rule:** If all AS policies follow the cost/provider rules, BGP is guaranteed to converge
        - Happens rarely
    - **Performance**

    - Selection is based on economy not performance
* **Anomalies**
    - Bloated and underspecified
    - Configuration is hard and often done manually by humans
    - Principle is fundamentally flawed
* **Relevance**
    - Policies are rapidly changing

6 Link Layer

- Transfers messages over one or more established links
- Messages are frames
- Frames have a limited size
- Adds header and sometimes trailer to network packet
- Link layer is split in two piece
  - ◇ OS driver
  - ◇ Network interface card (NIC)
    - ∗ Interface to physical layer

6.1 Framing

- Physical layer provides stream of bits
- Need to figure out where a frame starts and ends
- Different methods
- **Byte Count**
  - ◇ **Idea:** Add length field to header
  - \+ Simple
  - \- Hard to re-synchronize after error to length field
- **Byte Stuffing**
  - ◇ **Idea:** Byte length flag `Flag` which indicates start and end of frame
  - \- Flag may be part of the payload
    - ∗ Must be escaped: `ESC FLAG`
    - ∗ Escape may be part of the payload too
      - ○ Must be escaped itself: `ESC ESC`
  - \- Problematic on bitflip of flags/escapes
  - ◇ **Worst case**: Frame size gets double
  - ◇ **Average case**: ≈ 0.6 or $1/28$ or something, not sure TODO: Find average case
- **Bit Stuffing**
  - ◇ Same principle as byte stuffing
  - ◇ 6 consecutive 1s are a `FLAG`
  - ◇ **On transmit:** Insert a 0 after 5 consecutive 1s
  - ◇ **On receive:** delete each 0 which occurs after 5 1s
  - ◇ **Worst case:** Size gets longer by 20%
  - ◇ **Average case:** TODO: Find average case

6.2 Error Detection and Correction

- Get bit-errors due to signal attenuation and noise
  - ◇ We assume the bit flips are random
- **Goal:** Detect and possibly recover from such errors
- **Error Codes**
  - ◇ **Codeword:** Original data concatenated (‘+‘) with correction bits
    - ∗ Consists of `len(D) + len(R)` bits
    - ∗ **Original data D:** Data bits which if our message
    - ∗ **Check bits R:** Check bits which are computed as $R = \text{fn}(D)$
  - ◇ **Sender:**
    - ∗ Computes $R = \text{fn}(D)$
    - ∗ Sets R and sends frame of

    - ◇ **Receiver:**
        - ∗ Takes D' and computes $R' = \text{fn}(D')$
            - ○ **D′:** is D with possible, unknown errors
        - ∗ Compare $R = R'$
    - ◇ Set of correct codewords should be a tiny fraction of the set of all codewords
        - ∗ Reduces probability of a getting a correct codeword by randomly selection
- **Hamming Distance**
    - ◇ **Distance:** Number of bit flips required to turn $D_1 + R_1$ into $D_2 + R_2$
        - ∗ Where $D_1 \neq D_2$ and $R_1 = \text{fn}(D_1), R_2 = \text{fn}(D_2)$
    - ◇ **Hamming Distance** for a code is the minimal distance between any pair of codewords
    - ◇ **Error Detection:** Code of hamming distance $d + 1$ can detect up to d errors
    - ◇ **Error Correction:** Code of hamming distance $2d + 1$ can correct up to d errors
- **Error Detection**
    - ◇ Add check bits to the message
    - ◇ Different methods
    - ◇ **Duplicate Send of Message**
        - ∗ **Idea:** Send each message twice
        - ∗ Error if both are different
        - ∗ Can only detect a single error
        - ∗ Fails if same bit is flipped in both copies
    - ◇ **Parity Bit**
        - ∗ R has length 1
        - ∗ For D date bits, R is the sum (modulo or XOR) of the all bits of D
            - ○ I.e. $R \equiv_2 \sum_{i=1}^{D} d_i$
        - ∗ **Distance:** 2
            - ○ Can detect 1
            - ○ Can correct 0
        - ∗ Detect errors with 50% chance
        - - Fails for systematic errors where bits of the D are swapped
    - ◇ **Checksum**
        - ∗ $\text{len}(R) = N$
        - ∗ **Internet Checksum:** Production example
            - ○ Used in TCP, UDP etc.
            - ○ **Sender side:**
                - ▷ Arrange data in 16-bit words
                - ▷ Put zero in checksum position
                - ▷ Add numbers
                - ▷ Add any carry-over back
                    - · I.e. $abbbb \implies 000a + bbbb$
                - ▷ Calculate $ffff-$ res
            - ○ **Receiver side:**
                - ▷ Arrange data in 16-bit words
                - ▷ Add numbers (including checksum)
                - ▷ Add any carry-over back
                - ▷ Checksum will be non-zero, add
                - ▷ Calculate $ffff-$ res
                - ▷ If 0 ok, else error
            - ○ **Distance:** 2

        - ▷ Can detect 1
        - ▷ Can correct 0
      - ○ Often errors occur in consecutive bits
        - \+ Errors are always detected if first and last error bit are within 16 bit range
      - \- Vulnerable to systematic errors (switching of blocks)
    - \+ Easy and simple
    - \+ Stronger than parity
  - ◇ **Cyclic Redundancy Check (CRC)**
    - ∗ Widely used: Ethernet, WIFI, ADSL
    - ∗ **Generator:** bit pattern of length $k + 1$
      - ○ Both sides agree on a specific one
      - ○ Can be expressed as a polynomial over $GF(2)$
    - ∗ **Sender side:**
      - ○ Extend the $len(D) = d$ date bits with k zeros
      - ○ Divide by the generator C
        - ▷ If 1 in leading position, take XOR
        - ▷ If 0 in leading position, take new value down and drop leading 0
      - ○ Add insert remainder as check bits
    - ∗ **Receiver side:**
      - ○ Divide received message by C
      - ○ If 0 no error, else error
    - ∗ Protection depends on the generator C
    - ∗ **Distance:** 4
      - ○ For standard CRC-32 generator
      - ○ Can detect 3
      - ○ Can correct 1
    - ∗ Detect odd number of errors
    - \+ Not vulnerable to systemanis errors
    - ∗ Detect consecutive bit errors of length $\leq k$
- **Error Correction**
  - ◇ Add more check bits to recover from error
  - ◇ Hard because check bits can contain errors too
  - ◇ **Hamming Code**
    - ∗ Has hamming distance 3
    - ∗ **Sender side:**
      - ○ Calculate minimal k such that the equation is fulfilled $n \leq 2^k - k - 1$
        - ▷ **n:** Message length
        - ▷ **k:** Number of check bits
      - ○ Insert check bits into original message at positions $2^i, i \in \mathbb{N}_0$
      - ○ Check bit i is a parity for values with a 2^i in their value
        - ▷ $2^i = 1$ iff "sum of all values which contain 2^i" $\equiv_2 1$, else 0
    - ∗ **Receiver side:**
      - ○ Recompute check bit sums
        - ▷ Including the check bit value itself
      - ○ **Syndrome:** Result of sums arranged as binary number
        - ▷ $\cdots + p_3 + p_3 + p_1$
          - · **+:** Concatenation operator
      - ○ No error if syndrome is 0
      - ○ Bit flip at position $i =$ "syndrome" if syndrome $\neq 0$

        - ▷ Undo bit flip and get correct data
    - ◇ Practically used error correcting codes are much more involved
- **Detection vs Correction**
    - ◇ Which is better depends on the pattern of errors
    - ◇ **Error Detection:** more efficient if no errors are expected and they are large when they occur
        - ∗ Heavily used in link layer and application layer
    - ◇ **Error Correction:** needed when we expect errors (small number only) or if there is not time to retransmit
        - ∗ Heavily used in physical layer

6.3 Retransmission

- How to deal with:
    - ◇ lost messages
    - ◇ messages with unrecoverable errors
- **Automatic Repeat reQuest (ARQ)**
    - ◇ Used when errors are common or must be corrected
    - ◇ **Receiver side:**
        - ∗ Sends ACK iff receives message and it is correct
    - ◇ **Sender side:**
        - ∗ Resends message if no ACK received within a set timeout
    - ◇ **Stop-And-Wait:** implementation of ARQ

6.4 Multiplexing

- **Multiplexing:** Share a limited resource
- One of the core goals in networking
- Two main principles
    - ◇ **Time-Division Multiplexing (TDM):** User sends at fixed schedule
        - ∗ Send high rate at a fraction of time
    - ◇ **Frequency-Division Multiplexing (FDM):** Users send at different frequencies
        - ∗ Send low rate all the time
- **Multiple Access**
    - ◇ Network traffic is bursty
    - ◇ Infeasible to use TDM or FDM for every user
    - ◇ **Statistical Multiplexing:** Method of oversubscribing a limited resource
        - ∗ Assume that not everyone will use the resource at the same time
    - ◇ Need Multiple Access protocols to allocate users according to their demand
    - ◇ Two main principles
        - ∗ **Randomized Multiple Access**
            - ○ Nodes randomize their resource access attempts
            - \+ Low load situations
        - ∗ **Contention-Free Multiple Access**
            - ○ Nodes order their resource access attempts
            - \+ High load situation or when guaranteed QoS required

6.4.1 Randomized Multiple Access

- Assume we have a distributed system w/o leader

- **ALOHA**
  - ◇ Used in late 1960s
  - ◇ Send when there is data
  - ◇ On collision (no ACK received), drop packet
  - ◇ Try to resend after a random timeout
  - \+ Works well when little traffic
- **Carrier Sense Multiple Access (CSMA)**
  - ◇ Improves ALOHA
  - ◇ Listens before sending to reduce collision probability
  - ◇ Called **1-persistent** CSMA since once channel is free, it sends with probability 1
  - \- Collisions can still happen:
    - ∗ Two nodes listen at the same time and here nothing and start sending at the same time
    - ∗ One node is sending, another is listening, but not hearing anything due to delay
  - \- Upon collision, time is lost since some sending nodes may keep sending
    - ∗ Because they may not realize that there was a collision
- **CSMA with Collision Detection (CSMA/CD)**
  - ◇ Upon collision, abort sending and jam the wire
    - ∗ Make everyone sending aware of the collision so that they abort sending
  - ◇ **Jam:** Send for a short amount of time to make every sender aware of the collision
  - ◇ **Contention Slot / Slot time:** Time a node must wait before it can be sure that no other node's tarnmission has collided with its transmission
    - ∗ Is $2D$
      - ○ **D:** Longest transmission delay possible in the network
    - ∗ Imposes a minimal frame size such that sending takes at least $2D$ seconds
    - ∗ Else a sender may not detect a collision
- **CSMA "Persistence"**
  - ◇ Problem:
    - ∗ While one node is sending, other nodes which want to send will queue up
    - ∗ Once the wire is free, all try to send and will collide
  - ◇ **Binary Exponential Backoff (BEB)**
    - ∗ **Idea:** When N nodes are queued, each will send with a probability of $\frac{1}{N}$
    - ∗ Working
      - ○ Initially, node directly sends
      - ○ On first collision, node sends directly or 1 frame later with each 50% chance
      - ○ On second collision, node waits either $0, 1, 2, 3$ frames (with same probability) before sending
      - ○ On i-th collision, node waits either $0, \ldots, 2^i - 1$ frames
    - \+ Highly efficient
    - ∗ Used by Ethernet
  - ◇ Called **P-persistent** CSMA
- **Classical Ethernet**
  - ◇ Introduced in 1973
  - ◇ Hosts share a single coaxial cable
  - ◇ Allowed for 10 Mbps
  - ◇ Uses 1-persistent CSMA/CD with BEB
  - ◇ $2D$ is 64 bytes
  - ◇ **Frame Format**
    - ∗ Contains source and destination MAC address

        * Puts the IP package into the DATA field
            - Has size $0 - 1500$ bytes
        * Padding is added in PAD to mache frame reach minimal 64 byte size
        * CRC-32 checksum for error detection
- **Modern Ethernet**
    ◇ Based on switches
    ◇ No multiple access
    ◇ No problems of collisions anymore
- **Wireless Multiple Access**
    ◇ Main challenges
        * Nodes cannot hear while sending
            - Cannot detect collisions
            - Send full frame and only realize after that there was a collision
        * **Signal to Noise Ration (SNR):** Node can receive signal only if SNR is large enough
        * Nodes may have different coverage areas. Two nodes are:
            - **Hidden Terminals:** iff they cannot reach each other but can collide at an intermediate station
            - **Exposed Terminals:** iff they can hear each other but do not collide
                ▷ Both sender think there is a collision but there is actually none
                ▷ Want to send concurrently to increase performance
    ◇ **Multiple Access with Collision Avoidance (MACA)**
        * Alternative approach to CSMA
        * **Idea:** Send short handshake to ask if allowed to send
        * Rules
            - **Request-to-Send (RTS):** Send by sender node to ask for allowance
                ▷ Contains the frame length the sender wants to send
                · This prevents hidden terminals
            - **Clear-to-Send (CTS):** Send by receiver if sender is allowed to send frame
                ▷ Contains the frame length the sender wants to send
                · This prevents exposed terminals
            - Sender transmits frame
        * Hidden terminal does not receive senders RTS but receivers CTS
            - Terminal knows that it should not send to receiver
            - Terminal knows from the frame length of the CTS how long the transmission will take
        * Exposed terminal does receive RTS but not CTS
            - Terminal knows that it should not send to sender but is free to send to any receiver
        - Collision still possible
            - But less likely
    ◇ **WiFi**
        * Standard 802.11
        * Started in 1990s
        * Clients are wireless connected to wired AP
        * **Physical Layer**
            - Uses 20/40 MHz channels on ISM (public available but only very local) band
            - Different frequencies depending on the WiFi Version
        * **Link Layer**

        - Multiple access using CSMA/CA
        - RTS/CTS is optional
        - Uses ARQ
        - Funky addressing due to AP TODO: What does it mean?
        - Errors detection with a 32-bit CRC
        - MTU is larger than for Ethernet
        - Can have many additional features
      * **CSMA with Collision Avoidance (CSMA/CA):**
        - After link gets free, queued sender select a random backoff (gap)
        - Node with smallest backoff goes first
        - Prevent starvation by always using the rest of the backoff and only select a new backoff after having send

6.5 Contention-Free Multiple Access

- **Turn-Taking Multiple Access Protocols**
  - **Idea:** Order nodes and each can send (or pass) if it is their turn
  - Different possible orderings
  - **Token Ring**
    * Arrange nodes in a ring
    * "Send permission" token rotates to each node in turn
    + Fixed overhead
    + No collision
    + Regular change to send
    - Complexity
      - Who controls the rotation and what if controller does?
      - What if token is lost?
    - Higher overhead at low load
- Random multiple access is dominant and hard to beat

6.6 LAN Switches

- Used in modern Ethernet instead of a shared wire
- Hosts are wired to a switch ports using a twisted pair cables
- There are three types of devices:
  - **Hub or Repeater:** On physical layer
    * All ports are wired together
    * Signals are amplified among all ports
    * Similar to classic Ethernet but more convenient and reliable
  - **Switch:** Up to link layer
  - **Router:** Up to network layer
- **Inside a Switch**
  - Uses frame addresses (MAC address) to connect input port to the right output port
    * Done by a *fabric*
  - Multiple frames may be switched in parallel
  - **Full-Duplex:** Traffic can flow in both direction
  - Not multiple access control, so hosts can just send
  - Need buffers at input and output ports
- **Forwarding**

- ◇ Need to find the right output port based on the destination address in the Ethernet frame (MAC address)
- ◇ **Backward Learning**
  - ∗ **Idea:** Use table to map MAC address to port
  - ∗ Works in two steps
    - ○ **Receive frame:** "Learn" mapping my looking at source address and port on which the frame was received
      - ▷ Insert new mapping into table
    - ○ **Sending frame:** Index into table to get port, if not existing, broadcast frame on all ports
  - \+ Also works if multiple addresses are behind one port (e.g. another switch)
  - \- Table is of fixed size and can get overflow
    - ○ E.g. due to a hack
    - ○ Switch acts equivalent to a repeater (broadcasts every message)
  - \- Not working if there are forwarding loops
- ◇ **Forwarding Loops**
  - ∗ Loop in a topology
  - ∗ Used for reliability or by mistake
  - ∗ Cheap switches do not have a duplicate detection
  - ∗ Frame will loop infinitely
- ◇ **Spanning Tree**
  - ∗ Forwarding method fixing forwarding loops
  - ∗ **Idea:**
    - ○ Switches collectively find a spanning tree for the topology
    - ○ Frames are forwarded along the tree and not all links
    - ○ Broadcasts will go up to the root and down to all branches
    - ○ Use backward learning on the tree
  - ∗ **Spanning Tree Algorithm**
    - ○ Required
      - ▷ All switches run the same algorithm
      - ▷ They start with no information
      - ▷ Operate in parallel and send messages
      - ▷ Always search for the best solution
    - ○ Steps
      - ▷ Elect a root node of the tree
        - · Switch with the lowest address
        - · At the beginning, each nodes selects itself
      - ▷ Grow tree as shortest distance from the root
        - · On tie, select the one with the lower MAC address
      - ▷ Turn of ports for forwarding if they are not part of the tree
    - ○ **Root Port (RP):** Outgoing port towards the root node
    - ○ **Designated Port (DP):** Incoming port to a switch
    - ○ **Blocked Ports (BP):** Do not forward any traffic
    - ○ **Note:** When we have to run the algo on a network, we do not have to to it step-by-step but more directly
      1) Select node with lowest address ($\Longrightarrow$ root node)
      2) Find shortest-path tree from root
    - ○ Nodes send periodic updates to neighbours
      - ▷ Contains own address, address of the root and distance in hops to the root

    - ▷ Done to detect failure
  - Final topology may not be optimal for some nodes
- Tree may break when a link fails

7 Physical Layer

- How to send digital signals in an analog way
- Convert to digital bits

7.1 Link Model

- Abstraction
- Physical channel is characterized by:
  - ◇ **Rate R** in bps
    - ∗ Also called **bandwidth**, **capacity** or **speed**
  - ◇ **Delay D** in seconds
    - ∗ Depends on message length and media length
    - ∗ Also called **latency**
  - ◇ Other properties like *is channel broadcast* and its error rate
- **Latency L:** Delay to send a message over a link
  - ◇ Composed of two parts
    - ∗ **Transmission Delay T:** Time to put M-bit message on the wire
      - ○ $T = \frac{M[\text{bits}]}{R[\text{bits/sec}]} = \frac{M}{R}[\text{sec}]$
      - ○ Dominant today
    - ∗ **Propagation Delay P:** Time for bits to propagate across the wire
      - ○ $P = \frac{\text{Length}}{\text{speed of signal}} = \frac{\text{Length}}{\frac{2}{3}c} = D[\text{sec}]$
  - ◇ $L = \frac{M}{R} + D$
  - ◇ High if:
    - ∗ Slow rate or
    - ∗ Long link
- **Bandwidth-Delay (BD) Product:** Amount of data in flight
  - ◇ $\text{BD} = R \cdot D$
  - ◇ Measured in bits or messages

7.2 Types of Media

- **Wires - Twisted Pair**
  - ◇ Very common
  - ◇ Twists can reduce radiated signal or reduce effect of external interference
    - ∗ Subtracting the signal of both wires cancels out interference TODO: Or something like that
  - \+ Cheap
- **Wire - Coaxial Cable**
  - ◇ Also common
  - ◇ Copper core shielded with multiple layers of insulation and protection
  - \+ Better performance due to better shielding
- **Wire - Fiber**
  - ◇ Strands of glass inside multiple layer of protection
  - ◇ Transmit light captured inside tue to total reflection
  - \+ Enormous bandwidth
  - \+ Usable for long distances
  - ◇ Two variants
    - ∗ **Multi-Mode:** Send multiple wavelength of light simultaneously
      - ○ For shorter links

        - Cheaper
    - **Single-Mode:** Send a single wavelength
        - Up to $\sim$ 100km
- **Wireless**
    - Sender radiates signal over a region
    - Send to potentially many receivers
    - Nearby signals may interfere at a receiver
        - Coordination required
    - **Industry Science Medicine (ISM):** Bandwidth for everyone but only locally
        - Used by WiFi, Bluetooth etc.

7.3 Signals

- Analog signals encode digital bits
- A signal over time can be represented by its frequency components (Fourier analysis)
- More spectrum available the more data one can send
- Different frequencies have different properties (e.g. influence by rain etc.)
- Fewer frequencies $\implies$ less bandwidth $\implies$ degraded signal
    - The high frequencies carry a great deal of the data
- **Bandwidth:** Information carrying capacity
    - Measured in bits/sec
    - In electrical engineering it is measured in Hz
- **Signal on Wire**
    - It is delayed since it propagates at $\frac{2}{3}c$
    - It is attenuated
        - Especially signals above a cutoff frequency are highly attenuated
    - Noise is added to the signal
- **Signal on Fiber**
    - Light propagates with very low loss in three very wide frequency bands
        - Attenuation is low at these frequencies
- **Signal over Wireless**
    - Travel at speed of light
    - Attenuation increases quadratically over distance from the sender
    - Interference at the receiver if multiple senders use the same frequencies
        - **Spatial Reuse:** Interference leads to notion of spatial reuse
    - **Multipath:** Signals bounce of objects and interferes with itself

7.4 Modulation

- How to represents bits as signals?
- Multiple methods
- **Baseband Modulation**
    - Signal is sent directly onto the wire
        - Not working for fiber and wireless
    - **Non-Return to Zero (NRZ)**
        - High voltage (+1 V) represents 1, low voltage ($-V$) represents 0
        - Average voltage is 0 and therefore no current is flowing
        - Clock recovery is hard
    - **Clock Recovery**

        * Hard to count how many consecutive 1s (or 0s) were transmitted if the signal does not change for a longer time
    - **Non-Return to Zero Inverse (NRZI)**
        * Invert signal level on a 1
        + Breaks long runs of 1
        - Does not break long runs of 0
    - **Manchester coding:**
        * To send a 1 switch from 0 to 1
        * To send a 0 switch from 1 to 0
        * Value between clock holds the actual value
        + No clock recovery
        * There is also inverse manchester encoding
    - **4B5B**
        * Translate 4 bits pattern into specific 5 bits pattern
            - Done according to some translation dictionary
        * Has at most 3 consecutive 0s
        * Has at most 8 consecutive 1s
    - **More Signal Levels**
        * All introduced methods use 2 levels
        * By increasing it to e.g. 4 we could use 2 bits per symbol TODO: What does it mean?TODO: What are the cons?
- **Passband Modulation**
    - Carries signal by modulating a carrier
        * **Carrier:** Some base signal oscillating at a desired frequency
        * Modify this carrier to indicate 1s or 0s
    - Can module different things
        * **Amplitude Shift Keying:** Shift amplitude when sending 0
            - I.e. stop oscillation of carrier while sending 0
        * **Frequency Shift Keying:** Increase/decrease frequency depending if sending 0 or 1
        * **Phase Shift Keying:** Uncontinuous at change of single
    - Can also combine multiple modulations

7.5 Limits

- Channel is characterized by bandwidth B, signal strength S and noise N
    - B limits the rate of transmissions
    - S and N limit how many signals levels we can distinguish
- **Nyquist Limit:** Maximum symbol rate is $2B$
    - If there are V signal levels the maximum bit rate is $R = 2B \log_2 V$ [bits/s]
    - Ignores noise
- **Shannon Capacity**
    - Considers noise
    - **Signal-to-Noise Ration (SNR):** Limits the number of levels we can distinguish
        * Measure in decibels
    - SNR given as: $\text{SNR}_{dB} = 10 \log_{10}(S/N)$ [dB]
    - **Capacity C:** is the maximum information carrying rate of the channel
        * $C = B \log_2(1 + \frac{S}{N})$ [bits/sec]
- **Conclusion Wired/Wireless**
    - **Wires and Fiber:** For fixed R, engineers link to have required SNR and B

    - ◇ **Wireless:** For fixed B, SNR varies greatly
        - ∗ Designe R for worst SNR is wasteful
        - ∗ Must adopt R dynamically
- **Digital Subscriber Line (DSL)**
    - ◇ Used passband modulation
        - ∗ Modulate amplitude and phase
    - ◇ Uses separate bands for up- and downstream
    - ◇ On high SNR, use up to 15 bits/symbol
    - ◇ On low SNR, use 1 bit/symbol

8 Routing Security

- Can be divided into intra and inter-domain routing

8.1 Terminology

- **Secrecy:** Keep something hidden from unintended receivers
  - ◇ Most general term
- **Confidentiality:** Keeps someone else's data secret
- **Privacy:** Keep data about a person secret
- **Anonymity:** Keep identity of a person secret
  - ◇ More specific than privacy
- **Data Integrity:** Ensure data is correct
  - ◇ Correct syntax and unchanged
  - ◇ Prevents unauthorized or improper changes
  - ◇ For local data
- **Data Authenticity:** Ensure that data originates from claimed senders
  - ◇ For data send over network
- **Entity authentication/identification:** Verify the identify of a protocol participant
- **Encryption:**
  - ◇ **Asymmetric:** Public-private key
    - ∗ **Encryption Key/Public Key:** K
      - ○ Publicly known
    - ∗ **Decryption Key/Secret Key:** K^{-1}
      - ○ Kept secret
    - ∗ **Encrypt:** $E_K(\text{plaintext}) = \{\text{plaintext}\}_K = \text{ciphertext}$
    - ∗ **Decryption:** $D_{K^{-1}}(\text{ciphertext}) = \text{plaintext}$
    - ∗ Diffie-Hellman Key
    - ∗ Public-key encryption
    - ∗ Digital signature
      - ○ **Signature Generation:** $S_{K^{-1}}(\text{msg}) = \text{sig}$
      - ○ **Signature Verification:** $S_K(\text{msg}, \text{sig}) = \text{true/false}$
    - ∗ 3072 bit key for high security
    - \- Decryption and verification are computationally very expensive
  - ◇ **Symmetric:** Shared-Key
    - ∗ **Encryption key:** K
      - ○ Shared with recipient
    - ∗ **Decryption key:** K
      - ○ Shared with recipient
    - ∗ **Encrypt:** $E_K(\text{plaintext}) = \{\text{plaintext}\}_K = \text{ciphertext}$
    - ∗ **Decryption:** $D_K(\text{ciphertext}) = \text{plaintext}$
    - ∗ **Block cipher:** Encryption/decryption functions
    - ∗ 128 bit key for high security
    - \+ Encryption/decryption is computationally cheap
  - ◇ **Others:** Unkeyed symmetric
    - ∗ One-way function
    - ∗ Cryptographic hash function

8.2 Intra-Domain Routing

- It is assumed that attacks come from the outside (inter-domain)
  - ◇ Therefore the is not much security in intra-domain
- Compromise node (/router)
  - ◇ Flood any message we want
- Compromise link
  - ◇ Act as a man-in-the-middle (MITM)
- In both cases we are able to:
  - ◇ View to whole network topology
  - ◇ Inject any message they want
- We can do:
  - ◇ **Interception**
    - ∗ Aims at eavesdropping, dropping, modifying, injection or delaying packages
    - ∗ By injecting fake information, we can precisely control the network-wide behaviour
      - ○ Done by flooding about non-existing (virtual) nodes and links of arbitrary cost
  - ◇ **Denial-of-Service (DoS)**
    - ∗ Many different strategies
    - ∗ Induce chunk to overload routers by announcing and withdrawing routes at a fast pace
    - ∗ Flood routers link-state database by injecting thousands of prefixes
    - ∗ Induce congestion/high delay by steering traffic along fewer or low-throughput paths
    - ∗ Prevent reachability by steering traffic along black holes or loops
- **Problem:** Bogus advertisements can be injected or legitimate ones can be altered
- **Solution:** Use cryptic authentication
  - Routers are often underspecified for computationally expensive verification

8.3 Inter-Domain Routing

- Are regularly attacked
- BGP lacks any security measures
- Problems
  - ◇ Advertisement Origin Verification
    - ∗ **Problem:** No-one checks that a AS own the prefixes it advertises
    - ∗ Used for
      - ○ **Blackhole:** Data traffic is discarded
      - ○ **Snooping:** Data traffic is inspected, then redirected
      - ○ **Impersonation:** Traffic sent to bogus destinations
    - ∗ **IP Prefix Hijacking**
      - ○ AS advertises a prefix it does not own
      - ○ Some traffic is redirected to a bogus destination
    - ∗ **Sub-Prefix Hijacking**
      - ○ More specific prefix is announced
      - ○ Longest prefix matching selects the most specific prefix
      - ○ Bogus route is picked up by everyone
    - Difficult to detect and debug
      - ○ Often, no loss of connectivity but only performance degradation

        - Requirements
            - Access to router with BGP session
                - Local PC would not work
            - Getting access to the router
                - Configuration mistake
                - Operator himself launches the attack
                - Outsider breaks into
    - Advertisement Content Verification
        - **Problem:** No-one checks the content an AS sends
        - Possible attacks
            - **Remove ASes from the AS path**
                - Motivation
                    - Attract source ASes which try to avoid the AS we have removed to send traffic to us
                    - Pretend to some outer AS that it is closer to the core of the internet
                - Hard or impossible to figure out
                    - The AS who would route to the AS we have removed may realize that the link does not exists
            - **Added ASes to the AS path**
                - Called **BGP Path Poisoning**
                - Motivation
                    - Trigger loop detection in newly added AS
                        - DoS on added AS
                            - Because this path gets dropped due to loop detection
                        - Blocking unwanted traffic
                            - E.g. reroute traffic but in the end route traffic to right destination and thanks to loop detection nodes between us and destination ignore our announcement
                    - Make you AS look like it has richer connectivity
                - Inserted link or one of the adjacent could notice
            - **Add ASes to the end of the path**
                - Motivation
                    - Route traffic to bogus AS and from there to the actual destination
                - Hard to detect
            - **Advertise invalid paths**
                - Called **Route Leak**
                - Valid sequence but BGP policy gets violated
                - E.g. valley routing
                - Detected and defenced by filter of routes by prefixes and AS paths
            - **Export missing/inconsistent paths**
                - Do not advertise all links to connected AS
                - **Cold Potato Routing:** Make the other AS route the traffic from longer instead of us
                - Detected by analysis of BGP updates or inconsistencies in traffic
- BGP Today
    - Based on best common practices (BCPs)
        - Secure BPG session
        - Filter routes
        - Filter packets

    - ◇ Not good enough
- Proposed Enhancements
    - ◇ Multiple solutions
    - ◇ **BGP$_{\text{SEC}}$**
        - ∗ Should replace BGP
        - ∗ **Address Attestations**
            - ○ ASes need to claim the right to advertise prefixes
            - ○ Keys are signed and distributed out-of-band (not via the network)
            - ○ It is checked through the delegation chain from ICANN that prefixes are advertise by rightful ASes
        - ∗ **Route Attestations**
            - ○ Routes are signed by each AS as the route traverses the network
        - ∗ **Resource Public Key Infrastructure (RPKI)**
            - ○ Per-prefix certificate are issued by Regional Internet Registries (RIR)
            - ○ Used to authenticate first AS hop through Route Origin Authorization (ROA)
        - ∗ Can validate
            - ○ The path was not reordered
            - ○ No intermediate AS was added or removed to/from a path
        - ∗ **Challenges**
            - Need complete and accurate overview of prefix owner
            - Need public key infrastructure to know and validate the keys of ASes
            - Support for cryptographic operations is required
            - All operations need to be performed quickly
            - Incremental deployment is hard
                - ▷ Would need backwards compatibility
    - ◇ **Detecting Suspicious Routes**
        - ∗ Monitoring BGP update messages and use history to detect anomalies in the router's behaviour
        - High probability for false positive
        - Does not work well
    - ◇ **Avoiding Suspicious Routes**
        - ∗ Soft response to suspicious routes
        - ∗ Prefer routes that were used in the past and delay adoption of unfamiliar routes when possible
        + Attacks will go away over time
        + Gives operators time to investigate
        - ∗ Why is this good enough?
        - ∗ How well would it work?
- Data Plane Attacks
    - ◇ Routers should forward packets along the path chosen by the control plane
        - ∗ No checking mechanism
    - ◇ Attacker must get access of a router along the path (or hijack a prefix)
    - ◇ Attacks
        - ∗ **Drop Packets** but still send announcement packets
            - ○ Can filter for certain application
            - ○ Hard to detect
        - ∗ **Slow Down Traffic** but let traceroute through quickly
            - ○ Hard to debug
        - ∗ **Send packets to bogus destination:**

        - Packages can be snooped
        - After, they are forwarded along the right path
        - Hard to detect
    - How to
        - Adversary must control a route along the path or hijack prefix
        - How to gain control of a router?
    - Drop packets in the data plane
    - Easier to evade detection
    - Evade easier if you just slow done some traffic
- BGP is Very hard to fix
    - Complex system
    - Hard to reach agreement on the right solution
    - Hard to deploy

9 SCION

TODO: Improve this section

- It has been known for a long time that the internet could be better
- Many projects started over 25 years ago
- Weaknesses
  - DoS
  - Path hijacking
  - **Kill Switch:** Something wich can partly or completely shut down the internet
  - Faking certificates
  - Non-scalability of trust
- Architecture Principles
  - Stateless packet forwarding
    - No inconsistent forwarding state
    - Solves many problems
  - Instant convergence routing
  - Path-aware networking
  - Multi-path communication
  - High security through design and formal verification
  - Sovereignty and transparency for trust roots
- Control plane
  - Structure
    - **Isolation Domain:** grouping of ASes
    - **ISD core:** ASes that manage the ISD
    - **Core AS:** AS that is part of ISD core
    - Control plan for intra and inter ISD is organized hierarchically
    - **Trust Rooter Configuration (TRC):** Elected router for each ISD which verify ISD operations
      - Defined by each IDS
    - Allows for scalability
    - **Path Servers:**
      - Run by core ASes and normal ASes
      - One or multiple per AS
      - **Core AS**
        - Consistent, replicated store of down-path segments and core-path segments
      - **Normal AS**
        - Serves up-path segments to local clients
        - Resolved and caches response of remote AS lookup TODO: What is this?
    - **Beacon Server**
      - Run by core ASes and normal ASes
      - One or multiple per AS
      - Forward PCBs to downstream ASes
  - Intra-ISD Path Exploration
    - **Path-segment Construction Beacons (PCBs):** Build up communication path to communicate with the core ASes
      - Contains a field with its creation time
      - Send by core ASes
      - Send via SCION service anycast packets
      - Traverse ISD as a flood to reach downstream ASes

        - Forwarded to downstream ASes by the beacon servers of ASes
        - Each AS on paths adds:
            - AS name
            - Hop fields
                - Link identifiers
                - Expiration time
                - Message Authentication Code (MAC)

              used for data-plane forwarding
            - AS signature
        - **Up-Path Segment:** From AS to core AS
        - **Down-Path Segment:** From core AS to AS
    - Inter-ISD Path Exploration
        - Only between core ASes of ISDs
    - **Up-Path Segment Registration**
        - AS selects path segment to announce as up-path segments to all local hosts
        - Registered at local path serversAS select path segment to announce as up-path segments to all local hosts
        - Registered at local path servers
    - **Down-Path Segment Registration**
        - AS selects path segment to announce as down-path segments to all other which use the AS
        - Down-path segments are uploaded to core path server in core AS
- Data Plane
    - **Path Lookup**
        - Host obtains path segment
            - Host contacts RAINS server with a name
            - Host contacts local path server to query path segments
            - Host combines path segments to obtain end-to-end paths
        - Local ISD
            - Client request path segment to `<ISD, AS>` from local path server
            - Local path server sends request to core path server
                - Only if down-path segments are not locally cached
            - Local path server replies
                - Up-path segment to local ISD core ASes
                - Down-path segment to `<ISD, AS>`
                - Core-path segments are needed to connect to up-path and down-path segments
        - Remote ISD
            - Host request path segment to `<ISD, AS>` from local path server
            - Local path server contacts core path server
                - Only if not cached
            - Core path server contacts remote core path server
                - Only if not cached
            - Host receives up-segment, down-segment and core-segment
        - Path combination
            - After the path is evaluated, ASes may take a number of shortcuts (e.g. peering links)
        - Path segments compared to current internet
            - **Up-Path Segment:** Traverse lower-tier ISPs to reach tier-1 ISP

        - **Core-Path Segment:** Tier-1 ISPs path close to destination
        - **Down-Path Segment:** Traverse lower-tier ISPs to reach destination
        - **Peering Links:** Same
    - Packet
        - Header
            - Version
            - Destination and Source access types
            - Total packet and header length
            - Point to current info and hop fiel
            - Next header type field
            - Source address
            - Destination address
            - Info field (info about path segment)
                - Several flags
                - Timestamp
                    - Contains the creation time
                - ISD identifier
                - Path segment length
                - Consists of one or multiple hop fields
            - Hop field
                - Several flags
                - Expiration time
                    - Relative to timestamp in info field
                - Ingress and egress interface identifiers
                - Message Authentication Code (MAC)
        - Path
            - TODO: Add path encoding
        - Ingress and Egress Interface Identifiers
            - Each AS assigns a unique integer identifier to each interface that connects to a neighbouring AS
            - Interface identifies identity ingress/egress links for traversing AS
            - ASes use internal routing protocol to find route from ingress SCION border router to egress SCION border router
        - Hop field MAC Versification
            - MAC computation and verification of Hop Field MAC value is based on local AS secret
            - Computed as $\text{MAC}_{\text{Key}}(\text{Timestamp}, \text{Some Flags}, \text{ExpTime}, \text{Ingress ID}, \text{Egress ID}, \text{Hop fiel}$
            + Can quickly be computed
                - SCION forwarding is faster and used less energy and IP forwarding
- Observations
    - SCION provides a rich set of path choices
    - SCION is highly scalable
    - SCION provides much lower connection time
- Deployment
    - Core AS Deployment
        - Duties
            - Manage and distribute the ISD's TRC
            - Sign TRCs of neighbouring ISDs and endorse other ISDs
            - Maintain a list of all recognized ISDs

        - Issue certificates to all ASes in the ISD
        - Provide connectivity to neighbouring ISDs
        - Generate and disseminate inter-ISD path-segment construction beacons (core PCBs)
        - Generate and disseminate intra-ISD PCBs
        - Provide highly available service: beacon, name (RAINS), path, certificate, COLIBRI, and time server
      * Need core service and border routers
      * SCION reuses existing intra-domain networking infrastructure
    ◇ Normal AS Deployment
      * Duties
        - Provide connectivity to neighbouring ASes
        - Disseminate PCBs
        - Provide available service: beacon, name (RAINS), path, certificate, and COLIBRI server
    ◇ ISP Business Models
      * Premium link offerings
        - E.g. low earth orbit satellite links
      * Geofencing
        - I.e. traffic does not leave Switzerland
      * High availability
      * Presude-leased line
      * Low latency path
    ◇ ISP Advantages
      * Additional offerings
      * Less network management overhead
      * Increased Network capacity
      * Fine-graded traffic engineering
      * CO2 based routing and accounting
    ◇ Leaf AS Deployment
      * Steps
        - Obtain AS certificate from core AS
        - Deploy beacon, RAINS, path, certificate and COLIBRI server
    ◇ End Domain Deployment
      * Also easy
- Use Case
    ◇ IoT Protection
      * Can have hidden paths which cannot be seen by intruders
    ◇ DDOS Defense
      * Build-in DDoS defense system
        - High-speed source authentifikation
        - Multi-path communication to circumvent congested areas
        - Hidden paths to prevent flooding
        - COLIBRI as QoS system
      + Guaranteed communication during attacks
    ◇ Hinterdomdomain Failover
      * SCION has backup paths already available
    ◇ Low-Latency Connectivity
      * Can use multiple paths at the same time

- ◇ Internet Sovereignty
  - ∗ ISD are isolated
  - ∗ Transparency on which entities need to be trusted
- ◇ Low Earth Orbit Satellite Network
  - ∗ BGP is not scalable enough to handle the number of satellites

10 Appendix

10.1 List of Abbreviations

Acronym	Long Form	Explanation
ACK	acknowledgment	used in many network protocols to confirm the receipt of data
AIMD	additive increase, multiplicative decrease	algorithm used by congestion-control algorithms to reach an efficient and fair bandwidth allocation
AP	access point	device to provide WiFi connectivity
API	application programming interface	interface that defines the interactions between different applications
ARP	Address Resolution Protocol	translates between IP and MAC addresses
ARQ	automatic repeat request	strategy to handle errors in data transmission
AS	autonomous system	independent network that interconnects with others to form the Internet
BCP	best current practice	rules and recommendations for the application of protocols
BDP	bandwidth-delay product	product of bandwidth and delay; measures amount of data in flight
BEB	binary exponential backoff	approach to double waiting time after each consecutive collision
BGP	Border Gateway Protocol	today's de-facto standard inter-domain routing protocol (path-vector protocol)
CDN	content distribution (or delivery) network	distributed network of data centers and caches to deliver content to users with low latency
CRC	cyclic redundancy check	error-detection mechanism
CSMA	carrier-sense multiple access	protocol that listens for other transmissions before sending data
CSMA/CD	CSMA with collision detection	additionally to CSMA check for collisions while sending
CTS	clear to send	confirmation to send data in MACA
CWND	congestion window	maximum amount of data in flight to not overload the network
DHCP	Dynamic Host Configuration Protocol	protocol for automatically configuring IP hosts
DNS	Domain Name System	hierarchical system mapping names to IP addresses and other data
DoS	denial of service	type of attack where an attacker deteriorates service for legitimate users
DSL	digital subscriber line	technology to transmit digital data over telephone lines
DV	distance vector	one of two main approaches to intra-domain routing based on distributed Bellman-Ford
ECMP	equal-cost multi path	approach to use multiple paths simultaneously when they have equal cost
FDM	frequency-division multiplexing	one of two main types of sharing bandwidth

FN	false negative	type of error of probabilistic algorithms and data structures
FP	false positive	type of error of probabilistic algorithms and data structures
Gbps	gigabit per second	unit to measure network bandwidth (see also bps, kbps, Mbps)
HTML	HyperText Markup Language	markup language for documents, primarily used for websites
HTTP	Hypertext Transfer Protocol	application-layer protocol used (among others) for web browsing
ICMP	Internet Control Message Protocol	protocol to provide error and diagnostic information for IP
IETF	Internet Engineering Task Force	standards organization which develops and Internet standards
IGP	interior gateway protocol	intra-domain routing protocol
IP	Internet Protocol	today's de-facto standard network-layer protocol
ISM	industrial, scientific, and medical	unlicensed part of the wireless frequency spectrum
ISP	Internet service provider	company that provides internet connectivity
IS-IS	Intermediate System to Intermediate System	one of the two most common intra-domain routing protocols (link-state protocol)
IXP	Internet exchange point	location where many ASes interconnect
LAN	local area network	small network within a household or organization
LDPC	low-density parity-check	type of error-correcting code
LP	linear program(ming)	approach for optimization problems with linear constraints and objectives
LS	link state	one of two main approaches to intra-domain routing based on flooding and Dijkstra
LSP	link-state packet	local topology information flooded in a LS routing protocol
MAC	media access control	part of the link layer; often used in "MAC address" for link-layer addresses
MAC	message-authentication code	tag to authenticate data based on symmetric cryptography (not very relevant for this lecture)
MACA	Multiple Access with Collision Avoidance	protocol for wireless communication to mitigate issues with exposed and hidden terminals
MitM	man in the middle	type of attack where an attacker inserts themselves into the communication of two parties
MPLS	Multiprotocol Label Switching	virtual-circuit protocol between link and network layer
MSS	maximum segment size	maximum size of a transport-layer payload
MTU	maximum transmission unit	maximum size of a network-layer packet
NAT	network address translation	system to replace IP address and port of packets at the network layer
NDP	Neighbor Discovery Protocol	protocol that provides ARP and DHCP functionality for IPv6

NIC	network interface card	hardware component that connects a computer to the network
OFDM	orthogonal frequency-division multiplexing	type of FDM to carry data in parallel
OSPF	Open Shortest Path First	one of the two most common intra-domain routing protocols (link-state protocol)
PPP	Point-to-Point Protocol	link-layer framing protocol
PRG	pseudo-random generator	symmetric cryptography primitive to generate a stream of pseudorandom bits (aka stream cipher)
PRP	pseudo-random permutation	symmetric cryptography primitive to permute blocks of data (aka block cipher)
QUIC	(originally) Quick UDP Internet Connections	new secure and reliable transport-layer protocol built on UDP
RFC	request for comment	standards document, mainly published by IETF
RIP	Routing Information Protocol	routing protocol in the early Internet
ROA	route origin authorization	certificate within RPKI to originate a certain IP prefix
RPKI	Resource Public Key Infrastructure	hierarchy of resource certificates to mitigate BGP hijacks
RTS	request to send	request to send data in MACA
RTT	round-trip time	two-way latency of a connection
RWND	receiving window	maximum amount of data in flight to not overload the receiver
SNR	signal-to-noise ratio	important measure of communication channels
TCAM	ternary content-addressable memory	specialized hardware for longest-prefix matching
TCP	Transmission Control Protocol	transport-layer protocol providing reliable stream transport
TDM	time-division multiplexing	one of two main types of sharing bandwidth
TLD	top level domain	highest DNS hierarchy
TLS	transport-layer security	security protocol building on TCP (not relevant for this lecture)
TTL	time to live	used in several protocols to define the validity of data or packets
UDP	User Datagram Protocol	transport-layer protocol providing unreliable datagram transport
URL	Uniform Resource Locator	reference to a web resource
VC	virtual circuit	one of two main networking models
VoIP	voice over IP	technologies for voice communication over the Internet

The following is not really exam relevant

10.2 Basic Tools

- **ping**
  - ◇ `ping some_address`
  - ◇ Check connectivity and measure RTT
  - ◇ `-c some_num` limits the number of pings

  - ◇ Sends ARP packets
- **dig**
  - ◇ `dig some_domain`
  - ◇ Queries DNS
  - ◇ The anders is in the ‘ANSWER SECTION‘ on the right
  - ◇ The ‘SERVER‘ at the bottom is who answered (DNS server address I guess)
- **ip**
  - ◇ `ip addr`
  - ◇ Gives Local IP information
- **iftop**
  - ◇ `iftop -i some_interface`
  - ◇ Displays all traffic over the given interface
  - ◇ `Tx:` stands for send
  - ◇ `Rx:` stands for receive
  - ◇ Columns on the right display average over $2, 10, 40$ seconds
- **Wireshark**
  - ◇ Monitor network
  - ◇ Has UI
- **tcpdump**
  - ◇ Similar to wireshark
  - ◇ Cli application
  - ◇ **-w:** write output to file
  - ◇ **-c N:** stop after N packages
  - ◇ **-i some_interface:** listen on specified interface
  - ◇ **-n:** do not convert names to ip
- **netcat**
  - ◇ Allows for quick communication between two hosts
  - ◇ Send text
    - ∗ Server: `netcat -l 4444`
    - ∗ Client: `netcat localhost 4444`
  - ◇ Send file
    - ∗ Server: `netcat -l 4444 > received.txt`
    - ∗ Client: `netcat -l 4444 < send.txt`
- **ss**
  - ◇ Displays sockets
  - ◇ `ss -at`: show TCP services
  - ◇ `ss -au`: show UDP services
  - ◇ `-n`: show port instead of service name
- **lsof**
  - ◇ Show used ports
  - ◇ `lsof -Pn -i4`

10.3 DNS Stuff

- Figure out where data comes from
  - ◇ I.e. if from content delivery or directly
  - ◇ `curl -Ls -o /dev/null -w %{url_effective} some_resource`
- Get IP we are actually talking to
  - ◇ `sudo hping3 -c 1 -S --tcp-mss 1460 --tcp-timestamp -L 0 -w 1024 -s 39826 -p 443`
- Reverse DNS Query

    - ◇ `dig +noall +answer -x some_ip`
- Reverse DNS Query
    - ◇ `whois some_ip`

10.4 Congestion Control

- `/lib/modules/$(uname -r)/kernel/net/ip4` contains all available congestion control algorithms
- `/proc/sys/net/ipv4/tcp_congestion_control` contains the current algorithm
- `tcp_bbr` is a good algorithm
- `iperf` or `iperf3` can be used to speed measurements

TODO: Mention throughput vs goodput