Telekommunikációs hálózatok

Előadás és gyakorlat jegyzet

Készült Laki Sándor előadásai és gyakorlatai,
valamint Gombos Gergő gyakorlatai alapján

Sárközi Gergő, 2022-23-1. félév
Nincsen lektorálva!

Tartalomjegyzék

1. Bevezetés

1.1. Alapfogalmak

- Végpont
  - (Hálózati) hoszt: eszköz, amely hálózattal össze van kötve
  - Szolgáltatásokat biztosíthat, információt oszt meg, stb.
- Switch, router
- Link (pl. WiFi, optikai szál, coax, RJ-45)
  - Átviteli csatorna, médium, közeg: amelyen kommunikáció folyik
- Hálózati sávszélesség: elérhető/felhasznált kommunikációs erőforrás
  - Mértékegység: bit per second, SI váltószám: 10^3 (IEC: 2^{10})
- Internet: hálózatok hálózata
- Hálózati folyam, flow: két fél közötti hálózati forgalom
  - Más definíció: valamilyen szempontból összetartozó forgalom

1.2. Topológiák

- Teljesen összekötött (Full-mesh)
- Lánc/gyűrű
- Busz: egy speciális link, ez ágazik le mindenhova

1.3. Pár internet szolgáltatás technológia

1.3.1. Digital Subscriber Line (DSL)

- Telefon vonalon keresztül (nagy sávszélességű) internet háztartásba
- 3 csatorna
  - Letöltés: néhány 100 Mbps
  - Feltöltés: néhány 10 Mbps
  - Hang: 2 irány
- Kapcsolatunk direkt egy nagyobb egységig (nincs megosztva)

1.3.2. Cable Access Technology (CATV)

- Koax TV kábellel (nagy sávszélességű) internet
- 2 csatorna
  - Letöltés: néhány 100 Mbps
  - Feltöltés: néhány 10 Mbps
- Közeg meg van osztva háztartások között

1.3.3. Ethernet

- Leggyakoribb LAN (Local Area Network, helyi hálózat) technológia
- Kábel: UTP+RJ45 vagy optikai+SFP
- Sebesség: $<$1 Mbps-től $>$1 Gbps-ig (szimmetrikus, full-duplex)
- Hálózati kártyába csatlakozik (NIC, Network Interface Controller)

1.3.4. Egyéb

- Telefon (pl. 4G)
- Műholdas
- Stb.

1.4. Internet belseje

- Követelmények
  - Hibatolerancia: több útvonal források és célok között
  - Rugalmasság: linkek száma nem lehet túl nagy, meg kell osztani az erőforrásokat (költséghatékonyság és megvalósíthatóság érdekében)
  - Megfelelő csomópont kapacitás: linkek száma nem lehet túl kicsi
- Kompromisszumos megoldás: switchelt/kapcsolt, csomagkapcsolt hálózatok
  - Előny: erőforrás megosztás van és csomópontok kapacitása igény szerint alakítható
  - Hátrány: a switch-nek okosnak kell lennie (csomagtovábbítás, forgalomirányítás, erőforrás kiosztás)
  - Linkek, switch-ek meg vannak osztva: egyszerre több flow is használja
- Rendszerfüggetlenség, nincs központi felügyelet
- Építőeleme a LAN

1.4.1. ISP (Internet Service Provider, Internet szolgáltató)

- Hierarchiában vannak
  - Tier-1 (interkontinentális), Tier-2 (nemzeti), Tier-3 (felhasználó)
  - Tier-1-nek nincsen szolgáltatója, ő csak szolgáltat
- Peering link: ingyen van az adatátvitel rajta
  - Azonos tierek között
  - Nem használható akármire: a link egyik végén kell keletkeznie, a másikban terminálnia a forgalomnak
- Transit link: fizetni kell a használt adatmennyiség után
  - pl. Tier-3 és Tier-2 között a Tier-2 fizet (szolgáltató-vásárló viszony)
- Internet eXchange Point (IXP): sok ISP összekötése
  - Költséghatékony: egyszerre sok ISP találkozik, nem csak 1-1

1.5. Erőforrás kezelés

1.5.1. Előre foglalás

- Folyam szintű multiplexálás
- Előre lefoglal sávszélesség, amin aztán valamikor küld adatot
- Több kiosztási módszer:
  - First-come first-served: első lefoglaló lefoglalhatja akár az egészet
  - Egyenlő elosztás: link sávszélessége egyenletesen el van osztva
- Kihasználtság kiszámolása:
  - Legyen P a csúcsráta (peak rate) és A az átlagos ráta
  - Kihasználtsági szint: A/P (pl. $P = 100$ és $A = 10$ esetén 10%)
  - Előre foglalás akkor jó választás, ha P/A kicsi
    - Hang esetén általában ez az arány 3 vagy kevesebb
- Megvalósítás: áramkörkapcsolt hálózat
  - Használja: vezetékes telefon hálózatok
- Előnyök:
  - Egyszerű elveken alapuló gyors kapcsolás
  - Kiszámítható teljesítmény
- Hátrány:
  - Hibatolerancia alacsony
  - Extra késleltetés: áramkör felépítése
  - Alacsony hatékonyság (sávszélesség pazarló)

1.5.2. Resource Reservation Protocol

- Áramkörkapcsolt hálózatok esetén egy erőforrás foglaló protokoll
- 1.: forrás foglalási kérést küld $XMbps$ igényről a célállomásnak
- 2.: switchek kialakítják az "áramkört"
- 3.: forrás megkezdi az adatküldést
- 4.: forrás áramkör-lebontó üzenetet küld a célállomás felé (teardown)

1.5.3. Igény szerinti

- Csomag szintű multiplexálás
- Foglalás nélkül akkor küld adatot, amikor szükséges
- Előre foglaláshoz képest általában nagyobb kihasználtság érhető el
  - Különbség függ: források számától, folyam börsztösségtől
  - Kommunikáció általában löketes (börsztös), P/A jellemzően 100+
- Megvalósítás: csomagkapcsolt hálózat
  - Használja: internet
  - Minden csomag tartalmazza a cél címét/azonosítóját
- Előnyök:
  - Hatékony erőforrás gazdálkodás
  - Egyszerű hálózatot készíteni, bővíteni
  - Hiba észlelése és kijavítása gyors, magas hibatolerancia
- Hátrányok:
  - Kiszámíthatatlan teljesítmény
  - Ha túl sok csomag jön egyszerre, valami el lesz dobva
  - Bonyolult hardver kell: puffer-kezelés, torlódás-vezérlés

1.6. Internet rétegmodelljei

- Internet célja: különböző eszközökön különböző folyamatok tudjanak kommunikálni egymással
  - Továbbá legyen hatékony, gyors, stb.
- Protokoll és API kapcsolatban van
- Sok a protokoll, stb.; megoldás: modularizáció, rétegződés

1.6.1. TCP/IP modell

- 4 réteg
- Link layer (kapcsolati réteg): 2 szomszédos pont közötti kommunikáció
  - Hardver közeli, itt még nem teljesen protokollokról beszélünk
  - Vastag réteg, sok minden történik itt (hibajavítás, stb.)
  - Pl. Ethernet
- Internet layer (hálózati réteg): routing, 2 bármely pont közötti kommunikáció
  - Globális kommunikáció miatt nincs sokféle protokoll
  - Pl. IPv4, IPv6
- Transport layer (szállítói réteg): processzek közötti kommunikáció
  - (De)multiplexálás (packet fragmentation and reassembly)
  - Alkalmazás követelmények biztosítása (pl. megbízhatóság)
  - Pl. TCP, UDP
- Application layer (alkalmazási réteg): sztenderd alkalmazások ezt használják
  - Pl. socket api-n keresztül telnet szolgáltatás nyújtása

1.6.2. ISO OSI referencia modell

- Open System Interconnection Reference Model, 7 réteg
- Minden kommunikációs rendszert le tud írni (nem csak internetet)
- Rétegek jellemzése
  - Szolgáltatás: mit csinál a réteg
  - Interfész: hogyan férünk hozzá a réteghez?
  - Protokoll: hogyan implementáljuk a réteget?
- Fizikai réteg (physical layer):
  - Szolgáltatás: két fizikailag összekötött eszköz között adatátvitel
  - Interfész: specifikálja egy bit átvitelét
  - Protokoll: bit kódolási séma, feszültségi szintek, jelek időzítése
  - Példa: koaxiális kábel, optikai kábel
- Adatkapcsolati réteg (data link layer):
  - Szolgáltatás:
    - Adat keretekre tördelése (pl. hibajavítás miatt)
    - Közeghozzáférés vezérlés (MAC) (pl. WiFi esetén)
    - Per-hop megbízhatóság és folyamvezérlés (pl. pause frame)
  - Interfész: keret küldése két közös médiumra kötött eszköz között
  - Protokoll: fizikai címzés, pl. MAC cím
  - Példa: Ethernet, WiFi
  - Sokszor összeforr a fizikai réteggel, általában nem függetlenek
- Hálózati réteg (network layer): mint a TCP modellben
  - Szolgáltatás: csomagtovábbítás, útvonalválasztás, fragmentálás
    - Csomag ütemezés (priority queueing), puffer kezelés
  - Interfész: csomag küldése egy adott végpontnak
  - Protokoll: routing táblák karbantartása, globálisan egyedi címek
  - Példák: IPv4, IPv6

- Szállítói réteg (transport layer): mint a TCP modellben
  - Szolgáltatás: multiplexálás, megbízhatóság, sorrendhelyesség
    - Torlódásvezérlés: enélkül újraküldések miatt telítődne a puffer
  - Interfész: üzenet küldése cél processznek
  - Protokoll: port szám, folyamfelügyelet, megbízhatóság
  - Példa: UDP, TCP
- Munkamenet réteg (session layer):
  - Ilyen nincsen az internetben, alkalmazásnak kell csinálni
  - Szolgáltatás: kapcsolat menedzsment, szinkronizációs pontok
  - Interfész: attól függ
  - Protokoll: token menedzsment, szinkronizációs checkpoint
- Megjelenítési réteg (presentation layer):
  - Ilyen nincsen az internetben, alkalmazásnak kell csinálni
  - Szolgáltatás: adatkonverzió különböző reprezentációk között
  - pl. ascii-unicode vagy endianness (little-big)
  - Interfész: attól függ
  - Protokoll: adatformátum és transzformációs szabályok
- Alkalmazási réteg (application layer): bármi

1.6.3. Hibrid TCP/IP modell

- 5 réteg: OSI mínusz session, presentation (a 2 nem implementált)
- Mi ezt használjuk, Tanenbaum és mások találták ki
- L1 physical: fizikai bit továbbítás (twisted pair, fiber)
- L2 link: lokális best-effort továbbítás (Ethernet, Wifi)
- L3 network: globális best-effort továbbítás (IP)
- L4 transport: végpontok (processzek) közötti továbbítás (TCP, UDP)
- L5 application: magas szintű hálózat hozzáférés (HTTP, FTP)

1.7. Rétegek alkalmazása

- IP réteg a legszűkebb: ezen a ponton egységes protokoll kell globalitáshoz
- Rétegek adategysége (protocol data unit, PDU):
  - L1 physical: bit
  - L2 data link: frame (keret)
  - L3 network: packet (csomag)
  - L4 transport: segment (szegmens)
  - L5 application: message (üzenet)
- Első 2-3 réteget minden eszköz implementálja
  - Szállítói, alkalmazási réteg általában a végpontokban van implementálva
  - Kivétel: pl. olyan tűzfal, ami ezeket figyelembe veszi
  - Switch: első 2 réteget tudja; router: első 3 réteg
- L2 - L4 akár szoftveres, akár hardveres is lehet
  - L4 transport hardverben: smartNIC (TCP rásegítés)
  - L3 network: NIC vagy szoftveres routing tábla
  - L2 link: általában NIC (vagy switch), de pl. virtuális hálózat
  - Hardver gyorsabb, szoftver flexibilisebb, de ez a határ kezd elmosódni
    - SDN, P4 hardver oldalról; DPDK szoftver oldalról
- Rétegek egymást enkapszulálják: a réteg/protokoll fejlécét odarakjuk a többi mellé, legvégén pedig a payload van
  - Kivétel: Ethernet a csomag végére, payload után is rak
  - Switchek, routerek egyes fejléceket módosítanak, de a többit békén hagyják és csak tovább küldik

1.8. Hálózatok karakterizációja

- Három szempont:
  - Késleltetés, delay
  - Csomagvesztés, loss
  - Fogadási adatráta, throughput

1.9. Késleltetés, delay

- Total delay = transmission + propagation + processing + queueing
  - transmission, propagation: linktől függ
    - Melyik fontosabb? Helyzettül függ
    - Kicsi csomag: propogation miatt lesz lassú
    - Nagy csomag: transmission miatt lesz lassú
  - processing, queueing: switchtől, switchen adatforgalomtól függ
- Propagation delay = link length / signal propagation speed
  - Rézkábel: 2/3 fénysebesség
  - Optikai kábel: 2/3 fénysebesség (fény pattog; út nem egyenes)
  - Fénysebesség konstans, ezért csökkentésre: Content Delivery Network
    - Közelebből kelljen az információt elküldeni a célba
- Transmission delay: 1 Gbps link esetén 1 bit kiírása $10^{-9}s$
- Queueing delay: pufferben töltött idő
  - Nehéz megbecsülni, csomagonként változik
  - Mérőszámok: átlag, variancia, esély X feletti értékre
  - Függ: érkezési ráta, küldési ráta, traffik börsztösség
  - Traffik intenzitás = csomag bitméret * érkezési pps / küldési bps
    - Ha ez nagyobb mint 1, akkor idővel biztosan tele lesz a puffer
  - Delay felső korlát: puffer méret * csomag bitméret * küldési bps

1.10. Loss (csomageldobás, csomageldoboódás)

- Ha teli a puffer és jön egy újabb packet, akkor valamit el kell dobni
  - El lehet dobni az új csomagot
  - El lehet dobni a puffer legrégebbi csomagját
  - Létezik priority queueing (QoS, quality of service)
- Miért lehet teli a puffer?
  - Pillanatnyilag nagyon nagy börszt van
  - A traffik intenzitás értéke 1-nél több (több adat jön, mint megy)

1.11. Throughput, (lehetséges maximum) sávszélesség

- Throughput (bps) = adat méret (bit) / átviteli idő (s)
  - Több linkból álló lánc esetén a legkisebb értékű link számít
  - Optimálisan több TCP stream: sávszélesség egyenletesen eloszlik
  - TCP: megtalálja a bottleneck-et (nagy fluktuációval)

2. Fizikai réteg (physical layer)

2.1. Áttekintés

- Szolgáltatás: két fizikailag összekötött eszköz között adatátvitel
- Interfész: specifikálja egy bit átvitelét
- Protokoll: bit kódolási séma, feszültségi szintek, jelek időzítése
- Példa: koaxiális kábel, optikai kábel

2.2. Elméleti alapok

- Küldeni könnyű, de a fogadó félnek torz jelet kell tudni értelmezni
- Jel elnyelődés, signal attenuation
  - Fogadó túl kevés energiájú, túl tompa jelet kap
  - Frekvenciafüggő: más frekvenciák máshogy nyelődnek el
    - A jelünk több frekvenciából áll, ez ezért probléma
  - Távolság növelésével nő az elnyelődés
  - Megoldás: vannak frekvencia sávok közelítőleg egyenletes elnyelődéssel
    - Ezek a sávok a közegtől függenek
  - Számolása: $\alpha\ [dB] = 10dB \times \log_{10} \frac{\text{küldési energia}}{\text{vételi energia}}$
- Fáziseltolódás: más frekvenciájú a hullámok terjedési sebessége eltér
- Zaj: hő, más rendszerek (mikrohullámú sütő)
- Adatátvitel valamilyen fizikai jellemző (pl. feszültség, áramerősség) változtatásával lehetséges
  - Különböző frekvenciájú szinuszhullámok végtelen sörösszegével mindenféle periodikus jel felírható

2.3. Szimbólumok, bitek

- Bitek helyett szimbólumokat küldünk
- Azonos küldési rátával magasabb sávszélesség érhető el
  - Mert 2 helyett több jelszint (pl. 4) használható
  - Olyan fogadó kell, ami meg tudja különböztetni a több jelszintet
- Két mérőszám: adat ráta (bps) és szimbólum ráta (Baud, symbols/sec)

2.4. Átviteli közegek

2.4.1. Vezetékes

- Egy kamion tele HDD-vel (azaz mágneses adathordozók)
  - Magas sávszélesség, nagy késleltetés
- Sodort érpár (twisted pair), pl. UTP
  - Analóg és digitális jel átvitelére is alkalmas
- Koaxiális kábel: sodort érpárhoz képest nagyobb sebesség és távolság
  - Sodrás helyett egyenes kábel + fonott külső "Faraday kalitka"
  - Kétharmad fénysebesség van a rézszál közegben
- Optikai szál (kétharmad fénysebességre képes pattogás miatt)

2.4.2. Vezeték nélküli

- Elméleti alapok, $\lambda f = c$
  - Frekvencia (f, Hz): EM hullám másodpercenkénti rezgésszáma
  - Hullámhossz (λ, m): két egymást követő hullámcsúcs távolsága
  - Fénysebesség ($c = 3 * 10^8 \frac{m}{s}$): EM terjedési sebessége vákuumban
- Rádiófrekvenciás átvitel
  - Egyszerű, használható bel- és kültéren, frekvenciafüggő terjedés
  - Alacsony frekvencia: nagy távolság, kis sávszélesség
  - Alacsony frekvenciás jel visszapattan az ionoszféráról, így a föld görbületét "meg lehet oldani"
- Mikrohullámú átvitel
  - Egyenes vonal mentén terjed, elnyelődés egy probléma, olcsó
- Infravörös és milliméteres hullámú átvitel
  - Kis távolságra képes, szilárd testeken nem hatol át
- Látható fényhullámú átvitel (lézerforrás és fényérzékelő)
  - Olcsó, nagy sávszélesség, nem engedélyköteles
  - Időjárás erősen befolyásolja
- Kommunikációs műholdas átvitel: annyira nem fontos
  - Magas pálya: nagy területet képes kiszolgálni, de nagyobb a késleltetés
  - Starlink: alacsony pálya, minimális késleltetés, rengeteg műhold

2.5. Adatátvitel

2.5.1. Kiinduló feltételek

- Két jelszint van: magas (1) és alacsony (0)
- Szinkron átvitel: egy óra vezérli a mintavételezést
- Jel amplitúdója és időbeli kiterjedése számít

2.5.2. Non-Return to Zero (NRZ) kódolás

- 1-es bit esetén magas jel, 0-as bit esetén alacsony jel
- Előny: hatékony
- Probléma: hosszú azonos jelszint esetén elveszik az adó és a vevő közötti szinkronizáció (hogy mikor kell mintavételezni)
  - Ha egy kicsit el tud csúszni az óra, akkor előbb utóbb el tud csúszni annyira, hogy egy teljes bit kimaradjon
- Probléma: "nincs jel" és "sok nulla jön" nem nagyon tér el egymástól

2.5.3. Szinkronizáció lehetőségek

- Explicit órajel küldése egy párhuzamos átviteli csatornán
- Kritikus időpontokban szinkronizáció (pl. szimbólum/blokk kezdetén)
- Szimbólum kódok: szignál tartalmazza szinkronizáláshoz szükséges infót
  - Önütemező jel, azaz külön órajel nélkül dekódolható

2.5.4. Manchester kódolás (10 Mbps Ethernet)

- 1-es bit: átmenet magasról alacsonyra
- 0-ás bit: átmenet alacsonyról magasra
- Probléma: 1 bithez 2 óraidő ciklus szükséges (akár 2 átmenet kell)

2.5.5. Non-Return to Zero Inverted (NRZI) kódolás

- 1-es bit esetén átmenet, 0-as bit esetén nincs változás
- Előny: hatékony
- Probléma: csupa 0 bit sorozat esetén ugyanúgy elveszhet a szinkronizáció

2.5.6. 4-bit/5-bit kódolás NRZI előtt (100 Mbps Ethernet)

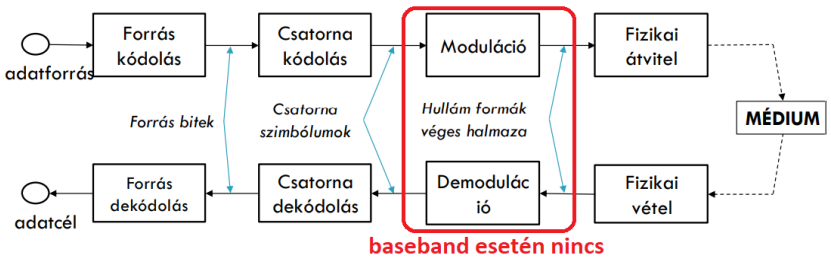
- Minden 4 hosszú bitsorozat 5 hosszúra kódolása
  - Nem lehet 1-nél több 0 az elején
  - Nem lehet 2-nél több 0 a végén
  - kódolás megoldható egy $16 = 2^4$ elemű táblázattal
- Hátrány: 80%-os hatékonyság NRZ-hez képest
- Előny: nincs többé szinkronizáció elvesztés

2.5.7. 8-bit/10-bit kódolás NRZI előtt (Gigabit Ethernet)

- Ugyan az, mint a 4-bit/5-bit, csak más mennyiségű bitet map-el

2.6. Jelátvitel

- Digitális esetben véges halmazzal dolgozunk, így van lehetőségünk hibajavításra

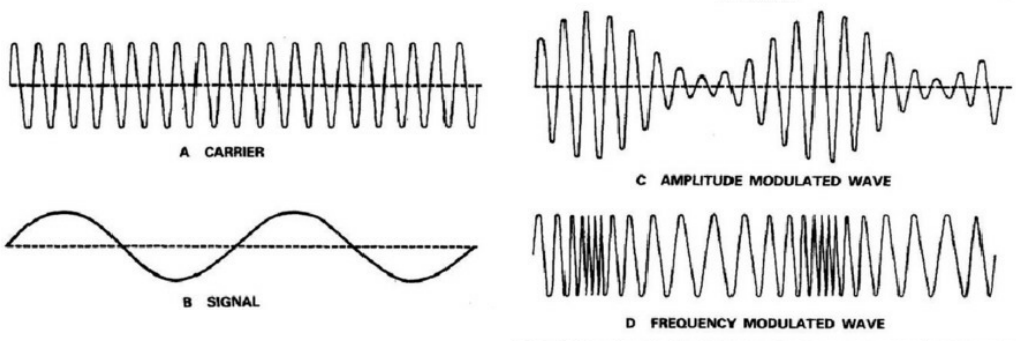


2.6.1. Alapsáv, baseband (korlátos technológia)

- Digitális jel árammá vagy feszültséggé alakul
- Jel minden frekvencián a sávon belül átvitelre kerül

2.6.2. Szélessáv, broadband

- Széles frekvenciatartományban történik az átvitel
- Egy vivőhullám mindig ott van a közegen, ezt változtatjuk meg
- Modularizáció, vivőhullámba a digitális jel kódolásának lehetőségei:
  - Amplitúdó Moduláció (AM): adatok vivőhullámra ültetése
  - Frekvencia (FM) vagy fázis moduláció: vivőhullám változtatása
  - Különböző vivőhullámok egyidejűleg felhasználása



2.6.3. Amplitúdó Moduláció (AM)

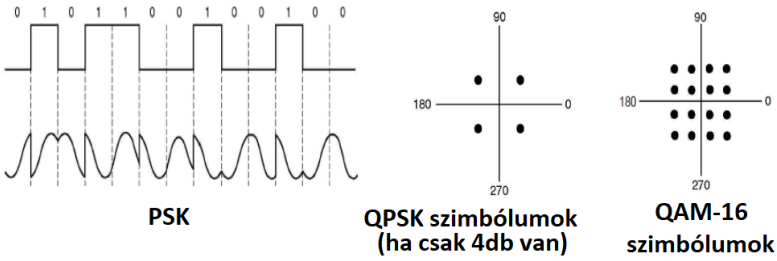
- Digitális szignál: amplitúdó-eltolás keying (szimbólumonként más amplitúdó)
- $f_A(t) = s(t) * \sin(2\pi f t + \varphi)$

2.6.4. Frekvencia Moduláció (FM)

- Digitális: frekvencia-eltolás keying (szimbólumonként különböző frekvencia)
- $f_F(t) = \alpha * \sin(2\pi s(t) t + \varphi)$

2.6.5. Fázis moduláció

- Analóg esetben nem működik túl jól
- $f_P(t) = a * \sin(2\pi ft + s(t))$
- Digitális szignál: fázis-eltolás keying (PSK)
- Quadrature Phase Shift Keying (QPSK)
  - Diszkrét halmaz kódolja a szimbólumokat
  - 4 szimbólum $\implies$ kétszeres adatráta a szimbólumrátához képest



PSK | QPSK szimbólumok (ha csak 4db van) | QAM-16 szimbólumok

2.6.6. Quadrature Amplitude Modulation (QAM)

- Kombinálja az amplitúdó és a fázis modulációt
- Diszkrét halmaz kódolja a szimbólumokat
  - 16 szimbólum $\implies$ négyszeres adatráta szimbólumrátához képest
- Máshogy felfogva: két merőleges (90-fokkal eltolt), azonos frekvenciájú hullámot amplitúdó modulálunk, ezzel koordinátákat írva le

2.7. Multiplexálás

- Statikus csatorna hozzáférés módszereket nézünk (statikus allokáció)
- Egy fizikai közegen több jel párhuzamosan utazik
  - Fizikai csatornát logikai alcsatornákra bontjuk
- Küldő oldalon kell speciális eszköz (multiplexer), ami a jelet a megfelelő alcsatornára helyezi

2.7.1. Térbeli multiplexálás (space-division multiplexing)

- Minden csatornához külön fizika eszköz
  - Vezetékes átvitel: külön pont-pont vezeték
  - Vezeték nélküli átvitel: külön antenna
- Celluláris hálózatokra jellemző

2.7.2. Frekvencia multiplexálás (frequency-division multiplexing)

- Szignálonként más frekvencia, közöttük "guardband"
- Analóg esetben szokás alkalmazni
- Többféle megvalósítás létezik:
  - Direct Sequence Spread Spectrum: szignálon XOR véletlen bitsorozattal
  - Frequency Hopping Spread Spectrum: pszeudo véletlen alapú választás sok frekvencia közül (mindkét fél ismeri a pseudo véletlen seed-jét)

2.7.3. Hullámhossz multiplexálás (wavelength-division multiplexing)

- Gyakorlatilag frekvencia multiplexálás, csak optikai szálon

2.7.4. Időbeli multiplexálás (time-division multiplexing, TDM)

- Minden állomás kap egy diszkrét időszeletet (slot)
- Koordináció vagy merev felosztás szükséges

2.7.5. Code Division Multiple Access (CDMA)

- 3G-s mobilhálózatok alapja, mobil kommunikációban elterjedt
- 1 bitnyi információ helyett m bitnyi információt küldünk
- Minden állomás sugározhat akár folyamatosan, párhuzamosan
- Minden állomáshoz tartozik egy töredéksorozat (chip sequence), ami egy m bites kód
  - Ezek a kódok ortogonális bázist alkotnak
- Egy állomás mit küldjön, ha egy adott bitet akar továbbítani?
  - 1-es bit küldése esetén az állomás elküldi a kódját
  - 0-ás bit küldése esetén a kódjának az egyes komplemensét
- Ortogonális bázis miatt az interferált jelből csak projekciót kell végezni

3. Adatkapcsolati réteg (data link layer)

3.1. Áttekintés

- Szolgáltatás:
  - Adat keretekre tördelése (pl. hibajavítás miatt)
  - Közeghozzáférés vezérlés (MAC) (pl. WiFi esetén)
  - Per-hop megbízhatóság és folyamvezérlés (pl. pause frame)
- Interfész: keret küldése két közös médiumra kötött eszköz között
- Protokoll: fizikai címzés, pl. MAC cím
- Példa: Ethernet, WiFi
- Sokszor összeforr a fizikai réteggel, általában nem függetlenek

3.2. Keret képzés, keretezés, framing

- Csomag-kapcsolt hálózat: csomagok routing információt is tartalmaznak
  - Azaz van fejléc, azaz ismerni kell az adathatárokat
- Nincs hibamentesség $\implies$ kell hiba felismerés és/vagy javítás
  - Nagy keret méret $\implies$ hiba valószínűbb
- Megbízható időzítésre nincs nagyon lehetőség

3.2.1. Bájt alapú: karakterszámlálás

- Fejléc tárolja a keretben lévő karakterek számát
- Probléma: ha a karakterszámban történik hiba, akkor az egész szétcsúszik

3.2.2. Bájt alapú: bájt beszúrás (byte stuffing)

- FLAG speciális bájt (jelölő bájt) jelzi a keret elejét és végét
  - ESC speciális bájt-tal escape-elni kell az adaton belül
  - ESC-et is ESC-elni kell
- Legrosszabb esetben kb. megkétszereződik az hossz
- Használat: pont-pont alapú protokollok, pl. modem, DSL, cellular

3.2.3. Bit alapú: bit beszúrás (bit stuffing)

- Minden keret speciális bitmintával kezdődik és végződik: 0**1111110**
- Minden 5db 1-es bit küldése után egy 0-ás bit lesz beszúrva
- Fogadásnál 11111 bitsorozat kezelése:
  - Következő bit 0: ez a 0 figyelmen kívül hagyása (beszúrás eredménye)
  - Következő bitek 1 majd 0: keret vége
  - Következő bitek 1 majd 1: keret eldobása, mert ez nem történhetne
    - Ez attól még nem egy hibadetektáló módszernek számít!
- Legrosszabb esetben 20%-os hossz növekedés
- Egyes hardverekben nehezebb megcsinálni a bit műveleteket

3.2.4. Óra alapú keretezés: SONET (Synchronous Optical NETwork)

- Optikai kábelen gyors átvitelt valósít még
- STS-n a nevük, most ezt nézzük: STS-1, 51.84 Mbps
- Minden keret pontosan $125\mu s$ hosszan tart, $9 * 90 = 810$ bájtosak
  - Van keret kezdő mintázat
- Fizika részhez tartozik:
  - NRZ kódolás van használva
  - Speciális 127 bites mintával van XOR-olva az adat, hogy kevés hosszú 0 és 1 sorozat legyen

3.3. Hibafelügyelet

- Zajforrások: kábelek között interferencia, rádiós áthallás, mikrosütő
- Hibák általában börsztösek (mert az interferencia általában ideig tart)
  - Neve: csoportos hiba (burst error)
  - m paraméter: védelmi övezet (guard band)
    - Egy csoporton belül két hibás bit között max $m-1$ helyes bit
    - Ellenkező esetben nem egy, hanem két csoportról beszélünk
  - Másik lehetőség: egyszerű bithiba (1 bit)
- Redundancia nélkül nincs lehetőség hibajavításra, hibadetektálásra

3.3.1. Hiba vezérlés

- Stratégiák
  - Hiba javítás (hiba javító kódok, forward error correction)
    - Akkor is működik, ha szinte minden keret hibás
    - Nagyobb az overhead
    - Hibajavítás mértéke változtatható pl. távolság függvényében
  - Hiba detektálás és újraküldés
    - ARQ: Automatic Repeat Request
    - Megbízható csatornán hatékonyabb
    - Gyakorlatban elterjedt
    - Semmire nem jó, ha szinte minden keret hibás
- Hiba javítás detektálás nélkül: pl. hangátvitel esetén lehetőség

3.3.2. Naiv hibadetektálás

- Minden keretben az adat kétszer legyen benne
  - Nem hatékony a méret növekedéshez képest
  - Kétszer akkora keret $\implies$ hibák számának várható értéke duplázódik
- Paritás bit: $+1$ bit az adathoz, hogy az 1-es bitek száma páros legyen
  - 1 bites hibák detektálhatok, de 2 bites hiba már nem
  - Börsztös hibák ellen nem megbízható

3.3.3. Redundancia

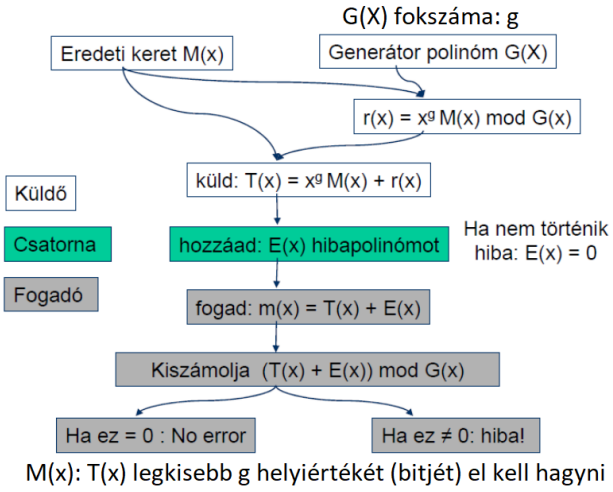
- Redundancia nélkül lehetetlen a hibafelismerés
  - Magasabb absztrakciós szinten igen, pl. HTTP szintaxis ellenőrzés
- Keret felépítése (n bit): m adat bit + r redundáns/ellenőrző bit
  - Elnevezés: n hosszú kódszó
- Hamming távolság: (ez egy metrika)
  - Kódszavak esetén: $d(x, y) =$ bitek száma, ahol x és y különbözik
  - Kódszavak halmaza: $d(S) = \min_{x,y \in S \wedge x \neq y} d(x, y)$
- $d(S) = d \implies d - 1$ hiba felismerhető, $\lfloor \frac{d-1}{2} \rfloor$ hiba javítható
  - Azaz d bit hiba felismeréséhez $d + 1$ Hamming távolság kell
  - Azaz d bit hiba javításához $2d + 1$ Hamming távolság kell
- Hamming korlát (bináris kódkönyvre):
  - Legyen $C \subseteq \{0, 1\}^n$ és $d(C) = k \in \mathbb{N}_+$
  - Ekkor $|C| * \sum_{i=0}^{\lfloor (k-1)/2 \rfloor} \binom{n}{i} \leq 2^n$
  - Jelentése: |kijavítható kódszavak| $\leq$ |lehetséges kódszavak|
  - Ha egyenlőség áll fenn: Perfekt-kód
  - Bizonyítások: polinom jegyzet, https://github.com/Trigary/uni-notes/tree/master/dimat2
- Jó kódnak a rátája és a távolsága is nagy
  - Ráta: $R_S = \frac{\log_2 |S|}{n}$ (hatékonyságot karakterizálja)
  - Távolság: $\delta = \frac{d(S)}{n}$ (hibakezelési lehetőségeket karakterizálja)

3.3.4. Hamming-kód

- 2 hatványai pozíciókban ellenőrző bitek (1,2,4,8,...)
  - A többi pozícióban vannak az adatbitek
  - Az első bit az 1-es pozícióban van
  - Ebből következik, hogy paritás bit csak a saját csoportjába tartozik
- Mindegyik ellenőrző bit paritás bit: párosra/páratlanra állítja a csoportot
  - Ez alapján even parity-ről vagy odd parity-ről beszélünk
  - Általában even parity-t használunk (legyen páros a darabszám)
- k-adik bitet mi ellenőrzi: k 2 hatványok összegeként felírva sorszámúak
  - pl. $k = 13 \implies k = 1 + 4 + 8 \implies$ 1-es, 4-es és 8-as paritás bit
  - Ez a k ez a paritás biteket is tartalmazó kódszóban a sorszám
- Példa (even parity-vel): 1000101 $\rightarrow$ **10**1**0**000**0**101
- Fogadás, dekódolás (1 bit hiba javítható)
  - Számláló 0-ról indul, k-nál nem jó a paritás bit $\implies$ k-t ad hozzá
  - Ha a számláló 0, akkor nincs detektálva hiba
  - Ha a számláló j, akkor a j-edik bitet meg kell fordítani

3.3.5. CRC kód (Cyclic Redundancy Check)

- Egy polinom kód (bitsorozat felfogható $\mathbb{Z}_2$ feletti polinomként)
- (CRC) checksum magyarul: (CRC) ellenőrző összeg
  - Ez kerül az átküldendő adat után az üzenet végére



- $G(x)$ többszöröseinek megfelelő hibákat nem ismerjük fel
- $\deg G \geq 2 \implies$ minden egybites hibát észrevesz ($E(x) = x^i$)
- Minden 2 bites hibát észrevesz, ha
  - $G(x)$ nem osztható x-szel
  - $G(x)$ nem osztható $x^k + 1$-gyel, ahol $k < \deg M(x)$
- $G(x)$ az $x+1$ többszöröse $\implies$ minden páratlan számú bithiba felismerhető
- $G(x)$ legmagasabb, legalacsonyabb fokú tagjai együtthatója legyen 1
  - Ekkor lesz a CRC működőképes, jó, hatékony
- Egy jó $G(x)$ minden max $\deg G$ hosszú csoportos hibát felismer
  - Emlékeztető: $G(x)$ többszöröseinek megfelelő hibák nem felismerhetők

3.4. Forgalomszabályozás (flow control)

- Cél: gyors adó, lassú vevő probléma ne lépjen fel (elárasztás)
- Visszacsatolás alapú forgalomszabályozás (feedback-based flow control)
  - Fogadó engedélyezi az adónak, hogy küldhet
  - Alternatíva, adatkapcsolati rétegben nem szokás:
    * Sebesség alapú forgalomszabályozás, rate-based flow control
    * Protokollba integrált sebességkorlát
- Továbbiakban: fejrészben vezérlési infók, lábrészben ellenőrző összeg
- Kommunikáció fajták:
  - Szimplex: kommunikáció egy irányba lehetséges
  - Fél-duplex: egyszerre csak egy irány, de egyébként kettő
  - Duplex: mindkét irányba szimultán kommunikáció lehetséges

3.4.1. Korlátozás nélküli szimplex protokoll

- Küldő csak küld végtelenségig, vevő csak fogad, puffer méret végtelen
- Nincs meghibásodás/csomagvesztés, nincs sorszámozás, nincs nyugta

3.4.2. Szimplex megáll-és-vár protokoll (stop-and-wait)

- Előzőhöz képest különbség: adó és vevő eltérő sebességgel működik
- Küldő küld egy keretet és vár, amíg a fogadó nyugtát nem küld
  - Tehát fél-duplex csatorna kell
- Alacsony csatorna kihasználtság (pl. magas propagation delay esetén)

3.4.3. Szimplex protokoll zajos csatornákhoz (igazi megáll-és-vár)

- Előzőhöz képest különbség: van csomagvesztés és hibásodás
- Egy keretet csak akkor adunk a hálózati rétegnek tovább, ha nem hibás
  - Hiba esetén a keret el van dobva és nincsen nyugta küldve
  - Tehát küldő oldalon kell "timeout", azután ismételt küldés
- Nyugta (ACK) vesztés esetén hálózati rétegbe duplikált keret kerül

3.4.4. Alternáló-bit protokoll (ABP)

- Duplikátum probléma megoldása: 1 bites sorszám bevezetése
- Küldő küld egy keretet és vár egy nyugtára (vagy timeout-ol)
  - Kezdő sorszám 0; nyugta megérkezése esetén megnöveli (mod 2)
  - Timeout esetén újraküldés (azonos sorszámmal)
- Vevő vár egy hiba nélküli keretet (hibás keretet eldobja)
  - Keretet továbbadja hálózati rétegnek, küld nyugtát, léptet sorszámot
- Alacsony csatorna kihasználtság: nincs megoldva
  - Csatorna kihasználtság: $\eta = \frac{T_{packet}}{T_{packet}+d+T_{ack}+d}$
    - T_{xyz}: keret kiküldéséhez szükséges idő; d: propagation delay
  - Magas propagation delay esetén η alacsony (nem hatékony)
  - Protokoll megtartásával hatékonyság növelés: keretméret növelése

3.4.5. Csúszó-ablak protokollok általános jellemzői

- Pipeline technika: nem várunk minden egyes keret után nyugtára
- Ablak (és puffer) méret: n (max párhuzamos nyugtázatlan keretek)
- Sorszám: sorozatban hányadik a keret
  - Adási/vételi ablak: küldhető/fogadható sorszámok halmaza
  - Adási ablak küldéskor szűkül, nyugta érkezésével nő
- Kumulatív nyugtázás: minden kisebb sorszámú keretet nyugtáz
  - Nyugta tartalmazza a következőnek várt keret sorszámát
  - Hibás keretet eldobjuk és nem küldünk nyugtát
  - Nem megengedett sorszámú keretet eldobjuk, de nyugtázzuk: mert ilyen akkor történhet, ha egy nyugta veszett el
- Küldő oldalon van timeout (és újraküldés)
- Duplex csatorna kell (gyakorlatban mindenképpen, elméletben határeset)
  - Piggybacking, kétirányú kommunikáció: adatkeret fejlécben a nyugta
  - Piggybacking magyarul: hátizsák
- Mi van, ha hosszú küldési sorozat közepén van a hiba?

3.4.6. Csúszó ablak stratégia: visszalépés N-nel (go-back-n)

- Összes hibás keret utáni keret eldobása, nyugtázás nélkül
- Küldő a timeout után újraküld minden keretet a hibástól kezdve
- Nagy a hibaarány $\implies$ nem hatékony

3.4.7. Csúszó ablak stratégia: szelektív ismétlés (selective repeat)

- Hibás keretet eldobja, de utána minden hiba nélküli keretet pufferel
- Küldő a timeout után újraküldi a hibás keretet
- Nyugta (ACK) és negatív nyugta (NAK) segítségével javul a hatékonyság
- Nagy vételi ablak $\implies$ nagy memória igény

4. Adatkapcsolati réteg: MAC alréteg (Medium Access Control)

- Szükséges, mert egy közegen több résztvevő osztozik
  - pl. Ethernet és WiFi többszörös hozzáférést biztosít
  - Ethernet: ma már dedikált kábelt használunk minden két állomás között, szóval nincsen ilyen típusú ütközés (max a kétirányú forgalom miatt lehetne gond, de modern Ethernet kábelek full-duplex-ek)
    - Régebben egy koaxiális Ethernet kábelre több állomás is rá volt kötve, vagy UTP kábel volt és hub-ok ($\sim$ ismétlők) voltak
- Eddig pont-pont összekötést néztünk két állomás között, most: adatszóró csatorna (broadcast channel)
- Probléma: egyidejű átvitel ütközést okozhat (ütköző keretek használhatatlanok)
- Cél: szabályok közeg megosztására és ütközés elkerülés/feloldás stratégiák

4.1. Statikus csatornakiosztás

- Példa: FDMA (Frequency-Division Multiple Access)
- Példa: TDMA (Time-Division Multiple Access)
- Jó megoldás, ha fix számú felhasználó van
- Nem hatékony, ha löketszerű (börsztös) a forgalom
- Továbbiakban nem foglalkozunk statikus csatornakiosztással

4.2. Dinamikus csatornakiosztás

- Továbbiakban csak ezzel foglalkozunk
- Mindegyik állomás egyenrangú, csak 1db csatornán kommunikálnak
- λ: érkezési folyam rátája (ráta * idő = hány csomag jött)
- Egyidejű küldés esetén ütközés, ekkor mindkét keret használhatatlan
- Kétféle időmodell:
  - Folytonos: bármelyik állomás bármikor elkezdhet küldeni
  - Diszkrét: idő diszkrét résekre van osztva, küldést csak időrés kezdetén szabad elkezdeni
    - Egy időrés lehet: üres, sikeres, sikertelen
- Vivőjel érzékelési (carrier sense) képesség:
  - Nem minden protokoll használja és nem minden hardver képes rá
  - Jelentése: állomás küldés előtt megnézni, hogy szabad-e a csatorna
- Hatékonyság mérése: átvitel (S) és terhelés (G)
  - S: időegység alatt sikeresen átvitt keretek száma
  - G: mennyi keretet kéne egy időegység alatt a protokollnak kezelni, azaz az elküldendő keretek száma az összes állomásról
  - Legyen az időegység az egy keret kiküldéséhez szükséges idő
    - Túlterhelés: $G > 1$
    - Ideális, cél: $G < 1 \implies S = G$ és $G \geq 1 \implies S = 1$

4.3. Ütközéses protokollok

4.3.1. ALOHA

- Állomások azonnal küldenek (ahogy van mit küldeni: küldenek)
- Fogadók minden csomagot nyugtáznak
  - Rádiós kommunikáció van és a fogadók egymástól messze vannak, így a fogadók egy másik fogadó nyugtáját nem kapják meg
- Ha nincs nyugta: véletlen ideig várakozás után újraküldés
- Kevés küldő esetére készült, nagyon egyszerű protokoll
- Teljesítmény elemzés (zajmentes csatornát feltételezünk)
  - Poisson folyam modellt használunk
    - Érkezések száma az intervallum hosszától függ és nem függ az intervallum kezdetétől
    - Több érkezés esélye közelít a nullához intervallum csökkentésével
    - Börsztös rendszerekre nem feltétlenül jó modell
  - k érkezés t hosszú intervallum alatt esélye: $P_k(t) = \frac{(\lambda t)^k e^{-\lambda t}}{k!}$
  - $S = S(G) = G \times$ sikeres átvitel esélye
  - Keret sikeres átviteléhez kell: senki más ne küldjön $2 * T_f$ ideig
    - T_f jelentése: keret küldés + propagálás + feldolgozás idő
    - Feltételezés: minden kerete mérete azonos $\implies T_f$ azonos
    - $2 * T_f$, mert küldés alatt se küldjön más és küldés előtt T_f-fel kezdődően se küldjön más (mert az belelógni a küldésbe)
  - $S = G * e^{-2G} = G * P_0(2 * T_f)$ ahol $\lambda = G/T_f$
    - S maximuma az $G = 0.5$ esetén van, ekkor $S = 0.18$

4.3.2. Réselt (slotted) ALOHA

- Folytonos időmodell helyett diszkrét időmodell
  - Ehhez szinkronizált óra szükséges, ami nem triviális
- Sebezhetőségi idő a felére csökken, $S = G * e^{-G}$
  - S maximuma: $G = 1$, ekkor $S = 0.37$

4.3.3. 1-perzisztens CSMA (Carrier Sense Multiple Access) protokoll

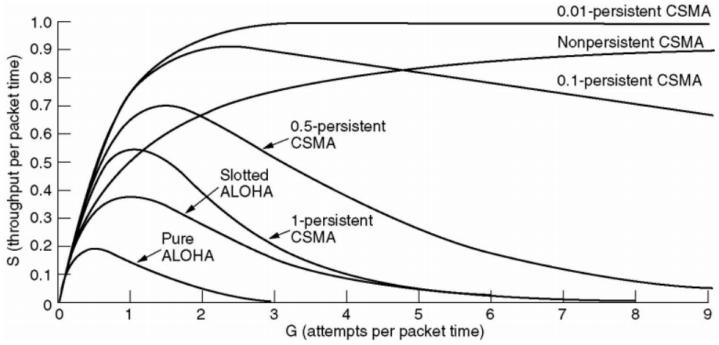
- Folytonos időmodellt használunk
- Szükséges a vivőjel érzékelési képesség
- Keret kiküldése előtt belehallgatunk a csatornába
  - Foglalt $\Longrightarrow$ várakozás és ahogy szabad, egyből küldünk
  - Szabad $\Longrightarrow$ küldünk
- Ütközés esetén véletlen hosszú ideig várakozás, majd újraküldés
- Teljesítményt a propagation delay nagyban befolyásolja
- ALOHA-hoz képest jobb kihasználtság (throughput)

4.3.4. Nem-perzisztens CSMA protokoll

- Előzőhöz képest különbség: belehallgatás során ha foglalt a csatorna, akkor véletlenül ideig várunk (majd megint megnézzük, hogy szabad-e)
- Jelentős javulás terhelt csatorna esetén: előző esetben több állomás kezdhet el küldeni, ahogy felszabadult a csatorna
- Minimális terhelés esetén az előző megoldás hatékonyabb
  - Mind késleltetés, mind kihasználtság szempontjából
- Magas terhelés esetén az előző megoldás késleltetése kevesebb, de ez a megoldás jobb kihasználtságot biztosít

4.3.5. p-perzisztens CSMA protokoll

- Ez már diszkrét időmodellt használ (folytonos helyett)
- Előzőtől eltérés még: szabad csatorna $\Longrightarrow$ p valószínűséggel küldünk ($1-p$ valószínűséggel várakozás következő időrésig)
- Alacsony p érték magas késleltetéshez és jó kihasználtsághoz vezet



4.3.6. CSMA/CD (CSMA with Collision Detection)

- Cél: ütközés érzékelése még küldés során
- Ütközés esetén küldés megszakítása és:
  - Véletlen ideig várakozás (binary exponential backoff alapján)
  - "jam" jel küldése (minden állomás értesüljön az ütközésről)
  - 16 sikertelen küldés után eldobjuk a keretet
- Minimális keret méret: kétszerese annak a bájtmennyiségnek, amennyit kiküldünk az alatt az idő alatt, amíg a legtávolabbi állomás is megkapja a keretünk legelejét
  - Valamennyi idő kell, hogy minden állomás megkapja a keret legelejét (értesüljön, hogy küldés van folyamatban)
  - Ha egy állomás ϵ idővel azelőtt kezdett el küldeni, akkor még egyszer annyi idő szükséges, hogy az "eredeti" küldő is értesüljön az ütközésről
- Nem szorosan kapcsolódik, de meg lett említve: maximális keret méret
  - MTU (Maximum Transmission Unit) általában 1500 bájt
  - Nagyobb keret $\implies$ nagyobb eséllyel hibás
  - Kisebb keret $\implies$ több keret kell $\implies$
    - Több bájtot pazarlunk fejlécekre
    - Összességében nagyobb feldolgozási idő: switch-ek csomagrátában korlátosak, csomagméret nem annyira lényeges
  - Adatközpontokban Jumbo keretek: pl. 9000 bájt

4.3.7. Binary Exponential Backoff (hátralék)

- Magyarul: bináris exponenciális hátralék
- Cél: véletlen várakozási idő generálás ütközések számától függően
- Legyen n az ütközések száma (van egy számláló, felső határ: 10)
- Válasszuk egy számot egyenletesen $[0; 2^n - 1]$ intervallumból
  - Ennyi ideig várakozzunk sikertelen küldés esetén
- Az m-edik ütközés után átlagosan:
  - Sikeres átvitel esélye: $1 - 2^{-m}$
  - Átlagos késleltetés, időegységekben mérve: $\frac{1}{2} + 2^{m-1}$
- Első ütközés után: 50% siker esély, átlagosan $1,5$ rés késleltetés
- Második ütközés után: 75% siker esély, átlagosan $2,5$ rés késleltetés

4.4. Ütközésmentes protokollok

- Állomások 0-tól N-ig vannak sorszámozva (előzetesen)
- Réselt időmodell

4.4.1. Alapvető bittérkép protokoll

- Ütköztetési periódus, versengési időrés:
  - Minden állomásnak van egy időrése és vagy küld vagy nem
  - Így minden állomás megismeri, hogy kik és hányan akarnak küldeni
- Ütköztetési periódus után adatkeretek küldése történik
  - Amelyik állomás bejelentette, hogy küldeni fog, az küld
  - Küldési sorrend: sorszámok sorrendje
  - Adatkeret időrés hossza előre rögzített
- Probléma: versengési időrés overhead túl nagy lehet sok állomás esetén

4.4.2. Bináris visszaszámlálás protokoll

- Egy versengési időrés és egy adatküldési időrés váltja egymást
- Minden állomásnak van egy bináris azonosítója
- Versengés: aki küldeni akar, elküldi az azonosítója következő bitjét
  - Ha bárki 1-es bitet küldött, akkor az eredmény is 1
  - 0-t küldők kiesnek; végén egyvalaki marad, aki küldhet egy adatkeretet
- Nem fair a megoldás

4.4.3. Mok és Ward bináris visszaszámlálás protokoll

- Állomások azonosítója változik (nem fix), hogy fair legyen a protokoll
- Azé a legnagyobb azonosító, aki a legrégebben tudott küldeni
- Aki küldött, az megkapja a minimális azonosítót
- Aki akart, de nem tudott küldeni: eggyel lépteti az azonosítóját

4.5. Korlátozott versenyes protokollok

- Cél: versenyhelyzetes és ütközésmentes protokollok ötvözése
- Kis terhelés $\Longrightarrow$ versenyhelyzetes technika (kis késleltetés érdekében)
- Nagy terhelés $\Longrightarrow$ ütközésmentes technika (jó kihasználtság érdekében)

4.5.1. Adaptív fabejárás

- Hasonlat: több embertől veszünk fertőzésmintát és összekeverjük a mintákat egybe, és ezt az egy nagyobb mintát teszteljük le (ami költséges)
  - Sok ember van letesztelve kevés költséggel
  - Pozitív eredmény esetén populáció felezése, tesztelés megismétlése
- Időrések vannak a fa csomópontjaihoz rendelve
  - 0. résben mindenki küldhet
  - Senki nem küld (üres időrés) $\Longrightarrow$ figyelmen kívül hagyjuk
  - Ütközés van $\Longrightarrow$ bal és jobb részfa megvizsgálása
  - Csak egyvalaki küldött $\Longrightarrow$ bejárás után küldhet
- Bejárás után 0, 1 vagy több állomás fog küldeni
  - Sorrend: bejárás sorrendje (balról jobbra)
- Nagyobb terhelés $\Longrightarrow$ mélyebben érdemes kezdeni a keresést

5. Adatkapcsolati réteg: legteteje, LAN, bridge

- LAN eredetileg adatszóró technológia volt
  - Minden adat elment minden gépbe a LAN-on (olcsó, egyszerű)
  - Probléma: nem skálázható, több állomás $\Longrightarrow$ sok ütközés

5.1. Bridge, híd

- Több LAN összekapcsolása, de lekorlátozza az ütközési tartományokat
  - Plug-and-play: önmagát konfigurálja gyakorlatilag
  - Hurkok feloldására is képes
- Hub-nál komplexebb (nem csak minden irányba ismétli a jelet)
  - Bridge-be ezért kell: puffer, csomag feldolgozás, routing táblák
- Nem szükséges semmit módosítani a LAN-okon: ami eddig működik vagy hub-bal működne, az ezzel is fog
- Switch: egyszerűsített bridge
  - Minden portja túloldalán csak 1 hoszt (terminál/switch) van
  - Így nincs szükséges CSMA/CD-re (nincs soha ütközés)
- Egész internet összekötésére nem alkalmas: ismeretlen hosztot megtalálni MAC alapján, internet összes MAC címét tárolni nem reális

5.2. Keret továbbítás, MAC cím tanulás

- Minden bridge-ben van egy továbbító tábla (forwarding table)
  - Oszlopok: MAC cím, fizikai port, életkor
  - Jelentése: adott MAC cím melyik fizikai port irányában található és utoljára mikor (perc nagyságrend) adott életjelet
  - Régi bejegyzéseket ki lehet törölni
- MAC cím tanulás
  - Bejövő keret forrás címét feljegyezzük a táblába
  - Táblát kézzel is be lehetne állítani, de akkor nem plug-and-play
- Továbbító tábla alapján tudjuk, hogy egy bejövő keretet merre kell tovább küldeni (keretben benne van a címzett MAC címe)

5.3. Feszítőfa protokoll (STP, Spanning Tree Protocol)

- Feszítőfa: fa, ami lefed minden csomópontot (elhagyunk éleket)
- Probléma, amit megold: hurok (kör) van a hálózatban
  - Ilyenkor egy broadcast csomag a végtelenségig kering a hálózatban
    - Broadcast csomag: hálózat minden hosztjának meg kell kapnia
  - Több kör esetén egy broadcast csomagból akár több is keletkezhet
  - Jelenség neve: broadcast storm
  - Eredmény: előbb-utóbb a switchek túl lesznek terhelve
- LAN-on általában minden link azonos (nagyságrendű) sávszélességű
  - Így általában nem számít, hogy melyik linket kapcsoljuk ki
  - De azzal, hogy kikapcsolunk linkeket, már rontjuk a hálózatban elérhető maximális sávszélességet
- Terhelés eloszlással nem foglalkozik
  - Sőt, könnyedén kialakulhatnak hot spot-ok: forgalom jelentős része egyetlen egy ponton (switchen) megy keresztül
- Feszítőfa felépítésének menete:
  - Egyik bridge ki van választva gyökérnek (erről később)
  - Minden bridge megkeresi a legrövidebb utat a gyökérhez
    - Felhasználható a szomszédos bridge legrövidebb út információja
  - Ezen legrövidebb utak uniója a feszítőfa
- Felépítéshez használt üzenetek: Bridge Protocol Data Units (BPDU)
- Gyökér bridge meghatározásának folyamata:
  - Kezdetben minden bridge azt feltételezi, hogy ő a gyökér
  - Bridge-ek minden irányban küldenek egy BPDU-t
  - Fogadott BPDU alapján minden bridge:
    - Gyökér frissítése: legkisebb ID-vel rendelkező
    - Gyökér irányába néző port frissítése
    - Kijelölt bridge frissítése (next hop a gyökér irányába)
- Újabb alternatívák: shortest path bridging, TRILL

6. Hálózati réteg (network layer)

6.1. Áttekintés

- Szolgáltatás: csomagtovábbítás, útvonalválasztás, fragmentálás
  - Csomag ütemezés (priority queueing), puffer kezelés
- Interfész: csomag küldése egy adott végpontnak
- Protokoll: routing táblák karbantartása, globálisan egyedi címek
- Példák: IPv4, IPv6

6.2. Forgalomirányítás: útvonal meghatározás

- Feladat: bejövő csomag melyik porton legyen kiküldve (továbbítva)
- Eszköz: forgalomirányító táblázat (routing table)
- Elvárások: helyesség, stabilitás, igazságosság, optimalitás, hatékonyság
- Algoritmus osztályok:
  - Adaptív: topológia és általában a forgalom is befolyásolja a döntést
    - Vannak különbségek ezen az osztályon belül
    - Honnan kapnak infót? Szomszédok? Minden router? Sehonnan?
    - Útvonal mikor frissül? Periodikusan? Terhelésre? Topológia?
    - Mit optimalizál? Távolság? Ugrások (hops)? Becsült delay?
    - Két fő alcsoport dinamikus algoritmusokon belül:
      - Távolságvektor alapú (distance vector routing)
      - Kapcsolatállapot alapú (link-state routing)
  - Nem-adaptív: offline meghatározott táblázat induláskor betöltve
- Optimalitási elv: A-ból B-be vezető optimális úton elhelyezkedő routerek optimális útja B-be az előbb említett út egy része
  - Nyelőfa: cél a gyökér, az optimális utak a fa ágai
- Feltételezés: szomszédokhoz vezető link-en lévő késleltetést valahonnan ismerjük (meg tudjuk becsülni)

6.2.1. Távolságvektor alapú forgalomirányítás (distance vector routing)

- Más néven: elosztott Bellman-Ford forgalomirányítási algoritmus
- Aszinkron működés
- Minden állomásnak saját távolság vektora (táblázata) van
  - Oszlopok: hova, költség, next hop (ami általában port, router IP)
  - Ezt periodikusan elküldi minden direkt szomszédnak
  - Minden "nem ismert" állomásnak a költsége: végtelen
- Kapott távolság vektor alapján frissül a saját táblázat
  - Frissítés: vektort küldőn keresztül költségek ellenőrzése (nőtt/csökkent)
  - Ha volt változás: saját táblát elküldjük minden szomszédnak egyből
- Először szomszédokról van infó; fokozatosan távolabbi állomásokról is
- Lassan végez az algoritmus, ha nagy a gráf átmérője
  - De a "jó hír" (alacsony költség) gyorsan terjed
- Probléma: végtelenig számolás (count to infinity)
  - Fellépése: legyen 3 állomás és 2 link: A-B-C. Az A-B link megszűnik. C elküldi a táblázatát, így B azt hiszi, hogy C-n keresztül elérhető A. C megkapja B táblázatát és frissíti a saját tábláját: B-n keresztül éri el A-t, de B-ben A költsége nőtt. Ez ismétlődik végtelenségig.
  - Költség végtelenségig nő, a csomagok pedig ciklusban ide-oda pattognak
  - Megoldás: adott költség felett végtelennek tekintjük a költséget
    - Emlékeztető: végtelen a költsége egy nem ismert állomásnak
  - Hasonló probléma: link megmarad, de költsége (jelentősen) megnő
    - Addig nő (lassan) a költség, amíg el nem éri a valódi költséget (vagy alternatív útvonal jobb költséggel nem bír)
    - Azaz "rossz hír" lassan terjed
  - Igazi megoldás: "split horizon with poisoned reverse"
    - Ha az állomás X-et Y-on keresztül éri el, akkor Y-nak küldött üzenetben X költsége végtelen (negatív információ visszaküldése)
    - Alternatíva: csak kihagyni az üzenetből az ilyen sorokat
- Egy implementáció: Routing Information Protocol (RIP)

6.2.2. Kapcsolatállapot alapú forgalomirányítás (link-state routing)

- Dijkstra algoritmusára épül
- 5 alapvető lépésből áll
  - Szomszédok keresése, hálózati címeik meghatározása (HELLO csomag)
  - Minden szomszédhoz költség (késleltetés) megmérése (ECHO ping)
  - Megismert infókból egy csomag összeállítása (pl. periodikusan)
    - Mezők: feladó, sorszám, korérték, szomszédokhoz költségek
  - Csomag elküldése az összes (nem csak a szomszédos) routernek
    - Csomag neve: Link State Advertisement
    - Küldés módja: elárasztás
  - Dijkstra algoritmusával: legrövidebb út számolása összes routerhez
    - Dijsktra működik: minden routertől kaptunk csomagot, tehát a hálózat összes router-jét és az összes link költségét ismerjük
- Nagy hálózatok esetén CPU és memória igényes
- Egy implementáció: OSPF (Open Shortest Path First)
  - Cégek és adatközpontok számára, jobban testre szabható
  - IPv4 felett van megvalósítva, IPv6-hoz OPSFv3 változat kell
  - Felépítés: átfedő területek, "Area 0" a hálózat magja
- Egy implementáció: IS-IS (Intermediate System to Intermediate System)
  - ISP-k számára, kisebb overheaddel rendelkezik
  - Többfajta eszközt támogat, ami heterogén hálózatokban előny
  - Nem függ IP-től; IPv4 és IPv6 felett is működik
  - Felépítés: 2-szintű hierarchia, ahol a 2. szint a gerinchálózat

6.3. Forgalomirányítás: csomag cím típusok

6.3.1. Unicast

- Csomag küldése két végpont között, leggyakoribb eset
- Forrás és cél egyedi azonosítóval rendelkezik (IPv4, IPv6)

6.3.2. Adatszórás, broadcast

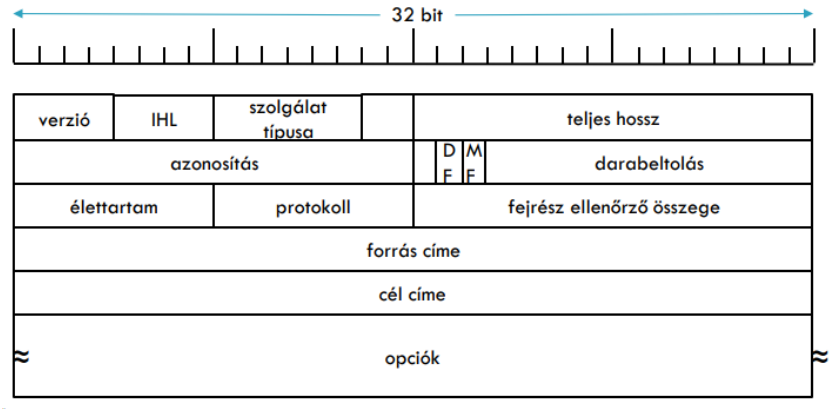
- Egy csomag egyidejűleg mindenhova elküldése
- Lehetséges megvalósítások:
  - Külön csomag küldése minden célállomásra: sávszélesség pazarló
  - Elárasztás: kétpontos kommunikációhoz nem megfelelő
  - Többcélú forgalomirányítás (multidestination routing)
    - Broadcast ez egyáltalán?
    - Csomagban van egy címzett lista (csomópontokban módosulhat)
  - Forrás routerhez tartozó nyelőfa használata
    - Csak a feszítőfához tartozó ágakon továbbít minden router
    - Szükséges, hogy ismert legyen a közös feszítőfa
  - Visszairányú továbbítás (reverse path forwarding)
    - Csak akkor broadcast-oljuk a broadcast csomagot, ha az azon a linken át jött, ami az optimális next hop a broadcast csomag forrásához
- Nem minden router tud minden implementációt

6.3.3. Többes-küldéses forgalomirányítás, multicast

- Ugyan az az infó megy mindenhova; cél: egy link-en egyszer menjen át
  - Példa: IPTV szolgáltatás sávszélesség kímélő megoldása
- Nem minden router támogatja
- Csoport kezelés szükséges: létrehozás, megszüntetés, csatlakozás, leválasztás
  - Ez nem a forgalomirányítás algoritmus része
  - Csoport rendelkezik egy adott címmel
- Minden router kiszámít egy feszítőfát, aminek azon ágain továbbítja a multicast csomagot, amelynek bármelyik pontja a csoport része

6.4. Címzés: IPv4

6.4.1. IPv4 fejrésze



- Verzió: IPv4 vagy IPv6
- IHL: fejléc hossza 32 bites egységekben (értéke legalább 5)
- Szolgálat típusa: QoS, 3 bites precedencia és 3 jelzőbit
- Teljes hossz: fejléc és adatréssz együttes hossza
- Azonosítás: egy datagram minden darabja azonos értéket hordozza
- DF: "ne darabold" flag
- MF: "több darab" flag; utolsó kivételével minden darabban "igaz"
- Darabeltolás: datagramon belül ezen darab pozíciója (8 bájt egy egység)
- Élettartam: minden ugrásnál eggyel csökken
- Protokoll: szállítási réteg protokolljának azonosítója (pl. TCP, UDP)
- Ellenőrző összeg: hibák ellen, minden ugrásnál újra kell számolni
- Forrás cím, cél cím: IP címek
- Opciók: opcionális (IHL megmutatja, hogy van-e), bővíthetőséget biztosít

6.4.2. Lehetséges címzési struktúrák

- Sík (flat): routernek minden hoszthoz kell külön bejegyzés
  - pl. MAC használata, de akkor nem tudjuk sehogy se rendezni a címeket: nagyon különböző címek is tartozhatnak azonos alhálóba
  - Eredmény: túl nagy routing táblák, amik nehezen karbantarthatók
- Hierarchikus: címek konkrét területet leíró szegmensekből állnak
  - Hasonló a telefonszámokhoz
  - Ha változik a cím, akkor nem kell minden router-t frissíteni: pl. ha a cím legvége változott, akkor elég az utolsó router-t frissíteni

6.4.3. IP cím

- Minden hoszt és router rendelkezik egyedi, 4 bájtos címmel
- Réges régen 5 címosztály lett definiálva: A, B, C, D, E
  - A: '0' + 7 bit hálózat + 24 bit hoszt
  - B: '10' + 14 bit hálózat + 16 bit hoszt
  - C: '110' + 21 bit hálózat + 8 bit hoszt
  - D: '1110' + ...: többesküldéses cím
  - E: '1111' + ...: jövőbeli felhasználás
  - Előny: alhálózati maszk nem kell, mert a cím struktúrája kódolja
- Hoszt és hálózat elkülönítése mögötti motiváció: hoszt bitekkel csak a hálózaton belül kell foglalkozni; kisebb routing táblákkal lehet dolgozni
- Speciális címek:
  - 0.0.X.Y: egy hoszt a helyi hálózaton
  - 255.255.255.255: adatszórás helyi hálózaton
  - X.Y.255.255: adatszórás egy adott hálózaton
  - 127.X.Y.Z: visszacsatolás (loopback; bitek: 0, 7db 1, utána bármi)
- Alhálózatok: alhálózati maszk (subnet mask) alapján, pl. 172.15.223.0/24
  - Egy router egy 24 bites hoszt cím tartományt nem tud kezelni
  - Táblázatban (hálózat,0) és (saját hálózat, hoszt) típusú bejegyzések
  - Nincs találat $\Longrightarrow$ alapértelmezett router felé továbbítás

6.4.4. CIDR

- Motiváció: cím osztályok nem praktikusak, mert a B címek gyorsan elfogytak, C címek pedig túl kevés hosztot tartalmazhatnak
- C osztályos címtartományok egy csoportját kapják az igénylők
- Forgalomirányítás bonyolódik:
  - Szükség van 32 bites alhálózati maszkra
  - Routing táblában innentől ilyen egy sor:
    IP cím, alhálózati maszk, következő ugrás
  - Cím alhálózat része alapján, a leghosszabban illeszkedő tábla bejegyzés alapján route-olunk
- Túl sok routing tábla bejegyzésre megoldás:
  - Aggregálás, azaz csoportos bejegyzések
  - Azonos next hop esetén elég a címek egyező kezdőszeletét tárolni

6.5. NAT (Network Address Translation, hálózati címfordítás)

- Gyors javítás IP címek elfogyása ellen
- Háztartásonként/cégenként elég egy globális IP cím, a hálózaton belül pedig minden gép egy másik címtartományból kap egyedi címet
  - Belső hálózat határán átmenő csomagok esetén címfordítás kell
  - Belső címtartományokhoz ezek vannak lefoglalva:
    10.0.0.0/8 és 172.16.0.0/12 és 192.168.0.0/16
- Címfordítás működése
  - Szállítási réteget is használja: UDP/TCP port mezőket használja
  - Kimenő csomagnál elmentjük, hogy a forrás port melyik belső IP címhez tartozik és egy új, egyedi forrás portra cseréljük
- Problémák:
  - Szállítási réteghez (TCP, UDP) szorosan hozzá van kötve
  - Csak 65536 lehetséges port van, ez nem feltétlenül elég

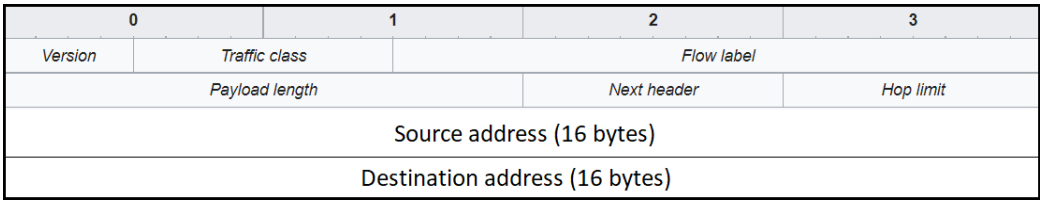
6.6. MTU és IP fragmentation (darabolás, fragmentáció)

- MTU: Maximum Transmission Unit
  - Minden hálózatnak (routernek) lehet saját MTU értéke
  - Gyakorlatban majdnem minden hálózatban 1500 bájt
- Ha egy csomag túllépi az MTU-t, akkor el lehet dobni
  - Meg van engedve: az IP csak "best effort delivery"-t ígér
  - Problémás, pl. VPN esetén: ami eredetileg MTU alatti csomagméret volt, a VPN extra fejlécek miatt már lehet túl nagy
- IPv4: ha egy csomag túllépi az MTU-t, akkor szét lehet vágni
  - IPv6-ban ilyen nincsen alapból (de MTU és MTU discovery van)
  - IPv4 fejléc második 32 bitje ehhez kapcsolódik
  - Eredeti csomaghoz azonosító, darabkák öröklik
    - "flow id"-val közösen egyedi azonosítót alkot
    - Flow-on belüli párhuzamosan több csomag lehet így fragmentálva
  - Flags: "reserved", "don't fragment" és "more fragments"
    - "more fragments" jelentése: ez nem az utolsó darab
  - Fragment offset: a darabkra a payload melyik részét tárolja
  - Akkor sikeres egy továbbküldés, ha minden kis darabka átment
    - Ha csak egy is elveszik, akkor az összes haszontalan
    - Ezért nem érdemes túllépni az MTU-t küldésnél
  - Egy csomag többször is szétvágható
  - Csomag összerakás: fogadó állomásnál
  - Kihívások fogadó félnél (mert nincs végtelen memória: puffer véges)
    - Darabok nem sorrendben érkeznek
    - Duplikáltan érkezhet meg darab
    - Darab hiányzik, nem érkezik meg
  - Problémás eset: virtualizáció van és a CPU-nak kell rengeteg csomagot fragmentálni
- IPv4/6-ban van MTU discovery protocol: "can't fragment" router válasz
  - Primitív megoldás: "don't fragment" bitű csomag küldése csökkenő mérettel, amíg válasz nem jön

6.7. IPv6

- Motiváció: kevés az IPv4 cím
- Megoldás: 32 bit helyett 128 bites címek
  - Formátum: 8 darab 16 bites hex csoport, elválasztó: `":"`
  - Csoportokban a prefix `"0"`-k elhagyhatók
  - Egy helyen `a:0:0:...:0:b` helyettesíthető `a::b`-vel
- Localhost: `::1`
- Címkiosztás:
  - Általában 64 bites alhálózat címtartományok vannak kiosztva
  - CIDR-ről már nem beszélünk, de router-ben aggregáció használt
- Teljesítmény növekedés:
  - Fejlécben nincsen checksum, amit számolni kéne
  - Egyszerűbb routing tábla szerkezet
  - Nincs szükség fragmentáció kezelésre

6.7.1. IPv6 fejléc



- 20 bájtos IPv4 fejléc helyett 40 bájt
- IPv4-ben is volt: verzió és DSCP/ECN (QoS)
- Flow label: adatfolyam azonosító
- Next header: mint IPv4-ben a protokoll mező (következő fejléc típusa)
- Hop limit: mint IPv4-ben a TTL (csak át lett nevezve)

6.7.2. További IPv6 lehetőségek

- Source routing (forrás routing): küldő határozza meg a csomag útvonalát
- Mobil IP: állomás magával viszi az IP címét másik hálózatba
  - Szólni kell az eredeti hálózat routerének, hogy most máshová mentem
  - Source routing-ot használ
- Privacy kiterjesztések: véletlenszerűen generált állomás azonosító
- Jumbogram: 4Gb-es datagram támogatás

6.7.3. Bevezetésnek nehézségei

- 1998-ban mutatták be az IPv6-ot, miért van még mindig IPv4?
- Ez a réteg felelős a globális konnektivitásért, itt homogenitás kell
- Minden router-nek, minden hosztnak kéne IPv6-ot támogatni
  - Hosztok (Windows, Android) és otthoni router-ek már támogatják
  - Gerinchálózatra (nagy, gyors, drága eszközök) várunk elsősorban
- Több protokollt is meg kell valósítani IPv6-ra (pl. ICMPv6, DHCPv6)
- Egy megoldás: IPv6 csomagokat tunnel-el becsomagolni IPv4-be

6.8. Forgalom irányítás az interneten

- Hálózati rétegben az internet autonóm rendszerek összekapcsolt együttese
  - Vannak gerinchálózatok, regionális hálók, vállalati hálók, stb.
  - Azaz többszintű routing-ról beszélünk, de mindegyik IP-t használ
    - Pl. vállalati háló használ spanning tree-t, gerincháló nem
  - LAN-on belül a routereket az adatkapcsolati réteg kapcsolja össze
- Két szintű hierarchia:
  - Autonóm rendszerek belsejei
  - Autonóm rendszerek összekapcsolva, ezek összessége

6.8.1. Autonóm rendszer, Autonomous system, AS

- Példa: Verizon, Telekom, ELTE
- Motiváció, miért van rájuk szükség
  - Eddig tanult routing algoritmusok nem elég hatékonyak, hogy az internet teljességén működjenek (megoldás: BGP)
  - Más szervezetek más forgalomirányítási policy-t akarnak
  - Lehetőséget ad a belső hálózat szerkezetének elrejtésére
  - Lehetőség a szervezetnek eldönteni, hogy merre irányítása a saját forgalmát és másik irányíthassa-e rajta keresztül
- Minden AS-t egy ASN azonosít (eredetileg 16 bit, most már 32)
- AS-k szélen olyan routerek kellenek, amik inter- és intra-domain routingot is tudnak
- Hálózatok csoportosítása:
  - Csonka (stub): egyetlen összeköttetés a BGP gráffal
  - Többszörösen bekötött (multihomed): több BGP router is van, de nincs átmenő forgalom
    - pl. adatközpontok: redundáns bekötés a cél
  - Tranzit (transit): fizetés ellenében harmadik fél csomagjait továbbítják
  - IXP, Internet eXchange Point: sok AS közvetlen összekötése

6.8.2. Intra-domain routing (AS-en belül)

- Pl. Routing Information Protocol
- Pl. Open Shortest Path First

6.8.3. Inter-domain routing (AS-ek között)

- Protokoll: BGP-4 (Border Gateway Protocol), nincs másik
  - Homogenitás szükséges, hogy globális konnektivitás legyen
  - Verziók visszafelé kompatibilisek
  - TCP-t használ a BGP a kommunikációhoz
- Tulajdonságok
  - Jól skálázható és rugalmas útvonal választás
  - Konfigurálható az útvonal, pl. adott AS-ek, vagy országok kizárása
- AS-ek hierarchiája
  - Peering link: egyenrangú AS-ek között (pl. két Tier-1 ISP között)
    - Csak olyan forgalom mehet át, ami a link egyik végén keletkezett és a másik végén terminál
    - Indoklás: más forgalomból az összekötő AS-nek nincs bevétele
  - Transit link: eltérő rangúak között (pl. ISP és végfelhasználó)
    - Az alacsonyabb rangú fizet a magasabb rangúnak a használatért
  - Egy Tier-1 ISP-nek nincsen szolgáltatója, ő a csúcs, ő minden másik Tier-1 ISP-vel peering agreement-ben van

- BGP működése: path vector routing (útvonal vektor routing)
  - Distance vector kiterjesztése: teljes útvonal számon van tartva
    - Nincs végtelenig számolás probléma
    - Hirdetés során is a teljes útvonalak vannak átadva
  - Longest Prefix Match alapján van az útvonal kiválasztva
  - Saját útvonal választási politika alkalmazható
    - Útvonal kiválasztás: melyik útvonal legyen használva?
    - Útvonal export: melyik útvonal legyen meghirdetve valakinek?
    - Peering szerződés is ilyen módon van megoldva
  - Példa routing mintára: hot potato routing: csomagot lehető leggyorsabban átadjuk másik AS-nek, akkor is, ha a miénkben több ideig maradna, akkor gyorsabban célba érhetne
  - Egy ISP milyen route-okat importál, exportál?
    - Importálás: customer, peer, provider összes route-ja
    - Exportálás customer felé: összes route
    - Exportálás peer és provider felé: customer route-jai
      - Ha peer-be megy a route, akkor peer felé van exportálva
- BGP elnevezése/változatai felhasználástól függően:
  - eBGP, External BGP: routing információk cseréje AS-ek között
  - IGP, Interior Gateway Protocol: útválasztás AS-en belüli célállomáshoz
    - Ez nem BGP, hanem pl. OSPF, IS-IS
  - iBGP, Internal BGP: útválasztás egy AS-en belül egy külső célállomáshoz
  - Példa működésre:
    - eBGP segítségével a határ router megismeri az útvonalat egy külső célállomáshoz
    - iBGP segítségével a AS többi routere is megismeri ezt az útvonalat (de valahogy el kell jutni a határ routerig először)
    - IGP segítségével egy csomag eljuttatható AS-en belül bárhova, a határ routerig például

7. Logikailag hálózati réteghez tartozó protokollok

7.1. ICMP, Internet Control Message Protocol

- Feladat: váratlan események jelentése
- Pár lehetséges üzenet és azok lehetséges okai:
  - Elérhetetlen cél: csomag kézbesítése nem sikerült
    - Lehetséges ok: nem darabolható csomag túllépte az MTU-t
  - Időtúllépés: csomag élettartam mezője elérte a 0-t
    - Lehetséges ok: hurok alakult ki pl. torlódás miatt
    - Lehetséges ok: számláló kezdeti értéke túl kevés volt
  - Paraméter probléma: fejrészben érvénytelen mező
    - Lehetséges ok: router vagy hoszt szoftver hiba
  - Forráslefojtás: egy fél küldheti, ilyenkor a fogadó állomásnak a forgalmazást lassítania kell
  - Visszhang kérés: ping parancs, a fogadó válaszol
  - Átirányítás: csomag rosszul irányítottságát jelzi
    - Lehetséges ok: nem optimális útvonalon jött a csomag

7.2. ARP, Address Resolution Protocol

- Feladat: IP cím megfeleltetése egy fizikai címnek
  - Adatkapcsolati és hálózati réteg összekapcsolása menedzsment szinten
  - LAN-on belül értelmes: csak LAN-on belüli címzéshez kell MAC
- Hozzárendelés menete:
  - Adatszóró csomag küldése Ethernet-re: "Kié a X.Y.Z.W IP cím?"
  - Hosztok összehasonlítják a kapott és a saját IP címüket
  - Egyezés esetén a hoszt válaszol a saját MAC címével
- Cache-eléssel hatékonyabbá tehető: IP-MAC cím párok elmenthetők
  - Timeout-ot szokás használni: így az IP-hez rendelt MAC megváltozhat
- Távoli hálózaton található hoszt kezelése
  - Router válaszol az ARP kérésekre (proxy ARP)
  - Összes távoli forgalomhoz az alapértelmezett Ethernet-cím használata

7.3. RARP, Reverse Address Resolution Protocol

- Feladat: fizikai cím megfeleltetése egy IP címnek
- Hozzárendelés menete:
  - Adatszóró csomag küldése Ethernet-re: "Az N bites fizikai címem F, mi az IP címem?"
    - Gyakorlatban mindig 48 bites MAC cím a fizikai cím
  - RARP szerver válaszol a megfelelő IP címmel
- Alternatívák, javítási lehetőségek:
  - BOOTP protokoll: statikus címkiosztás UDP használatával
  - DHCP protokoll: lásd lejjebb

7.4. DHCP, Dynamic Host Configuration Protocol

- Feladat: fizikai cím alapján a kommunikációhoz szükséges adatok átadása
  - Eredetileg a BOOTP kiterjesztése volt
- Kiszolgáló és kérő állomás más LAN-on is lehet, viszont ekkor a LAN-ban kell egy DHCP relay agent
- DHCP lehetőségei
  - Statikus címkiosztás: adott MAC mindig ugyan azt az IP-t kapja
  - Dinamikus címkiosztás: IP tartományból érkezési sorrendben kapnak
    - Ilyenkor elég akkora tartomány, ahány kliens egyidejűleg működik
  - Hálózati paraméterek átadása: hálózati maszk, alapértelmezett átjáró, névkiszolgáló, stb.
- Címek bérlése
  - Egy kliens csak egy adott ideig kap meg egy IP-t (lease time)
    - Kliens kérése tartalmazhatja, hogy ő mennyi időt szeretne
    - Szerver konfigurációja a kliens kérését felülírhatja
  - Lejárat előtt a bérlést meghosszabbíthatja a kliens
  - Az IP cím explicit módon vissza is adható

7.5. VPN, virtuális magánhálózatok

- Feladat, megoldandó probléma:
  - Probléma: cégnek több telephelye van vagy lehet otthonról dolgozni
  - Cél: cégnek egy nagy privát hálózata legyen (logikailag)
  - Régi megoldás: dedikált vonal vásárlása telephelyek között
- Működés: bérelt vonal helyett virtuális linkek
  - Virtuális linkek interneten átmenő alagutakkal vannak megvalósítva
  - Alagút endpoint lehet a LAN-ban lévő router vagy a kliens számítógép
  - Alagútba belépéskor a csomag egy másik csomagba kerül beágyazásra
    - Megnevezése: IP az IP-be mechanizmus
- Autentikáció, titkosítás: IPSEC segítségével
  - Autentikáció: ne tudjon bárki VPN-hez csatlakozni
  - Titkosítás: az alagútban lévő csomagok titkosítva legyenek

7.5.1. IPSEC

- Hosszú távú cél: biztonságos IP réteg
- Műveletek, mit foglal magába:
  - Hoszt párok kommunikációjához kulcsok biztosítása
  - Kommunikáció kapcsolatorientáltabbá tétele
  - Fejlécek és láblécek hozzáadása az IP csomagok védelme érdekében
- Több módot támogat, ezek egyike az alagút mód
  - ESP header: Encapsulating Security Payload
  - HMAC: Hash-based Message Authentication Code

8. Szállítói réteg (transport layer)

- Szolgáltatás: multiplexálás, megbízhatóság, sorrendhelyesség
  - Torlódásvezérlés: enélkül újraküldések miatt telítődne a puffer
  - Fairség és csatorna kihasználtság egyensúlyozása
  - Hosszú élettartamú kapcsolatok támogatása
- Interfész: üzenet küldése cél processznek
- Protokoll: port szám, folyamfelügyelet, megbízhatóság
- Példa: UDP, TCP

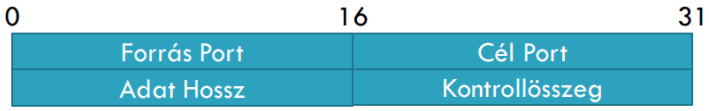
8.1. Multiplexálás

- Egy kliens számos alkalmazást futtathat egy időben
  - Másik eset: egy szerver számos klienssel beszél egyszerre
- El kell dönteni, hogy egy adott csomag melyikhez tartozik
- IP fejléc protokoll mezője 8 bites, ez nem elég nagy
- Megoldás: portok bevezetése

8.2. Réteg modellek áttekintés

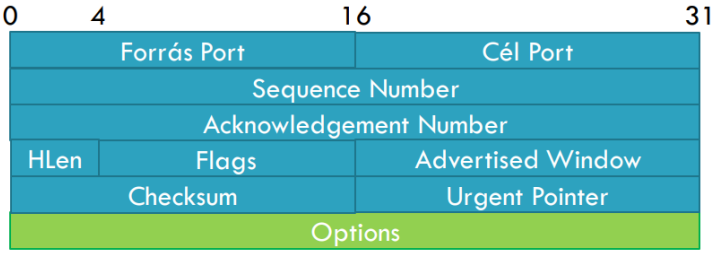
- Alacsonyabb rétegekben (fizikai, adatkapcsolati, hálózati) a szomszéd hosztokkal kommunikálnak az eszközök
- Szállítói (és alkalmazási) réteg: közvetlenül a cél hoszttal kommunikálunk
  - Peer-to-peer kommunikáció: gyakorlatilag a cél lett a szomszédunk
  - Szállítói réteg fejléceit csak a forrás és cél végpontok olvassák ki
    - Nem teljesen igaz: pl. tűzfal vagy DDoS védelem miatt routerek is sokszor megnézik az UDP/TCP fejléceket

8.3. UDP



- Egyszerű, kapcsolat nélküli átvitel
- Kicsi, 8 bájtos fejléc
  - Demultiplexálás portszámok segítségével
    - 16 bites port, 0-ás port nem megengedett → 65535 lehetőség
  - Kontrollösszeg segítségével hibás csomagok felismerhetőek
    - Nincs védelem elveszett, duplikált és helytelen sorrendben érkező csomagoktól
- Motiváció (TCP után vezették be)
  - TCP nem minden alkalmazásnak megfelelő
  - UDP felett egyedi protokollok valósíthatóak meg
    - Megbízhatóság, helyes sorrend nem mindenkinek szükséges
      - pl. real-time média streamelés megfelelő enkódolással
    - Adatközpontokban saját folyamvezérlés protokollok alkalmazása

8.4. TCP



- TCP egy absztrakt bájt folyam megvalósítása
- Megbízható, sorrend helyes, kapcsolat alapú két irányú bájt folyam
- 20 bájtos fejléc + options fejlécek (UDP-vel azonos port rendszer)
- TCP-ben van: folyam vezérlés, torlódás vezérlés, fair csatorna hozzáférés

8.4.1. Kapcsolat kezelés

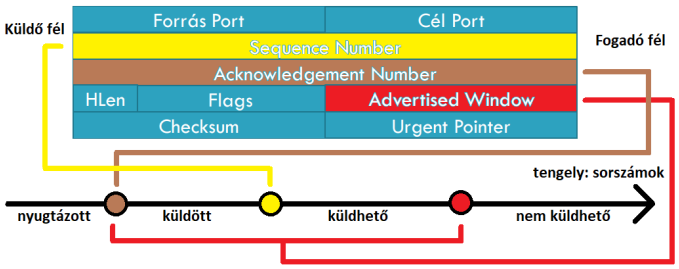
- Miért szükséges? Miért kapcsolat-alapú a TCP?
  - Állapot kialakítása mindkét végponton
  - Példa: sorszámok (sequence numbers) számontartása
- Pár fontos TCP flag (1 bites jelölő):
  - SYN: szinkronizációs, kapcsolat felépítéshez
  - ACK: fogadott adat nyugtázása
  - FIN: vége, kapcsolat lezáráshoz
- Kapcsolat felépítése: three way handshake (három-utas kézfogás)
  - Cél: mindkét fél ismerje meg a másik fél kezdő sorszámát és mindkét fél nyugtázza a másik fél kezdő sorszámát
  - Első lépés: kliens → szerver: [SYN, SeqK, 0]
  - Második lépés: szerver → kliens: [SYN/ACK, SeqS, SeqK+1]
  - Harmadik lépés: kliens → szerver: [ACK, SeqK+1, SeqS+1]
    - Ebben az üzenetben már payload is lehet
  - Támadási felület: minden SYN állapotot foglal a szerveren
    - SYN flood, DoS egy fajtája: csak SYN küldése, de belőle sok
    - Megoldás a SYN cookies: kezdeti sorszámba enkódolja a tárolandó állapotot ahelyett, hogy az első SYN-nél letárolná a szerver
- Kapcsolat lezárása
  - Bármely oldal kezdeményezheti a lezárást
  - Menete: FIN flag küldése, majd másik fél is küld FIN-t
    - FIN-t tartalmazó csomagokat is nyugtázni kell
  - Kihívás: egyik fél lezárta, de másik fél még küld újabb csomagot
  - Félig nyitott kapcsolat: egyik fél lezárta, másik még nem

8.4.2. Sorszámok

- Minden bájtnak van egyedi sorszáma: 32 bites érték, idő után körbefordul
  - Kezdeti sorszám biztonsági okokból véletlen szám: nehéz kitalálni
- Bájt folyam szegmensekre (TCP csomagokra) van bontva
  - Maximális méret: MSS (Maximum Segment Size)
  - MSS-t úgy kell beállítani, hogy ne legyen fragmentáció
  - Szegmenseknek van egyedi sorszámuk: első bájt sorszáma
- Két irányba két különböző sorszám létezik
  - Hiszen mindkét fél küldhet és fogadhat is adatot, akár párhuzamosan

8.4.3. Folyamvezérlés

- Probléma: küldő milyen gyorsan küldheti az adatot?
  - Túl sok csomag túlterheli a fogadót
  - Fogadó oldal puffer mérete változhat a kapcsolat során
    - Ez a puffer az (app általi) feldolgozásra váró bájtokat tárolja
- Megoldás: csúszóablak (sliding window)
  - Fogadó megosztja a pufferének a méretét (ez az advertised window)
    - Magyarul meghirdetett ablak, és értéke akár 0 is lehet
  - Küldő N ablakméretnél N bájtot küldhet nyugta fogadása nélkül
  - Csúszóablak előre lép (csúszik) minden nyugta fogadásakor
  - Átvitel, átvitt adatmennyiség $\sim \frac{\text{w (window)}}{\text{RTT (Round Trip Time)}} = \frac{\text{küldési ablakméret}}{\text{körülfordulási idő}}$
    - Rövid RTT $\implies$ ACK gyorsan jön $\implies$ ablak gyorsan lép



- Nyugtázás típusok
  - Minden csomag külön nyugtázása
  - Kumulált nyugta: minden $K < N$ sorszámú csomag nyugtázása
    - TCP-ben alapértelmezett, de nem ez a leggyakoribb
  - Negatív nyugta (NACK): adott csomag hiányának jelzése
  - Szelektív nyugta (SACK): megérkezett csomagot felsorolása
    - Nem az alap TCP része: SACK TCP nevű kiterjesztés
    - Gyakorlatilag minden TCP implementáció ezt használja
    - Felesleges újraküldések számát jelentősen csökkenti
- Túl kicsi ablak $\implies$ sok kicsi csomag, fejlécek dominálják az átvitelt
- Nagle algoritmusa: kicsi csomagok küldésének elkerülése
  - Cél: ne pazaroljuk a sávszélességet fejlécekkel, stb.
  - Ha ablak >= MSS és elérhető adat >= MSS: küldés
  - Különben ha van nem nyugtázott adat: várakozás nyugtára/adatra
  - Különben: "nem teljesen telített" csomag küldése
  - Probléma: késleltetve van az átvitel
    - Használható `TCP_NODELAY` flag, ha azonnali küldés szükséges

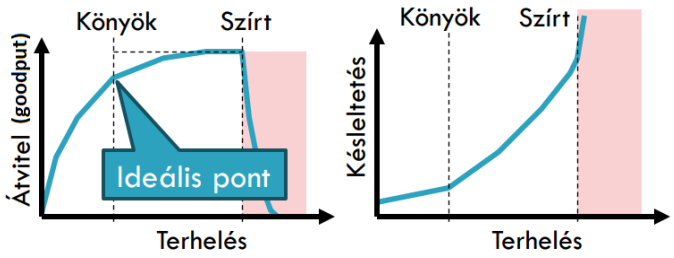
8.4.4. (Sorrend/kontrollösszeg) hiba detektálás, újraküldés

- Fogadó oldal:
  - Kontrollösszeg újraszámolása és összehasonlítása a fejlécbeli értékkel
    - A kontrollösszeg az IP, TCP fejlécből és az adatból van számolva
    - Hibás kontrollösszeg esetén a csomagot eldobjuk
  - Sorszámok segítségével sorrendhelyes átvitel garantálása
    - Duplikátumok eldobása
    - Helytelen sorrendben érkező csomagok sorba rendezése / eldobása
      - Fogadhatunk nem sorrendhelyesen, ha a pufferbe befér
    - Hiányzó sorszámok elveszett csomagot jeleznek
- Küldő oldal:
  - Minden nyugtázatlan csomag pufferelése a nyugta megérkezéséig
  - Időtúllépés (timeout) segítségével hiányzó nyugták érzékelése

- Időtúllépés újraküldéshez (RTO, Retransmission Time Outs)
  - RTO az RTT-től függ: ennyi idő alatt ér vissza ideálisan a nyugta
  - TCP-ben RTT becslése: mozgó átlaggal
    - `new_rtt = a * old_rtt + (1-a) * new_sample`
    - `new_sample` egy küldés és az ACK megérkezés közötti idő
    - Minden küldés-nyugta párnál frissítjük az értéket
      - Karn algoritmusaa: eldobjuk azokat a mintákat, melyek egy újraküldésből származnak
      - Indoklás: Mi van, ha nagyon megugrott az RTT, és közvetlenül az újraküldés után megérkezik az eredeti nyugta?
    - `a` javasolt értéke: 0.8 és 0.9 között (általában 0.875)
  - `RTO = 2 * new_rtt` (konzervatív becslés, internetre egész jó)
    - RTO alsó korlát: 300ms, ami adatközpontokban problémás (hiszen RTT $\approx$ 1ms), lásd pl. TCP incast problem

8.4.5. Torlódás bevezetés

- Torlódás akkor van, ha a hálózat terhelése nagyobb, mint a kapacitása
- Torlódás fellépésnek lehetséges okai:
  - Hálózatban a kapacitás nem egyenletes, pl. modem vs üvegszál
  - Több folyam verseng a sávszélességért, pl. filmnézés és böngészés
  - Időben egyenletlen a terhelés, pl. GoT-t mindenki egyszerre nézi
- Torlódás következménye: csomagvesztés
  - Router gyorsabban kap csomagokat, mint ahogy küldeni tud
  - Router véges pufferrel rendelkezik $\implies$ el kell dobni csomagokat
  - RED (Random Early Detection): ahogy telítődik a puffer, úgy nagyobb eséllyel dobjunk el véletlenszerűen csomagokat
- Csomagvesztés következménye:
  - Megnövekedett késleltetés (mert csomagok újraküldése szükséges)
    - Egyébként is nő: puffer végére kerül az új csomag, várnia kell
  - Sávszélesség pazarlás újraküldések miatt
  - Alacsony hálózati átvitel a pazarlások miatt (goodput)



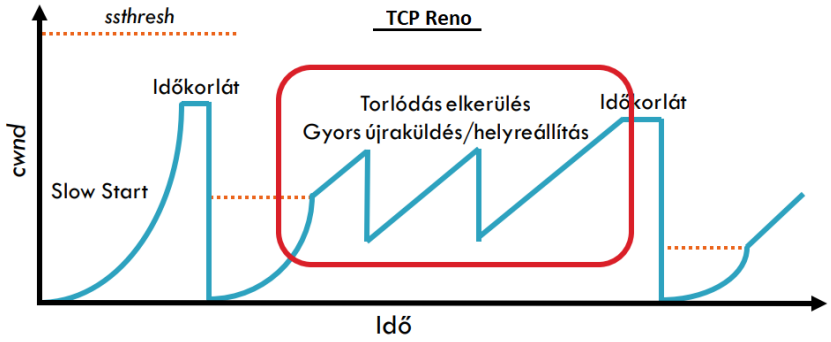
- Könyök (knee) után: átvitel alig, késleltetés gyorsan nő
- Szírt (cliff) után: átvitel $\to 0$, késleltetés $\to \infty$ (teljes összeomlás)
- Késleltetés egyszerűsített modellezése: késleltetés $\sim \frac{1}{1-\text{utilization}}$
- Torlódás elkerülés: könyök bal oldalán maradás
- Torlódás vezérlés: szírt bal oldalán maradás

8.4.6. Torlódás vezérlés (congestion control)

- TCP-nek alapból nem volt része, de később szükségessé vált
- Valójában torlódás elkerülést végez a TCP (könyök közelítése a cél)
- Konstans méretű ablak csak a fogadót védi a túlterheléstől
  - Nem segít, ha a hálózat a szűk keresztmetszet és nem a fogadó
  - Megoldás: állítható ablakmérettel a küldési ráta korlátozása
- Általános megoldási lehetőségek
  - Ne csináljunk torlódás vezérlést
    - Jósolhatatlan teljesítmény, akár teljes összeomlás
  - Erőforrás foglalás
    - Küldés előtt folyamokhoz előre sávszélességet foglalunk
    - Hálózati támogatás szükséges
  - Dinamikus beállítás (TCP ezt csinálja)
    - Torlódási szint becslése, ez alapján küldési ráta csökkentése/növelése
    - Elosztott koordináció, nem rendezett dinamika

- Küldő oldalon torlódási ablak bevezetése (cwnd, congestion window)
  - Cél: ablak méretének változtatása küldési ráta változtatásához
  - Hiszen a küldési ráta arányos a $\frac{\text{window}}{\text{RTT}}$ értékével
  - `window = min(congestion_window, advertised_window)`
- Torlódás detektálás
  - Késleltetés alapú megoldások nehezek és kockázatosak, pontatlanok
  - Eldobott csomag egy biztos jel
    - Időkorlát lejárt és nem jött nyugta $\implies$ csomagvesztés történt
    - Számos duplikált ACK jön sorban $\implies$ csomagvesztés történt
      - Duplikált nyugtákról, e mögötti logikáról később lesz szó
- Általános algoritmus
  - ACK jött $\implies$ cwnd növelése
    - Adat sikeresen átment, tehát valószínűleg gyorsabban is küldhetünk
    - Következmény: RTT-től függ a növelések gyakorisága
  - Csomagvesztés történt $\implies$ cwnd csökkentése
    - Adat elveszett, tehát valószínűleg torlódás van
  - Implementáció feladata: cwnd növelés/csökkentés mit jelent pontosan
- TCP Tahoe: TCP-ben az eredeti algoritmus implementáció
  - `ssthresh`: könyökpont alsó becslés, alapból `ssthresh = advertised_window`
  - Slow start, lassú indulás fázis: cwnd < ssthresh és nincs csomagvesztés
    - Cél: (ismeretlen "értékű") könyökpont (bottleneck) gyors elérése
    - Indulás: `cwnd = 1`
    - Minden nyugtázott szegmens esetén cwnd növelése eggyel
      - Tehát cwnd időben exponenciálisan nő: RTT-nként duplázódik
  - Torlódás elkerülés fázis: cwnd >= ssthresh
    - AIMD: Additive Increase Multiplicative Decrease
      - TCP Tahoe valójában még nem multiplicative decrease
    - Nyugtázott szegmens esetén `cwnd += 1/cwnd`
      - Tehát cwnd időben lineárisan nő: kb. RTT-nként eggyel
    - Csomagvesztés esetén: induljon egy slow start fázis elölről `ssthresh := cwnd / 2` értékkel
  - Ez az algoritmus működik, de nem túl hatékony, hamar lecserélték

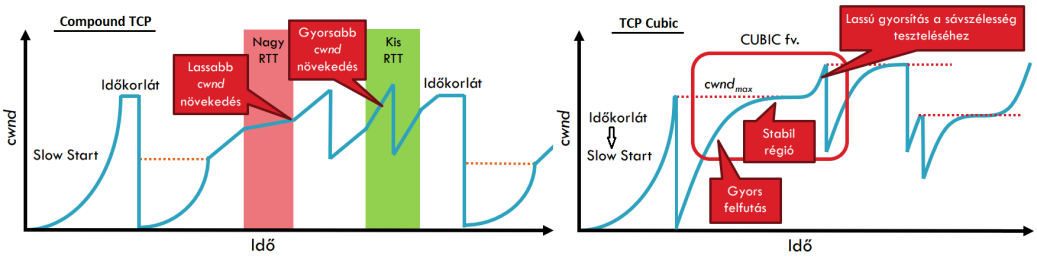
- TCP Reno: TCP Tahoe kiegészítése
  - Gyors újraküldés: 3 duplikált ACK $\Longrightarrow$ újraküldés, ne várjunk
    - Alapból meg kéne várni az RTO-t
    - Ha több küldött csomagból az első elveszik, akkor a fogadó az elveszett előttihez tartozó ACK-t küldi minden megérkezetthez
  - Gyors helyreállítás: gyors újraküldéskor `ssthresh:=cwnd:=cwnd/2`
    - Ezzel a felesleges slow start fázis el van kerülve
    - RTO lejárta esetén továbbra is `cwnd:=1` van érvényben
      - Ez nagyon súlyos torlódás esetén történik meg
  - Stabil állapotban a cwnd az optimális érték körül oszcillál



- TCP mindig csomagdobásokat kényszerít ki: mindig túlmegy
- Egyéb TCP torlódásvezérlés változatok
  - NewReno: javított gyors újraküldés
    - Minden egyes duplikált ACK újraküldést vált ki
    - Hiba: `>3` hibás sorrendben fogadott csomag is újraküldést okoz
  - Vegas: késleltetés alapú torlódás elkerülés
  - stb.

8.4.7. TCP a valóságban: manapság használt változatok

- TCP rosszul teljesít nagy RTT (késleltetés) vagy nagy kapacitás (sávszélesség) esetén, vagy ha ennek a kettőnek a szorzata nagy
  - Késleltetés-sávszélesség szorzat: maximális szállítás alatt lévő adatmennyiség
  - Indoklás: slow start és az additive increase csak lassan konvergál
  - Csak nyugta érkezésére reagál TCP: nagy RTT $\implies$ nagy reakcióidő
- Cél egy egyszerű TCP implementáció, ami
  - Küldési ablakot gyorsabban növeli (gyorsabb konvergencia)
  - Más változatokkal szemben fair: ablakot nem növeli túl agresszívan
  - TCP Tahoe/Reno-hoz képest fair-ebb: eltérő RTT esetén is fair
- Compound TCP (Windows által használt)
  - TCP Reno alapú: `cwnd` nem változik, ugyan úgy AIMD
  - Két torlódási ablak: késleltetés és vesztés alapú (összetett torlódásvezérlés)
  - `wnd := min(cwnd + dwnd, adv_wnd)` ahol `dwnd` késleltetés alapú
  - `dwnd` és RTT fordított arányos: egyik nő, másik arányosan csökken
  - Következmény: agresszíven reagál az RTT változására
  - Előny: gyors felfutás, fair viselkedés eltérő RTT esetén is
  - Hátrány: folyamatos RTT becslés szükséges



- TCP CUBIC (Linux által használt)
  - BIC (Binary Increase Congestion Control) továbbfejlesztése
  - AIMD helyett az ablakméretet egy harmadfokú egyenlet határozza meg, ami a legutolsó csomagvesztéstől eltelt idővel van paraméterezve
  - Gyors felfutások miatt kevésbé pazarolja a sávszélességet
  - Stabil régió és lassú gyorsítás segíti a fairség biztosítását
    - De az additive increase-hez képest túl agresszív a gyors felfutás

8.4.8. Problémák a TCP-vel

- Internetes forgalom jelentős része TCP
- Kis folyamok esetén gyenge teljesítmény
  - Legtöbb folyam kis folyam (<100 kB), pl. HTTP
  - Kapcsolat felépítéséhez 1 teljes RTT szükséges (idő pazarló)
  - cwnd mindig 1-ről indul, a kevés adat miatt soha nem gyorsul fel
  - Kis folyamok el sem hagyják a slow start fázist
  - Megoldási javaslat (Google-tól):
    - cwnd kezdetben legyen 10
    - TCP Fast Open: három-utas kézfogás helyett kriptográfiai hash (cookie) küldése a SYN csomagban
- Wireless hálózatokban gyenge teljesítmény
  - Vezetékes hálózatokban csomagvesztés tényleg torlódást jelent
  - Vezeték nélküli hálózatokban viszont nem csak ilyenkor van csomagvesztés
  - TCP: goodput $\sim \frac{1}{\sqrt{\text{vesztési ráta}}}$ (néhány csomagvesztés drasztikus)
  - Lehetséges megoldás: késleltetés alapú csomagvesztés (TCP Vegas)
  - Vagy rétegmodell megsértése és adatkapcsolati infó TCP-be
- DoS támadási felület (Denial of Service, szolgáltatás megtagadása)
  - Probléma: TCP kapcsolatok állapottal rendelkeznek
  - SYN flood: lásd korábban

9. Kitekintés: Queue Management, ECN és TCP

9.1. Switchben queue management

9.1.1. Egyszerű megoldás: FIFO + drop-tail

- Interneten elterjedt
- FIFO (first in first out): csomagok érkezési sorrendben kiküldése
  - Nem tesz különbséget csomagok között
  - Ez egy ütemezési (scheduling) beállítás
- Drop-tail: teli puffer esetén a beérkező csomag eldobása
  - Nem tesz különbséget csomagok között
  - Ez egy eldobási beállítás

9.1.2. RED algoritmus (Random Early Detection/Drop)

- Ez egy AQM (Active Queue Management) eldobási beállítás
  - Cél: átlagos queue delay csökkentése, de börsztösség engedélyezése
  - Megvalósítás: proaktív csomagdobás vagy ECN használata
- Legyen `avgq` a puffer hosszának futó átlaga
- Legyen `th_min` és `th_max` két küszöbérték (konstans)
- Bejövő csomag esetén:
  - `avgq < th_min` $\implies$ csomag továbbítása
  - `avgq > th_max` $\implies$ csomag eldobása
  - Egyébként $\frac{\texttt{avgq}-\texttt{th\_min}}{\texttt{th\_max}-\texttt{th\_min}}$ eséllyel csomag eldobása vagy ECN beállítása
- ECN: Explicit Congestion Notification, IP fejléc része
  - Router beállít egy flag-et és normál módon továbbítja a csomagot
  - Fogadó fél érzékeli a flag-et és TCP ACK-ben beállít egy másikat
  - Előny: nincs szükség újraküldésre
  - Előny: forrásnak a reagáláshoz nem kell timeout-ra várnia
  - Hátrány: végpontoknak támogatnia kell ezt a lehetőséget

9.2. Data Center TCP: DCTCP

9.2.1. Adatközpontok hálózatának jellemzői

- Adatközpontokban nagyon kicsi a RTT ($\approx$ 1ms)
- Egyes flow-k kicsik és késleltetésre érzékenyek
- Egyes flow-k nagyok (elephant flow) és sávszélességre érzékenyek

9.2.2. Adatközpontok hálózat szintű problémái

- Incast probléma
  - Egy aggregator szerver sok worker-nek ad feladatot, amelyek nagyjából ugyan akkor vannak kész, és így ugyan akkor válaszolnak és a router puffert megtelítik
  - Egyes worker válaszok elvesznek és az RTO legalább 300ms TCP-ben, ami RTT-hez képest túl sok
  - 300ms-on kívüli hibák is vannak, pl. túl pontatlan a becslés
- Queue buildup
  - Elephant flow-ok megtelítik a puffereket
  - Kicsi, késleltetés érzékeny flow-ok késleltetést szenvednek

9.2.3. DCTCP: TCP/ECN Control Loop

- Switch jelölje a bejövő csomagot, ha az aktuális pufferméret K feletti
  - Kizárólag ECN használata: nincs csomag eldobás jelölésnél
  - RED-től eltérően nem átlagot használ $\implies$ gyorsabb reakció
  - Gyors reakció börsztöknél fontos
- Küldő fél futóátlagot vezet az ECN-nel jelölt csomagok arányáról (α)
  - $\alpha := (1 - g) * \alpha + g * (\frac{\text{\# of ECN-marked ACKs}}{\text{Total \# of ACKs}})$
    - A két #-os számítás pl. az utolsó 10 csomagot veszi figyelembe
  - cwnd := $(1 - \frac{\alpha}{2} * \text{cwnd})$, azaz max. 50%-os csökkenés
  - TCP esetén egyetlen ECN-es csomag esetén már 50%-os a csökkenés
  - DCTCP: torlódás mértékével arányos a reakció, nem pedig a jelenlétével
  - Tehát fűrészfogás ráta helyett szépen oszcillál optimum közelében

10. Alkalmazási réteg (application layer)

- Bármi lehet, mi csak kb. két protokollt nézünk meg

10.1. DNS (Domain Name System)

10.1.1. Bevezetés

- Pl. Google vagy Facebook elérése szükség van annak IP címére
- Hiba az emberben van: nem képes megjegyezni Google, stb. IP címét
- DNS feladata: domain névről IP címre képzés
- Internet használata DNS előtt
  - `/etc/hosts` tartalmaz minden név-IP párt
  - Központosított: egy FTP szerverről periodikusan frissült ez a fájl
    - Probléma: nem minden gép volt naprakész
  - Manuális: új párok bevételéhez az FTP szerver üzemeltetőjéhez manuálisan kellett kérelmet benyújtani emailben
    - Az üzemeltető: SRI, Stanford Research Institute
    - Probléma: nem győzték a kérések feldolgozását
  - Minden név megengedett volt, hierarchia nélkül
    - Probléma: egyediséget hogyan biztosítsuk?

10.1.2. DNS felépítése

- DNS nem központosított: elosztott adatbázis
- Kliens-szerver architektúra
  - UDP 53-as portját szokta használni
  - Rövid kérés, rövid válasz típusú üzenetek
- Névtér hierarchikus: fával ábrázolható, teteje a `root`
  - Minden domain név egy részfa a fában
  - Legfelső szint: TLD (Top Level Domain)
    - Kb. 250-300 TLD létezik, nagyrészt ország kódok
  - Maximum 128 szint lehetséges

- Adminisztráció hierarchikus
  - A névtér fa különböző méretű zónákra bomlik
  - root zónáért, azaz a TLD-k kiosztásáért felölős szerv: ICANN
  - Pár TLD-ért például a Verisign felel (pl. `*.net`)
  - `*.elte.hu`-ért az ELTE felel
- (Név)szerver hierarchia, felügyelő szerverek
  - Minden szerver a hierarchia egy részét felügyeli
  - Minden szerver a saját zónájához tartozó bejegyzéseket ismeri
    - Ennek az adatösszességnek a neve: zónafájl
    - Ezen felül cache-elés miatt tárolhat extra bejegyzéseket
    - Egy szerver sem tárolja az összes létező bejegyzést
  - Valamint minden szerver ismeri a root szerver(ek) címét
    - Tehát egy DNS szerver bármely cím feloldására képes
    - A root szerver (fentiek alapján) minden TLD-t ismer
  - Egy szerver az alárendeltjeit ismeri (ha vannak)
    - Pl. a root szerver az összes TLD felelősét számon tartja
      - Ezt hívjuk Root Zone File-nak
      - Sok szerver cache-eli a Root Zone File-t
- Felügyelő névszerverek terhelés megosztása
  - Több szerver futhat párhuzamos, amik azonos zónáért felelnek
  - Pl. 13 darab root szerver van, 6db pedig anycast-olt: számos replika létezik
- Lokális névszerver
  - Minden hálózatnak van default névszerver, általában DHCP megadja
  - Minden DNS lekérdezés a lokális névszerver által van feldolgozva
  - Eredményeket általában cache-eli
  - Ismeretlen domain esetén egy másik névszerverhez fordul
- Authoratív névszerver: tárol név $\rightarrow$ IP-cím leképezést
- Cache-elés: minden, pl. a Root Zone File is maximum 72 óráig cache-elhető, utána kötelező frissíteni

10.1.3. Domainnév feloldás menete

- Hoszt a lokális DNS szervernek küld egy kérést
- Ha a lokális DNS szerver egyből tud válaszolni, akkor válaszol
  - Tud válaszolni, ha van cache-elt válasza vagy ő egy felügyelő
- Egyébként root-tól kezdve felülről lefelé végigmegy a fán
  - Minden lépésben kap egy következő szervert, aki tudja a választ
    - Vagy ha nem is tudja a választ, akkor tud valakit, aki tudja...
    - Itt pár lépés kihagyható, ha van cache-elt közbülső érték
      - Pl. az LTD névszerverek általában cache-elve vannak
  - Utolsó lépésben pedig megkapja a keresett IP címet
  - Az utolsó választ továbbítja a hoszt felé
- Lekérdezésnek két fő fajtája van
  - Rekurzív: névszerver végzi el a teljes feloldást; IP-vel válaszol
    - Például lekérdezi a root-ot, majd a com-ot, majd a google.com-ot és a választ visszaadja a kérdezőnek
    - Hátrány: meg kell jegyezni, hogy az IP-t kinek kell visszaküldeni
    - Előny: hatékonyabb cache-elés, kliens tehermentesítése
    - Lokális névszerver ezt csinálja
  - Iteratív: névszerver csak egy lépést teszt a névfeloldásban
    - Előny: nagyon gyors
    - Lokálison kívül minden névszervert ezt csinálja

10.1.4. DNS erőforrás rekordok (resource records)

- Lekérdezések két mezejük van: név és típus
- Válaszoknak 4 oszlopuk (név, érték, típus, TTL) és 1/több soruk van
- "A" (IPv4) vagy "AAAA" (IPv6) típus:
  - Név: domain név
  - Érték: IP cím
- "NS" típus:
  - Név: rész domain
  - Érték: rész domaint felügyelő DNS szerver neve
- "CNAME" típus:
  - Név: domain név
  - Érték: kanonikus név
  - CDN (Content Delivery Network) használja, alias nevekhez
- "MX" típus:
  - Név: emailben szereplő domain név
  - Érték: mail szerver kanonikus neve
- "PTR" típus: (fordított lekérdezés)
  - IP címhez tartozó domain nevet adja meg (ha van ilyen)
  - Külön hierarchiája van, melynek gyökerei: in-addr.arpa és ip6.arpa
  - Név: IP cím
  - Érték: domain név

10.1.5. DNS, mint absztrakciós szint

- Szerver IP címét meg lehet változtatni, a DNS bejegyzést frissíteni kell, de a domain név ugyan az maradhat
  - Azaz könnyű egy szolgáltatás üzemeltetését máshova költöztetni
- Egy gépnek számos alias neve (domain neve) lehet
  - Egy szervergép több szolgáltatást is üzemeltethet
- Egy domain névhez számos IP cím tartozhat
  - Példa: Content Delivery Network

10.1.6. DNS-sel kapcsolatos támadások

- Denial of Service lehetséges DNS szerverek ellen
  - Root szerver ellen nem sok hatása van cache-elés miatt
  - Lokális ellen hatásos: hosztok nem tudnak semmi újat feloldani
  - Authoratív ellen hatásos: zóna feloldhatatlan lesz
- DNS hijacking (eltérítés)
  - UDP protokoll nem ad biztonságot
  - Csak le kell hallgatni egy DNS kérést és gyorsabban kell válaszolni
- DNS cache poisoning
  - Küldünk egy kérést és közvetlenül utána egy választ rá egy szerverhez
  - Így a DNS szerver cache-eli, hogy az adott domainhez melyik IP címet vagy melyik felügyelő DNS szervert adtuk meg
  - Bejövő kérésekre a DNS szerver a mi megadott értékünkkel válaszol
  - DNS hijacking-gel ellentétben egy egész ISP megfertőzhető
- "Helytelen" DNS szerver választása lokális névszernek akár hasznos is lehet: adott domain nevek ne legyenek feloldva (pl. egy szoftver ne tudjon kommunikálni a központi gyártó szerverrel)

10.1.7. DNSSEC: egyes biztonsági problémákra megoldás

- 2010-től használják a root szerverek
- Új erőforrástípusok
  - "DNSKEY" típus:
    - Név: zóna domain
    - Érték: zóna publikus kulcsa
  - "RRSIG" típus:
    - Név: típus és név páros
    - Érték: lekérdezés eredményének kriptográfiai aláírása
- DNSSEC működése
  - Kliens először a megszokott módon végez egy DNS lekérdezést
    - Válasz extra mezőket tartalmaz: szignatúra és kulcs
  - Kapott válasz hitelességét utána ellenőrzi a kliens
    - Zóna felügyelője képes hitelesíteni a zónától kapott kulcsot
    - Rekurzívan a felügyelő kulcsa is hitelesíthető annak felügyelőjével
    - Ez a hierarchia megegyezik az előbbiekkel
    - Trust Anchor: publikus kulcsok gyökere, ebben meg kell bízni
      - A kliens DNS konfigurációjának része ennek a beállítása
- Reflection, DNS amplification attack ellen nem nyújt megoldást
  - Probléma forrása: DNS UDP-t használ és a forrás cím hamisítható
    - X küldhet olyan csomagot, aminek forrás mezejében Y áll
    - UDP-t használ a DNS, így ez nem derül ki
    - A szerver válaszol a megadott forrás címre, bármi is legyen az
  - Miért érdemes ezt csinálni?
    - Egyik ok: az igazi támadó IP címe titkos marad
    - Igazi ok: egyes DNS válaszok nagyobbak, mint a kérések
      - Azaz pl. 10 bájtot küld a támadó, az áldozat felé a DNS szerver viszont 100 bájttal "válaszol"
  - Eredmény: DNS szerver azt hiheti, hogy őt az áldozat támadja
    - Hiszen a DNS szervernek feltűnhet, hogy túl sok kérést kap

10.2. HTTP (HyperText Transfer Protocol)

10.2.1. Bevezetés

- Egy weboldal objektumokból áll (pl. HTML fájl, kép, Java applet)
- Minden objektum egy URL által azonosítható, címezhető
- Weboldalak kliens-szerver architektúrában működnek
  - Böngésző HTTP kéréseket küld
  - Webszerver válaszol
- HTTP protokoll szinten stateless protokoll
  - Cookie rendszer valójában pont a state tárolása
- Weboldalak típusai
  - Statikus: tartalom csak manuális adatszerkesztésre változik
  - Dinamikus: tartalom valamilyen kód végrehajtásának eredménye

10.2.2. Kapcsolat típusok

- HTTP egy TCP alapú protokoll, 80-as portot használ
- Nem-perzisztens: lekérdezett objektumonként külön-külön TCP kapcsolat
  - Ettől még futhat több lekérdezés párhuzamosan
  - Probléma: sok socket szükséges, OS limitációk felléphetnek
  - Válaszidő: $2 * \text{RTT} + \text{file-transmission-time}$
    - Kapcsolatot fel kell építeni először (három-utas kézfogás)
- Perzisztens: TCP kapcsolat nyitva marad amíg nem timeout-ol
  - Manapság ezt szokás használni
  - Válaszidő: $1 * \text{RTT} + \text{file-transmission-time}$
    - Ha már van egy nyitott TCP kapcsolat
  - HTTP/1.1 óta van lehetőség újabb kérdéseket küldeni, mielőtt megérkezne az előző kérésre a válasz
    - Így egyetlen TCP kapcsolat tökéletesen elég, nincs szükség párhuzamosan többre

10.2.3. Üzenetek, kérések, válaszok

- Üzenetek ember által olvasható szöveges formátumúak
- Kétféle üzenet: kérés és válasz
- Üzenetek formátuma:
  - Sorok \r\n karakterekkel vannak elválasztva
  - Első sor a kérés leírása vagy a válasz sikeressége/státusza
  - Utána header-ök (kulcs-érték) párok jönnek
  - Végül egy üres sor utána az üzenet teste (tartalma)
- Form-ok (űrlapok) tartalma enkódolható URL-be
  - Példa: `example.com/form?name=alice&value=bob`
- Kérés típusok:
  - GET: valami lekérdezése
  - POST, PUT, DELETE: valami feltöltése vagy törlése
  - HEAD: valami lekérdezése, az objektum átküldése nélkül
    - Pl. ellenőrizhető, hogy egy cache-elt képet újra le kell-e tölteni
  - stb.
- Válasz státusz kódok (válasz első sorában szerepel)
  - 200 OK: minden sikeres
  - 301 Moved Permamently: objektum máshol található
    - Válasz egy header-je tartalmazza, hogy hol
  - 400 Bad Request: kérést nem tudja a szerver feldolgozni
  - 404 Not Found: keresett objektum nem található
  - 505 HTTP Version Not Supported
  - stb.

10.2.4. Sütik (Cookies)

- Cél: felhasználó (böngésző) azonosítása
- Működés
  - Böngésző küld egy kérést egy szervernek cookie nélkül
  - Szerver válaszol, de azt is mondja, hogy ezen túl ezt a cookie-t használja a böngésző
    - Ehhez a szerver általában egy adatbázistól kér egy következő szabad azonosítót és el is menti az adatbázisba ezt az adatot
  - Böngésző ezen túl minden kérésébe belerakja (egy headerként)
    - El is menti a háttértárra
- Felhasználási területek
  - Session kezelés: bejelentkezés, bevásárlókosár
  - Beállítások: megjelenési téma, bármilyen beállítás
  - Felhasználók nyomon követése
  - Személyre szabott hirdetések szolgáltatása
- Privacy problémák
  - Beágyazott tartalmak is kezelhetnek cookie-kat
    - Hiszen ezek is normál HTTP lekérdezések
  - Így minden reklámon, minden Facebook "Share" gombon keresztül egy harmadik fél (pl. Facebook) is megtudja melyik oldalon jártunk

11. Gyakorlat jegyzet: Socket programozás

11.1. Python alapok

- Standard inputról olvasás: `input('Prompt:  ')`
  - File IO-ként használható: `sys.stdin`
- Parancssori paraméter: `sys.argv[1]` (ez az első paraméter)
- `bytes` típus (bájtok tömbje/sorozata)
  - Megadható így `b'Test'` vagy így `'Test'.encode()`
  - `b'ab' == 'ab'.encode()` és `b'ab'.decode() == 'ab'`
  - Indexelhető: `id, data = my_bytes[0:4], my_bytes[4:]`

11.1.1. Fájl műveletek

- `f = open('fájlnév', 'r')` és `f.close()`
  - r=read, a=append, w=write, b=binary
  - `with open('fájlnév', 'r') as f:  [...]`
- `for line in f` vagy `f.readLine()`
- `f.write('...')`
- `f.read(x)`: x bájt vagy karakter beolvasása
- `f.seek(offset, whence)`: `new_pos = ref_point + offset`
  - Whence: 0=fájl eleje (default); 1=pillanatnyi pos; 2=fájl vége

11.1.2. JSON

- `json.dump(data, file)` vagy `print(json.dumps(data))`
- `json.load(file)` vagy `json.loads(text)`
- Elemek iterálhatók, indexelhetők

11.1.3. `struct` csomag

- `values = (1, b'ab', 2.7)`
  `packer = struct.Struct('i 2s f')`
  `packed_data = packer.pack(*values)`
  `# Vagy: struct.pack('i 2s f', *values)`
- `unpacker = struct.Struct('i 2s f')`
  `unpacked_data = unpacker.unpack(data)`
  `# Vagy: struct.unpack('i 2s f', data)`
- `struct.calcsize('i 2s f') == packer.size`
  - Van internal padding (C-s), de csak `@` (default) byte order esetén
- Byte order: Native=`@,=` Little-endian=`<` Big-endian=`!,>`
- Formats: https://docs.python.org/3/library/struct.html#format-characters
- String kezelés
  - `struct.pack('7s', 'hello') == b'hello/x00/x00'`
  - `b'hello/x00/x00'.decode().strip('/x00') == 'hello'`

11.2. `socket` csomag

- `my_hostname = socket.gethostname()`
- `host_addr = socket.gethostbyname('www.example.org')`
  - `hname, aliases, addresses = socket.gethostbyname_ex(host)`
- `hostname, aliases, addresses = socket.gethostbyaddr(addr)`
  - Reverse lookup: akkor működik, ha cache-elve van
- Portszámhoz tartozó ismert szolgáltatás: `socket.getservbyport(port)`
- Endianness konverzió host/network között, pozitív számok esetén:
  - `htons()`, `htonl()`: host to network short / long
  - `ntohs()`, `ntohl()`: network to host short / long

11.3. Socket programozás (elsősorban TCP-vel)

- ```
  # Szerver oldal
  with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
      sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
      sock.bind(('localhost', 10000))  # port=10000
      sock.listen(1)  # Még el nem fogadott kapcsolatok max. száma
      conn, client_address = sock.accept()
      with conn:
          # [...]
  ```
  - Bind-ban 0-s port: random portot választ
  - Hoszt, amire a szerver bind-ol:
    - `localhost`, `127.0.0.1`: loopback interfész
    - `0.0.0.0`: minden interfész
- ```
  # Kliens oldal
  with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
      sock.connect(('localhost', 10000))
      # [...]
  ```
- ```
  # Kommunikáció
  my_bytes = conn.recv(max_byte_count_to_read)
  conn.sendall(my_bytes)
  ```
  - `recv` maximum annyi bájtot olvas: akár kevesebbet, de min. egyet
    - Fragmentálással órán nem foglalkoztunk
  - `sendall` helyett `send`: nem feltétlenül küld el minden bájtot
  - `sendall` több csomagot küld, ha a küldendő buffer túl nagy
- Hibakezelés: `try: [...] except socket.error as msg: [...]`

11.3.1. TCP helyett UDP

- `SOCK_STREAM` helyett `SOCK_DGRAM`
- Nem kell `listen` (eddig szerver oldalon kellett)
- Küldés: `sock.sendto(data, (addr, port))`
- Fogadás: `data, (addr, port) = sock.recvfrom(max_bytes)`
- Válasz tényleg a szervertől jött-e: `(addr,port)` ellenőrzése (nem elvárás)
  ```

11.3.2. Nem blokkoló socket; socket beállítások

- `sock.setblocking(True/False)` vagy `sock.settimeout(seconds/None)`
  - Mindkettő nem blokkolóvá teheti a socket-et
  - Timeout után exception-t kapunk
- Socket options
  - `sock.setsockopt(level, optname, value)`
  - Level értékek:
    - `SOL_SOCKET`: teljes socket
    - `IPPROTO_IP`: IP szintű
  - Optname értékek:
    - `SO_REUSEADDR`: port újrahasznosítása akkor is, ha nem szabad (`SOL_SOCKET` szintű és az érték 0 (hamis) vagy 1)
      - `bind` előtt kell beállítani (azaz szerver oldalon)
      - Hasznos pl. ha (hibásan) nem close-oljuk a socket-et
      - WSL-ben különösen fontos: valamiért egy nagyon hosszú packet timeout van, és a kernel a timeout kétszeresét megvárja, mielőtt törölné a bind-ot
  - Value értékek: sztring vagy egész szám, 0 általában hamis

11.3.3. `select.select`: több kliens kiszolgálása egy szálon

- `readable_list, _, _ = select.select([sock, sys.stdin], [], [])`
  - A "writeables"-t és "exceptionals"-t nem használjuk órán
- Kliens csatlakozása szerverhez:
  - A szerver `socket` benne lesz a `readable_list`-ben
  - Ekkor `accept()` és `select`-nek innentől azon `socket`-et is adjuk
- Kliens bontja a kapcsolatot:
  - Bekerül a kliens a `readable_list`-be
  - `recv` vagy `None`-t vagy 0 hosszút ad vissza
    - Ennek közös lekezelése: `bool(received_bytes)`
  - Ilyenkor le kell zárni a `socket`-et és `select`-nek ne adjuk át innentől

11.4. Checksum készítés

- CRC: `hex(binascii.crc32(b'Test'))` vagy `hex(zlib.crc32(b'Test'))`
- MD5: `hashlib.md5(b'Test').hexdigest()`
- SHA1: `hashlib.sha1(b'Test').hexdigest()`
- `hashlib`: lehet részletekben készíteni a hash-t
  - ```
    hasher = hashlib.sha1()
    hasher.update(b'Test')
    return hasher.hexdigest()
    ```
    ```