Számítógépes Hálózatok
2012

8. Hálózati réteg – Packet Forwarding, Routing,
Distance Vector Routing, Link State Routing, RIP, OSPF, IGRP

A hálózati réteg

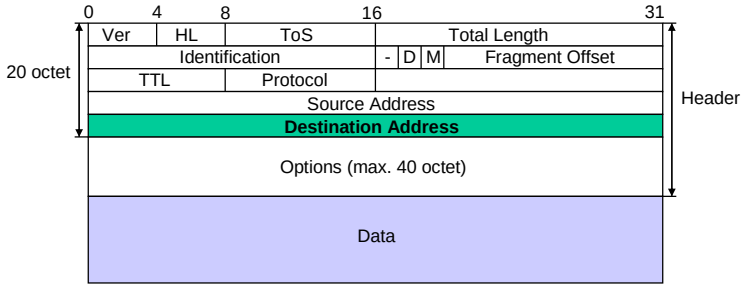
- Lokális hálózatokat összeköthetünk hub-okkal, switch-ekkel, bridge-ekkel az alacsonyabb retegekben
  - Hub(fizikai réteg): kollíziók száma nagyon gyorsan növekszik
  - Switch (Adatkapcsolati réteg):
    - Az útvonalakról a forgalom „megfigyelésével” gyűjt információt
    - Ismeretlen célcím esetén a broadcast problémákat okoz
  - Az Internet kb.10 Mio. lokális hálózatot tartalmaz...
- Nagy hálózatokban a csomagok továbbításához útvonal információk szükségesek.
- A hálózati réteg feladatai
  - Az útvonal információk felépítése (route detection)
  - A csomagok továbbítása (packet forwarding)
- Az Internet-Protokoll lényegében hálózati réteg protokoll

Routing-tábla és csomag továbbítás (packet forwarding)

- **IP-Routing-tábla**
  - Tartalmazza cél címekhez (destination) a következő számítógép (gateway) címét a hozzá vezető úton
  - A cél meghatározhat egy számítógépet vagy egy egész sub-net-et
  - Ezen kívül tartalmaz egy default-gateway-t
- **Packet forwarding** (korábban packet routing-nak nevezték)
  - **IP csomag (datagram)** tartalmazza a küldő IP címét és a cél IP címét
  - Amikor egy IP csomag megérkezik egy **router**hez:
    - Ha a cél IP cím = saját IP cím, akkor a csomagot kiszállítja
    - Ha a cél IP cím a routing-táblában van, továbbítja a megadott gateway-hez
    - Ha a cél IP-subnet a routing-táblában van, továbbítja a megadott gateway-nek
    - Egyébként továbbítja a default-gateway-nek

Internet Protocol IP

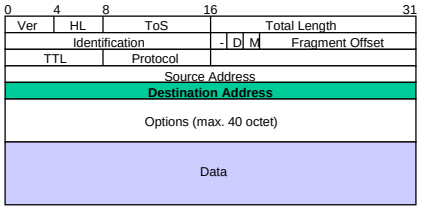
- Az adatok a küldőtől a cél-állomásig IP-csomagokban kerülnek átvitelre
- A csomagok fejléce tartalmazza a cél IP-címét
  - IPv4: 32 Bit-címek
  - IPv6: 128 Bit-címek



IPv4 csomag

Csomag továbbítás az Internet Protokollban

- IP-csomag (datagram) tartalmazza
  - TTL (Time-to-Live): hop-ok számát
  - Küldő IP címét
  - Cél IP címét
- Egy csomag kezelése a routerben
  - TTL = TTL - 1
  - Ha TTL ≠ 0 akkor packet-forwarding a routing-tábla alapján
  - Ha TTL = 0 vagy probléma lép fel a packet-forwarding-nél:
    - Töröljük a csomagot
    - Ha a csomag nem ICMP-csomag (Internet Control Message Protocol), akkor
      - Küldjünk ICMP-csomagot (TTL equals 0 during transit), melyben
        - Küldő IP címe = aktuális IP cím
        - Cél IP címe = az eredeti küldő IP címe



Statikus és dinamikus routing

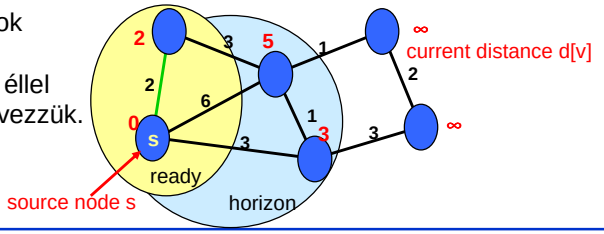
- Forwarding:
  - Csomagok továbbítása
- Routing:
  - Útvonalak meghatározása, azaz routing-tábla felépítése (rute detection)
- Statikus routing
  - A routing-táblát manuálisan építjük fel
  - Kis és statikus LAN-ok esetén értelmes
- Dinamikus routing
  - A routing-tábla felépítése és aktualizálása automatizált
  - Centalizált algoritmus, pl. Link State
    - Egy/minden állomásnak ismerni kell minden információt
  - Decentrális algoritmus, pl. Distance Vector
    - minden routeren lokálisan dolgozik, lokális információkkal

Legrövidebb utak fája – single source shortest paths

- Adott:
  - Egy irányított gráf G = (V,E), w : E → $\mathbf{R}_{\geq 0}$ nem negatív élsúlyokkal
  - Kezdő csomópont s ∈ V
- Legyen
  - **P útvonal súlya w(P) := $\sum_{e \in P} w(e)$** az élek súlyainak összege P-ben
  - **u és v távolsága** G-ben, u,v∈V, egy legrövidebb út súlya G-ben u és v között : **d(u,v) := min{ w(P) : P egy út u-tól v-hez G-ben}**.
- Keressük:
  - egy legrövidebb utat s kezdő csomóponttól minden más v ∈ V \ {s} csomóponthoz G-ben
  - Feltesszük, hogy minden v ∈ V \ {s} elérhető s-ből. Nem elérhető csomóponthoz nem létezhet legrövidebb út sem
- Megoldás:
  - Egy fa, melynek gyökere s és minden v ∈ V \ {s} csomóponthoz tartalmaz egy legrövidebb utat s-től v-hez G-ben

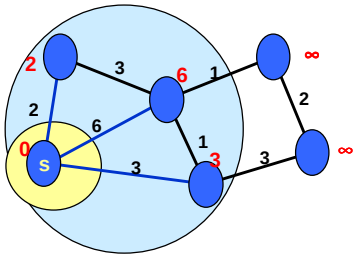
Dijkstra algoritmusa

- **Ötlet:** A legrövidebb utakat hosszuk szerint növekvő sorrendben számítjuk ki.
- Minden v ∈ V csomóponthoz kiszámítjuk a következő értékeket:
  - **d[v]**: egy legrövidebb út hossza s-től v-hez,
  - **pred[v]**: a v-t megelőző csomópont egy legrövidebb úton s-től v-hez.
- Az algoritmus végrehajtása után az élhalmaz **{ (pred[v],v) : v∈V \ {s} }** megadja egy legrövidebb utak fáját s gyökérrel G-ben.
- Egy v csomópontot „kész“-nek jelölünk: **ready[v] = true**, ha már meghatároztunk egy legrövidebb utat s-től v-hez (röv. legrövidebb s-v utat).
- A „nem kész“ csomópontok halmazát, amelyeket egy „kész“ csomópontból egy éllel elérünk, **horizont**-nak nevezzük.



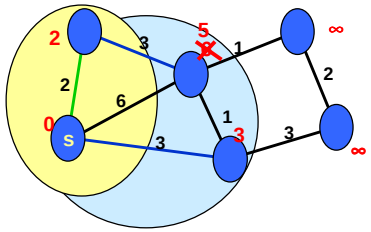
Dijkstra algoritmusa

- **Invariánsok:**
  - Minden horizont beli csomópontot egy **Q priority-queue**-ban tárolunk, úgy hogy minden v ∈ Q csomópontra a következő érvényes:
    - d[v] egy legrövidebb s-v út hossza mindazon utak között, melyek v-n kívül csak „kész“ csomópontokat tartalmaznak,
    - pred[v] a v-t megelőző csomópont egy ilyen úton,
    - v prioritása Q-ban d[v]
- **Inicializálás**
  - d[s]:=0, ready[s]:=true,
  - s minden v szomszédjára:
    - d[v]:=w(s,v), pred[v]:=s, ready[v]:=false,
    - Q.Insert(v,d[v]).
  - Minden v ∈ V \ {s} csomópontra:
    - d[v]:=∞, ready[v]:=false.



Dijkstra algoritmusa

- Az invariánsok megőrzése egy iteráció után
  - Minden lépésben egy új csomópont lesz „kész“, egy csomópont v minimális prioritással.
  - d[v] már tartalmazza a helyes értéket.
    Mivel v minimális prioritású csomópont, minden olyan s-v út súlya, amely „nem kész“ csomópontot is tartalmaz, legalább olyan nagy, mint annak az útnak a hossza, amit már megtaláltunk a csak „kész“ csomópontokat tartalmazó utak között.
  - Legyen Adj[v] := { u : (v,u) ∈ E }, v∈V, a v-hez adjacens csomópontok halmaza
  - minden u ∈ Adj[v], ha u ∈ Q,
    meg kell vizsgálni, hogy s-től u-hoz direkt v-ből egy rövidebb út vezet-e, mint azok az utak, amik csak v-től különböző „kész” csomópontot tartalmaznak.
    Ha igen, akkor aktualizáljuk
    - pred[u] := v és d[u] := d[v] + w(v,u),
    - csökkentsük u prioritását Q-ban.
  - minden u ∈ Adj[v], ha u ∉ Q és u „nem kész”:
    - pred[u] := v, d[u] := d[v] + w(v,u),
    - u-t be kell szúrni Q-ba d[u] prioritással.



Dijkstra algoritmusa

```
Dijkstra(G,s,w)
      Output: egy legrövidebb utak fája
      T=(V,E´) G-ben s gyökérrel
01    E´ := Ø;
02    ready[s] := true;
03    ready[v] := false;    ∀ v ∈ V \ {s};
04    d[s] := 0;
05    d[v] := ∞;      ∀ v ∈ V \ {s};

06    priority_queue Q;

07    forall v ∈ Adj[s] do
08       pred[v] := s;
09       d[v] := w(s,v);
10       Q.Insert(v,d[v]);
11    od
12    while Q ≠ Ø  do
13       v := Q.DeleteMin();
14       E´:= E´ U {(pred[v],v)};
15       ready[v] := true;
16       forall u ∈ Adj[v] do
17          if u ∈ Q and d[v] + w(v,u) < d[u]) then
18             pred[u] := v;
19             d[u] := d[v] + w(v,u);
20             Q.DecreasePriority(u,d[u]);
21          else if u ∉ Q and not ready[u] then
22             pred[u] := v;
23             d[u] := d[v] + w(v,u);
24             Q.Insert(u,d[u]);
25          fi
26       od
27    od
```


Dijkstra algoritmusa

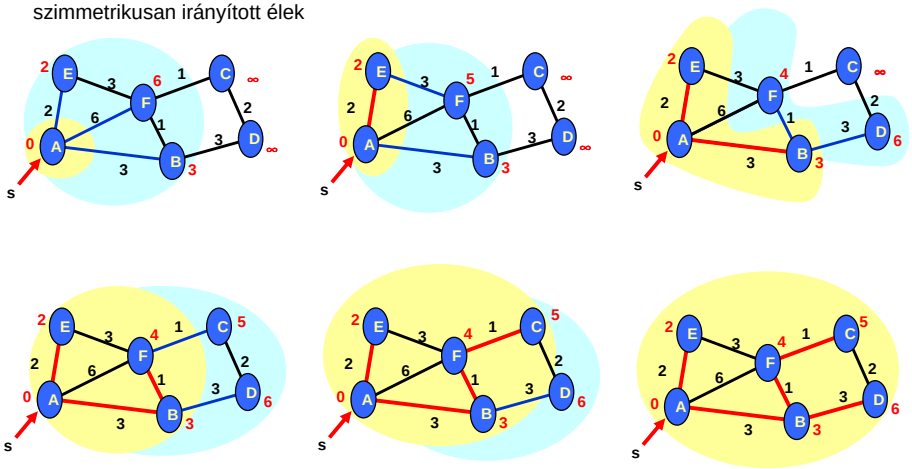
Futási idő (Fibonacci Heap-pel):

- # Q.Insert(): n (csomópontonként 1) -- O(n) idő
- # Q.DeleteMin(): n (csomópontonként 1) -- O(n log n) idő
- # Q.DecreasePriority(): ≤m (élenként ≤1) -- O(m) idő
- # A teszt a 17. és 21. sorban: m (élenként 1) -- O(m) idő
- Inicializálás: O(n) idő
- Összesen: O(n log n + m) idő

Tárigény: O(n+m)

Dijkstra: Példa

szimmetrikusan irányított élek



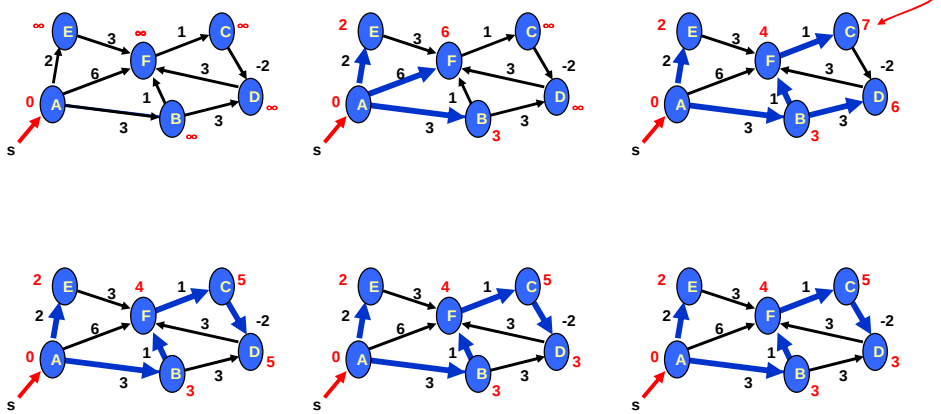
Bellman-Ford algoritmus

- Negatív élsúlyok esetén Dijkstra algoritmusa nem működik
- Bellman-Ford algoritmus (1957) megoldja a problémát O(|V| |E|) idő alatt.
- Dinamikus programozás: a k-adik iteráció után, k=1,…,|V|-1, minden v ∈ V:
  - ha d[v] ≠ ∞, akkor d[v] egy s-v út P_{sv} súlya és d[v] nem nagyobb mint egy legrövidebb s-v út súlya, amely ≤k élt tartalmaz
  - pred[v] = ∅ ha d[v] = ∞, egyébként pedig (pred[v],v) ∈ E az utolsó él a P_{sv} úton

```
Bellman-Ford(G,s,w)

01  forall v ∈ V  do
02     d[v] := ∞; pred[v] := ∅
03  d[s] := 0

04  for k := 1 to |V| – 1 do
05     forall (u,v) ∈ E do
06        if d[u] + w(u,v) < d[v] then
07           d[v] := d[u] + w(u,v)
08           pred[v] := u

09  forall  (u,v) ∈ E  do
10     if d[u] + w(u,v) < d[v] then
11        error „negatív súlyú ciklust találtunk"
```

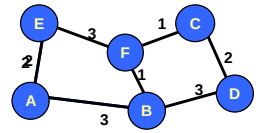

Bellman-Ford: Példa

Függ az élek feldolgozásának sorrendjétől



Distance Vector Routing Protokoll

- A Bellman-Ford algoritmusnak az **elosztott** változatát használja, azaz minden csomópont csak a direkt szomszédjaival kommunikál
- **Asszinkron** működés
  - A csomópontoknak nem ugyanabban a „körben" kell információkat cserélniük
- Minden router nyilvántart egy táblát minden lehetséges célhoz egy bejegyzéssel **(distance vector)**
  - egy bejegyzés tartalmazza
    - a legrövidebb út (becsült) költségét (delay, vagy #hops)
    - a következő csomópont címét ezen az úton (next hop)
- minden router ismeri a költséget a direkt szomszédaihoz
- Periodikusan elküldi a tábláját minden szomszédjának
- Amikor egy router megkapja a szomszéd tábláját aktualizálja a saját tábláját



Initial distance vector of A

A	cost	next hop
B	3	B
C	∞	-
D	∞	-
E	2	E
F	∞	-

Initial distance vector of B

B	cost	next hop
A	3	A
C	∞	-
D	3	D
E	∞	-
F	1	F

A's vector after A received B's vector

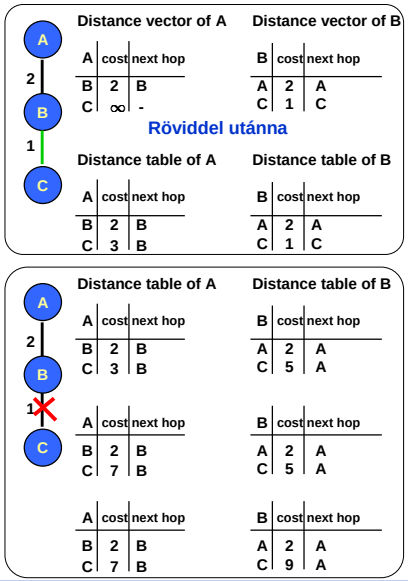
A	cost	next hop
B	3	B
C	∞	-
D	6	B
E	2	E
F	4	B

A's final distance vector

A	cost	next hop
B	3	B
C	5	B
D	6	B
E	2	E
F	4	B

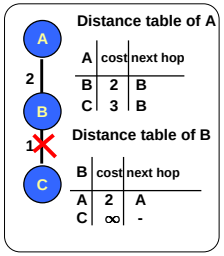
“Count to Infinity” Probléma

- „Jó hír” gyorsan terjed
  - Új kapcsolat létrejöttekor gyorsan aktualizálódnak a táblák
- „Rossz hír” lassan terjed
  - Kapcsolat kiesik
  - A szomszédok felváltva növelik a távolságokat
  - “Count to Infinity” Probléma
    - A és B nem tudja, hogy C nem elérhető (amíg a távolság el nem ér egy limitet, amit ∞-nek tekintenek)
  - Ciklusok keletkezhetnek



“Count to Infinity” Probléma

- Módosítások a Distance-Vector routing protokollokban
  - a ping-pong-ciklusokat (count to infinity) megakadályozásához
  - **split horizon**: olyan sorokat nem küld vissza a csomópont annak a szomszédjának, amit tőle „tanult”
    - a példában A nem küldi a (C,3,B) sort vissza B-nek, mert azt B-től kellett „tanulnia”
  - **split horizon with poison reverse**: negatív információt küld vissza
    - A pl. (C,∞) utat küldi vissza B-nek
- Mindkét módszer csak két csomópontból álló ciklust kerül el



Link State Protokoll

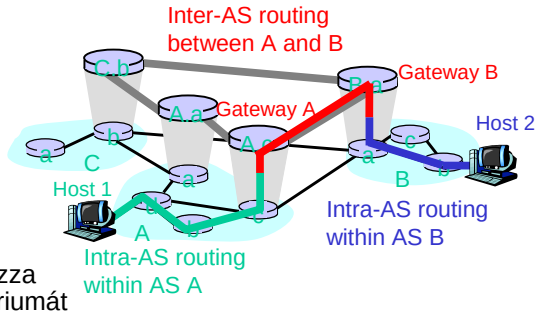
- Minden Link State router
  - tárolja a hálózat topológiáját
  - egy nem-elosztott legrövidebb utak algoritmust használ
- A routerek **Link State Packets (LSP)** által cserélnek ki információkat
- LSP tartalmazza
  - az LSP-t létrehozó r router IP címét
  - a költségét r minden direkt szomszédjához
  - sorozatszámot (SEQNO)
  - TTL (time to live) mezőt
- Megbízható elárasztás (Reliable Flooding)
  - minden csomópont aktuális LSP-jét tároljuk
  - továbbítjuk az LSP-ket minden szomszédos csomóponthoz
    - azon csomópont kivételével, amely az LSP-t felénk továbbította
  - A továbbításnál csökkentjük a TTL értékét
  - periodikusan létrehozunk egy új saját LSP-t
    - növekvő SEQNO-val

A „lapos” routing korlátai

- Link State Routing
  - O(D n) bejegyzésre van szükség, ahol n a routerek száma, D a maximális fok
  - Minden csomópont minden más csomópontnak el kell hogy küldje az információit
- Distance Vector
  - O(n) bejegyzés routerenként
  - Ciklusokat okozhat
  - Konvergencia ideje a hálózat méretével nő
- Az Internet több mint 10^6 routert tartalmaz
  - ezek a u.n. „lapos” routing módszerek nem használhatók az egész Internetre
- Megoldás:
  - Hierarchikus routing

Autonomous Systems (AS), Intra-AS és Inter-AS routing

- Autonomous Systems (AS)
  - Egy két szintű modellt ad a routinghoz az Interneten
  - Példa AS-re: elte.hu
- Intra-AS-routing
  - routing az AS-en belül
  - pl. RIP, OSPF, IGRP, ...
- Inter-AS-routing
  - Kapcsolódási pont: **átjáró (gateway)**
  - teljesen decentrális routing
  - Mindeki saját maga határozza meg az optimalizálási kritériumát
  - pl. BGP, EGP (korábban)



Intra-AS routing: RIP Routing Information Protocol (RFC 1058)

- Distance Vector algoritmus
  - távolság metrika = hop szám (linkek száma)
- A távolság vektorokat (distance vector) minden router minden 30s Response-üzenettel (advertisement) adja át a szomszédjának
- A szomszédok szintén egy új advertisement-et küldenek ha a táblájuk ezáltal megváltozott
- Minden Advertisement-ben
  - célhálózathoz hirdetik meg az utakat UDP-vel (UDP port 520)
- Ha 180s-ig nem kap a router advertisement-et egy szomszédjától
  - az utakat a szomszédon keresztül érvénytelennek deklarálja
  - új Advertisment-eket küld a szomszédainak
- Hogy elkerülje a ping-pong-ciklusokat (count to infinity), „split horizon with poison reverse” módszert használ
- Végtelen távolság = 16 Hop (limitet szab a hálózat átmérőjére)

Intra-AS routing: OSPF routing (Open Shortest Path First)

- “open” = nyilvánosan rendelkezésre álló
- Link-State algoritmus
  - LS csomagok terjesztése
  - a topológiát minden csomópontban tárolja
  - az útvonalakat Dijkstra algoritmusával számítja ki
- OSPF-advertisment
  - TCP-vel, növeli a biztonságot (security)
  - az egész AS-be elárasztja (broadcast)
  - több egyenlő költségű útvonal lehetséges

Intra-AS routing -- Hierarchikus OSPF

- Nagy hálózatokhoz két hierarchia szint:
  - Lokális terület és gerinchálózat (backbone)
    - Lokális: Link-state advertisement
    - Minden csomópont csak az irányt számítja ki más lokális területek hálózataihoz
- Local Area Border Router:
  - A saját lokális területeik távolságait foglalják össze
  - Ezeket más Lokal Area Border Router-eknek meghirdetik (advertisement)
- Backbone Routers
  - OSPF protokollt használnak a gerinchálózatra korlátozva
- Boundary Routers:
  - Más AS-ekkel kapcsolnak össze

Intra-AS routing: IGRP (Interior Gateway Routing Protocol)

- CISCO-Protokoll (1980-as évek közepe), a RIP utódja
- Distance-Vector-Protokoll, mint a RIP
  - Holddown time
  - Split horizon
  - Poison reverse
- Különböző költség metrikákat támogat
  - Delay, Bandwidth, Reliability, Load, stb…
- TCP-t használ a routing információk kicseréléséhez