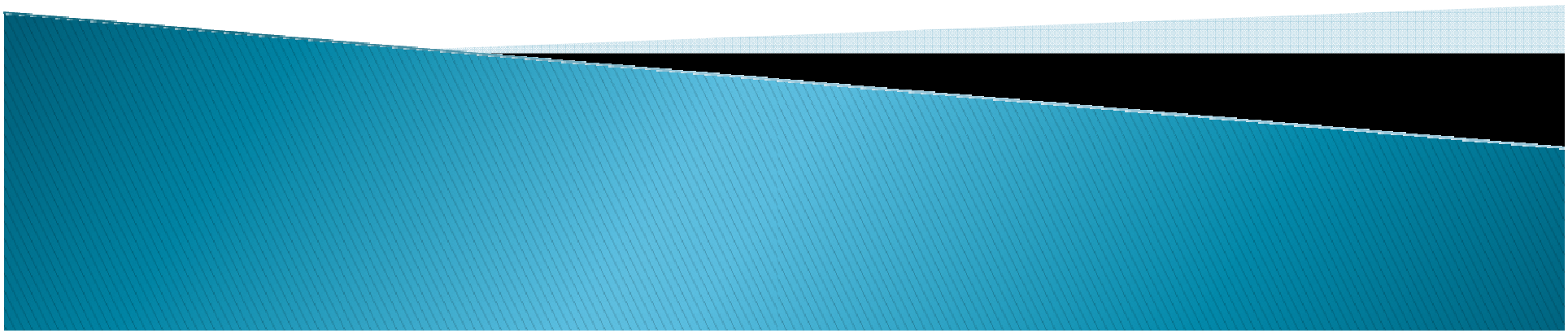
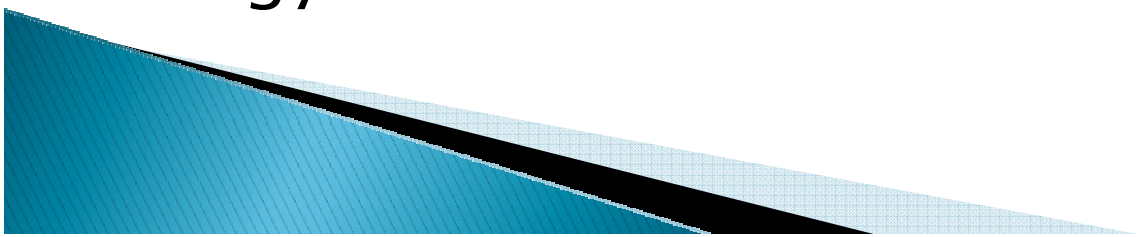
Hálózati időszinkronizáció referencia broadcasttal

Fine-Grained Network Time Synchronization using Reference Broadcast



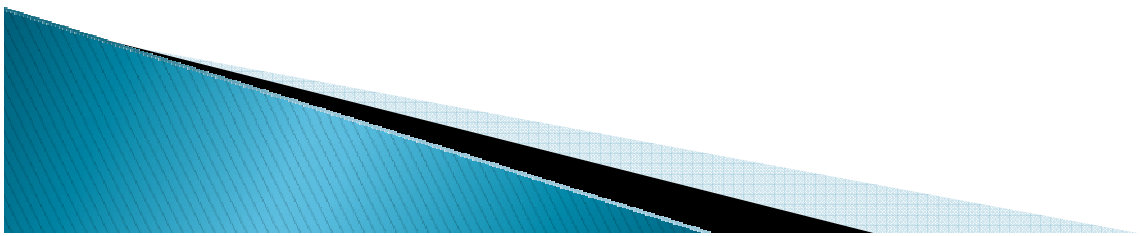
Mi okozza az órák közötti eltérést?

- Ofszet
  - Az indítás óta eltelt időt mérik
- Az ofszet változása: skew
  - Az órák „sebességének” különbsége
  - Oka: Az óra az oszcillátor pontatlanságát örökli
    - Gyártási pontatlanság
    - Stabilitás
      - Rövid távú: hőmérséklet, tápfeszültség
      - Hosszú távú: öregedés
- Ha ritkán akarunk szinkronizálni, legalább az ofszetet és konstans skew-t érdemes figyelembevenni



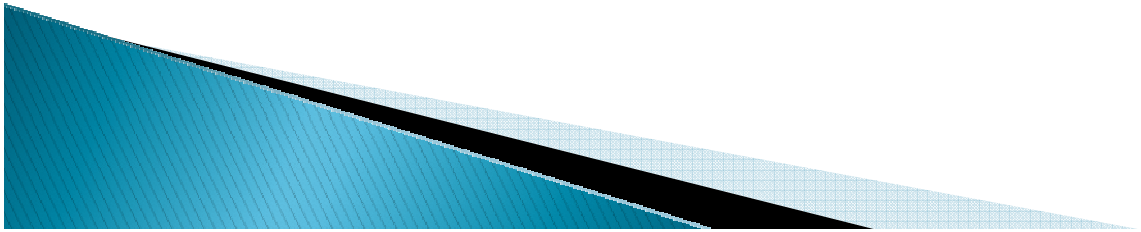
Hagyományos megoldások

- Szinkronizáló szerver elküldi a pontos időt a kliensnek bizonyos időközönként
- Javítás: A kliens kérésére küldi a szerver a pontos időt
  - A kliens a kérés és a válasz között eltelt időből tudja becsülni a csomag késleltetését
- Javítás: Többször megismételjük a fentit
  - A legrövidebb idejűt vesszük figyelembe



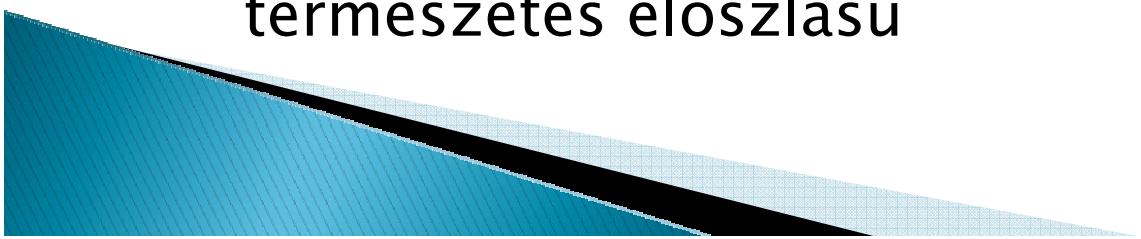
Mi a probléma a korábbi módszerekkel?

- Bizonyos alkalmazásokhoz pontosabb szinkron kell
  - Helymeghatározás hang késleltetésének mérésével
  - TDMA
- Sokszor dedikált szerverre van szükség
  - Nem mindig elérhető



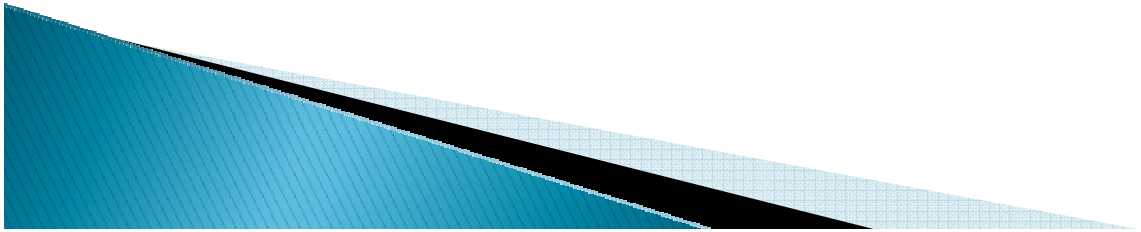
Késleltetések

- Küldési idő
  - Alkalmazási -> Közeghozzáférési réteg
- Csatorna hozzáférési idő
  - Közeghozzáférési réteg
- Terjedési idő
  - Fizikai réteg
  - 10ns-os nagyságrend
- Vételi idő
  - Közeghozzáférési -> Alkalmazási réteg
  - A Berkeley mote-okon végzett mérések szerint természetes eloszlású



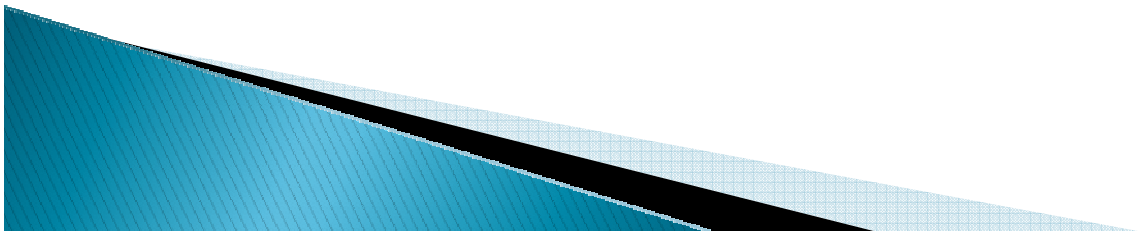
RBS – legegyszerűbb eset

- Egy adó broadcastol egy csomagot
- A csomag vételi pillanatát a vevők referencia pontnak tekintik
- A vevők megadják egymásnak mit mutatott saját órájuk a referenciaidőben
- A küldési és csatorna hozzáférési idő nem számít!
- A pontosság függ:
  - A vevők számától
  - A vételi idő bizonytalanságától
  - A vevők óráinak „sebességétől” (skew)



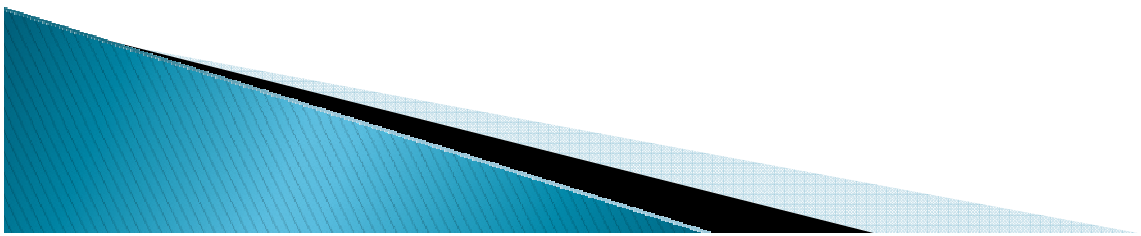
RBS – A pontosság javítása

- A vételi bizonytalanság normális eloszlású
  - Ha többször megismételjük a küldést, pontosabbá tehetjük
  - Két node közötti ofszet az összetartozó referenciapontjaik közötti időkülönbség átlaga lesz
- Skew
  - Több referenciapontból lineáris regresszió
  - Időben állandó ofszetet és sebességkülönbséget feltételez
    - Mindig újraszámoljuk az előző T időre



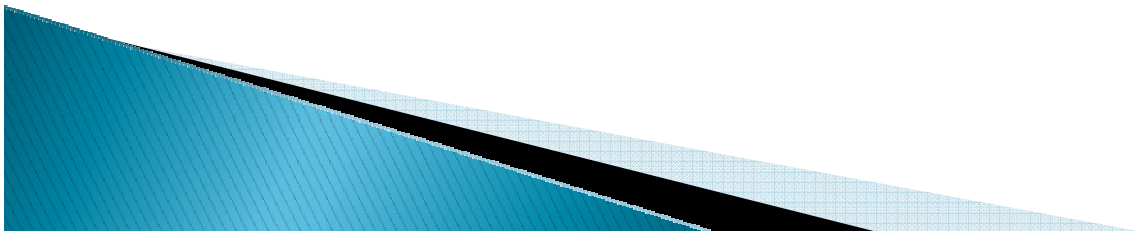
Megvalósítás: Berkeley Mote

- 2 µs pontosságú óra, 5 adó, utólagos feldolgozás
- Hiba négyzetes középértéke: 11,2 µs
- Mote-ok hálózati interface-nek használva két Linux OS között
  - Módosított Linux kernel: Időbélyeggel látja el a soros porton érkező csomagokat
  - Hiba: 7,4 µs



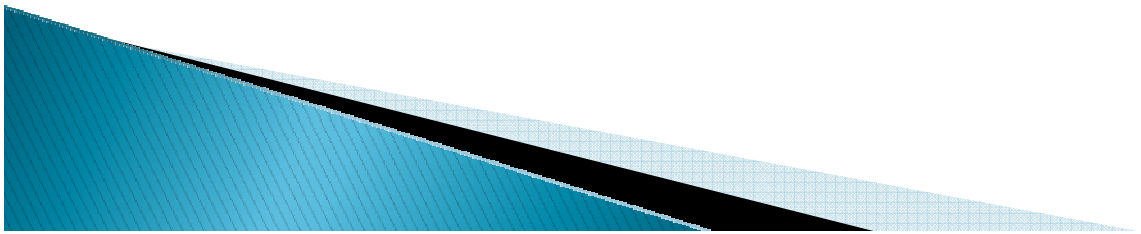
Megvalósítás: 802.11b

- Cél: RBS összehasonlítása NTP-vel
- Hardware: Compaq IPAQ
- Software: Linux v2.4
- Felbontóképesség: 1 µs
- Időbélyegzés a felhasználói programban
- Az eszközök adók és vevők is
- Csomagok UDP datagramban



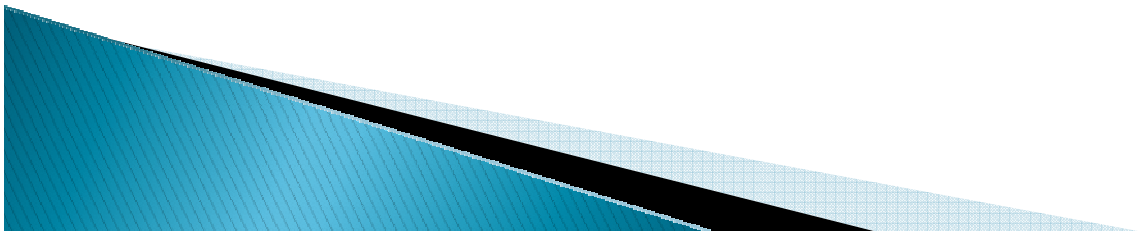
Megvalósítás: 802.11b

- 10 másodpercenként adnak
- A vevő visszaküldi mikor kapta meg a broadcastot
- Az adó kiszámolja az összes vevő közti páronkénti összefüggést
- Lineáris regresszió paraméterei:
  - Legkisebb négyzetek módszerével
  - Az utolsó 30 adatból
  - A „nagyon kilógó” adatokat nem veszi figyelembe



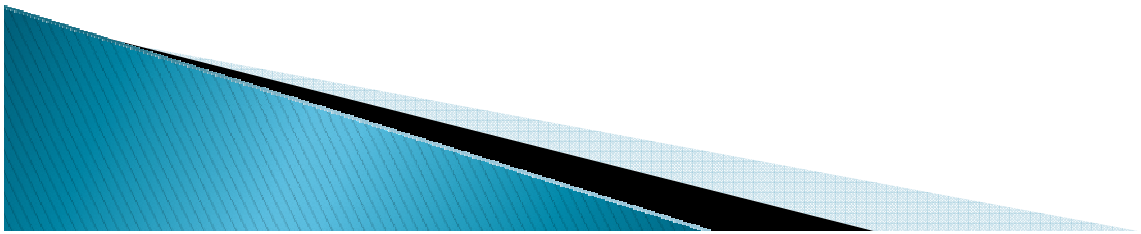
Megvalósítás: 802.11b

- A résztvevő algoritmusok:
  - RBS
  - NTP
  - NTP-Offset
    - Az NTP nem változtatja „túl gyorsan” az időt, ezt a tulajdonságát kiiktatták
- Tesztek:
  - Alacsony forgalom: Csak a szinkronizációhoz szükséges
  - Magas forgalom: Két, tesztben nem résztvevő IPAQ véletlen méretű UDP csomagokat cserél



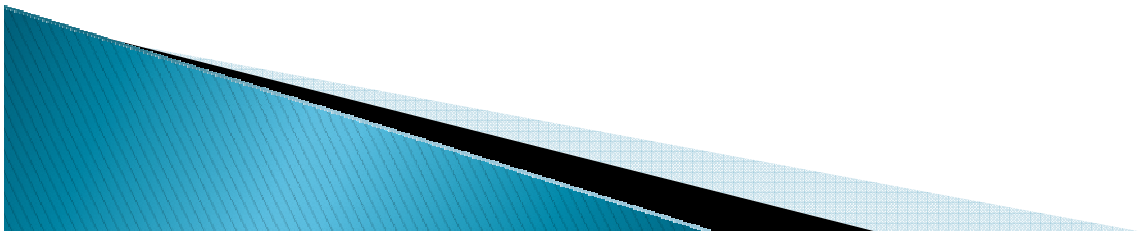
Megvalósítás: 802.11b: eredmények

- Alacsony terhelés:
  - NTP: átlag: 51,18±53,30 µs, 95% jobb, mint 131,2 µs
  - RBS: átlag: 6,29±6,45 µs, 95% jobb, mint 20,53 µs
- Magas terhelés:
  - NTP: átlag: 1542 µs, 95% jobb, mint 3889 µs
  - RBS: átlag: 8,44 µs, 95% jobb, mint 28,53 µs
- NTP-Offset szinte mindig rosszabb, mint az NTP
  - Valószínűleg a csomagok különböző késleltetése miatt



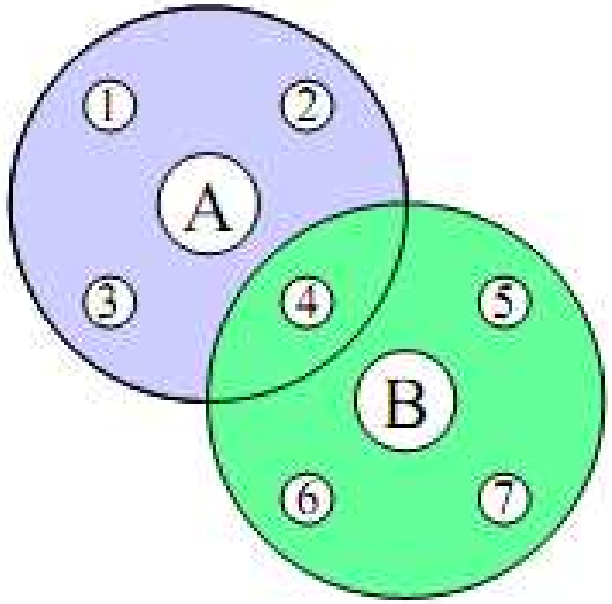
Megvalósítás: 802.11b, kernel időbélyegzéssel

- Az RBS pontatlanságának fő oka a vételi késleltetés
- Ezt minimalizálhatjuk kernel időbélyegzéssel
- Eredmény: 1,85±1,28 µs
  - A korlát valószínűleg az 1 µs felbontóképesség

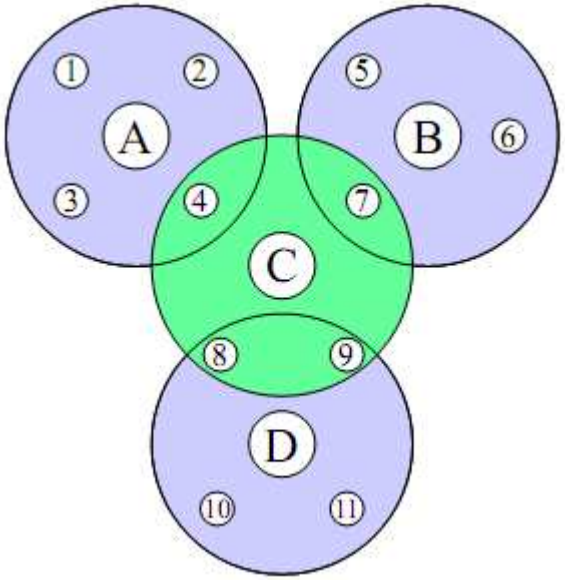


Multi-Hop szinkronizáció

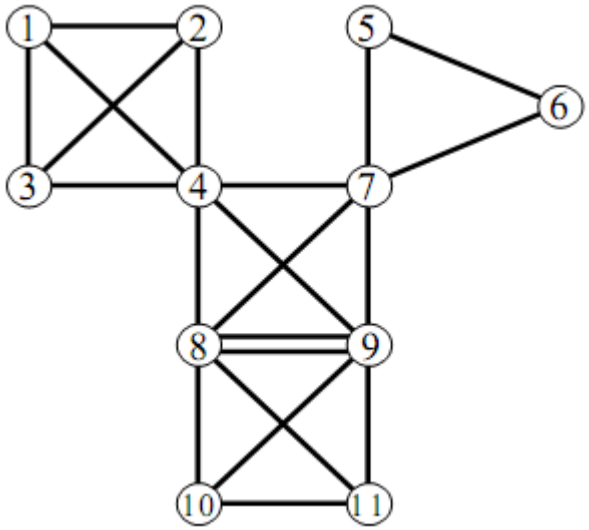
- A és B adók, a többi vevő
- 4 hallja A és B adását is
- Ebből ki tudja számolni A és B ideje közötti ofszetet és skew-t



Routing

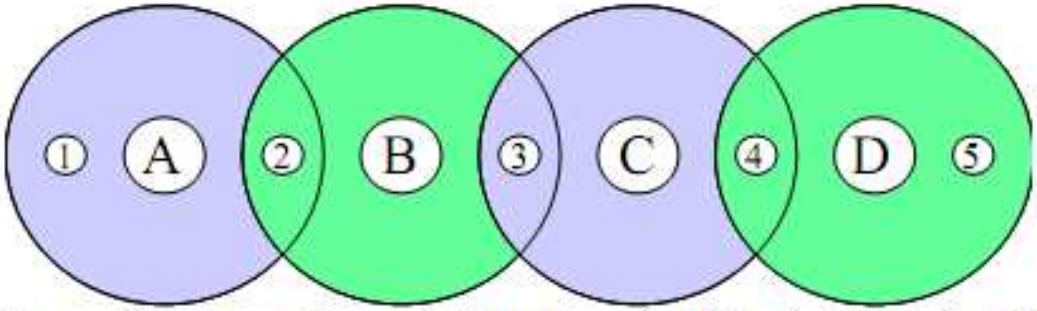


- $E_i(R_j)$: i esemény ideje j órája szerint
- $E_1(R_{10})$ számítása:
  $E_1(R_1) \rightarrow E_1(R_4) \rightarrow E_1(R_8) \rightarrow E_1(R_{10})$
- Kereshetünk legrövidebb utat a gráfban
  - De ismernünk kell a teljes gráfot
- Bármilyen létező routing módszert használhatunk
  - Minden lépésnél konvertáljuk az időt



Multi-Hop szinkronizáció

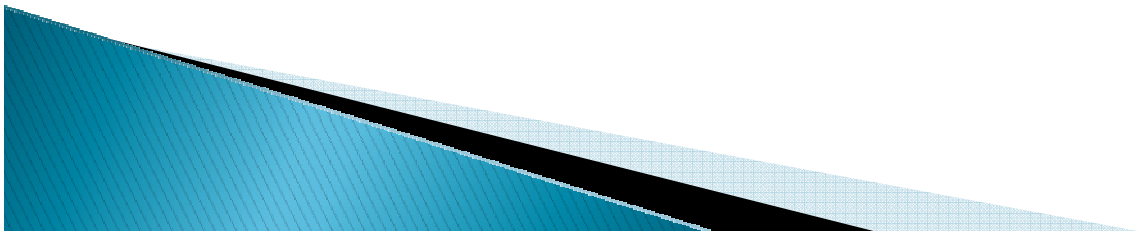
- A szinkronizáció hibája normális eloszlású
- A páronkénti szinkronizációk függetlenek
  - Ha az átlagos hiba ugrásonként σ, akkor n ugrás után $\sigma\sqrt{n}$



Hops	Mean Error	Std Dev	50%	95%	99%
1	1.85	1.28	1.60	4.51	5.85
2	2.73	1.91	2.41	6.49	7.74
3	2.73	2.42	2.22	6.93	9.92
4	3.68	2.57	3.18	8.26	10.70

Szinkronizálás referenciához

- Referencia: pl UTC
  - GPS segítségével
  - DCF77 segítségével
- Tipikusan másodpercenként jeleznek
  - Ez használható RBS adóként
  - A vevő az az eszköz, amire a hardware-t ráépítettük



RBS összefoglalva

- A csomagkésleltetésekből a legkiszámíthatatlanabbakra érzéketlen
  - Ami marad, az normális eloszlású, ezért több broadcasttal kiszámíthatóvá tehető
- Több broadcasttal a skewt is megállapíthatjuk
  - Így ritkábban kell szinkronizálni
- A multi-hop szinkronizáció jól skálázható, az ugrások számával csak gyökösen nő a hiba
- Ha nincs referencia szerver, akkor is ki tudunk alakítani egy a saját hálózatban globális időt