

Számítógépes Hálózatok

10. Előadás: Szállítói réteg++

Based on slides from **Zoltán Ács** ELTE and D. Choffnes Northeastern U., Philippa Gill from StonyBrook University , Revised Spring 2016 by S. Laki

Mi az a torlódás?

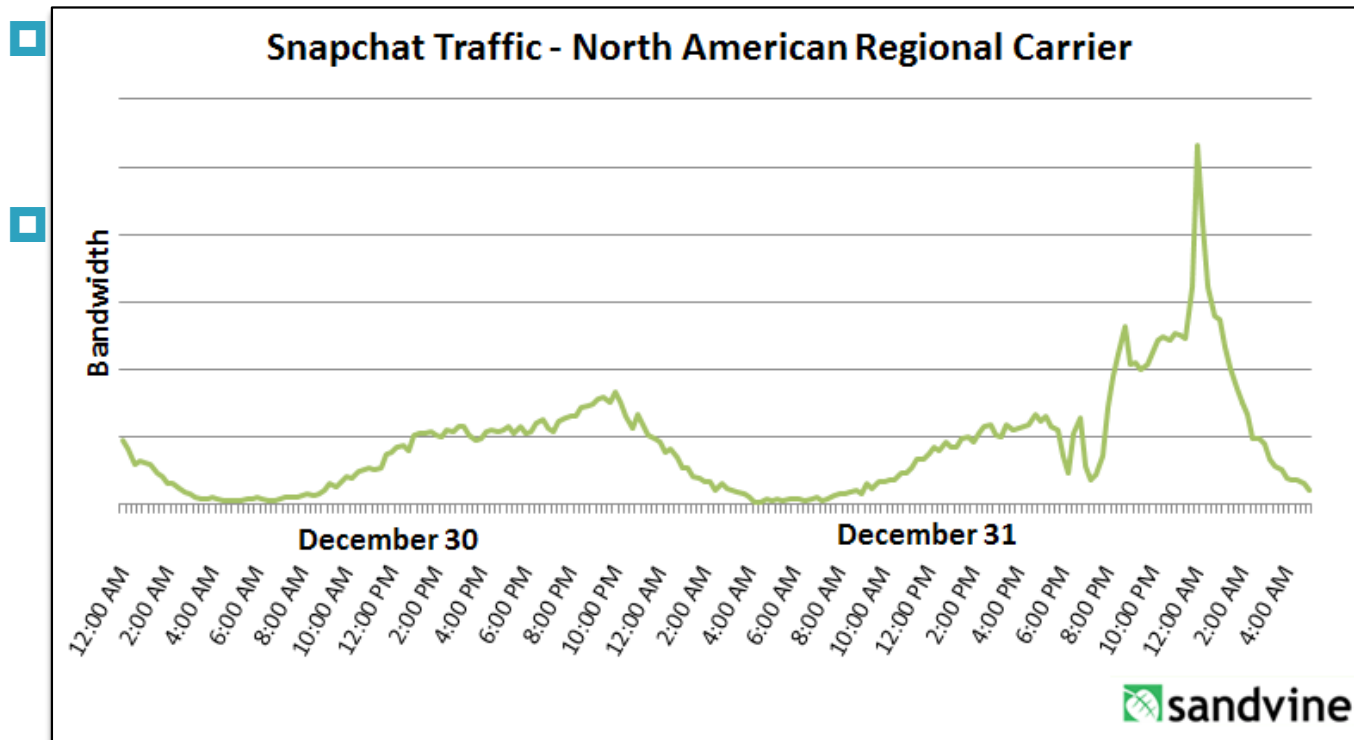
2

- A hálózat terhelése nagyobb, mint a kapacitása
 - ▣ A kapacitás nem egyenletes a hálózatban
 - Modem vs. Cellular vs. Cable vs. Fiber Optics
 - ▣ Számos folyam verseng a sávszélességért
 - otthoni kábel modem vs. corporate datacenter
 - ▣ A terhelés időben nem egyenletes
 - Vasárnap este 10:00 = Bittorrent Game of Thrones

Mi az a torlódás?

3

- A hálózat terhelése nagyobb, mint a kapacitása
 - ▣ A kapacitás nem egyenletes a hálózatban
 - Modem vs. Cellular vs. Cable vs. Fiber Optics



Miért rossz a torlódás?

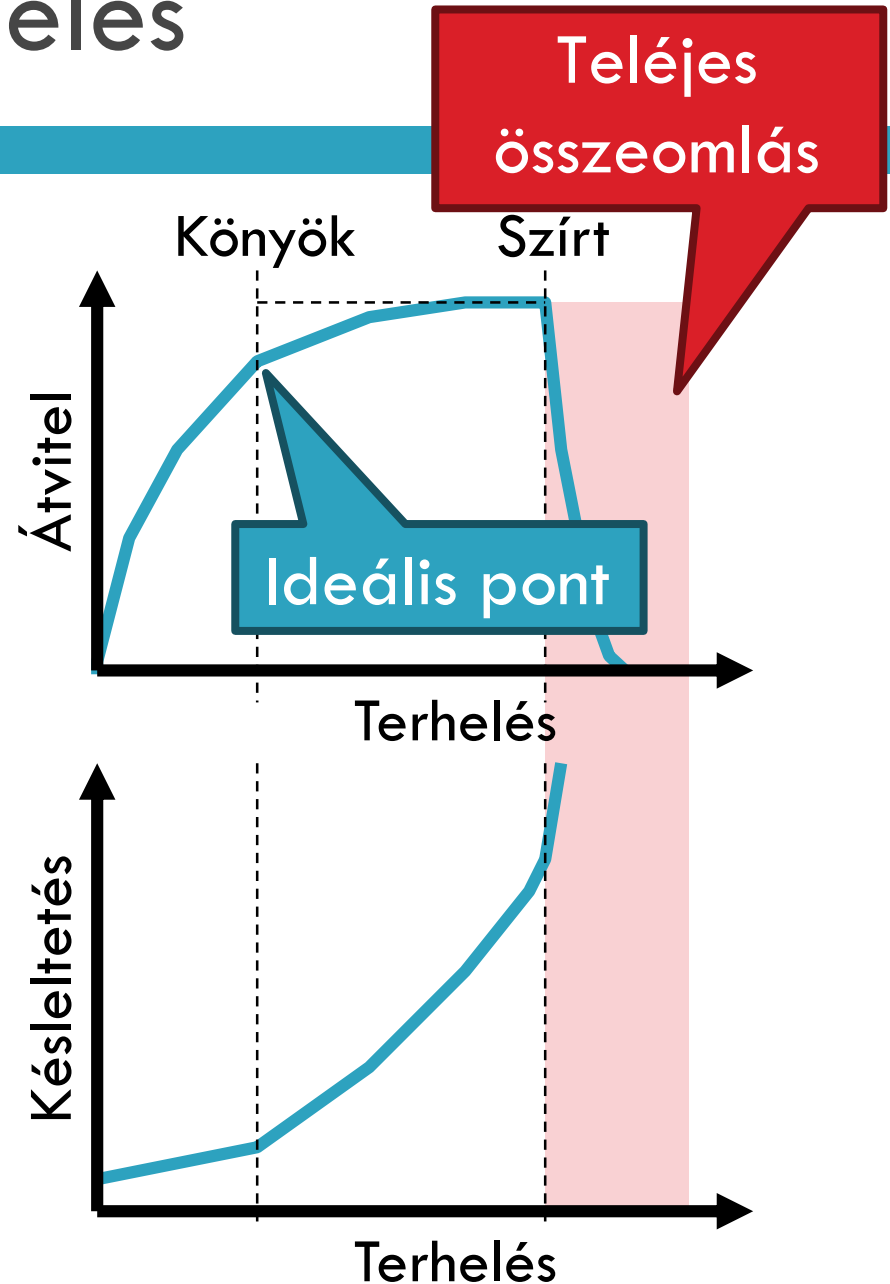
4

- ❑ Csomagvesztést eredményez
 - ▣ A routerek véges memóriával (puffer) rendelkeznek
 - ▣ Önhasonló Internet forgalom, nincs puffer, amiben ne okozna csomagvesztést
 - ▣ Ahogy a routerek puffere elkezd telítődni, csomagokat kezd eldobni... (RED)
- ❑ Gyakorlati következmények
 - ▣ A routerek sorai telítődnek, **megnövekedett késleltetés**
 - ▣ Sáv szélesség pazarlása az **újraküldések miatt**
 - ▣ Alacsony hálózati átvitel (goodput)

Megnövekedett terhelés

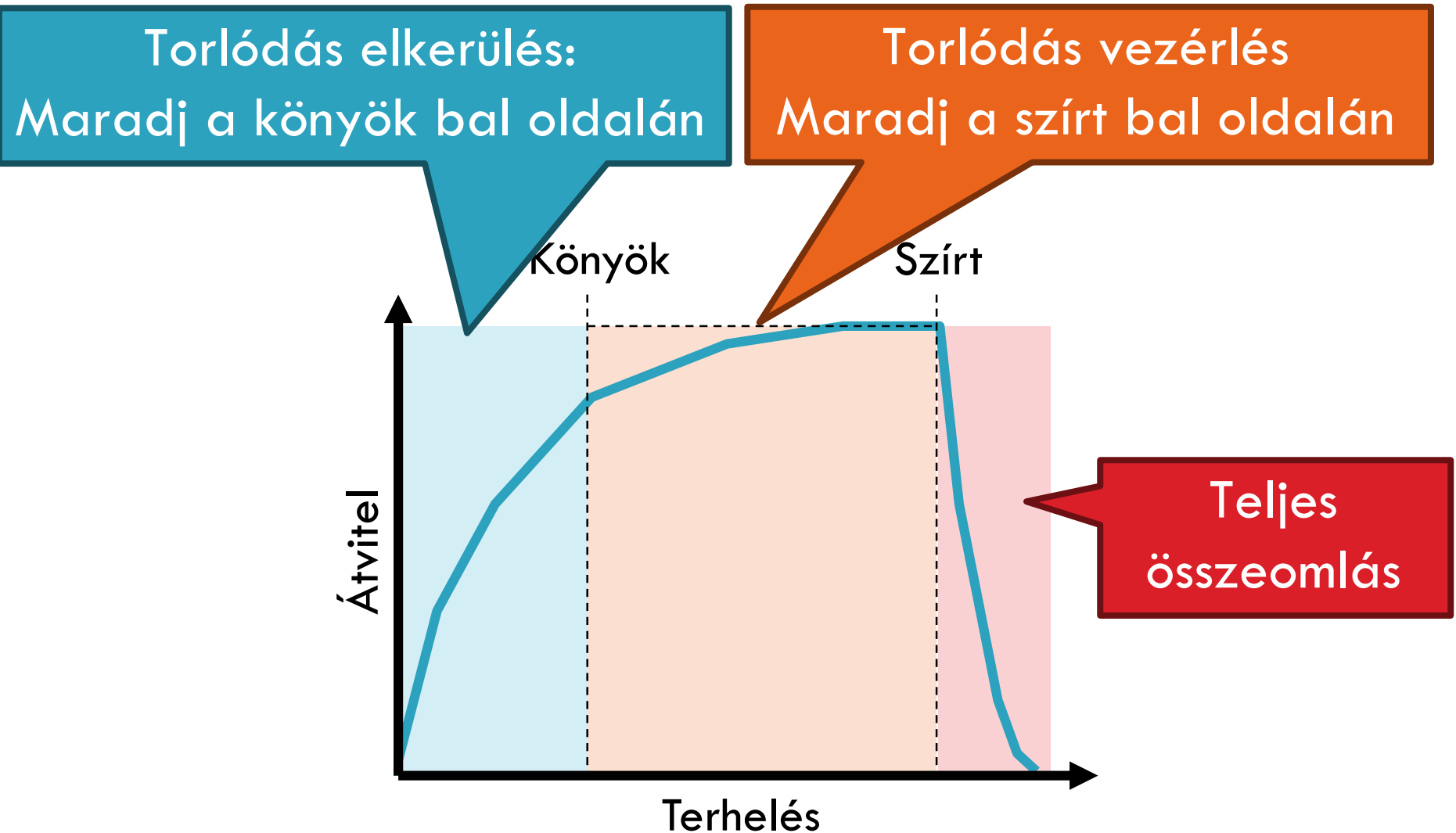
5

- ❑ Könyök („knee”)– a pont, ami után
 - ▣ Az átvitel szinte alig nő
 - ▣ Késleltetés viszont gyorsan emelkedik
- ❑ Egy egyszerű sorban (M/M/1)
 - ▣ Késleltetés = $1 / (1 - \text{utilization})$
- ❑ Szírt („cliff”) – a pont, ami után
 - ▣ Átvitel lényegében leesik 0-ra
 - ▣ A késleltetés pedig $\rightarrow \infty$



Torlódás vezérlés vs torlódás elkerülés

6



Advertised Window

Meghirdetett ablak, újragondolva

7

- ❑ Megoldja-e a torlódás problémáját a TCP esetén a meghirdetett ablak használata?

NEM

- ❑ Ez az ablak csak a fogadót védi a túlterheléstől
- ❑ Egy kellően gyors fogadó kimaxolhatja ezt az ablakot
 - ▣ Mi van, ha a hálózat lassabb, mint a fogadó?
 - ▣ Mi van, ha vannak konkurens folyamatok is?
- ❑ Következmények
 - ▣ Az ablak méret határozza meg a küldési rátát
 - ▣ Az ablaknak állíthatónak kell lennie, hogy elkerüljük a torlódás miatti teljes összeomlást...

Általános megoldások

8

- ❑ Ne csináljunk semmit, küldjük a csomagokat megkülönböztetés nélkül
 - ▣ Nagy csomagvesztés, jósolhatatlan teljesítmény
 - ▣ Teljes összeomláshoz vezethet
- ❑ Erőforrás foglalás
 - ▣ Folyamokhoz előre sáv szélességet allokálunk
 - ▣ Csomagküldés előtt egy tárgyalási szakaszra is szükség van
 - ▣ Hálózati támogatás kell hozzá
- ❑ Dinamikus beállítás
 - ▣ Próbák használata a torlódási szint megbecsléséhez
 - ▣ Gyorsítás, ha torlódási szint alacsony
 - ▣ Lassítás, amint nő a torlódás
 - ▣ Nem rendezett dinamika, elosztott koordináció

TCP Torlódásvezérlés

9

- ❑ Minden TCP kapcsolat rendelkezik egy ablakkal
 - ❑ A nem-nyugtázott csomagok számát vezérli
- ❑ Küldési ráta $\sim \text{window} / \text{RTT}$
- ❑ Ötlet: ablak méretének változtatása a küldési ráta vezérléséhez
- ❑ Vezessünk be egy **torlódási ablakot (congestion window)** a küldő oldalon
 - ❑ Torlódás vezérlés egy küldő oldali probléma
 - ❑ Jelölése: cwnd

Két fő komponens

10

1. Torlódás detektálás

- ▣ Eldobott csomag egy megbízható jel
 - Késleltetés alapú megoldások – nehéz és kockázatos
- ▣ Hogyan detektáljuk a csomag eldobását? Nyugtával
 - Időkorlát lejár ACK fogadása nélkül
 - Számos duplikált ACK jön be sorban (később lesz róla szó)

2. Ráta beállító algoritmus

- ▣ *cwnd* módosítása
- ▣ Sáv szélesség próba
- ▣ Válasz lépés a torlódásra

Ráta vezérlés

11

- Tudjuk, hogy a TCP ACK ütemezett
 - ▣ Torlódás = késleltetés = hosszú várakozás a nyugták között
 - ▣ Nincs torlódás = alacsony késleltetés = gyors ACK
- Alapvető algoritmus
 - ▣ ACK fogadása esetén: növeljük a *cwnd* ablakot
 - Adat leszállítva, valószínűleg gyorsabban is küldhetünk
 - *cwnd* növekedése arányos az RTT-vel
 - ▣ Csomagvesztés esetén: csökkentjük a *cwnd* ablakot
 - Adat elveszett, torlódásnak kell lennie a hálózatban
- Kérdés: milyen függvényt használjuk a növeléshez és csökkentéshez? !!!!

Torlódás vezérlés megvalósítása

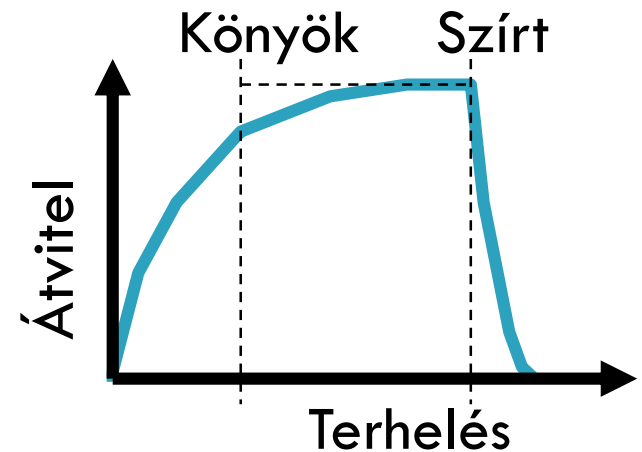
12

- Három változót kell nyilvántartani:
 - ▣ *cwnd*: torlódási ablak
 - ▣ *adv_wnd*: a fogadó meghirdetett ablaka
 - ▣ *ssthresh*: vágási érték (a *cwnd* frissítésére használjuk)
- Küldésnél használjuk: $wnd = \min(cwnd, adv_wnd)$
- A torlódás vezérlés két fázisa:
 1. Lassú indulás („Slow start”) ($cwnd < ssthresh$)
 - Az ún. bottleneck (legsűkebb) sáv szélesség meghatározása a cél.
 2. Torlódás elkerülés ($cwnd \geq ssthresh$)
 - AIMD – Additive Increase Multiplicative Decrease

Lassú indulás - Slow Start

13

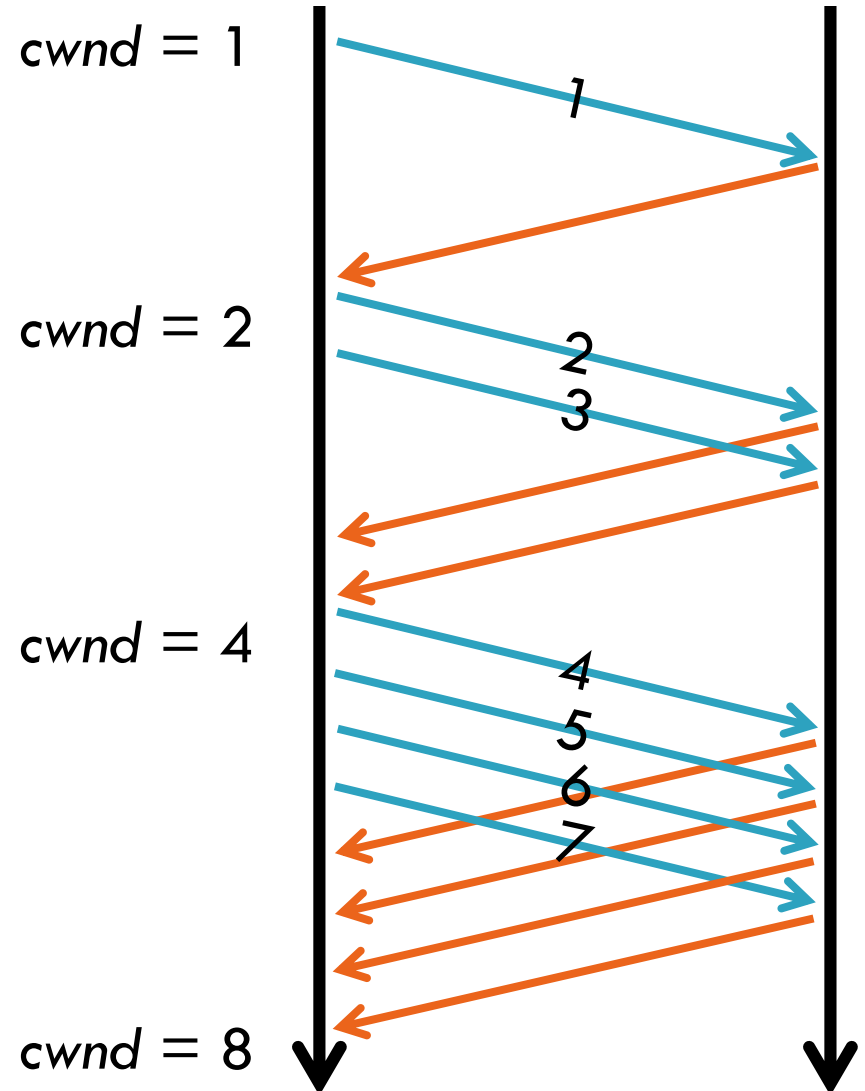
- ❑ Cél, hogy gyorsan elérjük a könyök pontot
- ❑ Egy kapcsolat kezdetén (vagy újraindításakor)
 - ▣ $cwnd = 1$
 - ▣ $ssthresh = adv_wnd$
 - ▣ Minden nyugtázott szegmensre: $cwnd++$
- ❑ Egészen addig amíg
 - ▣ El nem érjük az $ssthresh$ értéket
 - ▣ Vagy csomagvesztés nem történik
- ❑ A Slow Start valójában nem lassú
 - ▣ $cwnd$ exponenciálisan nő



Slow Start példa

14

- $cwnd$ gyorsan nő
- Lelassul, amikor...
 - ▣ $cwnd \geq ssthresh$
 - ▣ Vagy csomagvesztés történik



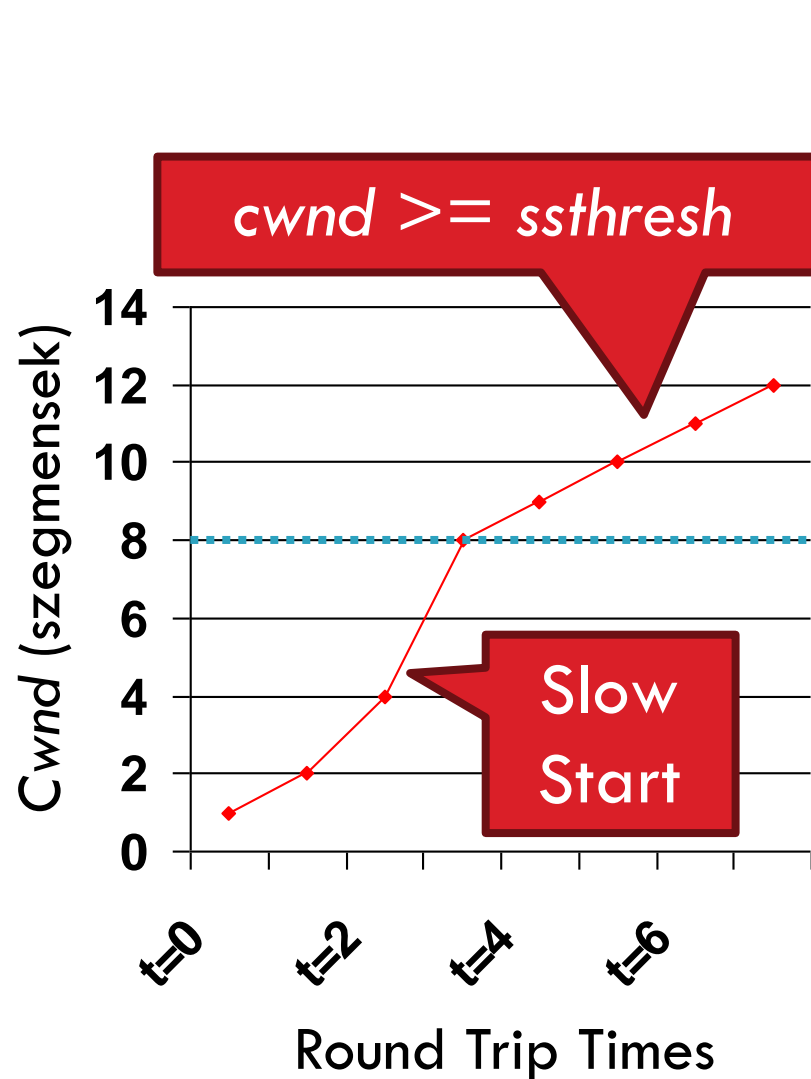
Torlódás elkerülés

15

- Additive Increase Multiplicative Decrease (AIMD) mód
- *ssthresh* valójában egy alsóbecslés a könyök pontra
- **Ha** $cwnd \geq ssthresh$ **akkor**
Minden nyugtázott szegmens alkalmával
növeljük a *cwnd* értékét $(1/cwnd)$ -vel
(azaz $cwnd += 1/cwnd$).
- Azaz a *cwnd* eggyel nő, ha minden csomag nyugtázva lett.

Torlódás elkerülés példa

16



$cwnd = 1$

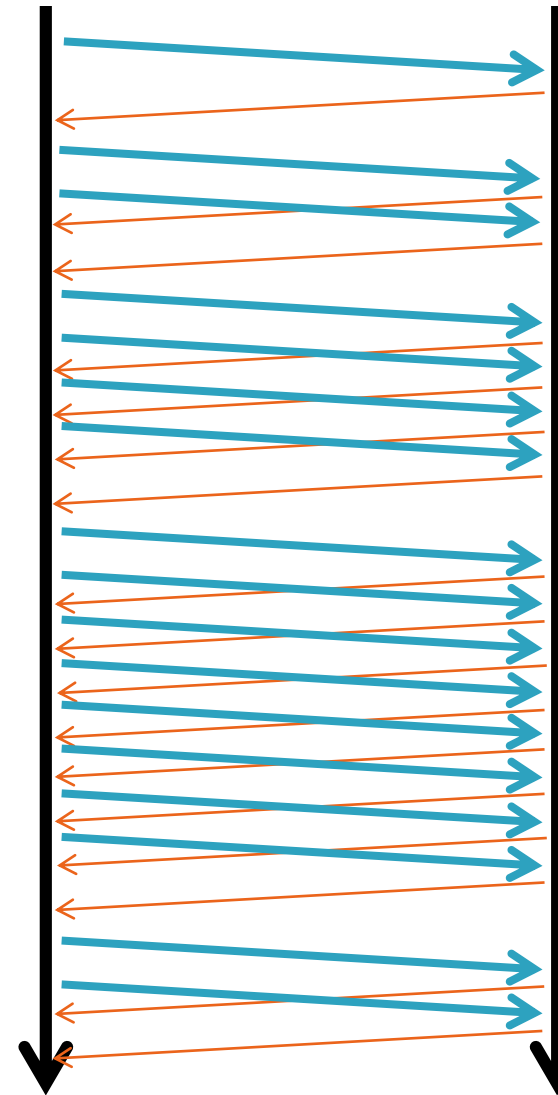
$cwnd = 2$

$cwnd = 4$

$ssthresh = 8$

$cwnd = 8$

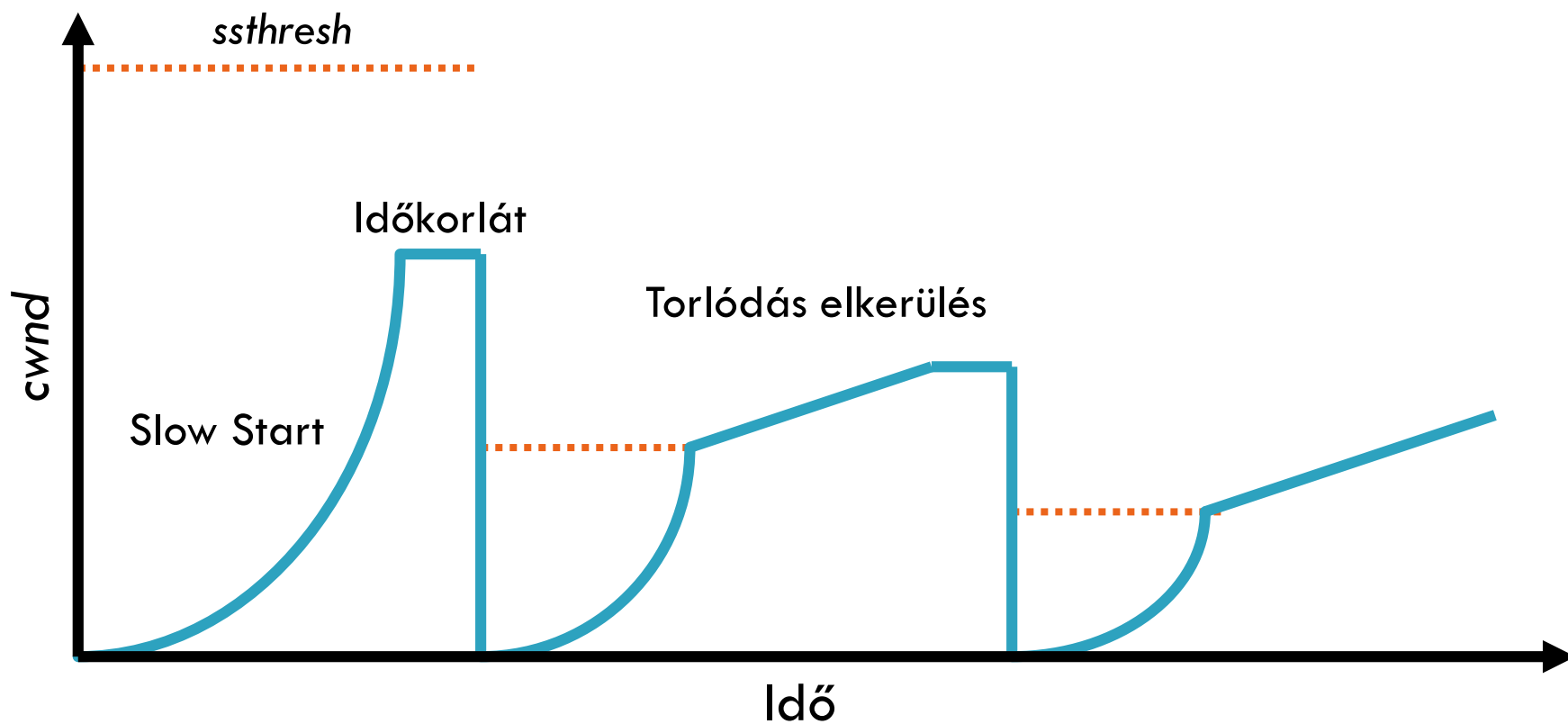
$cwnd = 9$



A teljes kép – TCP Tahoe

(az eredeti TCP)

17



Összefoglalás - TCP jellemzői

18

„A *TCP* egy kapcsolatorientált megbízható szolgáltatás kétirányú bájtfolyamokhoz.”

KAPCSOLATORIENTÁLT

- Két résztvevő, ahol egy résztvevőt egy *IP-cím* és egy *port* azonosít.
- A kapcsolat egyértelműen azonosított a résztvevő párral.
- Nincs se *multi-*, se *broadcast* üzenetküldés.
- A kapcsolatot fel kell építeni és le kell bontani.
- Egy kapcsolat a lezárásáig aktív.

Összefoglalás - TCP jellemzői

19

„A TCP egy kapcsolatorientált megbízható szolgáltatás kétirányú bájtfolyamokhoz.”

MEGBÍZHATÓSÁG

- ❑ Minden csomag megérkezése nyugtázásra kerül.
- ❑ A nem nyugtázott adatcsomagokat újraküldik.
- ❑ A fejléchez és a csomaghoz ellenőrzőösszeg van rendelve.
- ❑ A csomagokat számozza, és a fogadónál sorba rendezésre kerülnek a csomagok a sorszámuk alapján.
- ❑ Duplikátumokat törli.

Összefoglalás - TCP jellemzői

20

„A TCP egy kapcsolatorientált megbízható szolgáltatás kétirányú bájtfolyamokhoz.”

KÉTIRÁNYÚ BÁJTFOLYAM

- Az adatok két egymással ellentétes irányú bájtsorozatként kerülnek átvitelre.
- A tartalom nem interpretálódik.
- Az adatcsomagok időbeli viselkedése megváltozhat: átvitel sebessége növekedhet, csökkenhet, más késés, más sorrendben is megérkezhetnek.
- Megpróbálja az adatcsomagokat időben egymáshoz közel kiszállítani.
- Megpróbálja az átviteli közeget hatékonyan használni.

A TCP evolúciója

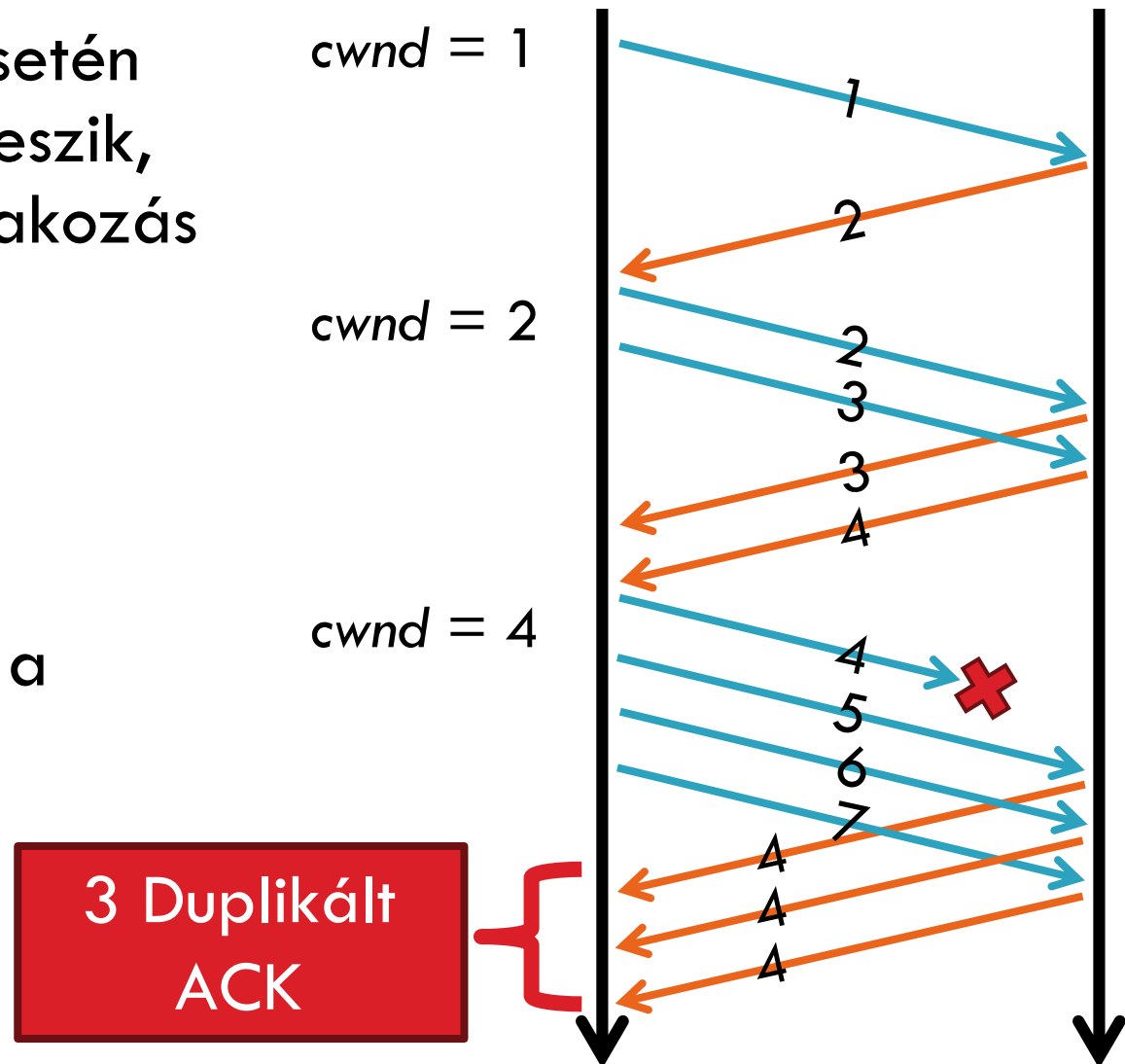
21

- Az eddigi megoldások a TCP Tahoe működéshez tartoztak
 - ▣ Eredeti TCP
- A TCP-t 1974-ben találták fel!
 - ▣ Napjainkba számos változata létezik
- Kezdeti népszerű változat: TCP Reno
 - ▣ Tahoe lehetőségei, plusz...
 - ▣ Gyors újraküldés (Fast retransmit)
 - 3 duplikált ACK? -> újraküldés (ne várjunk az RTO-ra)
 - ▣ Gyors helyreállítás (Fast recovery)
 - Csomagvesztés esetén:
 - $\text{set cwnd} = \text{cwnd} / 2$ (ssthresh = az új cwnd érték)

TCP Reno: Gyors újraküldés

22

- Probléma: Tahoe esetén ha egy csomag elveszik, akkor hosszú a várakozás az RTO-ig
- Reno: újraküldés 3 duplikált nyugta fogadása esetén
 - Explicit jele a csomagvesztésnek



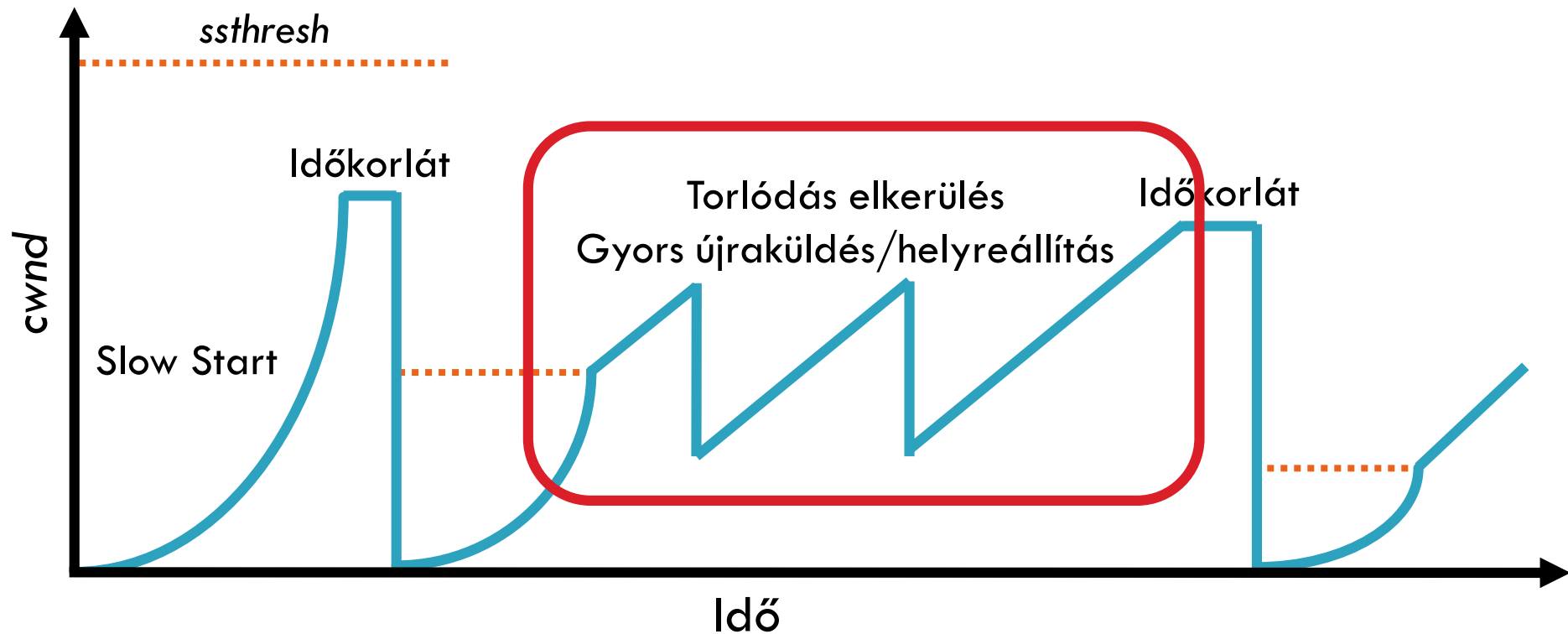
TCP Reno: Gyors helyreállítás

23

- ❑ Gyors újraküldés után módosítjuk a torlódási ablakot:
 - ▣ $cwnd := cwnd/2$ (valójában ez a *Multiplicative Decrease*)
 - ▣ $ssthresh :=$ az új $cwnd$
 - ▣ Azaz nem álltjuk vissza az eredeti 1-re a $cwnd$ -t!!!
 - ▣ Ezzel elkerüljük a felesleges slow start fázisokat!
 - ▣ Elkerüljük a költséges időkorlátokat
- ❑ Azonban ha az RTO lejár, továbbra is $cwnd = 1$
 - ▣ Visszatér a slow start fázishoz, hasonlóan a Tahoe-hoz
 - ▣ Olyan csomagokat jelez, melyeket egyáltalán nem szállítottunk le
 - ▣ A torlódás nagyon súlyos esetére figyelmeztet!!!

Példa: Gyors újraküldés/helyreállítás

24



- ❑ Stabil állapotban, a $cwnd$ az optimális ablakméret körül oszcillál
- ❑ TCP mindig csomagdobásokat kényszerít ki...

Számos TCP változat...

25

- ❑ Tahoe: az eredeti
 - ▣ Slow start és AIMD
 - ▣ Dinamikus RTO, RTT becsléssel
- ❑ Reno:
 - ▣ fast retransmit (3 dupACKs)
 - ▣ fast recovery ($cwnd = cwnd/2$ vesztes esetén)
- ❑ NewReno: javított gyors újraküldés
 - ▣ Minden egyes duplikált ACK újraküldést vált ki
 - ▣ Probléma: >3 hibás sorrendben fogadott csomag is újraküldést okoz (hibásan!!!)...
- ❑ Vegas: késleltetés alapú torlódás elkerülés
- ❑ ...

TCP a valóságban

26

- Mi a legnépszerűbb variáns napjainkban?
 - ▣ Probléma: TCP rosszul teljesít nagy késleltetés-sávszélesség szorzattal rendelkező hálózatokban (a modern Internet ilyen)
 - ▣ Compound TCP (Windows)
 - Reno alapú
 - Két torlódási ablak: késleltetés alapú és vesztes alapú
 - Azaz egy összetett torlódás vezérlést alkalmaz
 - ▣ TCP CUBIC (Linux)
 - Fejlettebb BIC (Binary Increase Congestion Control) változat
 - Az ablakméretet egy harmadfokú egyenlet határozza meg
 - A legutolsó csomagvesztéstől eltelt T idővel paraméterezett

Nagy késleltetés-sávszélesség szorzat (Delay-bandwidth product)

27

- Probléma: A TCP nem teljesít jól ha
 - ▣ A hálózat kapacitása (sávszélessége) nagy
 - ▣ A késleltetés (RTT) nagy
 - ▣ Vagy ezek szorzata nagy
 - $b * d =$ maximális szállítás alatt levő adatmennyiség
 - Ezt nevezzük késleltetés-sávszélesség szorzatnak
- Miért teljesít ekkor gyengén a TCP?
 - ▣ A slow start és az additive increase csak lassan konvergál
 - ▣ A TCP ACK ütemezett (azaz csak minden ACK esetén történik esemény)
 - A nyugták beérkezési gyorsasága határozza meg, hogy milyen gyorsan tud reagálni
 - Nagy RTT → késleltetett nyugták → a TCP csak lassan reagál a megváltozott viszonyokra

Célok

28

- ❑ A TCP ablak gyorsabb növelése
 - ▣ A slow start és az additive increase túl lassú, ha nagy a sávszélesség
 - ▣ Sokkal gyorsabb konvergencia kell
- ❑ Fairség biztosítása más TCP változatokkal szemben
 - ▣ Az ablak növelése nem lehet túl agresszív
- ❑ Javított RTT fairség
 - ▣ A TCP Tahoe/Reno folyamatok nem adnak fair erőforrás-megosztást nagyon eltérő RTT-k esetén
- ❑ Egyszerű implementáció

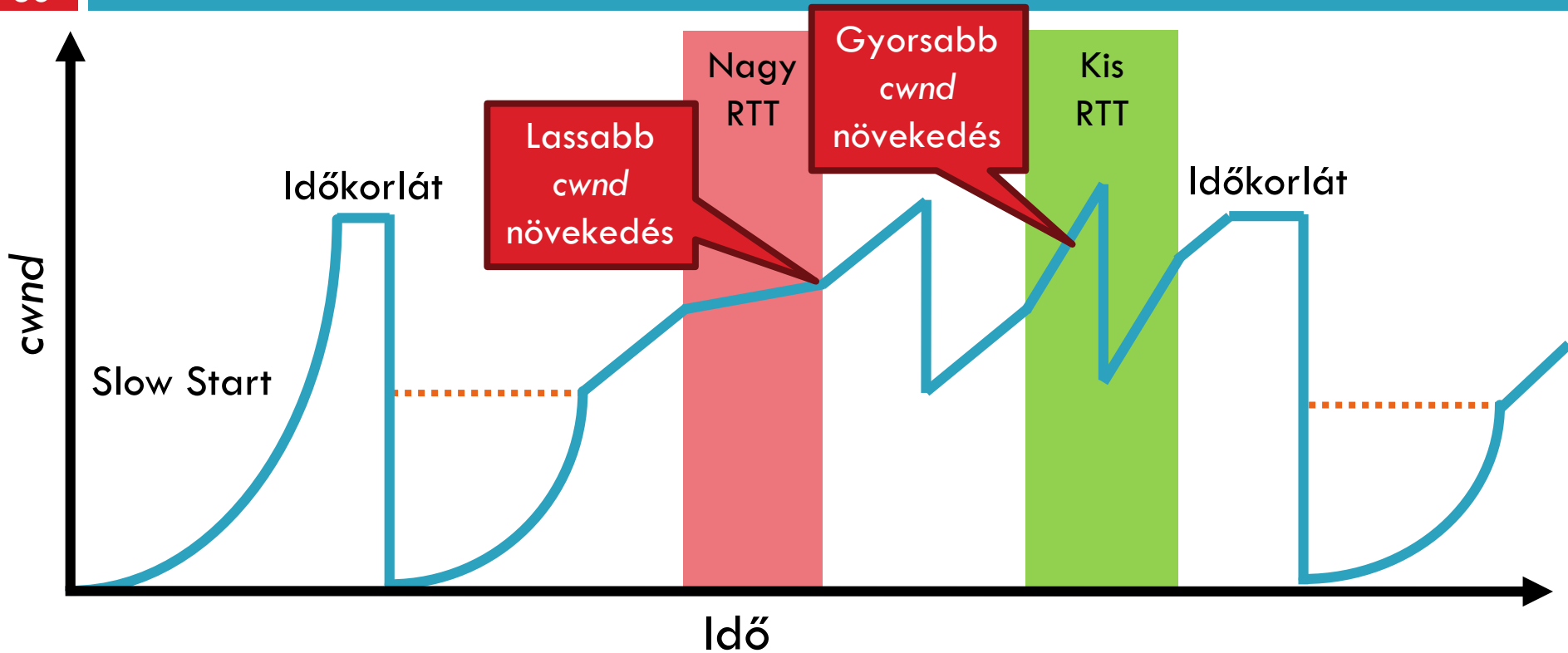
Compound TCP

29

- Alap TCP implementáció Windows rendszereken
- Ötlet: osszuk a *torlódási ablakot* két különálló ablakba
 - ▣ Hagyományos, vesztes alapú ablak
 - ▣ Új, késleltetés alapú ablak
- $wnd = \min(cwnd + dwnd, adv_wnd)$
 - ▣ $cwnd$ -t az AIMD vezérli AIMD
 - ▣ $dwnd$ a késleltetés alapú ablak
- A $dwnd$ beállítása:
 - ▣ Ha nő az RTT, csökken a $dwnd$ ($dwnd \geq 0$)
 - ▣ Ha csökken az RTT, nő a $dwnd$
 - ▣ A növekedés/csökkenés arányos a változás mértékével

Compound TCP példa

30



- Agresszívan reagál az RTT változására
- Előnyök: Gyors felfutás, sokkal fairebb viselkedés más folyamatokkal szemben eltérő RTT esetén
- Hátrányok: folyamatos RTT becslés

TCP CUBIC

31

- Alap TCP implementáció Linux rendszereken
- Az AIMD helyettesítése egy „kübös” (CUBIC) függvénnnyel

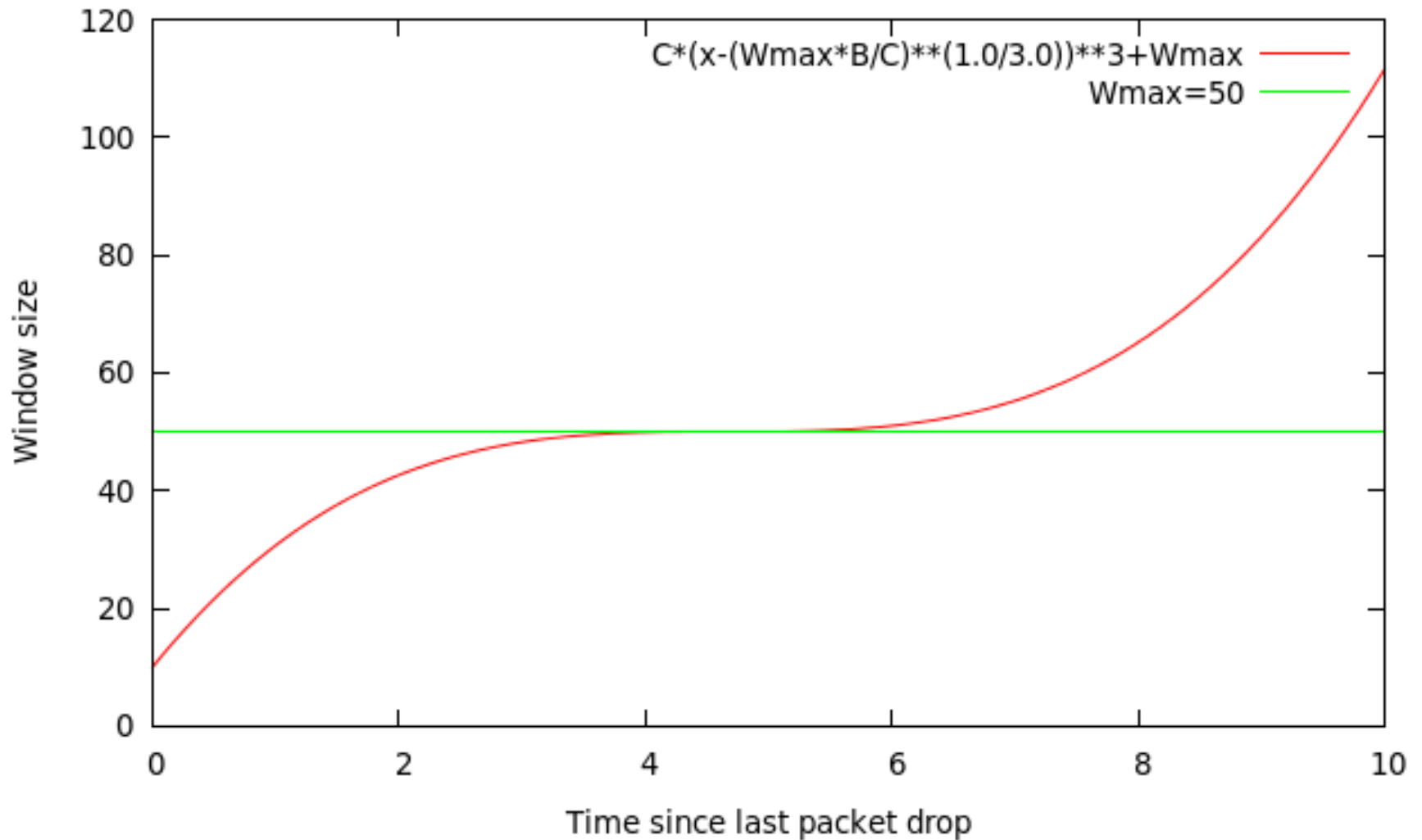
$$W_{cubic} = C(T - K)^3 + W_{max} \quad (1)$$

C is a scaling constant, and $K = \sqrt[3]{\frac{W_{max}\beta}{C}}$

- $B \rightarrow$ egy konstans a multiplicative increase fázishoz
- $T \rightarrow$ eltelt idő a legutóbbi csomagvesztés óta
- $W_{max} \rightarrow$ cwnd a legutolsó csomagvesztés idején

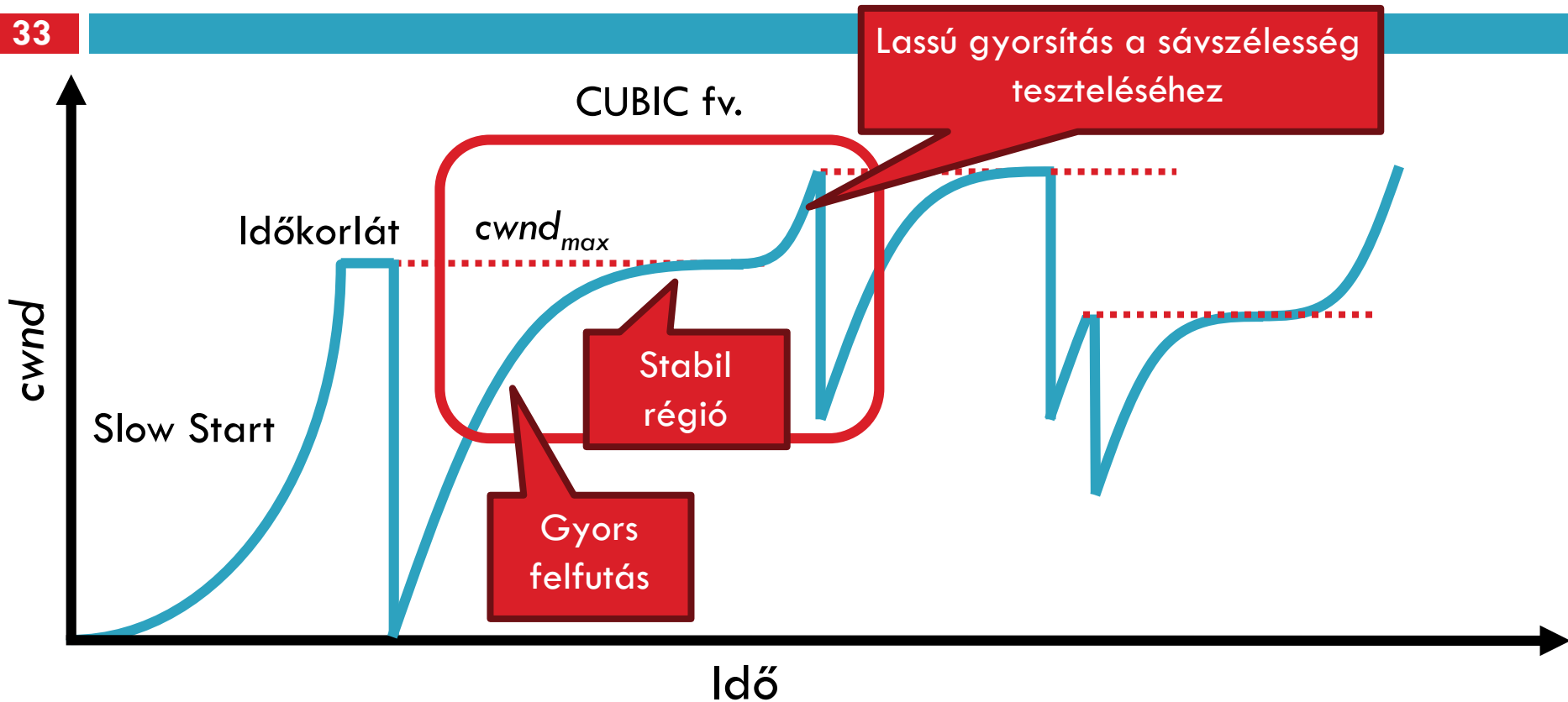
TCP CUBIC

32



TCP CUBIC példa

33



- Kevésbé pazarolja a sávszélességet a gyors felfutások miatt
- A stabil régió és a lassú gyorsítás segít a fairness biztosításában
 - ▣ A gyors felfutás sokkal agresszívabb, mint az additive increase
 - ▣ A Tahoe/Reno variánsokkal szembeni fairnesshez a CUBIC-nak nem szabad ennyire agresszívnak lennie

Problémák a TCP-vel

34

- Az Internetes forgalom jelentős része TCP
- Azonban számos probléma okozója is egyben
 - ▣ Gyenge teljesítmény kis folyamok esetén
 - ▣ Gyenge teljesítmény wireless hálózatokban
 - ▣ DoS támadási felület

Kis folyamok (flows)

35

- ❑ Probléma: kis folyamok esetén torz viselkedés
 - ❑ 1 RTT szükséges a kapcsolat felépítésére (SYN, SYN/ACK)
 - pazarló
 - ❑ *cwnd* mindig 1-gyel indul
 - Nincs lehetőség felgyorsulni a kevés adat miatt
- ❑ Az Internetes forgalom nagy része kis folyam
 - ❑ Többnyire HTTP átvitel, <100KB
 - ❑ A legtöbb TCP folyam el se hagyja a slow start fázist!!!
- ❑ Lehetséges megoldás (Google javaslat):
 - ❑ Kezdeti *cwnd* megnövelése 10-re
 - ❑ TCP Fast Open: kriptográfiai hashek használata a fogadó azonosítására, a három-utas kézfogás elhagyható helyette hash (cookie) küldése a syn csomagban

Wireless hálózatok

36

- ❑ Probléma: A Tahoe és Reno esetén csomagvesztés = torlódás
 - ▣ WAN esetén ez helyes, ritka bit hibák
 - ▣ Azonban hamis vezeték nélküli hálózatokban, gyakori interferenciák
- ❑ TCP átvitel $\sim 1/\sqrt{\text{vesztési ráta}}$
 - ▣ Már néhány interferencia miatti csomagvesztés elég a teljesítmény drasztikus csökkenéséhez
- ❑ Lehetséges megoldások:
 - ▣ Réteg modell megsértése, adatkapcsolati információ a TCP-be
 - ▣ Késleltetés alapú torlódás vezérlés használata (pl. TCP Vegas)
 - ▣ Explicit torlódás jelzés - Explicit congestion notification (ECN)

Szolgáltatás megtagadása

Denial of Service (DoS)

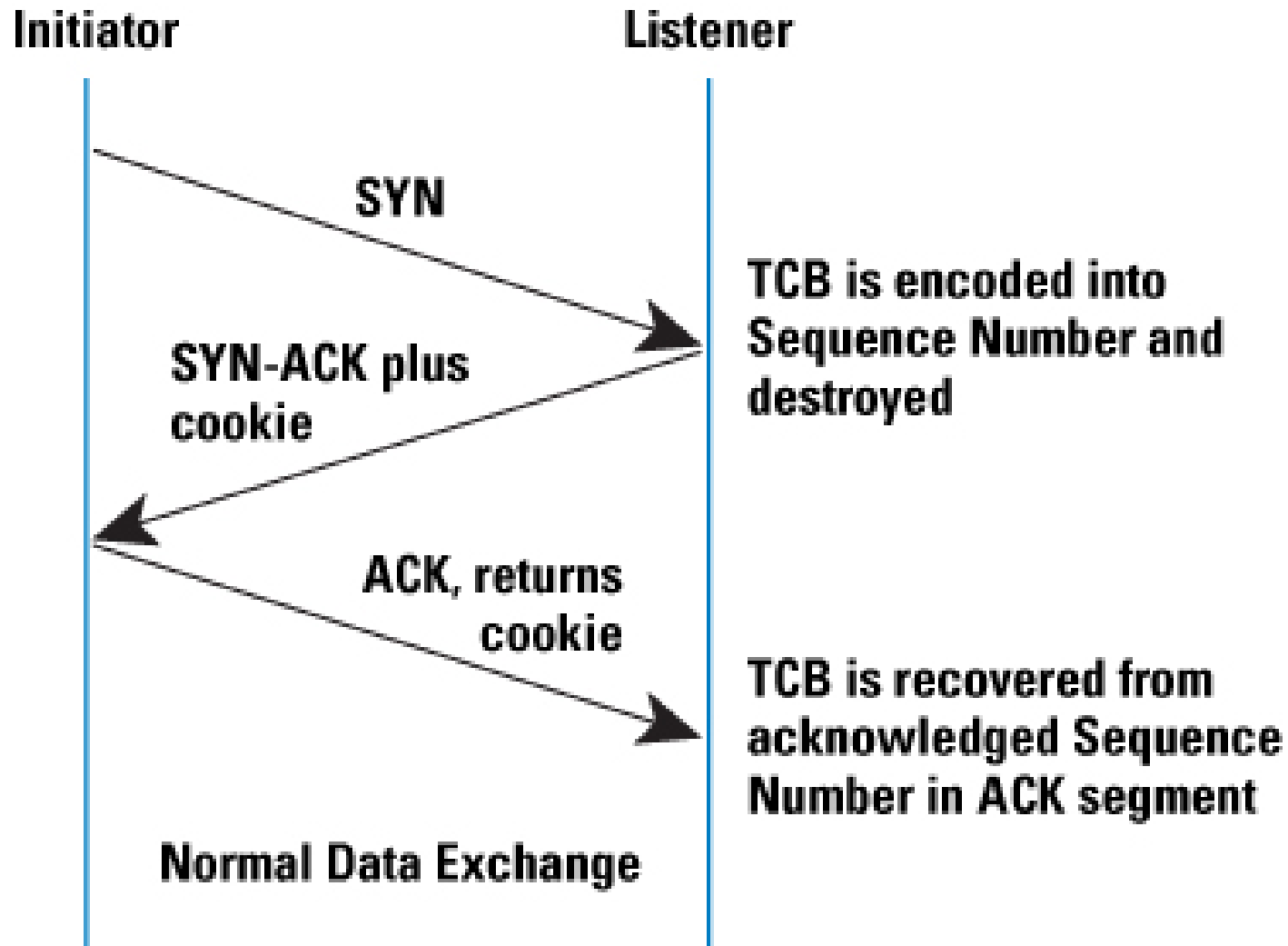
37

- ❑ Probléma: a TCP kapcsolatok állapottal rendelkeznek
 - ▣ A SYN csomagok erőforrásokat foglalnak az serveren
 - ▣ Az állapot legalább néhány percig fennmarad (RTO)
- ❑ SYN flood: elég sok SYN csomag küldése a servernek ahhoz, hogy elfogyjon a memória és összeomoljon a kernel
- ❑ Megoldás: SYN cookie-k
 - ▣ Ötlet: ne tároljunk kezdeti állapotot a serveren
 - ▣ Illesszük az állapotot a SYN/ACK csomagokba (a sorszám mezőbe (sequence number mező))
 - ▣ A kliensnek vissza kell tükrözni az állapotot...

Szolgáltatás megtagadása

Denial of Service (DoS)

38



Kitekintés

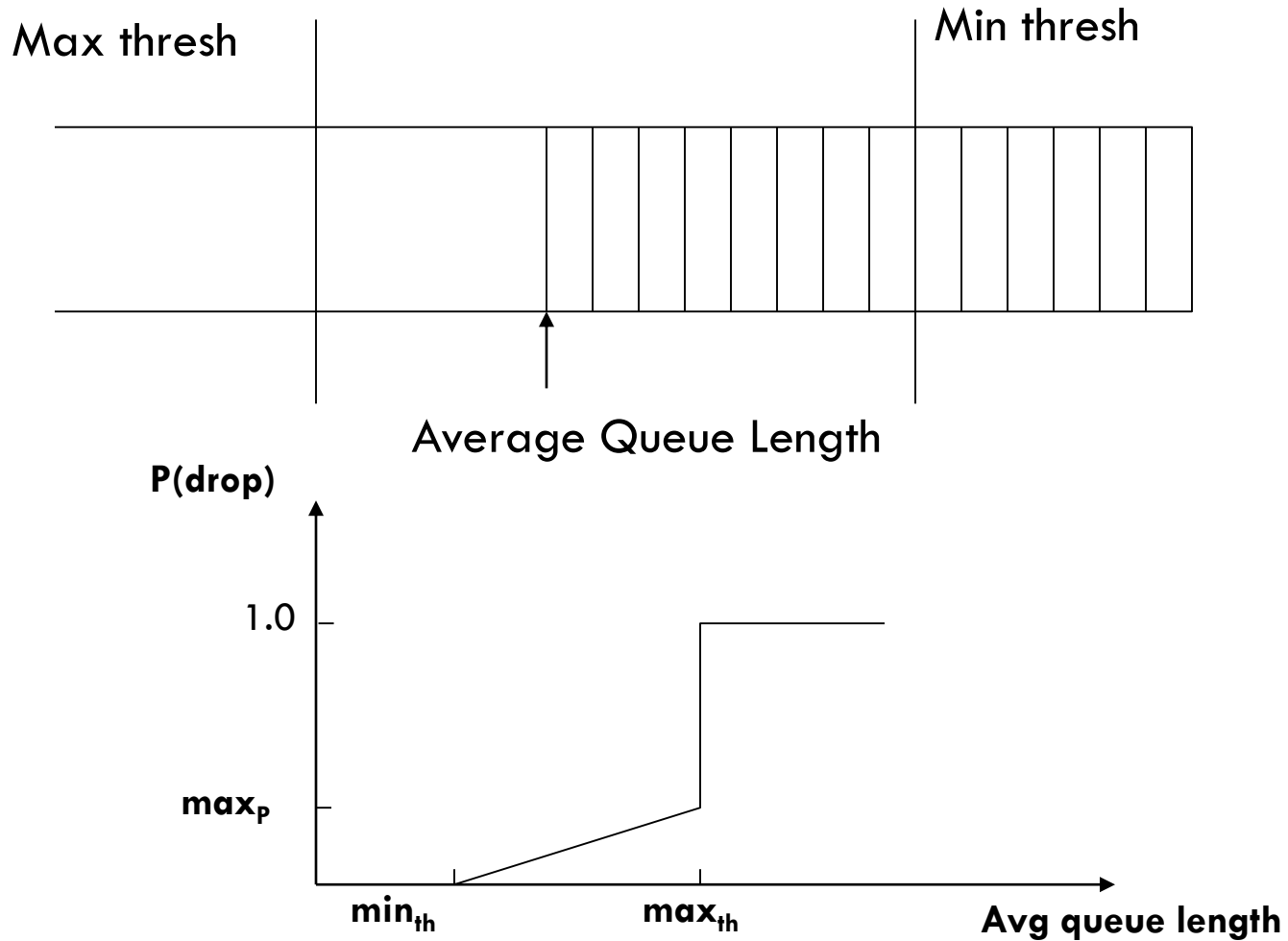
Typical Internet Queuing

- ❑ FIFO + drop-tail
 - ▣ Simplest choice
 - ▣ Used widely in the Internet
- ❑ FIFO (first-in-first-out)
 - ▣ Implies single class of traffic
- ❑ Drop-tail
 - ▣ Arriving packets get dropped when queue is full regardless of flow or importance
- ❑ Important distinction:
 - ▣ FIFO: scheduling discipline
 - ▣ Drop-tail: drop policy

RED Algorithm

- ❑ Maintain running average of queue length
- ❑ If $\text{avgq} < \text{min}_{\text{th}}$ do nothing
 - ▣ Low queuing, send packets through
- ❑ If $\text{avgq} > \text{max}_{\text{th}}$, drop packet
 - ▣ Protection from misbehaving sources
- ❑ Else mark packet in a manner proportional to queue length
 - ▣ Notify sources of incipient congestion
 - ▣ E.g. by ECN IP field or dropping packets with a given probability

RED Operation



RED Algorithm

- Maintain running average of queue length
- For each packet arrival
 - ▣ Calculate average queue size (avg)
 - ▣ If $\min_{th} \leq avg < \max_{th}$
 - Calculate probability P_a
 - With probability P_a
 - ▣ Mark the arriving packet: drop or set-up ECN
 - Else if $\max_{th} \leq avg$
 - ▣ Mark the arriving packet: drop, ECN

Csomag dobás vagy ECN jelölés

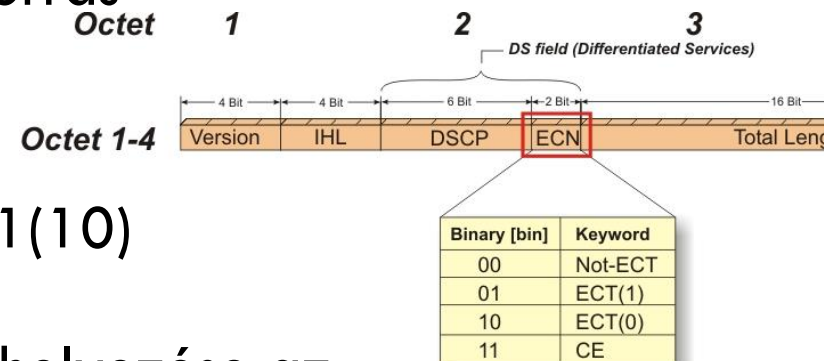
44

❑ Csomag dobás

- ▣ Újra küldés szükséges
- ▣ Egyszerűbb megvalósítás
- ▣ Timeout lejártá után tud reagálni a forrás

❑ ECN jelölés

- ▣ Végpont támogatás szükséges
- ▣ Az IP csomag ECT-0 (01) vagy ECT-1 (10) jelöléssel
- ▣ Dobás helyett -> ECN CE (11) jel elhelyezése az IP fejlécben
- ▣ A fogadó érzékeli a CE jelet, majd a visszamenő TCP nyugtába bebillent egy ECE flaget, mely jelzi a forrásnak a torlódást
- ▣ Hagyományos TCP (CUBIC, RENO, stb.) források az ECE flaget csomagvesztésnek értelmezik, de újra küldés nem szükséges.



Data Center TCP: DCTCP

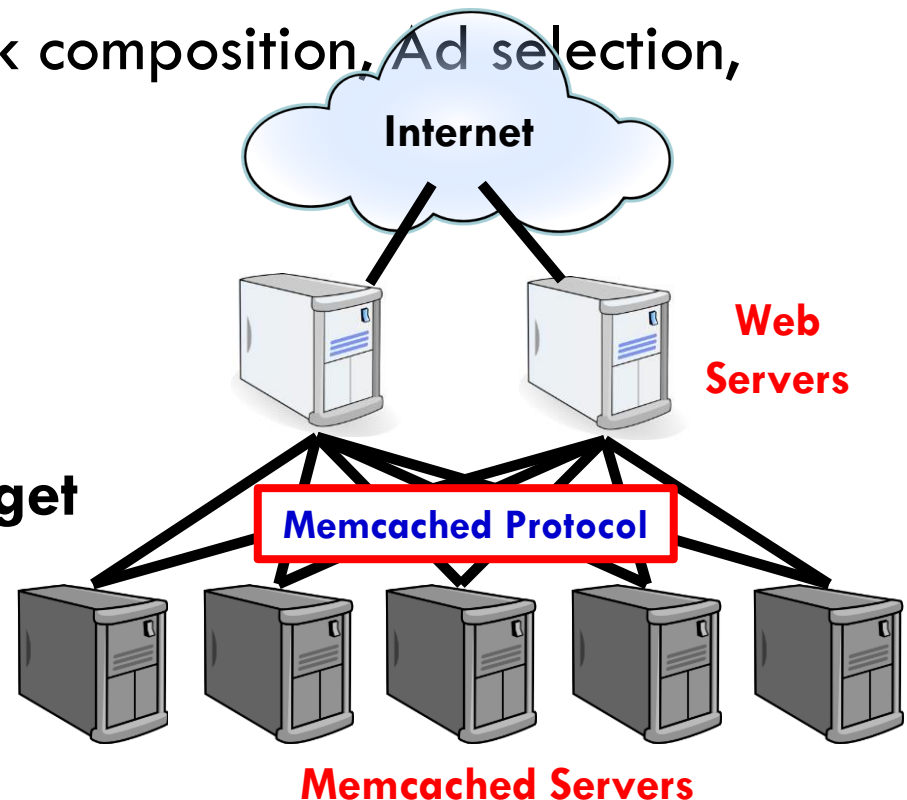
Generality of Partition/Aggregate

- The foundation for many large-scale web applications.
 - ▣ Web search, Social network composition, Ad selection, etc.

□ Example: **Facebook**

Partition/Aggregate ~ Multiget

- ▣ Aggregators: **Web Servers**
- ▣ Workers: **Memcached Servers**



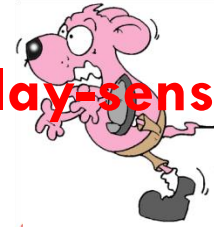
Workloads

47

□ Partition/Aggregate
(Query)



Delay-sensitive



□ Short messages [50KB-1MB]
(Coordination, Control state)



Delay-sensitive



□ Large flows [1MB-50MB]
(Data update)



Throughput-sensitive



Impairments

48

- ❑ Incast
- ❑ Queue Buildup
- ❑ Buffer Pressure

Incast

49

Worker 1

Worker 2

Worker 3

Worker 4

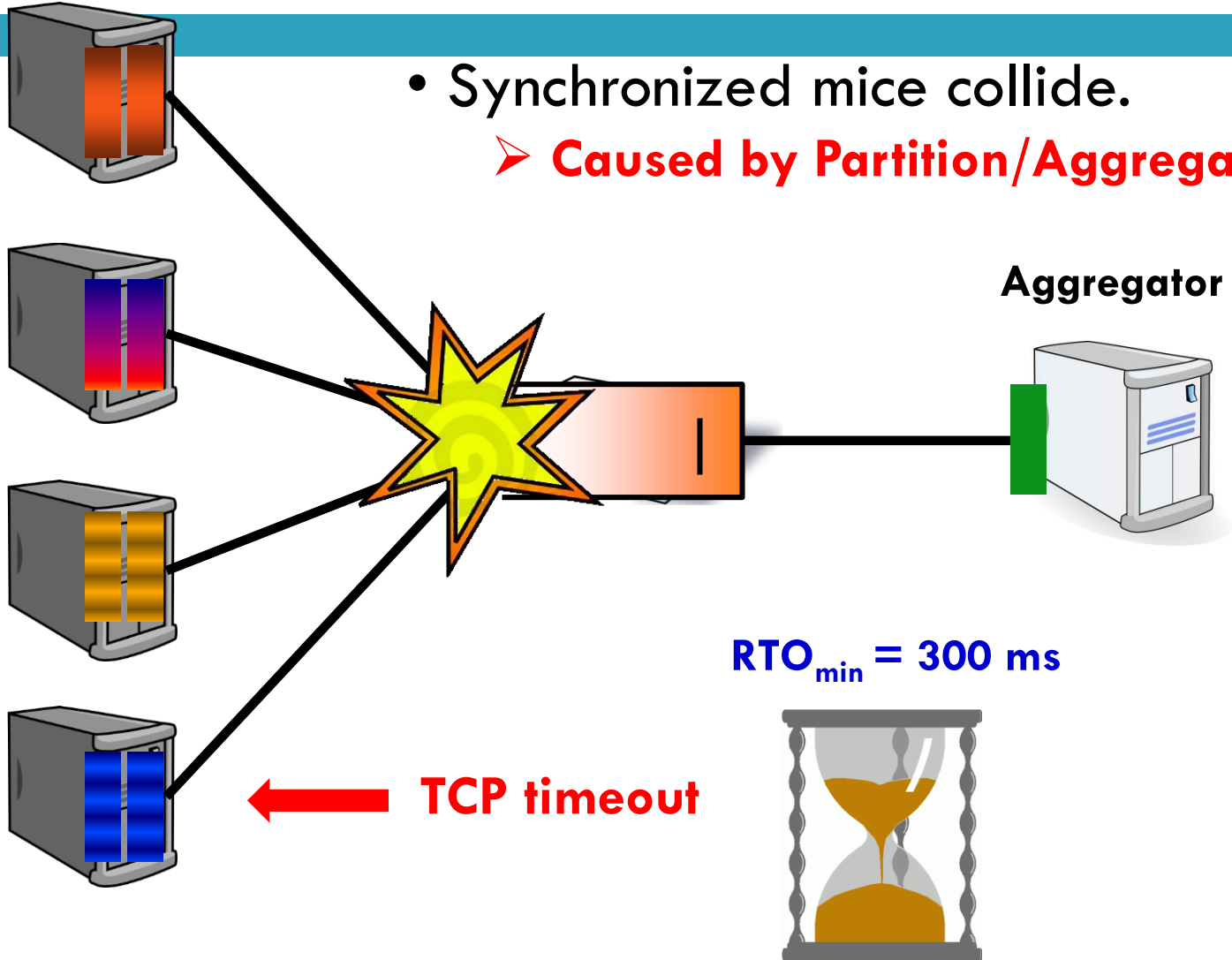
- Synchronized mice collide.

➤ **Caused by Partition/Aggregate.**

Aggregator

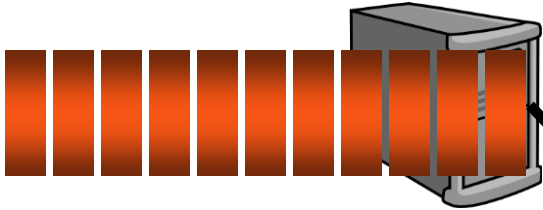
$RTO_{\min} = 300 \text{ ms}$

← **TCP timeout**



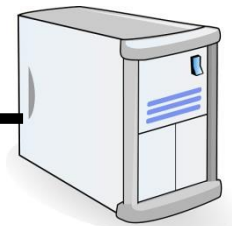
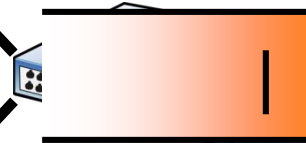
Queue Buildup

Sender 1

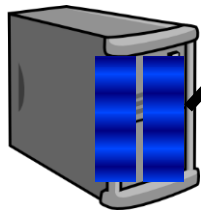


- Big flows buildup queues.
 - Increased latency for short flows.

Receiver



Sender 2



- Measurements in Bing cluster
 - For 90% packets: $RTT < 1ms$
 - For 10% packets: $1ms < RTT < 15ms$

Data Center Transport Requirements

51

1. High Burst Tolerance

- Incast due to Partition/Aggregate is common.

2. Low Latency

- Short flows, queries

3. High Throughput

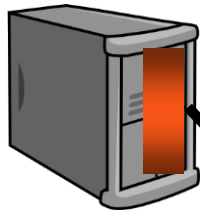
- Continuous data updates, large file transfers

The challenge is to achieve these three together.

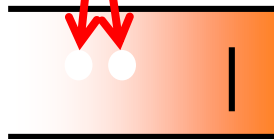
DCTCP: The TCP/ECN Control Loop

Sender 1

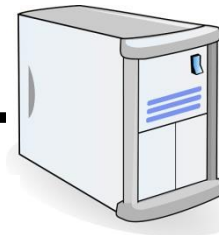
ECN = Explicit Congestion Notification



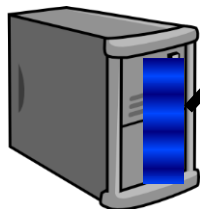
ECN Mark (1 bit)



Receiver



Sender 2



DCTCP: Two Key Ideas

18

1. React in proportion to the **extent** of congestion, not its **presence**.
 - ✓ Reduces **variance** in sending rates, lowering queuing requirements.

ECN Marks	TCP	DCTCP
1 0 1 1 1 1 0 1 1 1	Cut window by 50%	Cut window by 40%
0 0 0 0 0 0 0 0 0 1	Cut window by 50%	Cut window by 5%

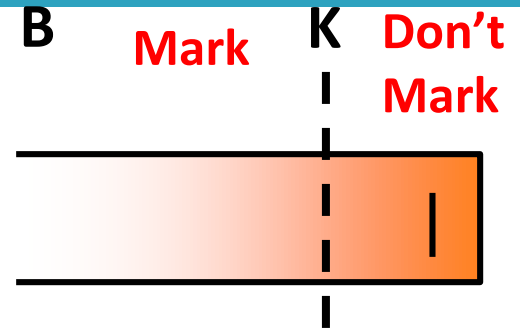
2. Mark based on **instantaneous** queue length.
 - ✓ Fast feedback to better deal with bursts.

Data Center TCP Algorithm

19

Switch side:

- Mark packets when **Queue Length > K**.



Sender side:

- Maintain running average of ***fraction*** of packets marked (α).

In each RTT:

$$F = \frac{\# \text{ of marked ACKs}}{\text{Total \# of ACKs}}$$

$$\alpha \leftarrow (1 - g)\alpha + gF$$

- **Adaptive window decreases:** $cwnd \leftarrow (1 - \frac{\alpha}{2})cwnd$

- Note: decrease factor between 1 and 2.

DNS

„8. réteg” (A szénelalapú csomópontok)

56

□ Ha...

- ▣ Fel szeretnél hívni valakit, akkor el kell kérned a telefonszámát
 - Nem hívhatod csak úgy “P I S T Á T”
- ▣ Levelet küldenél valakinek, akkor szükséged van a címére

□ Mi a helyzet az Internettel?

- ▣ Ha el akarod érni a Google-t, szükséges annak IP címe
- ▣ Tudja valaki a Google IP címét???

□ A probléma bennünk van:

- ▣ Az emberek nem képesek IP címek megjegyzésére
- ▣ Ember számára értelmes nevek kellene, melyek IP címekre képezhetők

Internetes nevek és címek

57

- ❑ Címek, pl. 129.10.117.100
 - ▣ Számítógépek által használt címkék a gépek azonosítására
 - ▣ A hálózat szerkezetét tükrözi
- ❑ Nevek, pl. www.northeastern.edu
 - ▣ Ember számára értelmes címkék a gépeknek
 - ▣ A szervezeti struktúrát tükrözi
- ❑ Hogyan képezzünk az egyikről a másikra?
 - ▣ Domain Name System (DNS)

Réges régen...

58

- ❑ A DNS előtt minden név-IP leképezés egy *hosts.txt*-ben volt
 - ▣ */etc/hosts* - Linuxon
 - ▣ *C:\Windows\System32\drivers\etc\hosts* - Windowson
- ❑ Központosított, manuális rendszer
 - ▣ A változásokat emailben kellett beküldeni a SRI-nek
 - SRI=Stanford Research Institute
 - ▣ A gépek periodikus időközönként letöltötték (FTP) a *hosts.txt* fájlt
 - ▣ Minden név megengedett volt – nem volt benne hierarchia („flat” (sík) felépítés)
 - *alans_server_at_sbu_pwns_joo_lol_kthxbye*

A DNS felé

59

- ❑ Végül a *hosts.txt* alapú rendszer szétesett
 - ❑ Nem skálázható, SRI nem bírta a terheléssel/igényekkel
 - ❑ Nehéz volt a nevek egyediségének biztosítása
 - Pl. MIT
 - Massachusetts Institute of Technology?
 - Melbourne Institute of Technology?
 - ❑ Számos gép rendelkezett nem naprakész *hosts.txt*-vel
- ❑ Ez vezetett a DNS megszületéséhez...

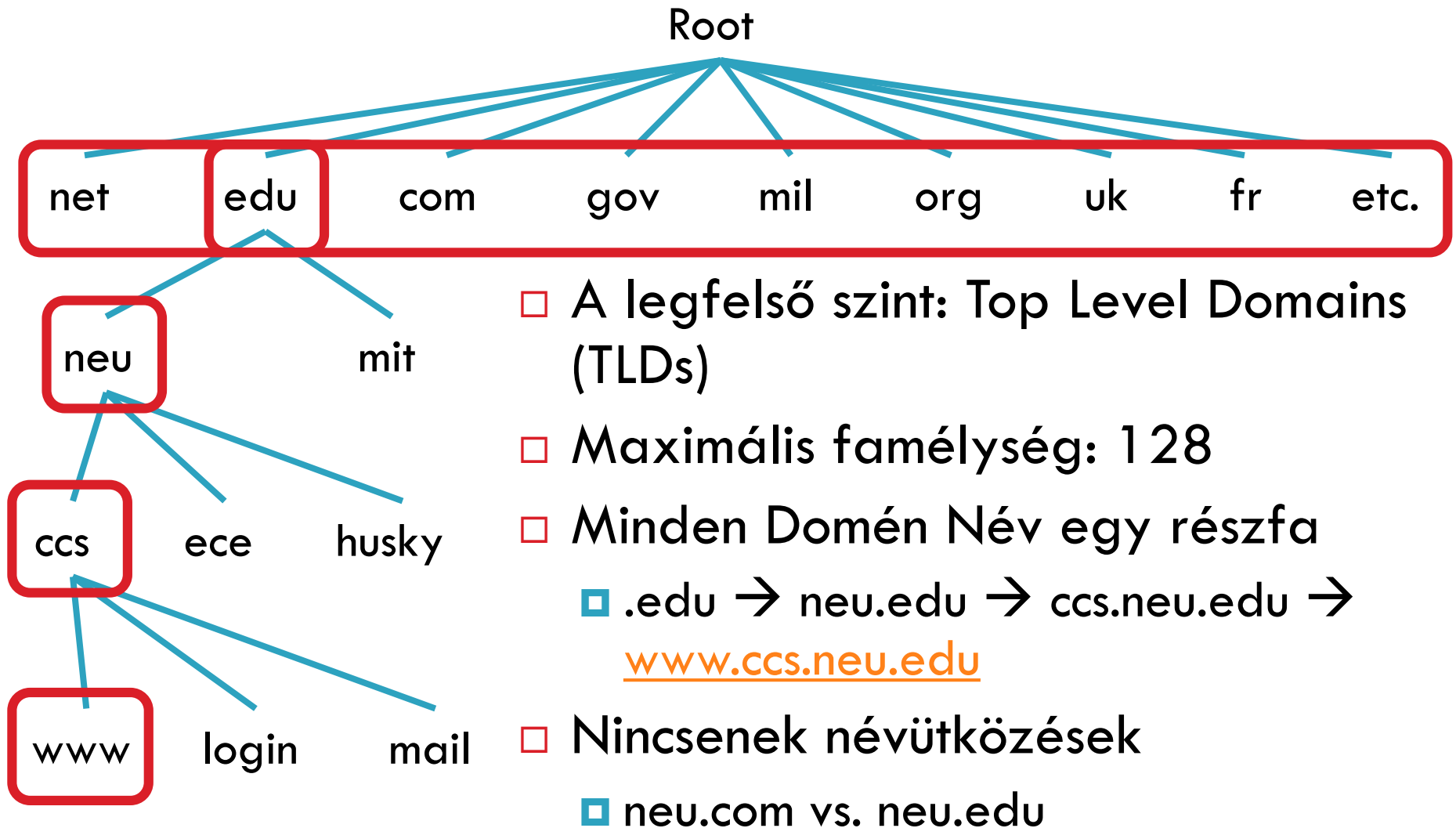
DNS általánosságban

60

- ❑ Domain Name System
- ❑ Elosztott adatbázis
 - ▣ Nem központosított
- ❑ Egyszerű kliens-szerver architektúra
 - ▣ UDP 53-as port, vannak TCP implementációk is
 - ▣ Rövid kérések – rövid válaszok; kérdés-válasz típusú kommunikáció
- ❑ Hierarchikus névtér
 - ▣ Szemben a hosts.txt alapú flat megoldással
 - ▣ pl. .com → google.com → mail.google.com

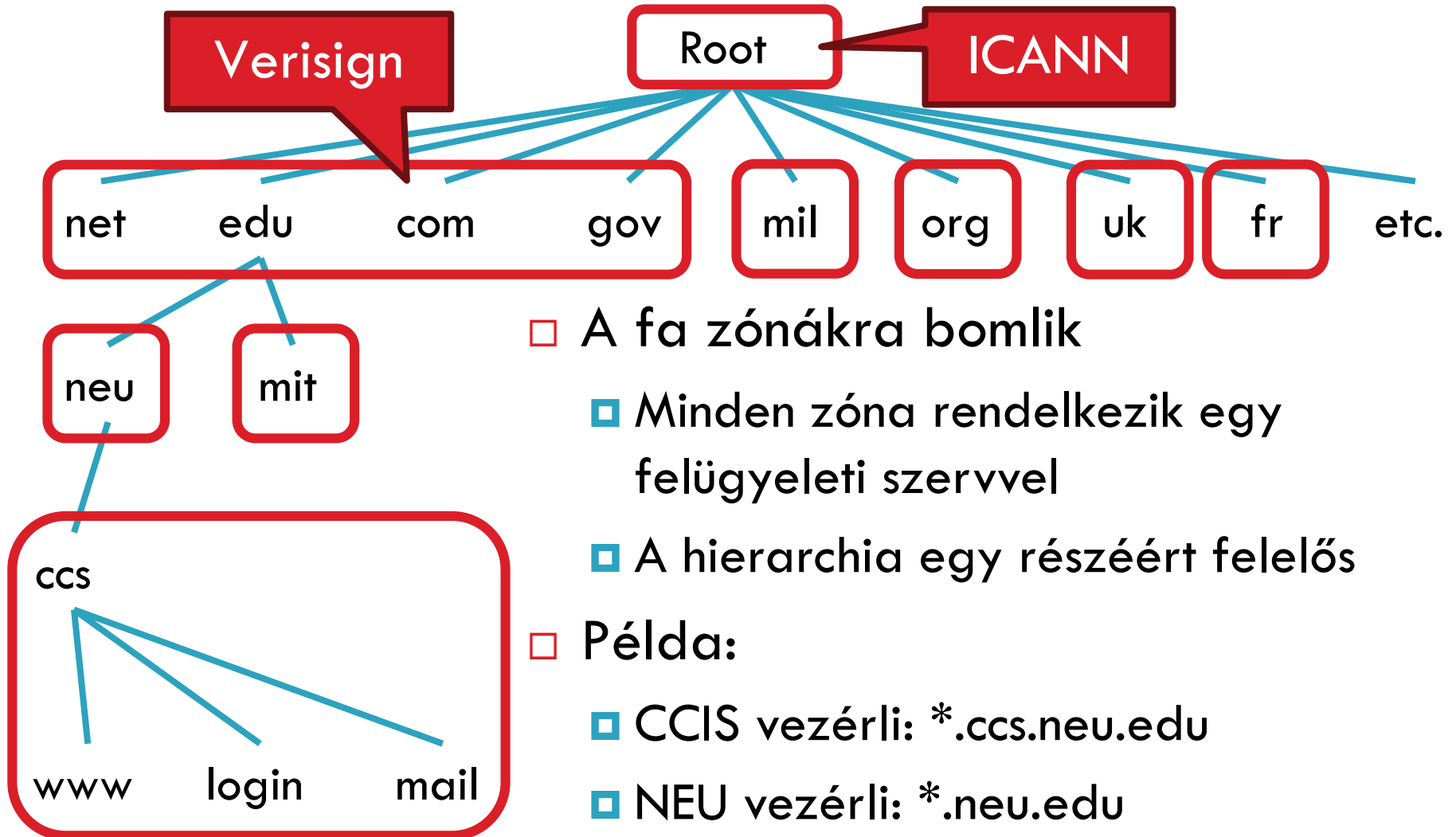
Név hierarchia

61



Hierarchikus adminisztráció

62



Szerver hierarchia

63

- ❑ Egy DNS szerver funkciói:
 - ❑ A hierarchia egy részét felügyeli
 - Nem szükséges minden DNS nevet tárolnia
 - ❑ A zónájához tartozó összes hoszt és domén rekordjainak tárolása
 - Másolatok lehetnek a robosztusság növelés végett
 - ❑ Ismeri a root szerverek címét
 - Ismeretlen nevek feloldása miatt kell
- ❑ A root szerverek minden TLD-t ismernek
 - ❑ Azaz innen indulva fel lehet tárni...

Top Level Domains

- ❑ Internet Corp. Assigned Names and Numbers (1998)
- ❑ 22+ **általános TLDs** léteznek
 - ❑ klasszikusok: .com, .edu, .gov, .mil, .org, .net
 - ❑ később keletkeztek: .aero, .museum, .xxx
- ❑ ~250 TLDs a különböző **ország kódok**nak
 - ❑ Két betű (mint például .au, .hu), 2010-től plusz nemzetközi karakterek (például kínai)
 - ❑ Több elüzletisedett, például a .tv (Tuvalu)
 - ❑ Példa domén hack-ekre: instagr.am (Örményország), goo.gl (Grönland)

Root Name Servers

65

□ A Root Zone Fájlért felelős

- ▣ Listát vezet a TLD-kről és arról, hogy ki felügyeli őket.
- ▣ ~272KB a fájl mérete
- ▣ Pl. bejegyzése:

com.	172800	IN	NS	a.gtld-servers.net.
com.	172800	IN	NS	b.gtld-servers.net.
com.	172800	IN	NS	c.gtld-servers.net.

□ Az ICANN adminisztrálja

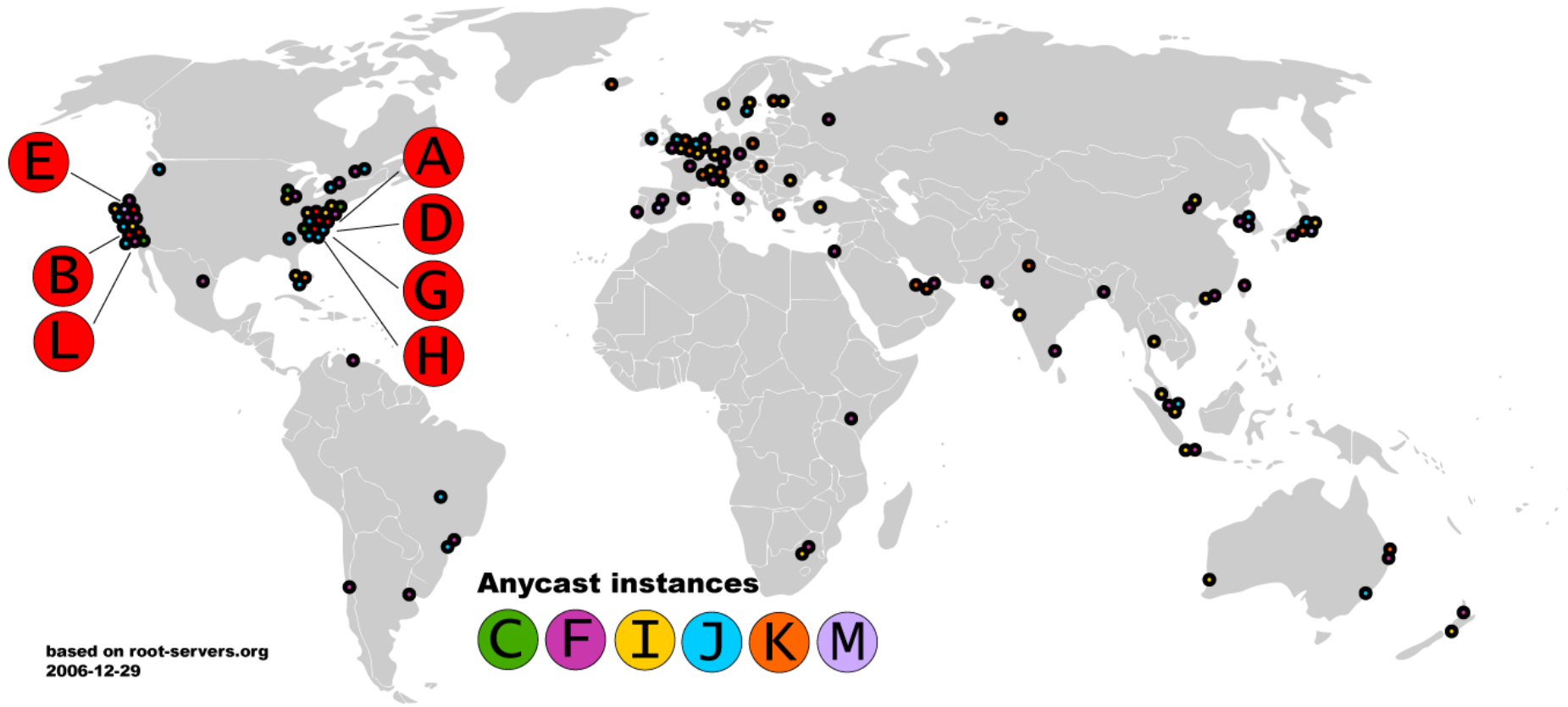
- ▣ 13 root szerver, címkék: A→M
- ▣ Pl.: i.root-servers.net
- ▣ 6 db ezek közül „anycastolt”, azaz globálisan számos replika létezik

□ Ha név nem feloldható (lokálisan), akkor hozzájuk kell fordulni

- ▣ A gyakorlatban a legtöbb rendszer lokálisan tárolja ezt az információt (cache)

Map of the Roots

66



Lokális névszerverek

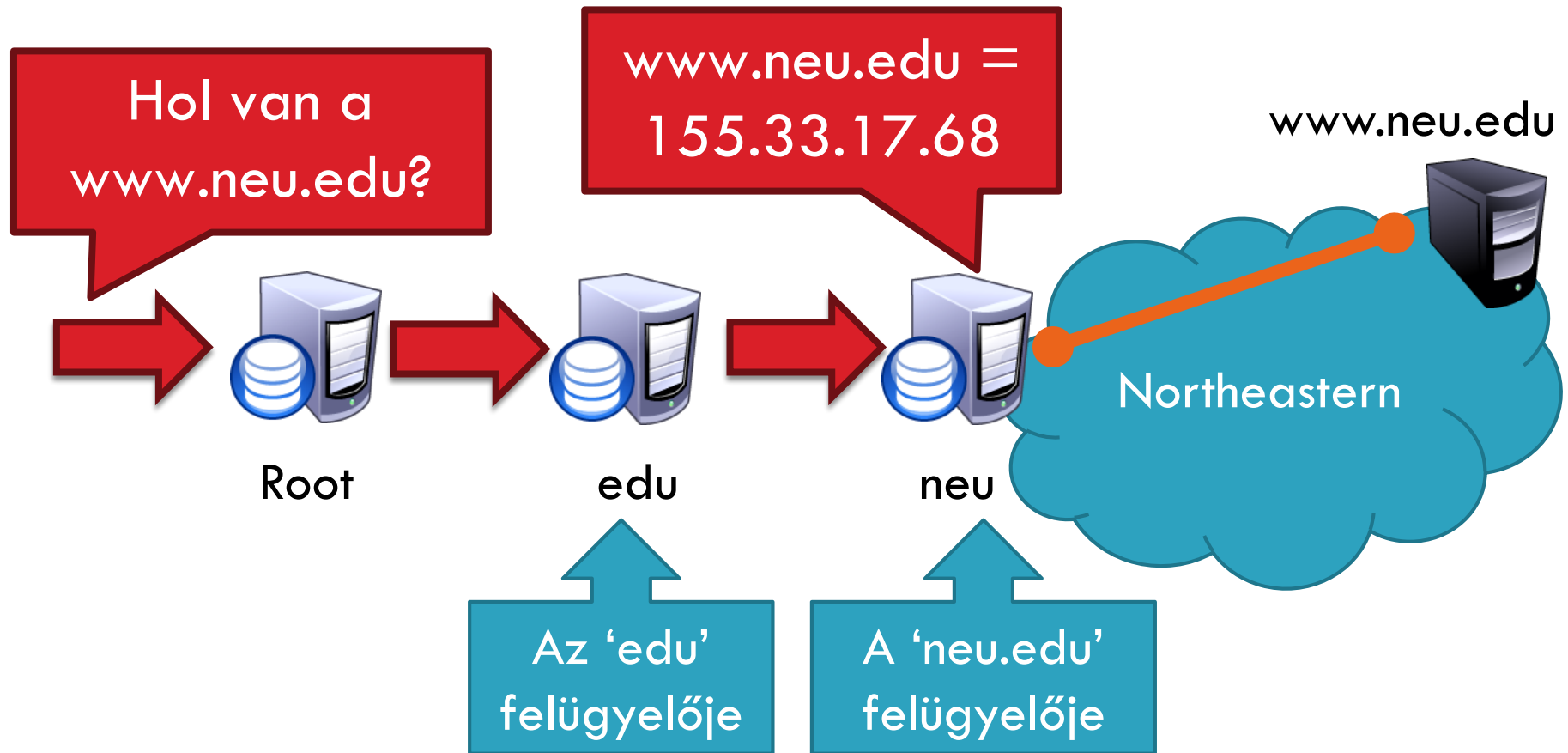
67



- ❑ Minden ISP/cég rendelkezik egy lokális, default névszerverrel
- ❑ Gyakran a DHCP konfigurálja fel
- ❑ A DNS lekérdezések a lokális névszervernél kezdődnek
- ❑ Gyakran cache-be teszik a lekérdezés eredményét

Authoratív Névszerverek

68



□ név → IP leképezéseket tárolja egy adott hoszthoz

Egyszerű doménnév feloldás

69

- ❑ Minden hoszt ismer egy lokális DNS szervert
 - ▣ Minden kérést ennek küld
- ❑ Ha a lokális DNS szerver tud válaszolni, akkor kész...
 1. A lokális szerver a felügyelő szerver az adott névhez
 2. A lokális szerver cache-ében van rekord a keresett névhez
- ❑ Különböznénk végig a teljes hierarchián felülről lefelé egészen a keresett név felügyeleti szerveréig
 - ▣ Minden lokális DNS szerver ismeri a root szervereket
 - ▣ Cache tartalma alapján bizonyos lépések átugrása, ha lehet
 - Pl. ha a root fájl tárolva van a cache-ben, akkor egyből ugorhatunk az „edu” szerverére.

Lekérdezések

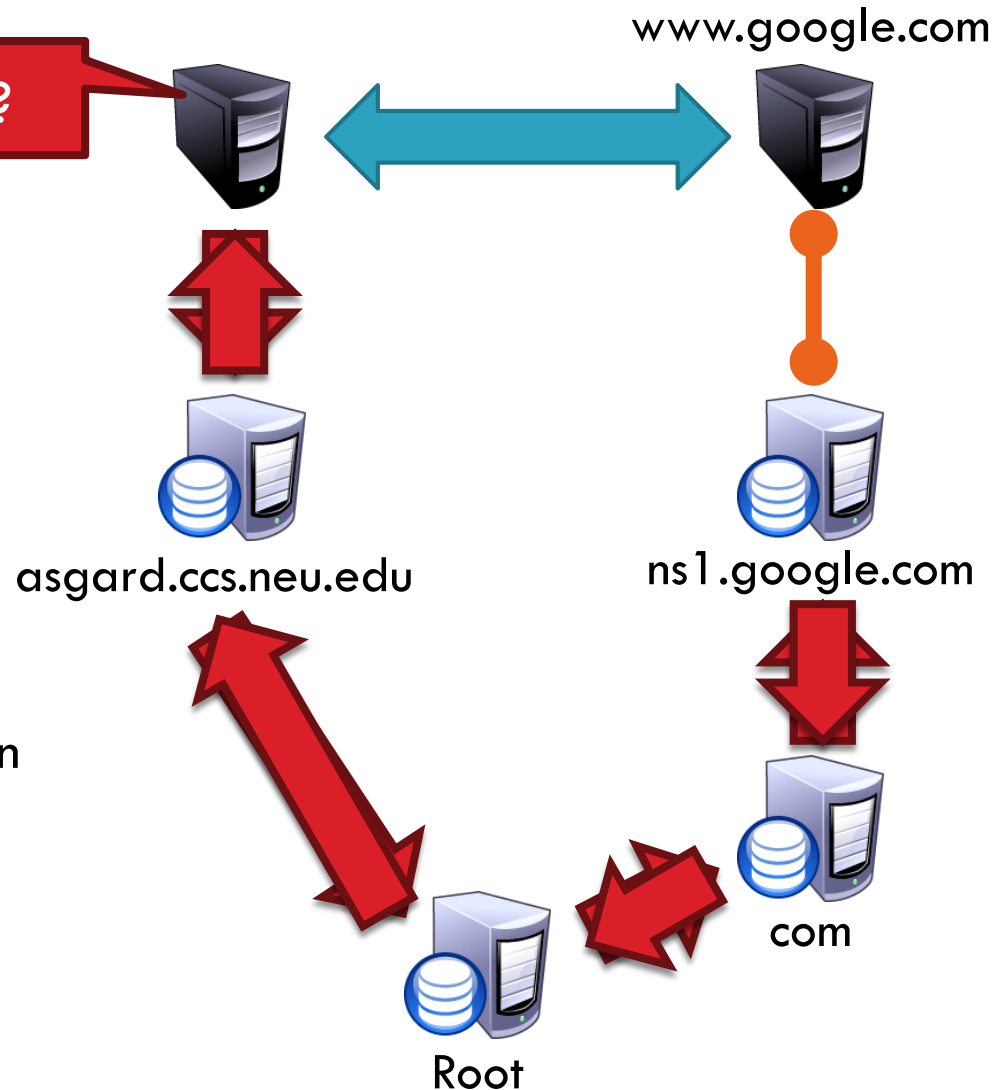
- A lekérdezésnek két fajtája van:
 - ▣ *Rekurzív lekérdezés* – Ha a névszerver végzi el a névfeloldást, és tér vissza a válasszal.
 - ▣ *Iteratív lekérdezés* – Ha a névszerver adja vissza a választ vagy legalább azt, hogy kitől kapható meg a következő válasz.
- Melyik a jobb?
 - ▣ *Rekurzív jellemzői*
 - Lehetővé teszi a szervernek a kliens terhelés kihelyezését a kezelhetőségért.
 - Lehetővé teszi a szervernek, hogy a kliensek egy csoportja felett végezzen cachelést, a jobb teljesítményért.
 - ▣ *Iteratív jellemzői*
 - Válasz után nem kell semmit tenni a kéréssel a névszervernek.
 - Könnyű magas terhelésű szervert építeni.

Rekurzív DNS lekérdezés

71

Hol van a www.google.com?

- A lokális szerver terhet rak a kért névszerverre (pl. root)
- Honnan tudja a kért, hogy kinek továbbítsa a választ?
 - ▣ Random ID a DNS lekérdezésben

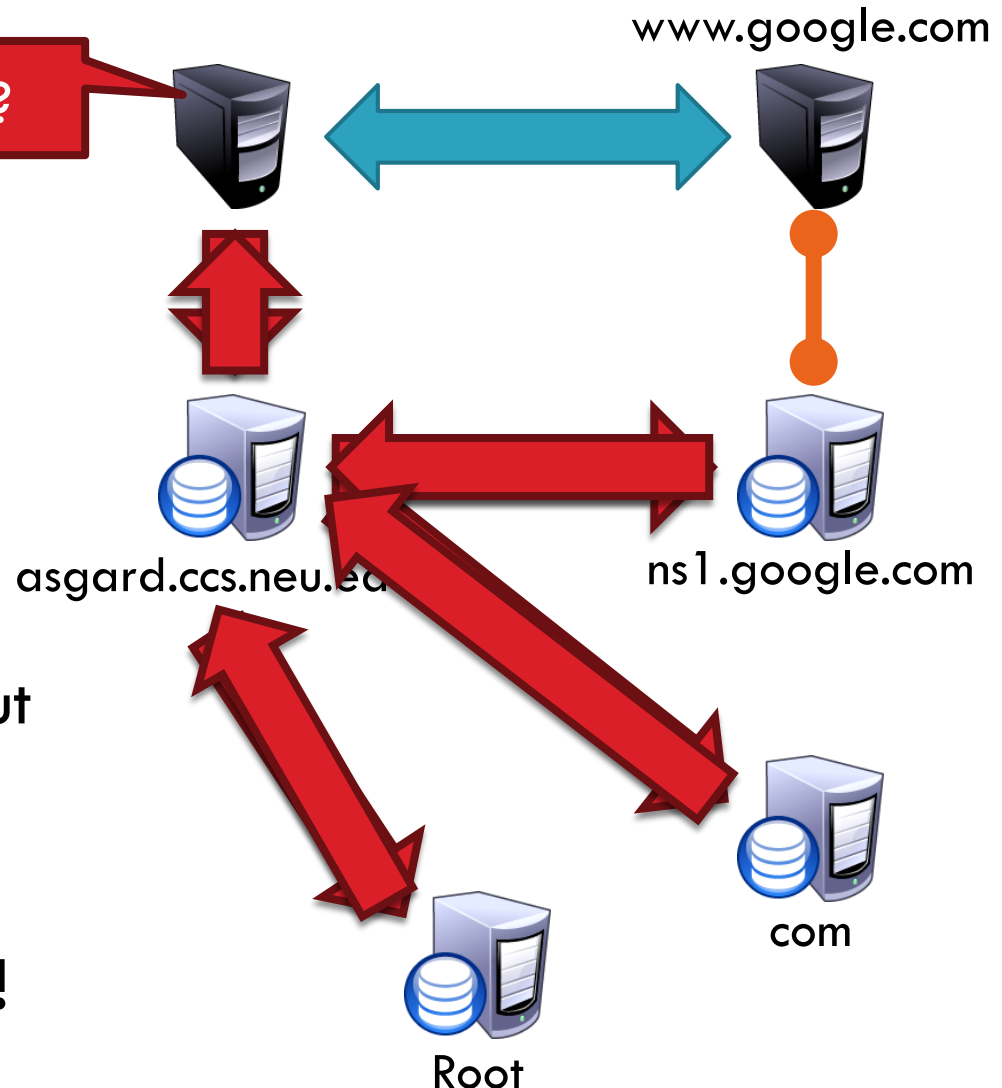


Iteratív DNS lekérdezés

72

Hol van a www.google.com?

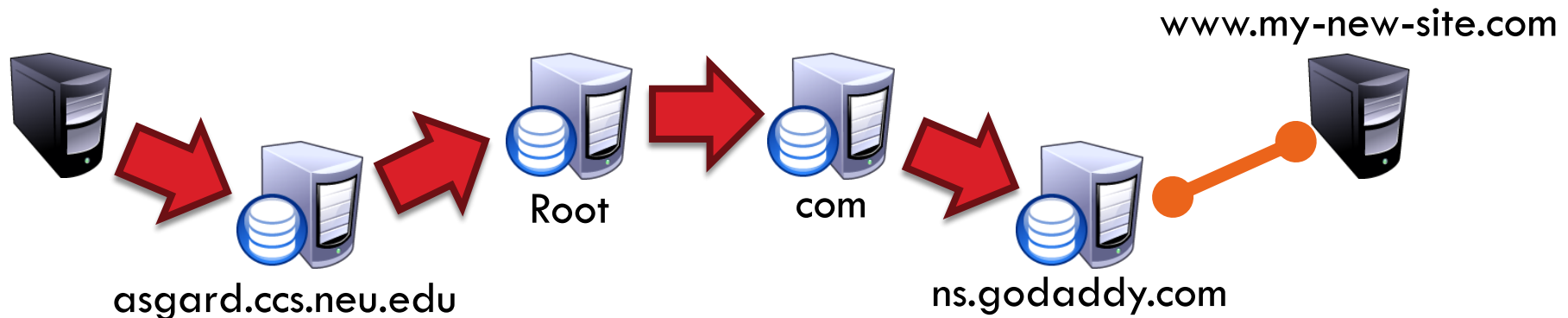
- A szerver mindig a következő kérdezendő névszerver adataival tér vissza
 - “I don’t know this name, but this other server might”
- **Napjainkban iteratív módon működik a DNS!!!**



DNS bejegyzés elterjedése

73

- Van-e a teremben olyan, aki vásárolt már domén nevet?
 - ▣ Észrevettétek-e, hogy kb. 72 óra kell ahhoz, hogy elérhető legyen a bejegyzés után?
 - ▣ Ez a késés a DNS Propagáció/DNS bejegyzés elterjedése



- Miért nem sikerül ez egy új DNS név esetén?

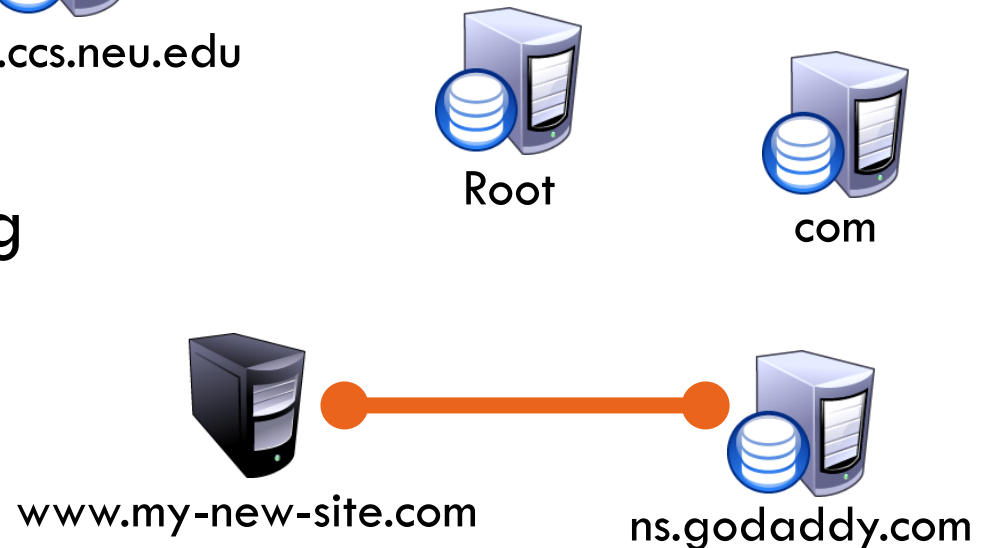
Cachelés VS frissesség

74

- DNS elterjedés késését a cache okozza



- A zóna fájlok 1-72 órig élnek a cache-ben



DNS Erőforrás rekordok (Resource Records)

75

- A DNS lekérdezéseknek két mezőjük van
 - ▣ **name** és **type**
- Az erőforrás rekord válasz egy DNS lekérdezésre
 - ▣ Négy mezőből áll: (**name**, **value**, **type**, TTL)
 - ▣ Egy lekérdezésre adott válaszban több rekord is szerepelhet
- Mit jelent a **name** és a **value** mező?
 - ▣ Ez a lekérdezés típusától (**type**) függ

DNS lekérdezés típusok

76

- Type = A / AAAA
 - ▣ Name = domén név
 - ▣ Value = IP cím
 - ▣ A = IPv4, AAAA = IPv6

- Type = NS
 - ▣ Name = rész domén
 - ▣ Value = a rész doménhez tartozó DNS szerver neve
 - ▣ “Menj és küldd a kérésed ehhez a szerverhez”

Query

Name: www.ccs.neu.edu
Type: A

Resp.

Name: www.ccs.neu.edu
Value: 129.10.116.81

Query

Name: ccs.neu.edu
Type: NS

Resp.

Name: ccs.neu.edu
Value: 129.10.116.51

DNS lekérdezés típusok

77

□ Type = CNAME

- ▣ Name = domén név
- ▣ Value = kanonikus név
- ▣ Alias nevek használatához
- ▣ CDN használja

Query

Name: foo.mysite.com
Type: CNAME

Resp.

Name: foo.mysite.com
Value: bar.mysite.com

□ Type = MX

- ▣ Name = emailben szereplő domén név
- ▣ Value = mail szerver kanonikus neve

Query

Name: ccs.neu.edu
Type: MX

Resp.

Name: ccs.neu.edu
Value: amber.ccs.neu.edu

Fordított lekérdezés (PTR rekord)

78

- ❑ Mi a helyzet az IP → név leképezéssel?
- ❑ Külön hierarchia tárolja ezeket a leképezéseket
 - ▣ Gyöker pont: in-addr.arpa és ip6.arpa
- ❑ DNS rekord típusa (**type**): PTR
 - ▣ Name = IP cím
 - ▣ Value = domén név
- ❑ Nincs garancia arra, hogy minden IP címre működik

Query

Name: 129.10.116.51
Type: PTR

Resp.

Name: 129.10.116.51
Value: ccs.neu.edu

DNS as Indirection Service

79

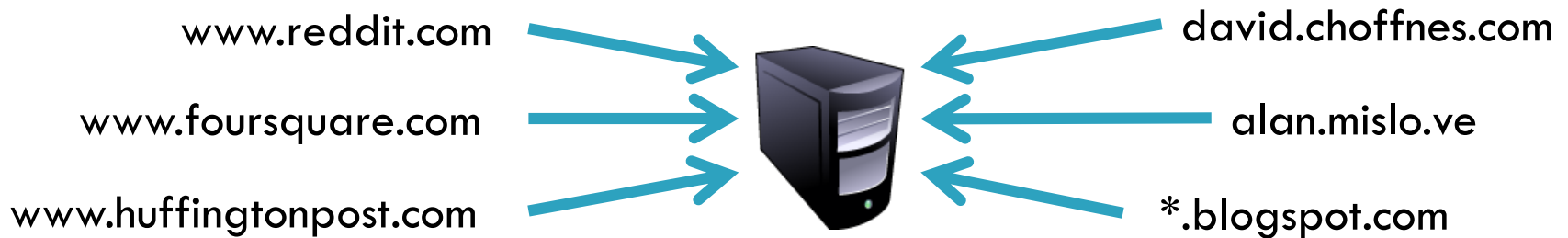
- DNS számos lehetőséget biztosít
 - ▣ Nem csak a gépekre való hivatkozást könnyíti meg!

- Egy gép IP címének lecserélése is triviális
 - ▣ Pl. a web szervert átköltöztetjük egy új hosztra
 - ▣ Csak a DNS rekord bejegyzést kell megváltoztatni!

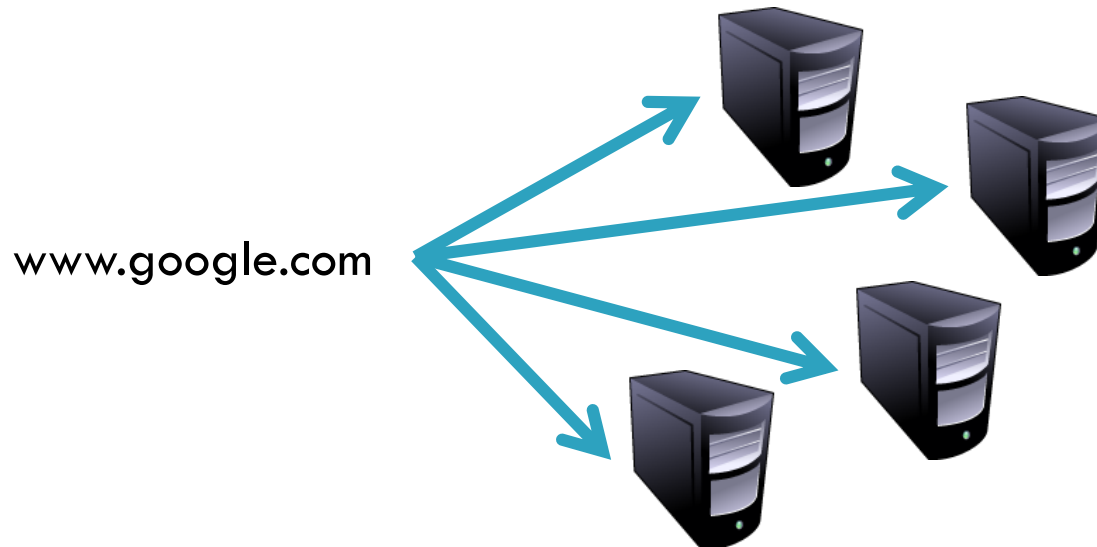
Aliasing/Kanonikus nevek és Load Balancing/Terhelés elosztás

80

- Egy gépnek számos alias neve lehet

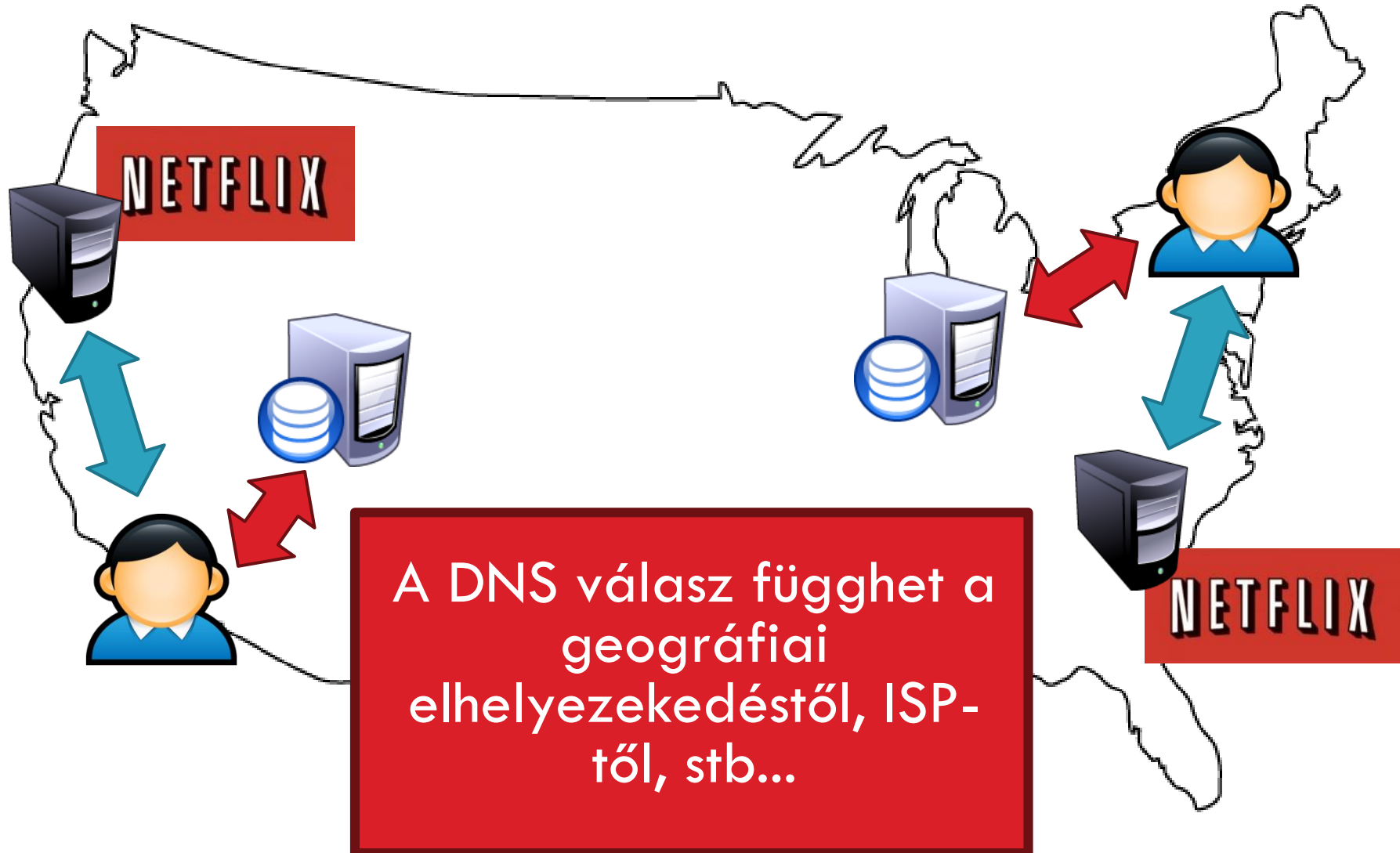


- Egy domén névhez számos IP cím tartozhat



Content Delivery Networks

81



A DNS fontossága

82

- ❑ DNS nélkül...
 - ▣ Hogyan találjuk meg egy weboldalt?
- ❑ Példa: a mailszervered azonosít
 - ▣ Email címet adunk meg weboldalakra való feliratkozásnál
 - ▣ Mi van, ha valaki eltéríti a DNS bejegyzést a mailszerveredhez?
- ❑ DNS a bizalom forrása a weben
 - ▣ Amikor a felhasználó begépel a www.bankofamerica.com címet, azt várja, hogy a bankja honlapja jelenjen meg.
 - ▣ Mi van, ha a DNS rekordot meghackelték?

Köszönöm a figyelmet!