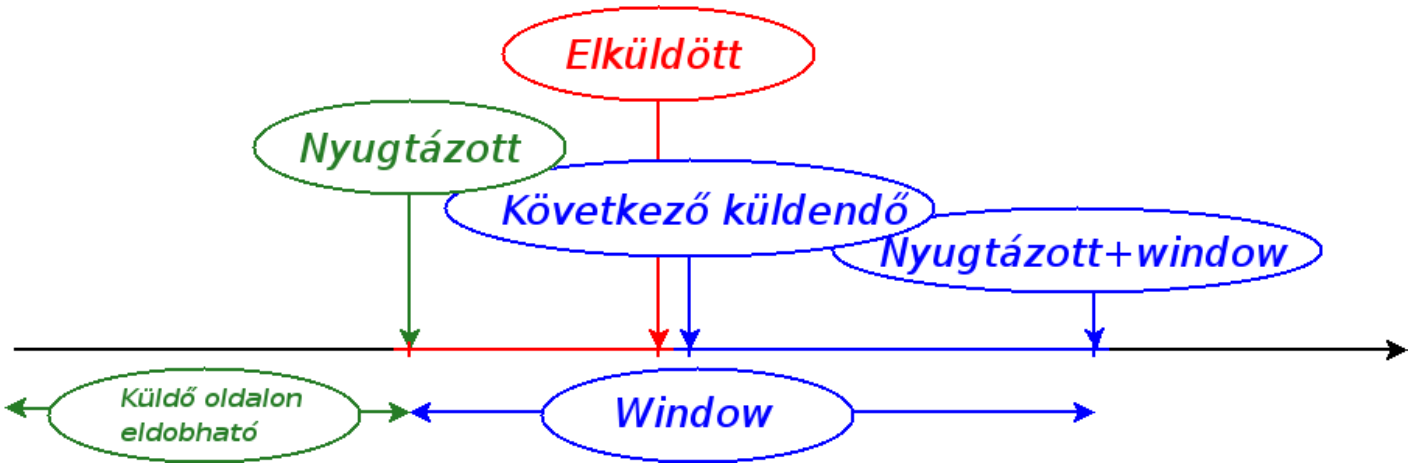
Számítógépes hálózatok – 8. előadás

TCP adatforgalom

Hogyan adjunk, ha ki akarjuk használni a sávszélességet?

- Figyelembe kell venni a késleltetést is (TFTP példa)
- Akkor küldhetünk folyamatosan, ha az első csomagunk nyugtájának megérkezéséig folyamatosan adhatunk
  - A csomagfordulási idő, **RTT** – Round Trip Time korlátoz
- *Sávszélesség * RTT* elküldött bit legyen a window
  - Az első nyugta megérkezésekor ekkor *sávszélesség * késleltetés* úton levő bit
- Pl. `100 Mb/sec` vonal, `0,01` sec RTT: `1Mbit = 125 KByte`
- Pl. `100 Mb/sec` vonal, `2` sec RTT: `200Mbit = 25 MByte` !

Sliding window



- A nyugtázott+window sorszámig folyamatosan küldhető adat
  - Rendszerint nem tölti ki az egész window-t
- A nyugtázott+window sorszámnál nagyobb sorszámú oktet nem küldhető
- A window folyamatosan csúszik jobbra az ábrán
- A window-t a fogadó csökkentheti/növelheti
- A window mérettel a fogadó szabályozhatja a küldés ütemét
  - Flow control: az alkalmazás igényeihez igazodhat
    - Pl. a window lehet az alkalmazás által vételre felkínált buffer mérete
  - Congestion control: torlódás és csomagvesztés elkerülésére
- Egy nyugta mindig addig az SN-ig nyugtáz mindent, nem csak az utolsó csomagot
- A window nyílik, ha a jobb széle jobbra mozdul
- A window becsukódik:
  - ha a hirdetett window 0,
  - az adó kimerítette a küldhető adatmennyiséget,
  - a vevő mindent nyugtázott
- A window zsugorodik (shrinks), ha a jobb széle balra mozdul
  - Lehetséges, de ellenjavalt
  - Vannak protokollok, ahol ez lehetetlen
- Újraküldés

- Ha három ugyanolyan ACK-t kapok, onnan kezdve újraküldök
- Ha timeout (RTO) lejár, és nem kapok ACK-t

TCP nyugták

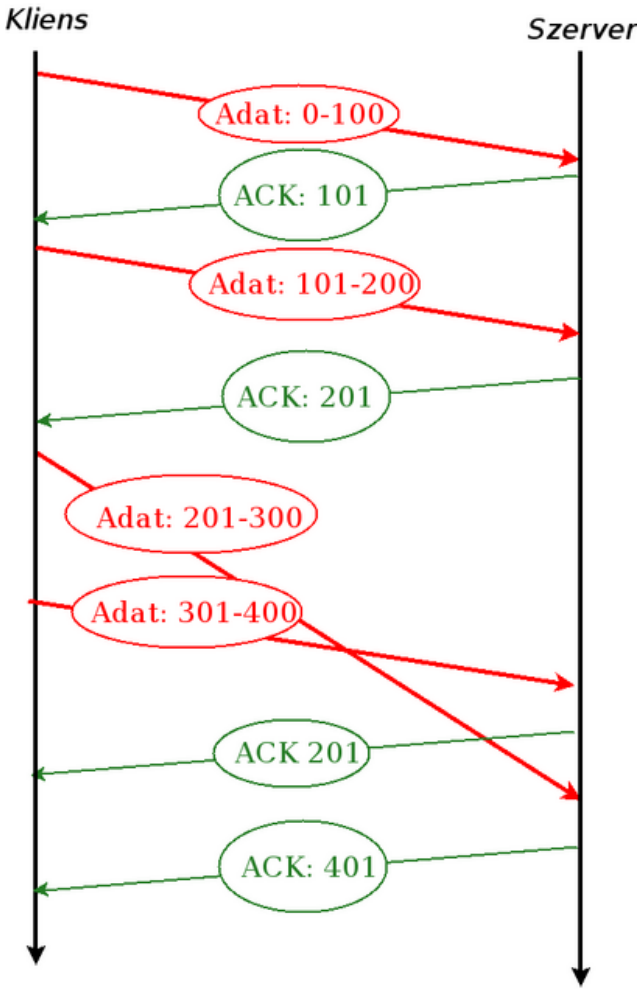


Figure 2: Csomag sorrend felcserélődés és TCP nyugták

Window scale opció

- **RFC1323**
- 3 byte-os opció a TCP fejrészben:

```
+--------+-------+-----------+
| kind=3 | len=3 |shift count|
+--------+-------+-----------+
```

- `2`-nek a *shift count* -adik hatványa lesz a window paraméter mértékegysége
- `0` azt jelenti, hogy a window byte-okban számolandó (nincs scale)
- `3` azt jelenti, hogy `8` byte-os darabokban
- Maximum 14 a megengedett = `2^30` = `1G`
- A SYN flag-gel együtt küldhető a three-way handshake során
- Akkor él, ha a SYN/ACK-ban is megismétli a partner
- Lehet különböző a két irányban

Delayed ACK

- A TCP réteg nem feltétlenül küld nyugtát, ha tud
  - Legalább 40 byte-ot kell küldeni!
- Egy timer lejártát megvárja, hátha addig lesz:
  - elküldendő adat, amivel „mellékesen” nyugtáz: piggyback nyugta
  - újabb bejövő adat, amit nyugtázhat.

ACK generálás

Esemény	Akció
A nyugtázotthoz, és vett sorszámhoz képest folyamatosan jön egy szegmens	Késleltetetés. Ne küldj ACK-t, még 200 ms-ig.
A vett sorszámhoz képest folyamatosan jön egy szegmens, van már nyugtáznivaló (timer ketyeg)	Azonnal küldj ACK-t, csúsztasd a window-t.
A vett sorszámmal összevetve kimaradást jelző sorszámmal jön csomag	Azonnal küldj ACK-t az előző csomagot nyugtázva (duplicate ack).
Egy lyukból eddig hiányzó, a lyuk aljához illeszkedő csomag érkezik	Azonnal küldj ACK-t.

TCP Retransmission timeout (RTO): a nyugtázatlan csomag újraküldésének időzítése

- Ha túl rövid, felesleges újraküldés történik
- Ha túl hosszú, nagy csuklást okoz egy csomag elvesztése
- **RTT** – Rount Trip Time
  - TCP szegmens újraküldésnél ez határozza meg a timeout-ot
  - Egy szegmens elküldése és a hozzá tartozó ACK megérkezése közti idő
    - Időben változik, sok mindentől függ
    - Átlagot érdemes venni
- Exponential Backoff: ha újraküldünk, az RTO-t duplázzuk, egy határig (jellemző érték: 9 perc)

Hogyan függ az RTO az RTT-től ?

- *Becsült-RTT = a * Becsült-RTT + (1-a) * Mért-RTT*, ahol *0<a<1*
- *RTO = b * Becsült-RTT*, ahol *b>1*
- Tipikus értékek: *a = 0.9, b = 2*
- Ravaszságok
  - *a* nem konstans: ha hirtelen nő a *Mért-RTT*, legyen kisebb: gyorsabban vegyük figyelembe a romlást
  - Karn algoritmus:
    - Ha ismételtünk egy csomagot, az arra jövő ack-ból ne számoljunk RTT-t
      - Nem tudhatjuk, hogy az ismételt csomagra jött, vagy az eredetire
    - Ha ismétlünk egy csomagot, az RTO-t ne módosítsuk (most dupláztuk exponential backoff miatt!), míg ismétlés nélkül nem csikarunk ki ack-t

Interaktív adatforgalom

- Kis csomagok
- Lehetőleg azonnali echo
- TOS: minimize delay
- Nagy a csomagokon az overhead: sokszor 1 byte-on 40, ráadásul a nyugta!

Nagle algoritmus

- Probléma: a lassú vonal végén ülő felhasználó gépelésével mindig újabb csomagokat generál
  - Ha torlódnak a csomagok, ezzel tetézi a bajt
- **RFC896**
- Nem küldünk újabb csomagot, bufferelünk, míg van nyugtázatlan kintlevő csomag
- Timeout után mindenképpen ürítünk
- Önszabályozó: ha gyors a hálózat, nincs is hatása
- Egyes alkalmazásoknál hátrányt jelent
  - X terminálok
  - Több karakteres vezérlőszekvenciák (pl. PF gomb)
  - Ki lehet kapcsolni (klasszikus mondás: `TCP_NODELAY`)

Torlódás kezelése

- Számítani kell rá, hogy sok eszközön megy át a csomag, amiket túlterhelhetünk, csomagokat dobnak el
- Ha bambán újraküldünk: növeljük a bajt, a túlterheltséget
- Congestion control: mit csináljunk, ha torlódás lett
- Congestion avoidance: mit csináljunk, hogy elkerüljük a torlódást
- Összefoglaló: **RFC2581** TCP Congestion Control
  - Az egyik szerző R.W. Stevens
  - Újabb változat **RFC5681**
- Nem csak a technológia tanulságos, hanem a szemléletmód is: a congestion avoidance a szolidaritásra szép példa (ahogy a DNS a szubszidiaritásra)

Slow start

- Nem csak a fogadó, a küldő is flow controlt alkalmaz
- Congestion window: a hálózat kímélése érdekében bevezetett ablak
  - A küldő becslése, a TCP masina belső változója: *cwnd*
  - A receive window és a congestion window jobb szélének minimuma határozza meg, hogy küldhetek-e csomagot
- Kezdetben a Congestion window (klasszikusan) 1 MSS méretű
  - Lehet 10 MSS, sőt van javaslat még nagyobbra
- Minden ack-zott szegmens 1 MSS-sel növeli a *cwnd*-t
  - Ideális esetben a *cwnd* nem korlátoz
  - A vevő képességét, illetve a hálózati kapacitást teljesen kihasználva tömjük a hálózatot
- Slow start alatt az átvitel exponenciálisan nő
- A slow start alkalmazásával elkerüljük, hogy azonnal bajt okozzunk
- Ha torlódást észlelünk, a congestion avoidance algoritmus jut szóhoz

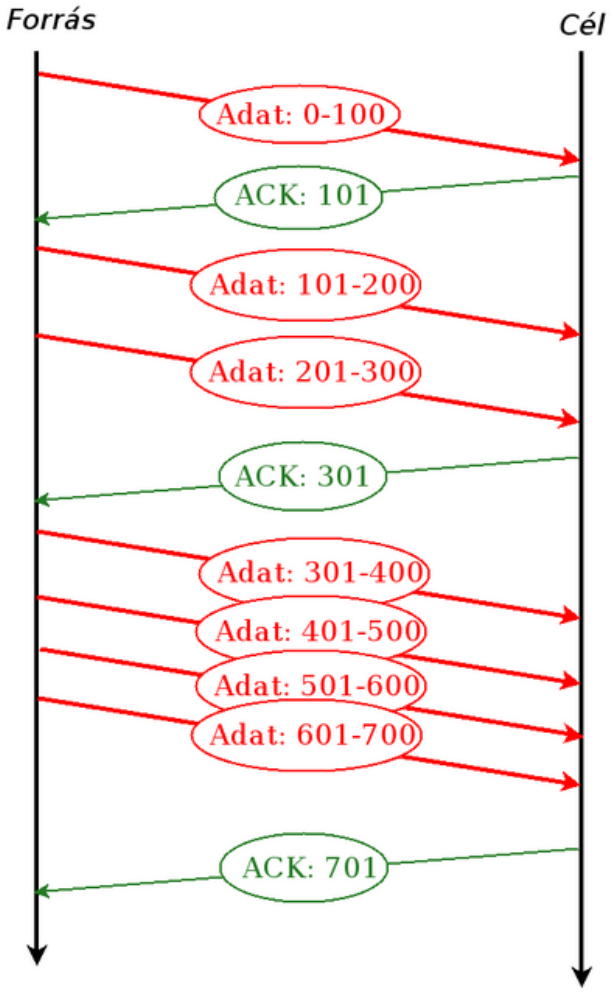
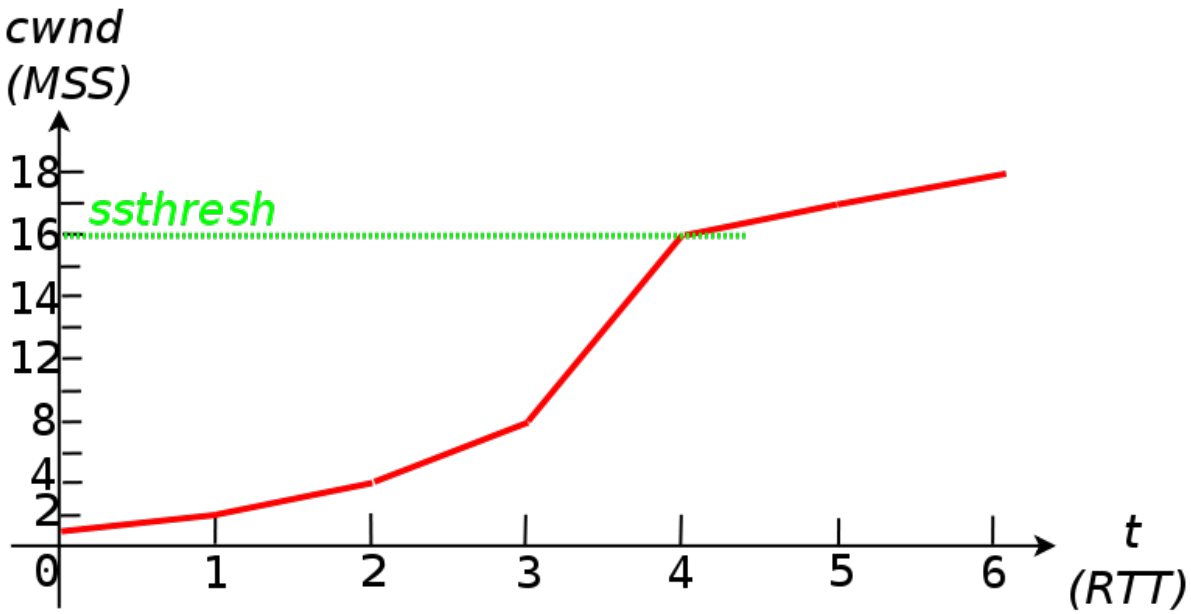


Figure 3: TCP slow start

Congestion avoidance

- Congestion avoidance-nál a *cwnd*-t 1 szegmensnyivel növeljük minden RTT alkalmával
  - lineáris növekedés
- Additive increase, multiplicative decrease (AIMD)
- Bevezetünk egy új változót: slow start threshold, *ssthresh*
  - Kezdetben a receiver window
- Slow startot alkalmazunk, ha *cwnd* < *ssthresh*, congestion avoidance-t különben
- Ha congestion-t észlelünk (RTO, vagy tripla ack), akkor
  1. Újraküldjük a szegmenst (fast retransmit)
  2. A nyugtázatlan_byte-ok/2-re, de legfeljebb 2 MSS-re csökkentjük *ssthresh*-t (multiplicative decrease)
  3. Ha RTO történt, akkor slow start-tal indítunk: *cwnd = 1*
  4. Ha tripla ack, akkor *cwnd = ssthresh + 3*MSS*



Fast retransmit és fast recovery

- Ha a küldő tripla ACK-kat kap ugyanarra az sequence numberre, akkor congestion-ra következtet
- Nem várja ki az RTO-t, hanem újra küld: ez a fast retransmit
- Ez után nem kezd elölről a Slow Start szerint, hanem a congestion avoidance algoritmust alkalmazza (*cwnd=sshthresh+3MSS*): ez a fast recovery

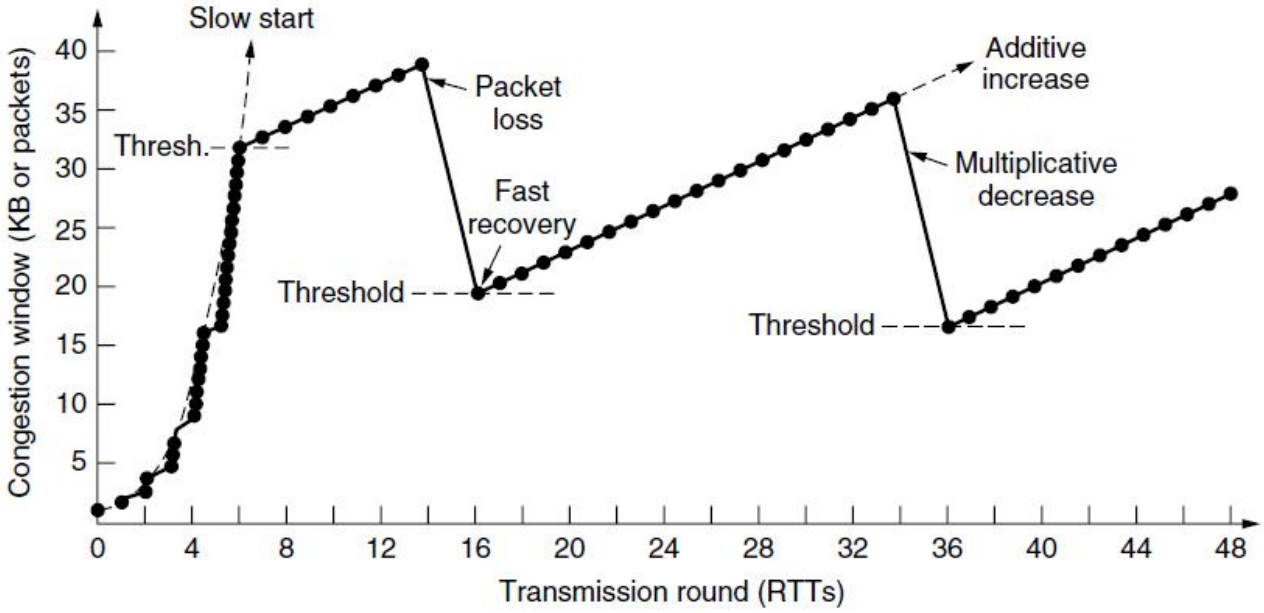


Figure 5: TCP forgalom kisimitas (Forras: Tanenbaum konyv)

RED: Random Early Detection

- A TCP congestion control eljárások egészségesebbé tették az internetet
  - A TCP kapcsolatok „reagálnak” a torlódásokra
  - Megszűnt a 80-as években előforduló „congestion collapse”
- A hálózat belsejében is lépéseket kell tenni:
  - Active queue management a routerekben
- **RFC2309**: Internet Performance Recommendations
- Első megoldás
  - A router queue-kat kezel egyes interfészeihez
  - Ha a queue-ba már nem fér egy csomag, eldobja
  - Hátrányok:
    - Egyes kapcsolatok monopolizálhatják az erőforrásokat
    - Ha beáll egy telítettség, nehéz kivergődni belőle
    - Újabb lökések tovább rontják a helyzetet
- Arra kell törekedni, hogy tartósan kicsik legyenek a Q-k
  - Úgyis lesznek lökések, amiket el kell viselni
  - Interaktív kapcsolatoknál a hosszú Q-k kibírhatatlanok a nagy késleltetés miatt
- RED működés:
  - Minden csomagot bizonyos valószínűséggel eldobunk
  - A valószínűség annál nagyobb minél nagyobb volt az elmúlt időszakban a Q hossza
  - Egy-egy eldobott csomag nem okoz nagy gondot, de zsinórban eldobott sok csomag igen
  - Le tudunk lassítani minden adatfolyamot, flow-t
  - Folytonosan mérjük az átlagos Q hosszt
    - A régebbi időket exponenciálisan kisebb súllyal vesszük figyelembe
    - A hossz mértékegysége csomag, de byte is lehet
  - Változók: minimum *(m)* és maximum *(M)* küszöb, $0 < maxp < 1$
  - Ha a Q mért hossza *m*-nél kisebb, nem dobunk el csomagot, ha *M*-nél nagyobb minden csomagot eldobunk
  - Ha a kettő közt van akkor a mért hosszal arányos *0* és *maxp* közti valószínűséggel dobjuk el a csomagot
- WRED - Weighted RED: vannak „egyenlőbb” csomagok: valamilyen szempont szerint bizonyos csomagokat kevésbé valószínűen dobunk el
  - IP címek
  - Protokoll
  - TOS
  - …

Persist timer

- Ha a fogadó bufferei elfogytak, bezárja a window-t
- Ha újra tud fogadni, nyitja: ACK megismételt acknowledgement numberrel, de nem 0 window-val
- Mi van, ha ez az ACK elvész?
  - A fogadó nem tudhatja, hogy van-e még a küldőnek mondandója
  - A küldő próbálkozik: küld egy 1 byte-os csomagot (hite szerint window-n kívül!) (window jobb széle után)
    - Ezt ismétli persist timer-enként
    - A persist timer exponenciálisan nő (Exponential Backoff)
      - Tipikusan 1,5 sec-ről indul, 1 percnél nem lesz nagyobb

Silly window szindróma

- Ha a vevő oldalon kicsi (akár 1 byte) szabadul fel, lehet 1 byte a window

- A küldő 1 byte-ot küld, a vevő megint 1 byte-tal nyit é.í.t.
- Erőforráspocsékolás
- Elkerülésére
  - A vevő nem nyitja a window-t, csak akkor, ha MSS nagyságrendűt nyithat
  - A küldő nem küld, hacsak
    - MSS-nyit küldhet
    - Mindent küldhet, amit az alkalmazás kért
      - Feltéve, hogy Nagle ezt nem tiltja (nem vár ACK-t, és Nagle nincs kikapcsolva)

Keep-alive timer

- TCP kapcsolatok eredendően nem bomlanak forgalom hiányában sem
  - Akár hónapokig „élve” maradnak
  - Közben a middlebox-okat újraindíthatták, átkonfigurálhatták, akár kicserélhették
- Az eredeti szándék szerint az alkalmazások használhatnak „AYT” (Are You There) mechanizmust
- Mégis divatba jött a keep-alive mechanizmus
- **RFC1122** leírja, bár ellenjavallva
  1. Feleslegesen lebonthat később még használható kapcsolatokat
  2. Feleslegesen hálózati erőforrásokat használ
  3. Kidobott pénz, ha forgalom után fizetünk
- **RFC1122** megengedi, hogy opcionálisan használni lehessen
- Keep-alive time: jellemzően 2 óra, általában rendszerparaméter
- Ha egy kapcsolat keep-alive ideig néma, akkor a mechanizmust alkalmazó oldal
  - Küld egy próba csomagot
    - Jellemzően 0 adattal, a küldött utolsó sorszámú SN-nel
    - Ez window-n kívül lesz! (window bal széle előtt)
- A másik oldal erre ACK-t küld

Wireshark és TCP

- A wireshark nagyon alkalmas arra, hogy sok TCP-vel kapcsolatos dolgot megértsünk, felfedezzünk
- Follow tcp-stream → tcp streamgraph → Stevens

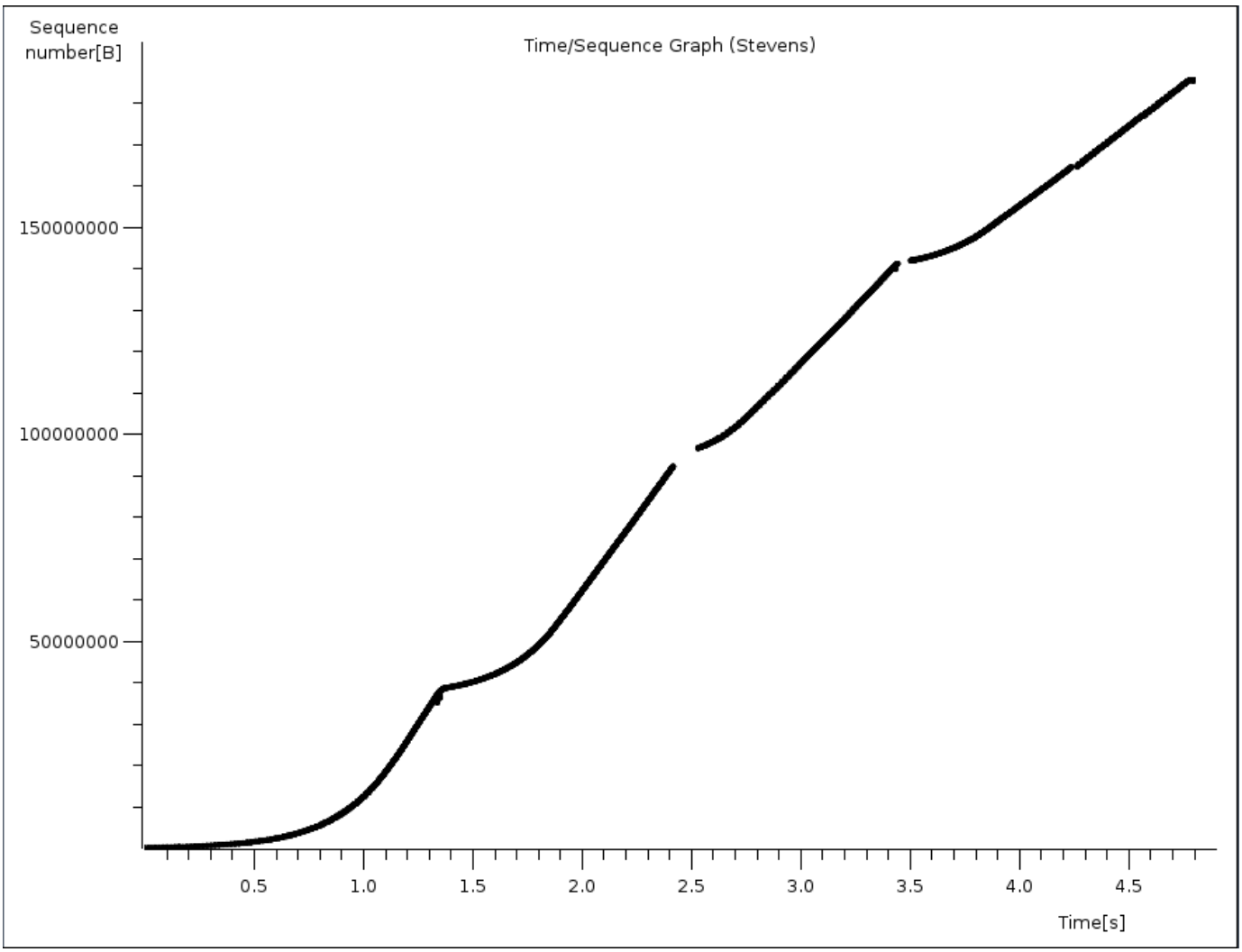


Figure 6: TCP slow start - Stevens graph