

Szállítási réteg (L4)

Gyakorlat



- A TCP-t nagyon sok környezetben használják
- A főbb mechanizmusok ismerete fontos
 - A programozónak
 - A hálózati szakembernek
 - Wired
 - Wireless
- Lassan az ipari környezetbe is beszivárog

- Kapcsolat fogalma
 - 5-tuple
- Kapcsolat kiépítése
 - UDP - nincs
 - TCP
 - 3 way handshake
 - Miért van rá szükség?

- Netstat parancs

TCP	127.0.0.1:1906	localhost:1907	ESTABLISHED
TCP	192.168.1.147:53699	13.77.87.52:https	ESTABLISHED
TCP	192.168.1.147:53703	91.190.216.57:12350	ESTABLISHED
TCP	192.168.1.147:53737	64.4.23.152:40008	ESTABLISHED
TCP	192.168.1.147:53759	108.177.96.188:5228	ESTABLISHED
TCP	192.168.1.147:53772	40.77.226.192:https	ESTABLISHED
TCP	192.168.1.147:54512	a104-96-129-73:https	CLOSE_WAIT
TCP	192.168.1.147:54513	a104-96-129-73:https	CLOSE_WAIT
TCP	192.168.1.147:54514	a104-96-129-73:https	CLOSE_WAIT

- Adatküldés: szegmensek
- Hibakezelés
 - ICMP: port nem elérhető
 - Loss: nincs visszajelzés
- Sáv szélesség, késleltetés

The Evolution of TCP



BME-TMIT

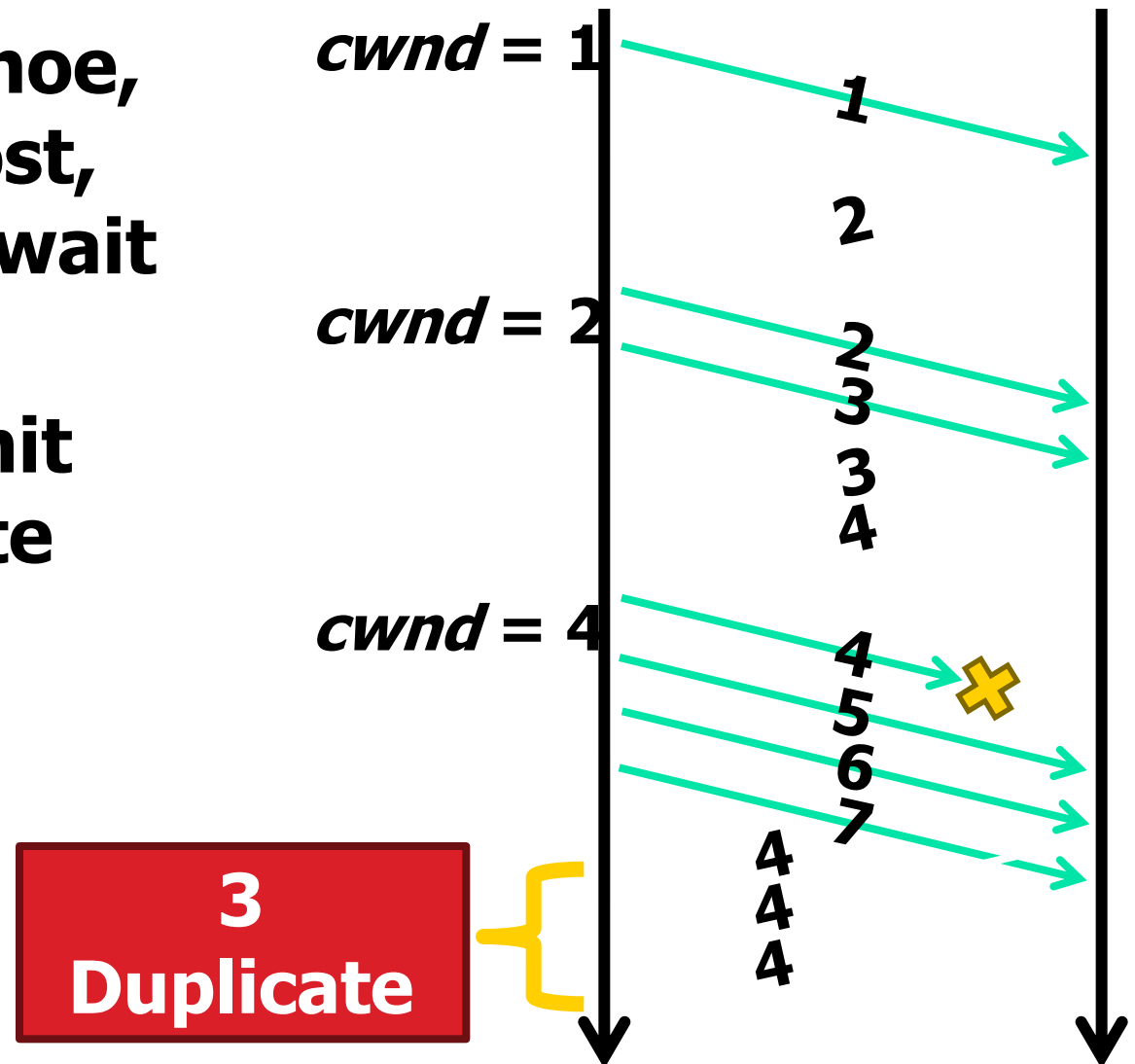
- TCP Tahoe
 - Kezdeti verzió
- A TCP 1974-ben volt kitalálva!
 - Manapság rengeteg változata van a TCP-nek
- Kezdeti, elterjedt: TCP Reno
 - Tahoe, plus...
 - Fast retransmit
 - Fast recovery

TCP Reno: Fast Retransmit



BME-TMIT

- **Problem: in Tahoe, if segment is lost, there is a long wait until the RTO**
- **Reno: retransmit after 3 duplicate ACKs**



TCP Reno: Fast Recovery



BME-TMIT

8

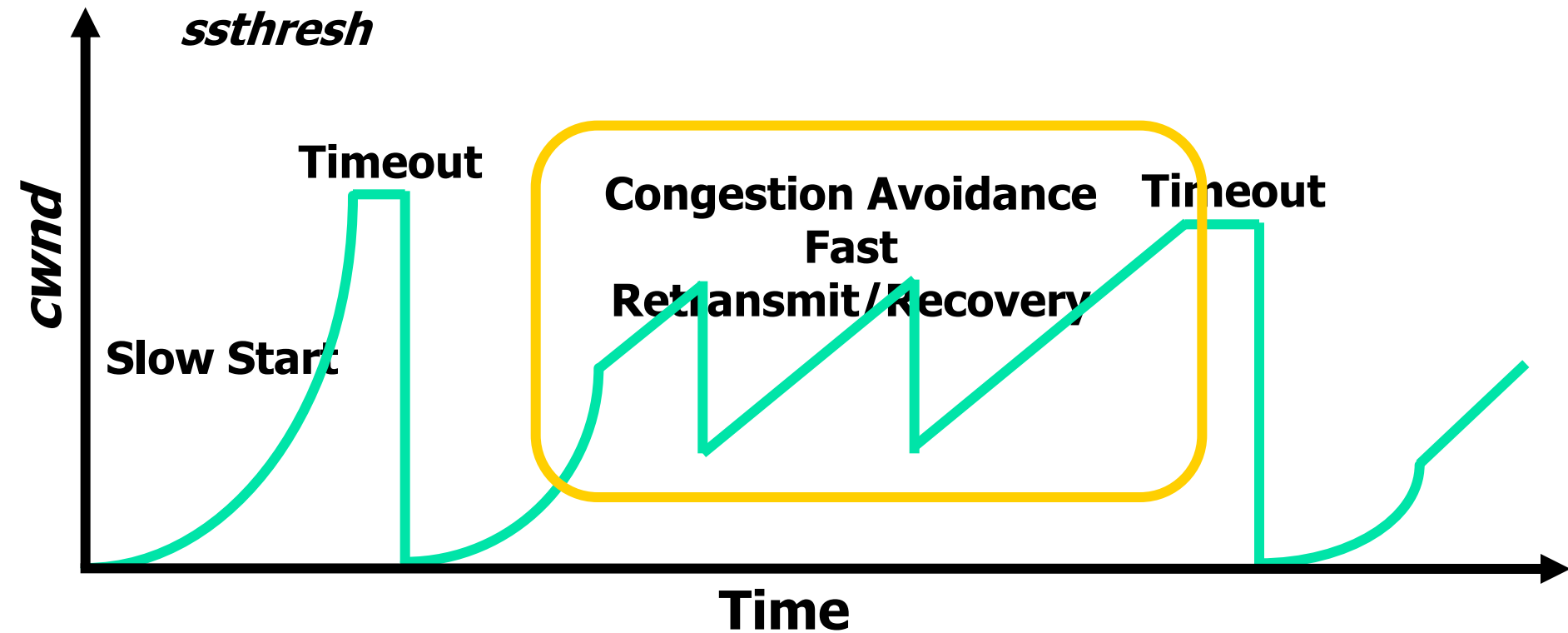
- After a fast-retransmit set *cwnd* to *ssthresh*/2
 - i.e. don't reset *cwnd* to 1
 - Avoid unnecessary return to slow start
 - Prevents expensive timeouts
- But when RTO expires still do *cwnd* = 1
 - Return to slow start, same as Tahoe
 - Indicates packets aren't being delivered at all
 - i.e. congestion must be really bad

Fast Retransmit and Fast Recovery



BME-TMIT

9



- At steady state, $cwnd$ oscillates around the optimal window size
- TCP always forces packet drops

Many TCP Variants...



BME-TMIT

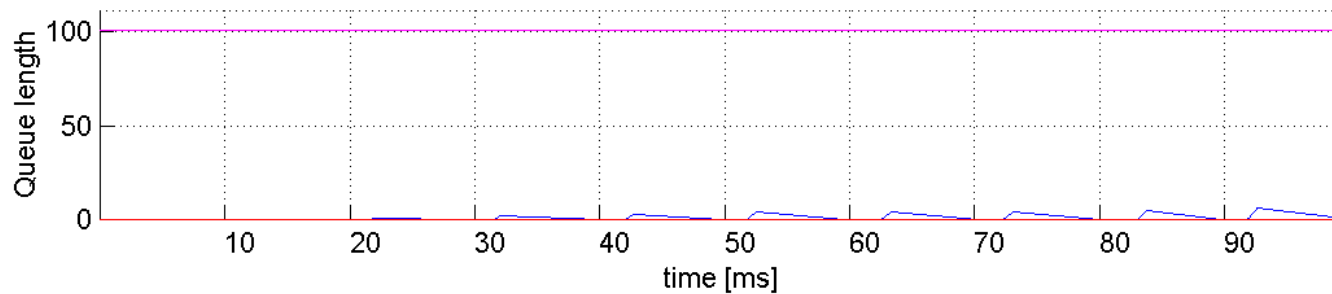
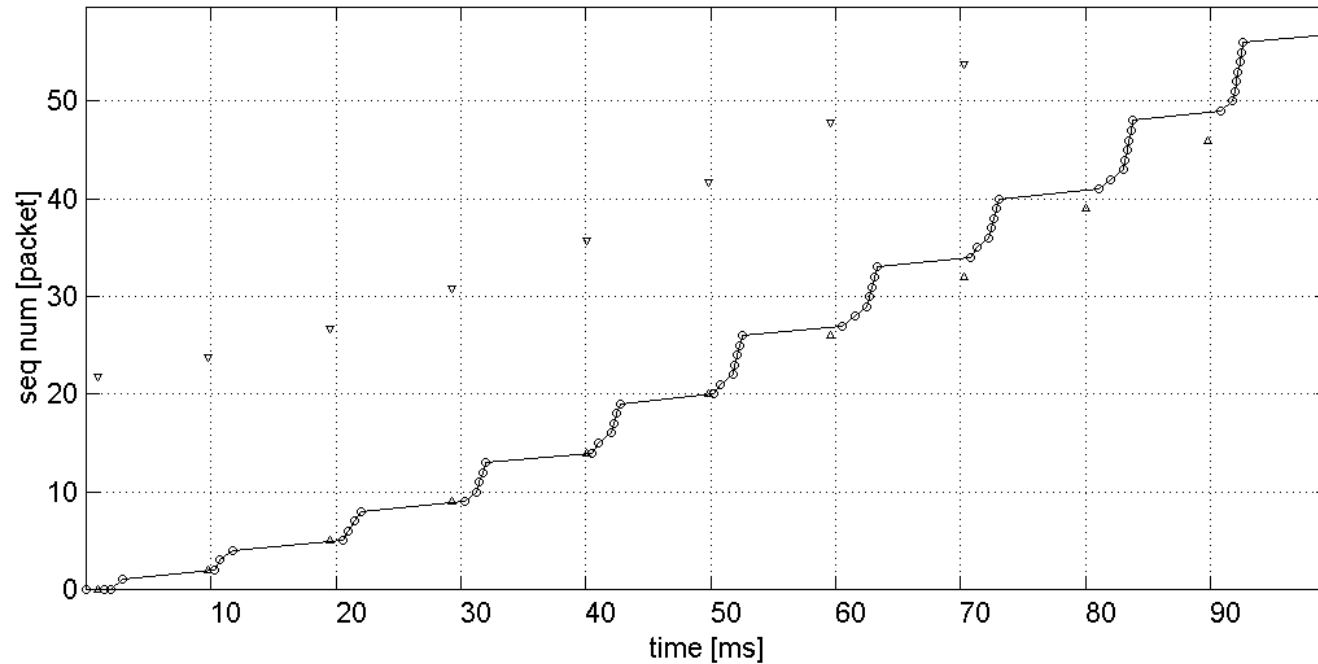
- Tahoe: the original
 - Slow start with AIMD
 - Dynamic RTO based on RTT estimate
- Reno: fast retransmit and fast recovery
- NewReno: improved fast retransmit
 - Each duplicate ACK triggers a retransmission
 - Problem: >3 out-of-order packets causes pathological retransmissions
- Vegas: delay-based congestion avoidance
- And many, many, many more...

- What are the most popular variants today?
 - Key problem: TCP performs poorly on high bandwidth-delay product networks (like the modern Internet)
 - Compound TCP (Windows)
 - Based on Reno
 - Uses two congestion windows: delay based and loss based
 - Thus, it uses a *compound* congestion controller
 - TCP CUBIC (Linux)
 - Enhancement of BIC (Binary Increase Congestion Control)
 - Window size controlled by cubic function
 - Parameterized by the time T since the last dropped packet

- Bandwidth delay product
- 8k – korlátozott sávszélesség
 - Telítődés
- 64K
 - jobb
 - Window scale option

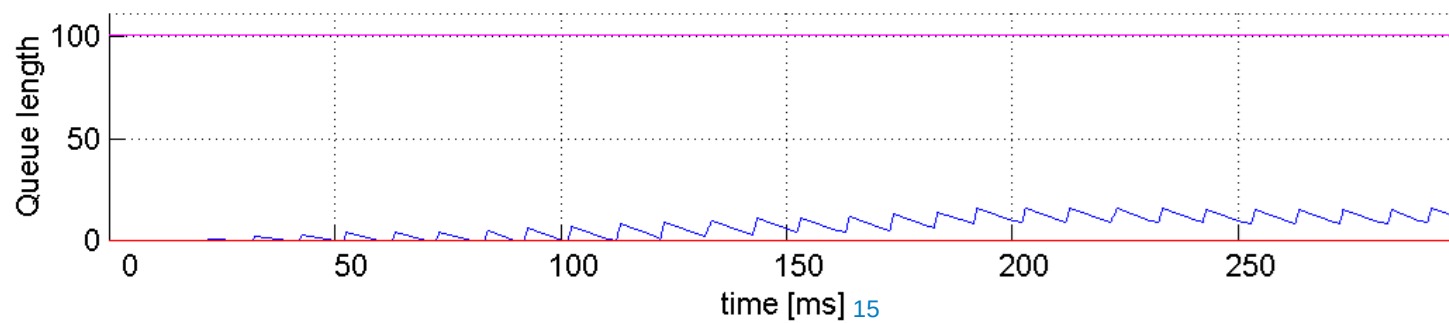
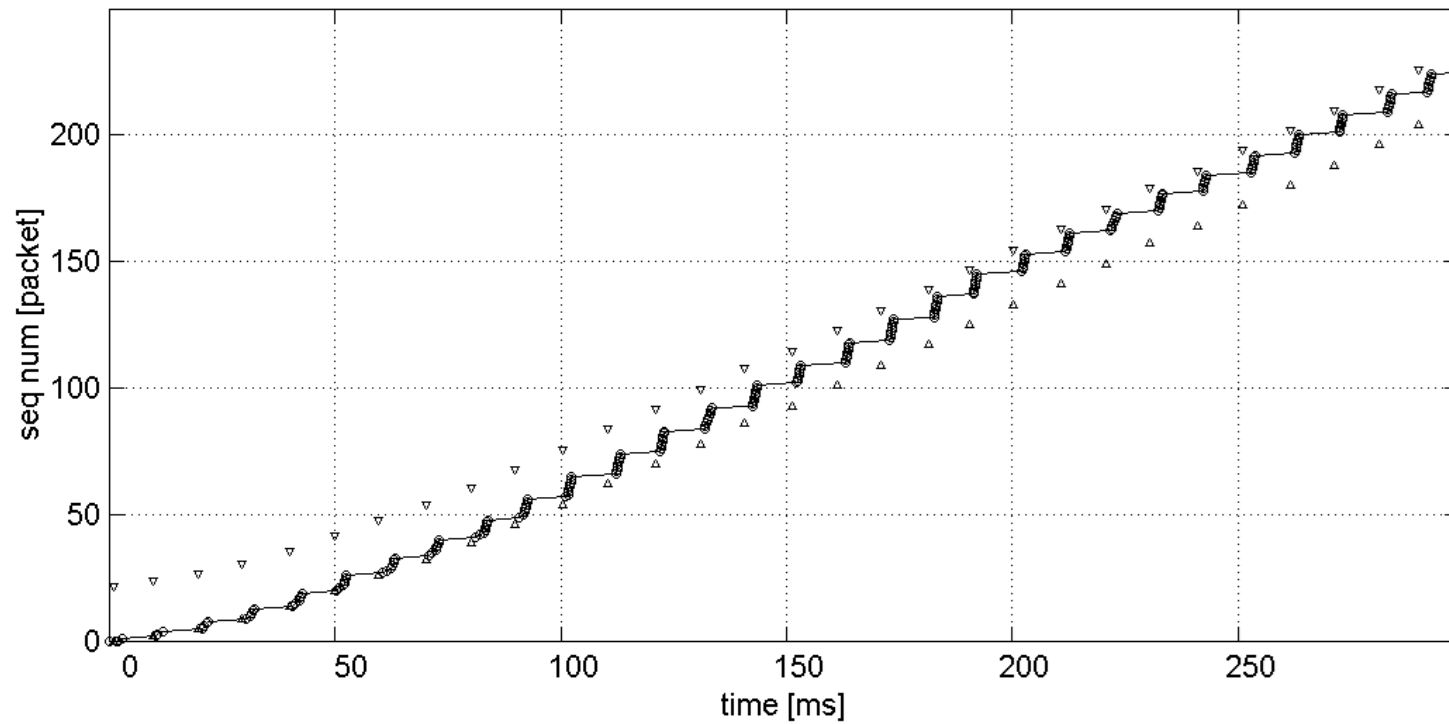
No.	Algo.	Sender	Traffic	# conn.	Receiver	Kbytes/sec	# ovfl.
413	None	Linux 2.1	one.1024	1	Linux 2.0	1097.7 (gw)	0

Trace: 413



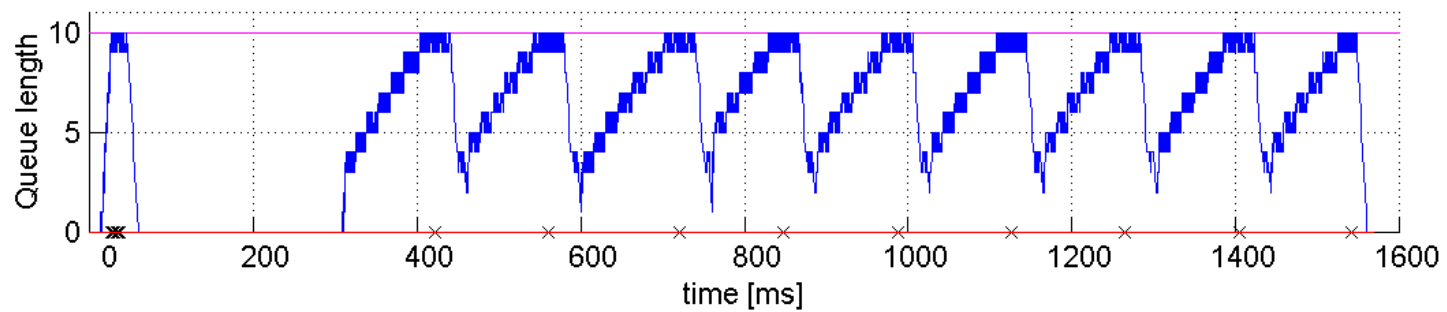
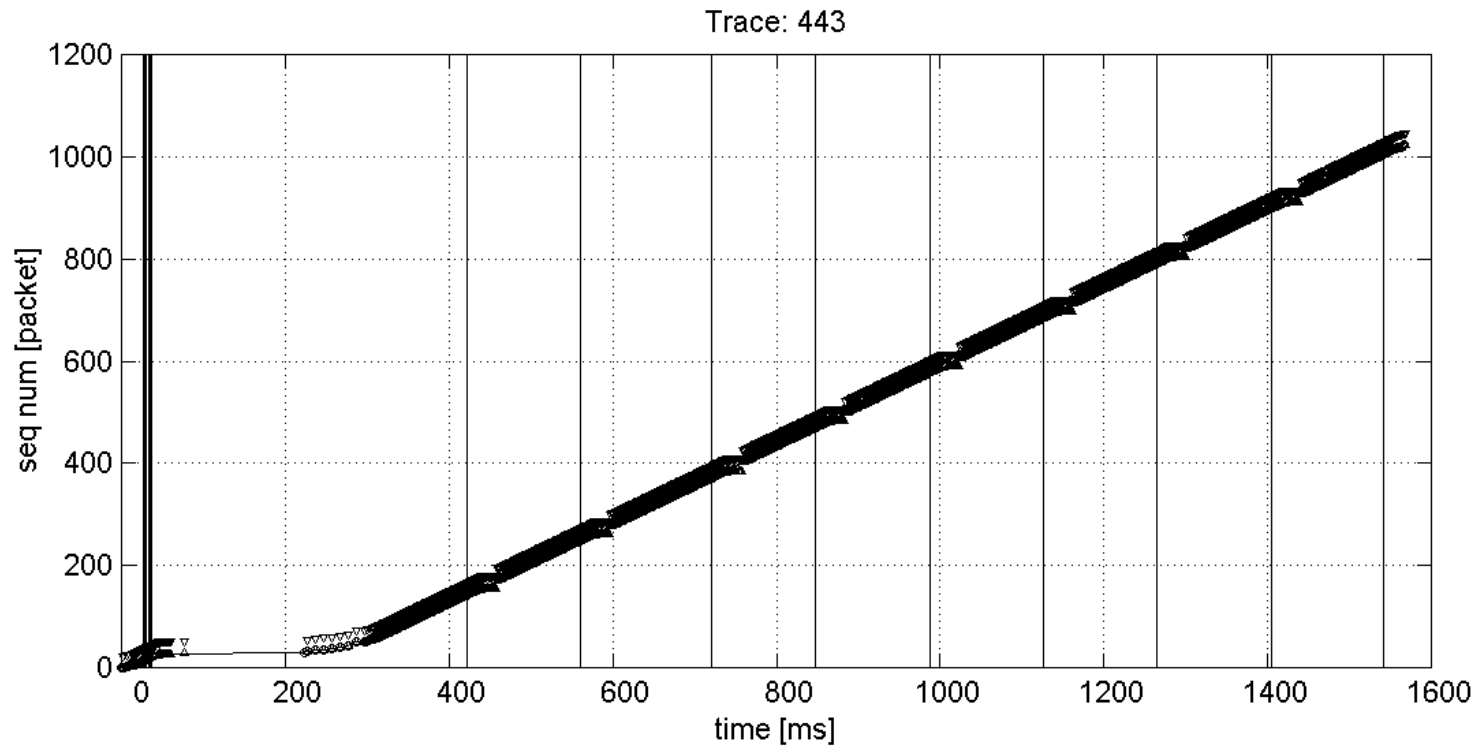
- A fogadó csak egy ACK-t küld egy csomagcsoportra
- Minden ACK érkezése a küldőnél egy csomagbörszt küldését eredményezi
- A küldő a *cwnd*-t minden egyes ACK érkezésekor eggyel növeli
- A sorok hossza növekszik a börsztök érkezésekor, a börsztök méretei pedig növekednek az idővel
- ACK érkezések 10 ms-ként

Trace: 413



- Kb 200ms után, a küldő eléri a fogadó meghirdetett ablakméretének felső határát
- A sorhosszúság kb. 15 körül stabilizálódik
- A sorok hossza a börsztök érkezésekor növekszik – ezeket a fogadó egy-egy csomagcsoportot nyugtázó ACK-ra küldött
- A sorok hossza fokozatosan csökken függően a 10 Mbps-os Ethernet kapacitásától

- Ablak – gyors felfutás
 - Hatása: többszörös loss
 - Timeout
- Sok timeout – lassú kapcsolat
 - Főleg a kapcsolat elején nagy gond
 - Inkább fast retransmit kellene

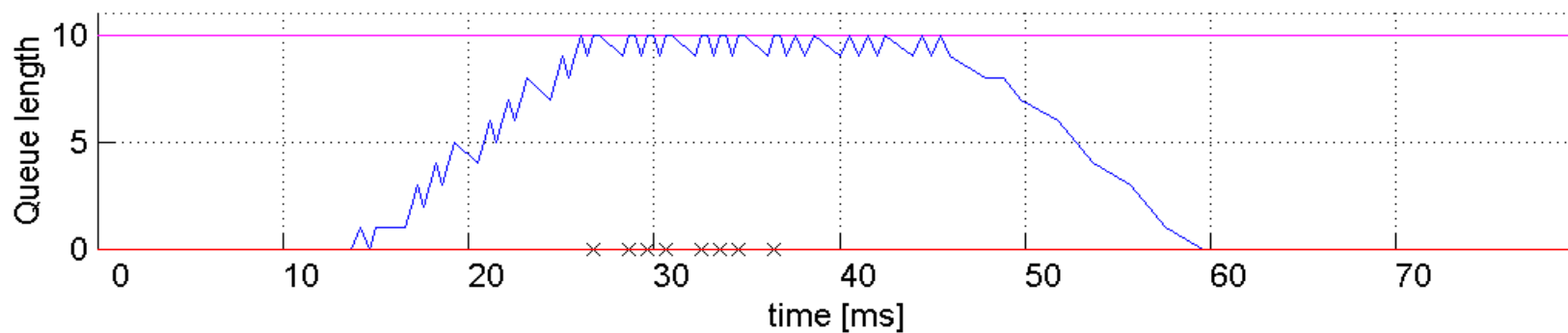
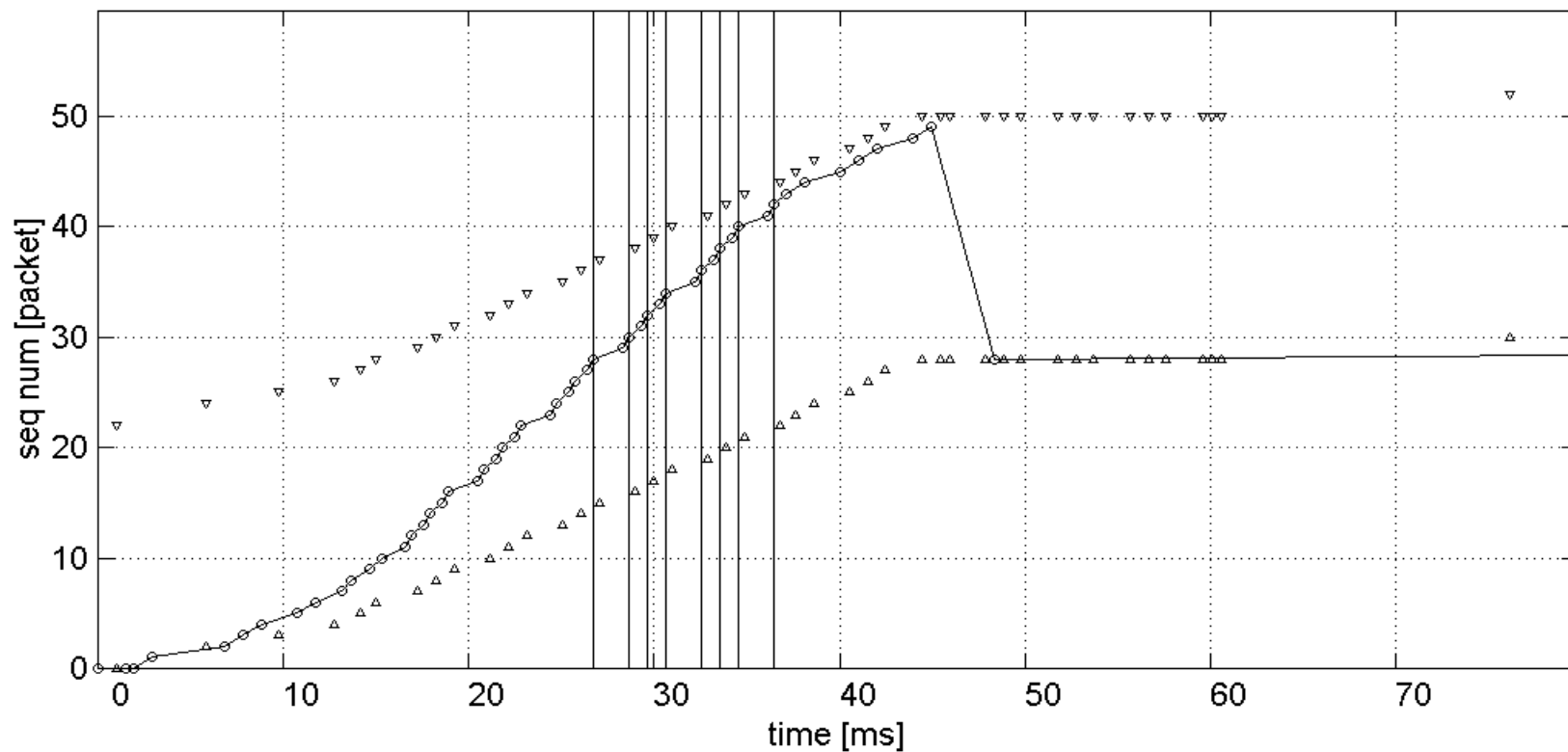


- A slow start börsztös csomagvesztéssel és az adás szüneteltetésével ér véget
- Periódikus veszteségek láthatók, melyeket a küldő gyorsan javít – a veszteségek nem okoznak jelentős teljesítménycsökkenést
- A sorhosszban periódikus minta van 300 ms után: lineáris növekedés, egy veszteség, egy esés. Ez mutatja a congestion avoidance mechanizmus működését a küldő oldalon: a veszteség észrevételekor csökkenti a congestion window értékét, majd ismét lineárisan (additívan) növeli

Az első 80 ms



Trace: 443

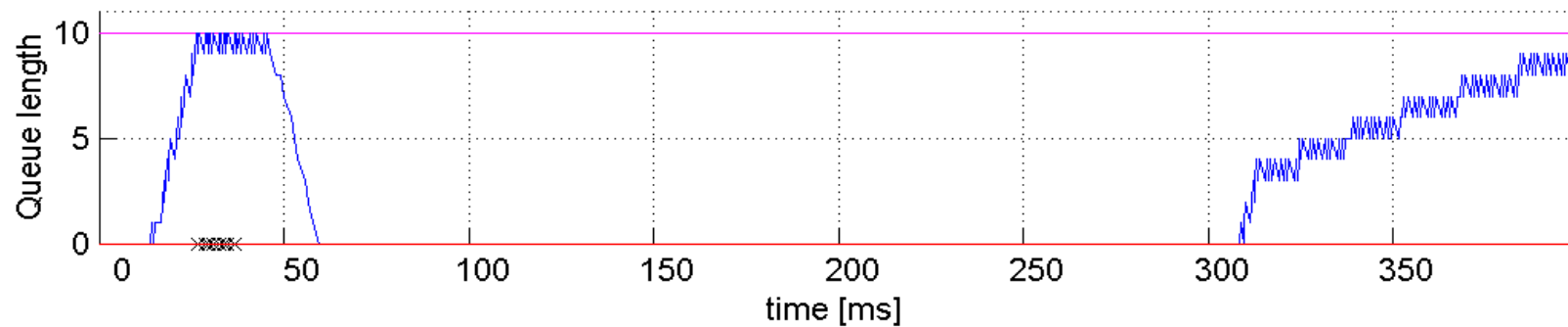
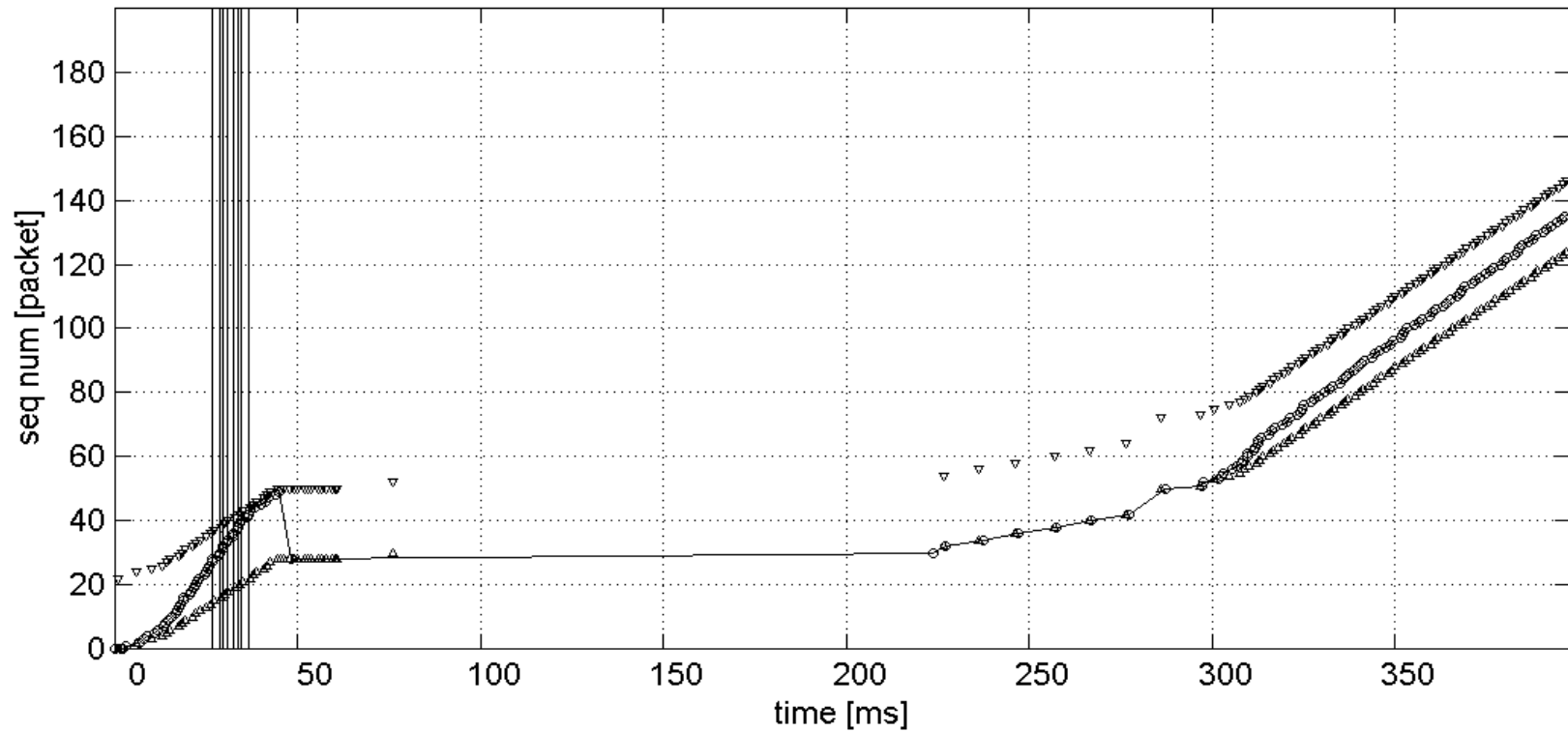


26-36ms	A fogadó felé a sorok túlcsoordulnak, a csomagok eldobásra kerülnek
37-45ms	A fogadó felé menő csomagok továbbításra kerülnek, a fogadó fogadja ezeket a csomagokat, de nem nyugtázza őket, mert néhány korábbi csomag is hiányzik
44-47ms	A fogadó duplikált ACK-t küld
48ms	A küldő újraküldi az első nemnyugtázott csomagot (fast retransmit).
49-61ms	A fogadó továbbra is duplikált ACK-kat küld
76ms	A fogadó nyugtázza az újraküldött csomagokat

200 ms környéke



Trace: 443



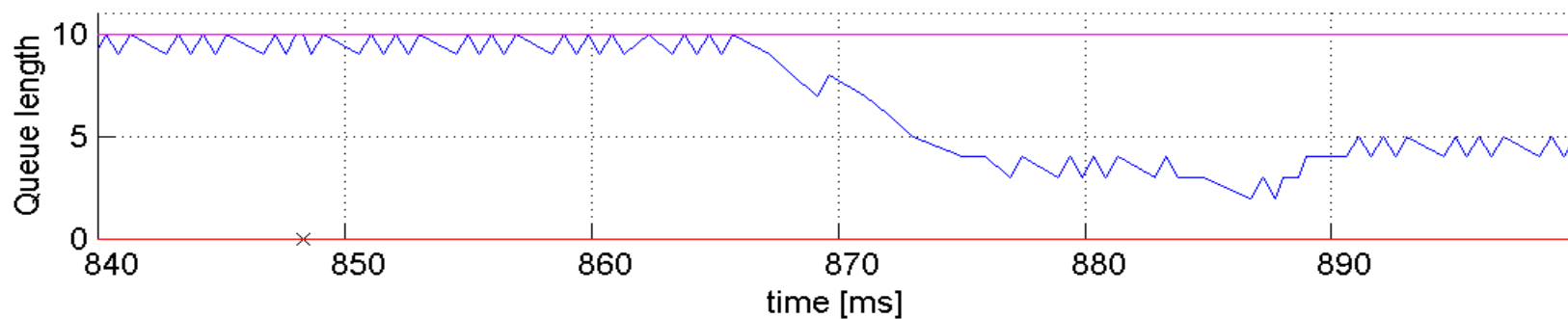
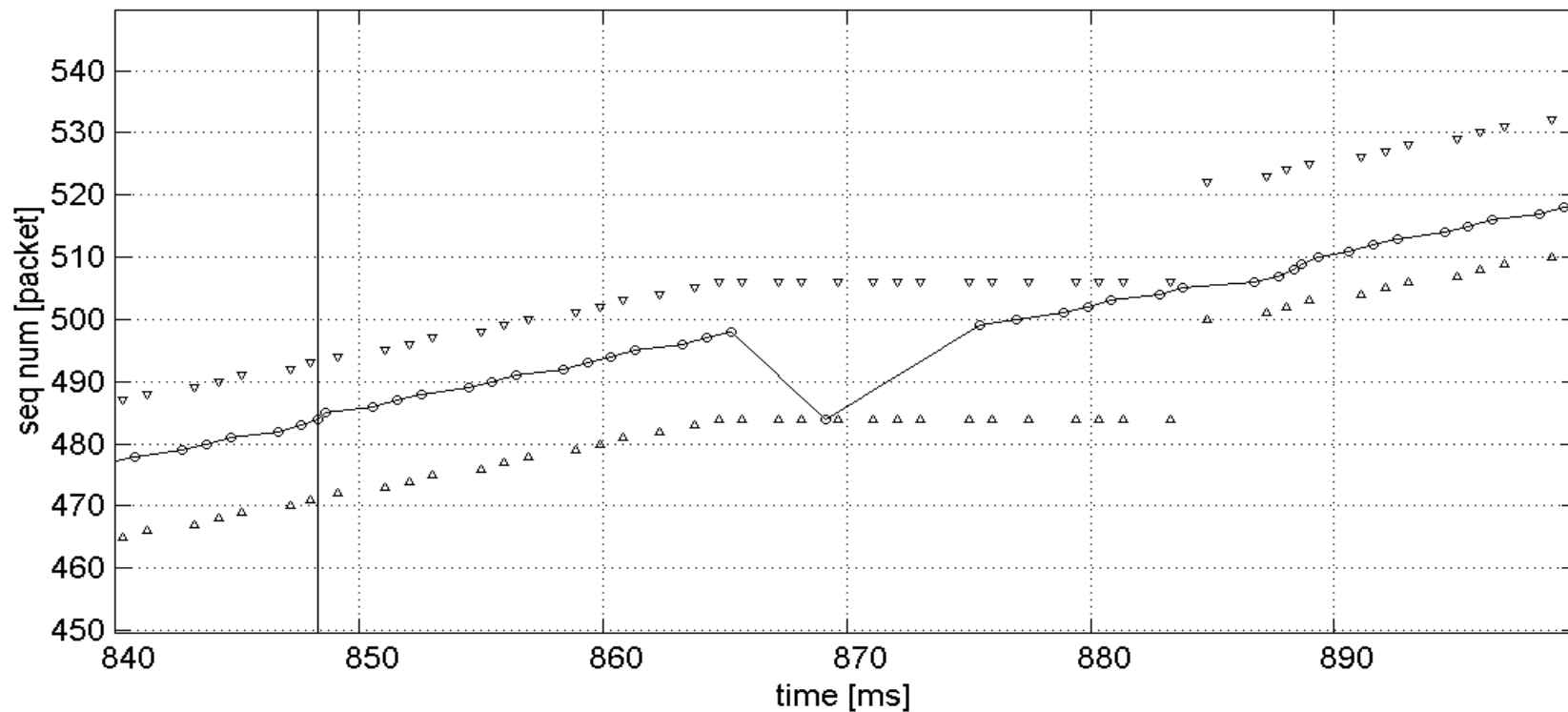
225ms	A küldő időzítője lejár és küld egy következő nemnyugtázott csomagot. Az időztés kb 200 ms
225-275ms	A küldő továbbít egy csomagot rögtön, amint a fogadó nyugtázta az előzőt. A küldő nem növeli a congestion window méretét, amíg újraküldést végez
275ms	A küldő minden elküldött adatára kap nyugtát (Az ACK számok ugrása). Slow start kezdődik.
330ms-	A gyors (exponenciális) növekedése a nemnyugtázott csomagoknak megáll és congestion avoidance szakasz következik – ez látszik a sorhossz lineáris növekedéséből

- Börsztös csomagvesztés történt
- Az első veszteszt a küldő vette észre a duplikált ACK-ból, és gyors újraküldés következett
- A következő veszteséket szintén a küldő vette észre az időzítők lejáratakor – emiatt slow start indult
- Ez a mechanizmus börsztös veszteséknél gyakori

870 ms környéke



Trace: 443



848ms	Egy csomag veszett el a közbenső sorok megtelése miatt
848-864ms	A fogadó a korábbi csomagokat nyugtázza
864-868ms	A fogadó a további csomagokat nem tudja nyugtázni, mert egy hiányzik. Emiatt a fogadó duplikált ACK-t küld
869ms	A küldő 3 duplikált ACK-t kap, majd gyors újraküldést hajt végre
875-884ms	A küldő folytatja a csomagok küldését, mivel nem érte el a fogadó advertised window méretét még.
885ms	Egy ugrás látható az ACK numberek között – a fogadó megkapta a hiányzó csomagot, és az azutániakkal együtt nyugtázta azt.
885ms-	A küldő folytatja a csomagok továbbítását (fast recovery történt: slow start nem következik most). A gyors újraküldés nem okozott nagy teljesítménycsökkenést

Csomagvesztés – jó vagy rossz?



BME-TMIT

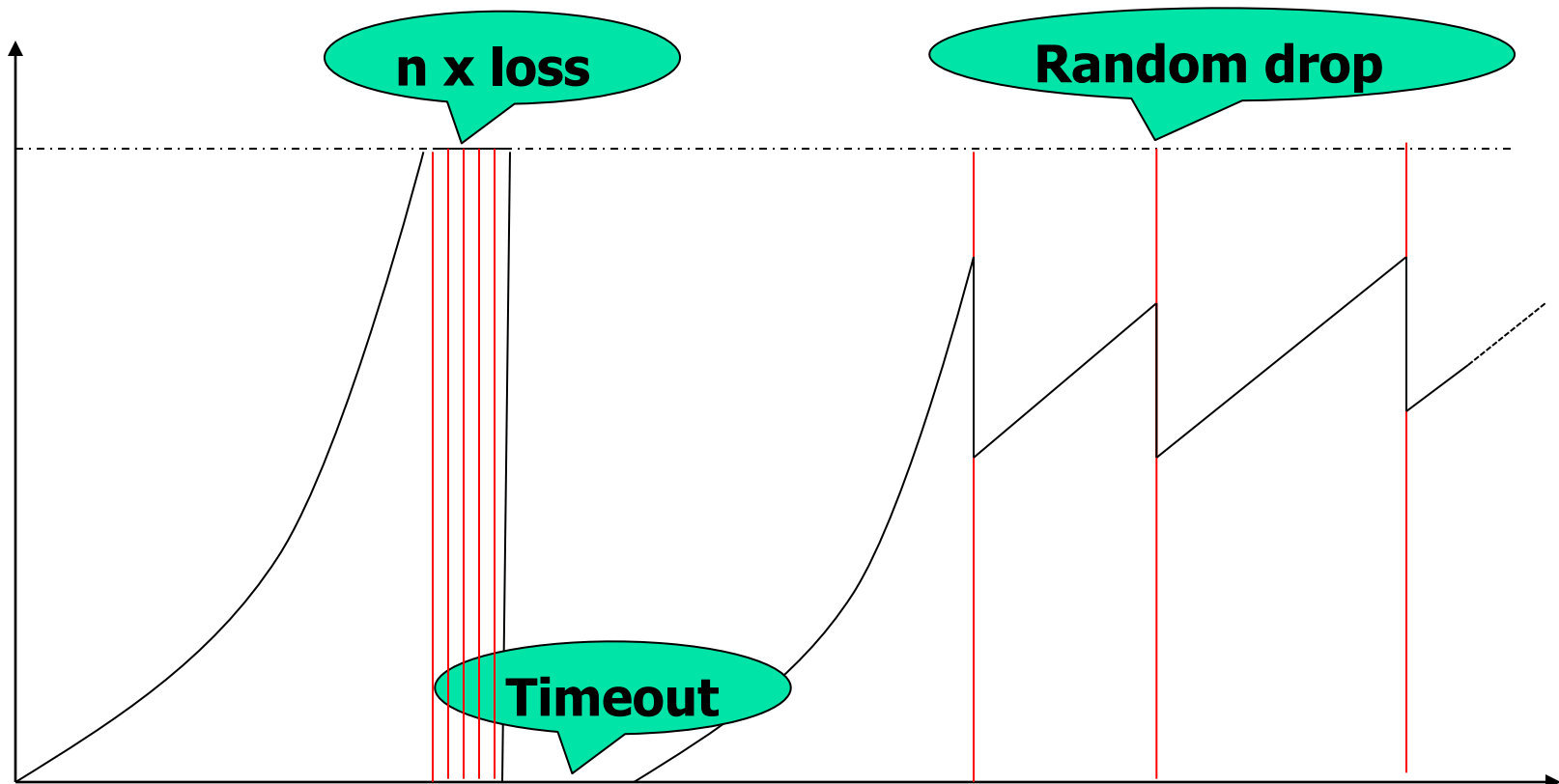
- Példa: DSL 10Mbit/s vonal, 2 lehetőség
 - a) $x\%$ loss vs
 - b) FEC – hibajavító kód, fix 20% adat
- TCP transport, letöltések: melyik jobb?
- a) 10Mbps – $x\%$ loss (pl. $\text{BER}=10^{-6}$)
 - ⇒ 1 csomag 1500 byte, 12000bit, minden 83.3 csomag elvész
- b) 8Mbps
 - a hibajavító kód 20% veszteség mindig

RED – Random Early Drop



BME-TMIT

- Buffer menedzsment a routerekben
 - Egy dobás jobb mint egy timeout



- Probléma:
 - Wireless – eleve van loss
 - de az nem túlcsordulás, nem szűk keresztmetszet!
 - TCP rosszul értelmezi, visszaveszi a cwnd-t
- Megoldások - többféle
 - WTCP – proxy
 - SACK

Köszönöm a figyelmet

- Vége -



Budapest University of Technology and Economics

**Department of
Telecommunications and Media Informatics**

