

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?

Forgalom szétosztása



BME-TMIT

Szerver alkalmazások
– több klienssel
kommunikálnak

Egyedi port
minden

Osztoznak az
alsóbb hálózati
rétegeken

Application

Transport

Network

P1

P2

P3

P4

Végpontok azonosítása $\langle src_ip, src_port, dest_ip, dest_port \rangle$

- Netstat parancs

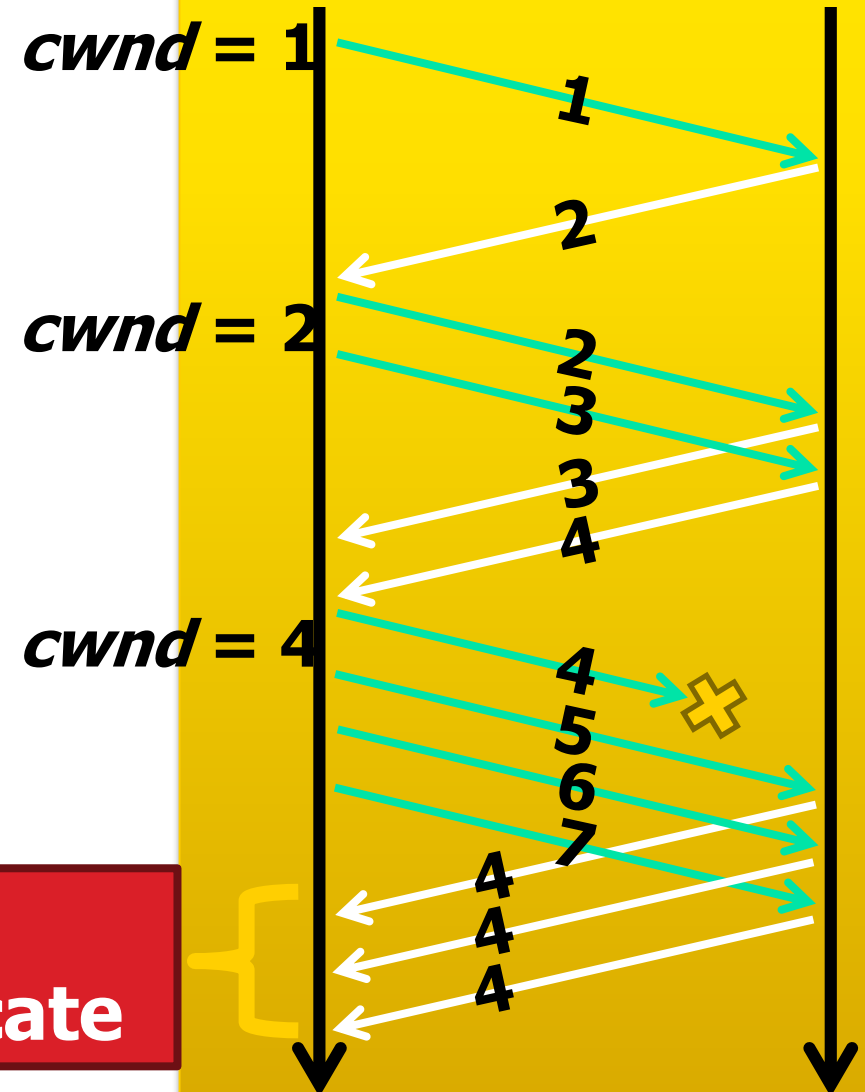
TCP	127.0.0.1:1906	localhost:1907	ESTABLISHED
TCP	192.168.1.147:53699	13.77.87.52:https	ESTABLISHED
TCP	192.168.1.147:53703	91.190.216.57:12350	ESTABLISHED
TCP	192.168.1.147:53737	64.4.23.152:40008	ESTABLISHED
TCP	192.168.1.147:53759	108.177.96.188:5228	ESTABLISHED
TCP	192.168.1.147:53772	40.77.226.192:https	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

- 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

- Tahoe: csomagvesztés után hosszú timeout
- Reno: újraküldés egy szegmens 3-szoros ACK után
- Ez a Fast Retransmit



TCP Reno: Fast Recovery



BME-TMIT

9

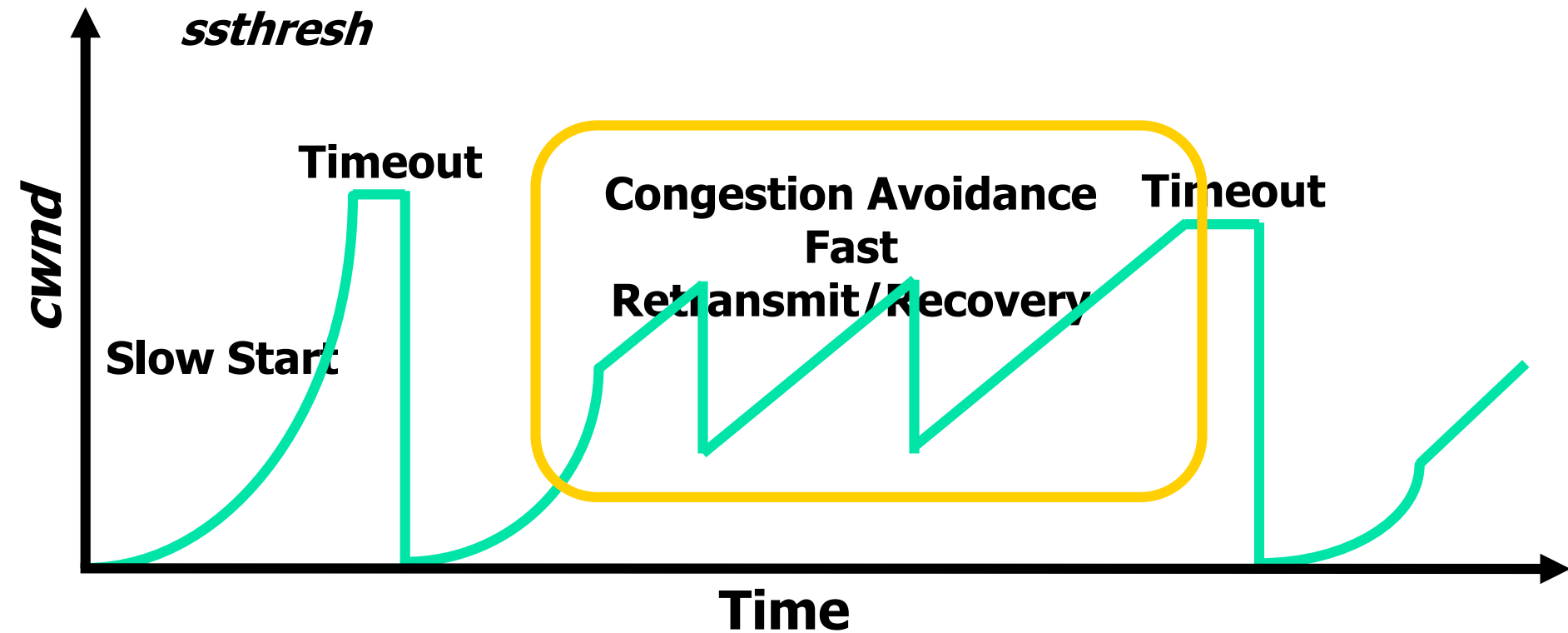
- Fast-retransmit után *cwnd*-t állítja *ssthresh/2-re*
 - Ne kezdje a *cwnd* 1-től
 - Fölösleges a slow start
 - Kiváltja a hosszú timeoutot
- De RTO timeout után továbbra is *cwnd* = 1
 - TRO után slow start, mint a Tahoe-nál
 - Azt jelzi hogy a csomagok nem jutnak el
 - i.e. komoly szűk keresztmetszet alakult ki

Fast Retransmit és Fast Recovery



BME-TMIT

10



- A *cwnd* oszcillál az optimális ablak körül
- TCP mindig erőlteti a csomageldobást

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. $BER=10^{-6}$)
 - ⇒ 1 csomag 1500 byte, 12000bit, minden 83.3 csomag elvész
- b) 8Mbps
 - a hibajavító kód 20% veszteség mindig

- Tahoe: az eredeti
 - Slow start és AIMD
 - Dynamic RTO egy RTT becsléssel
- Reno: fast retransmit és fast recovery
- NewReno: Javított fast retransmit
 - Minden duplicate ACK újraküldés
 - Probléma: >3 nem sorrendben érkezett csomag sok fölösleges újraküldés
- Vegas: delay-alapú vezérlés
- És még sok-sok változat...

- Manapság?
 - Nagy bandwidth-delay a jellemző, a TCP nem annyira szereti
 - Compound TCP (Windows)
 - Reno alapú
 - Két congestion window: delay és loss alapú
 - Ezért a *compound* vezérlés
 - TCP CUBIC (Linux)
 - BIC (Binary Increase Congestion Control)
 - Az ablakot egy cubic function vezérli

- Programozónak
- Hálózati operátornak
- Miért teszünk különbséget?
 - Programozóként nem érdekel a hálózat
 - Operátorként nem érdekel az alkalmazás

Programozóként...



BME-TMIT

1. Forgalom a kapcsolat kiépítésekor
 - A three-way handshake, blokkolás
2. Az adat blokkokban érkezik
 - Hacsak push/urgent biteket nem használunk (ritka)
3. TCP throughput függ az applikációtól is
 - Néha ez jó

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

TCP opciók - példa



BME-TMIT

No.	Time	Source	Destination	Protocol	Length	Info
4	0.168986	192.168.0.11	239.255.255.250	SSDP	175	M-SEARCH * HTTP/1.1
5	0.221892	fe80::d0f9:8c1:d62f:eb63	ff02::1:3	LLMNR	86	Standard query 0x7e01 A isatap
6	0.000117	192.168.0.11	224.0.0.252	LLMNR	66	Standard query 0x7e01 A isatap

▶	Frame 12: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
▶	Ethernet II, Src: Wistron_2d:ab:ba (00:1f:16:2d:ab:ba), Dst: 3Com_03:04:05 (00:01:02:03:04:05)
▶	Internet Protocol Version 4, Src: 192.168.0.11, Dst: 192.168.0.168
▲	Transmission Control Protocol, Src Port: 29385, Dst Port: 22, Seq: 0, Len: 0
	Source Port: 29385
	Destination Port: 22
	[Stream index: 0]
	[TCP Segment Len: 0]
	Sequence number: 0 (relative sequence number)
	[Next sequence number: 0 (relative sequence number)]
	Acknowledgment number: 0
	1000 = Header Length: 32 bytes (8)
▶	Flags: 0x002 (SYN)
	Window size value: 8192
	[Calculated window size: 8192]
	Checksum: 0x822a [unverified]
	[Checksum Status: Unverified]
	Urgent pointer: 0
▲	Options: (12 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Operation (NOP), No-Operation (NOP), SACK permitted
	▶ TCP Option - Maximum segment size: 1460 bytes
	▶ TCP Option - No-Operation (NOP)
	▶ TCP Option - Window scale: 2 (multiply by 4)
	▶ TCP Option - No-Operation (NOP)
	▶ TCP Option - No-Operation (NOP)
	▶ TCP Option - SACK permitted
▲	[Timestamps]
	[Time since first frame in this TCP stream: 0.000000000 seconds]
	[Time since previous frame in this TCP stream: 0.000000000 seconds]

4. A hálózat korlátozhatja a sávszélességet
 - Akkor gond ha nincs elég hálózati kapacitás
5. Socket opciók
 - Algoritmusok/paraméterek állítása
6. Portok újrahasználása
 - Fin bit – zárás után még nem fejeződik be a kapcsolat

1. A forgalom jellege

- Börsztös, vezérlés

2. Bottleneck detektálása

- A forgalom nem nő $\sim 80\%$ fölé (aggregált)

3. Congestion control algoritmusok a routerben

- Előre kezelik a szűk keresztmetszetet
- Fairness elérése

4. Lossy csatornák - rádió

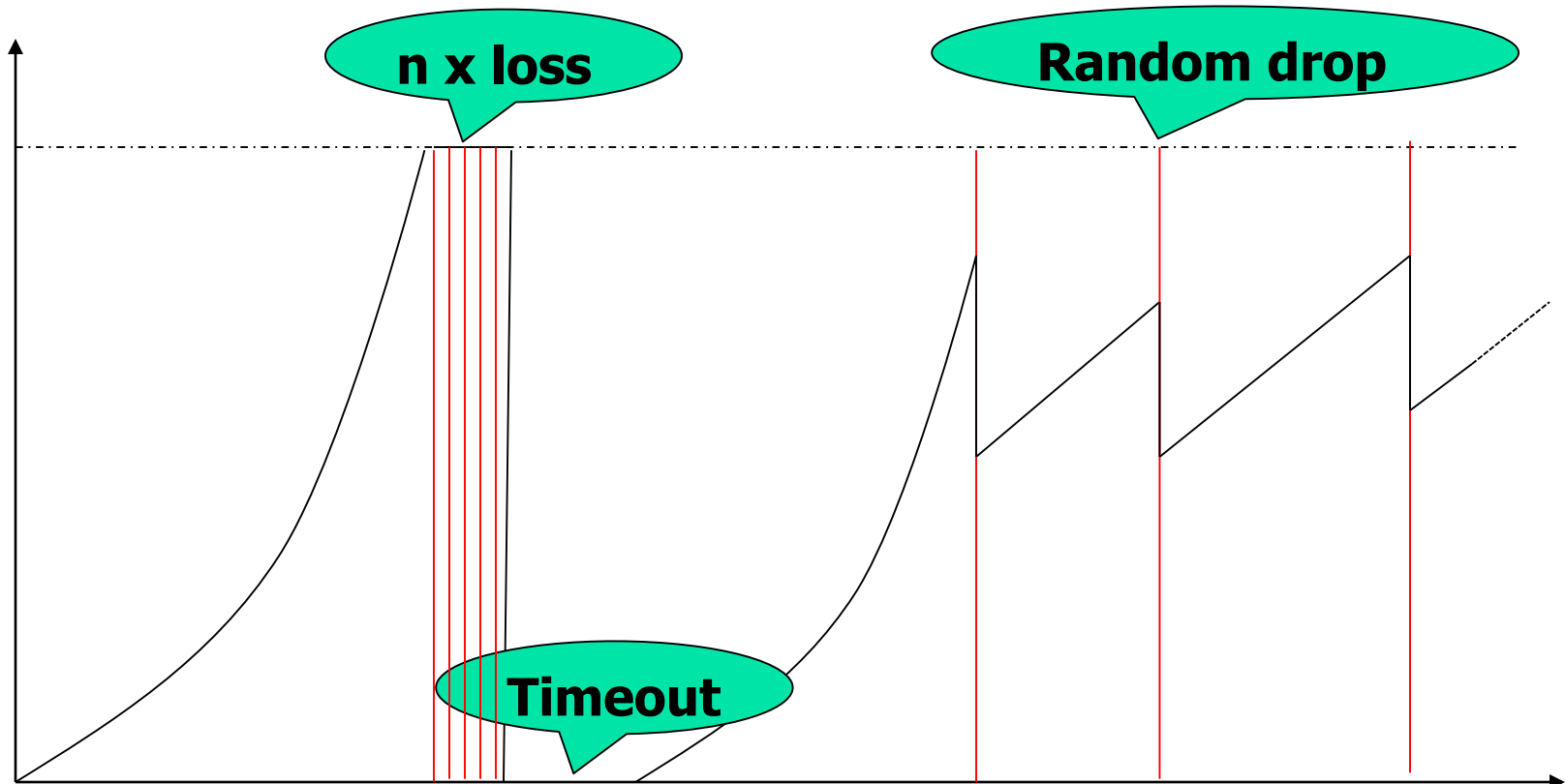
- A csomagvesztést félreértelmezi

RED – Random Early Drop



BME-TMIT

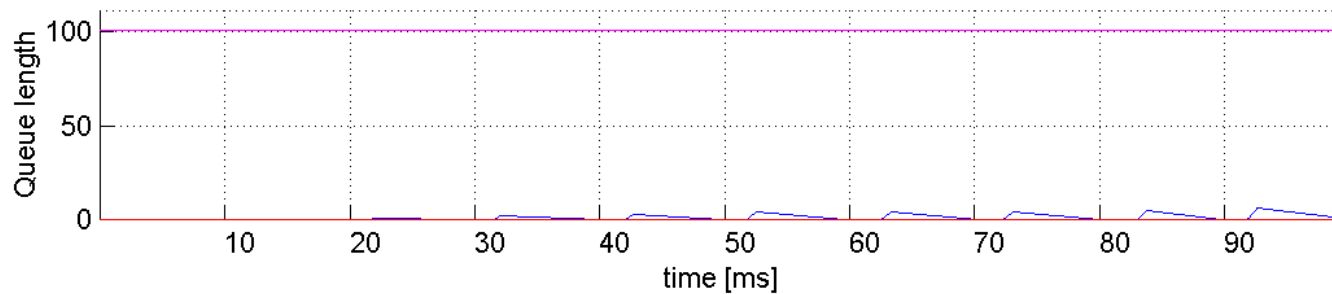
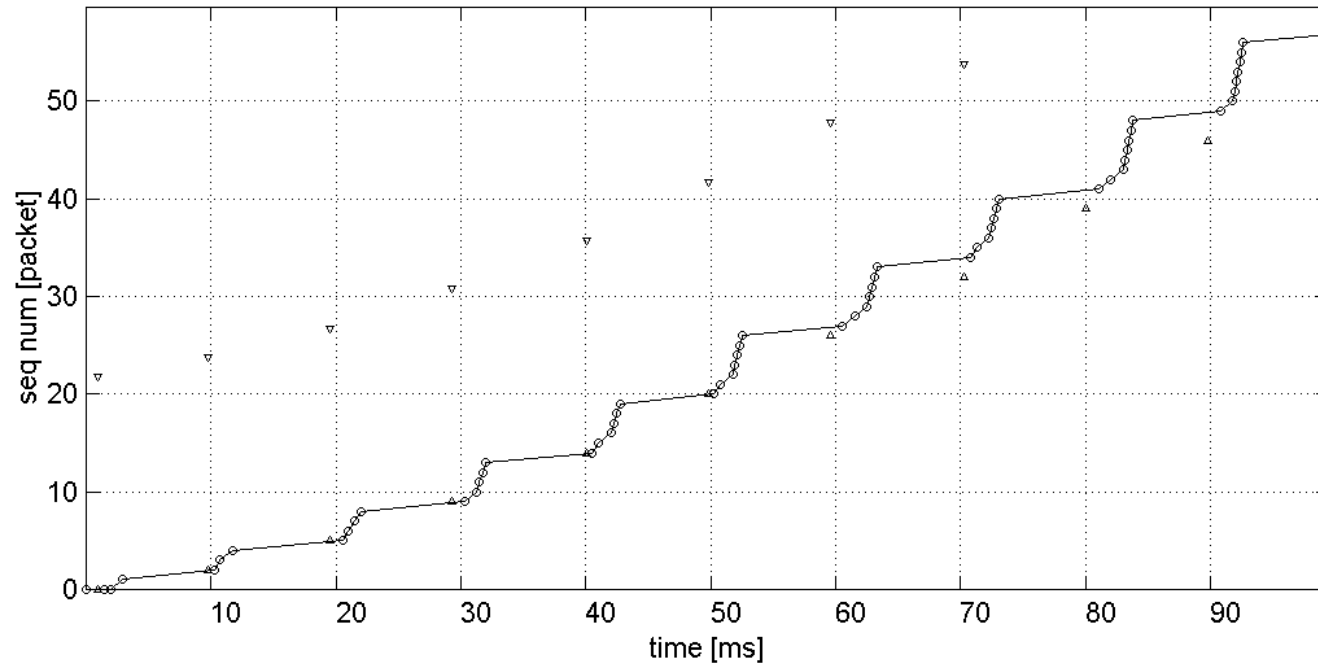
- Buffer menedzsment routerben
 - Egyedi dobás jobb mint a timeout



- Hibás működés:
 - Wireless – rádiós dobás
 - Nem szűk keresztmetszet!
 - TCP félreértelmezi, csökkenti a cwnd-t
- Megoldás
 - L2 újraküldések
 - WTCP – proxy
 - SACK – selective acknowledgements

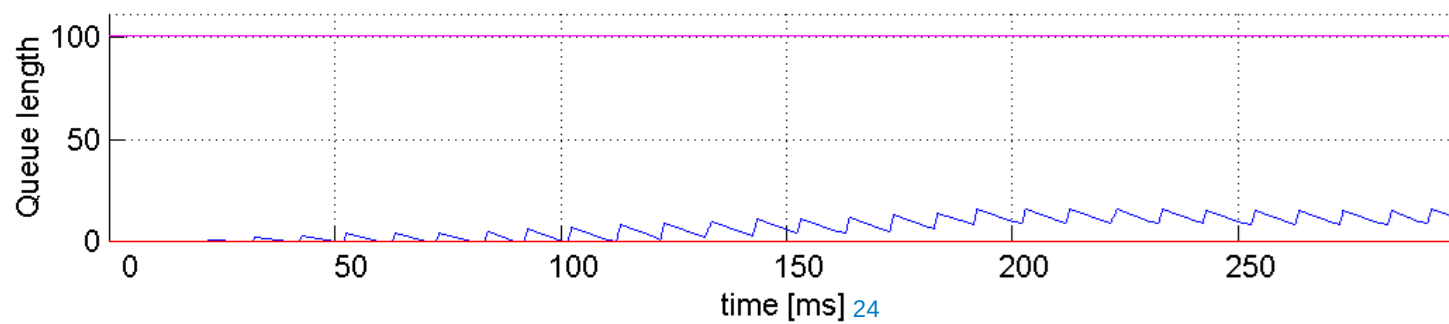
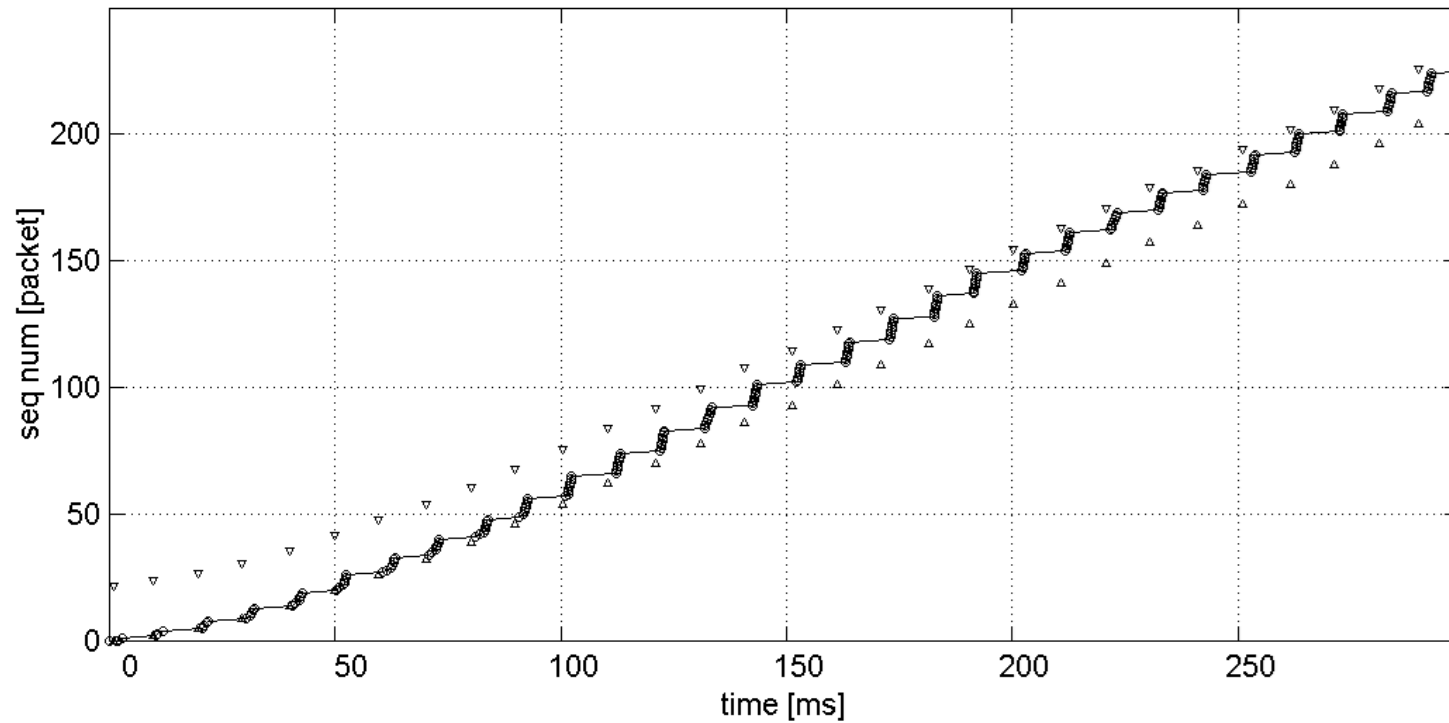
No.	Algo.	Sender	Traffic	# conn.	Receiver	Kbytes/sec	# ovfl.
413	None	Linux 2.1	one.1024	1	Linux 2.0	1097.7 (gw)	0

Trace: 413



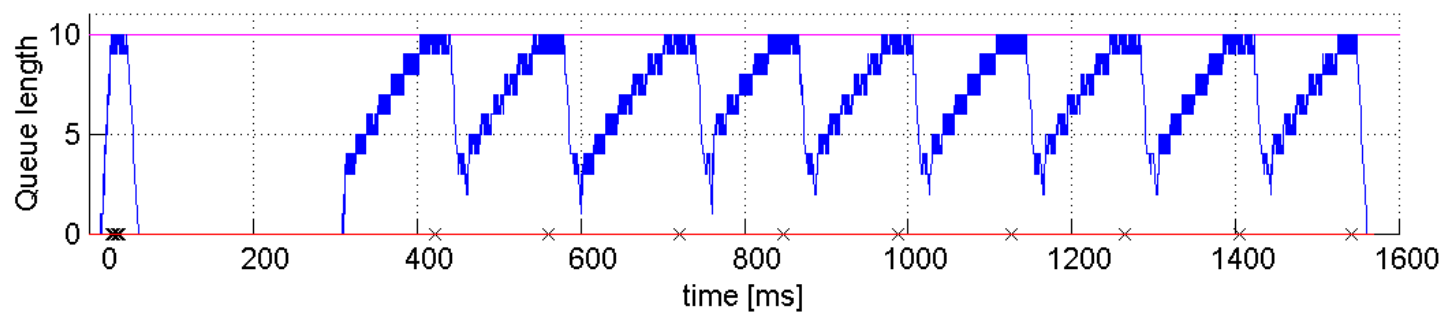
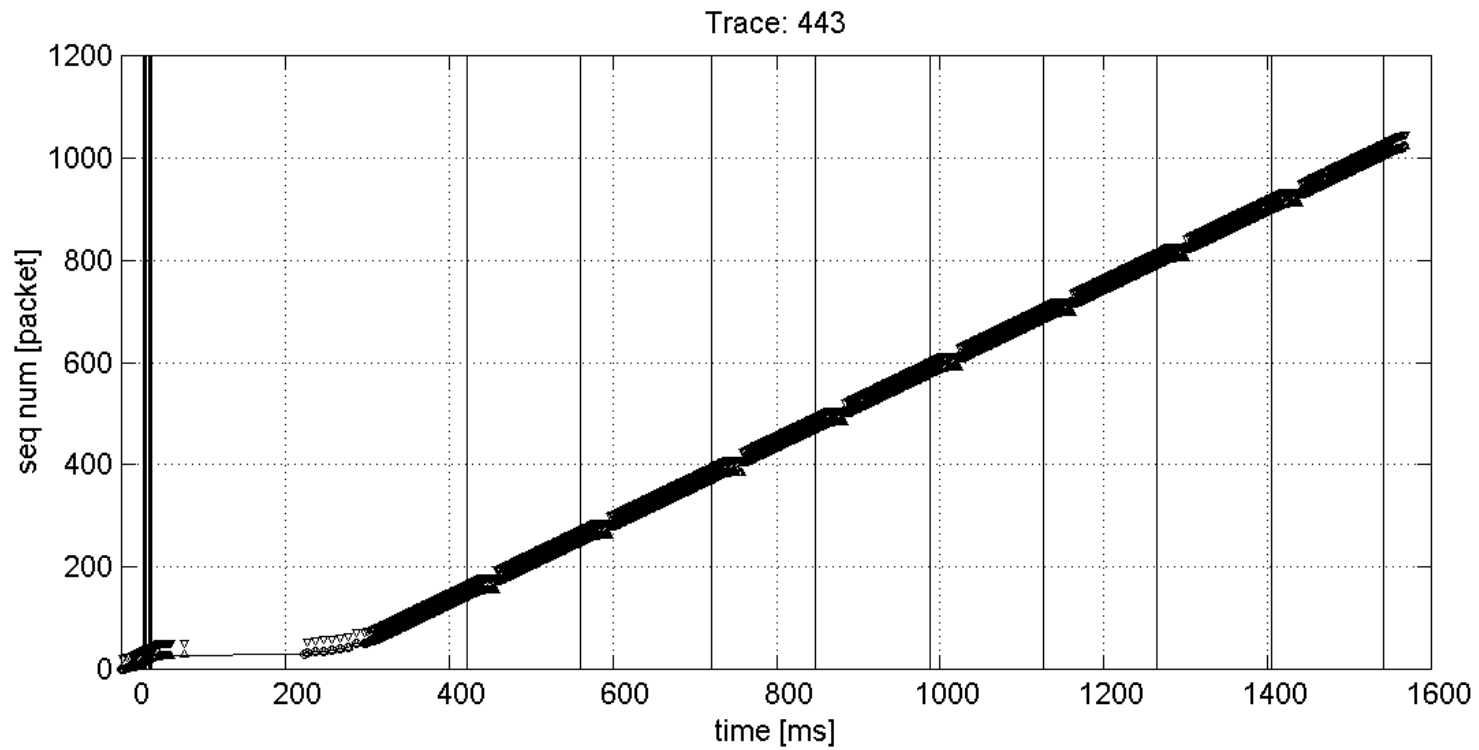
- A fogadó csak egy ACK-t küld egy csomagcsoportra
- Minden ACK érkezése a küldőnél egy csomagbörszt küldését eredményezi
- A küldő a *cwnd*-t minden egyes ACK érkezésekor eggyel növeli
- A sorok hossza növekszik a börsztök érkezésekor, a börsztök méretei pedig növekednek az idővel
- ACK érkezések 10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

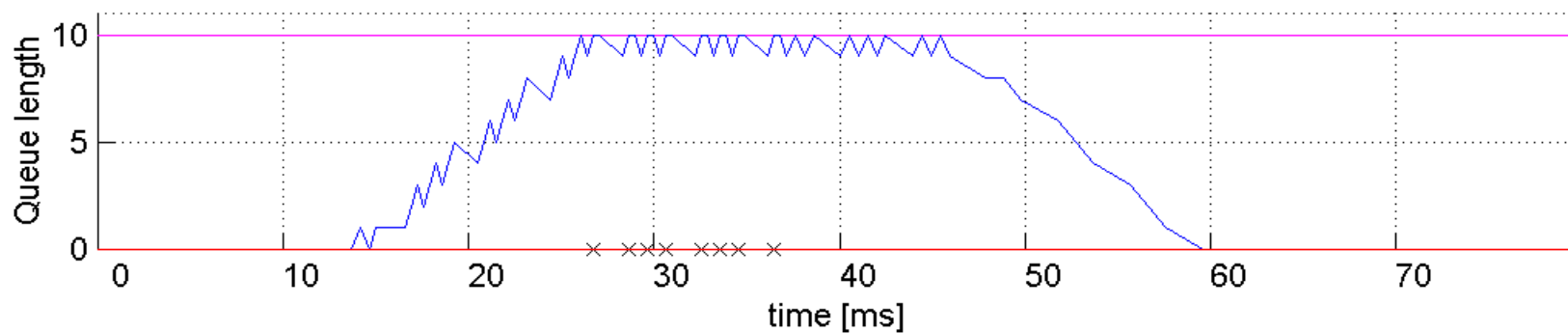
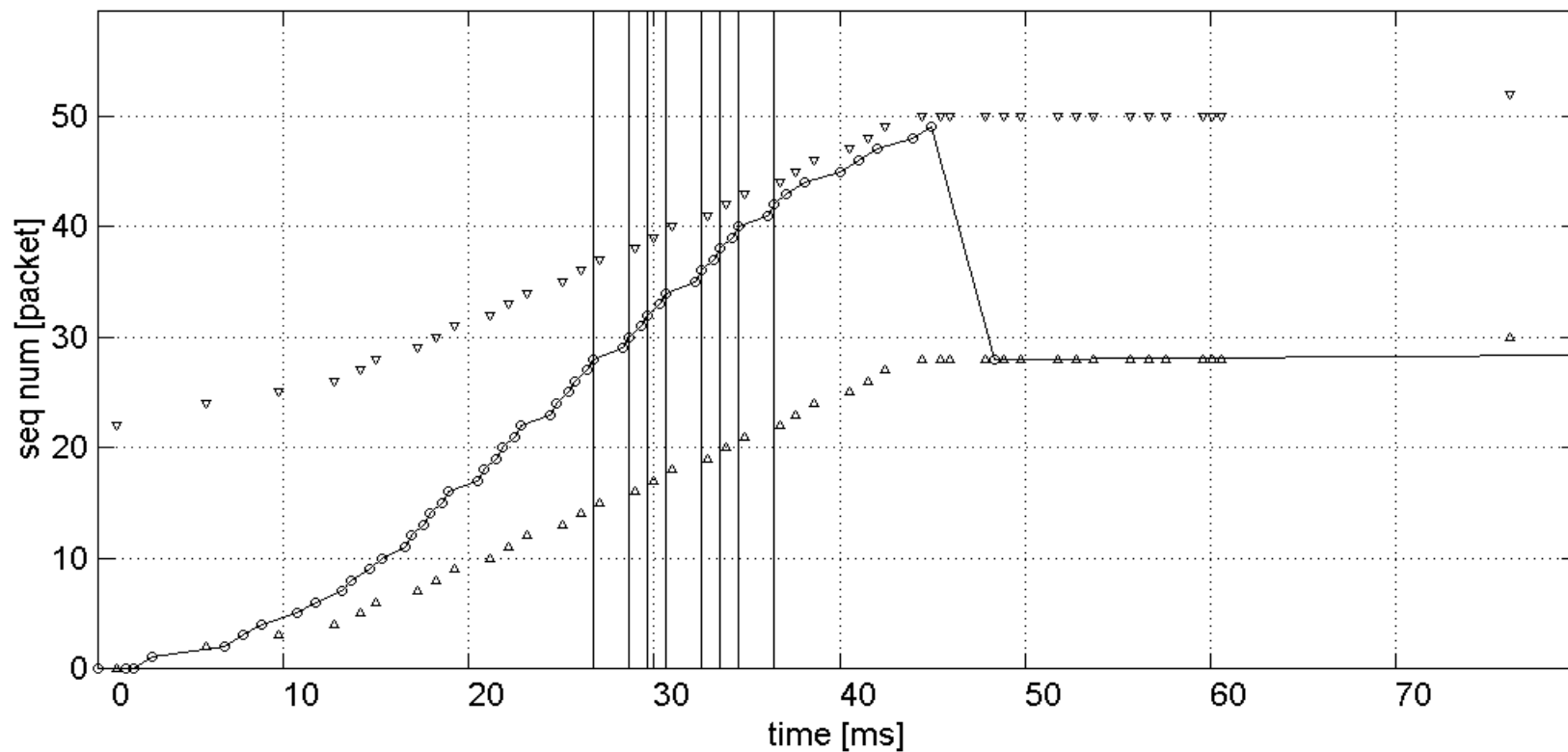


- A slow start börsztös csomagvesztéssel és az adás szüneteltetésével ér véget
- Periódikus veszteségek láthatók, melyeket a küldő gyorsan javít – a veszteségek nem okoznak jelentős teljesítménycsökkenést
- A sorhosszban periódikus minta van 300 ms után: lineáris növekedés, egy veszteség, egy esés. Ez 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

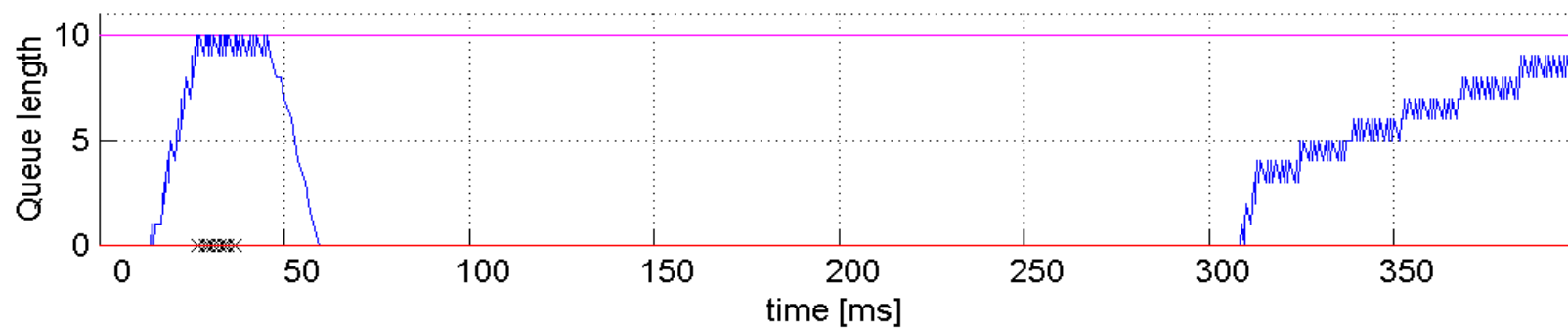
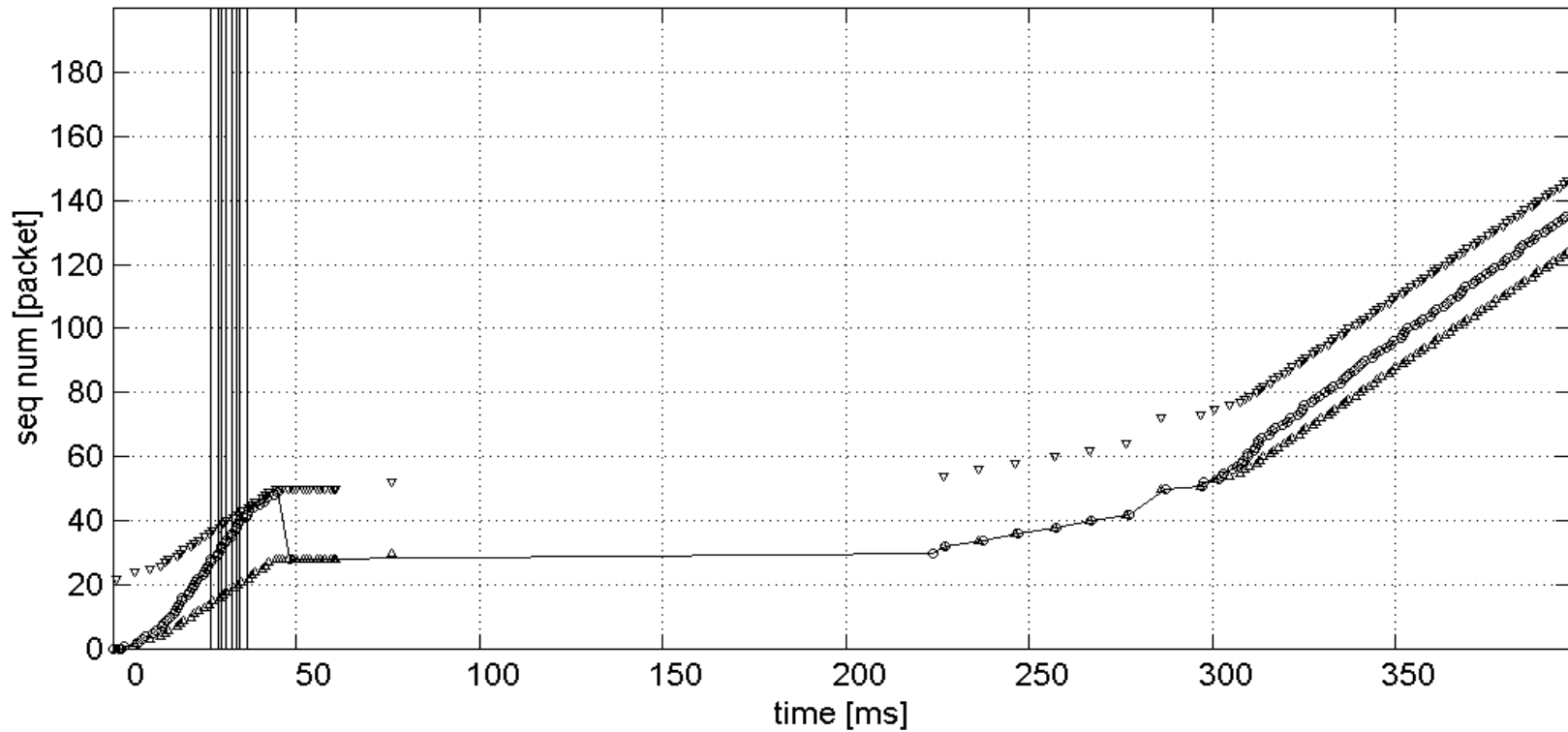


26-36ms	A fogadó felé a sorok túlcsordulnak, a csomagok eldobásra kerülnek
37-45ms	A fogadó felé menő csomagok továbbításra kerülnek, a fogadó fogadja ezeket a csomagokat, de nem nyugtázza őket, mert néhány korábbi csomag is hiányzik
44-47ms	A fogadó duplikált ACK-t küld
48ms	A küldő újraküldi az első 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



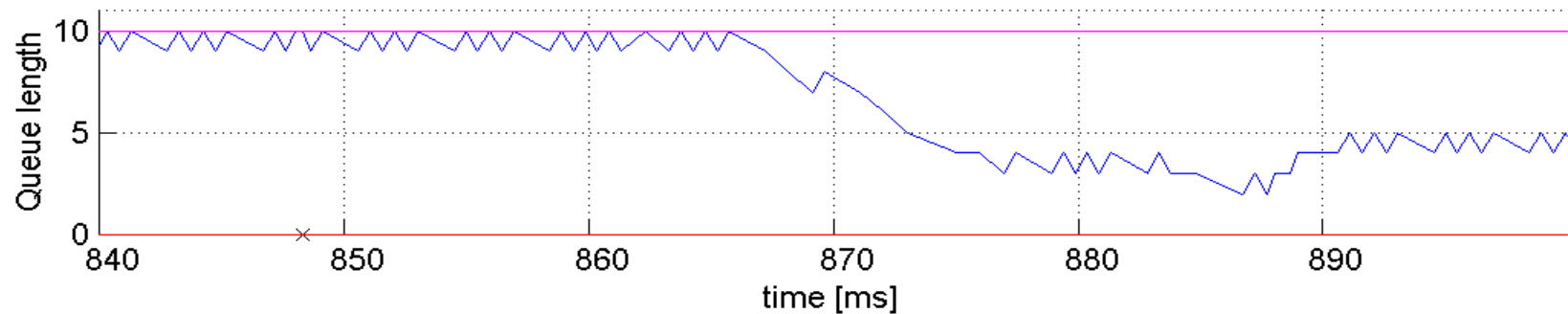
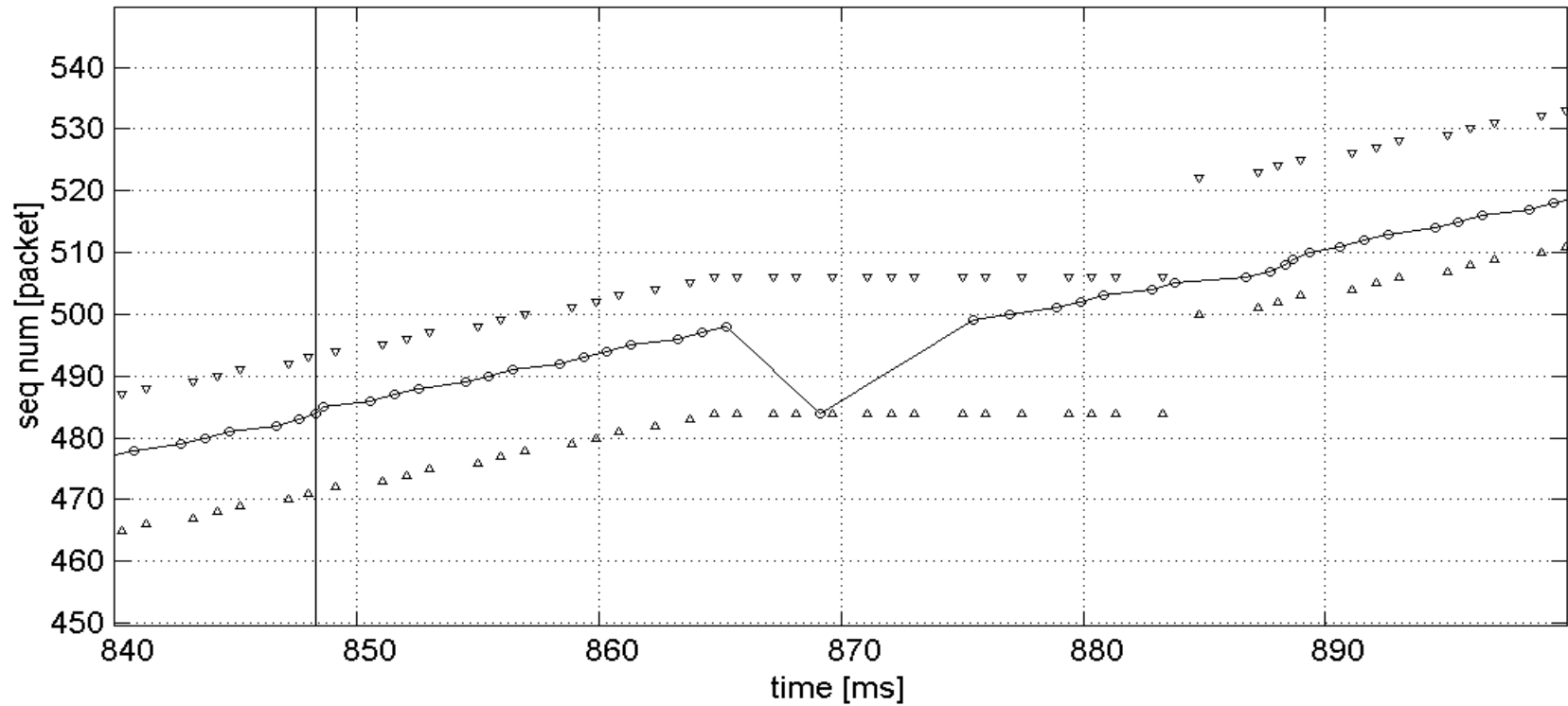
225ms	A küldő időzítője lejár és küld egy következő nemnyugtázott csomagot. Az időztés kb 200 ms
225-275ms	A küldő továbbít egy csomagot rögtön, amint a fogadó nyugtázta az előzőt. A küldő nem növeli a congestion window méretét, amíg újraküldést végez
275ms	A küldő minden elküldött adatára kap nyugtát (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

Köszönöm a figyelmet

- Vége -

