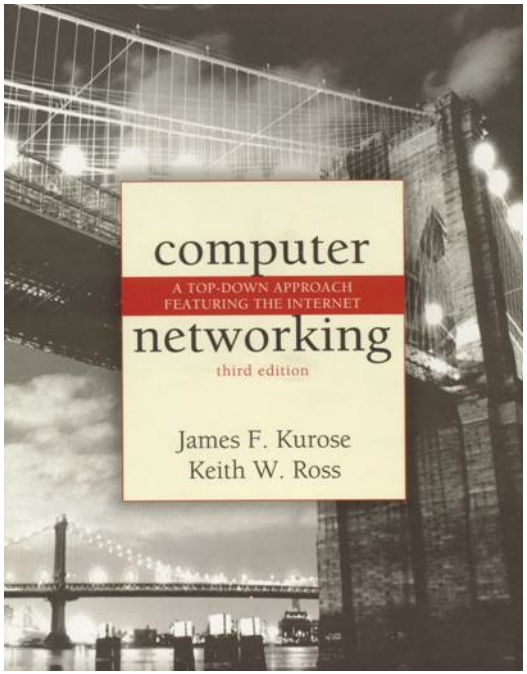
Taşıma Katmanı (Transport Layer)



Computer Networking: A Top Down Approach Featuring the Internet, 3rd edition.
Jim Kurose, Keith Ross
Addison-Wesley, July 2004.

A note on the use of these ppt slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) in substantially unaltered form, that you mention their source (after all, we'd like people to use our book!)
- If you post any slides in substantially unaltered form on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

Taşıma Katmanı

Hedefler:

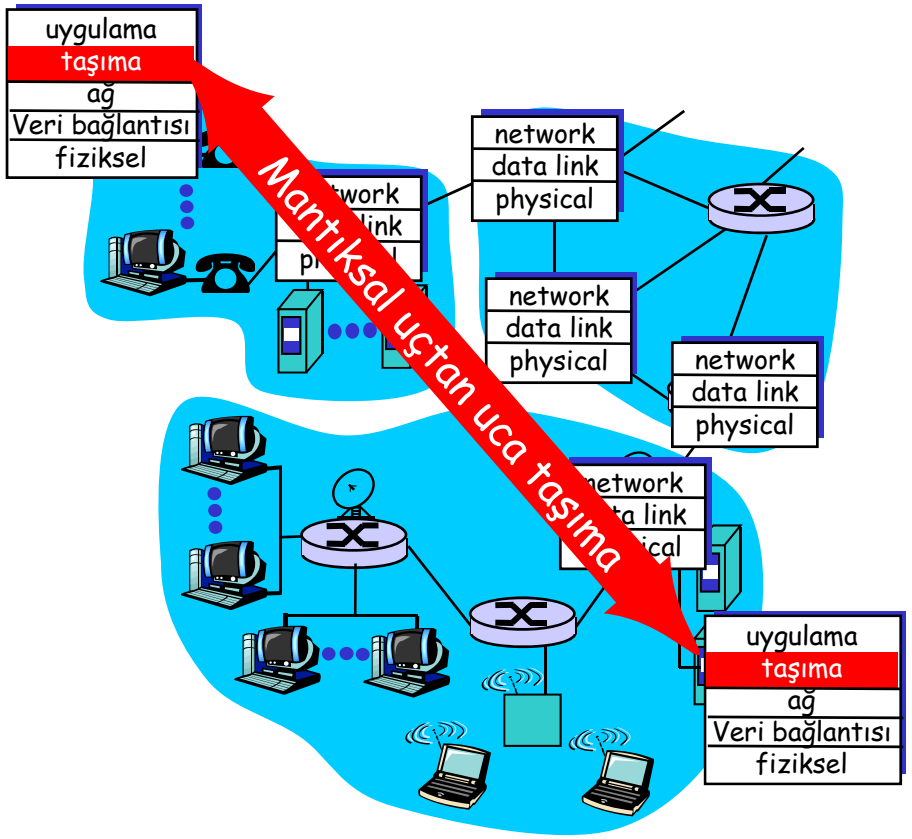
- Taşıma katmanı servislerinin prensiplerini anlamak:
  - Çoklama/çoklamanın çözülmesi (multiplexing /demultiplexing)
  - Güvenilir veri iletimi (reliable data transfer)
  - Akış kontrolü (flow control)
  - Tıkanıklık kontrolü (congestion control)
- Internet deki taşıma katmanı protokollerini öğrenmek:
  - UDP: bağlantısız taşıma (connectionless transport)
  - TCP: bağlantı yönelimli taşıma (connection-oriented transport)
  - TCP tıkanıklık kontrolü (congestion control)

Taşıma Katmanı

- 3.1 Taşıma katmanı servisleri
- 3.2 Çoklama (multiplexing) ve çoklamanın çözülmesi (demultiplexing)
- 3.3 Bağlantısız taşıma: UDP
- 3.4 Güvenilir veri iletiminin prensipleri
- 3.5 Bağlantı yönelimli taşıma: TCP
  - segment yapısı
  - güvenilir veri iletimi
  - akış kontrolü
  - bağlantı yönetimi
- 3.6 Tıkanıklık kontrolü prensipleri
- 3.7 TCP tıkanıklık kontrolü

Taşıma Servisleri ve Protokolleri

- Farklı ana sistemlerde çalışan uygulama süreçleri arasında *mantıksal bir bağlantı* sunar
- Taşıma katmanı protokolleri ağ yönlendiricilerinde değil uç sistemler de uygulanır
  - gönderici tarafı: gönderici uygulama süreci tarafından aldığı mesajları segmentlere çevirir, ve ağ katmanına geçirir.
  - alıcı tarafı: segmentleri mesaj haline birleştirir ve uygulama katmanına geçirir.
- Uygulamalar için birden fazla taşıma katmanı protokolü mevcuttur
  - Internet: TCP ve UDP



Taşıma ve ağ katmanları

- *ağ katmanı:* ana sistemler arasında mantıksal bağlantı sunar
- *transport layer:* farklı ana sistemler üzerinde çalışan süreçler arasında mantıksal bağlantı sunar
  - Ağ katmanı üzerinde yer alır ve onun sunduğu servislere dayanır

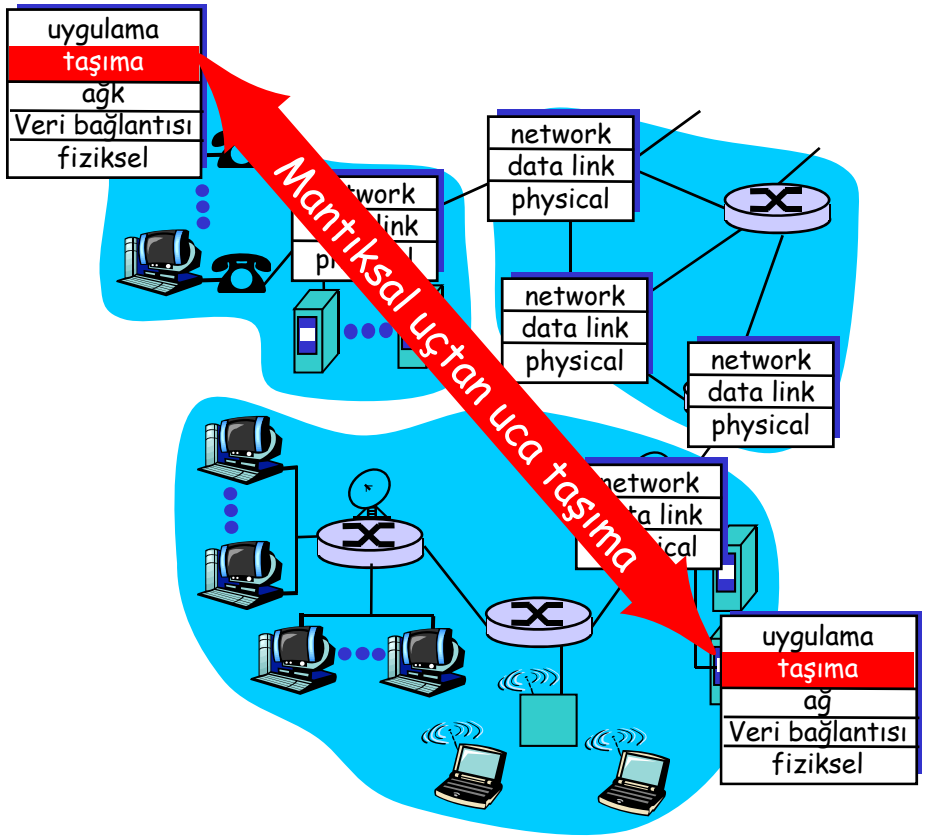
Ev halkı benzeştirmesi:

Birbirine mektup gönderen bir evdeki 12 çocuk ile diğerindeki 12 çocuk

- süreçler = çocuklar
- Uygulama mesajları = zarf içindeki mektuplar
- Ana sistemler = evler
- Taşıma katmanı protokolü = Ann ve Bill
- Ağ katmanı protokolü = posta servisi

Internet'de taşıma katmanı protokolleri

- güvenilir, sıralı teslim (TCP)
  - Tıkanıklık kontrolü (congestion control)
  - Akış kontrolü (flow control)
  - Bağlantı kurulumu (connection setup)
- güvenilmez, sırasız teslim: UDP
  - IP'nin en iyi çabayla teslim servisi üzerinde bir eklenti yoktur
- uygun olmayan servisler:
  - gecikme garantisi
  - Bant genişliği garantisi



Taşıma Katmanı

- 3.1 Taşıma katmanı servisleri
- 3.2 Çoklama (multiplexing) ve çoklamanın çözülmesi (demultiplexing)
- 3.3 Bağlantısız taşıma: UDP
- 3.4 Güvenilir veri iletiminin prensipleri
- 3.5 Bağlantı yönelimli taşıma: TCP
  - segment yapısı
  - güvenilir veri iletimi
  - akış kontrolü
  - bağlantı yönetimi
- 3.6 Tıkanıklık kontrolü prensipleri
- 3.7 TCP tıkanıklık kontrolü

Çoklama / çoklamanın çözülmesi (Multiplexing/demultiplexing)

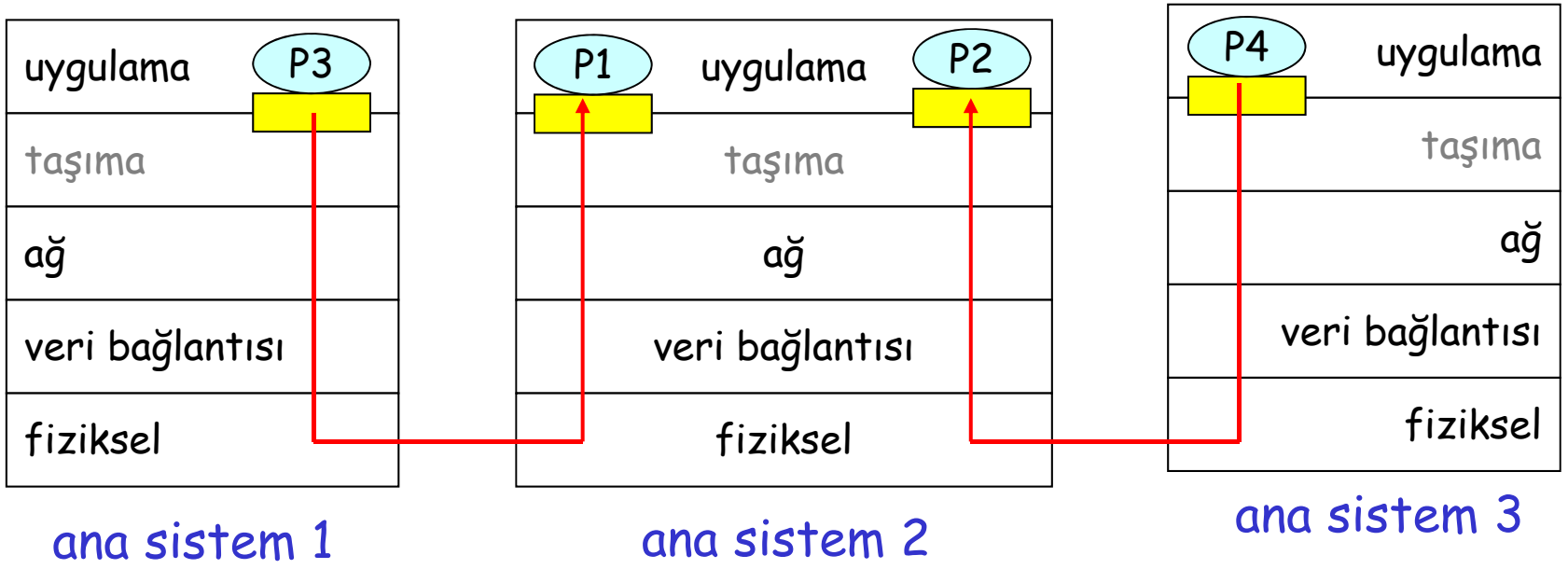
Alıcı ana sistemde çoklamanın çözülmesi

Alınan segmentlerin doğru soketlere teslimi

= soket = süreç

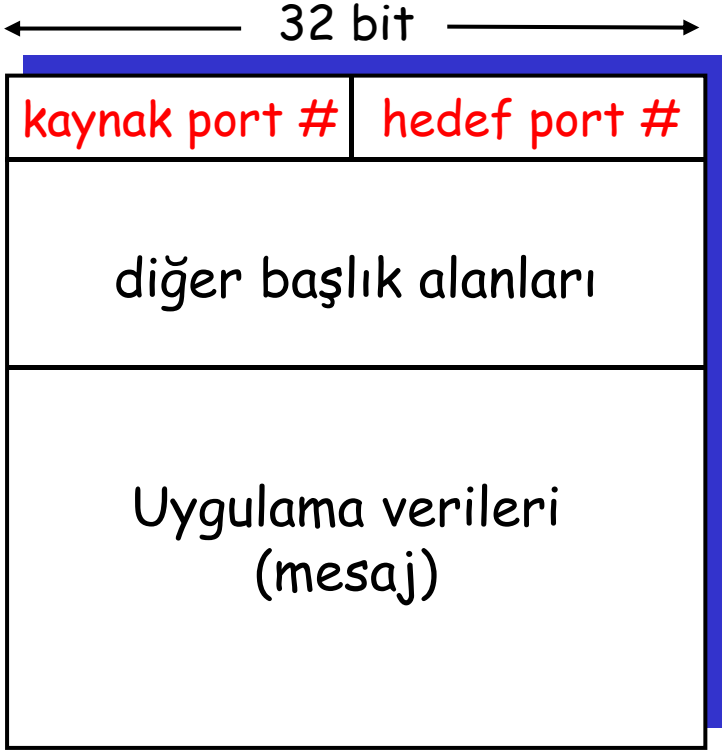
Gönderici ana sistemde çoklama:

Pek çok paketten verinin alınması, verinin başlık ile zarflanması (daha sonra çoklamanın çözülmesinde kullanılmak için



Çoklamanın çözülmesi (demultiplexing) nasıl çalışır?

- ana sistem IP datagramını alır
  - Her datagramın bir kaynak bir de hedef IP adresi vardır
  - Her datagram 1 taşıma katmanı segmenti taşır
  - Her segmentin kaynak ve hedef port numarası vardır (hatırla: bazı çok bilinen uygulamaların port numaraları)
- Ana sistem IP adresi ve port numarasını kullanarak segmenti uygun sokete yönlendirir.



TCP/UDP segment biçimi

Bağlantısız çoklamanın çözülmesi

- Port numaraları ile soket oluşturulur:

```
DatagramSocket mySocket1 = new
   DatagramSocket(99111);
DatagramSocket mySocket2 = new
   DatagramSocket(99222);
```

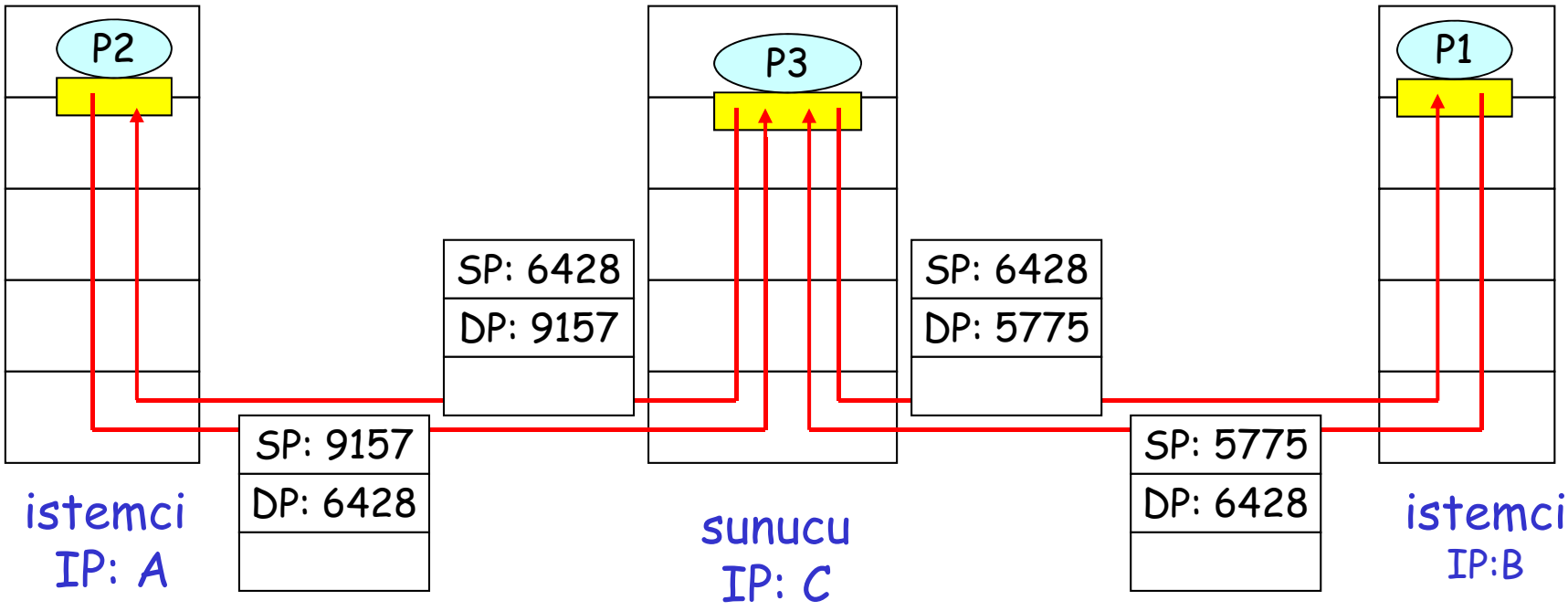
- UDP soketleri iki değişken tarafından tanımlanır:

(hedef IP adresi, hedef port numarası)

- Ana sistem UDP segmentini alınca:
  - Segmentdeki hedef port numarasını kontrol eder
  - UDP segmentini ilgili port numarasına yönlendirir.
- Farklı kaynak IP adresi ve/veya kaynak port numarası olan IP datagramları aynı sokete yönlendirilir

Bağlantısız çoklamanın çözülmesi

```
DatagramSocket serverSocket = new DatagramSocket(6428);
```

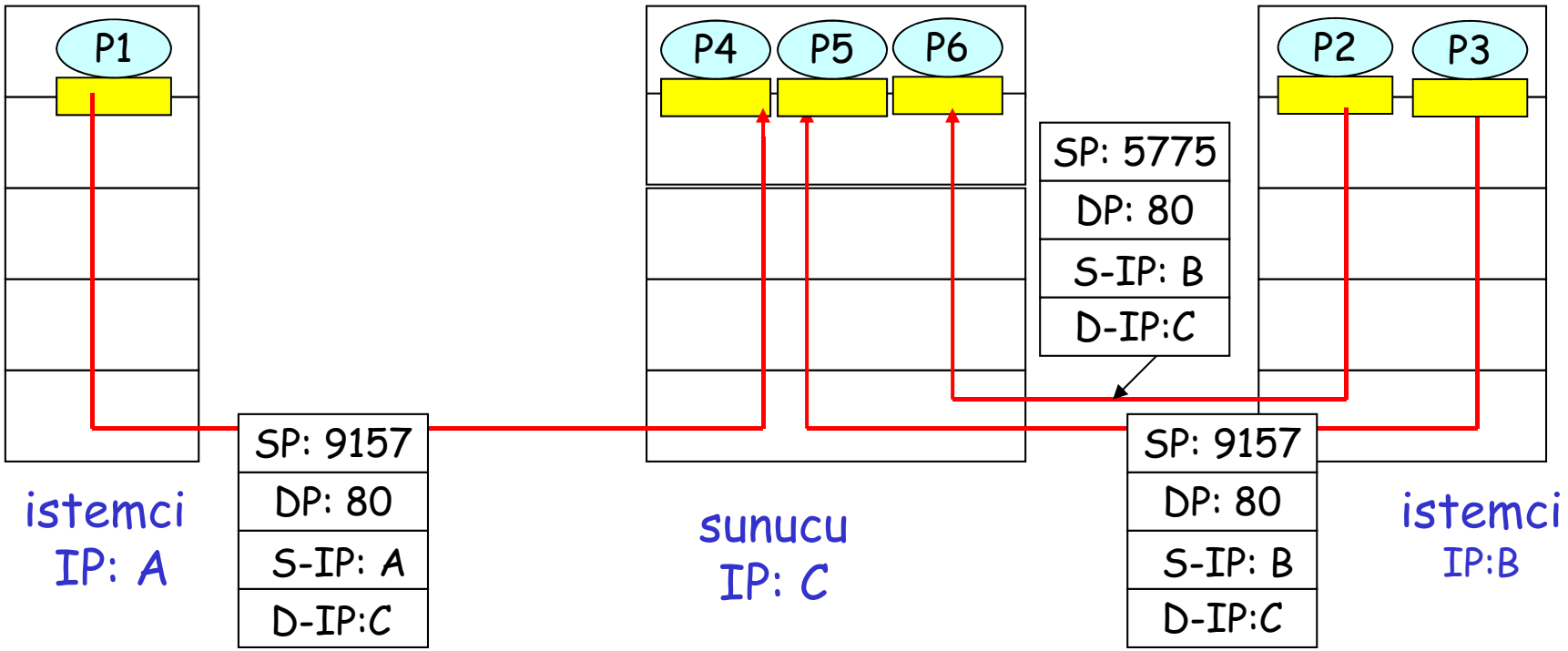


SP "geri dönüş adresi"ni tanımlar

Bağlantı-yönelimli çoklamanın çözülmesi

- TCP soketler dört-değişken tarafından tanımlanır:
  - kaynak IP adresi
  - kaynak port numarası
  - hedef IP adresi
  - hedef port numarasi
- Alıcı ana sistem segmenti uygun sokete yönlendirmek için dört değeri de kullanır
- Sunucu ana sistem pek çok eş zamanlı TCP soketini destekleyebilir:
  - Her bir soket kendi dörtlü değişkeni ile tanımlanır
- Web sunucularının her bağlanan istemci için farklı soketleri vardır
  - Sürekli olmayan HTTP'de her istek için farklı bir soket vardır

Bağlantı-yönelimli çoklamanın çözülmesi



Uygulama Katmanı

- 3.1 Taşıma katmanı servisleri
- 3.2 Çoklama (multiplexing) ve çoklamanın çözülmesi (demultiplexing)
- 3.3 Bağlantısız taşıma: UDP
- 3.4 Güvenilir veri iletiminin prensipleri
- 3.5 Bağlantı yönelimli taşıma: TCP
  - segment yapısı
  - güvenilir veri iletimi
  - akış kontrolü
  - bağlantı yönetimi
- 3.6 Tıkanıklık kontrolü prensipleri
- 3.7 TCP tıkanıklık kontrolü

UDP: User Datagram Protocol [RFC 768]

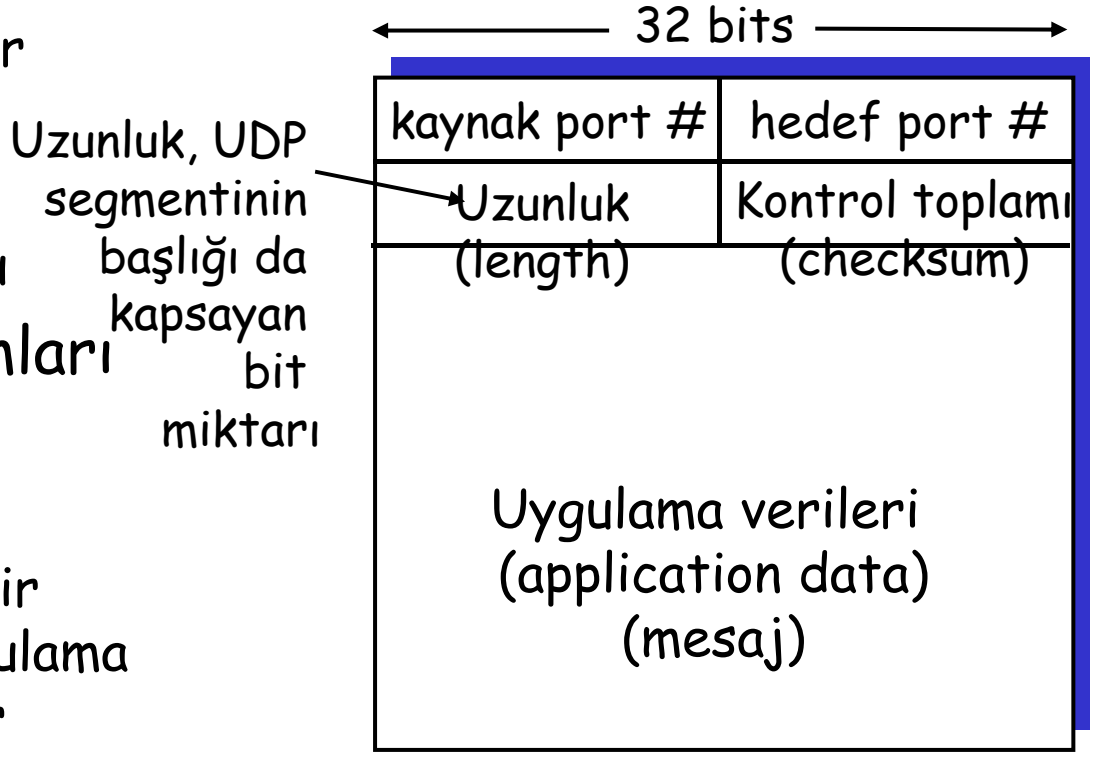
- Bir taşıma protokolünün yapabileceği en az şeyi yapar
- "en iyi çaba" teslim servisi , UDP segmentleri:
  - kaybolabilir
  - Uygulamaya sırasız bir şekilde ulaşabilir
- *bağlantısız:*
  - UDP gönderici ve alıcısı arasında bağlantı kurulumu yoktur
  - Her UDP segmenti diğerlerinden bağımsız olarak yürütülür

Peki neden UDP var?

- Bağlantı kurulumu yok (gecikmeye yol açabilir)
- basit: gönderici ve alıcı taraflarında bağlantı durum yoktur
- Küçük paket başlığı yükü
- Tıkanıklık kontrolü yoktur: UDP istenildiği kadar hızlı gönderebilir

UDP:

- Genellikle multimedia uygulamalarda kullanılır
  - Kayıpların tolere edilebildiği
  - Hızın önemli olduğu
- Diğer UDP kullanımları
  - DNS
  - SNMP
- UDP üzerinden güvenilir iletim: güvenilirlik uygulama katmanında eklenebilir
  - Uygulamaya dayalı hata düzeltimi!



UDP segment yapısı

UDP kontrol toplamı (checksum)

Amaç: iletilen segmentteki "hataları" (örn., değişen bitler) tespit etme

Gönderici:

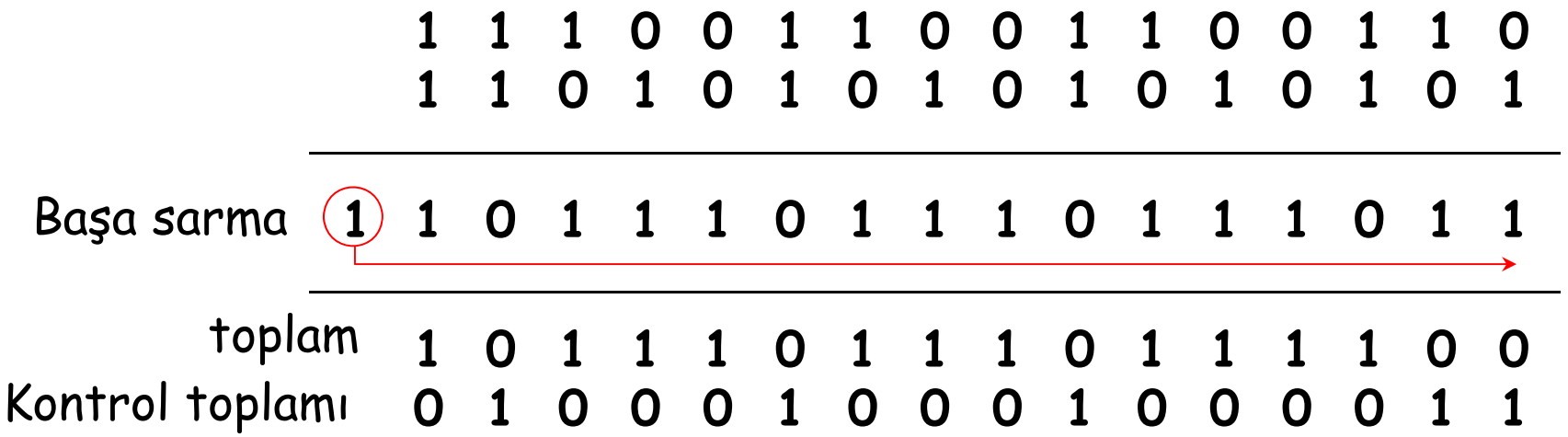
- Segment içeriklerini 16 bitlik sözcükler olarak ele alır
- Kontrol toplamı (checksum): segment içeriğini 1 tümleyicilerinin toplamı olarak toplar
- Gönderici UDP kontrol toplamı alanına kontrol toplamı değeri ekler

Alıcı:

- Alınan segmentin kontrol toplamını hesaplar
- Hesaplanan kontrol toplamının kontrol toplamı alanındaki değere eşitliğini kontrol eder:
  - HAYIR – hata tespit edilmiştir
  - EVET – hata tespit edilmemiştir. Fakat yine de hatalar olabilir? Daha sonra bahsedeceğiz

Internet Kontrol Toplamı (Checksum) Örneği

- Örnek: iki 16-bit integerı toplayalım

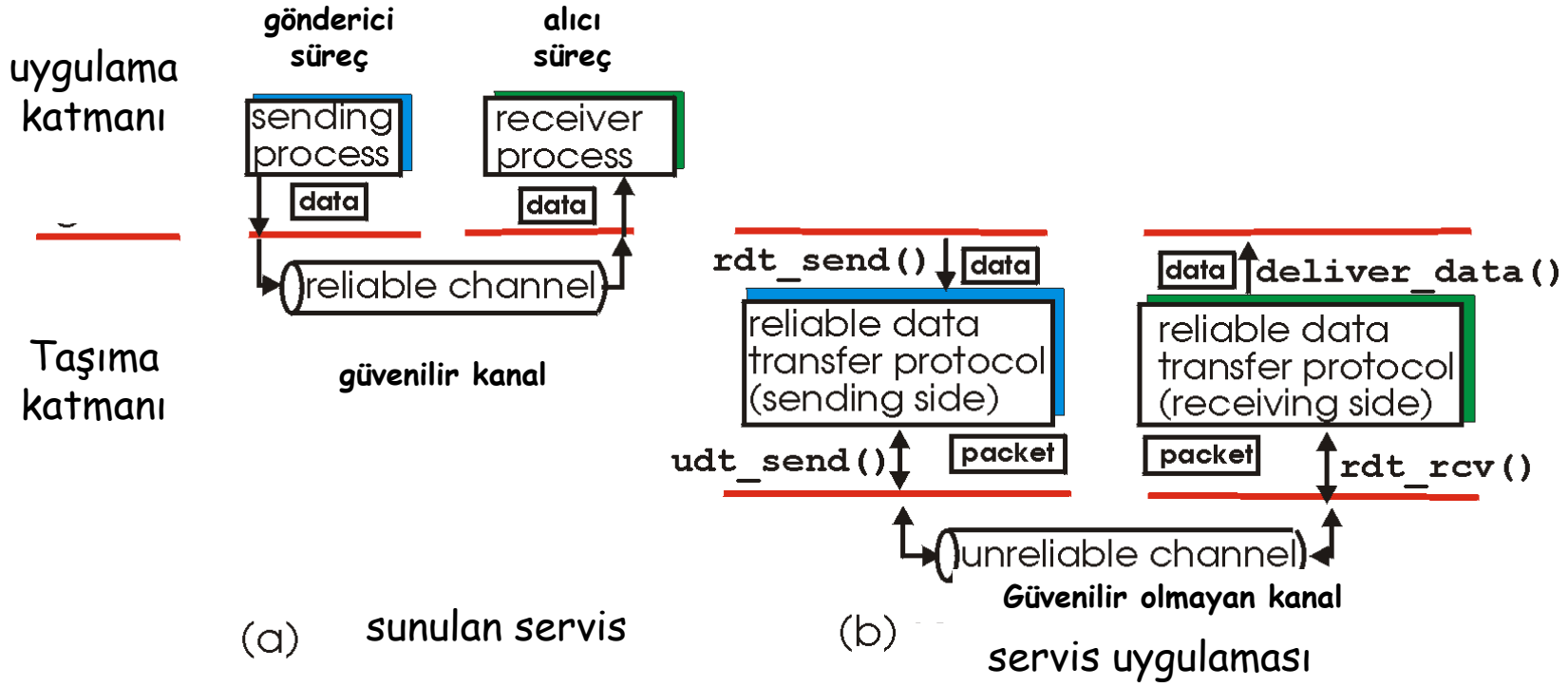


Taşıma Katmanı

- 3.1 Taşıma katmanı servisleri
- 3.2 Çoklama (multiplexing) ve çoklamanın çözülmesi (demultiplexing)
- 3.3 Bağlantısız taşıma: UDP
- 3.4 Güvenilir veri iletiminin prensipleri
- 3.5 Bağlantı yönelimli taşıma: TCP
  - segment yapısı
  - güvenilir veri iletimi
  - akış kontrolü
  - bağlantı yönetimi
- 3.6 Tıkanıklık kontrolü prensipleri
- 3.7 TCP tıkanıklık kontrolü

Güvenilir Veri İletiminin Prensipleri

- Sadece taşıma katmanında değil, uygulama ve bağlantı katmanlarında da meydana gelir
- En önemli ağ konularının ilk-10 listesinin başında gelen bir konudur!

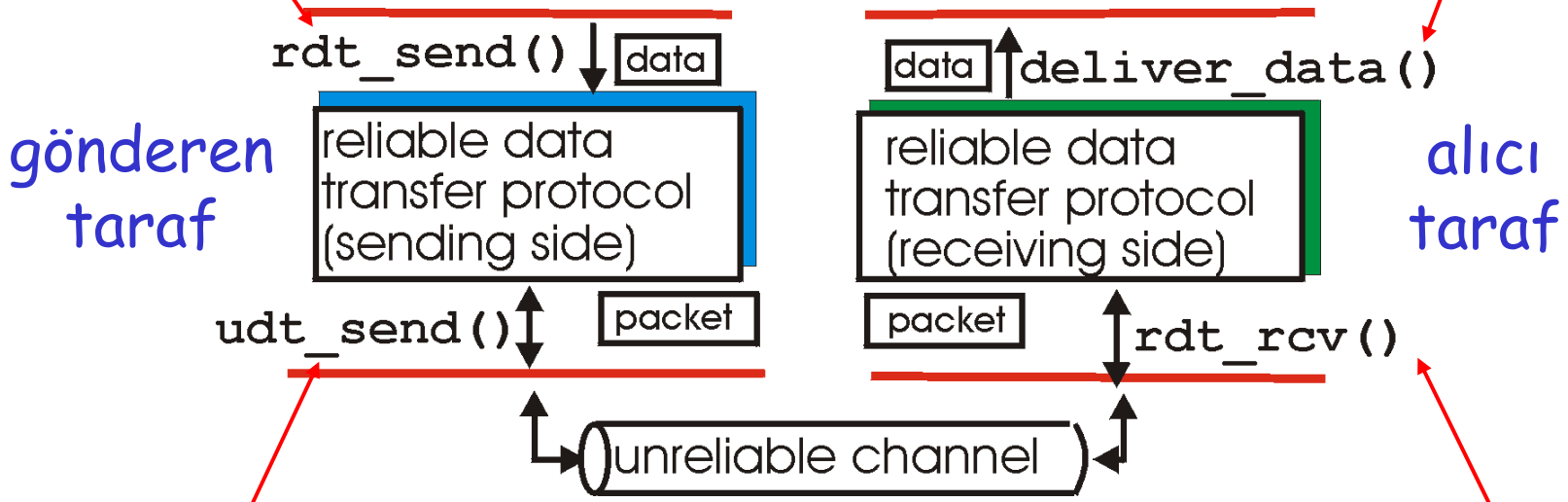


- Güvenilmeyen kanalın özellikleri güvenilir veri iletiminin (rdt) karmaşıklığını belirler

Güvenilir veri iletimi: başlangıç

rdt_send(): yukarıdan çağrılır, (e.g., uygulama tarafından). Teslim edilecek verileri alıcı taraftaki yukarı katmana aktarır

deliver_data(): rdt protokolü, üst katmana veri teslim etmek istediğinde kullanır



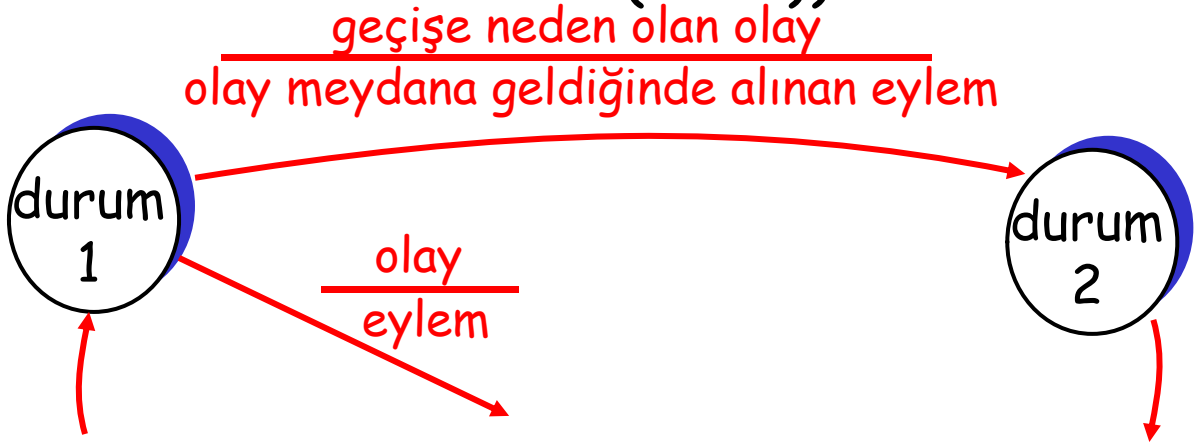
udt_send(): rdt protokolünün gönderici ve alıcı taraflarının ikisi de bu çağrı ile paketleri güvenilir olmayan kanalda diğer tarafa gönderirler

rdt_rcv(): alıcı tarafta bir paket, kanalın alıcı tarafına geldiğinde çağrılır

Güvenilir veri iletimi: başlangıç

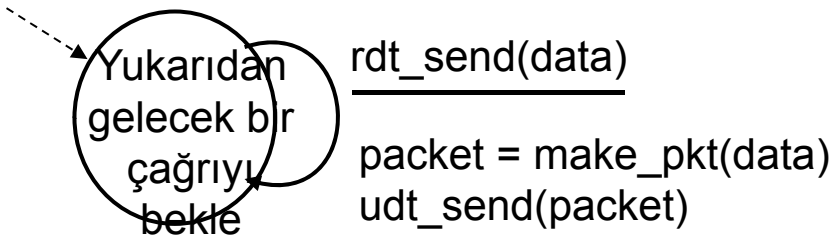
- Güvenilir veri iletim protokolünü (rdt) adım adım oluşturacağız
- Tek yönlü veri iletimi varsayalım
  - Fakat kontrol bilgisi her iki yönde de akar!
- Gönderici ve alıcıyı tanımlamak için sonlu durum makinası (use finite state machines (FSM)) kullanalım

durum: bu "durumda" iken diğer durum sonraki olay ile belirlenir

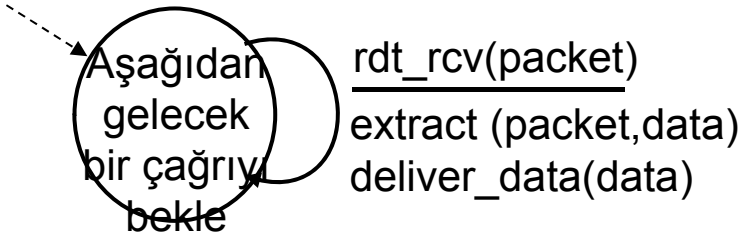


Rdt1.0: güvenilir bir kanal üzerinde güvenilir veri iletimi

- Alttaki kanal son derece güvenilir
  - bit hataları yok
  - paketler kaybolmuyor
- Alıcı ve göndereci için ayrı FSM'ler:
  - gönderici veriyi alttaki kanala gönderir
  - alıcı veriyi alttaki kanaldan okur



gönderici

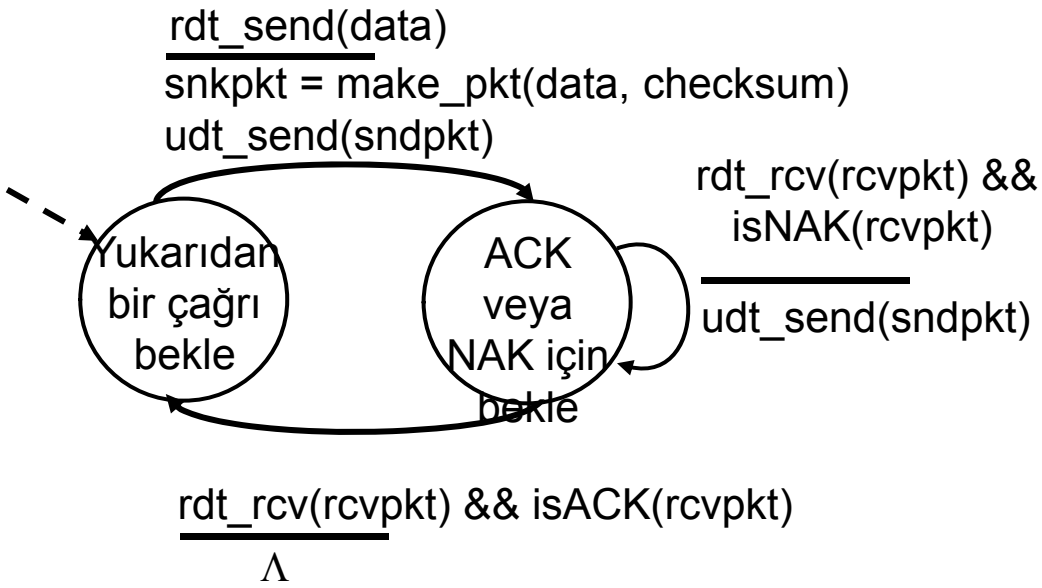


alıcı

Rdt2.0: bit hatalarına sahip bir kanal

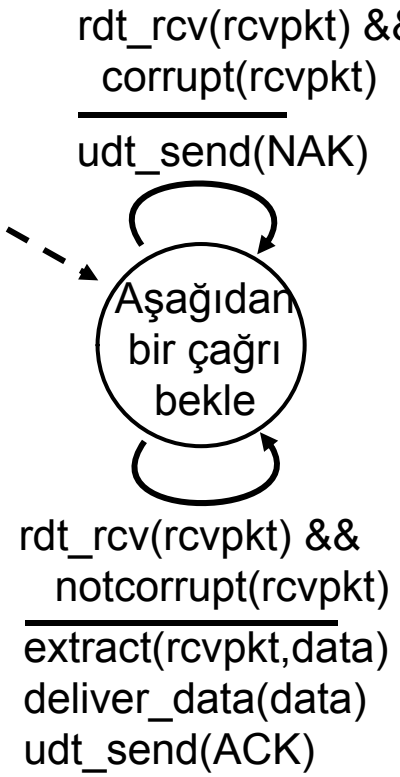
- Alttaki kanal paketlerdeki bitleri değiştirebilir
  - bit hatalarını tespit etmek için kontrol toplamı
- *Soru :* hataları nasıl düzelteceğiz?
  - *pozitif alındı (acknowledgements (ACKs)):* alıcı açık olarak göndericiye paketin alındığını bildirir
  - *negatif alındı (negative acknowledgements (NAKs)):* alıcı göndericiye pakette hatalar olduğunu söyler
  - Gönderici NAK üzerine paketine yeniden gönderir
- **`rdt2.0`** daki yeni mekanizma (**`rdt1.0'in üstünde`**):
  - Hata tespiti
  - Alıcı geribildirimi: gönderici alıcı arasındaki kontrol mesajları (ACK,NAK)
  - Tekrar iletim

rdt2.0: FSM belirtimi

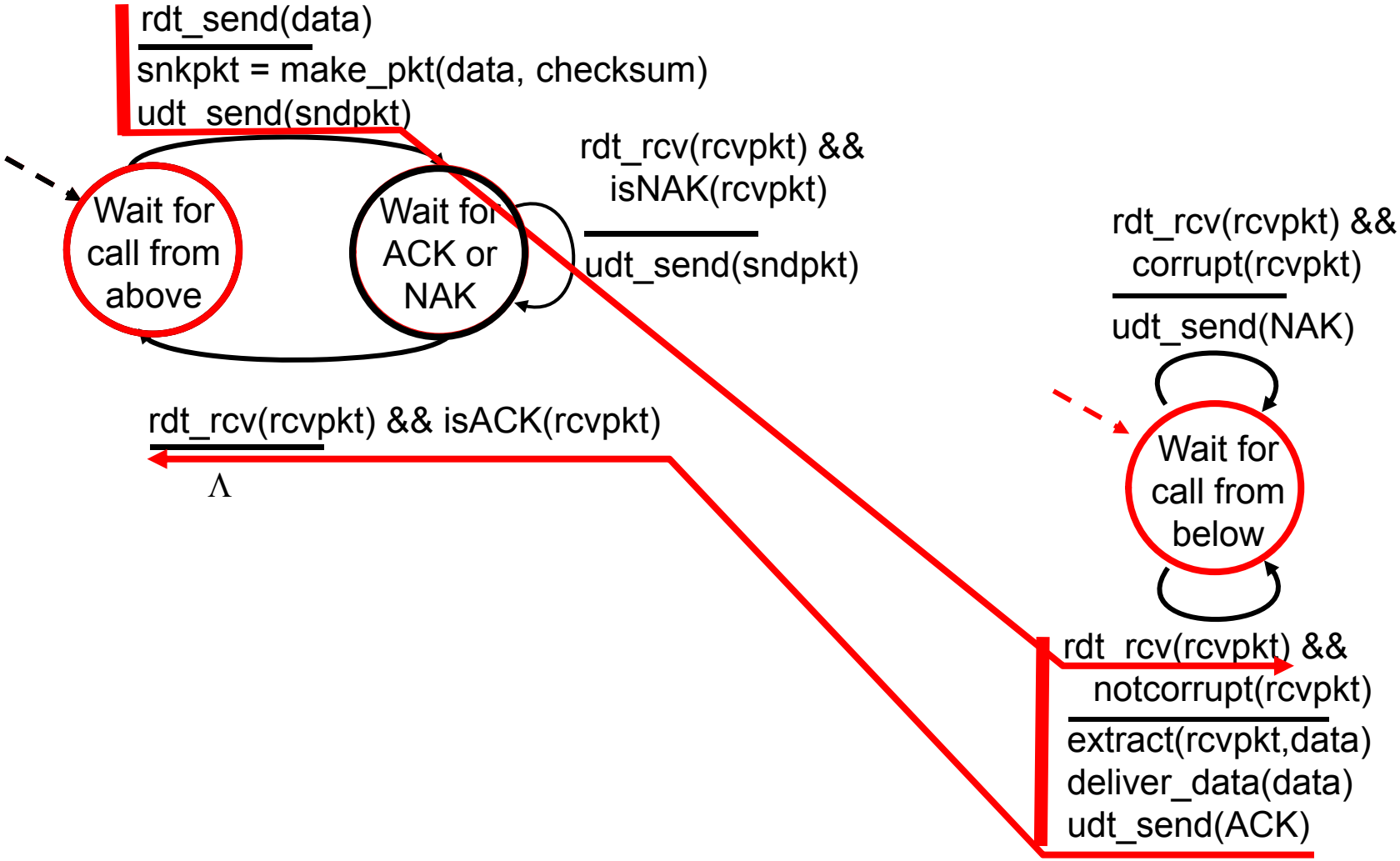


gönderici

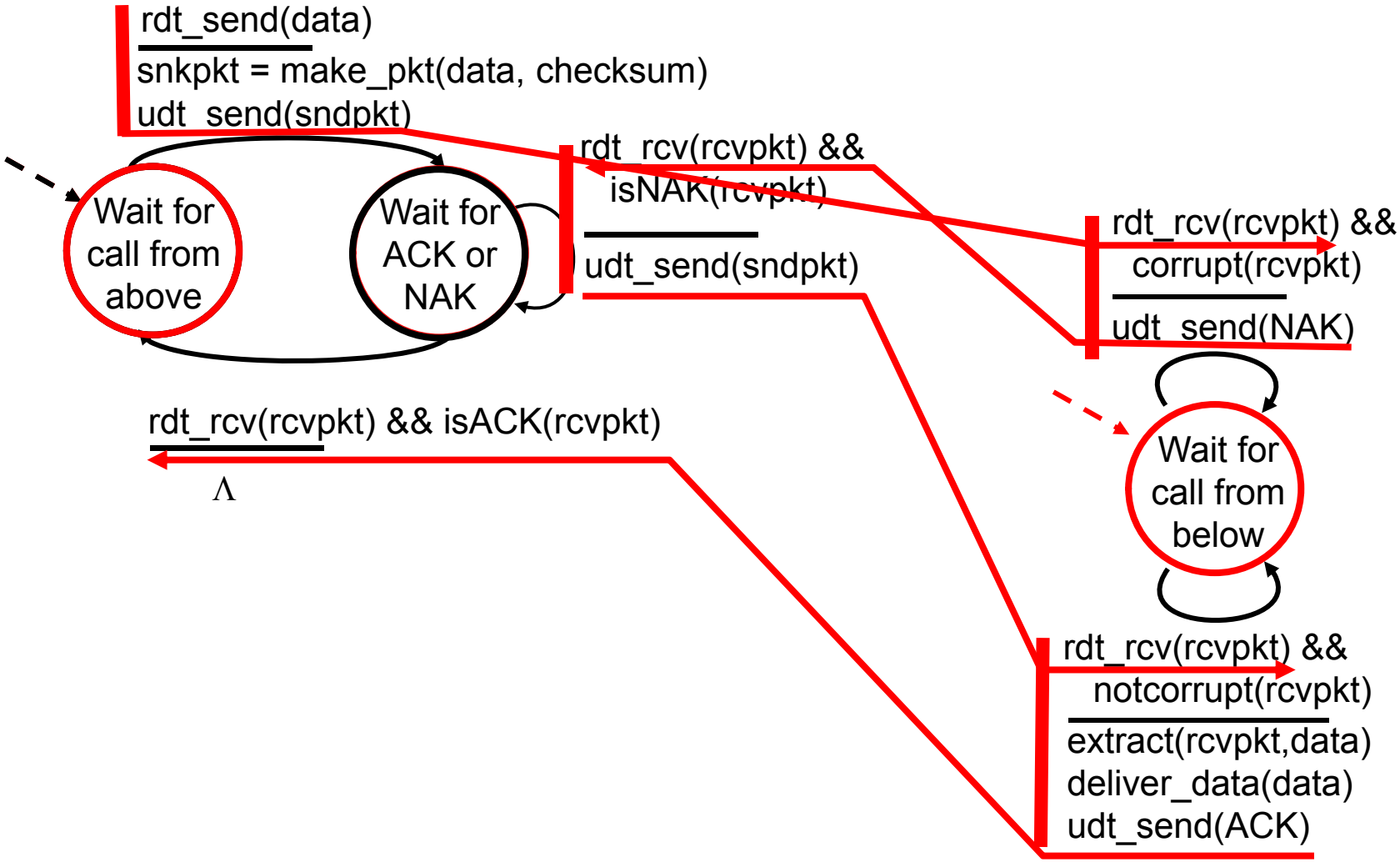
alıcı



rdt2.0: hatasız işlem



rdt2.0: hatalı senaryo



rdt2.0 önemli bir eksiği var!

Eğer ACK ya da NAK bozulursa ne olacak?

- Gönderici alıcıda ne olduğunu bilmiyor!
- Yeniden gönderirse : pek çok kopya paket ortaya çıkar

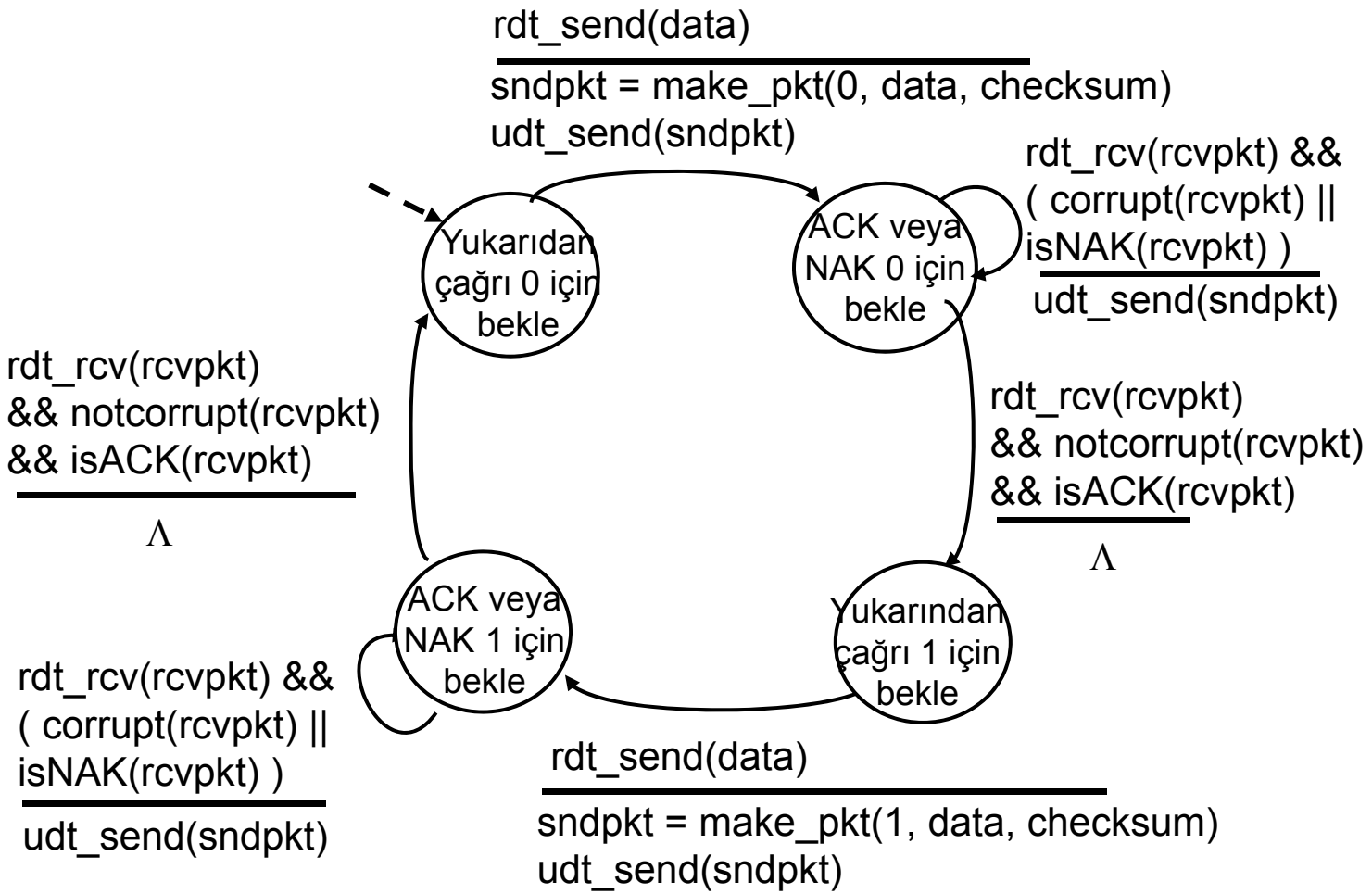
Kopyaların üstesinden gelmek:

- Gönderici her pakete *dizi numarası (sequence number)* ekler
- Gönderici eğer ACK/NAK bozulduysa paketi yeniden gönderir.
- Alıcı kopya paketleri atar (yukarı göndermez)

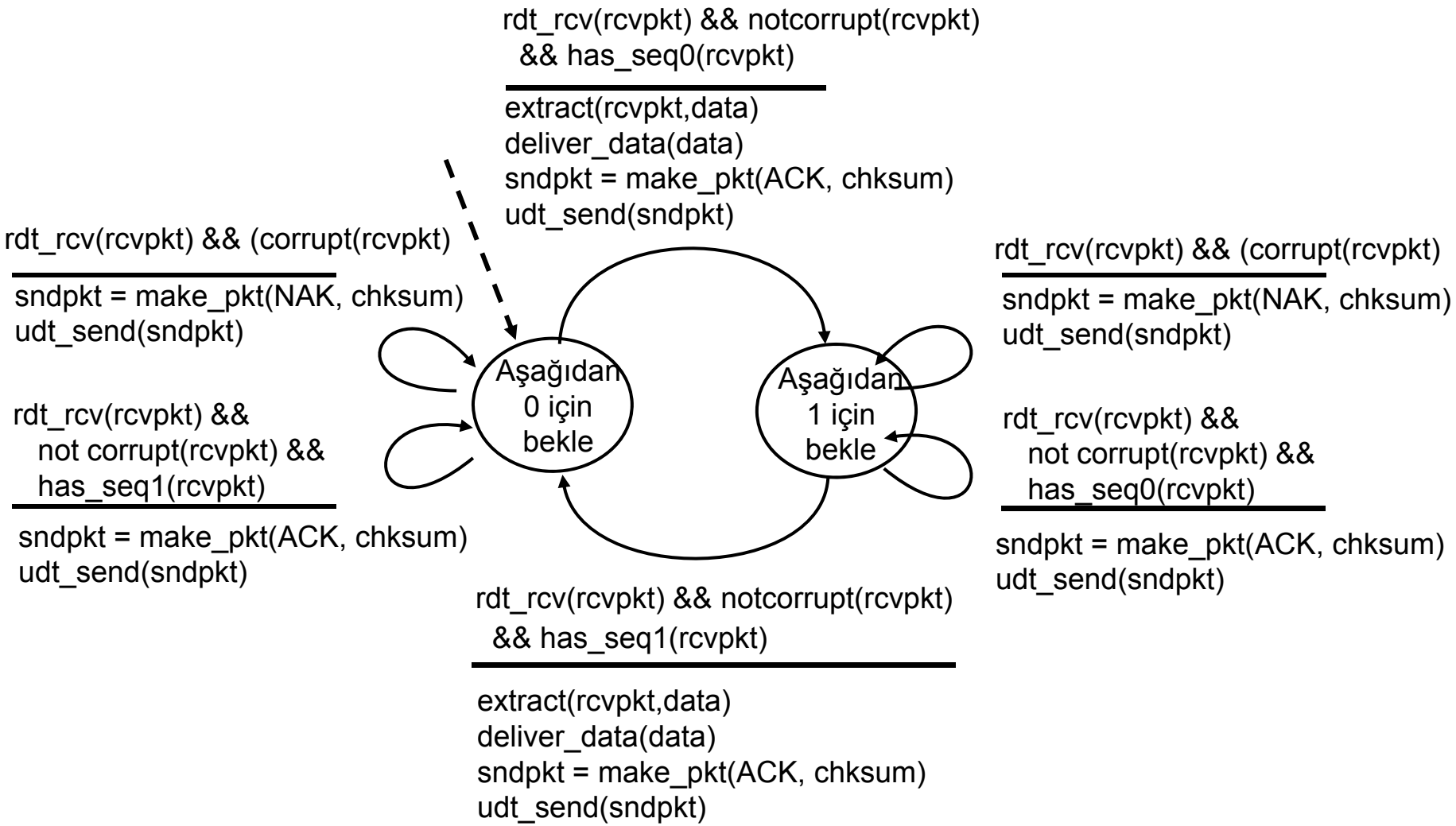
Dur ve bekle
stop and wait

Gönderici bir paket gönderir sonra alıcının cevabını Bekler.

rdt2.1: göndericinin bozuk ACK/NAK'lerin üstesinden gelmesi



rdt2.1: alıcının bozuk ACK/NAK'lerin üstesinden gelmesi



rdt2.1: özet

Gönderici:

- Paketlere dizi numarası eklendi
- (0,1) iki dizi numarası yeterli mi? Neden?
- Alınan ACK/NAK'in bozuk olup olmadığı kontrol edilmelidir
- FSM'ler daha öncekinin iki katı duruma sahiptir
  - Durum paketin 0 ya da 1 dizi numarasına sahip olup olmadığını bilmelidir

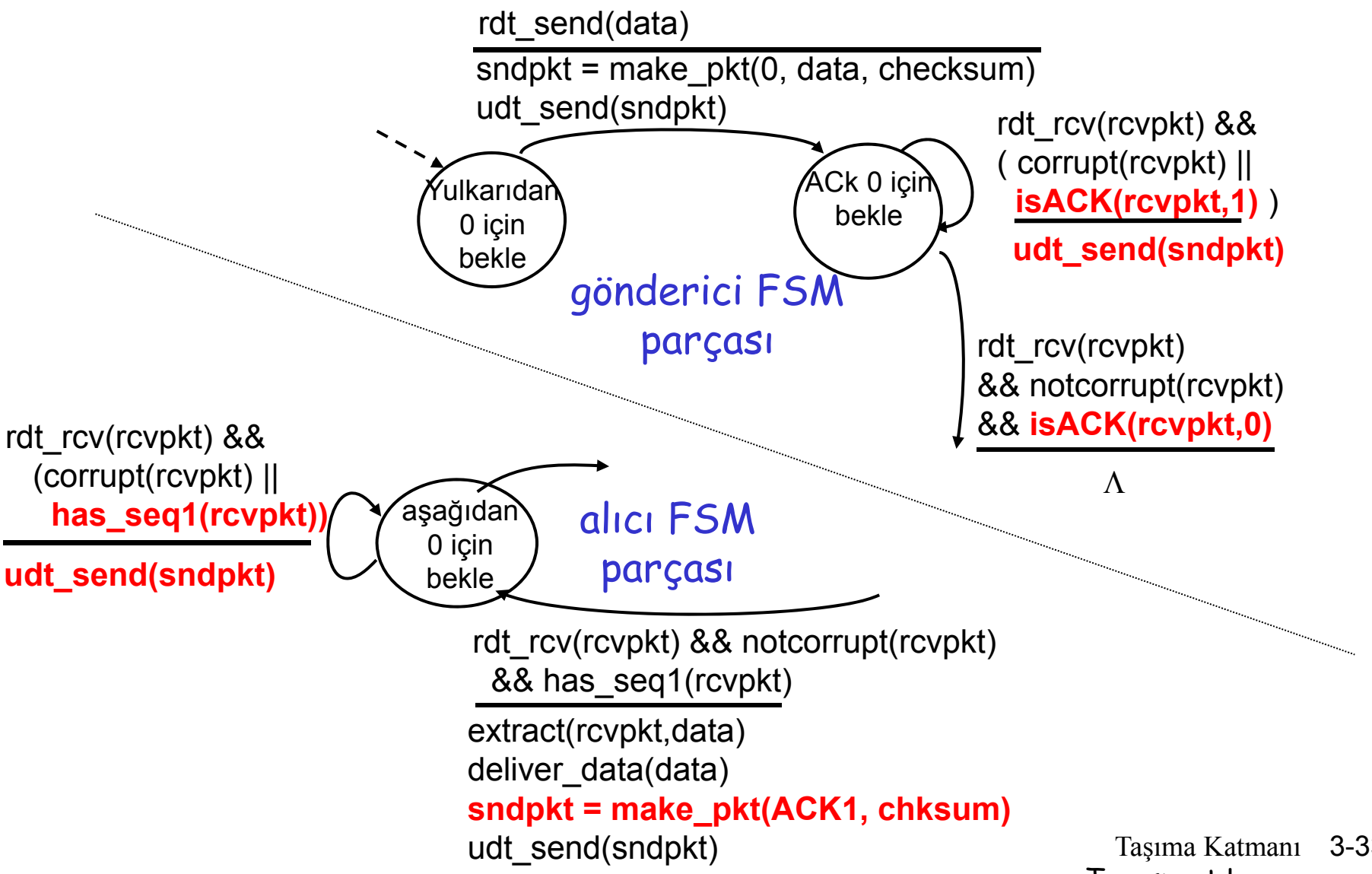
Alıcı:

- Alınan paketin kopya olup olmadığını kontrol etmelidir
  - Durum beklenen paket dizi numarasının 0 ya da 1 olmasını belirler
- not: alıcı göndericideki alınana son ACK yada NAK'ın OK olup olmadığını bilemez

rdt2.2: NAK kullanmayan protokol

- Sadece ACK kullanarak rdt2.1'deki aynı fonksiyonlara sahiptir
- NAK yerine, alıcı sadece son doğru alınan paket için ACK gönderir
  - Alıcı ACK'lenen paket için dizi numarasını açıkça belirtmelidir
- Göndericide kopya ACK ler NAK ile aynı eyleme sebep olur: *paketi yeniden göndermek*

rdt2.2: gönderici alıcı parçaları



rdt3.0: bit hatalı ve kayıplı kanal için

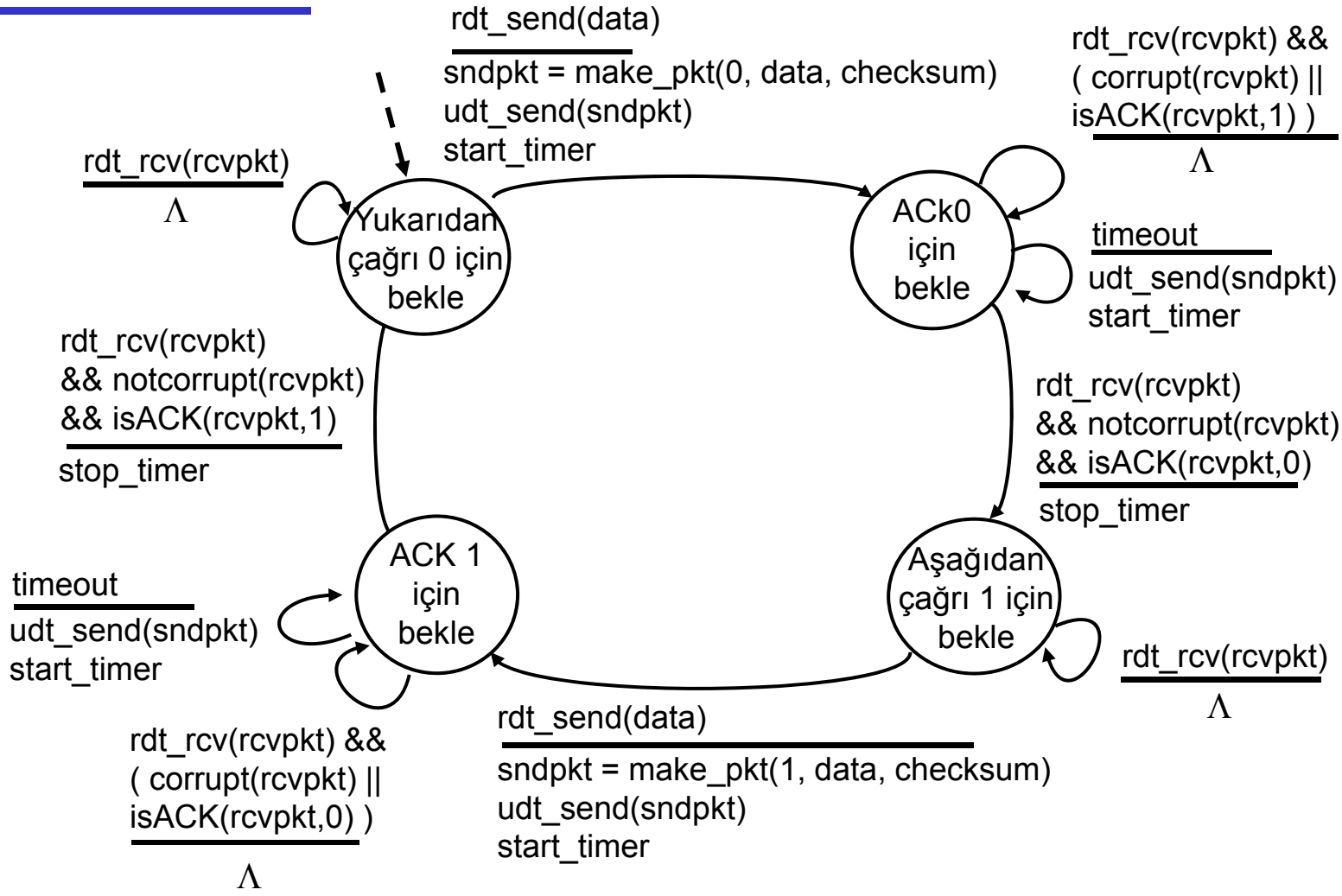
Yeni varsayım: alttaki kanal paketleri kaybedebilir (veri aya da ACK'leri)

- Kontrol toplamı (checksum), dizi sayısı (seq. #), ACKler, yeniden gönderimler faydalı ancak yeterli değiller

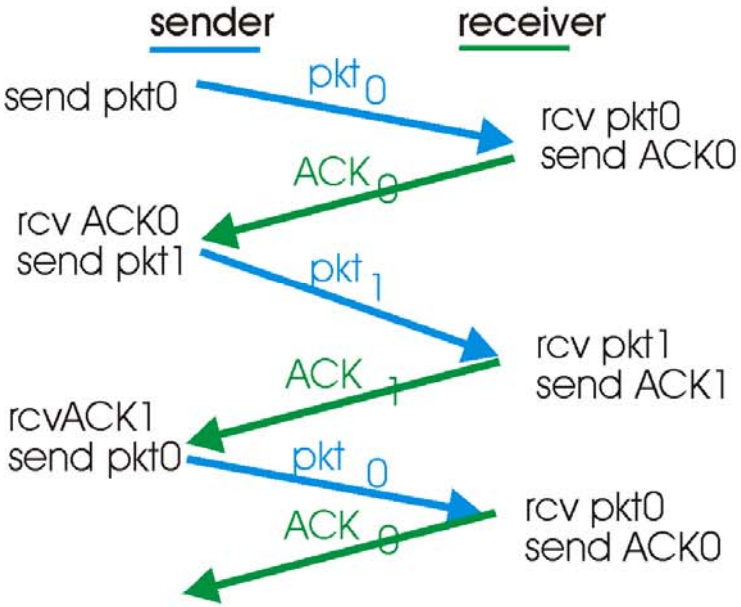
Yaklaşım: gönderici "makul" bir süre kadar ACk için beklemelidir.

- Bu sürede ACk gelmezse yeniden gönderir
- Eğer paket (veya ACK) sadece geciktiyse (kaybolmadıysa):
  - Yeniden gönderilen kopya olacaktır, ancak dizi numarası kullanımı bunu üstesinden gelecektir
  - Alıcı ACK'lenen paketin dizi numarasını belirtmelidir
- Geri sayım zamanlayıcısı (countdown timer) gerektirir

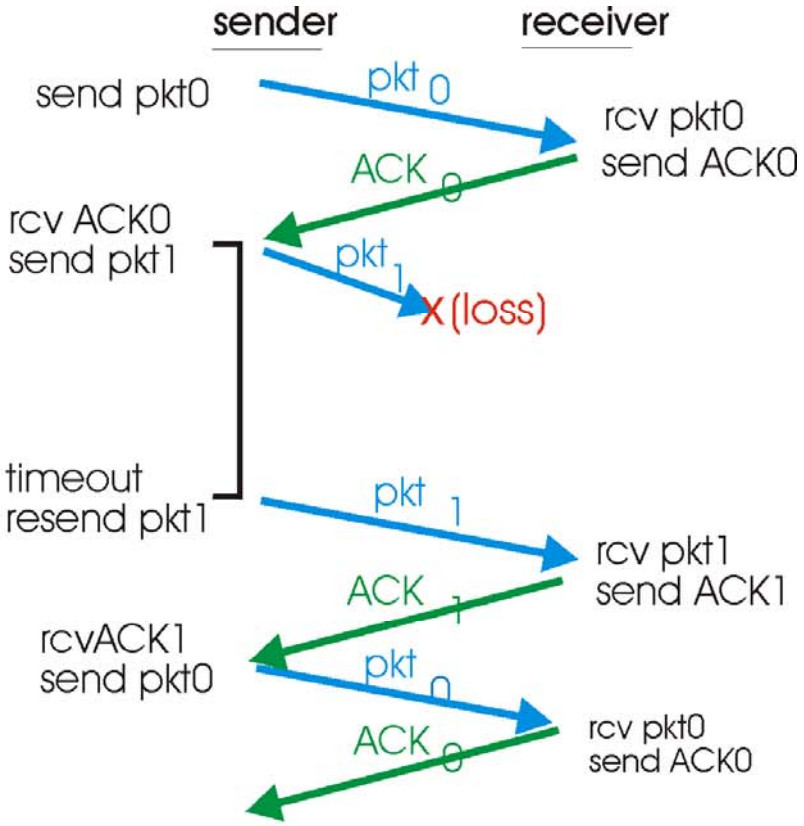
rdt3.0 gönderici



rdt3.0 protokolünün işleyişi

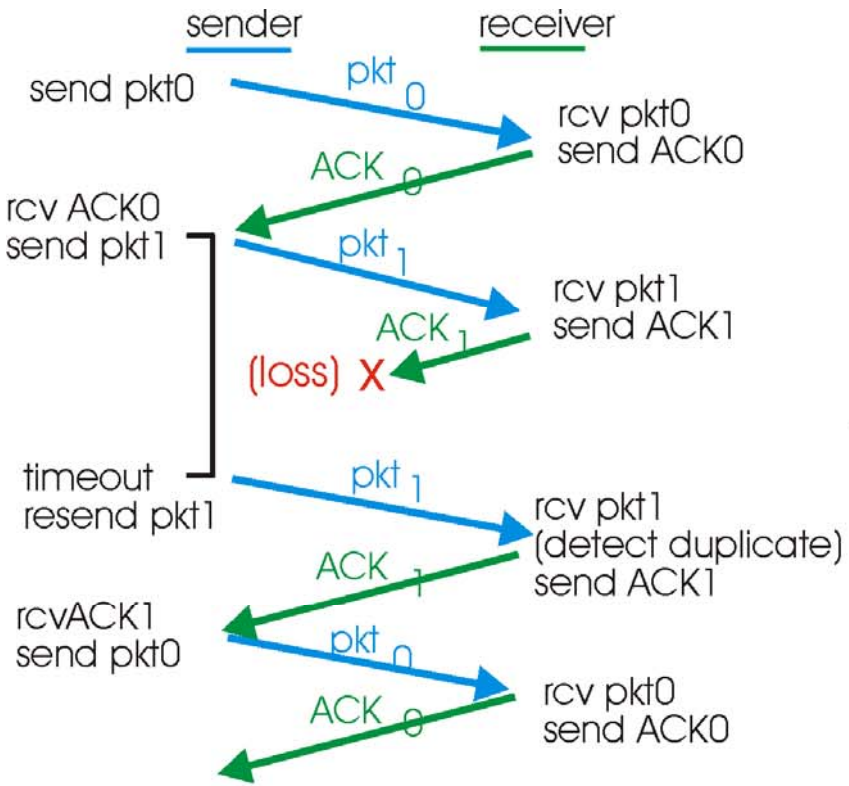


(a) kayıpsız işlem

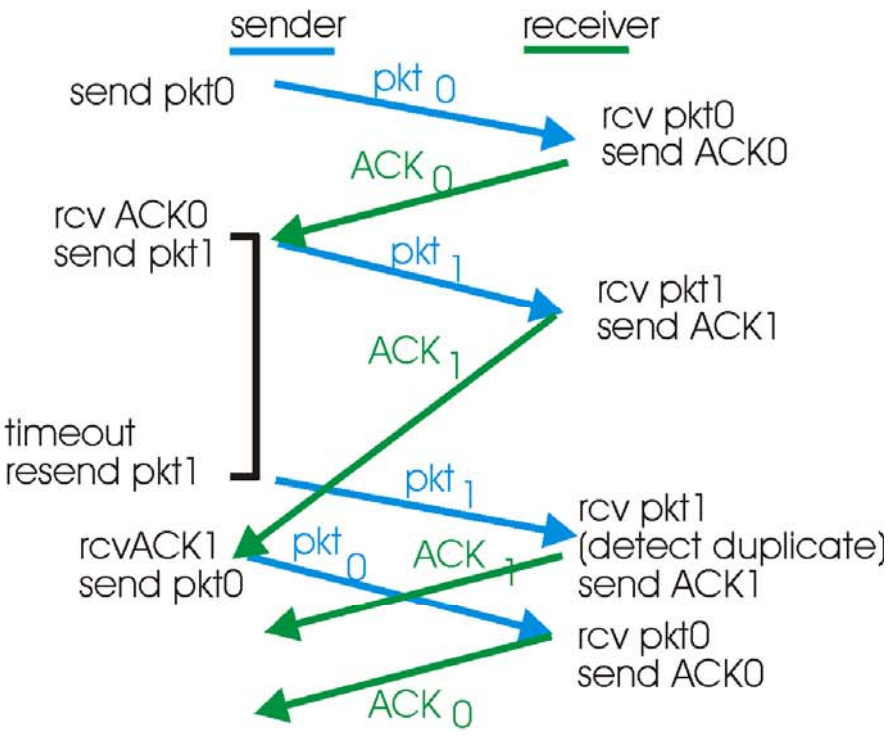


(b) kayıp paket

rdt3.0 in action



(c) kayıp ACK



(d) Erken zaman aşımı

rdt3.0 protokolünün performansı

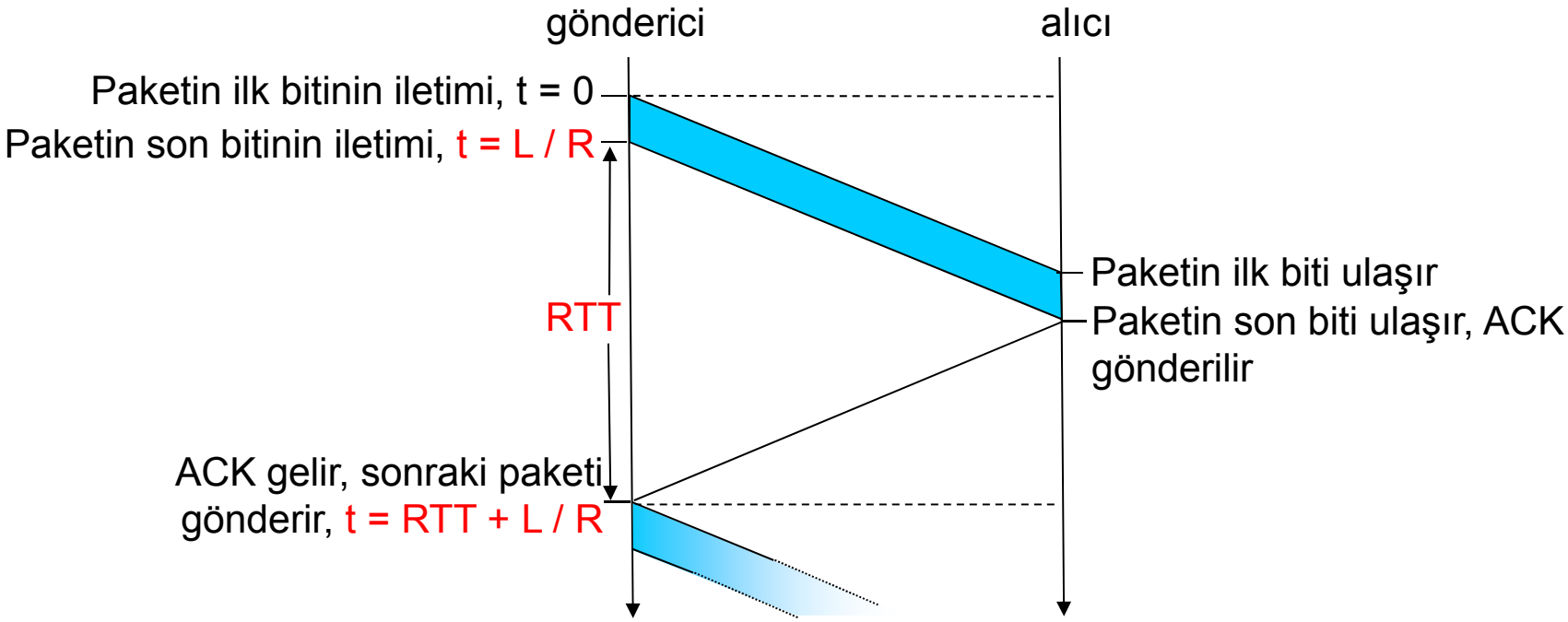
- rdt3.0 çalışır fakat performansı pek de iyi değildir
- örnek: 1 Gbps hatta, 15 ms uçtan uca yayılma gecikmesi ile, 1KByte'lik paket gönderirken:

$$d_{ilet} = \frac{L \text{ (packet length in bits)}}{R \text{ (transmission rate, bps)}} = \frac{8kb/pkt}{10**9\ b/sec} = 8 \text{ microsn}$$

$$U_{gönderici} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

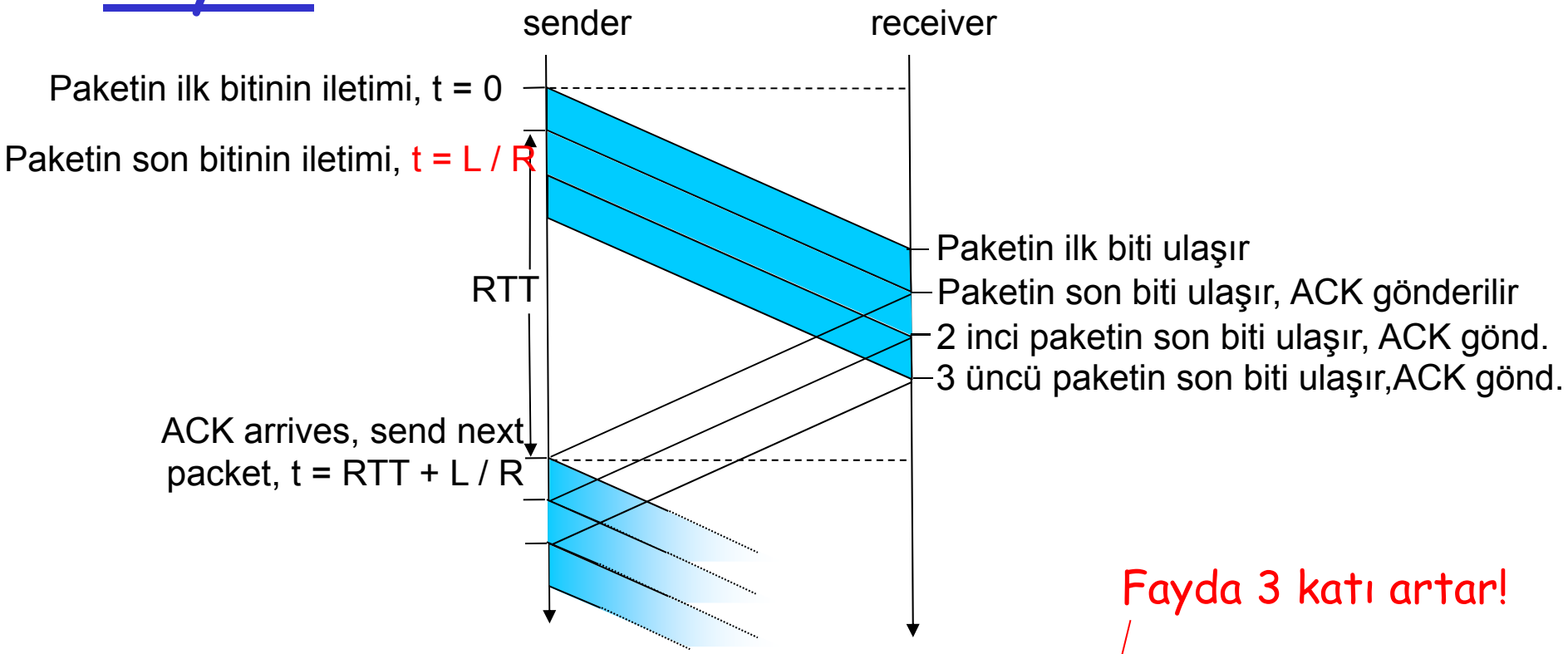
  - $U_{gönderici}$: fayda – göndericinin göndermekle meşgul olduğu sürenin kesri
  - her 30.008 msn'de 8KB'lik paket -> 1 Gbps'lik hatta 267Kbps etkili akış
  - Fiziksel kaynakların kullanımını ağ protokolü kısıtlar!

rdt3.0: dur ve bekle işleyişi



$$U_{\text{gönderici}} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

İç içe geçme (pipeline) hattı: artan fayda



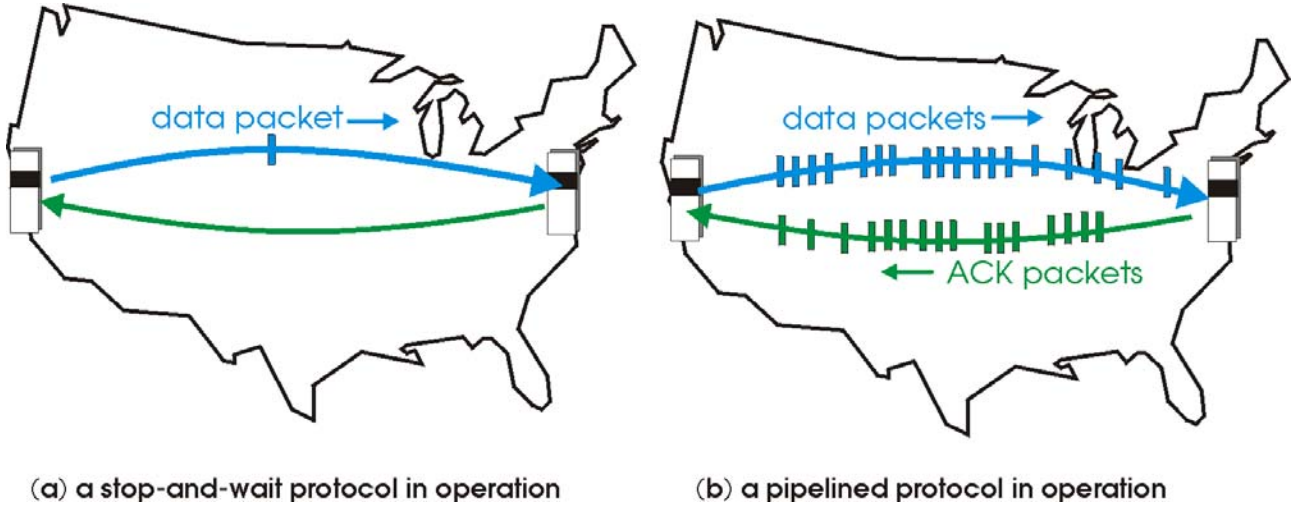
Fayda 3 katı artar!

$$U_{\text{gönderici}} = \frac{3 * L / R}{RTT + L / R} = \frac{.024}{30.008} = 0.0008$$

İç içe geçme (pipeline) protokolü

Boru hattı: gönderici alındı geri bildirimi yapılacak pek çok paketi ardı ardına gönderir

- Dizi numara aralığı arttırılmaldır
- Gönderici ve alıcı taraflarında tampona alma gerekir



(a) a stop-and-wait protocol in operation

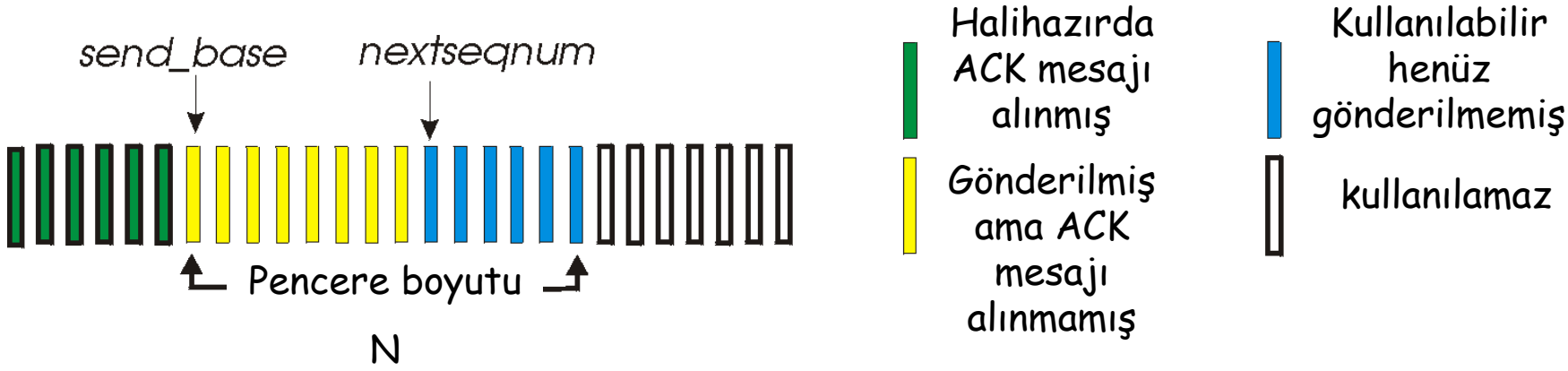
(b) a pipelined protocol in operation

- İç içe geçme (pipeline) protokollerinde iki temel yaklaşım:
  - *Geri Git N (go-Back-N),*
  - *Seçici Tekrarlama (selective repeat)*

Geri Git N (Go-Back-N: GBN)

Gönderici:

- Paket başlığında k-bitlik dizi numarası
- Henüz alındı bilgisi alınmamış paketler için N boyutunda "pencere" (window) boyutu

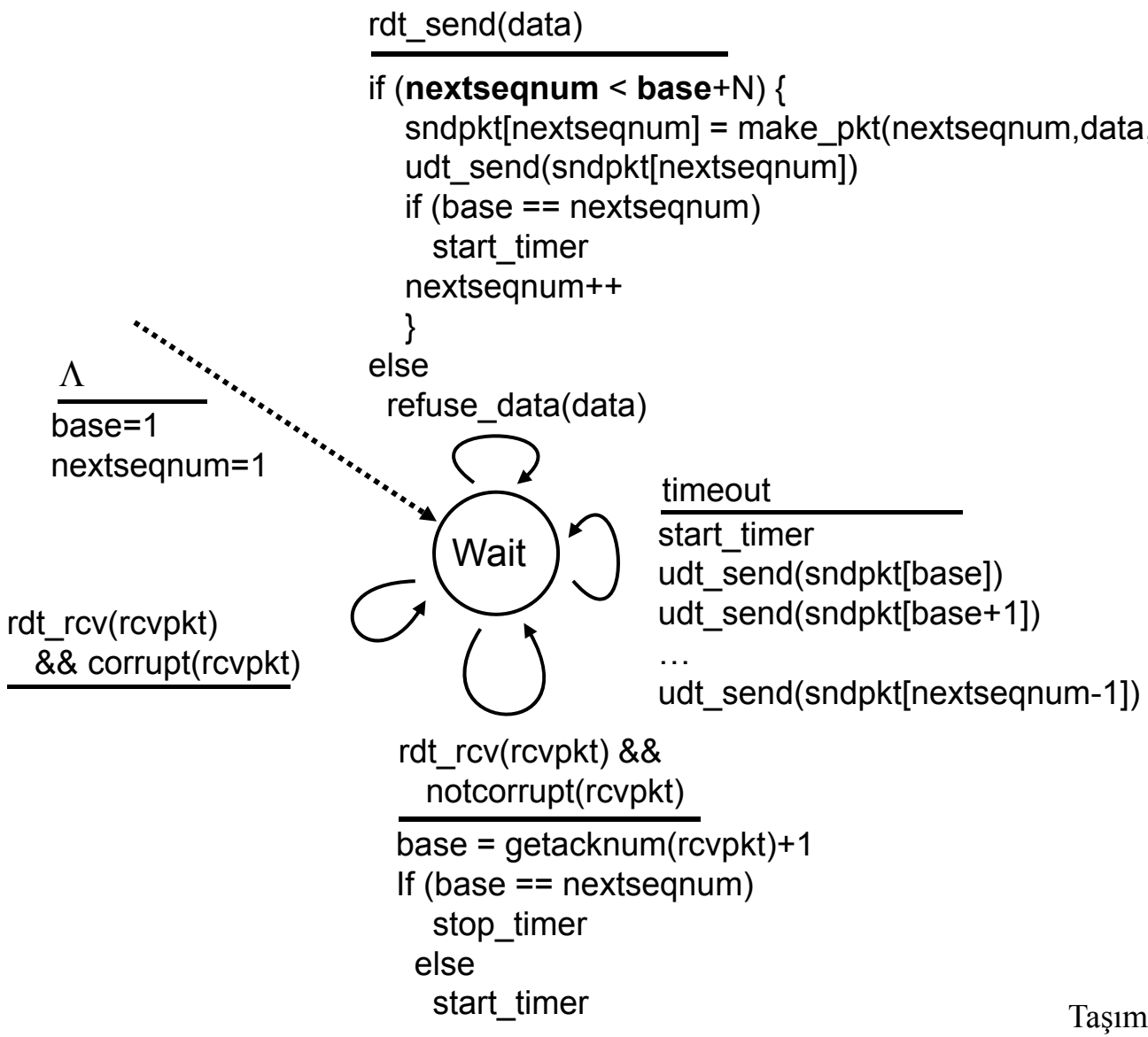


- ACK(n): dizi numarası n olan dahil tüm paketleri ACKler – "kümülatif ACK" (cumulative ACK)
- Gönderimdeki her paket için *zamanlayıcı*
- *Zaman aşımı (n):* penceredeki paket n ve dizi numarası daha yüksek olan paketleri yeniden iletir

Geri Git N (Go-Back-N: GBN)

- Protokol çalıştıkça "N Penceresi" yavaş yavaş sağa doğru kayacaktır. Bu yüzden N'e Pencere Boyutu (Window size), GBN protokolüne de "sliding window" adı verilmektedir.
- Neden bir pencere tanımlamak zorundayız? Kullanıcının limitsiz bir window size (pencere boyutu) kullanarak veri aktarımına izin veremez miyiz?

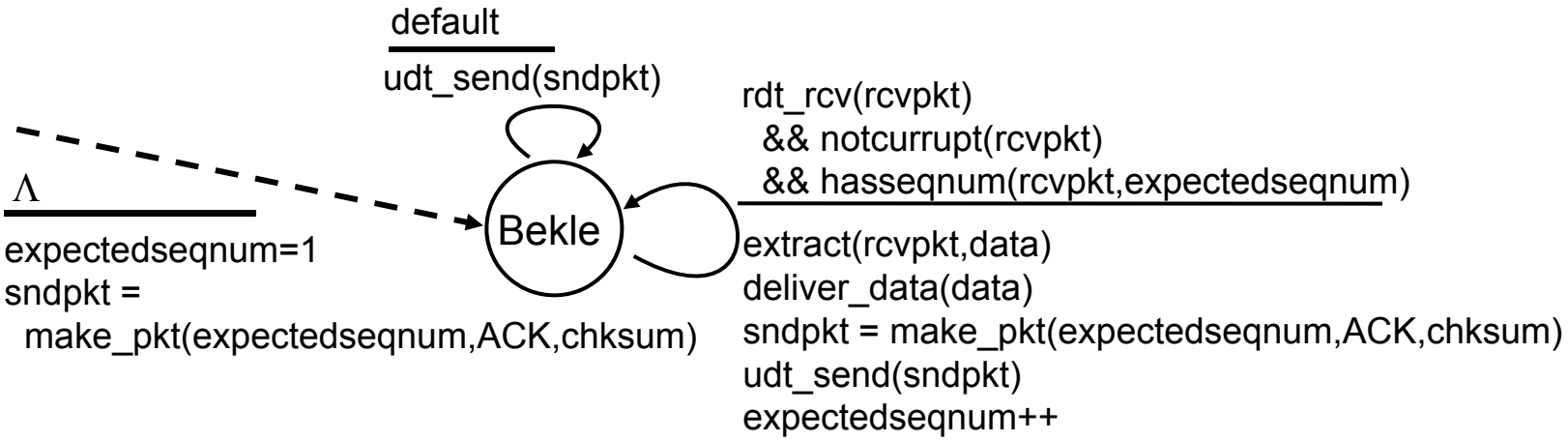
GBN göndericisinin genişletilmiş tanımı



GBN göndericisi

- Gönderici tarafında GBN'in cevap vermesi gereken durumlar:
  - Üst katmandan uyandırılma: Eğer window boşsa, paketi hazırla ve gönder. Eğer Window dolmuşsa uygulamayı uyar.
  - ACK alımı: Sequence number'ı N olan ACK alındığında, o Sequence number'a kadar olan tüm paketlerin sorunsuz ulaştığı anlaşılacaktır.(kümülatif alındı (ACK))
  - Zaman aşımı durumu: doğrulanmamış olan son paket için bir kronometre çalıştırılabilir. Her ACK alımında eğer ACK'sı bekleyen paket varsa kronometre baştan başlatılabilir.

GBN alıcısının genişletilmiş tanımı



sadece ACK'li: doğru olarak alınan en büyük sıralı dizin numarasına sahip paket için her zaman ACK gönderir

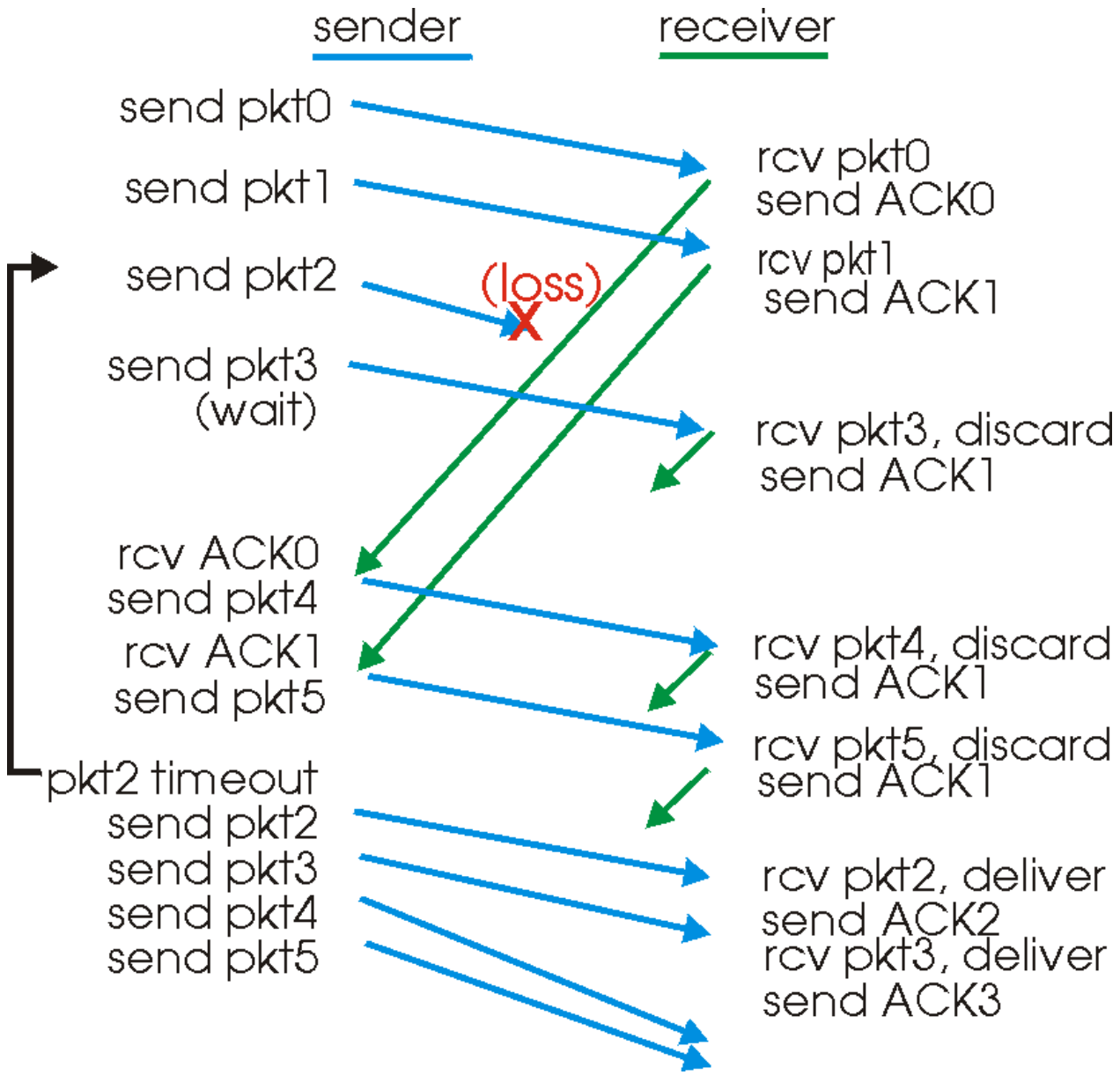
- Kopya ACK'ler oluşabilir
- **`Expectedseqnum`** ı bilmelidir

- Sıralama dışı paket:
  - atılır (tampona alınmaz) -> alıcıda tampona alma yoktur!
  - En büyük sıralı dizi numarası olan paket yeniden ACK'lenir

GBN alıcısı

- Alıcı tarafında GBN:
  - N sequence number'ı olan doğru ve sıralı bir paket alınmışsa (yani bir üst katmana en son gönderilmiş olan paket n-1 ise) alıcı N sequence number'lı paket için bir ACK oluşturur ve bunu gönderir, paketi de uygulama katmanına iletir.
  - Tüm diğer durumlarda paket'i atar ve son düzgün alınmış ve sıralı olan paket için bir ACK oluşturur ve gönderir. Paketler tek tek uygulama katmanına iletildiğine göre K sequence number'lı paketin alınıp uygulama katmanına iletilmesi K'ya kadar olan tüm paketlerin düzgün ver sıralı alındığını anlamına gelir. Dolayısıla kümülatif doğrulama (cumulative acknowledgements) GBN'nin doğasında bulunmaktadır.

GBN in işleyişi



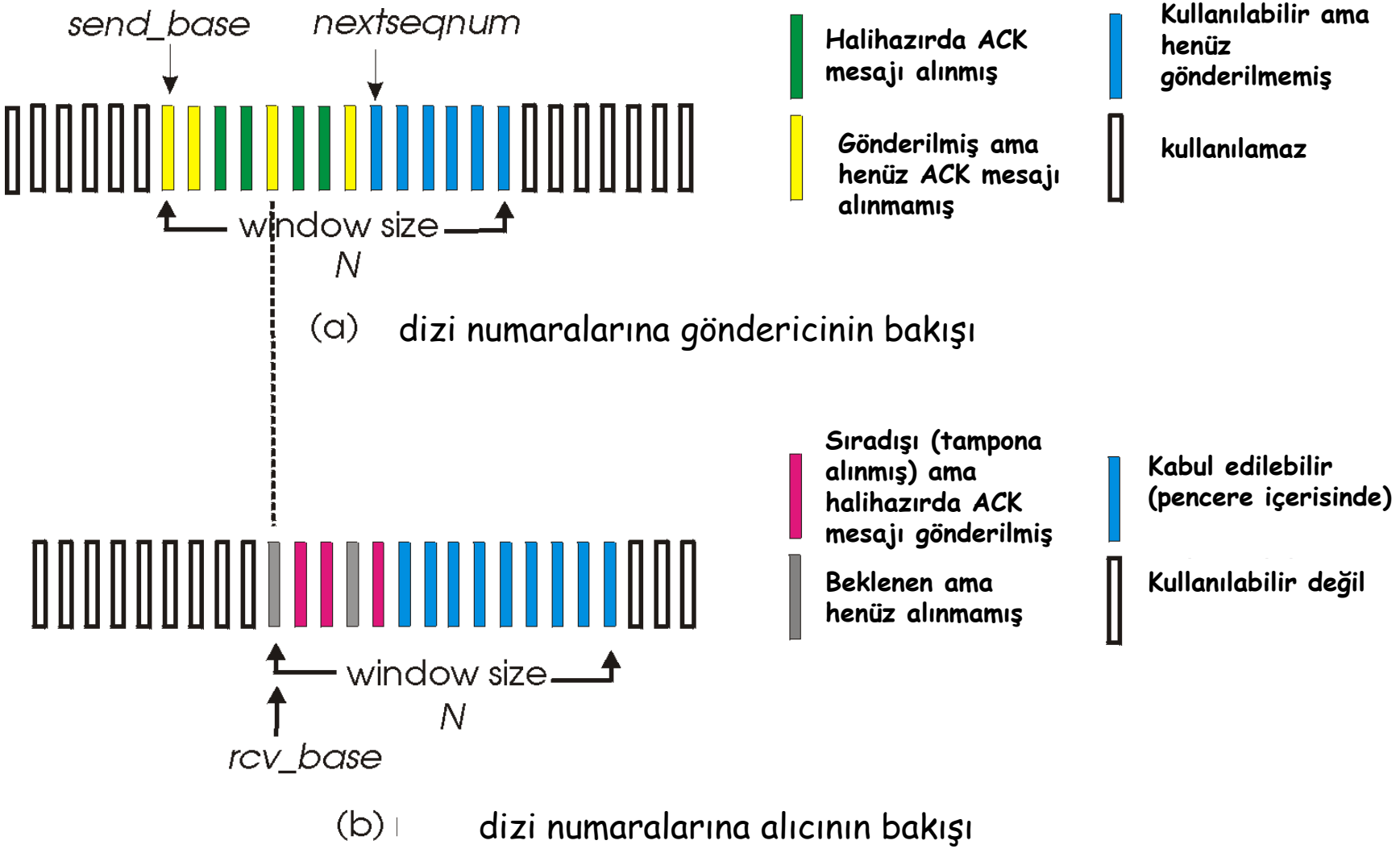
Seçici Tekrarlama (Selective Repeat -SR)

- Her ne kadar GBN başarılı bir mekanizma olsa da sıkıntı çekebileceği durumlar mevcuttur. Özellikle eğer Pencere genişliği ve bant genişliği gecikmesi (bandwidth-delay) büyükse, birçok paket pipeline'da bulunabilir. Burada oluşacak tek bir paket hatası birçok paketin tekrar gönderilmesine sebebiyet verecektir.
- Hatta oluşabilecek hataların olasılığı arttıkça gereksiz yere gönderilen paket sayısı da önemli derecede artacaktır.
- Selective Repeat protokolleri sadece hatalı olan paketlerin gönderilmesi prensibine dayanır. Burada her bir doğru alınan paketin ACK edilmesi gerekecektir. Tekrar N boyunda bir penceremiz olacak, pipeline'da bulunan doğrulanmamış paket sayısını sınırlamak için. GBN'den farklı olarak gönderen penceredeki belli paketlerin ACK'sını almış olacaktır.

Seçici Tekrarlama (Selective Repeat -SR)

- Alıcı doğru olarak alınan herbir paket için ayrı ayrı alındı mesajı (ack) gönderir
  - Gerekirse paketleri bir üst katmana sıralı iletebilmek için tamponda tutar
- Gönderici sadece alındı mesajı (ack) gelmeyen paketleri yeniden gönderir
  - Her alındı mesajı alınmamış paket için gönderici zamanlayıcısı
- Gönderici penceresi (sender window)
  - N ardışık dizin #'sı
  - Yine gönderilmiş ancak alındı bilgisi alınmamış (unACKed) paketlerin dizin #sını sınırlar

Seçici Tekrarlama: gönderici, alıcı pencereleri



Seçici Tekrarlama

gönderici

yukarıdan alınan veriler :

- Eğer penceredeki bir sonraki dizi #sı müsaitse, paketi gönderir

zaman aşımı(n):

- pkt n'i tekrar gönder, zamanlayıcı yeniden başlatır

ACK(n) in [sendbase,sendbase+N]:

- pkt n'i alındı olarak işaretler
- eğer n en küçük alındı bilgisi gelmemiş pkt ise, pencere boyutunu bir sonraki alındı bilgisi gelmemiş dizi # sına ilerletir

alıcı

[rcvbase, rcvbase+N-1] deki pkt n

- n için alındı mesajı ACK(n) gönderir
- Sıralı değilse: tampona alır
- Sıralı ise: iletir (aynı zamanda tapona alınmış sıral pakt leri de ileteir), pencere boyutunu bir sonraki alındı bilgisi gelmemiş pakt in dizi # sına ilerletir

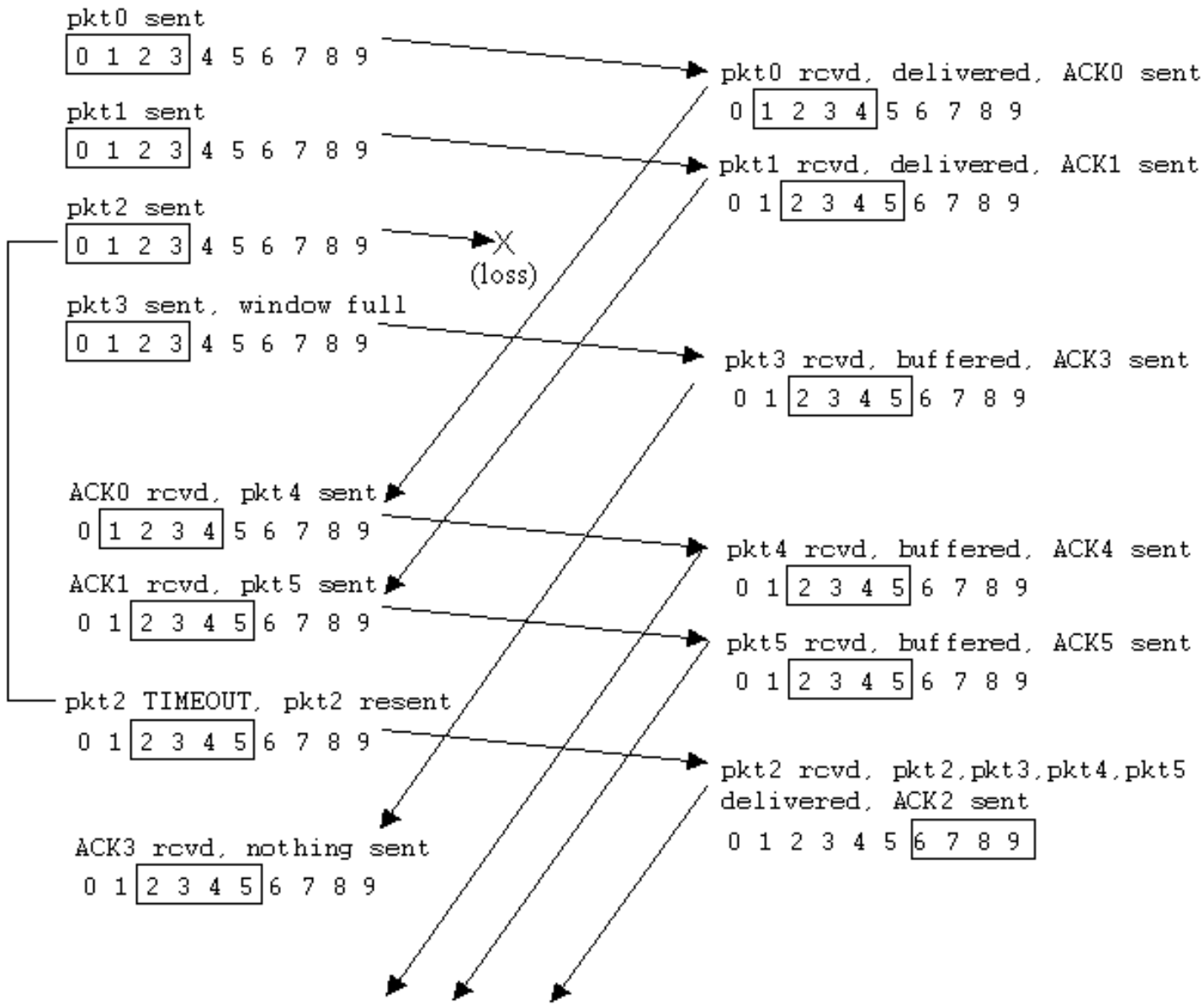
[rcvbase-N,rcvbase-1] deki pkt n

- ACK(n) alındı mesajı üretir

diğer türlü:

- yok say

Seçici Tekrarlama İşlemi



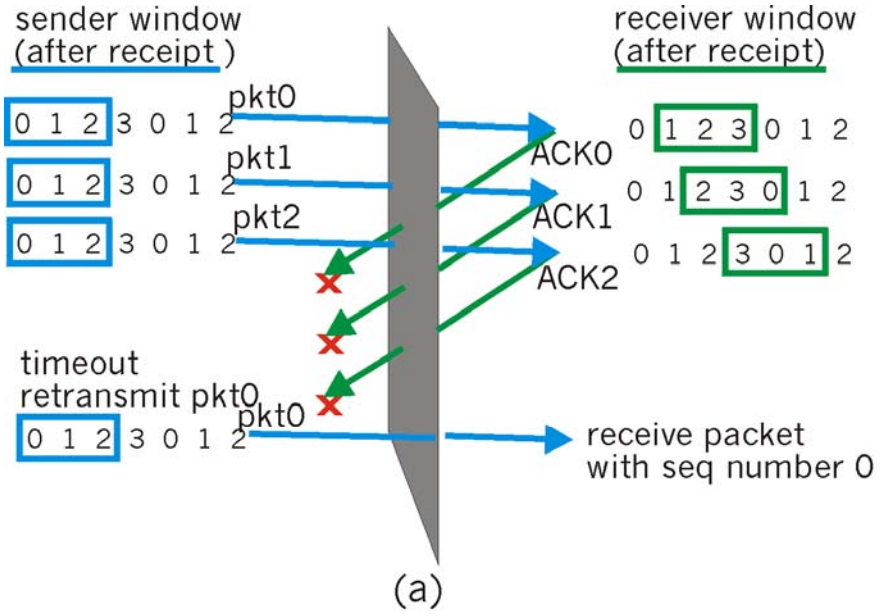
Seçici Tekrarlama: eşleme

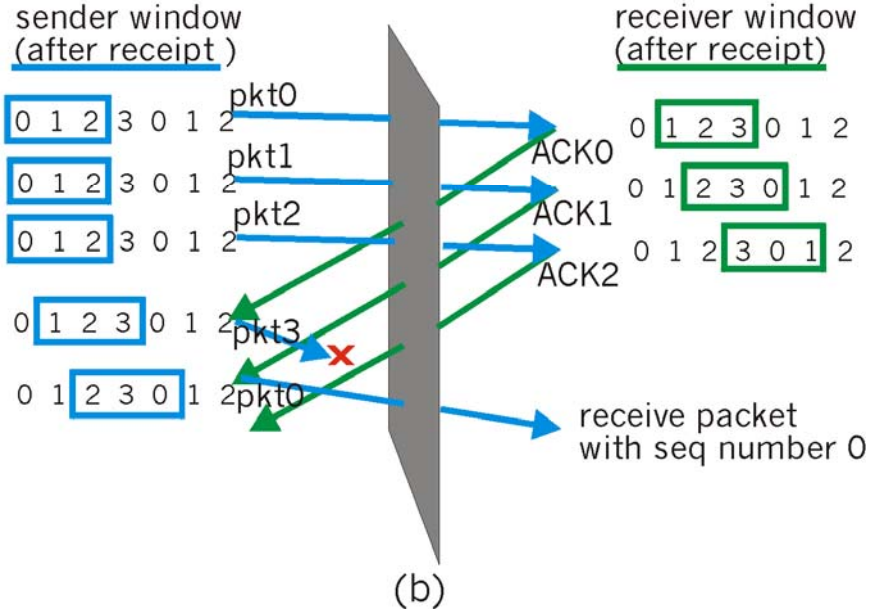
Örnek:

- Dizi # ları: 0, 1, 2, 3
- Pencere boyutu=3

- Alıcı iki senaryodaki farkı anlamaz!
- (a) daki gibi kopya veriyi yeni gibi hatalı olarak geçirir

Q: dizin # ile pencere boyutu arasındaki ilişki nedir?





Güvenilir Veri İletimi Mekanizmaları ve Kullanımları

- Kontrol toplamı: bit hatalarını tespit etmek için kullanılır
- Zamanlayıcı: kanal içerisinde kaybolan paketlerin yeniden iletilmesini sağlamak için kullanılır
- Dizi numarası: kayıp paketlerin tespitini ve kopya paketlerin belirlenmesini sağlar
- Alındı: bir paketin ya da paket setinin doğru olarak alındığı bilgisinin göndericiye bildirilmesi için kullanılır
- Negatif alındı: göndericiye bir paketin doğru alınmadığını söylemek için kullanılır
- Pencere, boru hattı: Belli bir aralık içerisinde bulunan paketlerin gönderilmesini sağlar

SORU

- 3 büyüklüğünde bir gönderici penceresine sahip bir GBN protokolü düşünün . t anında alıcının beklediği, sıradaki paketin k dizi numarasına sahip olduğunu varsayın. Ortamın mesajları yeniden sıralamadığını da varsayın. Aşağıdaki soruları cevaplayın:
- A) t anında göndericinin penceresindeki olası dizi numarası seti hangi aralıklar arasında başlıyor olabilir?
- B) t anında halihazırda göndericiye geri yayılan tüm olası mesajlardaki ACK alanının değeri hangi aralıklar arasında olabilir?

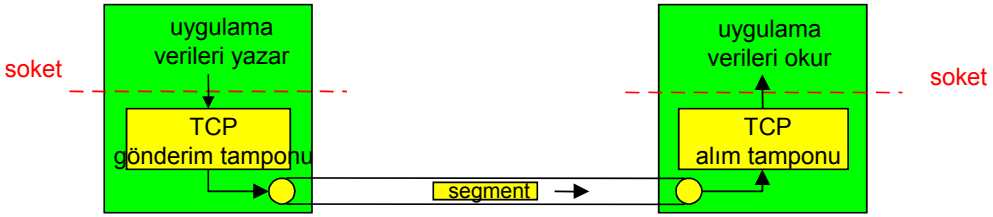
Taşıma Katmanı

- 3.1 Taşıma katmanı servisleri
- 3.2 Çoklama (multiplexing) ve çoklamanın çözülmesi (demultiplexing)
- 3.3 Bağlantısız taşıma: UDP
- 3.4 Güvenilir veri iletiminin prensipleri
- 3.5 Bağlantı yönelimli taşıma: TCP
  - segment yapısı
  - güvenilir veri iletimi
  - akış kontrolü
  - bağlantı yönetimi
- 3.6 Tıkanıklık kontrolü prensipleri
- 3.7 TCP tıkanıklık kontrolü

TCP: Genel Bakış

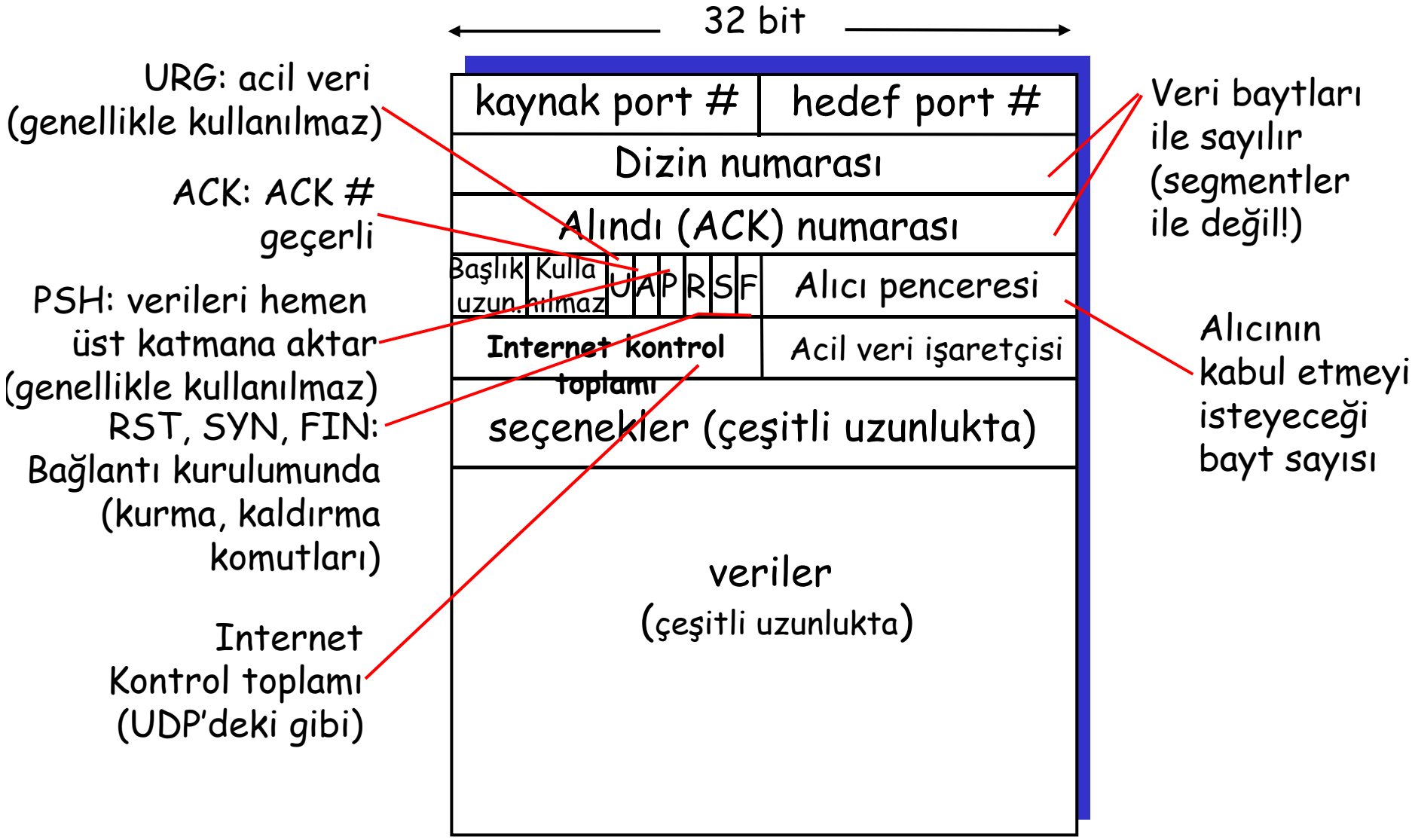
RFCs: 793, 1122, 1323, 2018, 2581

- Noktadan noktaya:
  - Bir gönderici, bir alıcı
- güvenilir, sıralı *bayt akışı:*
  - "mesaj sınırları" yoktur
- Boru hattı:
  - TCP tıkanıklık ve akış kontrolü pencere boyutunu ayarlar
- *gönderme & alma tamponları*



- Tam dubleks veri:
  - Aynı bağlantıda iki yönlü veri akışı
  - MSS: maksimum segment boyutu
- Bağlantı-yönelimli:
  - El sıkışma (kontrol mesajlarının değişimi) ➔ veri değişiminden önce alıcı ve göndericinin durum bilgileri
- Akış kontrolü:
  - Gönderici alıcıyı zorlamaz

TCP segment yapısı



TCP dizin # sı ve ACK ler

Dizin #'ları:

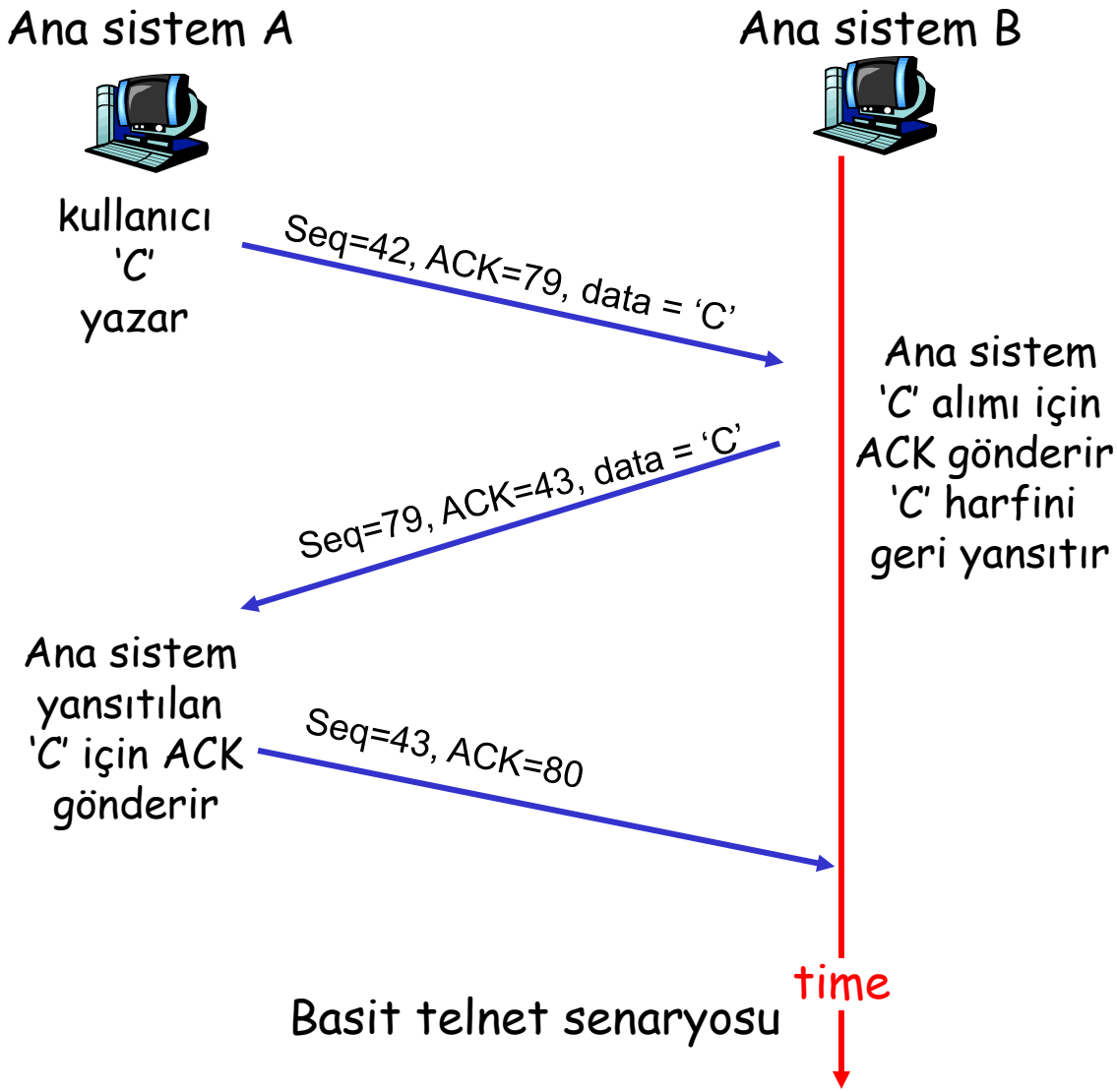
- Segment verisindeki ilk baytın **bayt akış** numarası

ACKler:

- Diğer tarafdan beklenen bir sonraki baytın dizin # sı
- toplu ACK

Soru: alıcı sırasız segmentlerin üstesinden nasıl gelir?

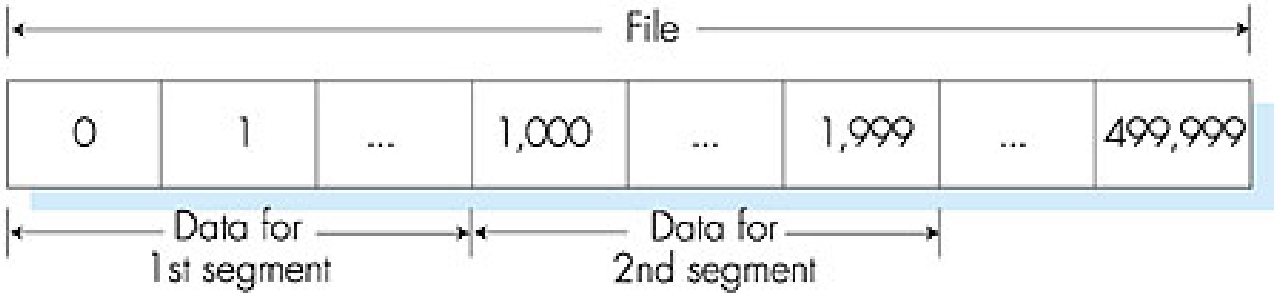
- A: TCP belirtimi bir şey söylemez – uygulayıcıya bağlıdır



Basit telnet senaryosu

TCP dizin # sı ve ACK ler

- Farzedelim ki A'daki bir process B'deki bir process'e bir grup veri göndermek istemekte. A'daki TCP bu veri yığınındaki her bir byte'ı birbirinden farklı olacak şekilde numaralandıracaktır. Veri akışının 500,000byte'lık bir dosyadan oluştuğunu, MSS'in 1,000 byte olduğunu ve veri akışındaki ilk byte'ın 0 olarak numaralandırıldığını farzedelim.
- TCP 500 tane segment oluşturacak. Birinci segment'in Sequence Number'ı 0, ikinci segment'in sequence number'ı 1000, üçüncü segment'in sequence number'ı 2000 vs. olacaktır. Her bir sequence sayısı TCP header'ındaki Sequence Number alanına eklenir.



TCP dizin # sı ve ACK ler

- Her ne kadar burada verilen örnekte ilk byte'ın Sequence number'ını 0'dan başlatmış olsak ta, gerçek kullanımında TCP ilk byte'a rastgele(random) bir sayı atamakta ve bunu alıcıya da iletmektedir.
- Bu şekilde bir random number'ın seçilmesindeki en temel sebep, daha önceden kurulmuş ama şimdi tamamlanmı,ş bu iki host arasındaki eski bir bağlantıdan kalma ve aynı Sequence Number'ı taşıyan bir segment olmasıdır. Bu şekilde rastgele bir sayı seçilerek bu ihtimal azaltılmaya çalışılmaktadır.

TCP dizin # sı ve ACK ler

- ACK Number alanı biraz daha karışık. TCP full-duplex olduğundan A, B'ye veri gönderirken aynı anda B'den veri alabilmektedir. B'den gelen her bir segment'in, B'den A'ya akan veri ile ilintili olarak bir Sequence Number'ı bulunmaktadır. A'nın Acknowledgement alanına koyduğu sayı, A'nın B'den beklediği bir sonraki byte'ın sequence number'ıdır.
- Farzedelim ki A B'den 0'dan 535'e kadar olan tüm byte'ları aldı ve B'ye bir segment göndermek üzere. Başka bir deyişle A şu aşamada B'den 536. byte'ı ve devamını beklemekte. Bu durumda A göndereceği segment'in Acknowledgement number alanına 536 yazıp B'ye gönderir.

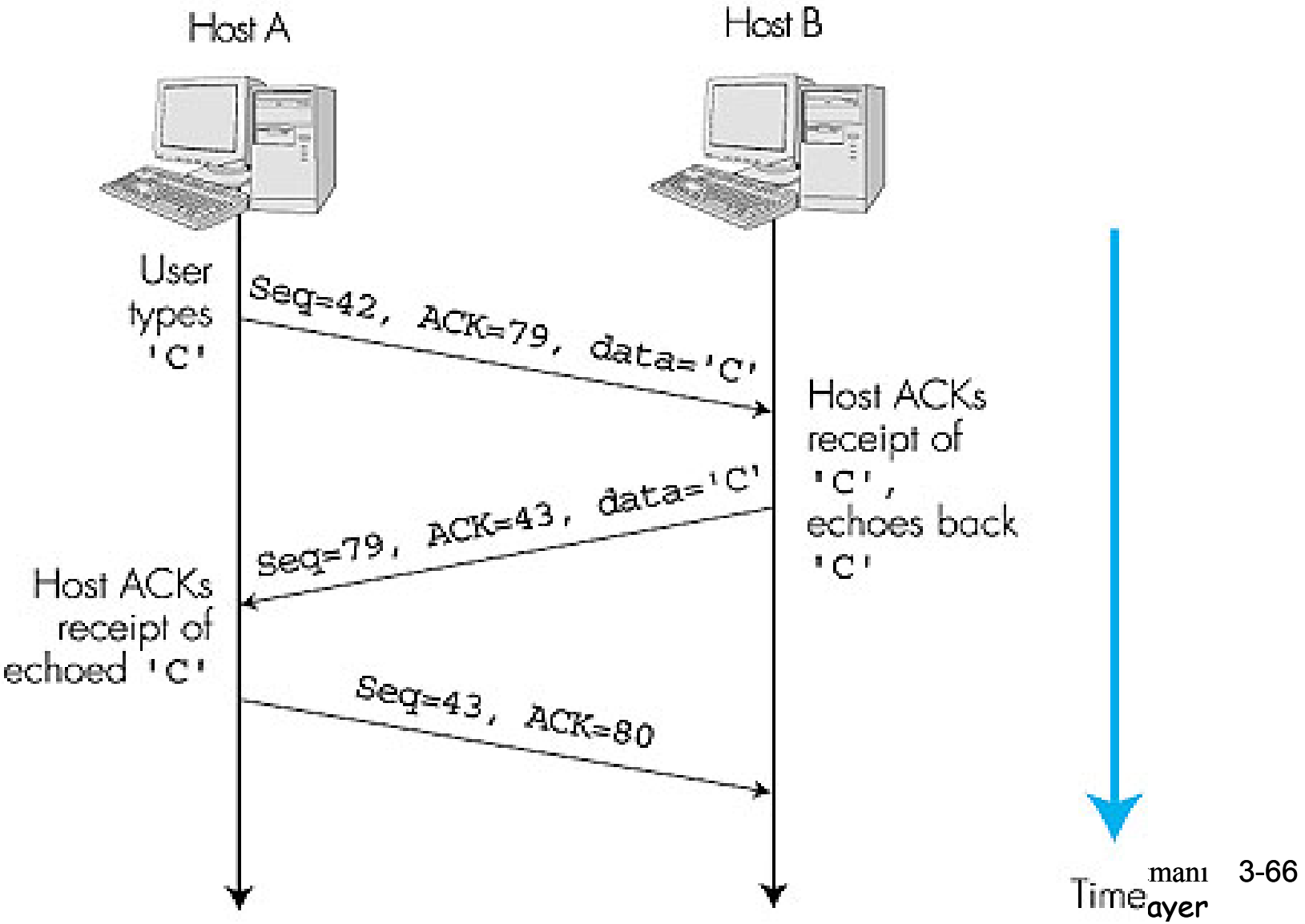
TCP dizin # sı ve ACK ler

- Başka bir durumda da A B'den 0'dan 535'e ve 900'den 1000'e kadar olan byte'ları aldığını, fakat herhangi bir sebepten dolayı 536'dan 899'a kadar olan byte'ları almadığını farzedelim. Bu yüzden A'nın bir sonraki göndereceği segment'te Acknowledgement alanına 536 yazacaktır. TCP, ilk kayıp byte'a kadar olan kısmı doğruladığı için TCP'nin kümülatif doğrulama yaptığı söylenir.
- Fakat burada başka önemli bir nokta ortaya çıkmakta: A, üçüncü segment'i yani 900'den 1000'e kadar olan byte'ları ikinci segment'ten önce aldı, yani üçüncü segment sıralama dışı ulaştı.
- Peki sırası bozulmuş segment'ler aldığı alıcı taraf ne yapmalı? Burada bir temel kural olmamasına rağmen iki temel yaklaşım kullanılmaktadır:
  - Sıralamayı bozan byte'ları anında atmak
  - Sıralamayı bozan byte'ları önbellekte tutup eksik byte'ları beklemek

Örnek: TELNET

- Farzedelim ki A, B ile bir telnet bağlantısı kurmakta. A bu durumda istemci (client), B'de sunucu (server) olmaktadır. İstemci tarafından istemcideki uygulamada her yazılan karakter sunucuya gönderilecek, sunucu da ekrana yazdırmak için aynı karakteri istemciye gönderecektir. Bu karakter kullanıcının Telnet arayüzünde gözükecek ama aslında aynı karakter iki host arasında gidip gelmiş olacak TCP vasıtasıyla (traverse twice the network).
- Kullanıcının sadece 'c' harfini bastığını farzedelim ve gönderilen ve alınan TCP segmentlerine kısaca bir bakalım.

Örnek: TELNET



Örnek: TELNET

- İstemci ve sunucu için başlangıç sequence number'larının 42 ve 79 olduğunu farzedelim.
- İstemciden ilk gönderilen segment'in sequence number'ı 42, sunucudan ilk gönderilen segment'in sequence number'ı 79 olacaktır.
- Her hangi bir veri gönderilmeden önce ama bağlantı kurulduktan sonra istemci sunucudan 79 sequence number'lı segment'i, sunucu da 42 sequence number'lı segment'i beklemektedir.
- Örnekte de gösterildiği üzere 3 segment gönderilmektedir. İlk segment istemciden sunucuya gönderilmiş, içinde tek byte'lık C bulunmakta ve sequence number'ı 42. Sunucudan da herhangi bir veri almadığı için de acknowledgement number kısmında 79 bulunmakta.

Örnek: TELNET

- İkinci segment sunucudan istemciye gönderilmekte ve çift amaca hizmet etmekte:
  - İlk önce almış olduğu veriyi doğrulamakta: Acknowledgement kısmına 43 koyarak, sunucu istemciye 42. byte'a kadar herşeyi aldığını ve şimdi 43'ü beklediğini ifade etmekte.
  - İkinci amacı bu segment'in 'C' harfini ekrana yazdırmak, dolayısıyla C harfini tekrar göndermekte.
- Bu ikinci segment'in sequence number'ı 79, sunucunun göndereceği byte'lara verdiği ilk değerdir.
- Dikkat ederseniz istemciden sunucuya giden verinin Acknowledgement'i sunucudan istemciye giden veriyi taşıyan segment'in içinde bulunmaktadır. Buna piggybacking (sırtta taşıma) adı verilmektedir.

Örnek: TELNET

- Üçüncü segment istemciden sunucuya gönderilmektedir: Tek amacı sunucudan almış olduğu veriyi doğrulamaktır. Bu segment'in taşıdığı veri alanı boş olacak, ve acknowledgement alanında taşıdığı değer 80 olacaktır. Bunun sebebi 80'e kadar olan tüm verileri almış olması ve şimdi 80 ve sonrasını beklemesidir.
- İşin ilginç tarafı herhangi bir veri taşımamasına rağmen bir sequence number'ının bulunmasıdır. Bunun sebebi, bu şekilde tanımlanmış bir zorunlu alanın bulunması ve dolayısıyla TCP'nin bu alanı bir sonraki sequence number'la doldurmasıdır.

Taşıma Katmanı

- 3.1 Taşıma katmanı servisleri
- 3.2 Çoklama (multiplexing) ve çoklamanın çözülmesi (demultiplexing)
- 3.3 Bağlantısız taşıma: UDP
- 3.4 Güvenilir veri iletiminin prensipleri
- 3.5 Bağlantı yönelimli taşıma: TCP
  - segment yapısı
  - güvenilir veri iletimi
  - akış kontrolü
  - bağlantı yönetimi
- 3.6 Tıkanıklık kontrolü prensipleri
- 3.7 TCP tıkanıklık kontrolü

TCP Gidiş Dönüş Süresi ve Zaman aşımı

Q: TCP zaman aşımı (timeout) değeri nasıl belirlenir?

- RTT'den daha uzun
  - RTT ler farklı farklıdır
- Çok kısa: prematüre zaman aşımı (premature timeout)
  - Gereksiz yeniden iletimler
- Çok uzun : segment kaybına yavaş reaksiyon verme

Q: RTT nasıl tahmin edilebilir?

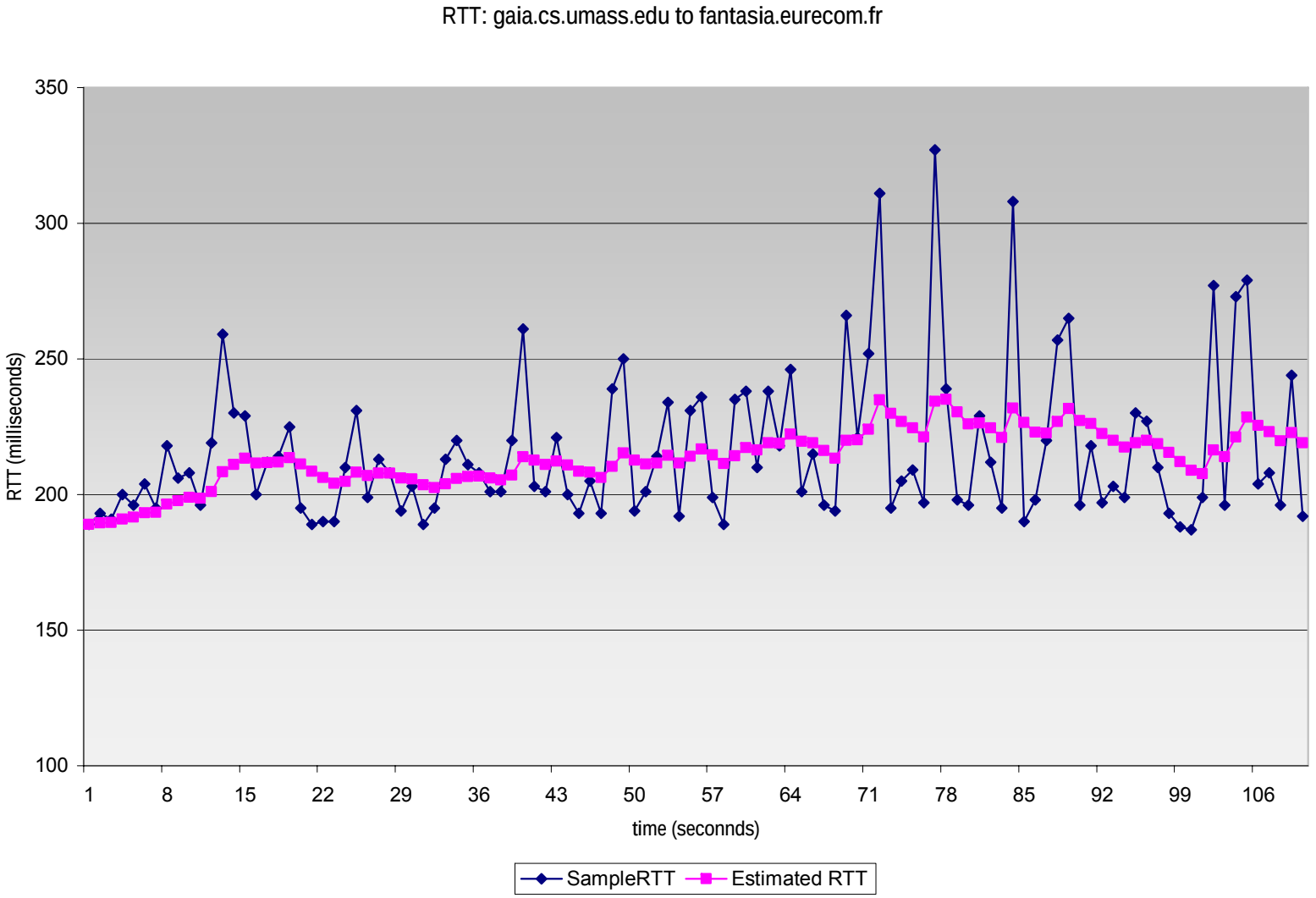
- **SampleRTT**: Segmentin iletiminden ACK alındısına kadar geçen süre
  - Yeniden iletimleri yok say
- **SampleRTT** farklı farklı olabilir, want estimated RTT "smoother"
  - Sadece şu andaki SampleRTT yerine yakın zamandaki birkaç ölçümün ortalaması alınır

TCP Gidiş Dönüş Süresi ve Zaman aşımı

EstimatedRTT = (1- α)*EstimatedRTT + α*SampleRTT

- Üssel ağırlıklı hareket eden ortalama (Exponential weighted moving average – EWMA)
- Ağırlıklı ortalama, en son örneklere eski bir örnekten daha çok ağırlık katar
- Tipik değer: $\alpha = 0.125$

Örnek RTT tahmini:



Gidiş Dönüş Süresi ve Zaman aşımı

Zaman aşımını (timeout) ayarlamak

- **EstimtedRTT** artı "güvenlik marjı" ("safety margin")
  - **EstimatedRTT'de geniş değişiklikler varsa ->** daha geniş güvenlik marjı (safety margin)
- SampleRTT'nin EstimatedRTT'den ne kadar değişiklik gösterdiğinin ilk tahmini aşağıdaki şekilde hesaplanır :

DevRTT = (1-β)*DevRTT + β*|SampleRTT-EstimatedRTT|

(typically, β = 0.25)

Zaman aşımı (timeout) aralığı

Zaman aşımı (TimeoutInterval) = EstimatedRTT + 4*DevRTT

olarak ayarlanır

Taşıma Katmanı

- 3.1 Taşıma katmanı servisleri
- 3.2 Çoklama (multiplexing) ve çoklamanın çözülmesi (demultiplexing)
- 3.3 Bağlantısız taşıma: UDP
- 3.4 Güvenilir veri iletiminin prensipleri
- 3.5 Bağlantı yönelimli taşıma: TCP
  - segment yapısı
  - güvenilir veri iletimi
  - akış kontrolü
  - bağlantı yönetimi
- 3.6 Tıkanıklık kontrolü prensipleri
- 3.7 TCP tıkanıklık kontrolü

Güvenilir Veri İletimi

- TCP IP'nin güvenilmeyen servisi üzerinde rdt (güvenilir veri iletim) servisi oluşturur
- Pipelined segments
- Kümülatif acks
- TCP bir tek yeniden iletim zamanlaması kullanır

- Yeniden iletimler:
  - Zaman aşımı (timeout)
  - Kopya alındı (duplicate acks)

  gerçekleşmesinde uygulanır
- Öncelikle basit bir TCP göndericisi düşünelim:
  - Kopya alındıları (acks) yok sayalım
  - Akış kontrolü (flow control) ve tıkanıklık kontrolünü (congestion control) yok sayalım,

TCP göndericisi olayları:

Uygulamadan alınan veri:

- dizi # (seq #) ile segment oluşturulur
- dizi # (seq #) segmentteki ilk veri baytının bayt-akis (byte-stream) numarasıdır
- Eğer çalışmıyorsa zamanlayıcıyı başlatır (zamanlayıcıyı en eski alındı mesajı alınmamış segment olarak düşün)
- expiration interval: `TimeOutInterval`

Zaman aşımı (timeout):

- Zaman aşımına yol açan segmenti yeniden ilet
- Zamanlayıcıyı yeniden başlat

ACK (alındı bilgisi) alımı:

- Eğer daha önce alındısı alınmamış (unacked) segment için alındı (ack)alıyorsa
  - Ack lenenleri günceller
  - Hala ack lenmemiş segment varsa zamanlayıcı yeniden başlatılır.

TCP göndericisi (basit)

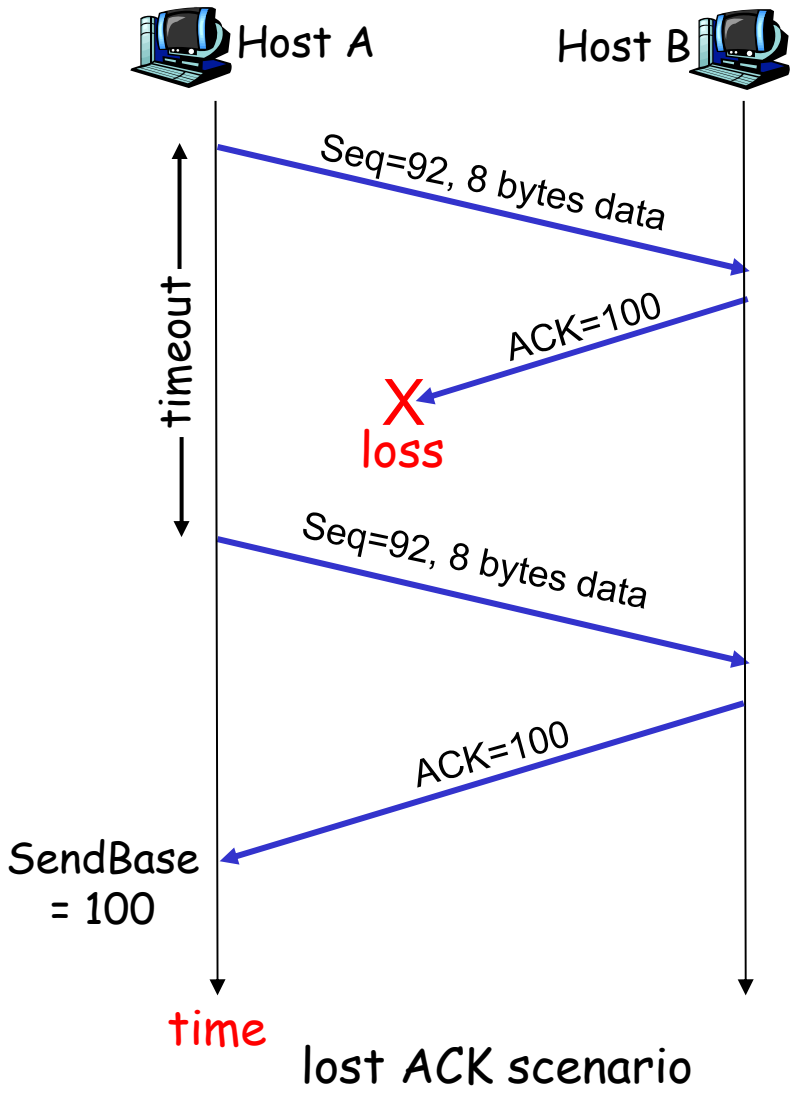
```
NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum

loop (forever) {
  switch(event)

  event: Yukarıdaki uygulamadan alınan veriler
      Dizi numarası NextSeqNum olan TCp segmenti oluştur
      If (zamanlayıcı çalışmıyorsa)
          zamanlayıcıyı başlat
      segmenti IP'ye ilet
      NextSeqNum = NextSeqNum + length(data)

  event: zaman aşımı
      henüz alındı bilgisi alınmamış en düşük
                dizi numarasına sahip segmenti
                yeniden ilet
      start timer

  event: ACK değeri y olan bir ACK alındığında
        if (y > SendBase) {
          SendBase = y
          if (henüz alındı bilgisi alınmamış segmentler varsa)
                zamanlayıcıyı başlat
          }

} /* end of loop forever */
```

Comment:
- SendBase-1: en son kümülatif olarak alındı bilgisi gelmiş olan bayt

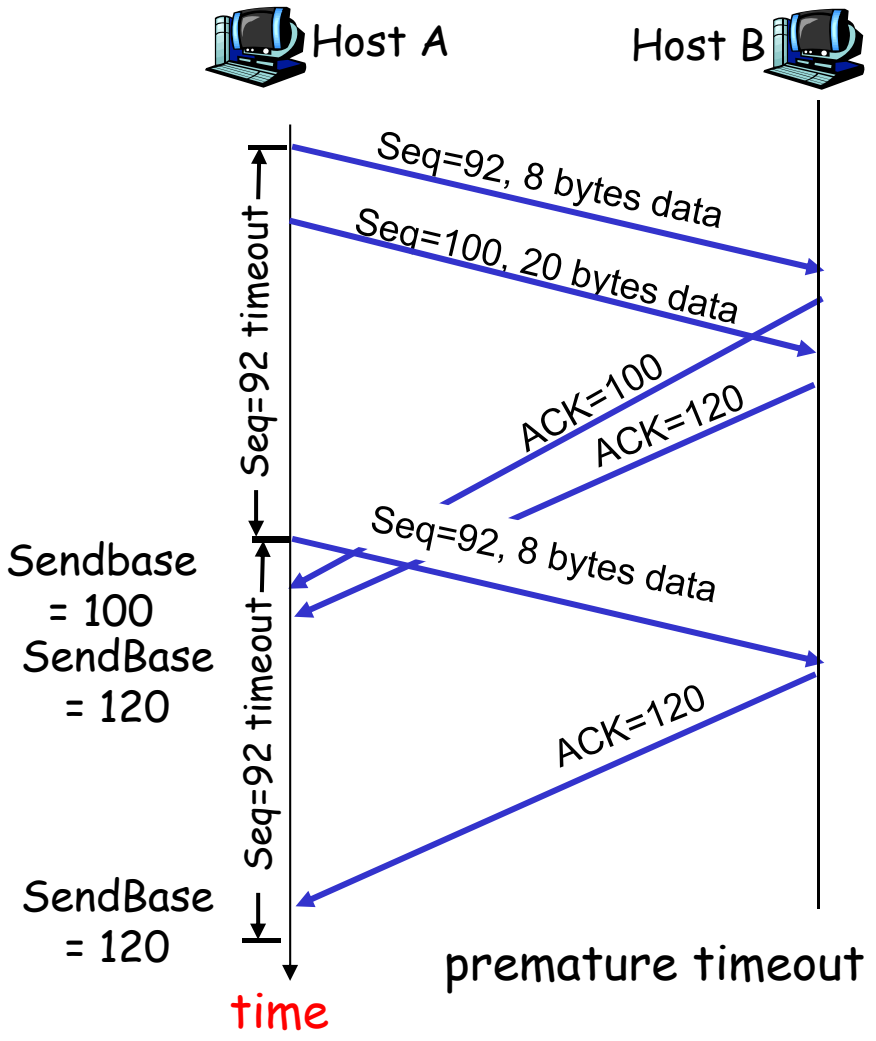
Örnek:
- SendBase-1 = 71;

y= 73, ise alıcı 73 ve sonrasını beklemektedir;
y > SendBase, ise yeni veri ACK'lenmiştir

TCP: yeniden iletim senaryoları

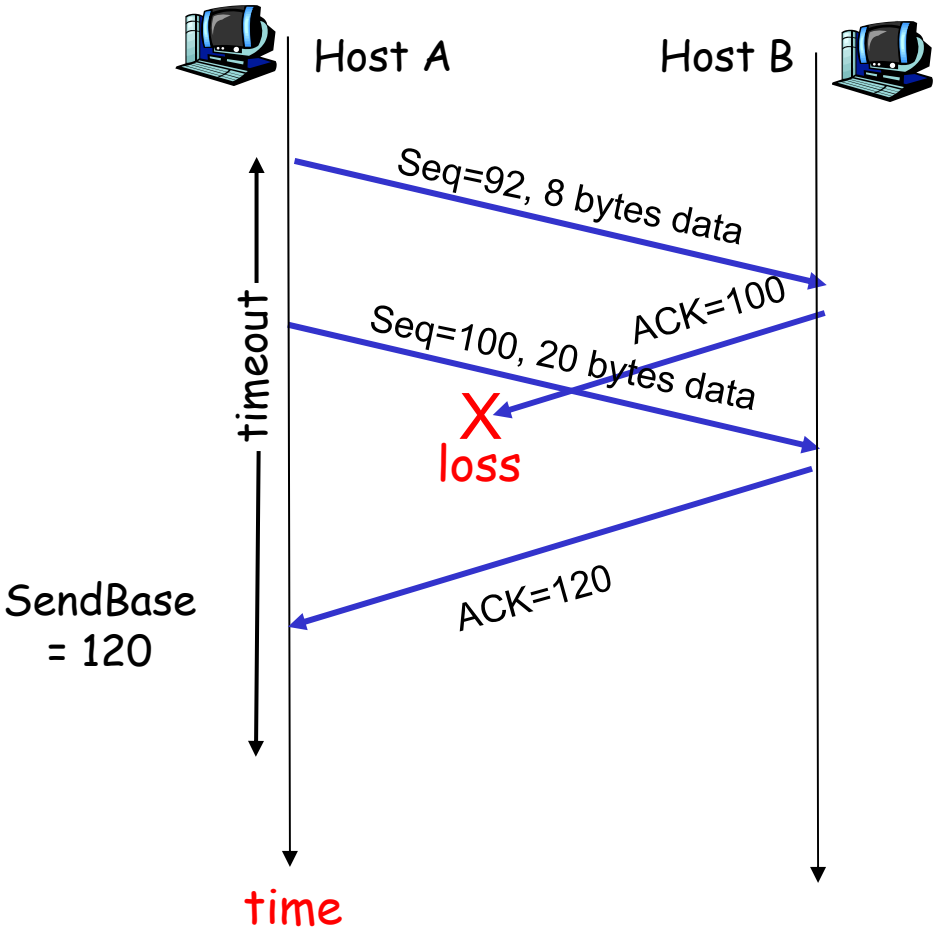


lost ACK scenario



premature timeout

TCP yeniden iletim senaryoları



Cumulative ACK scenario

TCP ACK üretimleri [RFC 1122, RFC 2581]

Olaylar	TCP Alıcısı
Beklenen dizi #'na sahip sıralı segmentin varışı. Beklenen dizi #'na sahip tüm veri ACK'lenmiştir	Geciktirilmiş ACK. Diğer sıralı segment için 500msn bekler. Bir sonraki segment gelmezse, ACK gönderir.
Beklenen dizi #'na sahip sıralı segmentin varışı. ACK iletimi için bekleyen bir önceki segment varsa	Toplu ACK ile sıralı iki segment için alındı gönderilir
Beklenenden daha büyük dizi #'lı sıradışı bir segmentin varışı. Boşluk tespit edilmiştir.	Bir sonraki beklenen byte'ın dizi # belirterek kopya ACK gönderir.
Alınan verilerdeki boşluğu kısmen veya tamamen dolduran segmentin varışı	Segmentin boşluğun küçük kenarından başladığını sunarak hemen ACK gönderir.

Hızlı Yeniden İletim (Fast Retransmit)

- Zaman aşımı periyodu göreceli olarak uzun olabilir:
  - Kayıp paketi yeniden göndermek için uzun bekleme
- Kayıp segmentleri kopya ACK'ler ile tespit etme.
  - Gönderici sıklıkla pek çok segmenti arka arkaya gönderir
  - Eğer segment kayıp ise, pek çok kopya ACK olacaktır.
- Eğer gönderici aynı data için 3 ACK alırsa, ACK'lenen segmentden sonra gelen segmentin kaybolduğunu varsayar:
  - Hızlı yeniden iletim(fast retransmit): segmenti zamanlayıcının süresi dolmadan yeniden gönderir.

Hızlı yeniden iletim algoritması:

```
event: ACK received, with ACK field value of y
            if (y > SendBase) {
                SendBase = y
                if (there are currently not-yet-acknowledged segments)
                    start timer
              }
            else {
                increment count of dup ACKs received for y
                if (count of dup ACKs received for y = 3) {
                    resend segment with sequence number y
                }
```

Daha önce ACK'lenmiş bir segment için kopya ACK

Hızlı yeniden iletim (fast retransmit)

Zaman aşımı aralığını çiftlemek:

- Bir zaman aşımı sonrasında zaman aşımı aralığının uzunluğu:
  - Bir zaman aşımı meydana geldiğinde TCP, alındı mesajı gönderilmemiş olan en küçük dizi numarasına sahip segmenti yeniden iletir.
  - Ancak bir sonraki zaman aşımı aralığını son EstimatedRTT ve DevRTT değerlerinden türetmek yerine, bir önceki değeri iki ile çarpar.
    - Örn. İlk kez zaman aşımı meydana geldiğinde değer 0,75 ise bir sonrakinde unu 1,5 olarak ayarlar.
  - Bu bize bir anlamda tıkanıklık kontrolü de sağlamış olur

TCP GBN mi SR midir?

- GBN:
  - Alındı mesajları topludur
  - Sıradış segmentler için ACK gönderilmez
- SR:
  - Doğru ancak sıradışı paketleri tampona alır
  - Seçici alındı, seçici yeniden iletim yapar

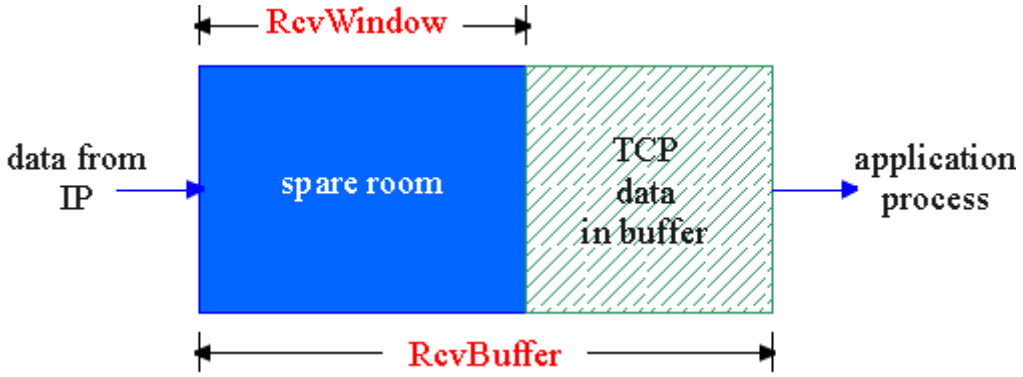
    - GBN ve SR nin melezidir diyebiliriz!

Taşıma Katmanı

- 3.1 Taşıma katmanı servisleri
- 3.2 Çoklama (multiplexing) ve çoklamanın çözülmesi (demultiplexing)
- 3.3 Bağlantısız taşıma: UDP
- 3.4 Güvenilir veri iletiminin prensipleri
- 3.5 Bağlantı yönelimli taşıma: TCP
  - segment yapısı
  - güvenilir veri iletimi
  - akış kontrolü
  - bağlantı yönetimi
- 3.6 Tıkanıklık kontrolü prensipleri
- 3.7 TCP tıkanıklık kontrolü

TCP Akış Kontrolü

- TCP bağlantısının alcı tarafında alıcı tamponu (receive buffer) vardır



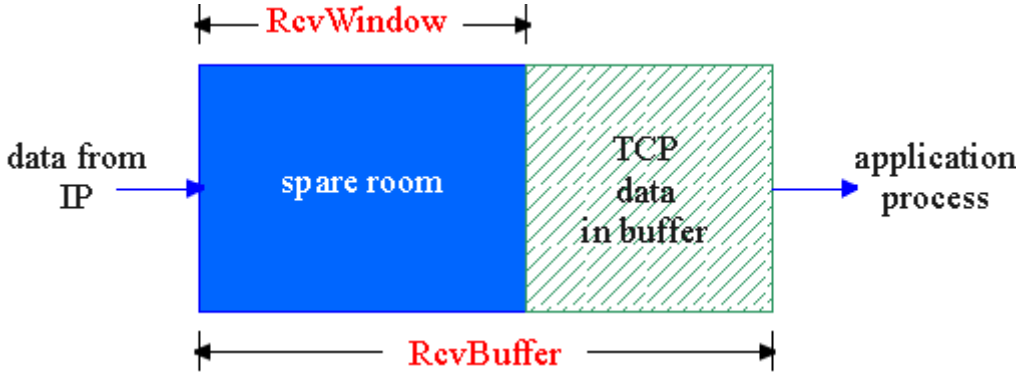
- Uygulama süreci verileri bu tampondan okumada yavaş olabilir

Akış kontrolü

Göndericinin alıcının tamponunu çok fazla ve çok hızlı göndererek taşırmasını engeller

- Hız eşleştirme servisi: göndericinin gönderdiği hızı, alıcının okuduğu hızla eşleştirir.

TCP akış kontrolü: nasıl çalışır?



(TCP alıcısının, sıralama dışı segmentleri attığını varsayarak başlayalım)

- kullanılabilir boş tampon

= **RcvWindow**

= **RcvBuffer-[LastByteRcvd - LastByteRead]**

- Alıcı ne kadar boş alan olduğunu segmentteki **RcvWindow** değeri ile bildirir.
- Gönderici ACK'lenmemiş verileri **RcvWindow** değeri ile sınırlar
  - Alıcı tamponunun taşmasını engeller

Taşıma Katmanı

- 3.1 Taşıma katmanı servisleri
- 3.2 Çoklama (multiplexing) ve çoklamanın çözülmesi (demultiplexing)
- 3.3 Bağlantısız taşıma: UDP
- 3.4 Güvenilir veri iletiminin prensipleri
- 3.5 Bağlantı yönelimli taşıma: TCP
  - segment yapısı
  - güvenilir veri iletimi
  - akış kontrolü
  - bağlantı yönetimi
- 3.6 Tıkanıklık kontrolü prensipleri
- 3.7 TCP tıkanıklık kontrolü

TCP Bağlantı Yönetimi

Hatırla: TCP göndericisi ve alıcısı veri segmentlerini görndermeden önce "bağlantı" kurarlar

- TCP değişkenleri belirlenir:
  - Dizi numaraları
  - tamponlar, akış kontrol bilgileri (örn. **RcvWindow**)
- *istemci:* bağlantı başlatıcı

```
Socket clientSocket = new
Socket("hostname","port
number");
```

- *sunucu:* istemci tarafından iletişime geçilen

```
Socket connectionSocket =
welcomeSocket.accept();
```

3 yönlü el sıkışma:

Adım 1: İstemci TCP SYN segmentini sunucuya gönderir
- İlk dizi # nı belirler
- Uygulama katmanı verisi bulunmaz

Adım 2: Sunucu SYN alır, SYNACK ile cevap verir
- Sunucu tamponları ayırır
- Sunucu için ilk dizi # nı belirler

Adım 3: İstemci SYNACK alır, ACK ile cevap verir, veri içerebilir

TCP Bağlantı Yönetimi (devam.)

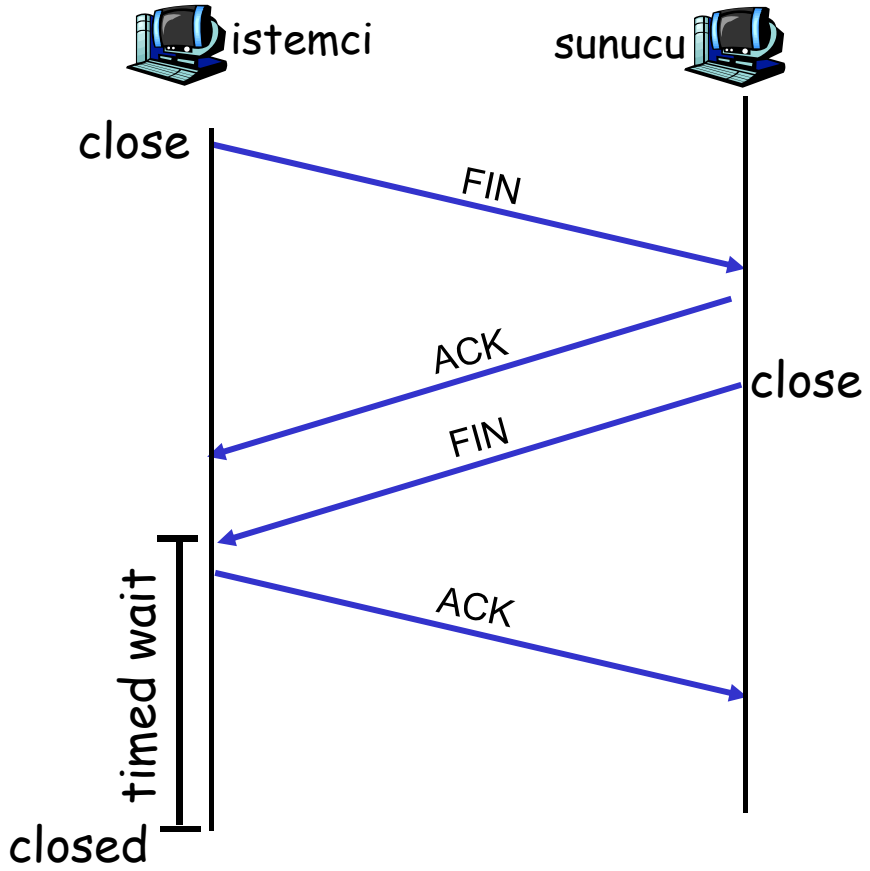
Bağlantıyı kapama:

client closes socket:

```
clientSocket.close();
```

Adım 1: İstemci sunucuya TCP FIN kontrol segmenti gönderir

Adım 2: Sunucu FIN alır, ACK ile cevap verir. Bağlantıyı keser, FIN gönderir.

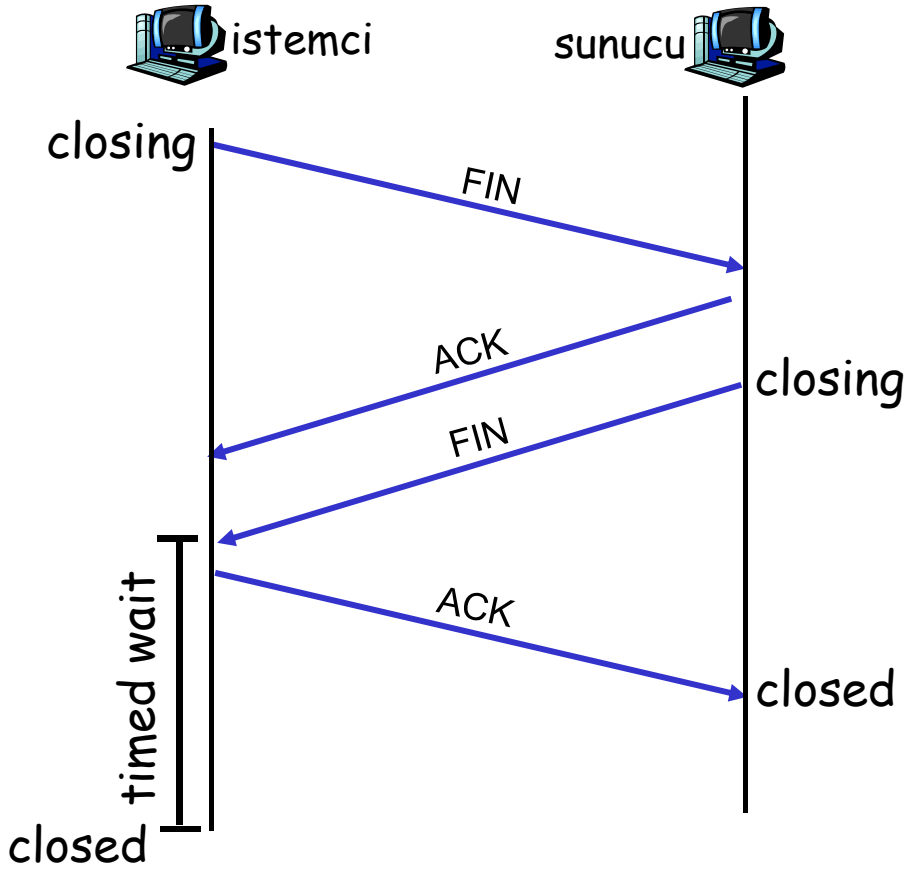


TCP Bağlantı Yönetimi (devam)

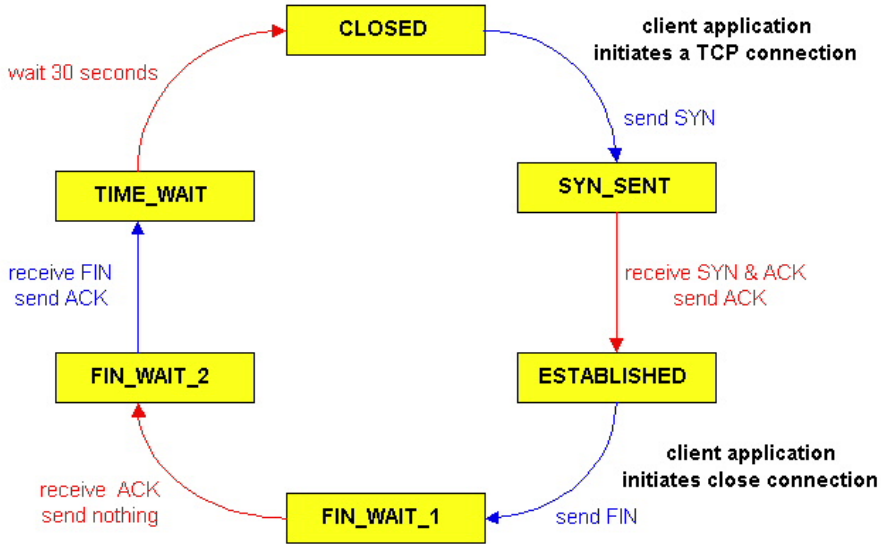
Adım 3: İstemci FIN alır, ACK ile cevaplar.

- "timed wait" değeri girer – Alındandan FINlere ACK ile cevap verir

Adım 4: Sunucu, ACK alır. Bağlantı kapanır.

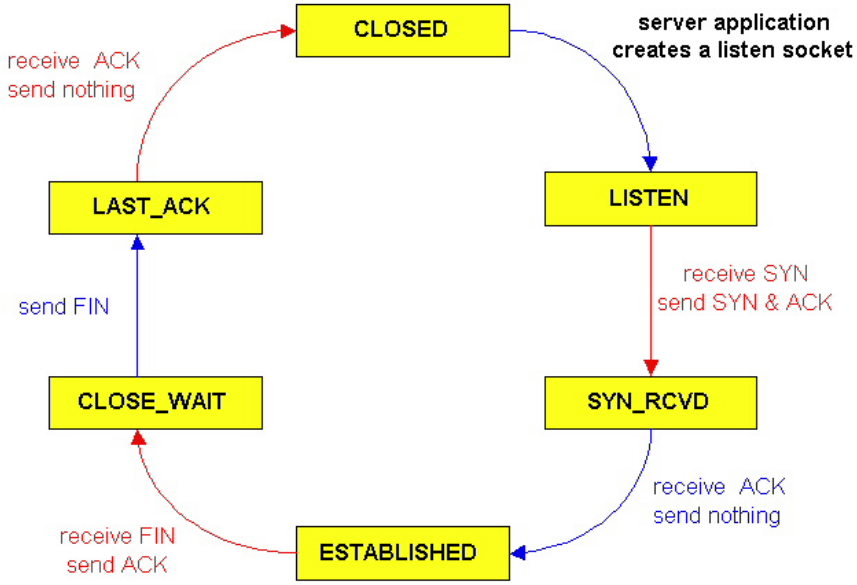


TCP Bağlantı Yönetimi (devam)



TCP istemci döngüsü

TCP sunucu döngüsü



Taşıma Katmanı

- 3.1 Taşıma katmanı servisleri
- 3.2 Çoklama (multiplexing) ve çoklamanın çözülmesi (demultiplexing)
- 3.3 Bağlantısız taşıma: UDP
- 3.4 Güvenilir veri iletiminin prensipleri
- 3.5 Bağlantı yönelimli taşıma: TCP
  - segment yapısı
  - güvenilir veri iletimi
  - akış kontrolü
  - bağlantı yönetimi
- 3.6 Tıkanıklık kontrolü prensipleri
- 3.7 TCP tıkanıklık kontrolü

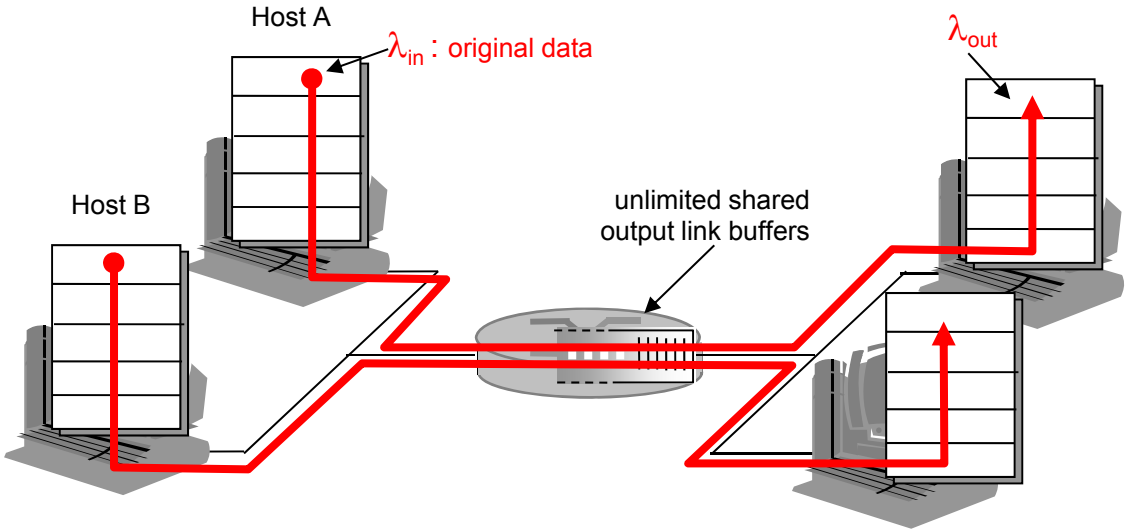
Tıkanıklık Kontrolü Prensipleri

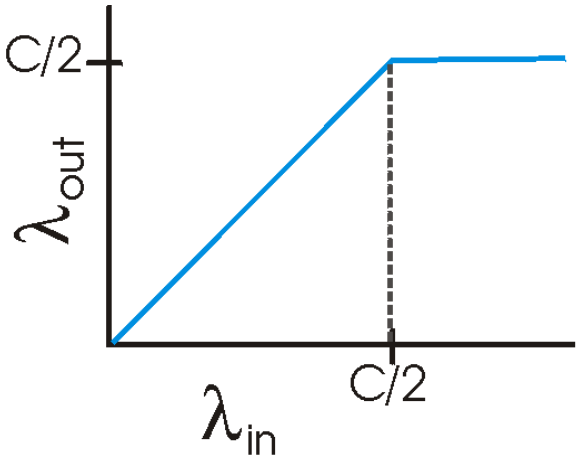
Tıkanıklık:

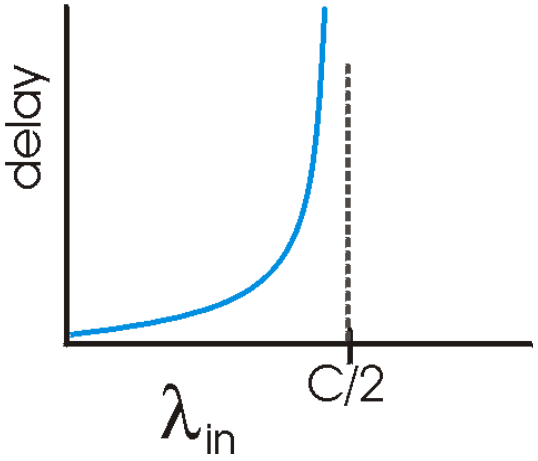
- "çok fazla kaynağın *ağın* kaldırabileceğinden çok fazlasını çok hızlı göndermesinden" kaynaklanır
- Akış kontrolünden farklıdır!
- belirtileri:
  - Kayıp paketler (yönlendiricilerdeki tamponlarda taşma)
  - Uzun gecikmeler (yönlendirici tamponlarında kuyruklar)
- İlk 10 ağ problemi arasındadır!

Tıkanıklığın nedenleri/maliyeti: Senaryo 1

- İki gönderici, iki alıcı
- Sonsuz tamponu olan bir yönlendirici
- Yeniden iletim yok



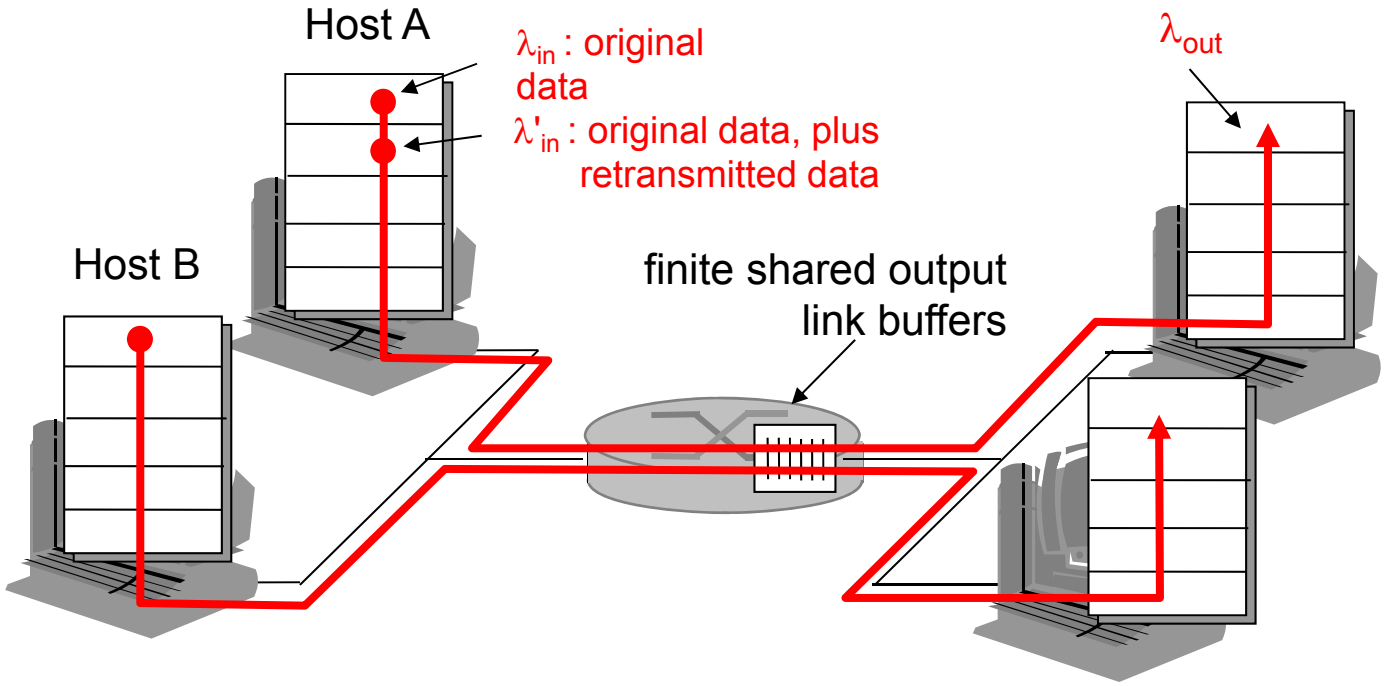




- Tıkanıklık olduğunda uzun beklemeler

Tıkanıklığın nedenleri/maliyeti: Senaryo 2

- *sonlu* tampona sahip bir yönlendirici
- Gönderici kaybolan paketleri yeniden iletir



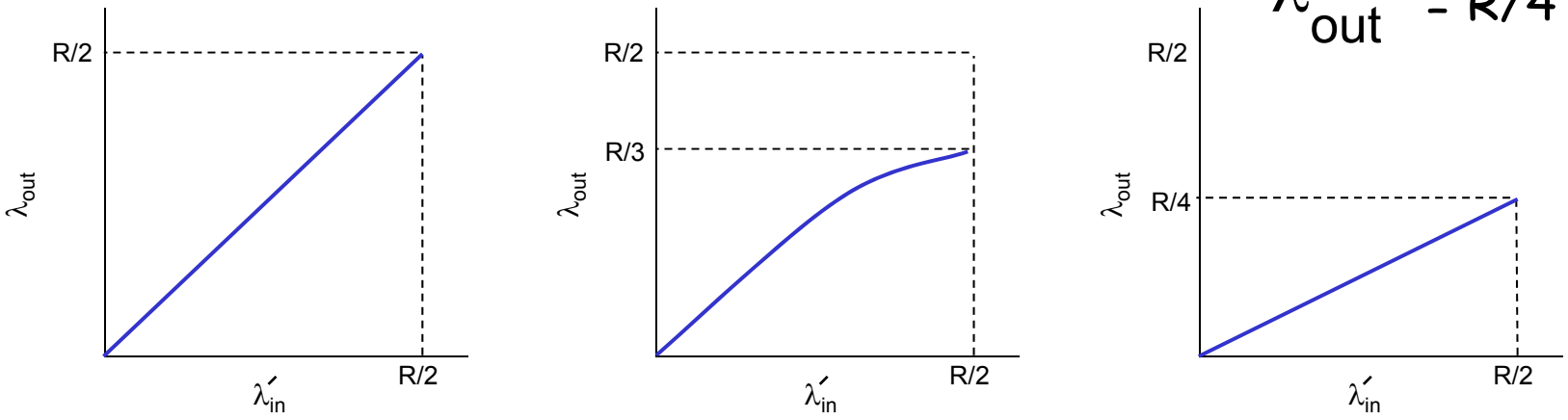
Tıkanıklığın nedenleri/maliyeti: Senaryo 2

- Tampon boş olduğunda paket gönderdiği bir durum varsayarsak

$$\lambda_{in} = \lambda_{out} = R/2$$

- Sadece bir paketin kesin olarak kayıp olduğu bilinirse: $\lambda'_{out} = R/3$
- Uzun gecikme nedenli yeniden iletimler, yönlendiricilerin hat bant genişliğini gereksiz yere kullanımına yol açar

$\lambda_{out} = R/4$



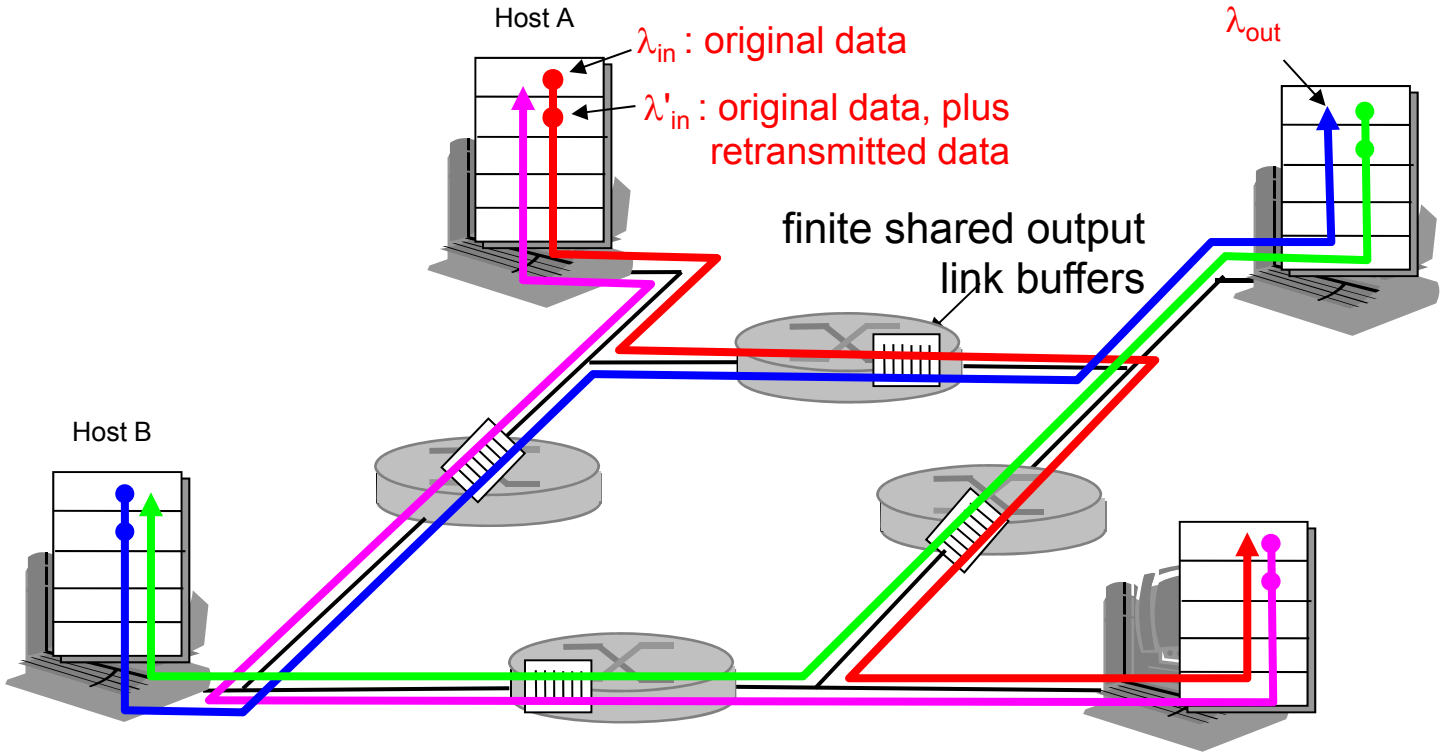
Tıkanıklığın "maliyeti" :

- Geciken bir paketin yeniden gönderimi durumlarında, orijinal paketin yeniden iletim işi bir kayıptır
- Gereksiz yeniden iletimler: hatta aynı paketin pek çok kopyası olacaktır

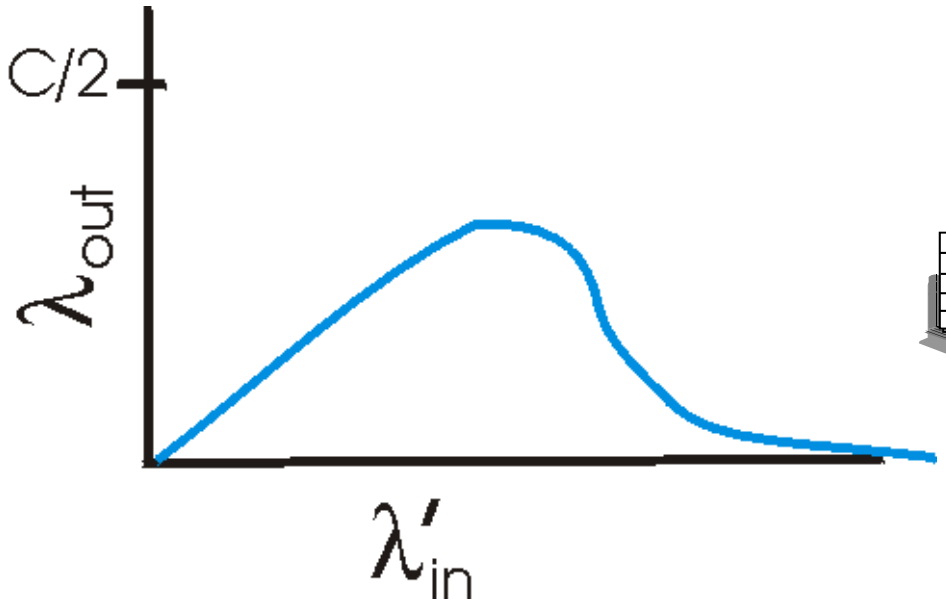
Tıkanıklığın nedenleri/maliyeti: Senaryo 3

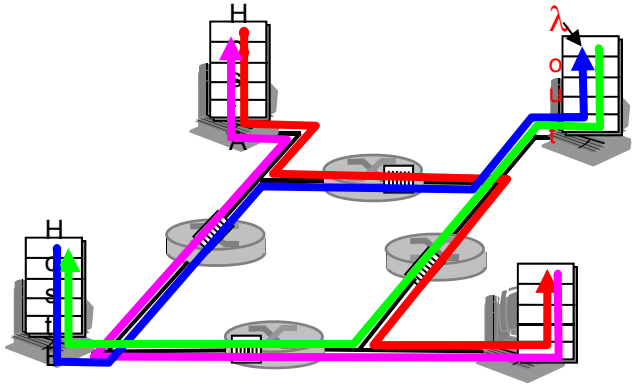
- Dört gönderici
- Birden çok atlamalı yol
- Zaman aşımı/yeniden iletimler

Q: λ_{in} ve λ'_{in} artarsa ne olur?



Tıkanıklığın nedenleri/maliyeti: Senaryo 3





Tıkanıklığın başka bir "maliyeti":

- Bir paket yol esnasında atıldığında, bu paketi atıldığı noktaya kadar iletmek içi her yukarı akış hattı tarafından kullanılan iletim kapasitesi kaybedilmiş olur!

Tıkanıklık kontrolü yaklaşımları

Tıkanıklık kontrolü için iki temel yaklaşım vardır:

Uçtan uca tıkanıklık kontrolü yaklaşımı :

- Ağdan herhangibir geri bildirim alınmaz
- Tıkanıklık uç sistemlerin kayıp ve gecikmeleri takibi ile anlaşılır
- TCP'deki yaklaşımdır

Ağ yardımı ile tıkanıklık kontrolü:

- Yönlendiriciler ağdaki tıkanıklık durumu ile ilgili uç sistemlere açık bir geri besleme sunarlar
  - Tıkanıklığı belirten bir bit kullanılır (SNA, DECbit, TCP/IP ECN, ATM)
  - Yönlendirici, göndericiyi hat üzerinde destekleyebileceği iletim hızı ile ilgili olarak açık şekilde bilgilendirir

Örnek: ATM ABR tıkanıklık kontrolü

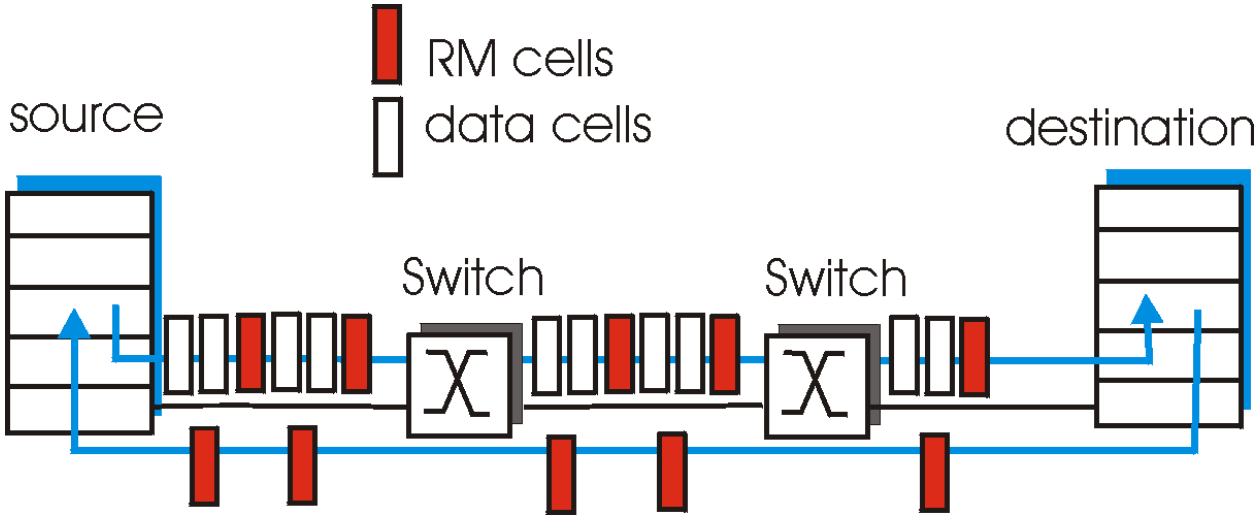
ABR: kullanılabilir bit hızı (available bit rate):

- "elastik" bir veri iletim servisi
- Ağ "yük altında değilken":
  - Kullanılabilir bant genişliğini kullanabilmenin avantajından yararlanmayı
- Ağ tıkanık iken ise:
  - Göndericinin daha önceden belirlenmiş minimum bir iletim hızına kısılmasını sağlar

RM – Kaynak yönetim Hücreleri (resource management cells):

- Ana sistemler ve anahtarlar arası tıkanıklıkla ilgili bilgileri taşır
- RM hücrelerindeki bit'ler switchler tarafından ayarlanır switches ("*network-yardımlı*")
  - NI bit: hızda artış yok -no increase in rate ➔orta dereceli tıkanıklık
  - CI bit: tıkanıklık göstergesi
- RM hücreleri göndericiye bit'ler değiştirilmeden geri gönderilir.

Örnek: ATM ABR tıkanıklık kontrolü



- RM hücrelerindeki iki-byte'lık ER-kesin hız (explicit rate) alanı
  - Tıkalı bir anahtar RM hücresindeki ER alanındaki değeri düşürebilir
  - Böylece kaynaktan hedefe doğrutüm anahtarlardaki desteklenen minimum hız ayarlanabilir
- Veri hücrelerindeki EFCI biti : tıkanık switch'de 1 olarak ayarlanır
  - Eğer RM hücresini takip eden veri hücresinde EFCI bit'i ayarlanmışsa, gönderici geri dönen RM hücresindeki CI bit'ini ayarlar

Taşıma Katmanı

- 3.1 Taşıma katmanı servisleri
- 3.2 Çoklama (multiplexing) ve çoklamanın çözülmesi (demultiplexing)
- 3.3 Bağlantısız taşıma: UDP
- 3.4 Güvenilir veri iletiminin prensipleri
- 3.5 Bağlantı yönelimli taşıma: TCP
  - segment yapısı
  - güvenilir veri iletimi
  - akış kontrolü
  - bağlantı yönetimi
- 3.6 Tıkanıklık kontrolü prensipleri
- 3.7 TCP tıkanıklık kontrolü

TCP Tıkanıklık Kontrolü

- TCP'nin congestion control mekanizması, her bir kullanıcının ağdaki sıkışıklığı algıladığı oranda veri göndermesini azaltması ve limitlemesi prensibi ile çalışmaktadır.
- Gönderen, kendisi ile alıcı arasında düşük bir tıkanıklık olduğunu algılıyorsa o zaman gönderme oranını artırabilir, eğer yoğun tıkanıklık olduğunu düşünüyorsa da o zaman gönderme oranını düşürmelidir.
  - TCP bağlantıya bıraktığı trafiği nasıl kısıtlayabilir?
  - Kendisi ile alıcı arasında tıkanıklık olduğunu nasıl anlıyor?
  - İki nokta arasındaki tıkanıklığa dayalı olarak aktarım oranını değiştirecek ne tür bir fonksiyon kullanmalı?

TCP Tıkanıklık Kontrolü

- TCP bağlantıya bıraktığı trafiği nasıl kısıtlayabilir?

Bir göndericideki alındı mesajı alınmamış olan veri miktarı CongWin (Tıkanıklık penceresi) ve RcvWindow değerlerinin minimumunu geçmemelidir

LastByteSent-LastByteAcked ≤ min {CongWin,RcvWindow}

Kabaca göndericinin gönderim hızı

$$\text{rate} = \frac{\text{CongWin}}{\text{RTT}} \text{ Bytes/sec}$$

TCP Tıkanıklık Kontrolü

- Kendisi ile alıcı arasında tıkanıklık olduğunu nasıl anlıyor?
  - Zaman aşımı
  - 3 kopya ACK

- Tıkanıklık yoksa doğru ve düzenli gelen alındıları CongWin boyutunu artırmak için kullanır

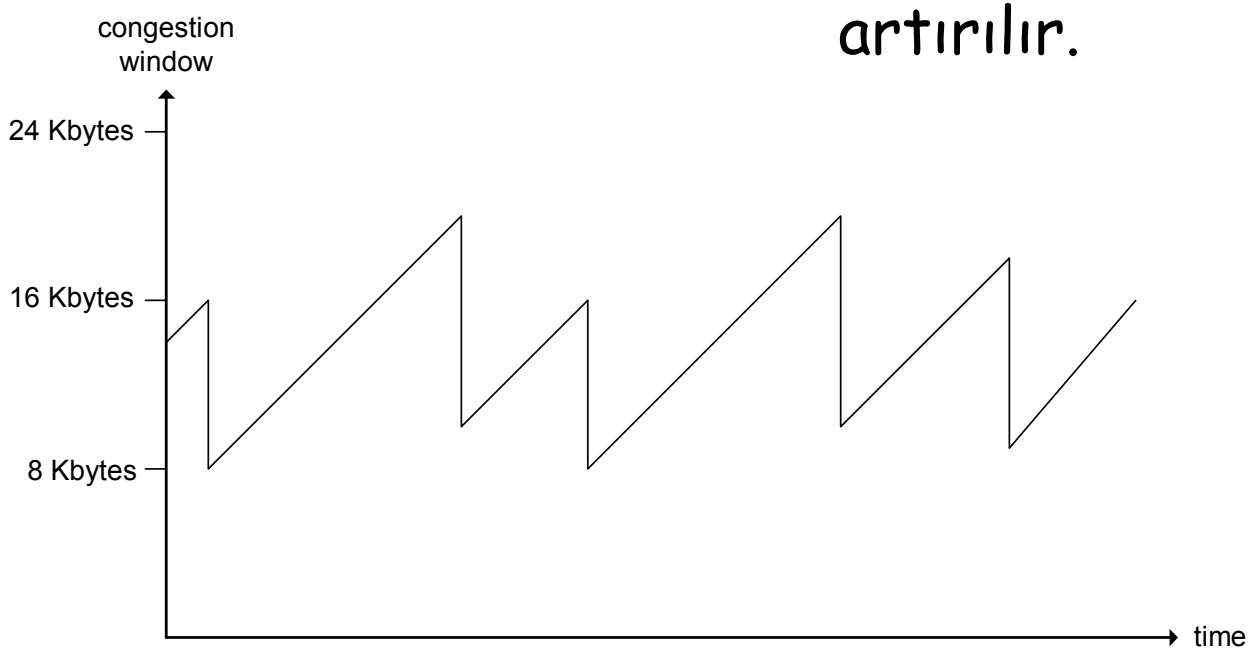
TCP Tıkanıklık Kontrolü

- TCP tıkanıklık kontrolü algoritması
- 3 ana bileşeni vardır:
  - Toplamsal artış, çarpımsal düşüş (additive increase, multiplicative decrease - AIMD)
  - Yavaş başlangıç (slow start)
  - Zaman aşımı olayına tepki

TCP AIMD

Çarpımsal düşüş: kayıp meydana geldiğinde CongWin değeri yarıya indirilir

Toplamsal artış: Kayıp olmadığı durumda CongWİn değeri her RTT için 1 MSS artırılır.



Long-lived TCP connection

TCP Slow Start

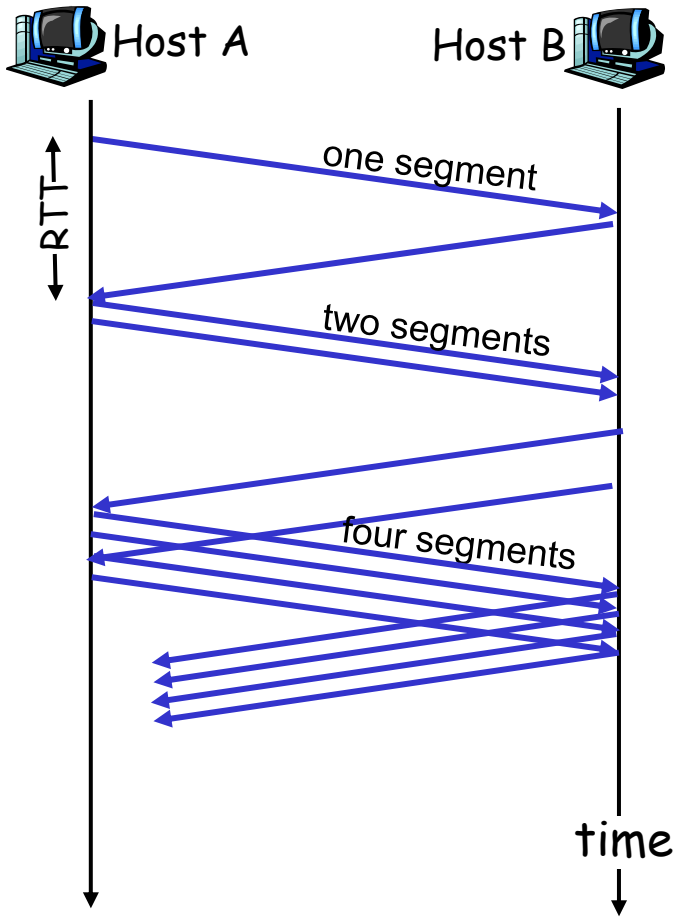
- Bağlantı kurulduğunda, **`CongWin`** = 1 MSS olarak belirlenir
  - Örn: MSS = 500 bytes & RTT = 200 msec
  - İlk hız = 20 kbps
- Kullanılabilir bant genişliği >> MSS/RTT
  - bunu adım adım 1 MSS ile artırmak kayıp olurdu

- Kayıp meydana gelirse CongWin değeri yarıya düşer, sonra da önce tanımladığımız gibi çizgisel artar.

TCP Slow Start

Başlangıç fazında hızı doğrusal-çizgisel artırmak yerine TCP göndericisi hızını, her RTT'de CongWin değerini ikiye katlayarak,üssel olarak artırır

Özet: ilk hız yavaştır ancak üssel olarak artar



Zaman aşımı

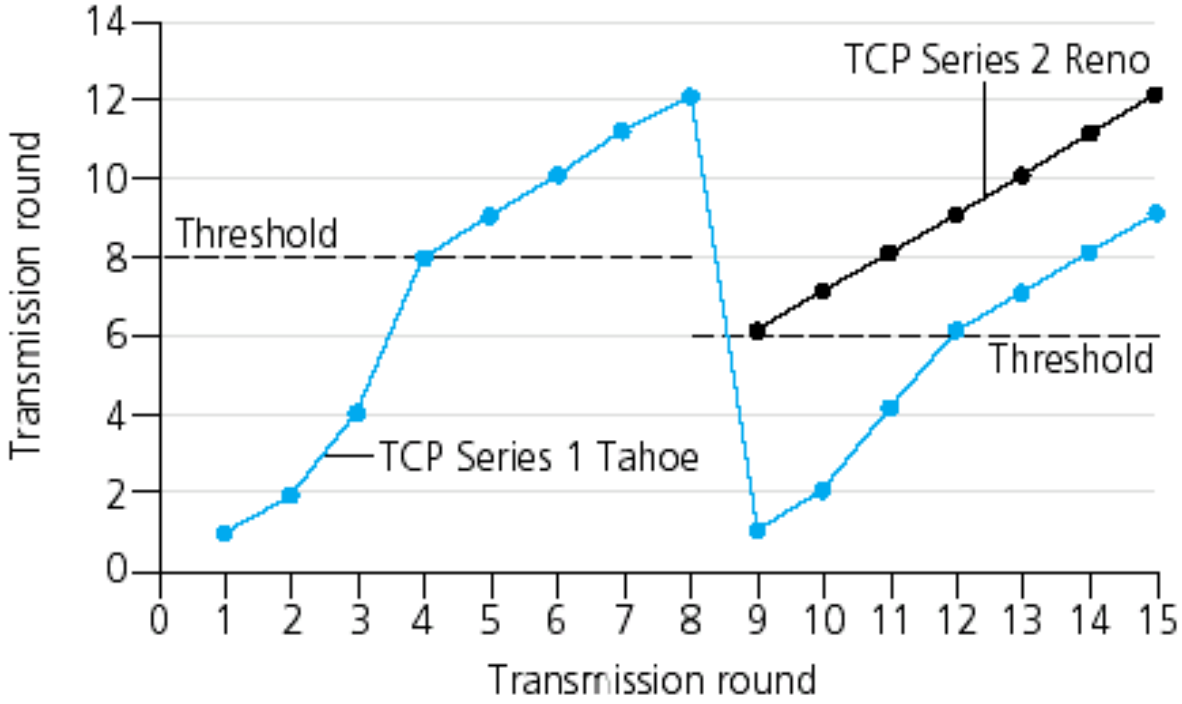
- TCP yavaş başlamanın sona ereceği bir **Threshold** (eşik) değeri belirler ➔ Başta 65Kbyte gibi büyük bir değer seçilir
- Zaman aşımı yaşandığında zaman aşımı olayının yaşandığı zamanki CongWin değerinin yarısı yeni Threshold olarak belirlenir
- Zaman aşımından sonra TCP göndericisi yavaş başlama fazına girer :
  - CongWin=1MSS
  - Üssel olarak yeni belirlenen Threshold değerine kadar hızını artırır

Tıkanıklık Kontrolü

- Neden TCP, zaman aşımında Congwin'i 1 MSS'ye düşürürken, 3 kopya ACK alımında yarıya indirerek tepki verir?
- 3 kopya ACK alımı networkün hala belli miktar veriyi iletebildiği anlamına gelir
- Zaman aşımı ise daha ciddidir

➔ hızlı kurtarma (fast recovery)

TCP Reno vs TCP Thoe



Özet: Tıkanıklık Kontrolü

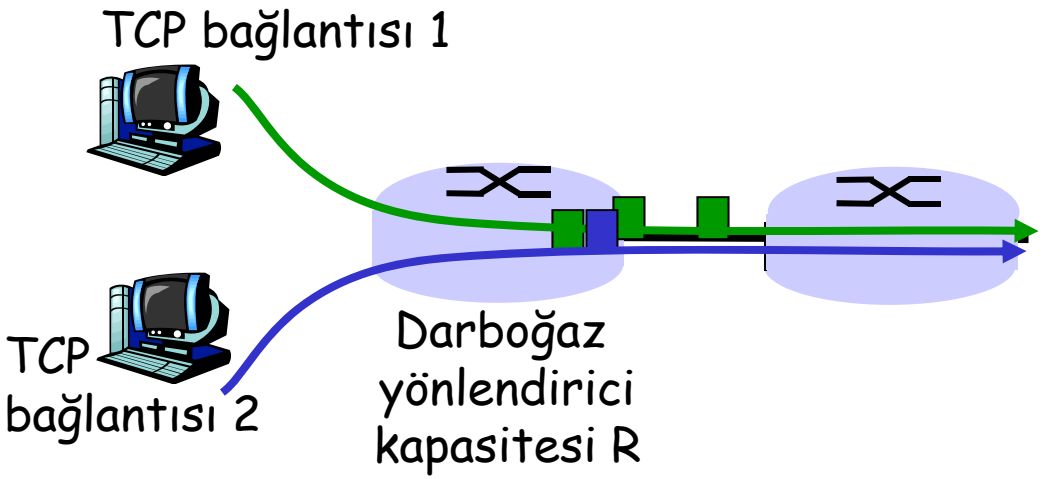
- **CongWin Threshold değerinin altındaysa**, gönderici slow-start fazındadır, pencere üssel artar.
- **CongWin Threshold değerinin üstündeyse**, gönderici tıkanıklık önleme fazındadır, pencere lineer artar.
- 3 kopya ACK meydana geldiğinde, **Threshold CongWin/2** olarak ayarlanır ve **CongWin** de **Threshold'a** eşitlenir, pencere lineer artar
- Zaman aşımı meydana geldiğinde, **Threshold CongWin/2** olarak ayarlanır ve **CongWin** 1 MSS'ye eşitlenir, pencere Threshold'a kadar üssel artar

Tıkanıklık Kontrolü

Olay	Durum	TCP Gönderici Eylemi	Açıklama
Daha önce alındı bilgisi alınmayan veriler için ACK alımı	Yavaş Başlama Slow Start (**SS**)	CongWin = CongWin + MSS, If (CongWin > Threshold) durum "Congestion Avoidance" (Tıkanıklıktan kaçınma) olarak ayarlanır	Her RTT'de CongWin değerinin ikiye katlanması ile sonuçlanır
Daha önce alındı bilgisi alınmayan veriler için ACK alımı	Tıkanıklıktan Kaçınma Congestion Avoidance (**CA**)	CongWin = CongWin+MSS * (MSS/CongWin)	CongWin değerinin her RTT için 1 MSS çarpımı ile büyümesiyle sonuçlanan toplamsal artış.
Üç kopya ACK tarafından tespit edilen kayıp olayı	SS veya CA	Threshold = CongWin/2, CongWin = Threshold, durum "Congestion Avoidance" olarak ayarlanır	Çarpımsal düşüşü uygulayan ***hızlı kurtarma***. CongWin 1 MSS altına düşmez.
Zaman aşımı	SS veya CA	Threshold = CongWin/2, CongWin = 1 MSS, durumu "Slow Start" olarak ayarlanır	Yavaş başlamaya girer
Çift ACK	SS veya CA	Alındı bilgisi alınan segment için çift ACK sayımını artırır.	CongWin and Threshold değişmez

TCP Fairness (Adalet)

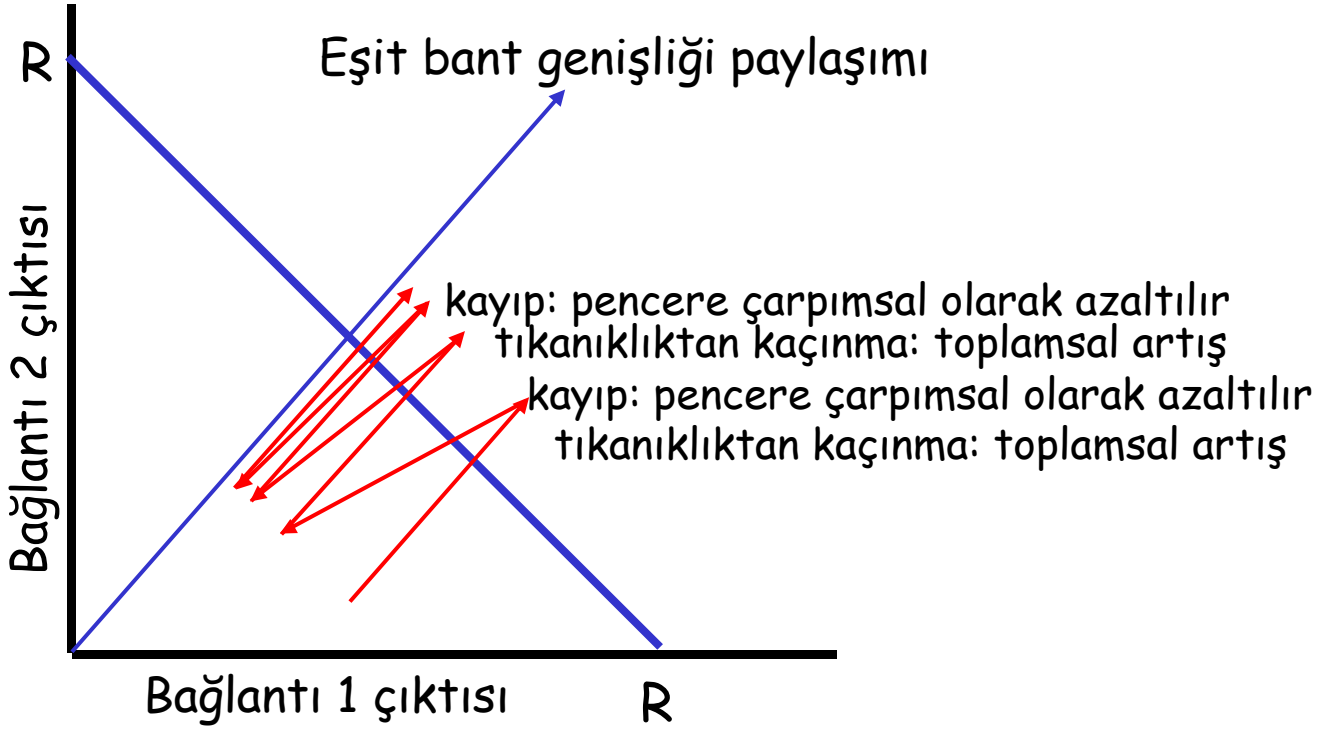
Fairness amacı: Eğer K TCP bağlantısı aynı R bant genişliğine sahip dar boğaz hattını paylaşıyorlarsa, her biri ortalama R/K iletim hızına sahip olmalıdır



Why is TCP fair?

İki yarışan bağlantı:

- Toplamsal artış, çıktı arttıkça 45 derecelik eğim verir
- Çarpımsal düşüş çıktıyı (throughput) orantılı olarak azaltır



Adalet

Adalet ve UDP

- Multimedia uygulamaları genellikle TCP kullanmazlar
  - Hızın tıkanıklık kontrolünden etkilenmesini istemezler
- Yerine UDP kullanırlar:
  - audio/video paketlerini sabit hızda gönderir, paket kaybını tolere edebilirler

Adalet ve paralel TCP bağlantıları

- Uygulamaların iki ana sistem arasında paralel bağlantı kurmalarını hiçbir şey engelleyemez.
- Web tarayıcıları bunu kullanır

Taşıma Katmanı: Özet

- Taşıma katmanı servisleri ardındaki presnsipler:
  - Çoklama, çoklamanın çözülmesi (multiplexing, demultiplexing)
  - Güvenilir veri iletimi (reliable data transfer)
  - Akış kontrolü (flow control)
  - Tıkanıklık kontrolü (congestion control)

- Tüm bunların Internet'teki uygulamalarını gördük
  - UDP
  - TCP

Haftaya:

- ağ "sınır"ından çıkıp (application, transport layers)
- Ağ "çekirdeği"ne geçiyoruz