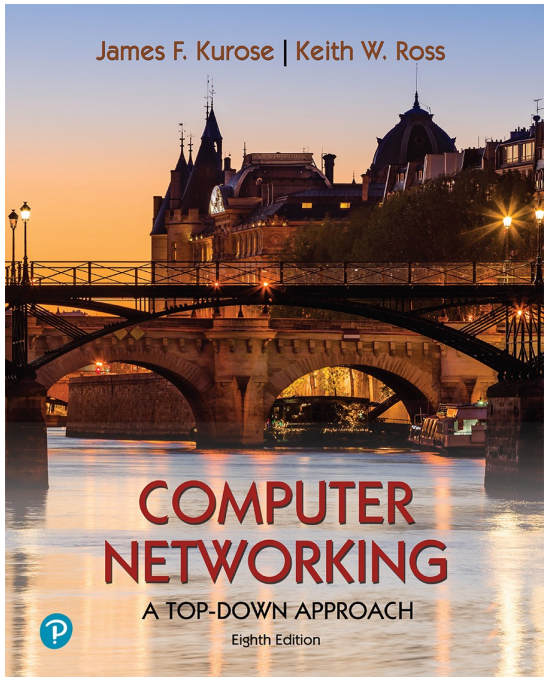
Bölüm 3 Taşıma Katmanı

Doç. Dr. Mehmet Dinçer Erbaş
Bolu Abant İzzet Baysal Üniversitesi
Mühendislik Fakültesi
Bilgisayar Mühendisliği Bölümü



Computer Networking: A Top-Down Approach
8th edition
Jim Kurose, Keith Ross
Pearson, 2020

Bölüm 3: konular

3.1 taşıma-katmanı hizmetler

3.2 çoklama ve çoklama çözme

3.3 bağlantısız taşıma: UDP

3.4 güvenilir veri transferi prensipleri

3.5 bağlantı-odaklı taşıma: TCP

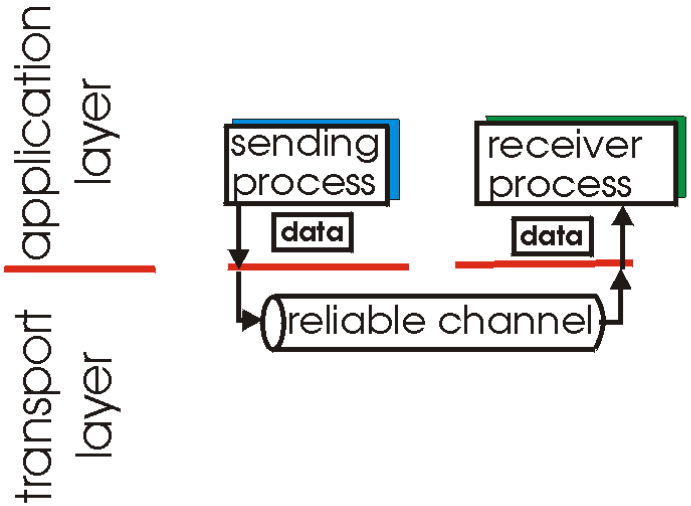
- Segment yapısı
- Güvenilir veri transferi
- Akış kontrolü
- Bağlantı yönetimi

3.6 sıkışma kontrolü prensipleri

3.7 TCP sıkışma kontrolü

Güvenilir veri transferi prensipleri

- Uygulama, taşıma ve bağlantı katmanları için önemli
  - Ağ konusunda en önemli konulardan biri.

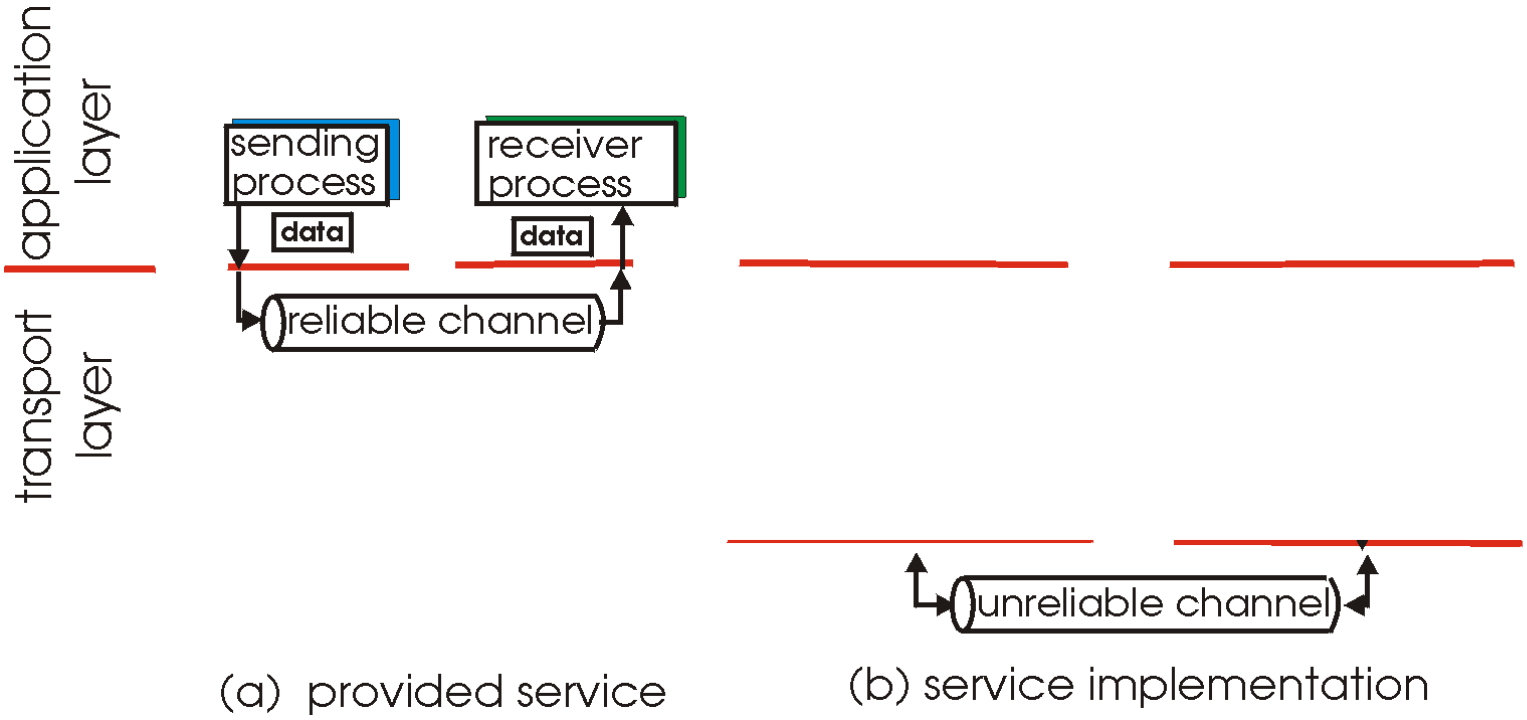


(a) provided service

Güvenilir olmayan kanalın özellikleri güvenilir veri transferi protokolünün özelliklerini (rdt) belirleyecektir.

Güvenilir veri transferi prensipleri

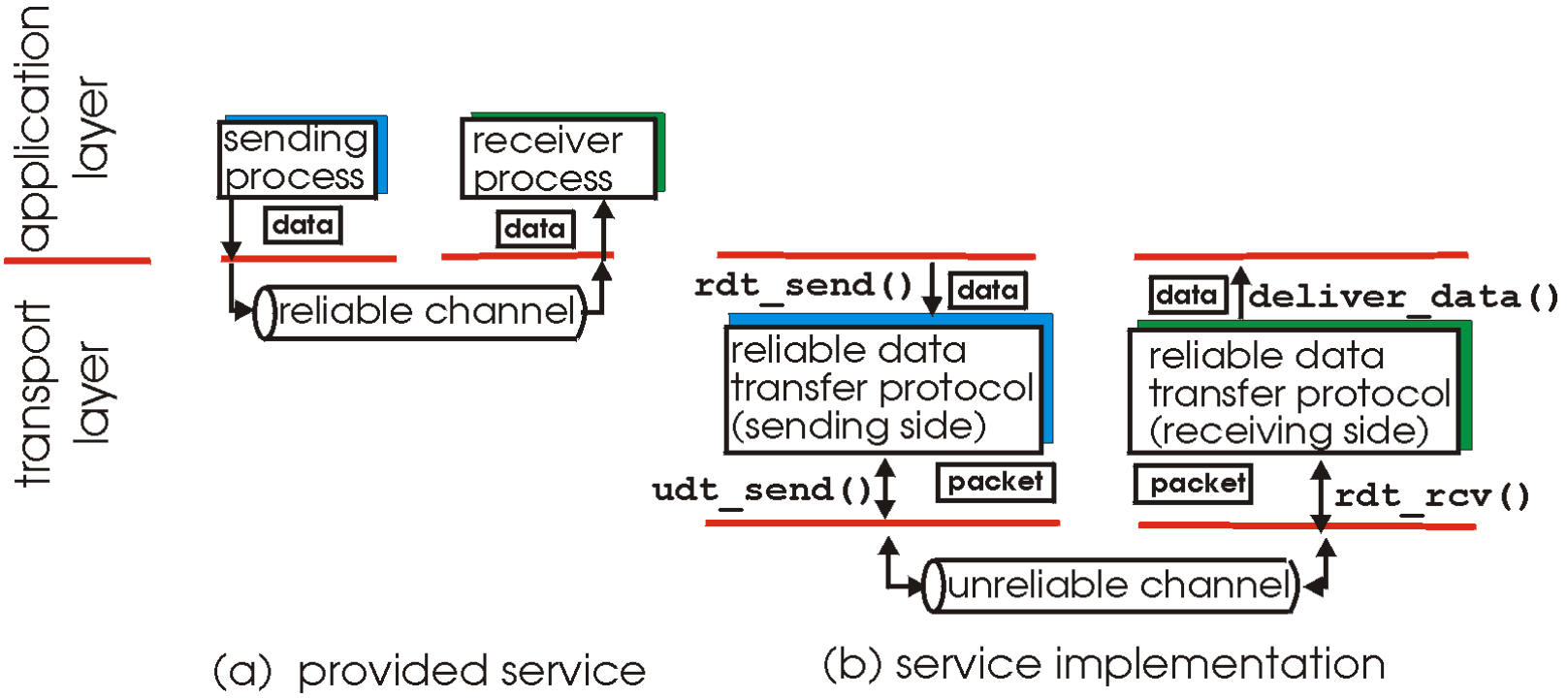
- Uygulama, taşıma ve bağlantı katmanları için önemli
  - Ağ konusunda en önemli konulardan biri.



- Güvenilir olmayan kanalın özellikleri güvenilir veri transferi protokolünün özelliklerini (rdt) belirleyecektir.

Güvenilir veri transferi prensipleri

- Uygulama, taşıma ve bağlantı katmanları için önemli
  - Ağ konusunda en önemli konulardan biri.

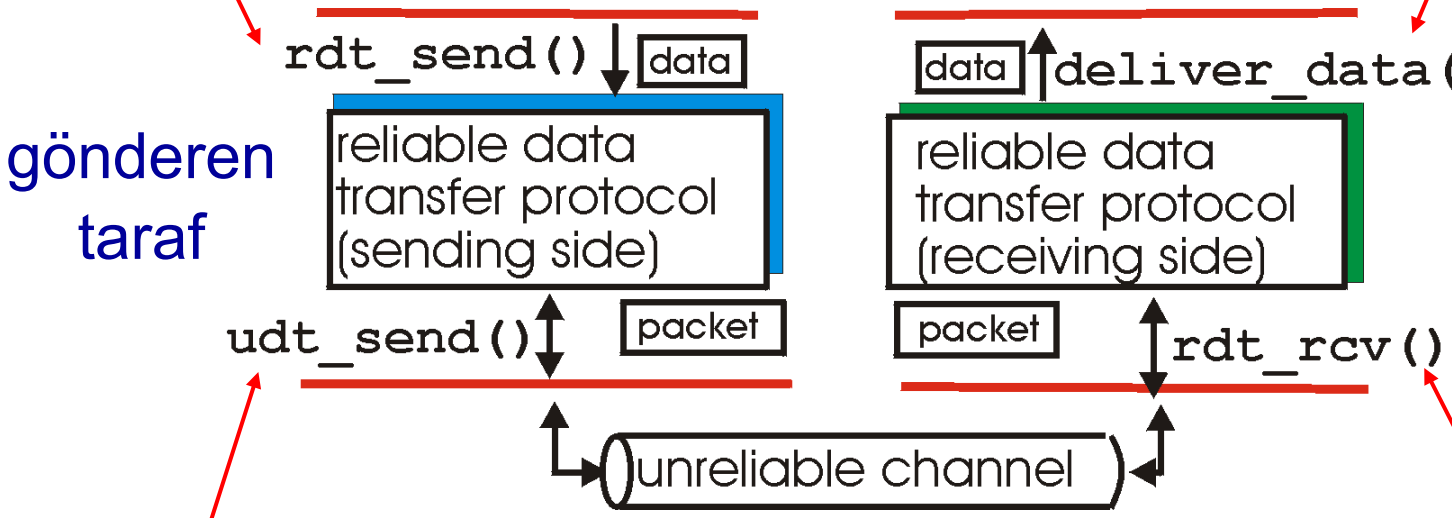


Güvenilir olmayan kanalın özellikleri güvenilir veri transferi protokolünün özelliklerini (rdt) belirleyecektir.

Güvenilir veri transferi

rdt_send(): yukarıdan çağırılır, (uyg.), alıcıya güvenli veri transferini sağlar.

deliver_data(): veriyi üst katmana göndermek için rdt tarafından çağırılır.



udt_send(): rdt tarafından çağırılır, paketi güvenilir olmayan kanal üzerinden alıcıya gönderir.

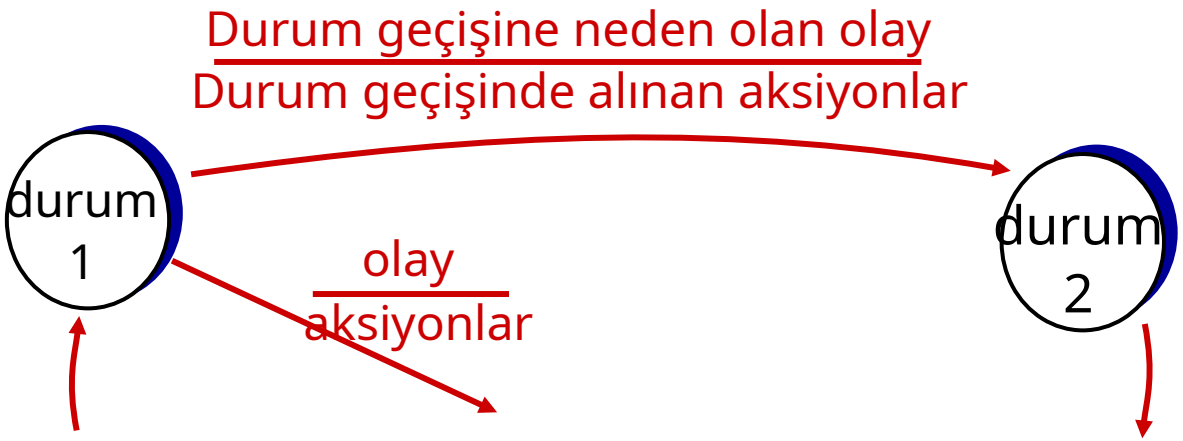
rdt_rcv(): kanalın alan tarafındanda paket alındığında çağırılır.

Güvenilir veri transferi

Bu bölümde

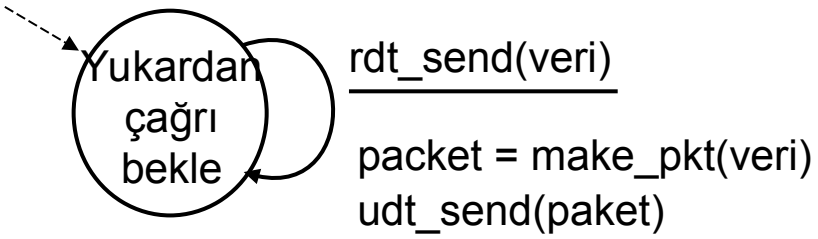
- Aşamalı olarak gönderici ve alıcı tarafından güvenilir veri transferi protokolünü (rdt) geliştireceğiz.
- Tek taraflı veri transferine odaklanacağız.
  - Ancak kontrol bilgisi iki taraflı akacak.
- Sonlu durum bilgisi (FSM) yöntemi ile gönderici ve alıcıyı tanımlayacağız.

durum: bu durumdayken bir sonraki durum bir sonraki olay ile belirlenir.

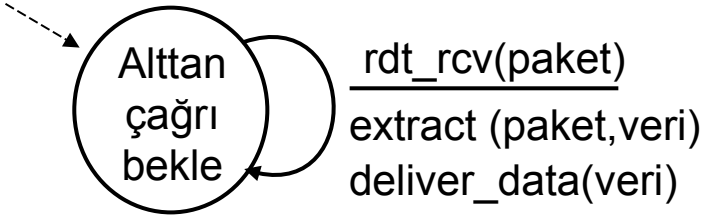


rdt1.0: güvenilir kanal üzerinden güvenilir transfer

- Altaki kanal tam anlamıyla güvenilir.
  - Bit hatası yok
  - Paket kaybı yok
- Gönderici ve alıcı için ayrı FSM:
  - Gönderici alttaki kanala veri gönderir.
  - Alıcı alttaki kanaldan veri alır.



gönderen



alan

rdt2.0: bit hataları olabilen kanal

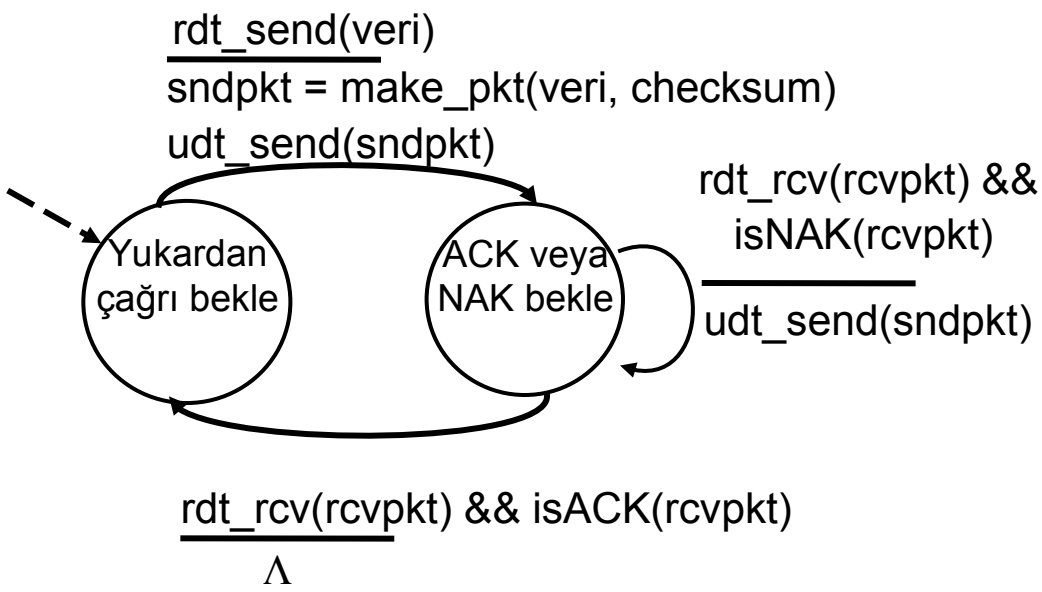
- Alttaki kanal paketteki bitleri değiştirebilir.
  - checksum ile bit hatalarını tespit edebiliriz.
- *Sorun*: hata durumunda ne yapılmalı?

İnsanlar sohbet sırasında "hata" olursa bu durumu nasıl çözer?

rdt2.0: bit hataları olabilen kanal

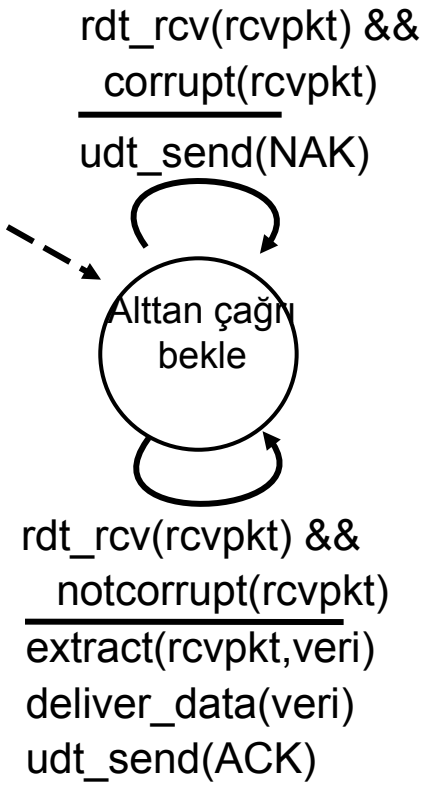
- Alttaki kanal paketteki bitleri değiştirebilir.
  - checksum ile bit hatalarını tespit edebiliriz.
- *Sorun*: hata durumunda ne yapılmalı?
  - *Onaylamalar (ACKs):* alıcı göndericiye paketin doğru alındığını açıkça söyler.
  - *negatif onaylama (NAKs):* alıcı göndericiye pakette hata olduğunu açıkça söyler.
  - Gönderici NAK alırsa paketi tekrar yollar.
- **`rdt2.0`** (geliştirilmiş **`rdt1.0`**):
  - Hata tespiti
  - alıcı geribildirimi: kontrol mesajları (ACK,NAK) - alıcı → gönderici

rdt2.0: FSM tanımı

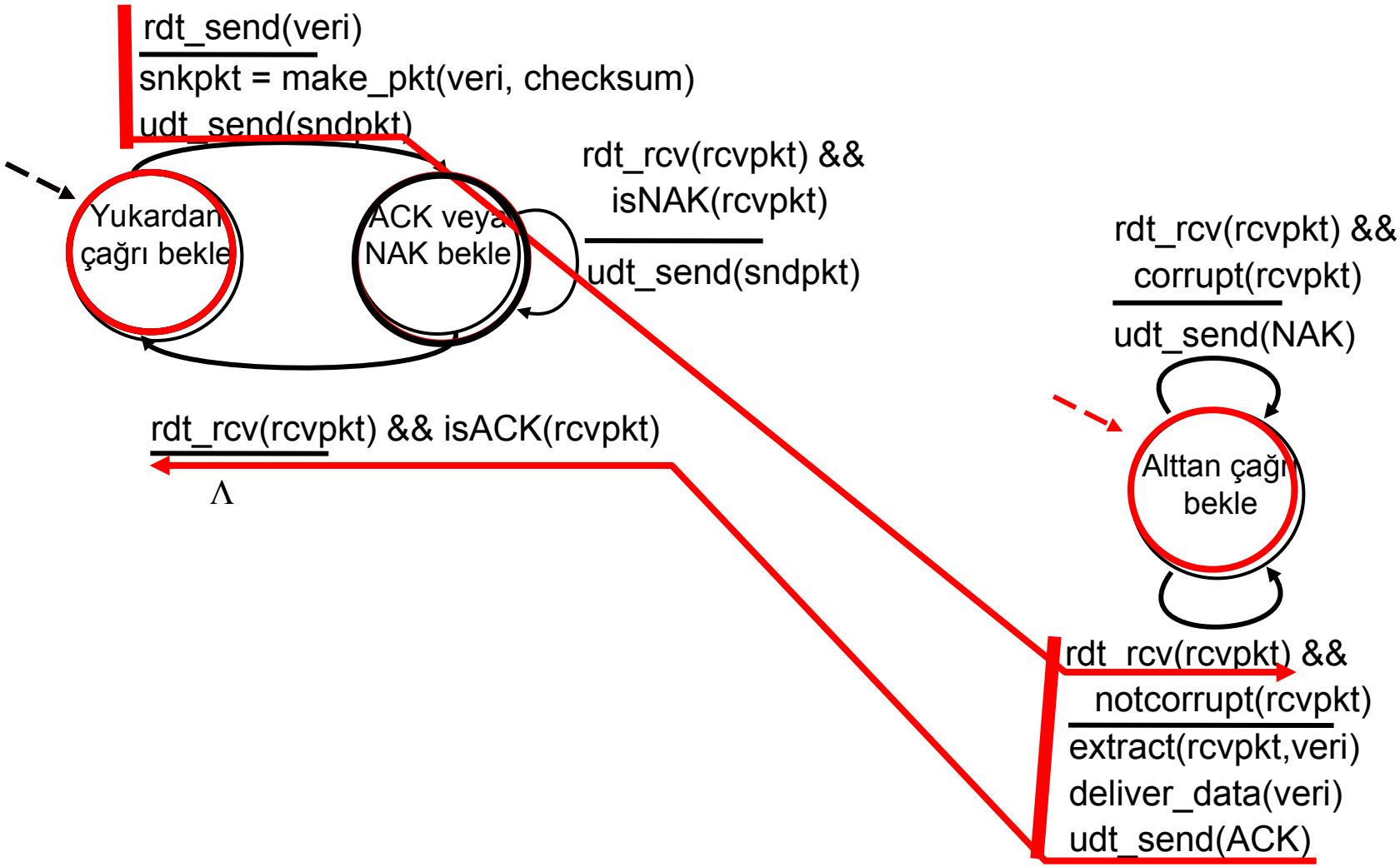


gönderici

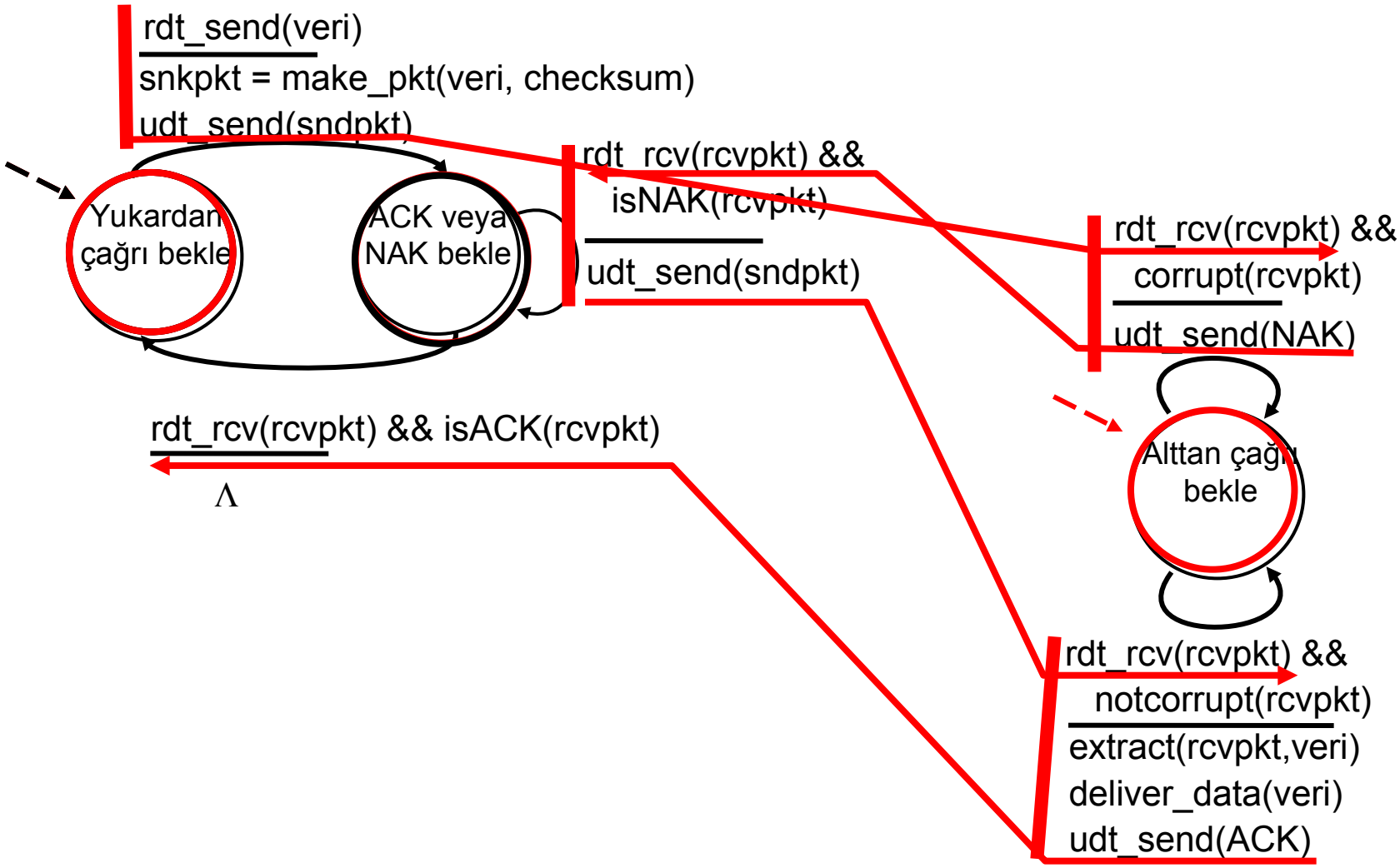
alıcı



rdt2.0: operation with no errors



rdt2.0: error scenario



rdt2.0 önemli bir sorun içeriyor!

ACK/NAK mesajı bozulursa ne olacak?

- Gönderici alıcı ne olduğunu bilemez!
- Kafasına göre tekrar gönderemez: aynı paketten iki kopya olabilir.

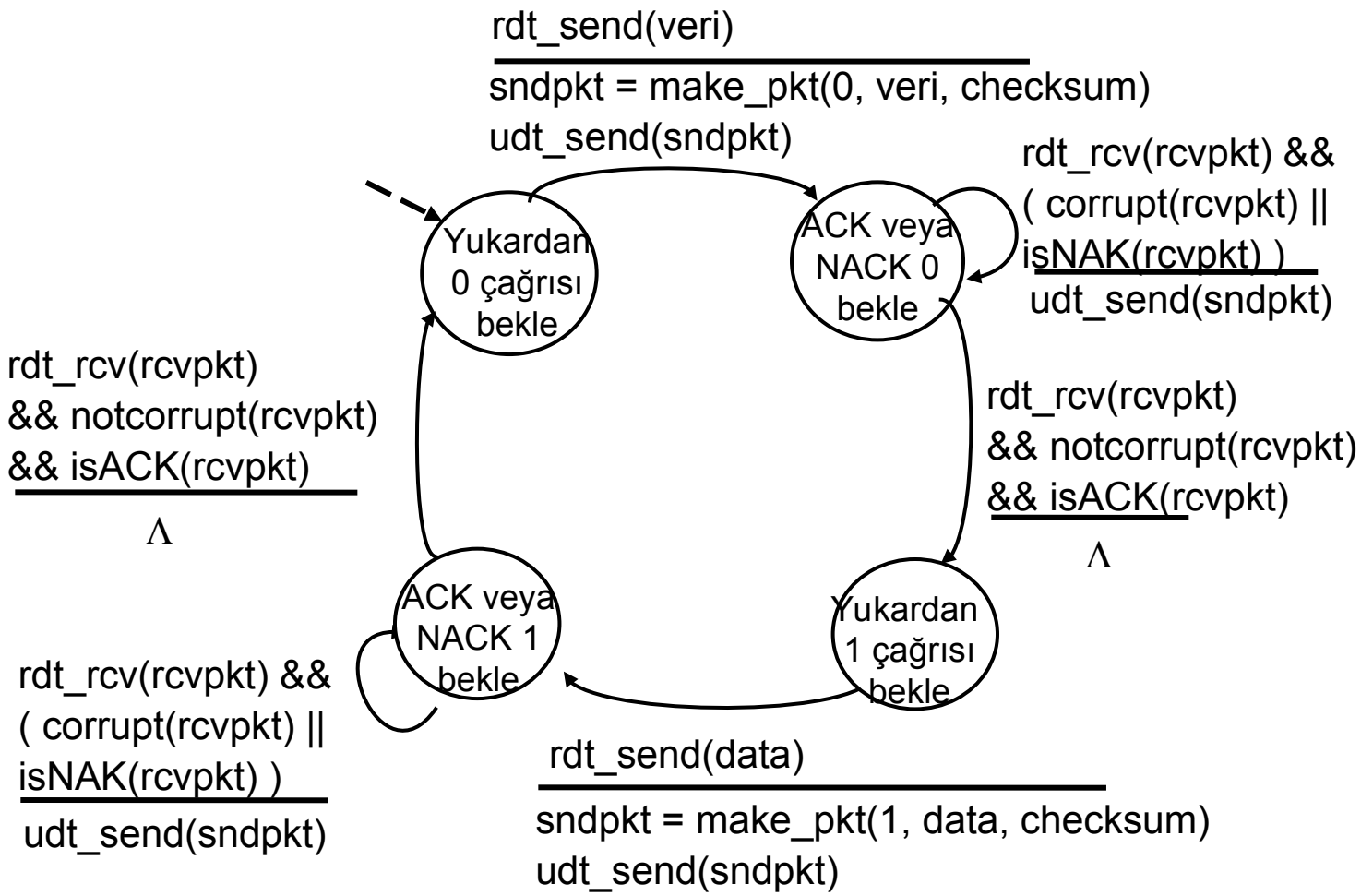
Fazla kopya sorunu şu şekilde halledilir:

- Gönderici bozuk ACK/NAK alırsa paketi tekrar yollar.
- Gönderici her pakete sıra numarası ekler.
- Alıcı kopya paket alırsa görmezden gelir (yukarı yollamaz).

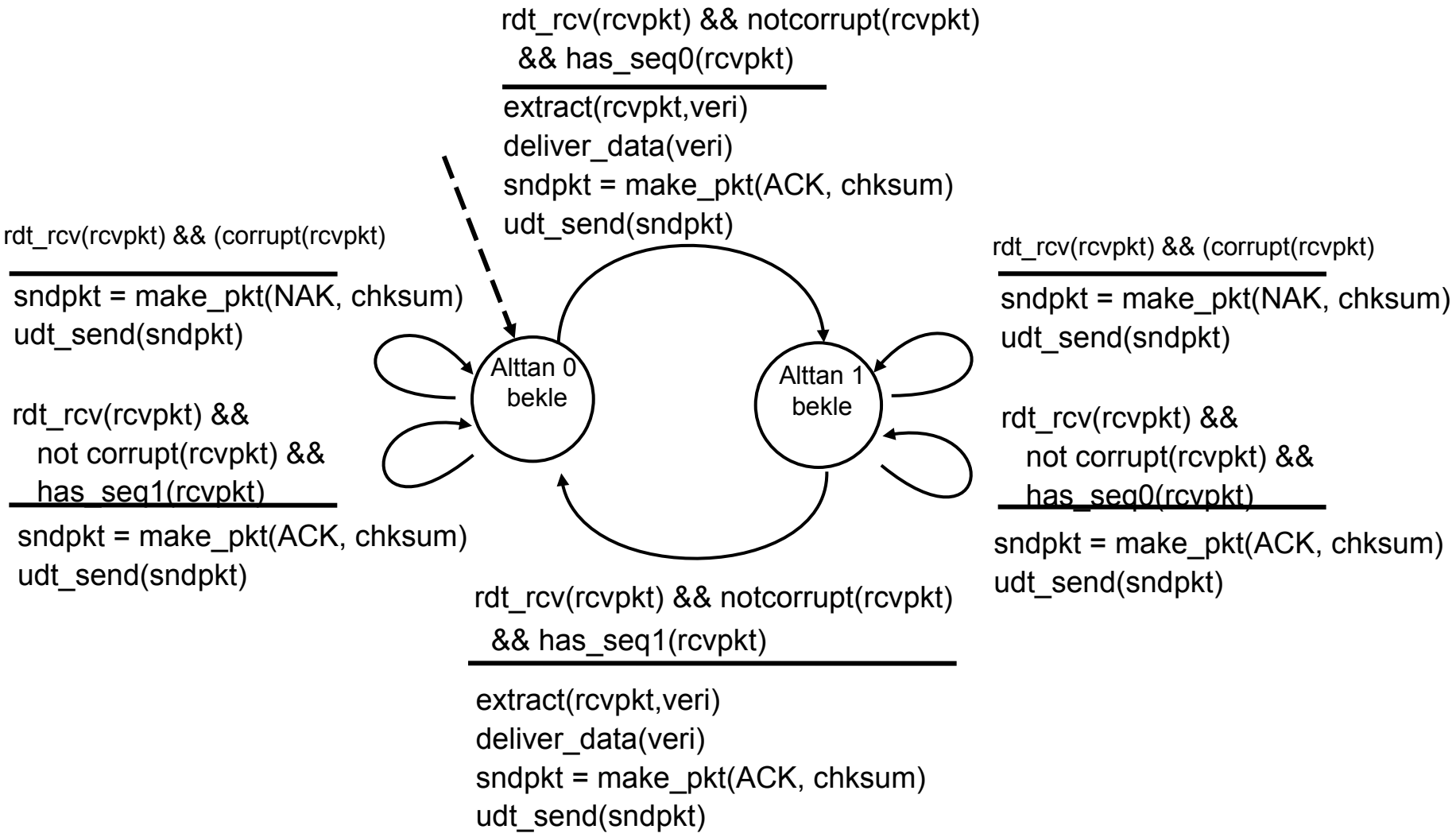
dur ve bekle

Gönderici bir paket yollar, sonra alıcıdan karşılık bekler.

rdt2.1: gönderici, bozuk ACK/NAK sorununu çözdü



rdt2.1: alıcı, bozuk ACK/NACK sorununu çözdü



rdt2.1: İnceleme

gönderici:

- Paketlere sıra # eklendi
- Iki sıra # (0,1) yeterli olacaktır. Neden?
- Alınan ACK/NACK bozuk mu diye kontrol edilmelidir.
- İki kat fazla durum gerekti.
  - Durumlar sonraki "beklenen" paketin sıra numarası 0 veya 1 olması gerektiğini "hatırlamalıdır".

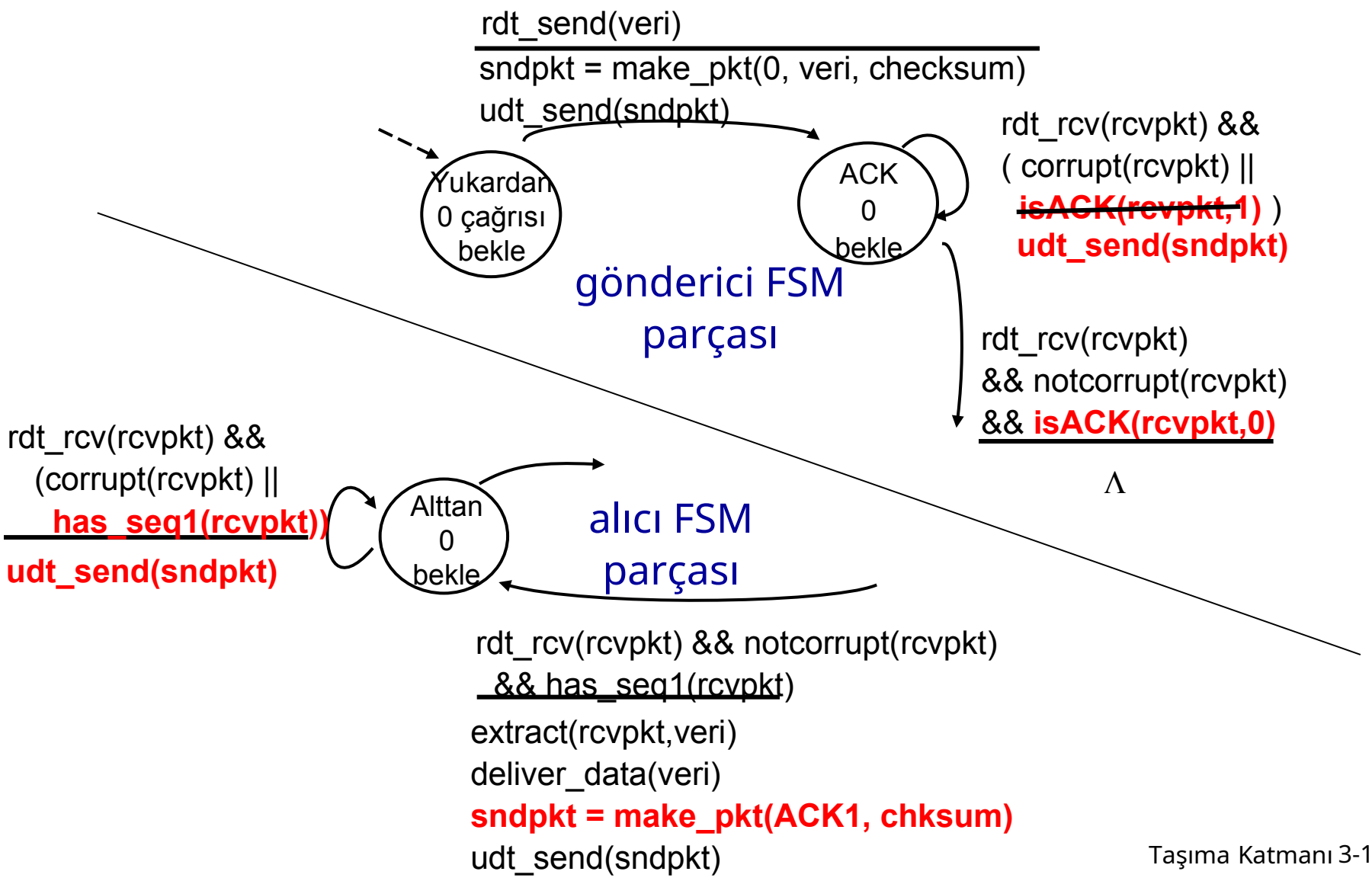
alıcı:

- Alınan paketin kopya olup olmadığını kontrol etmelidir.
  - Durum bir sonra beklenen paketin sıra numarasının 0 veya 1 olması gerektiğini belirliyor.
- Not: alıcı son gönderdiği ACK/NACK mesajının gönderici tarafından alındığını ve alınmadığını bilemez.

rdt2.2: NAK gerektirmeyen protokol

- rdt2.1 ile aynı fonksiyonlara sahip, ancak sadece ACK kullanıyor.
- NAK yerine, alıcı düzgün şekilde aldığı son paket için ACK gönderiyor.
  - Alıcı ACK gönderilen paketin sıra numarasını açıkça belirtmelidir.
- Gönderici kopya ACK alırsa NAK almış gibi aksiyon alır: sırada gönderilen paketi tekrar gönder.

rdt2.2: gönderici ve alıcının bir kısmı



rdt3.0: hata ve kayıp olabilen kanal

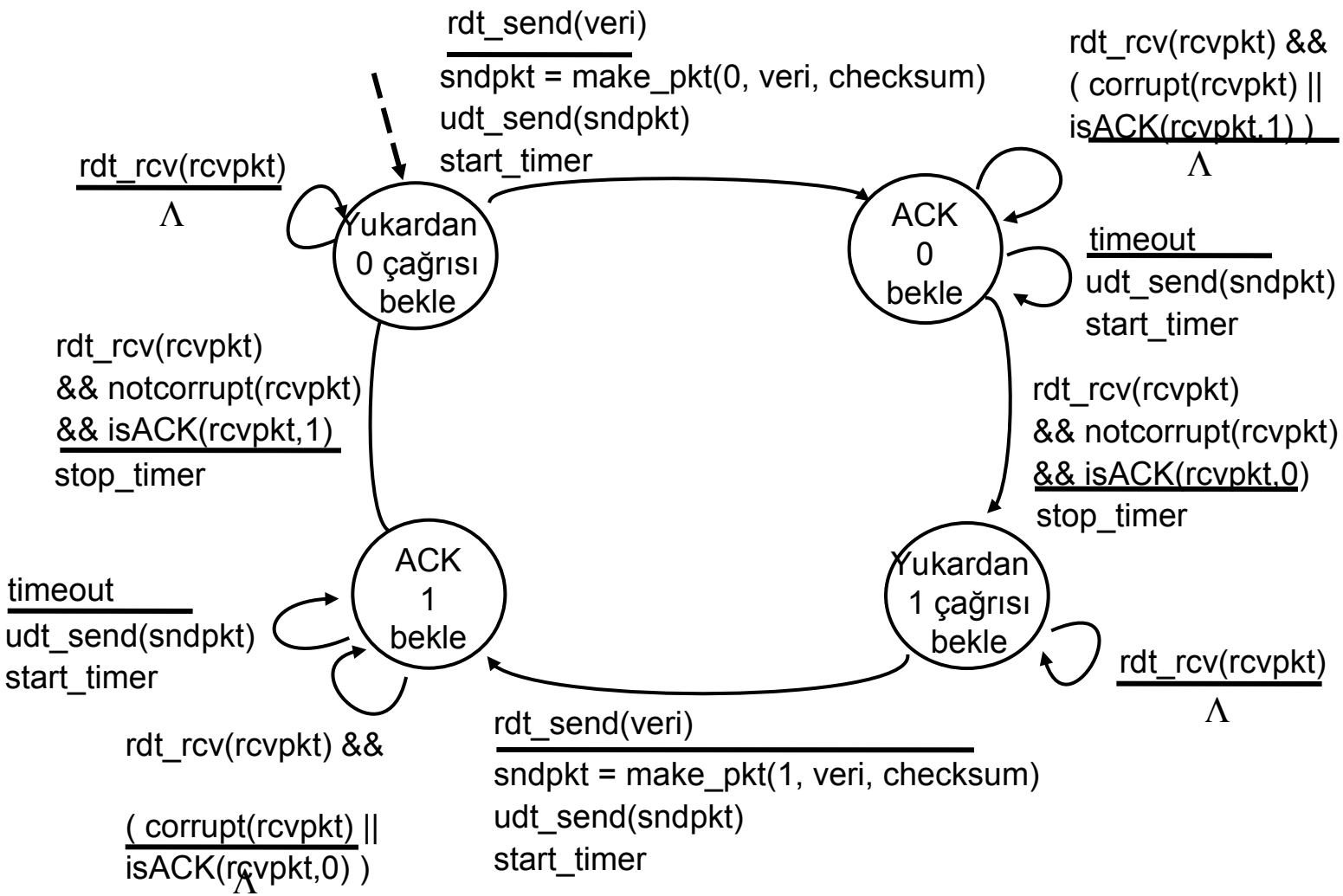
Yeni varsayım: alttaki kanal paketleri kaybedebilir (ACK ve veri).

- checksum, sıra #, ACK mesajı, tekrar gönderme yararlıdır … ama yeterli değil.

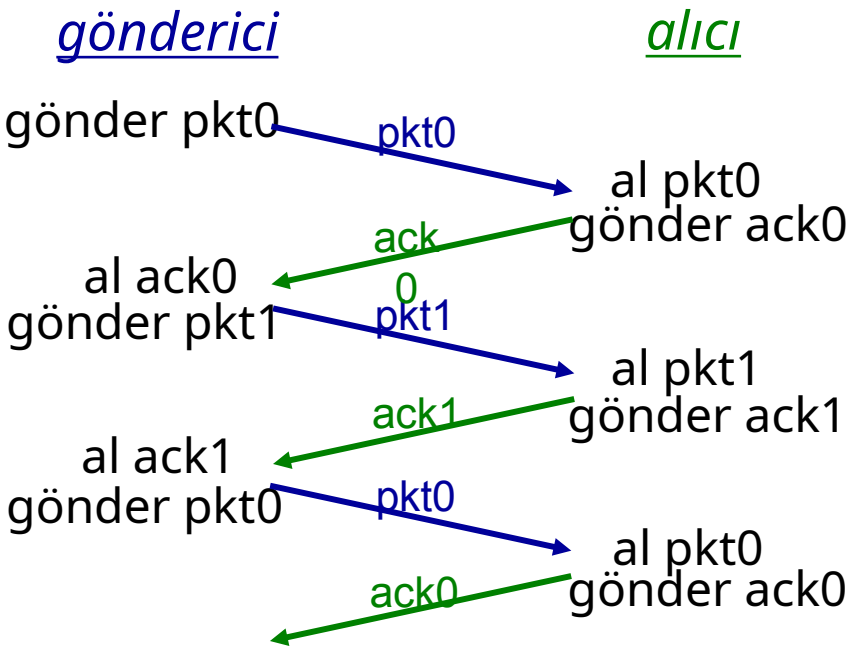
Yaklaşım: gönderici "makul" bir süre ACK mesajı için bekler.

- Bu süre içerisinde ACK gelmezse tekrar gönderir.
- Eğer paket (veya ACK) sadece gecikmişse (kaybolmamışsa):
  - Tekrar gönderim ile aynı paketin kopyası oluşacaktır, ancak sıra # sayesinde bu sorun çözülür.
  - Alıcı ACK gönderilen paketin sıra numarası belirtmelidir.
- Geri sayım sayacı gerekir.

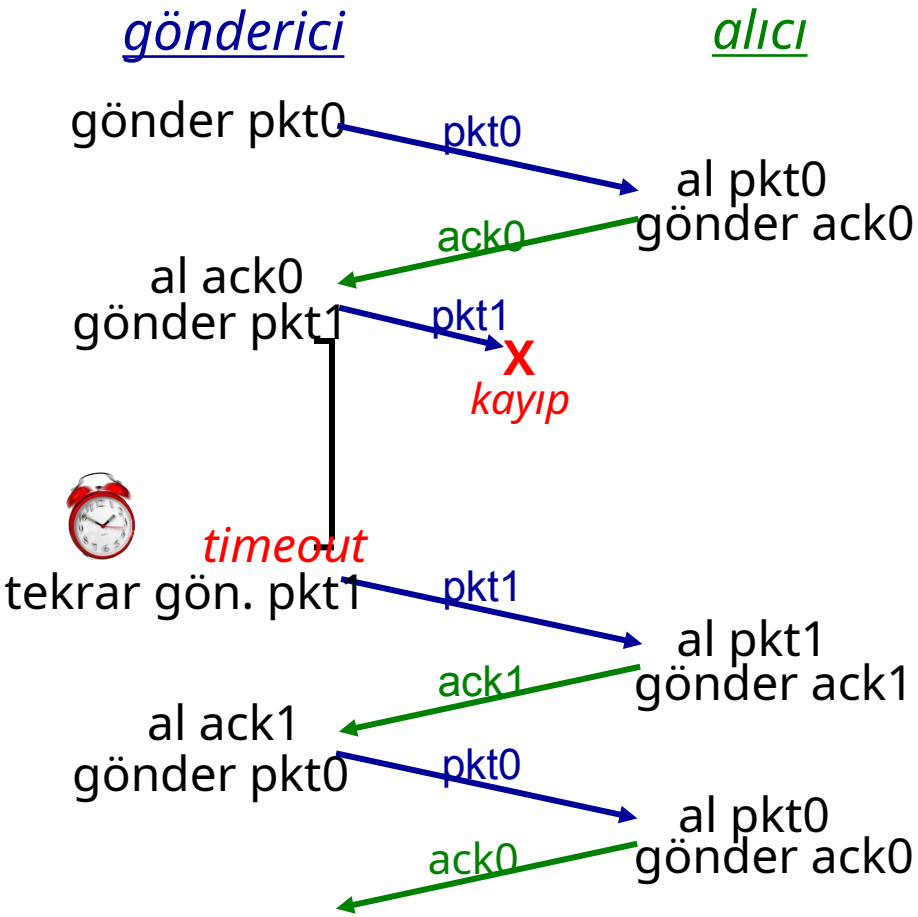
rdt3.0 sender



rdt3.0 örneği

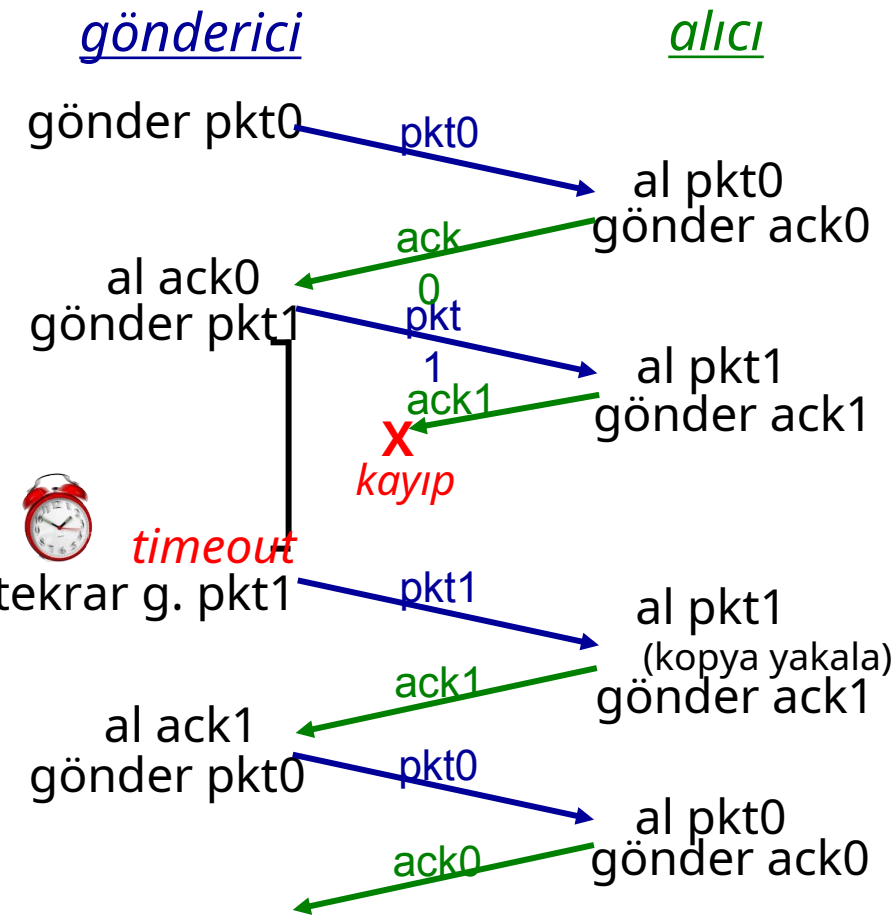


(a) kayıp yok

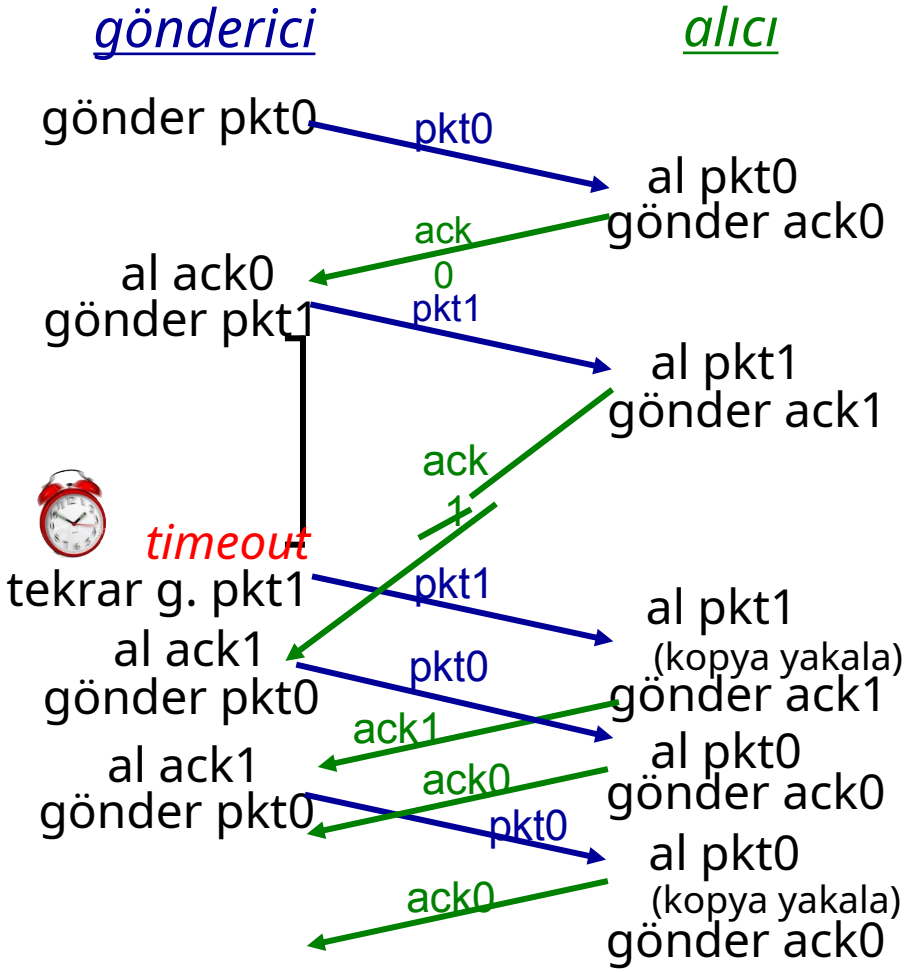


(b) paket kaybı

rdt3.0 örneği



(c) ACK kaybı



(d) zamansız timeout/ gecikmiş ACK

rdt3.0 performansı

- rdt3.0 doğru çalışır, ancak performansı gerçekten kötüdür.
- Ör: 1 Gb/sn bant, 15 ms yayılma gecikmesi, 8000 bit paket:

$$D_{iletim} = \frac{L}{R} = \frac{8000\ bit}{10^9\ bit/sn} = 8\ microsecs$$

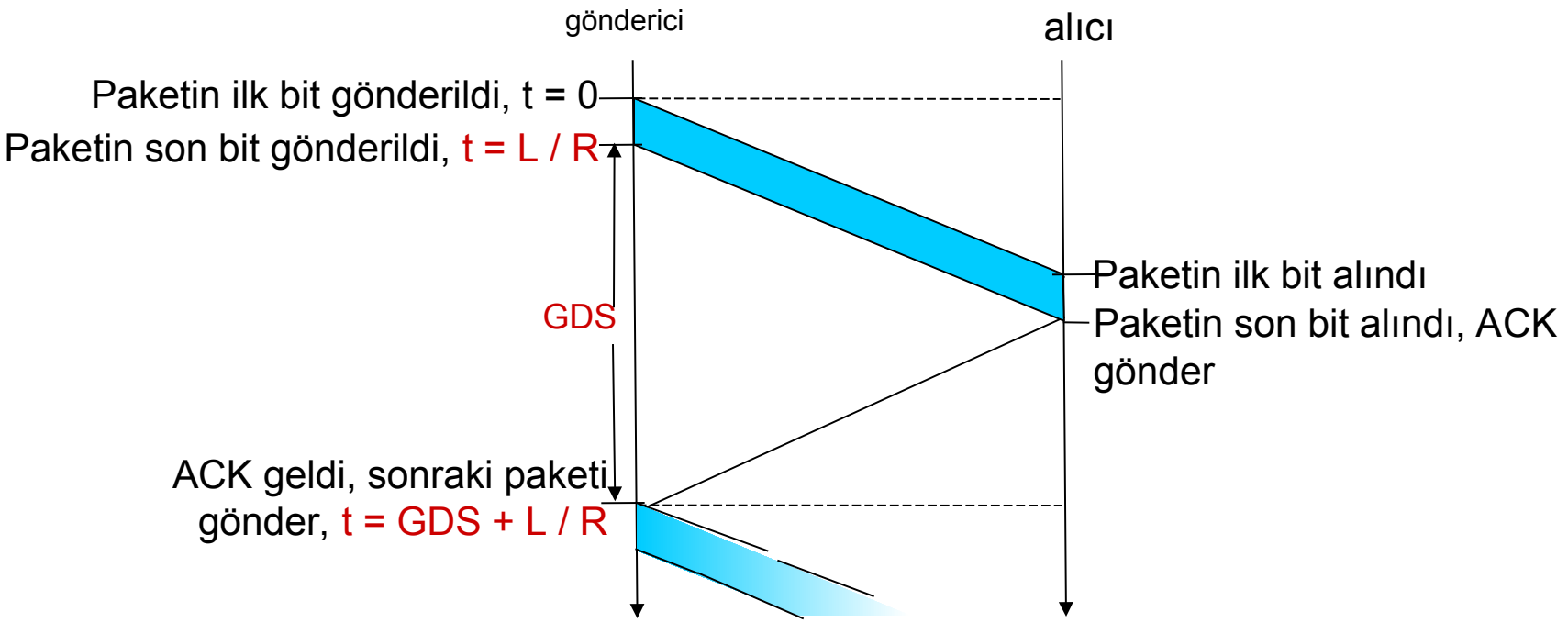
  - $U_{gönderici}$: *kullanım* – göndericinin gönderme ile meşgul olduğu süre oranı

$$U_{gönderici} = \frac{L/R}{GDS + L/R} = \frac{0.008}{30.008} = 0.00027$$

    - Eğer GDS=30 ms, 1KB pkt her 30 msec: 33kB/saniye işhacmi 1 Gb/sn link üzerinde

- Görüldüğü üzere ağ protokolü fiziksel kaynakların kullanımını sınırlandırıyor.

rdt3.0: dur-ve-bekle operasyonu

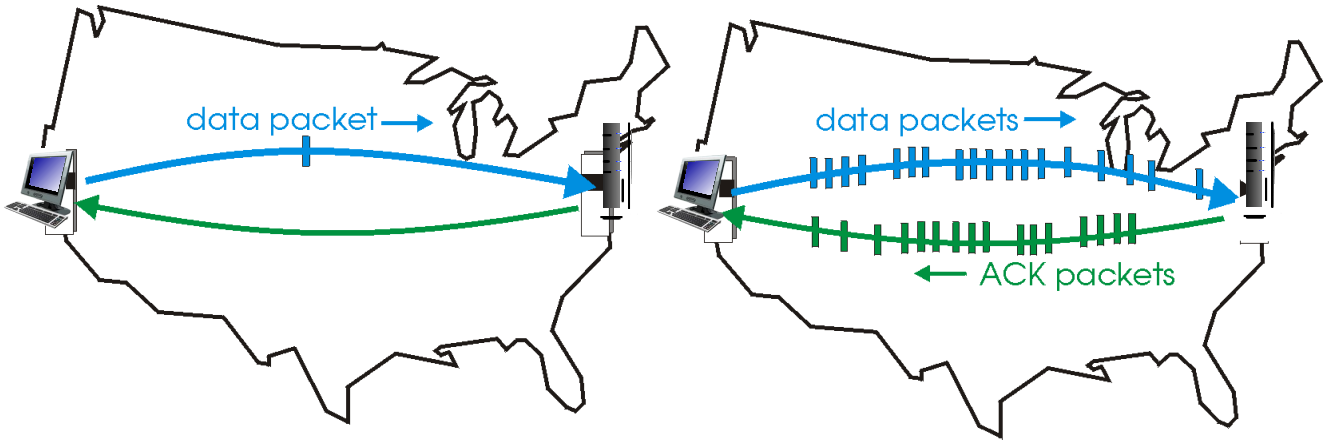


$$U_{gönderici} = \frac{L/R}{GDS+L/R} = \frac{0.008}{30.008} = 0.00027$$

Arka arkaya gönderen (İng: pipelined) protokoller

pipelining: gönderici çoklu, "henüz ulaşmamış", ACK-almamış paket gönderimine izin verir.

- Sıra sayısı miktarı artırılmalıdır.
- Gönderici ve alıcıda önbellekleme yapılmalıdır.

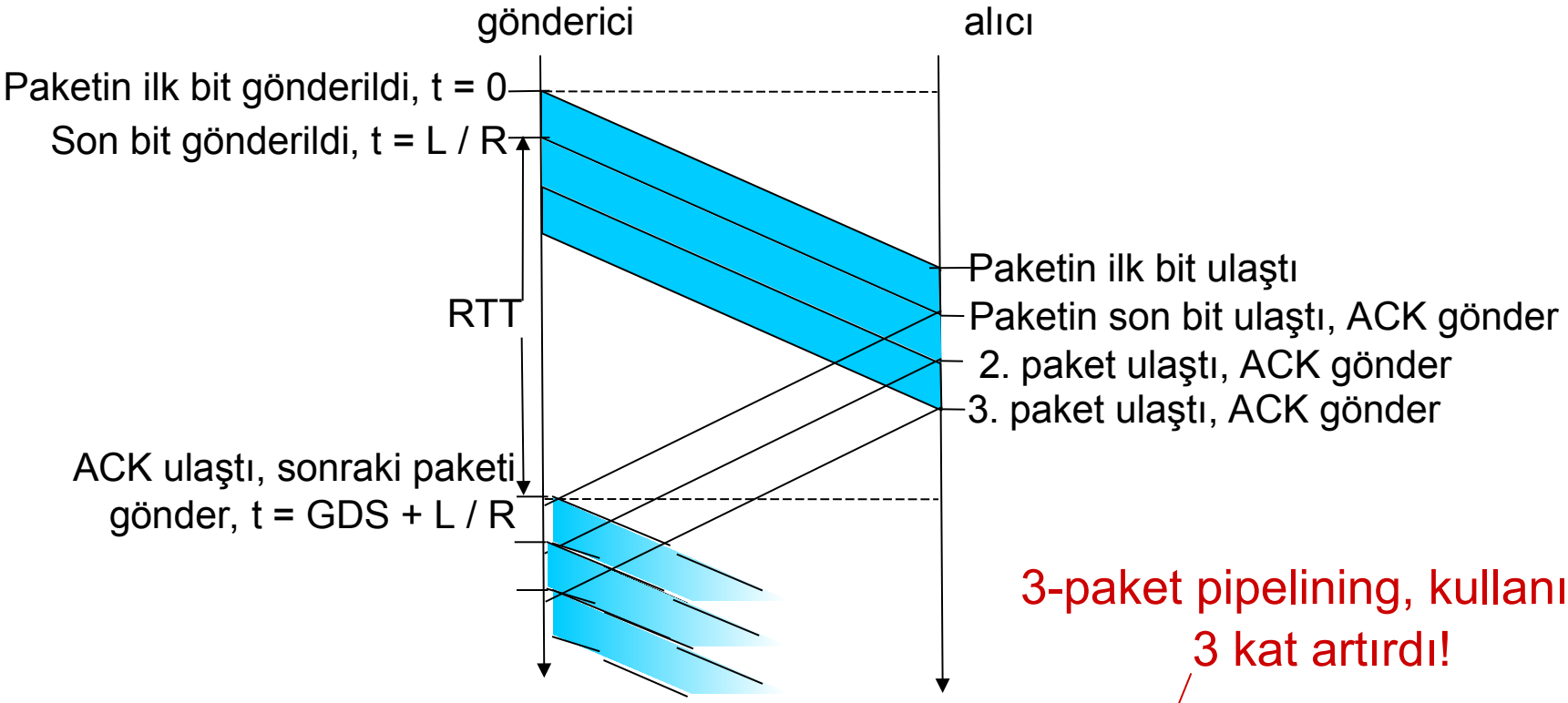


(a) a stop-and-wait protocol in operation

(b) a pipelined protocol in operation

- Arka arkaya gönderen protokollerin iki türü vardır: *geri-Git-N (İng: go-Back-N), seçici tekrar (İng: Selective repeat)*

Pipelining: yüksek kullanım



3-paket pipelining, kullanımı 3 kat artırdı!

$$U_{gönderici} = \frac{3L/R}{GDS+L/R} = \frac{0.024}{30.008} = 0.00081$$

Pipelined protokoller

Geri-git-N:

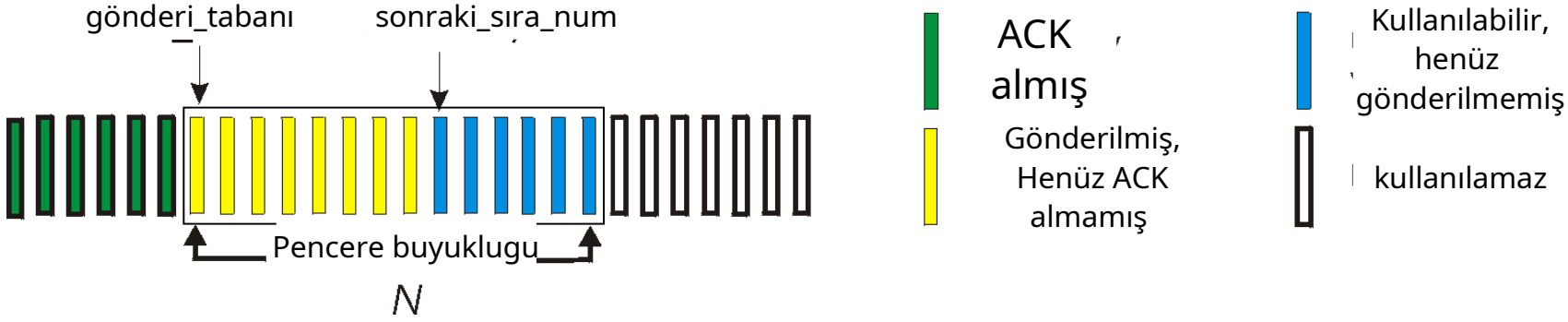
- Gönderici N taneye kadar ACK'lenmemiş paketi gönderebilir.
- Alıcı sadece toplu ACK gönderir.
  - Arada boşluk olursa ACK göndermez.
- Gönderici ACK gönderilmemiş en eski paket için kronometre çalıştırır.
  - Bekleme süresi geçtiğinde, ACK gönderilmemiş tüm paketler tekrar gönderilir.

Seçici tekrar:

- Gönderici N taneye kadar ACK'lenmemiş paketi gönderebilir.
- Alıcı her paket için bireysel ack gönderir.
- Gönderici her ACK gönderilmemiş paket için kronometre çalıştırır.
  - Zaman geçtiğinde, sadece zamanı geçen ACK gönderilmemiş paketi tekrar gönderir.

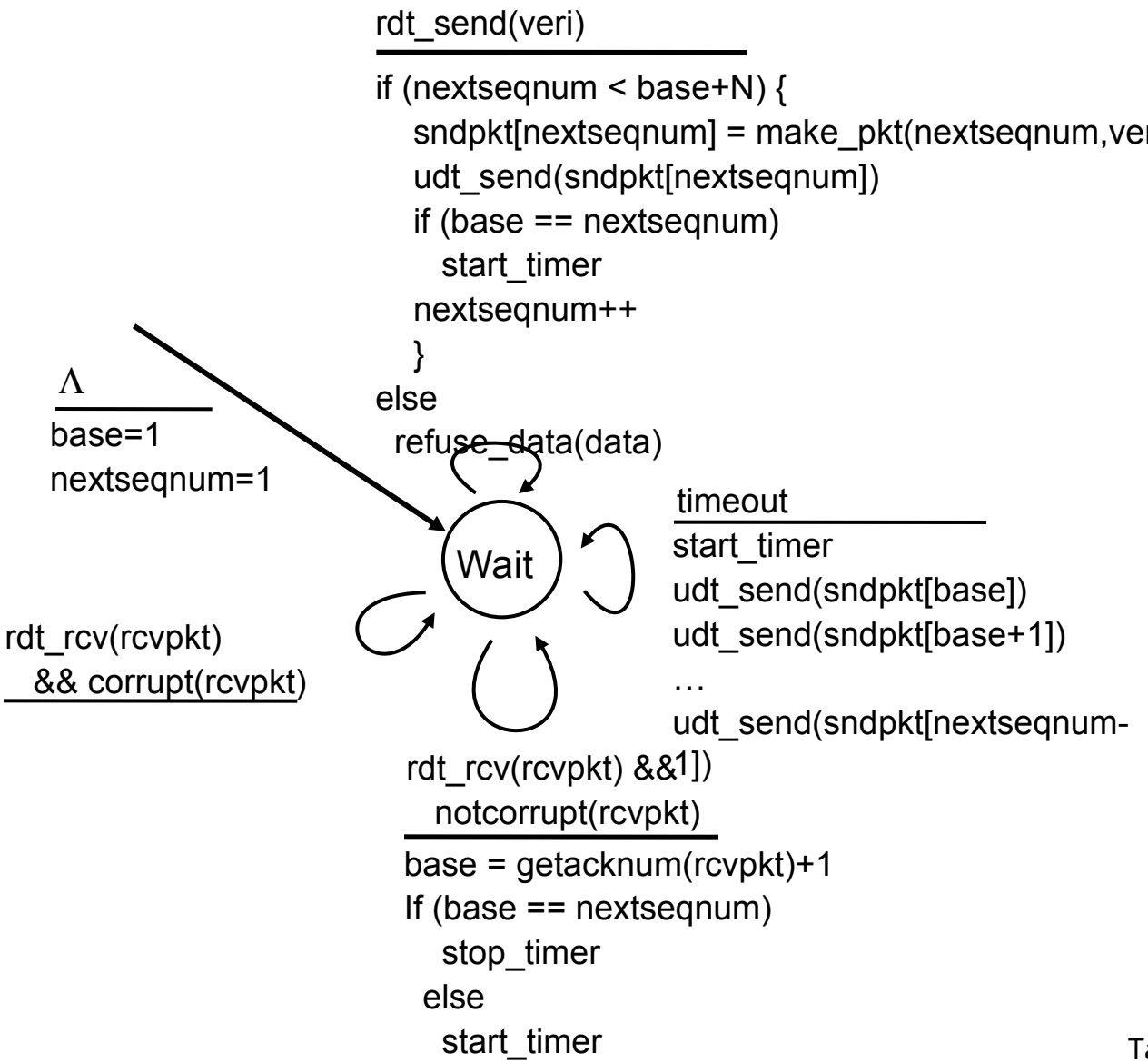
Geri-Git-N: gönderici

- Paket başlığında k-bit sıra #
- N büyüklüğünde “pencere” kadar, birbirini takip eden ACK almamış paket gönderilebilir.

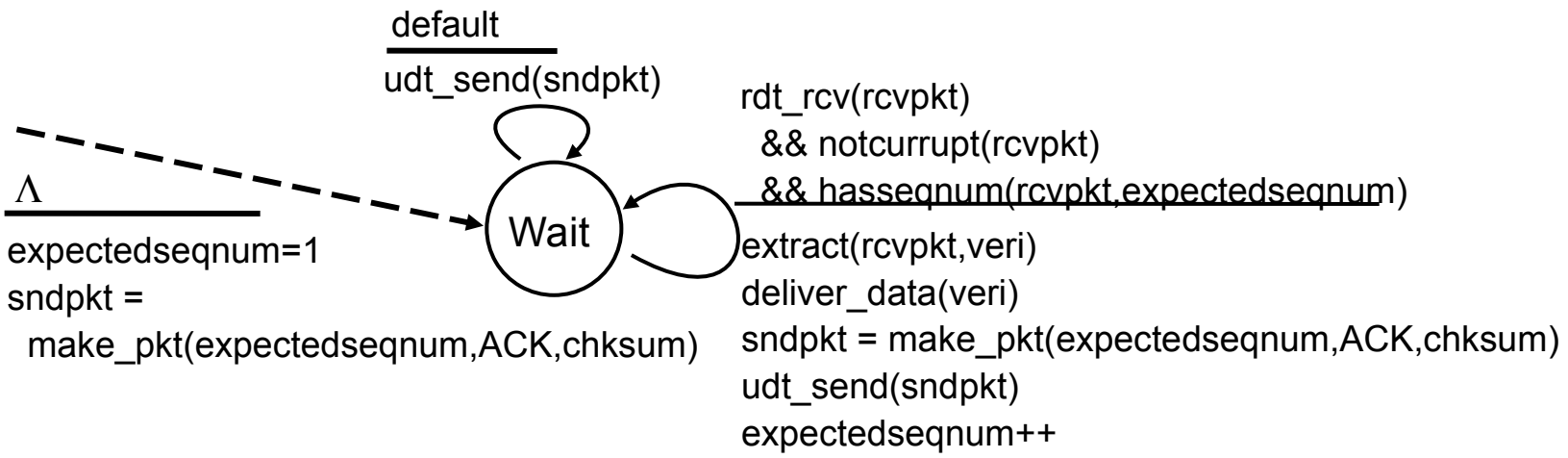


- ACK(n): sıra # n dahil, bütün paketler için ACK gönderir - *“toptan ACK”*
  - Kopya ACK alınabilir (alıcıya bakınız).
- Henüz ACK almamış en eski paket için zaman tutulur.
- *timeout(n):* paket n ve pencere içerisindeki daha büyük sıra # sahip bütün paketler tekrar yollanır.

GGN: göndericinin genişletilmiş FSM



GGN: alıcı genişletilmiş FSM



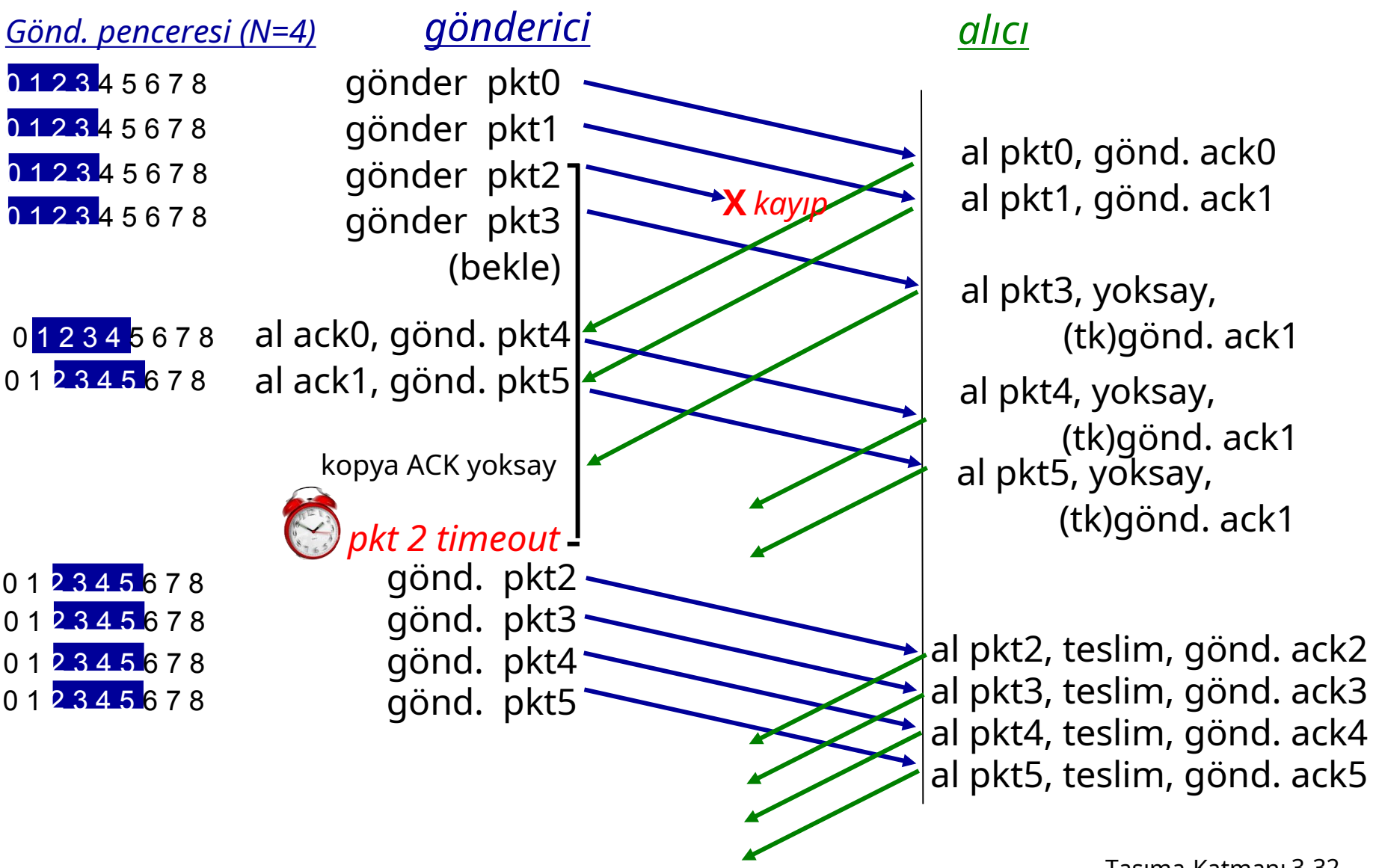
Sadece-ACK: daima en yüksek sıra # sahip sıralı, düzgün alınmış paket için ACK gönderilir.

- Kopya ACK gönderilebilir.
- sadece **`expectedseqnum`** hatırlanması yeterlidir.

❖ Sıra dışı pkt:

- Görmezden gel (önbellekleme yapma): *alıcı önbelleği yok!*
- En yüksek sıra # sahip, sıralı gelmiş paket için tekrar ACK gönder.

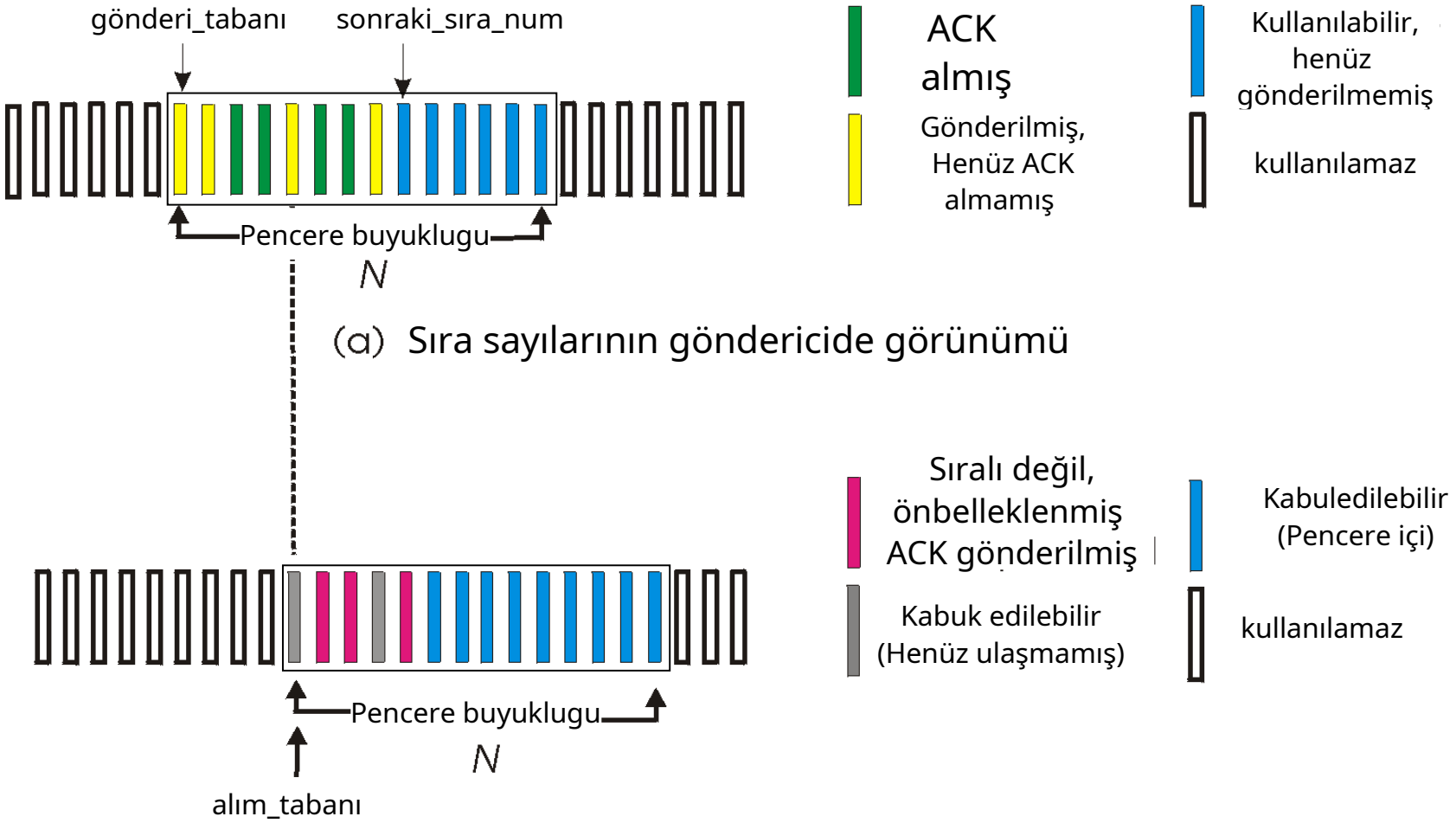
GGN örneği


Gönd. penceresi (N=4)
gönderici
alıcı
0 1 2 3 4 5 6 7 8
gönder pkt0
gönder pkt1
gönder pkt2
gönder pkt3
(bekle)
X kayıp
al pkt0, gönd. ack0
al pkt1, gönd. ack1
al pkt3, yoksay,
(tk)gönd. ack1
0 1 2 3 4 5 6 7 8
al ack0, gönd. pkt4
0 1 2 3 4 5 6 7 8
al ack1, gönd. pkt5
al pkt4, yoksay,
(tk)gönd. ack1
kopya ACK yoksay
al pkt5, yoksay,
(tk)gönd. ack1
pkt 2 timeout
0 1 2 3 4 5 6 7 8
gönd. pkt2
0 1 2 3 4 5 6 7 8
gönd. pkt3
0 1 2 3 4 5 6 7 8
gönd. pkt4
0 1 2 3 4 5 6 7 8
gönd. pkt5
al pkt2, teslim, gönd. ack2
al pkt3, teslim, gönd. ack3
al pkt4, teslim, gönd. ack4
al pkt5, teslim, gönd. ack5

Seçici tekrar

- Alıcı düzgün alınmış paketlerin her biri için ACK gönderir.
  - Gerektiği gibi paketleri önbellekler, bu sayede paketler üst katmana sıralı teslim edilir.
- Gönderici sadece ACK alınmayan paketleri tekrar gönderir.
  - ACK almamış her paket için kronometre çalıştırılır.
- Gönderici penceresi
  - *N* tane birbirini takip eden sıra #.
  - Gönderilen ancak ACK almamış sıra # sınırlar.

Seçici tekrar: gönderici, alıcı pencereleri



(a) Sıra sayılarının göndericide görünümü

(b) Sıra sayılarının alıcıda görünümü

Seçici tekrar

gönderici

Ustten veri gelince:

- Bir sonraki uygun sıra # pencere içinde ise, paketi gönder.

timeout(n):

- pkt n tekrar gönder, kronometreyi tekrar çalıştır.

ACK(n) [sendbase,sendbase+N] içinde:

- Pkt n alındı olarak işaretle
- n en küçük sıra numarasına sahip ACK almamış paket ise, pencere taban değerini bir sonraki en küçük ACK almamış sıra numarasına ilerlet.

alıcı

[alım_tabanı, alım_tabanı+N-1] içinde pkt

- gönder ACK(n)
- Sıra dışı: önbellekle
- Sıralı: teslim et (ayrıca diğer sıralı pktleri teslim et), pencereyi bir sonraki henüz alınmayan sıra numarasına ilerlet.

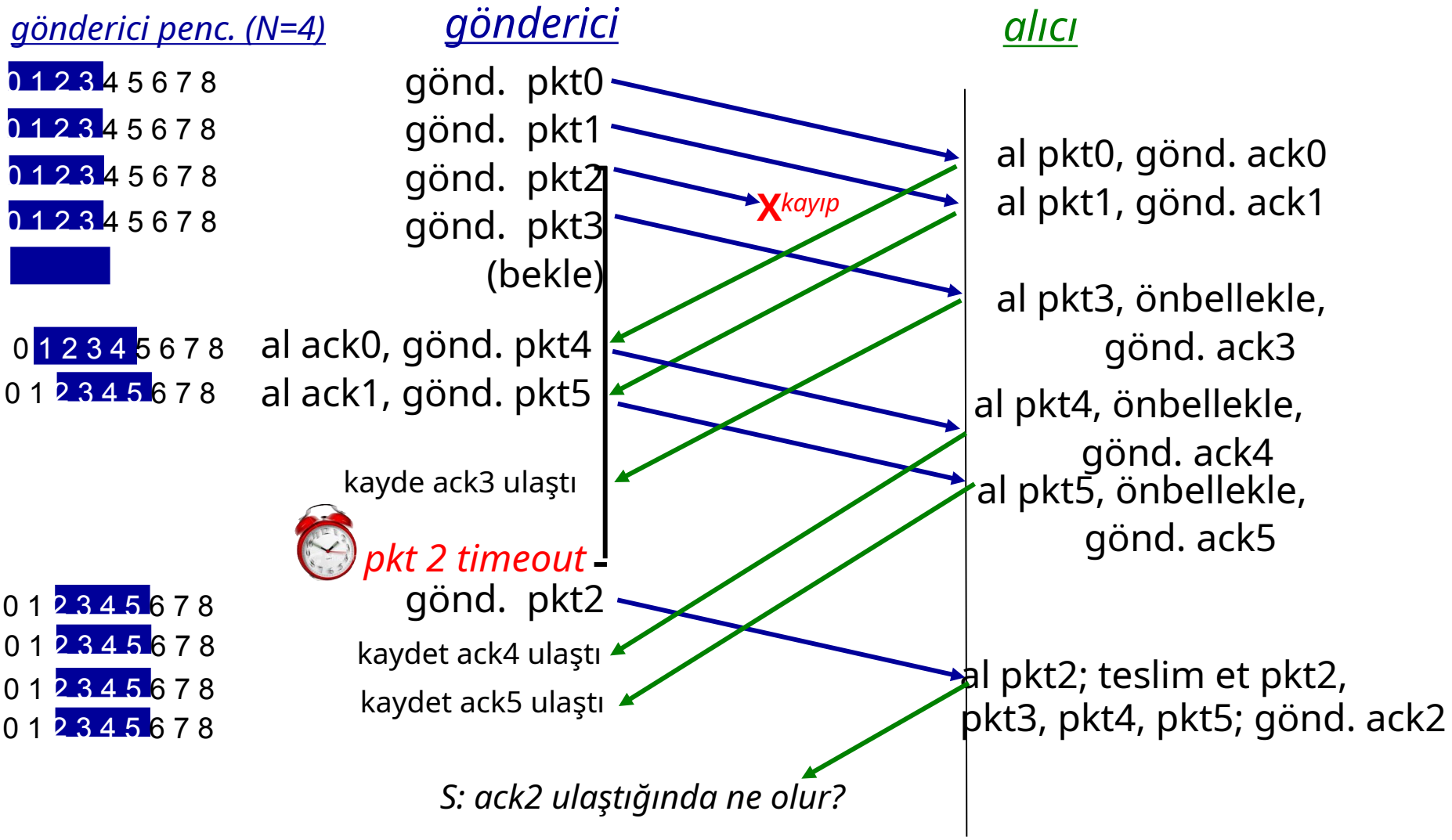
[alım_tabanı-N,alım_tabanı-1] içinde paket n

- ACK(n)

aksi takdirde:

- yoksay

Seçici tekrar örneği



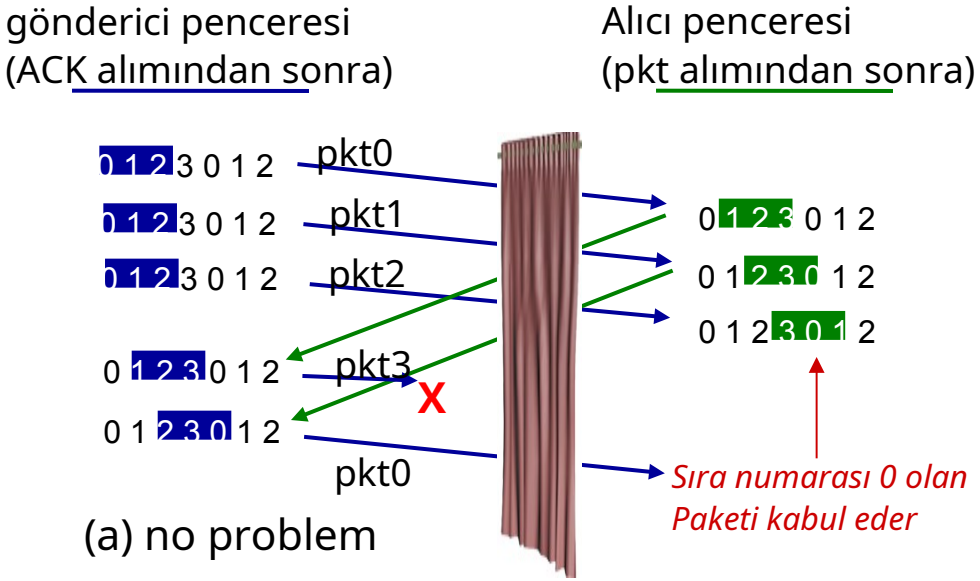
Seçici tekrar: ikilem

örnek:

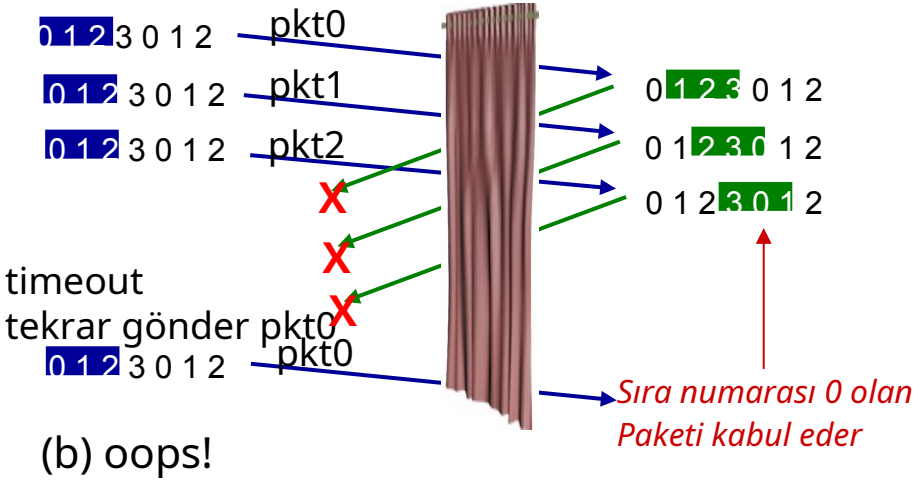
- sıra #: 0, 1, 2, 3
- pencere genişliği=3

- Alıcı iki farklı senaryoda fark görmez.
- Kopya veri yeni olarak kabul edildi (b).

S: (b) durumunu engellemek için sıra # sayısı ile pencere büyüklüğü arasındaki ilişki ne olmalıdır?



Alıcı karşı tarafta ne olduğunu göremez.
Alıcı davranışı iki durumda aynıdır.
Bir şeyler yanlış gidiyor.



Bölüm 3: konular

3.1 taşıma-katmanı hizmetler

3.2 çoklama ve çoklama çözme

3.3 bağlantısız taşıma: UDP

3.4 güvenilir veri transferi prensipleri

3.5 bağlantı-odaklı taşıma: TCP

- Segment yapısı
- Güvenilir veri transferi
- Akış kontrolü
- Bağlantı yönetimi

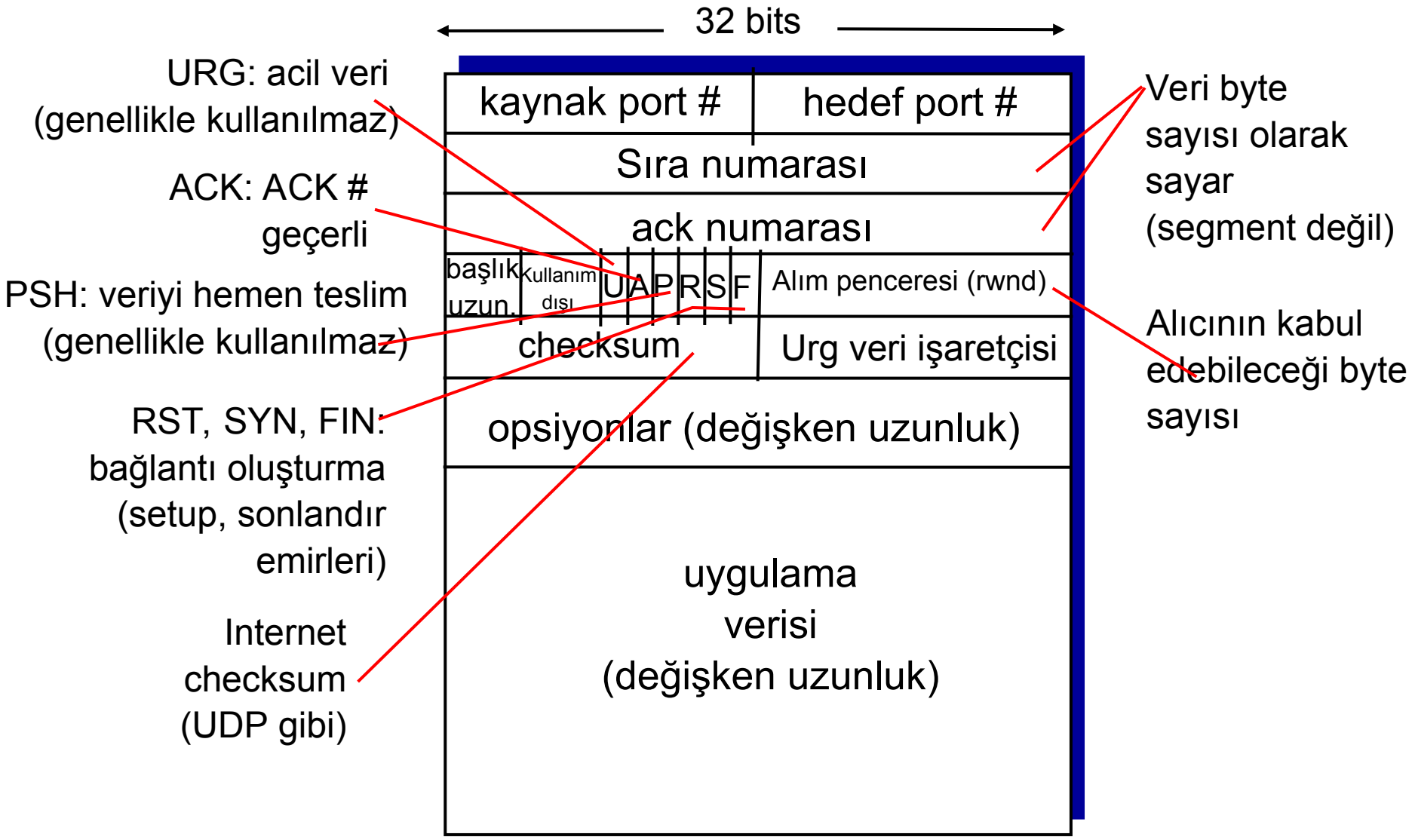
3.6 sıkışma kontrolü prensipleri

3.7 TCP sıkışma kontrolü

TCP: Genel bakış RFCs: 793,1122,1323, 2018, 2581

- uçtan-uca:
  - Bir gönderici, bir alıcı.
- Güvenilir, sıralı byte yayını:
  - “Mesaj sınırı” yoktur.
- pipelined:
  - TCP sıkışma and akış kontrol pencere büyüklüğünü belirler.
- tam iki-yönlü veri:
  - Aynı bağlantıda çift yönlü veri akışı.
  - MSB: maksimum segment büyüklüğü
- bağlantı-odaklı:
  - El sıkışma (kontrol mesajları değiş tokuşu) gönderici, alıcı durumunu veri gönderimi öncesi tanımlar.
- Akış kontrollü:
  - Gönderici aşırı veri ile alıcıyı boğmaz.

TCP segment yapısı



TCP numaraları, ACK mesajları

sıra numaraları:

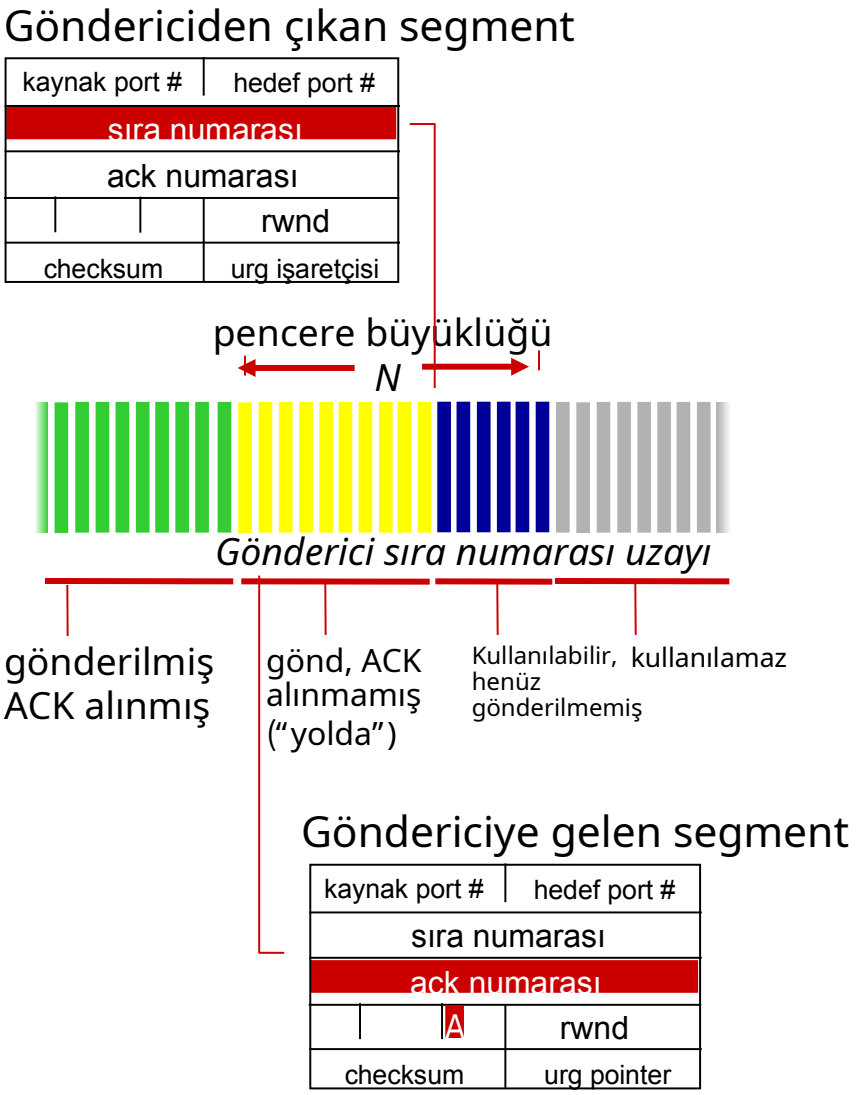
- Segment verisindeki ilk byte için yayın sırası.

acknowledgements:

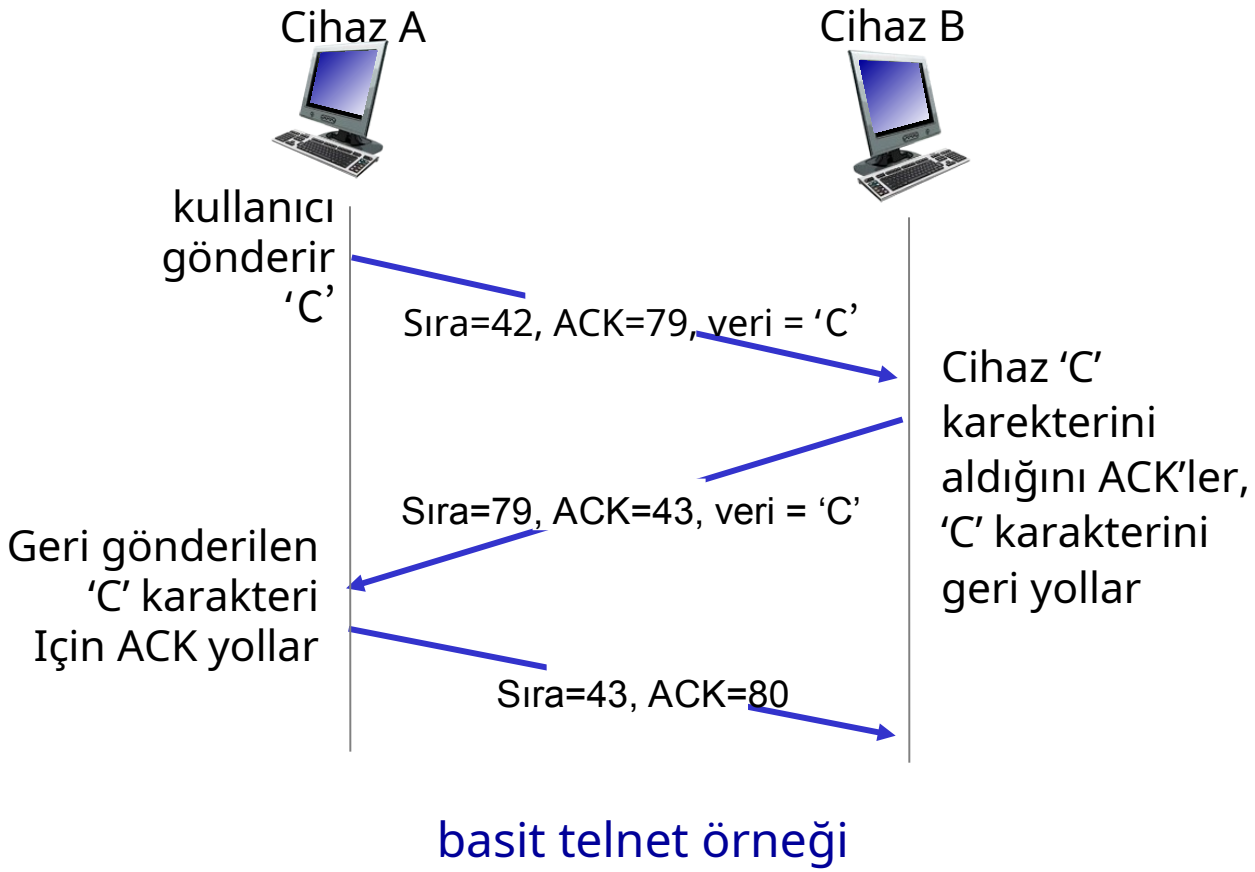
- Karşı taraftan bir sonraki beklenen byte sıra numarası.
- Toplu ACK

S: Alıcı sıra dışında gelen segmentlere ne yapar?

- A: TCP bunu tanımlamamıştır, programcıya bırakılmıştır.



TCP sıra numarası, ACK



basit telnet örneği

TCP gidiş dönüş süresi, timeout

S: TCP timeout süresini nasıl seçelim?

- GDS'den uzun olmalı.
  - Ama GDS değişken.
- *Çok kısa:* vakitsiz timeout, gereksiz tekrar göndermeler.
- *Çok uzun:* segment kaybına geç reaksiyon.

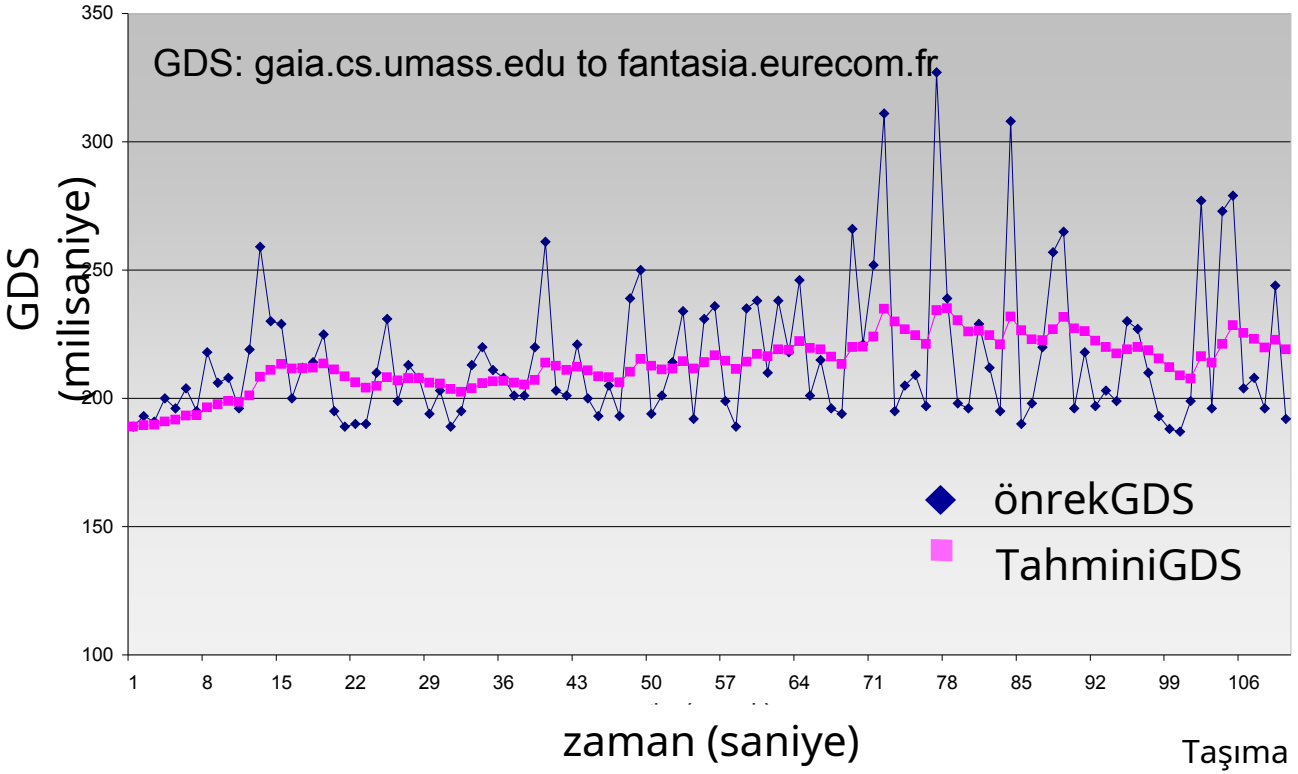
S: GDS değerini nasıl kestirebiliriz?

- **`ÖrnekGDS`**: segment gönderiminden ACK alımına kadar geçem süreyi ölçeriz.
  - Tekrar gönderimleri görmezden geliriz.
- **`ÖrnekGDS`** değişkendir, daha "düzgün" bir tahmin isteriz.
  - Yakın zamanda hesaplanan değerlerin ortalaması alırız, sadece en son **`ÖrnekGDS`** kullanmayız.

TCP gidiş dönüş süresi, timeout

TahminiGDS = (1- α)*TahminiGDS + α*ÖrnekGDS

- Üstel ağırlıklı hareketli ortallama (İng:exponential weighted moving average)
- Eski gözlemlerin etkisi üstel olarak azalır.
- Çoklukla kullanılan değer: $\alpha = 0.125$



TCP gidiş dönüş süresi, timeout

- timeout süresi: **`TahminiGDS`** üstüne "güvenlik payı"
  - yüksek değişim gösteren **`TahminiRTT ->`** geniş güvenlik payı
- ÖrnekGDS sapmasını TahminiGDS üzerinden hesapla:

`DevGDS = (1-β)*DevGDS +`
`β*|ÖrnekGDS-TahminiGDS|`

`(genelde, β = 0.25)`

`TimeoutSüresi = TahminiGDS + 4*DevGDS`

Tahmini GDS

"güvenlik payı"