Bilgisayar Programcılığı
Uzaktan Eğitim Programı

e-BİLG 121 AĞ TEKNOLOJİLERİNİN TEMELLERİ

Öğr. Gör. Bekir Güler

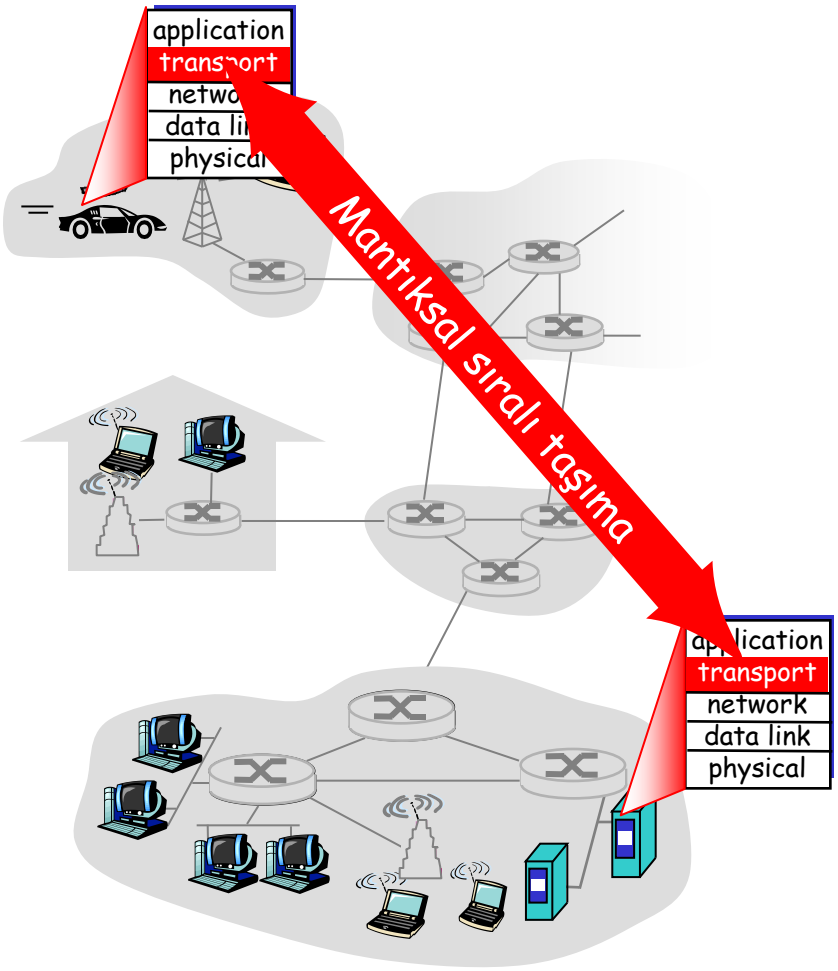
E-mail: bguler@fatih.edu.tr

Hafta 4: Taşıma (Transport) Katmanı

- 3.1 Taşıma katmanı hizmetleri
- 3.2 Multiplexing ve demultiplexing
- 3.3 Bağlantısız taşıma : UDP
- 3.4 Güvenli taşımanın temelleri
- 3.5 Bağlantı-yönelimli taşıma: TCP
  - segment yapısı
  - Güvenli veri taşıma
  - Akış denetimi
  - Bağlantı yönetimi
- 3.6 Tıkanıklık denetim temelleri
- 3.7 TCP tıkanıklık denetimi

3. 1. Taşıma hizmetleri ve protokolleri

- Faklı bilgisayarlarda çalışan uygulama işlemleri için *mantıksal iletişim* sağlar
- Taşıma protokolleri son kullanıcı sistemlerinde çalışırlar
  - Gönderen taraf: Uygulama mesajlarını segment'lere böler, ağ katmanına iletir
  - Alan taraf: Segment'lerden mesajı tekrar oluştur, uygulama katmanına iletir
- Birden fazla taşıma katmanı protokolleri vardır
  - Internet: TCP ve UDP



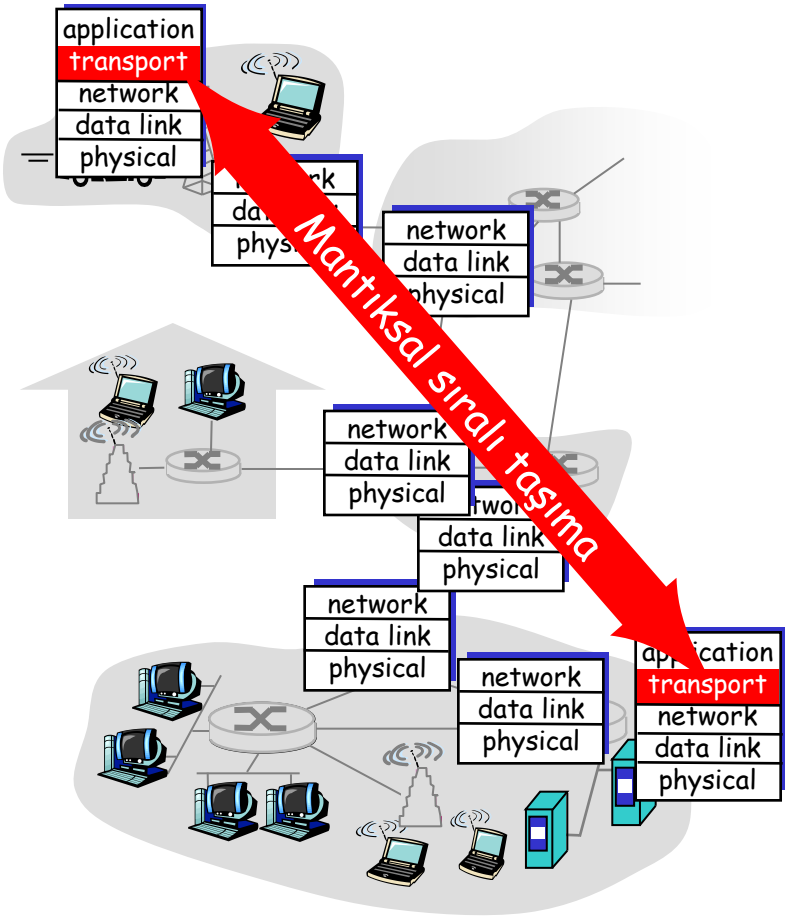
Taşıma ve ağ katmanı

- *Ağ katmanı:* Bilgisayarlar arasında mantıksal iletişim sağlar

- *Taşıma katmanı:* İşlemler(processes) arasında mantıksal iletişim sağlar

İnternet taşıma katmanı protokolleri

- Güvenilir, sıralı teslim (TCP)
  - Tıkanıklık kontrolü
  - Akış denetimi
  - Bağlantı kurulumu
- Garantisiz, sırasız teslim: UDP
  - Teslim garantisi yok
  - Sıralı teslim yok
  - Bant genişliği garanti edilir



3.2 Multiplexing(çoğullama) /demultiplexing(azaltma)

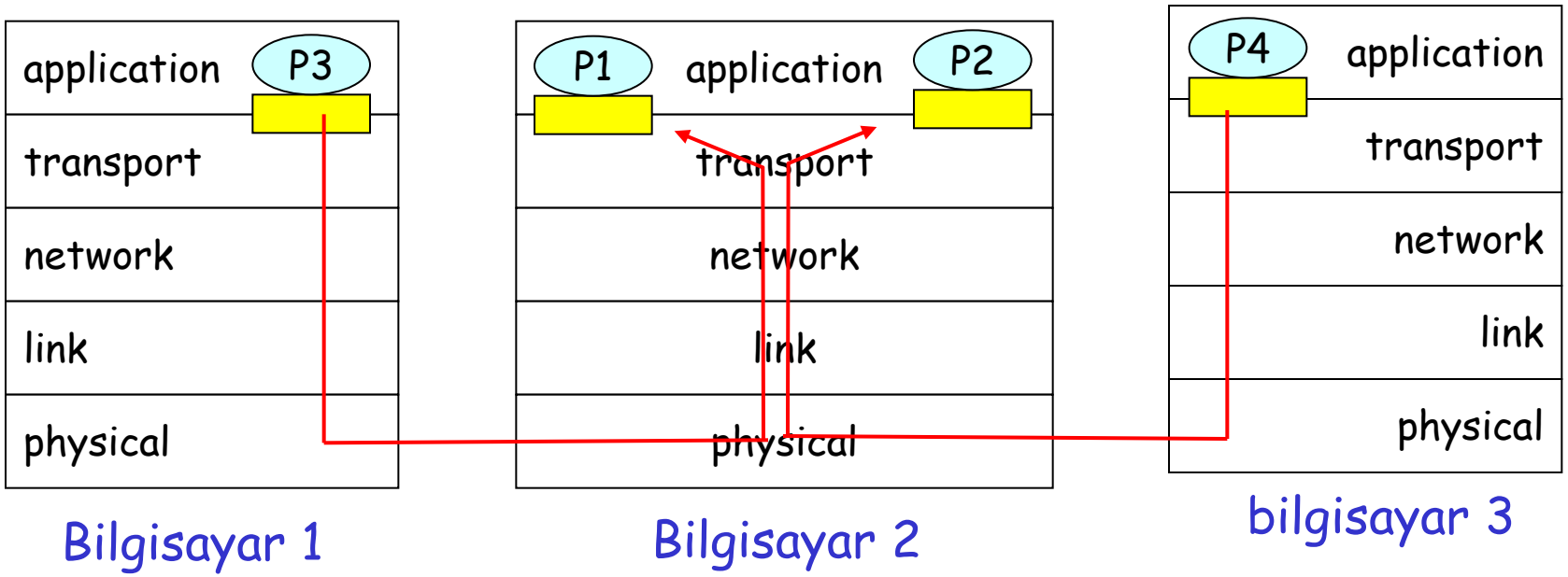
Gönderen bilgisayarda multiplexing:

Birden çok soketten verileri toplar, başlık ekleyerek verileri segment haline getirir

Alıcı bilgisayarda demultiplexing:

Alınan segment'leri doğru sokete teslim eder

= soket = işlem



Demultiplexing nasıl çalışır?

- Bilgisayar IP datagram'ları alır
  - Her datagram'ın kaynak IP adresi ve hedef IP adresi vardır
  - Her datagram bir segment taşır
  - Her bir segment'in kaynak ve hedef port numaraları vardır
- Bilgisayar, uygun soketle bağlantı kurmak için IP adreslerini ve port numaralarını kullanırlar

TCP/UDP segment formatı

Bağlantısız(UDP) demultiplexing

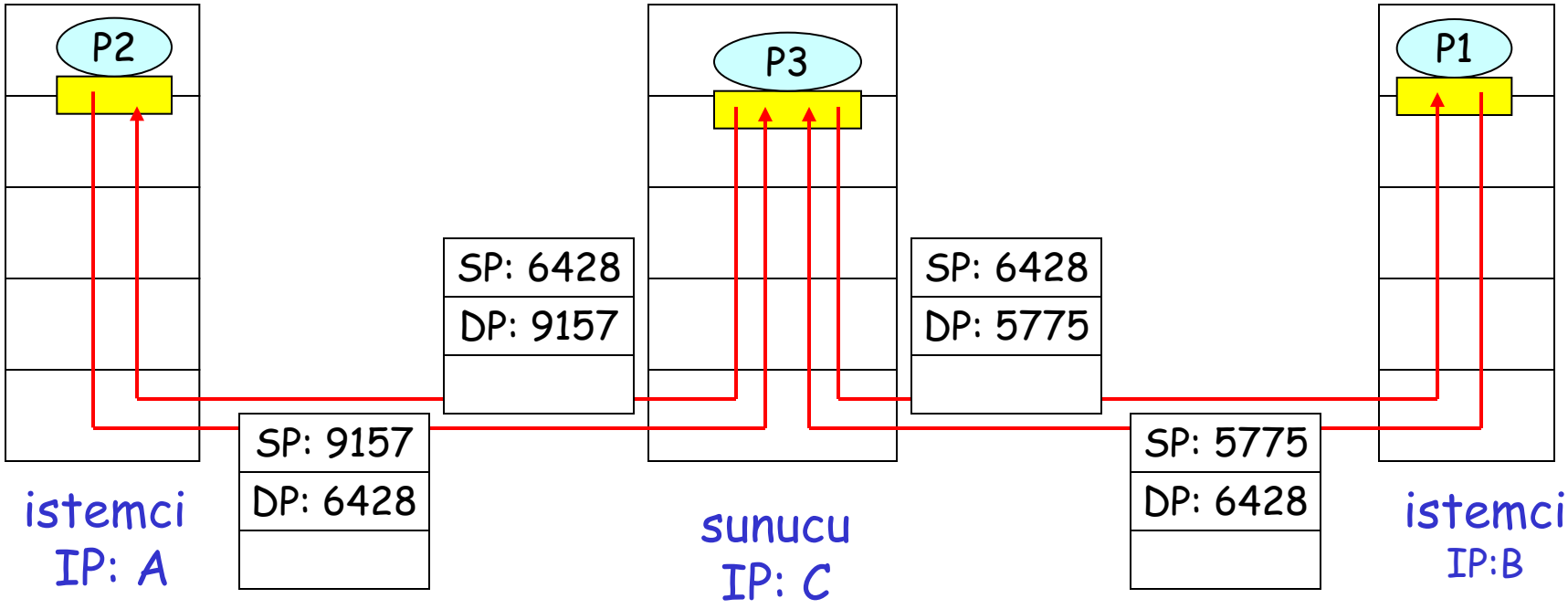
- IP adresi ve port numarası soket'i oluşturur:
- UDP soketleri iki alan ile tanınır:

(hedef IP adresi, hedef port numarası)

- Bilgisayar UDP segment'ini aldığında:
  - Segment'teki hedef port numarasını kontrol eder
  - UDP segment'ini o port numaralı sokete yönlendirir
- Farklı kaynak IP adresli datagram'lar ve/veya kaynak port numaraları aynı sokete yönlendirilir

Bağlantısız(UDP) demux (devamı)

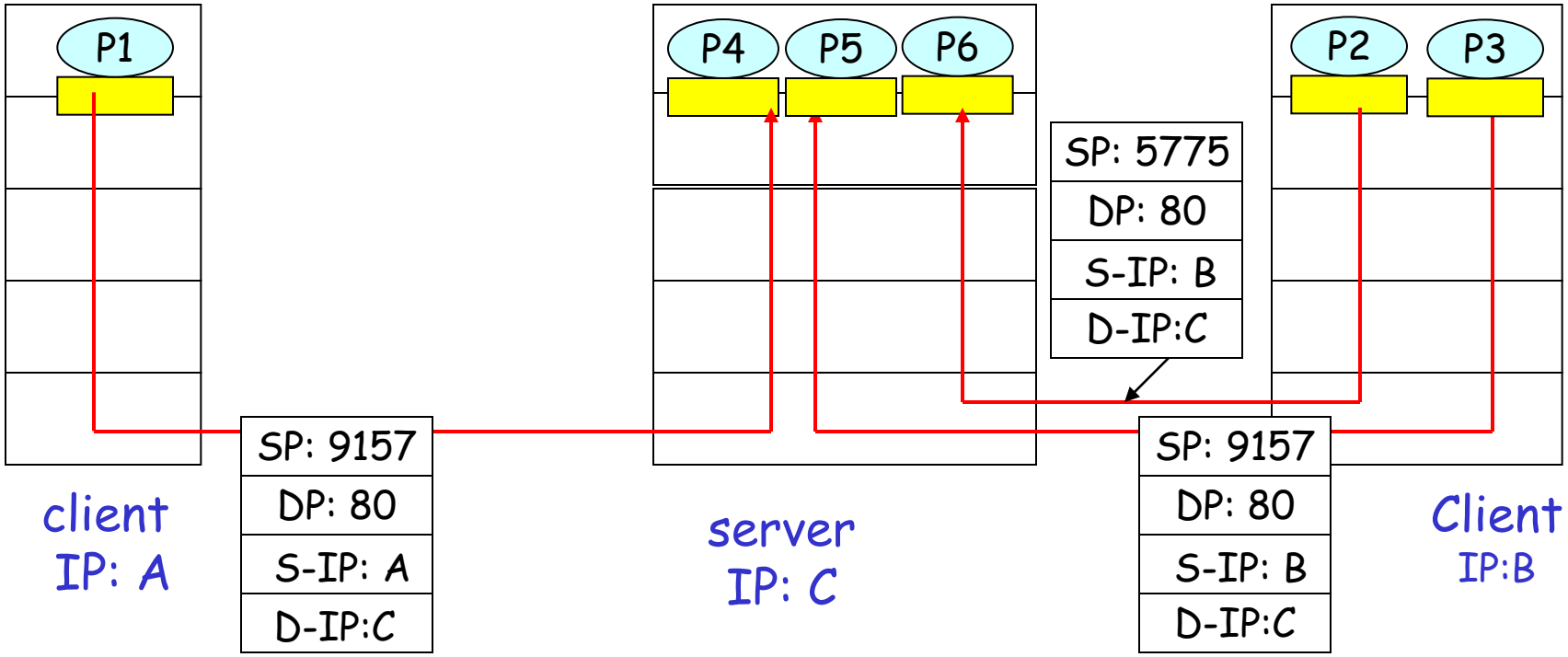
- SP: Source Port DP: Destination Port



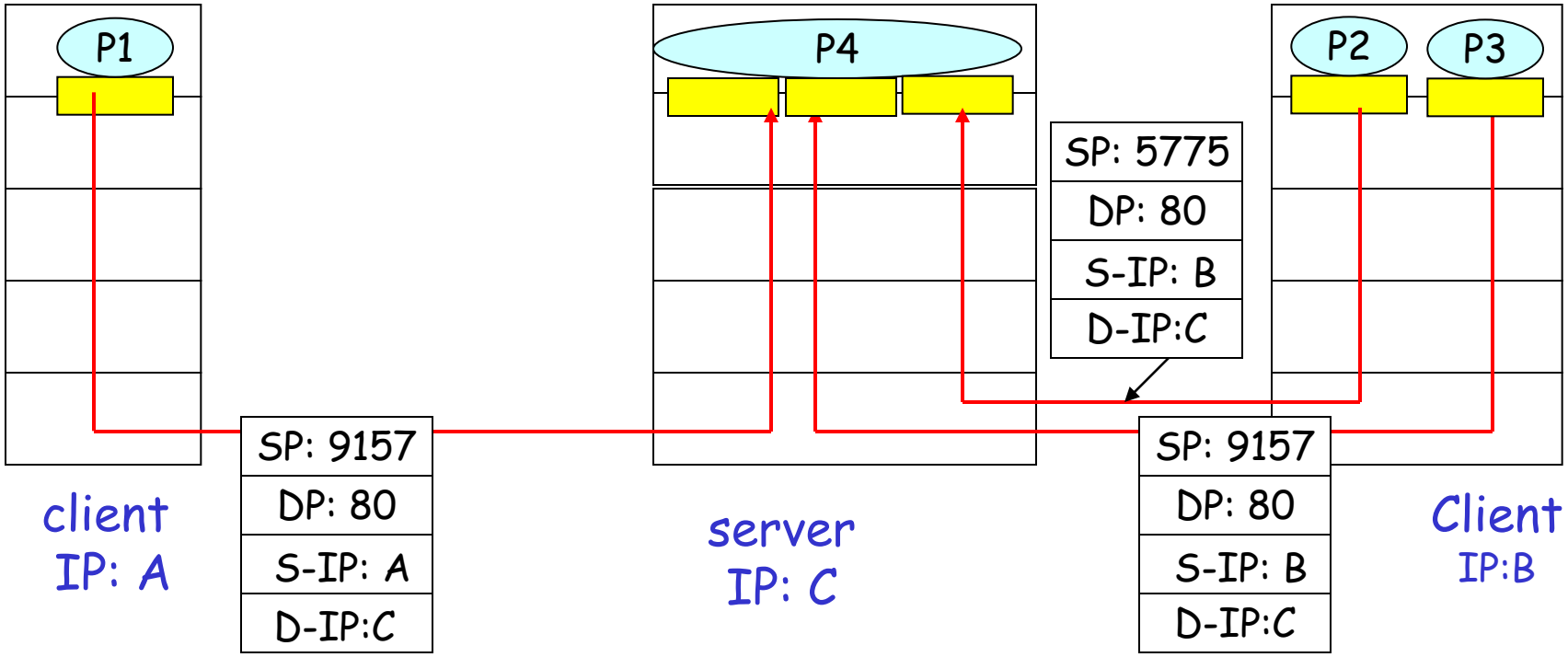
Bağlantıya dayalı (TCP) demux

- TCP soketi 4 özelliği ile tanımlanır:
  - Kaynak IP adresi
  - Kaynak port numarası
  - Hedef IP adresi
  - Hedef port numarası
- Alıcı bilgisayar, segment'i uygun sokete yönlendirmek için 4 değeri kullanır
- Sunucu bilgisayar aynı zamanda birçok TCP soketini destekleyebilir :
  - Her soket kendi 4 özelliği ile tanımlanır
- Web sunucuları her bağlanan istemci için farklı soket kullanır
  - Kalıcı bağlantısı olmayan HTTP, her istek için farklı soket kullanır

Bağlantıya dayalı (TCP) demux (devamı)



Bağlantıya dayalı (TCP) demux: Web Server



3.3 UDP: User Datagram Protocol

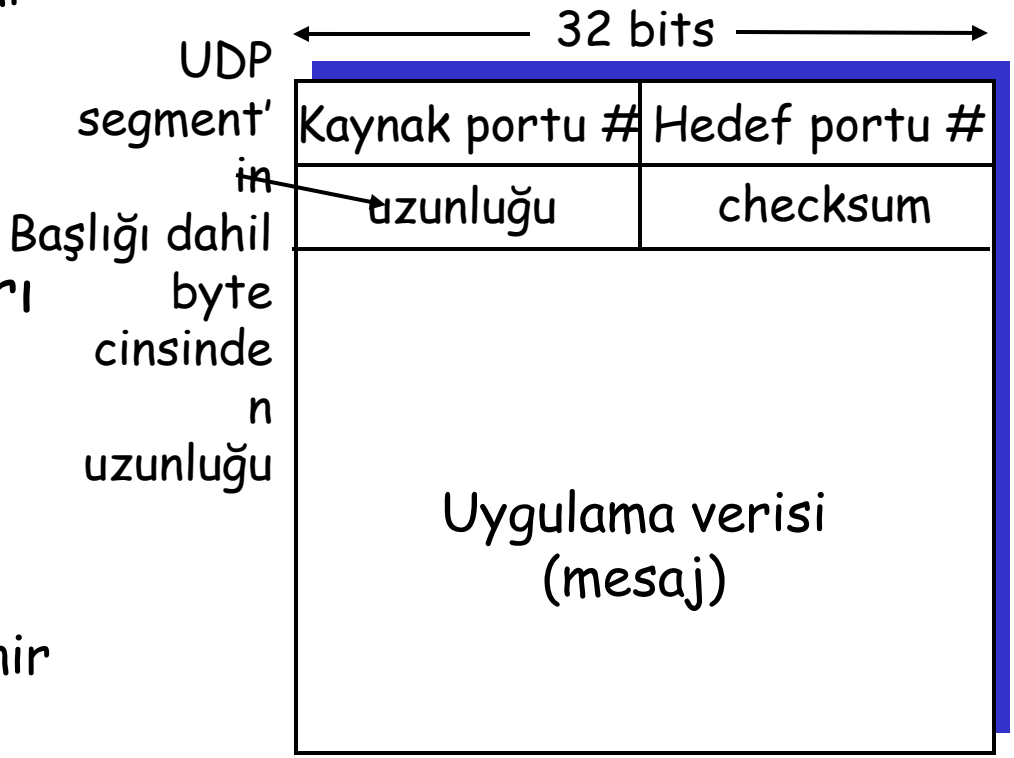
- Bağlantı gerektirmeyen İnternet taşıma katmanı protokolüdür
- UDP segment'leri kayıp olabilir ve teslimleri sırasız olabilir
- *Bağlantısız:*
  - UDP gönderen ve alan arasında önceden el sıkışma (handshaking) olmaz. Her UDP sergment'i bağımsız olarak ele alınır

Niçin UDP var?

- Bağlantı kurmaya gerek yok (zaman kaybına neden olabilir)
- Basit: Gönderici ve alıcıda bağlantı durumları tutulmaz
- Küçük segment başlığı
- Tıkanıklık kontrolüne ihtiyaç yok: UDP mümkün olduğu kadar hızlı gönderir

UDP: (devamı)

- Multimedya uygulamalarını yayınlamak için kullanılır
  - Kayba karşı dayanıklı
  - Hızı önemli
- Diğer UDP kullanımları
  - DNS
  - SNMP
- UDP üzerinden güvenli aktarım: Uygulama katmanında güvenlik eklenir
  - Uygulama ile hataları kurtarma!



UDP segment formatı

UDP sağlama toplamı (checksum)

Hedef: Aktarılan segment'deki hataları algılamak

Gönderici:

- Segment içeriklerine 16-bit tamsayı dizisi gibi davranır
- Sağlama toplamı (checksum): segment içeriklerinin toplamı
- Gönderici sağlama toplamı değerini UDP checksum alanına koyar

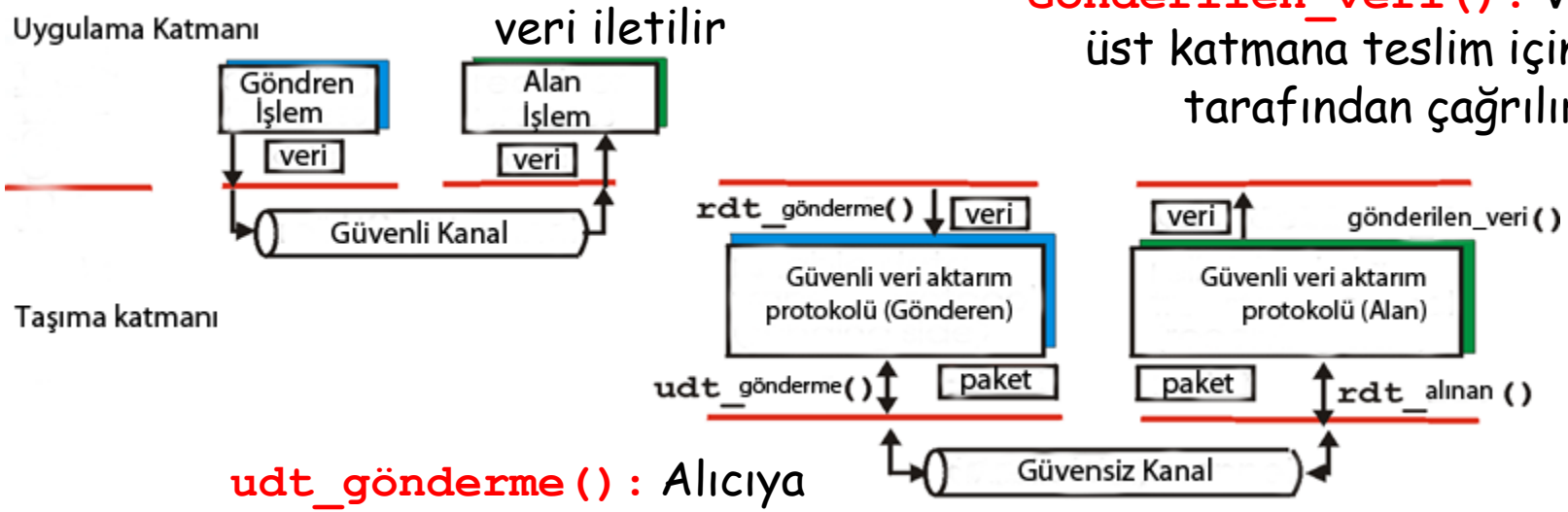
Alıcı:

- Alınan segment'lerin sağlama toplamı hesaplanır
- Hesaplanan checksum ve checksum alanındaki değer karşılaştırılır:
  - Aynı değilse-hata algılandı
  - Aynı ise-hata algılanmadı

3.4 Güvenilir veri aktarımı temelleri

`rdt_gönderme()`: Yukarıdan çağrılır, (örneğin, uygulama). Alıcı üst katmana teslim için veri iletilir

`Gönderilen_veri()`: Veriyi üst katmana teslim için `rdt` tarafından çağrılır



`udt_gönderme()`: Alıcıya paketleri güvenli olmayan kanal üzerinden iletmek için rdt tarafından çağrılır

`rdt_alınan ()`: Paket kanalın alıcı tarafına ulaştığında çağrılır

Rdt1.0: Güvenli kanal üzerinden güvenli veri aktarımı

- Veri aktarılan kanal tam güvenli bir kanaldır
  - Bit hataları yoktur
  - Paket kayıpları yoktur
- Gönderici ve alıcı:
  - Gönderici verileri güvenli kanaldan gönderir
  - Alıcı verileri güvenli kanaldan alır

Rdt2.0: Bit hatalı kanal

- Veri aktarılan kanal paket içindeki bitleri ters çevirebilir
  - Sağlama toplamı (checksum) hataları algılar
- *Soru*: Hatalar nasıl düzeltilir:
  - *Acknowledgements (ACKs):* Alıcı, gönderene paketlerin alındığı söyler
  - *Negative acknowledgements (NAKs):* Alıcı, gönderene paketin hatalı olduğunu söyler. Gönderen, NAK mesajı aldığı alıcılara paketleri tekrar gönderir
- `rdt2.0` yeni özelliği:
  - Hata algılama
  - Alıcının geribildirimi: Kontrol mesajları (ACK,NAK) alıcı->gönderen

rdt2.0 önemli bir kusuru!

ACK/NAK mesajları bozulursa ne olur?

- Gönderen, alıcıda ne olduğunu bilemez!
- Hemen gönderemez: Birden fazla paket gönderilmiş olabilir

Birden fazla göndermeyi engelleme:

- Gönderici, ACK/NAK bozulursa geçerli paketi yeniden gönderir
- Gönderici her pakete bir *segment numarası* ekler
- Alıcı birden fazla gelen paketleri atar

Durma ve bekleme

Gönderici bir paket Gönderir ve sonra cevap için bekler

rdt2.1

Gönderici:

- Pakete segment # (numarası) ekler
- ACK/NAK bozuk alınıp alınmadığını kontrol etmeli

Alıcı:

- Alınan paketin çift olup olmadığını kontrol etmeli
- Not: Alıcı, son ACK/NAK mesajlarının gönderen tarafından alınıp alınmadığını bilemez

rdt2.2: NAK içermeyen protokol

- Sadece ACK kullanarak rdt2.1 ile aynı görevi yapar
- Alıcı, NAK yerine son paketin başarılı alındığına dair ACK gönderir
  - Alıcı, ACK yapılan paketin segment numarasını açıkça belirtir
- Göndericide çift ACK, NAK gibi kabul edilir: *Geçerli paket yeniden gönderilir*

rdt3.0: Kanallarda hatalar ve kayıplar

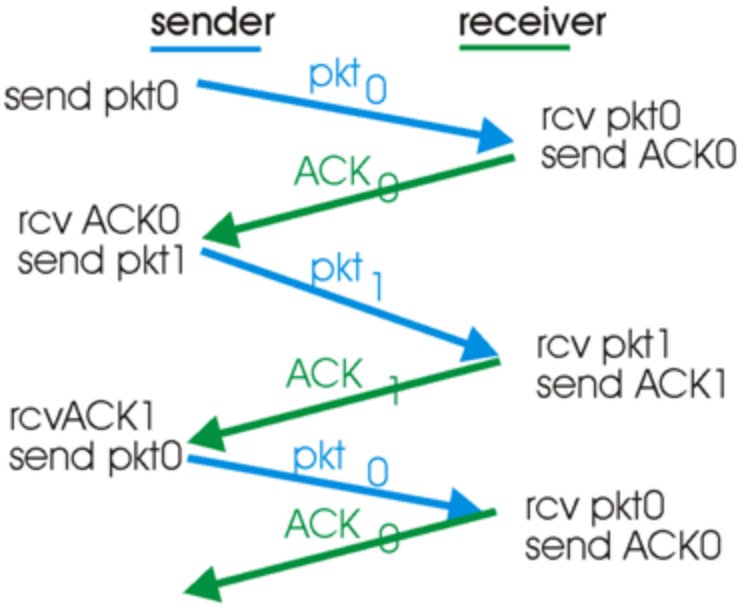
Yeni varsayım: İletim kanalı paketleri (veri veya ACKs) kayıp edebilir

- Sağlama toplamı, segment numarası, ACKs ve yeniden gönderme yeterli değildir

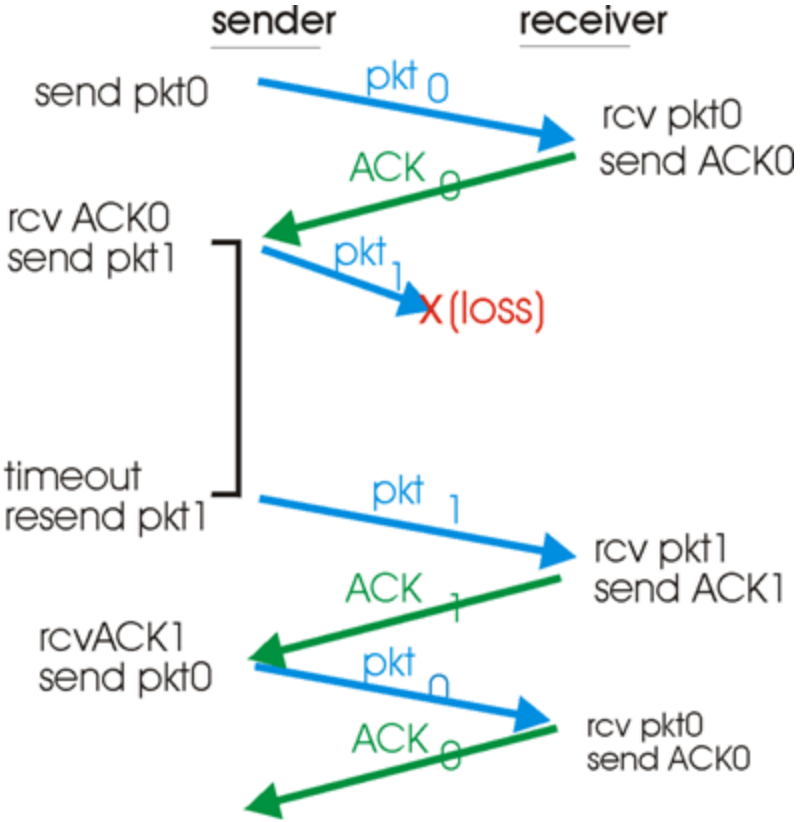
Yaklaşım: Gönderen, ACK mesajını makul bir süre bekler

- Makul sürede ACK mesajı alınmazsa paket yeniden gönderilir
- Eğer paket veya ACK mesajı gecikmişse:
  - Yeniden gönderme ile çift paket oluşur, segment numarası etkin olan kullanılır
  - Alıcı ACK yapılan paketin segment numarasını belirtmeli
- Zamanlayıcıyı azaltmak gerekir

rdt3.0 çalışması



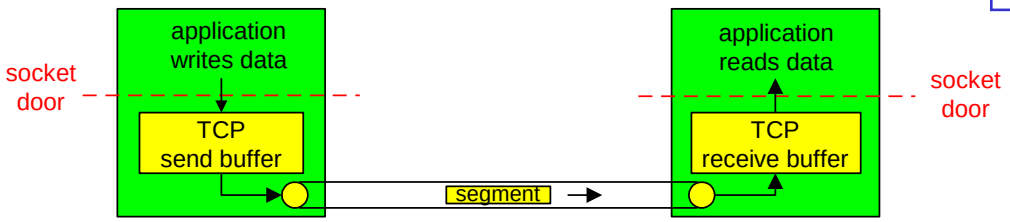
Kayıpsız aktarım



Kayıplı aktarım

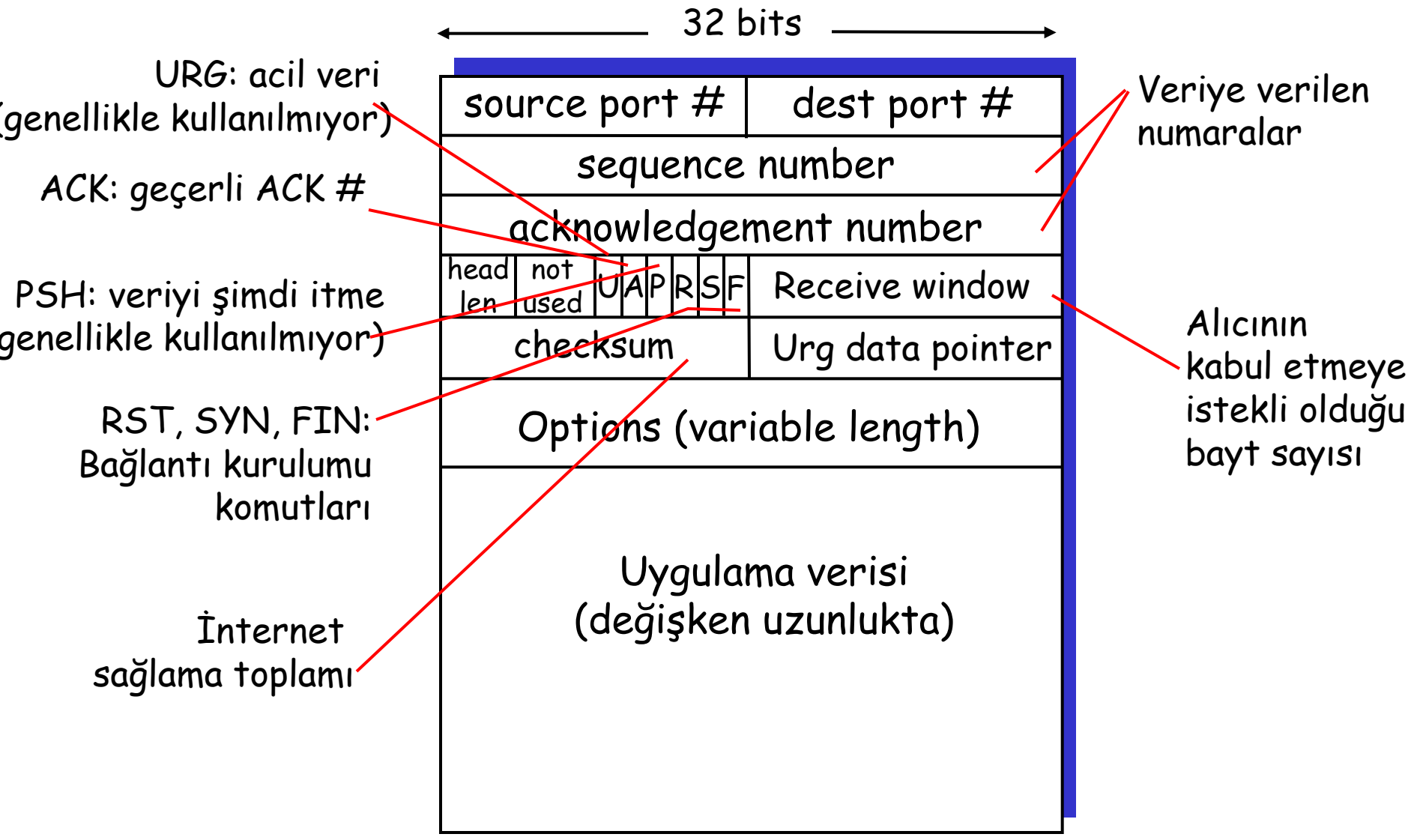
3.5 TCP: Genel bakış

- Noktadan noktaya:
  - Bir gönderici, bir alıcı
- Güvenilir, sıralı bayt akışı:
  - Mesaj sınırları yok
- Genişliğine çoklu kanal (pipelined):
  - TCP trafik yoğunluğu ve akış kontrolü pencere boyutunu belirler
- *Gönderici & alıcının arabellekleri*



- Tam çift yönlü (full duplex) veri:
  - Aynı bağlantıda çift yönlü veri akışı
  - MSS: Maksimum segment boyutu
- Bağlantı tabanlı:
  - El sıkışma- (handshaking) bağlantının önceden kurulması
- Veri akışının kontrolü:
  - Gönderen, alıcıyı alabildiği kadar veri gönderir

3.5.1 TCP segment yapısı



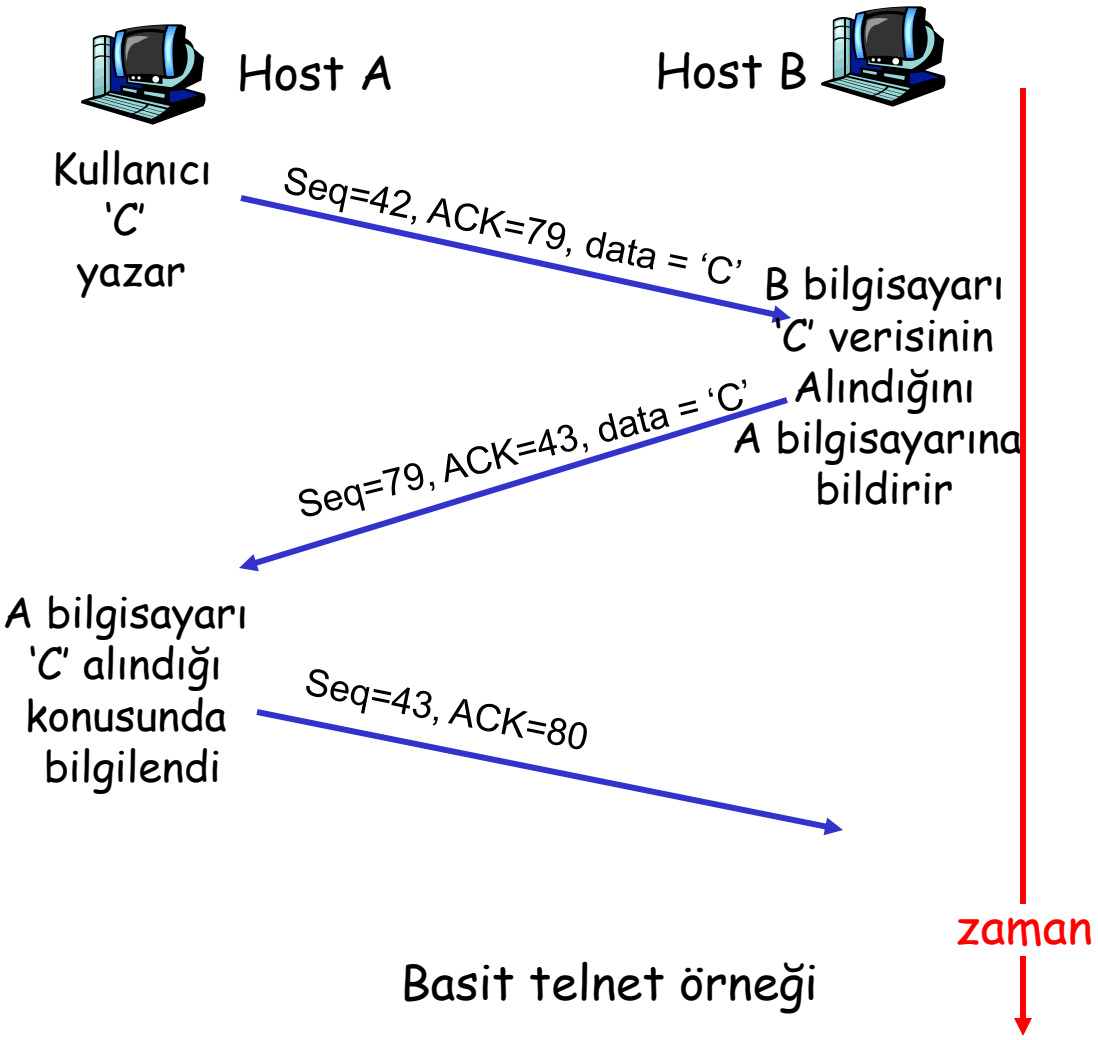
TCP segment numarası ve ACKs

segment (seq) numaraları:

- Segment verisindeki ilk baytın akış numarası

ACKs:

- Karşı taraftan beklenen sonraki baytın segment numarası



Basit telnet örneği

TCP RTT(Round-trip) süresi ve zaman aşımı

S: TCP zaman aşımı süresini nasıl ayarlar?

- RTT(gidiş-dönüş süresi)nden daha fazla bir zaman kabul edilir
  - RTT süresi değişir
- Çok kısa olursa: Erken zaman aşımı olur
  - Gereksiz yeniden iletimler olur
- Çok uzun olursa: segment kayıplarına yavaş tepki verilir

S: RTT nasıl tahmin edilir?

- `örnekRTT`: segmentin gönderilmesinden ACK alındısı gelene kadar geçen zaman
  - Yeniden gönderimler yok sayılır
- `örnekRTT` değişebilir, en iyisi
  - Son ölçümlerin ortalamasını almak

3.5.2 TCP güvenilir veri aktarımı

- TCP, IP güvensiz hizmetin üzerine rdt protokolünü oluşturur
- Segment'ler çoklu kanaldan iletilir
- Toplu ACKs
- TCP, tek yeniden iletim zamanlayıcı kullanır
- Yeniden iletim tarafından tetiklenir:
  - Zaman aşımı olayları
  - Yinelenen ACKs
- Başlangıçta basit bir TCP göndericisi düşün:
  - Yinelenen ACK'leri yok say
  - Akış denetimi ve tıkanıklık kontrolünü yok say

TCP gönderen olayları:

Uygulamadan alınan veri ile:

- Seq # (segment numarası) olan segment oluşturulur
- seq #, segment içindeki ilk veridir
- Zamanlayıcı çalışmıyorsa başlatılır
- Zaman aşımı aralığı: `TimeOutInterval`

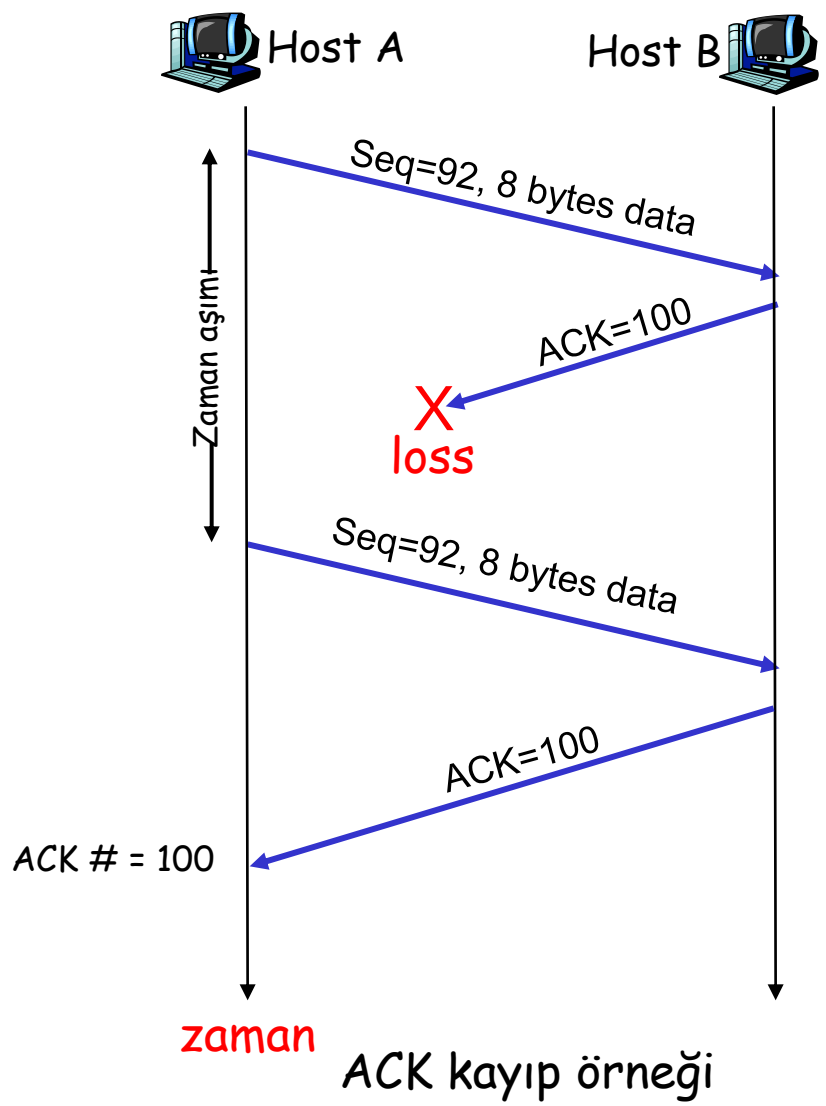
Zaman aşımı (timeout):

- Zaman aşımına uğrayan segment tekrar gönderilir
- Zamanlayıcı tekrar başlatılır

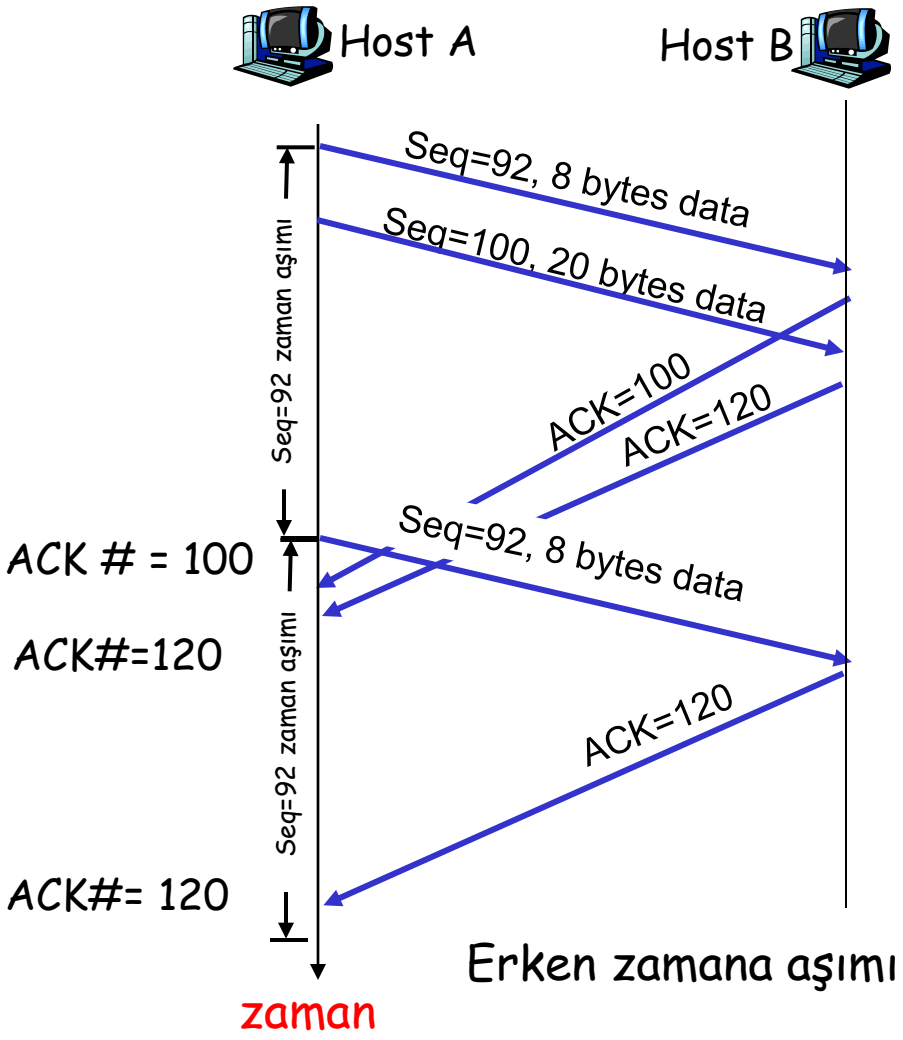
Alınan ACK :

- Eğer alındığını bildiren mesaj daha önce alınmadığı bildiren bir segment'in
  - Alınan segment bilgisini güncelleştir
  - Gidecek segment'ler varsa zamanlayıcıyı başlat

TCP yeniden iletim örneği

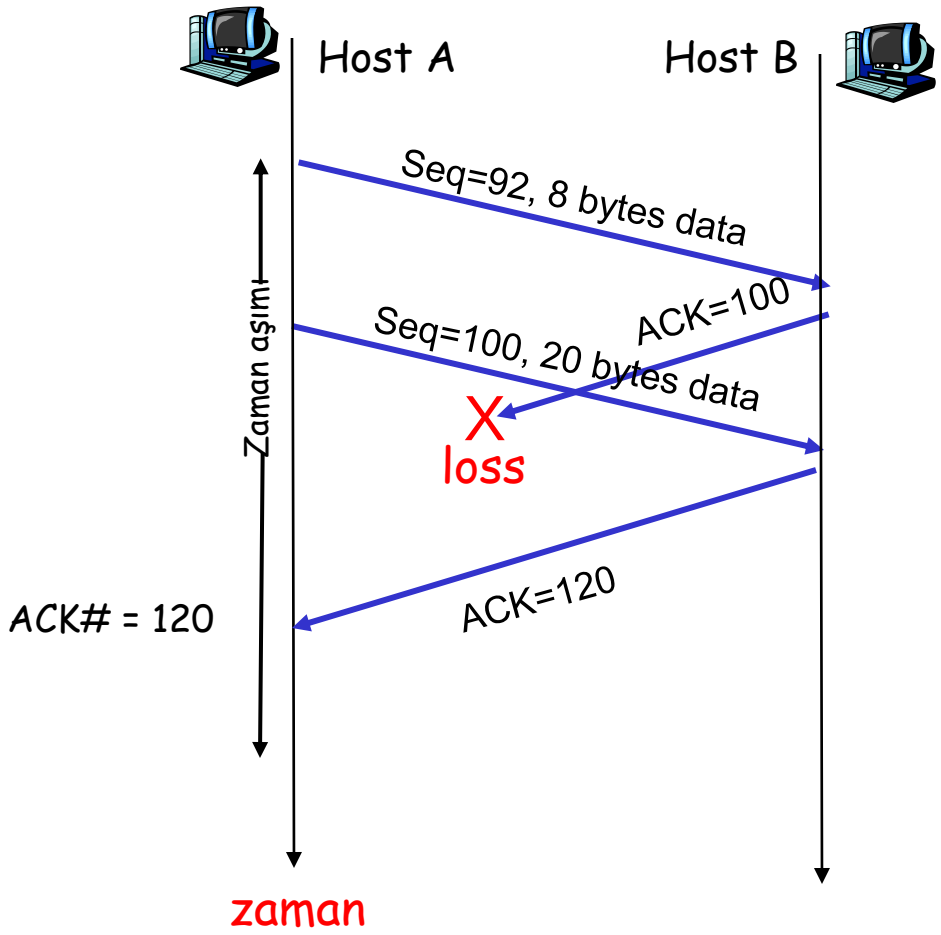


ACK kayıp örneği



Erken zamana aşımı

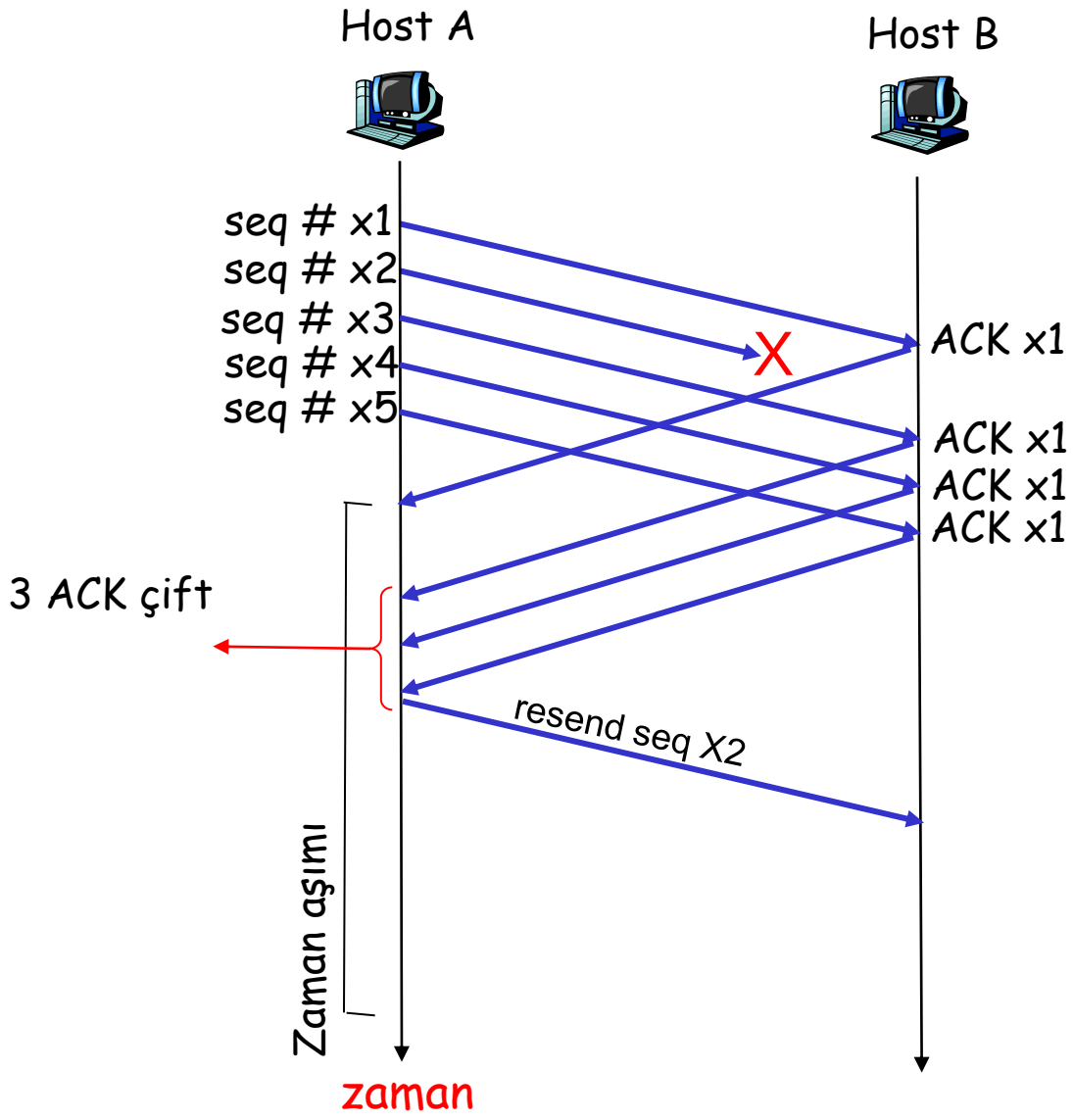
TCP yeniden iletim örneği(devamı)



Toplu ACK örneği

Hızlı yeniden iletim

- Zaman aşımı süresi genellikle uzundur:
  - Kayıp paketler yeniden gönderilmeden önce uzun süre beklenir
- Kayıp segment'ler çift ACK ile algılanır
  - Gönderen genelde birçok segmenti arka arkaya gönderir
  - Eğer segment'ler kaybolursa birçok ACK çiftleri alınır
- Eğer gönderen aynı veri için 3 ACK alırsa verinin kaybolduğu varsayılır:
  - Hızlı yeniden gönderim: zaman aşımı süresi bitmeden veri segmentleri yeniden gönderilir


Host A
Host B
seq # x1
seq # x2
seq # x3
seq # x4
seq # x5
X
ACK x1
ACK x1
ACK x1
ACK x1
3 ACK çift
resend seq X2
Zaman aşımı
zaman

3.5.3 TCP Akış denetimi

Akış denetimi

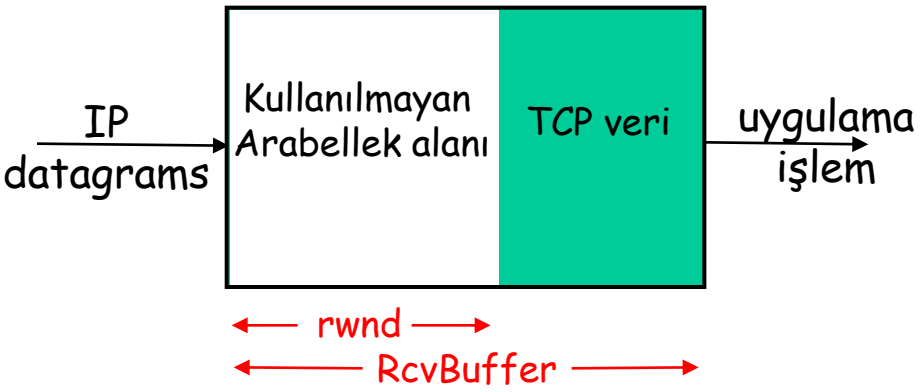
Gönderen, çok veriyi çok ve hızlı göndererek alıcının arabelleğini taşırmamalıdır

- TCP bağlantısının alan tarafının bir alma arabelleği vardır:

- *Hız eşitleme hizmeti:* Gönderme hızını, alıcı uygulamanın hızına eşleştirir

- Uygulama işlemi arabellekten okumada yavaş olabilir

TCP Akış denetimi nasıl çalışır?



(TCP alıcısı arabellekte yer yoksa segment'leri atar)

- Kullanılmayan arabellek alanı:

= rwnd

= RcvBuffer-[LastByteRcvd - LastByteRead]

- Alıcı: segment başlığında yer alan `rwnd` değeri ile kullanılmayan arabellek alanını bildirir
- Gönderen: bayt sayısını sınırlandırır
  - Alıcı arabelleğinin taşmamasını garanti eder

3.5.4 TCP bağlantı yönetimi

Arama (Recall): TCP gönderici ve alıcı veri segment'lerinin alış verişini yapmadan önce bağlantı kurarlar

- TCP değişkenleri başlatılır:
  - Segment numarası. #s
  - Arabellekler, akış denetimi bilgileri (örneğin, **RcvWindow**)
- *client:* bağlantı başlatıcı

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

- *server:* istemci tarafından bağlanılan

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

Three way handshake (3 yollu el sıkışma):

Adım 1: Client, server'a TCP SYN segment'ini gönderir
- İlk segment numarasını belirtir
- Veri yok

Adım 2: Server, SYN'yi alır ve SYNACK segment ile cevaplar
- server arabellekte yer ayırır
- Server ilk segment numarasını belirtir

Adım 3: Client, SYNACK alır ve ACK(veri içerebilir) segment ile cevaplar

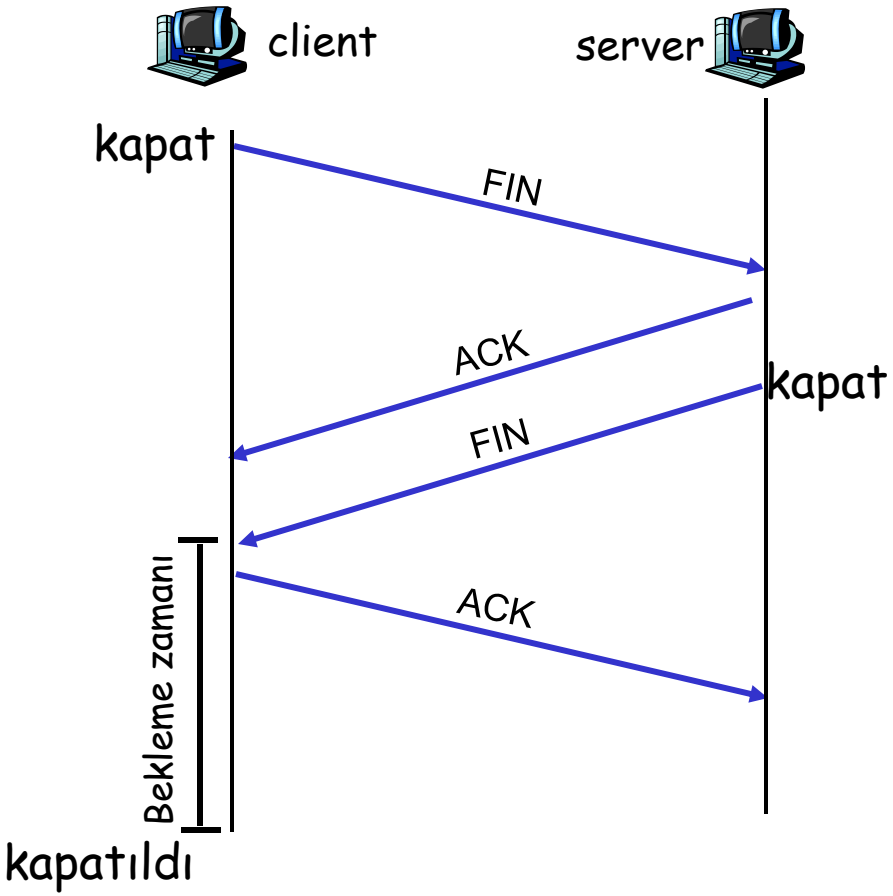
TCP bağlantı yönetimi(devamı)

Bağlantının kaptılması

client soketi kapatır:

Adım 1: client, TCP FIN denetim segment'ini server'a gönderir

Adım 2: server, FIN'i alır, ACK ile cevaplar. Bağlantıyı kapatır, FIN gönderir.

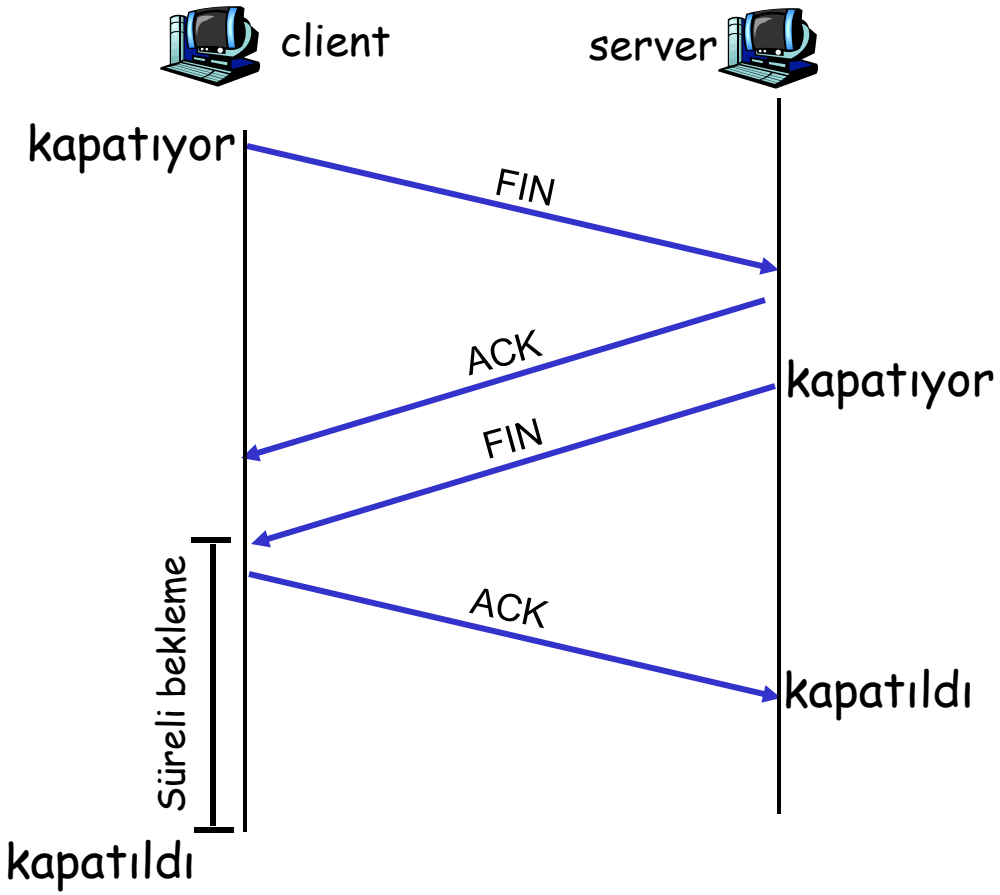


TCP bağlantı yönetimi(devamı)

Adım 3: client, FIN'i alır, ACK ile cevaplar.

- Beklemeye başlar, aldığı FIN'e ACK ile cevap verir

Adım 4: server, ACK'i alır. Bağlantı kapatılır.



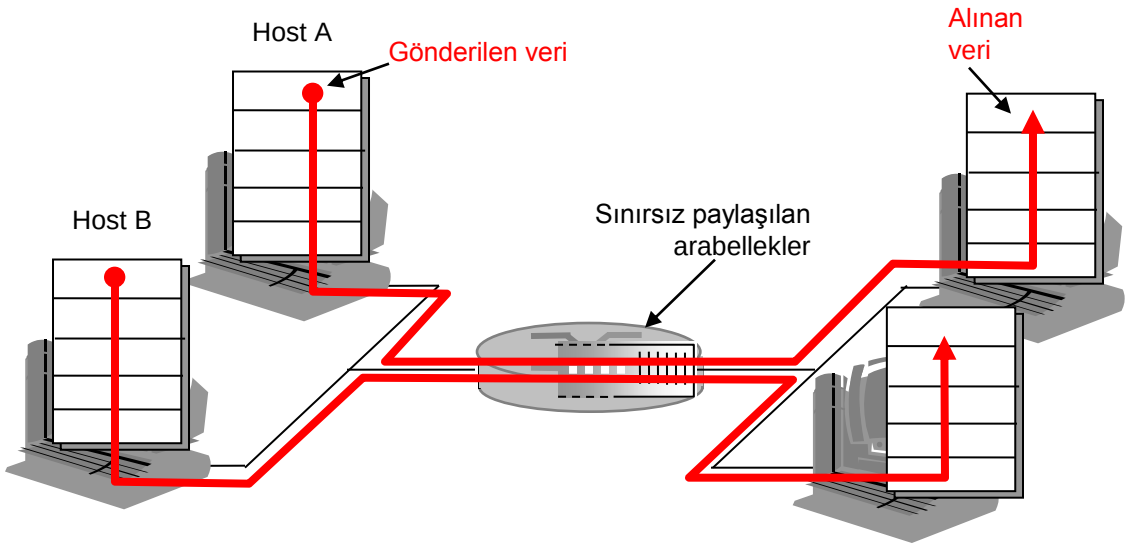
3.6 Tıkanıklık denetiminin ilkeleri

Ağ yoğunluğu (Congestion):

- Çok fazla kaynak, çok fazla veriyi (ağın iletebileceğinden fazla) ağa çok hızlı gönderiyor
- Akış denetimden farklı bir durum!
- Belirtileri:
  - Paketlerin kaybolması (yönlendiricilerde arabelleğin taşması)
  - Uzun gecikmeler (yönlendirici arabelleğinde kuyruk)
- En önemli ağ problemlerinden!

Tıkanıklığın nedenleri ve maliyetleri örnek 1

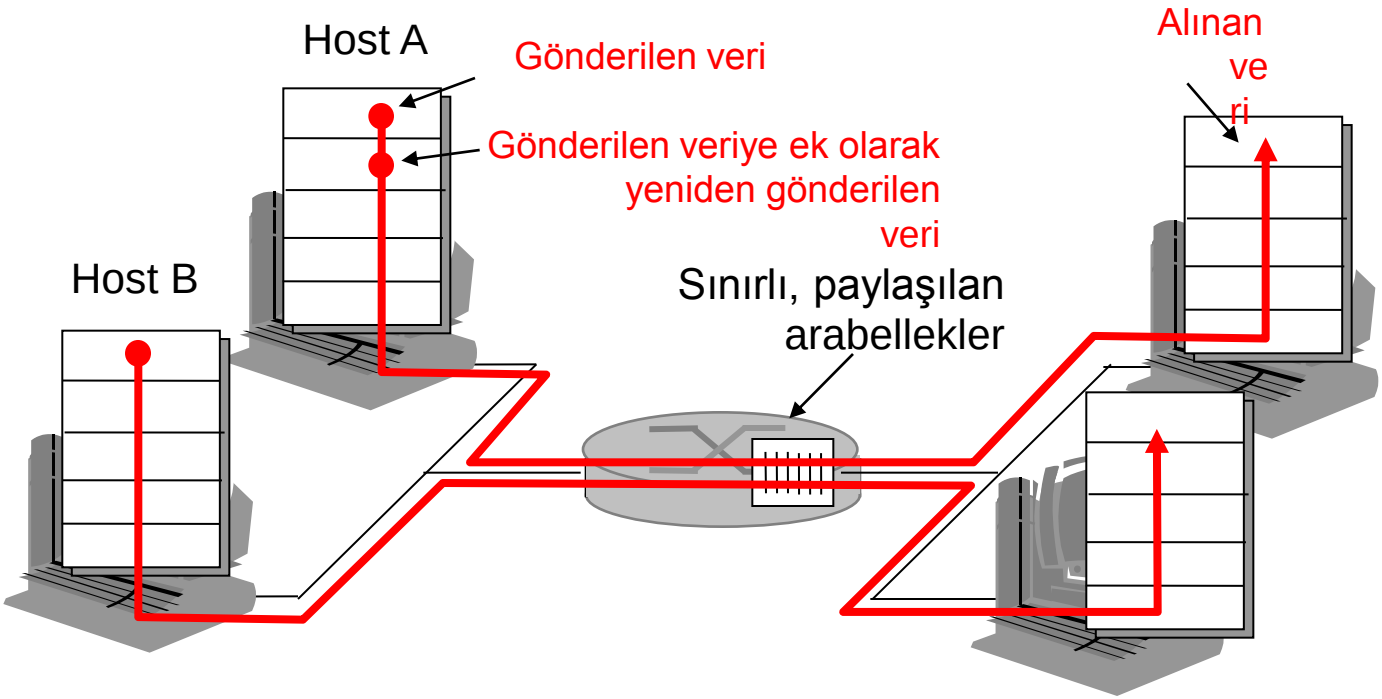
- İki gönderen, iki alıcı
- Bir yönlendirici, sonsuz arabellekler
- Yeniden iletime gerek yok



- Ulaşılabilir en yüksek performans

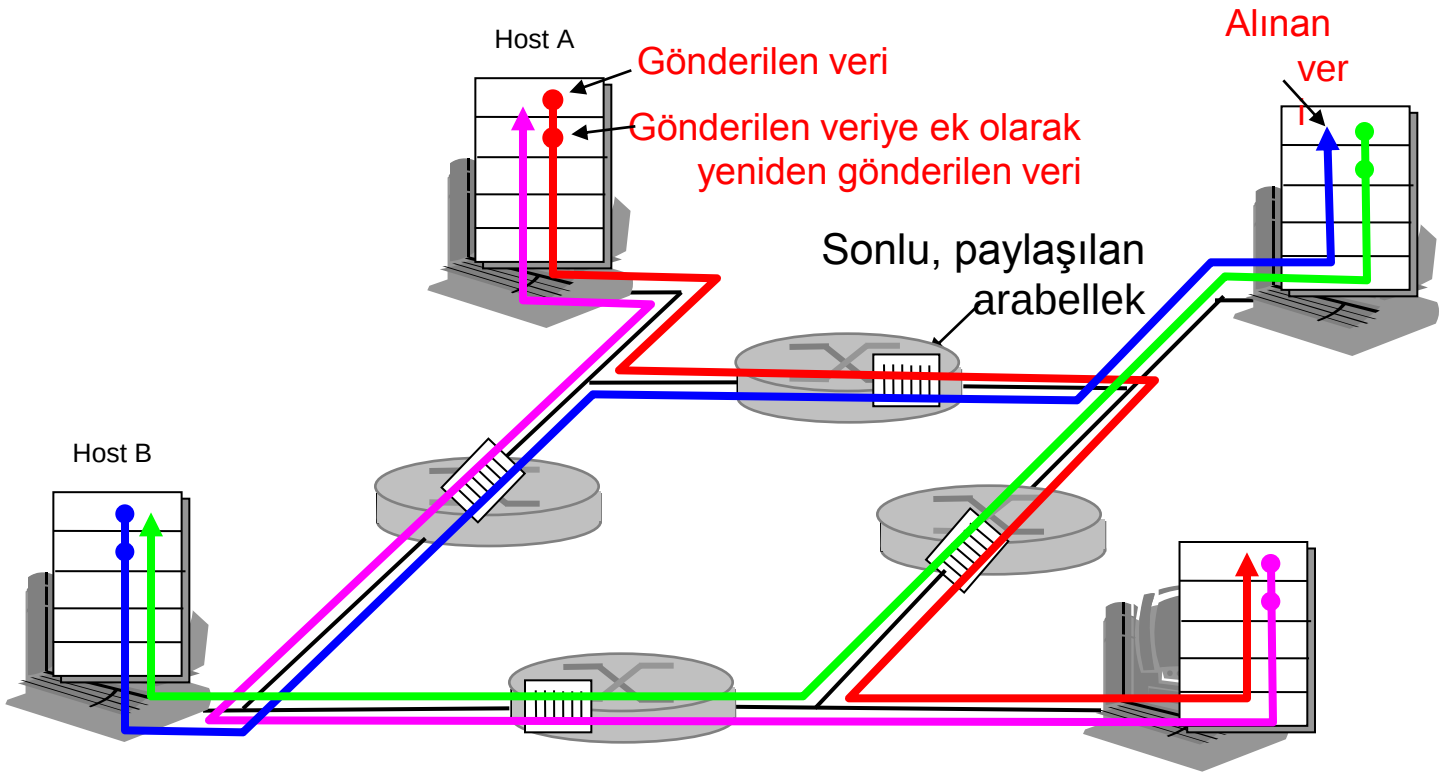
Tıkanıklığın nedenleri ve maliyetleri örnek 2

- Bir yönlendirici, *sınırlı* arabellekler
- Gönderen kaybolan paketleri tekrar gönderir



Tıkanıklığın nedenleri ve maliyetleri örnek 3

- 4 gönderen
- Birden çok yönlendirici
- Zaman aşımı/yeniden gönderme



Tıkanıklık denetimi yaklaşımları

Tıkanıklık denetiminde yaygın kullanılan 2 yöntem:

Uç sistemler arası tıkanıklık denetimi:

- Ağ destekli bir geri bildirim yoktur
- Tıkanıklık, uç sistemlerde gözlenen kayıp ve gecikmelerden anlaşılır

Ağ destekli tıkanıklık denetimi:

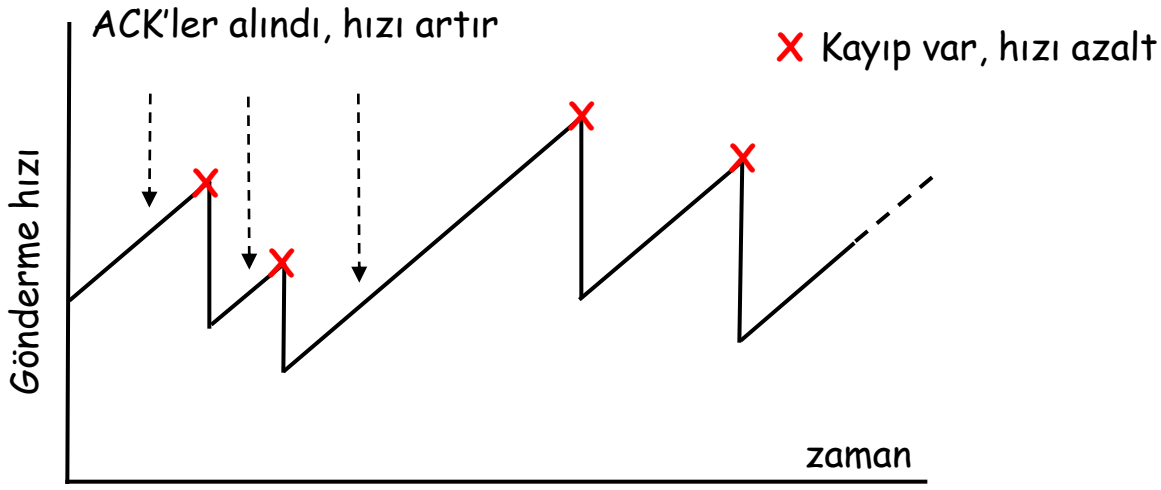
- Yönlendirici uç sistemlere geri bildirim sağlar
  - Tek bit tıkanıklığı gösterir (SNA, DECbit, TCP/IP ECN, ATM)
  - Gönderici, göndermesi gereken oranda gönderir

3.7 TCP tıkanıklık denetimi:

- *Hedef:* TCP gönderici, ağı tıkamadan olabildiğince hızlı veri göndermelidir
  - S: Tıkanıklık seviyesine yakın olan hız nasıl bulunur?
- Merkezi olmayan: Her TCP gönderici kendi hızını ayarlar, geribildirimlere dayanarak:
  - *ACK:* Alınması iyi bir durum, ağda tıkanıklık yok demektir, gönderme hızı artırılabilir
  - *Segment kayboluyorsa:* kayıpların ağ tıkanıklığından kaynaklandığı kabul edilir, gönderme hızı azaltılır

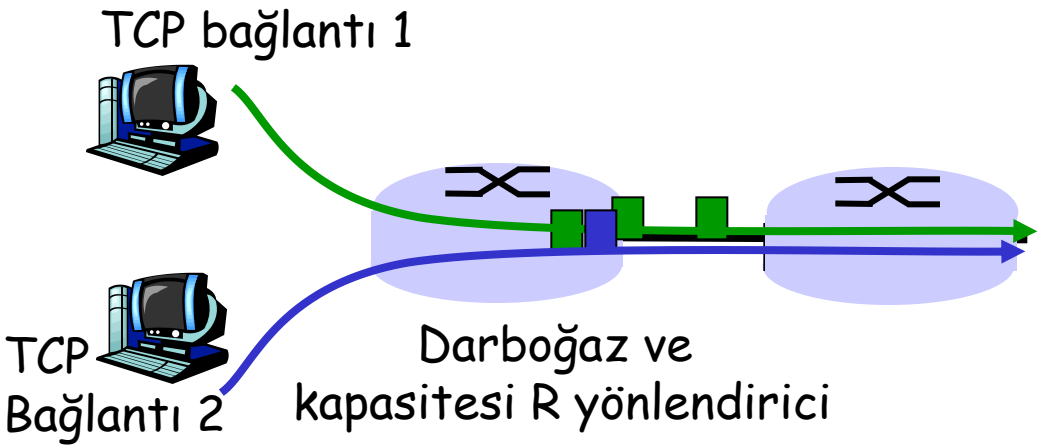
TCP tıkanıklık denetimi

- "Bant genişliği için kontrol": ACK alınırsa iletim hızını artır, paketler kaybolursa iletim hızını azalt
  - Hızın artması veya azaltılması ağ bağlantılarına bağlı olarak bant genişliğinin değişmesine neden olur



TCP adaleti

Adaletin amacı: Eğer TCP bağlantıları R bant genişliğindeki aynı darboğazı, sıkışıklığı paylaşıyorlarsa ortalama hızları R/bağlantı sayısı olur



TCP adaleti(devamı)

UDP

- Multimedya uygulamaları çoğu kez TCP kullanmaz
  - Tıkanıklık denetimi ile hızın azaltılmasını istenmediği için
- UDP kullanılır:
  - Ses ve videolar sabit hızda iletilir, paket kayıplarına tolerans gösterilir

Paralel TCP bağlantıları

- İki bilgisayar arasında paralel bağlantılar açılabilir
- Web tarayıcılar bunu yapar
- Örnek: Hızı R olan bir bağlantı 9 bağlantı destekliyorsa;
  - Yeni uygulama 1 TCP bağlantısı ister ve yeni hız R/10 olur