Bilgisayar Ağları
Computer Networks

Hazırlayan: M. Ali Akcayol
Gazi Üniversitesi
Bilgisayar Mühendisliği Bölümü

Bu dersin sunumları, "James Kurose, Keith Ross, Computer Networking: A Top-Down Approach 6/e, Pearson, 2013." kitabı kullanılarak hazırlanmıştır.

İçerik

- Tıkanıklık denetimi
- Tıkanıklığın nedenleri ve maliyeti
- Tıkanıklık denetimi yaklaşımları
- Ağ destekli tıkanıklık denetimi
- Uçtan uca tıkanıklık denetimi
- TCP tıkanıklık denetimi

Tıkanıklık denetimi

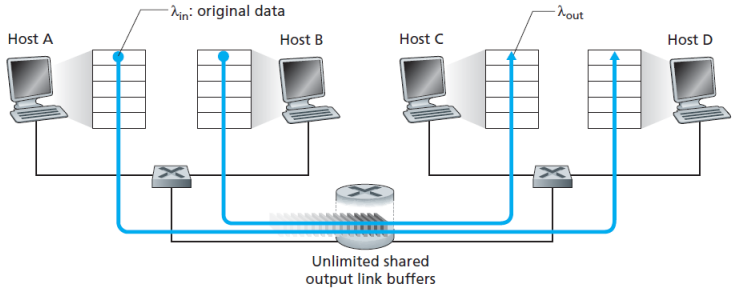
- TCP protokolü paket kayıplarına karşı güvenilirliği sağlamak için gerekli mekanizmalara sahiptir.
- TCP protokolü timer kullanarak **kayıp paketlerin tekrar gönderimini sağlar.**
- **Paket kayıpları** genellikle **router buffer'ının dolu olmasından kaynaklanır.**
- **Paketlerin tekrar gönderim sıklığının artması ağda tıkanıklık olduğunu gösterir.**
- Ağdaki tıkanıklığın nedenini ortadan kaldırmak için göndericide yeni mekanizmalara ihtiyaç vardır.
- **Tıkanıklık denetimini gönderici, akış denetimini alıcı yönetir.**

İçerik

Tıkanıklığın nedenleri ve maliyeti

Senaryo 1: İki gönderici, bir router ve sınırsız buffer

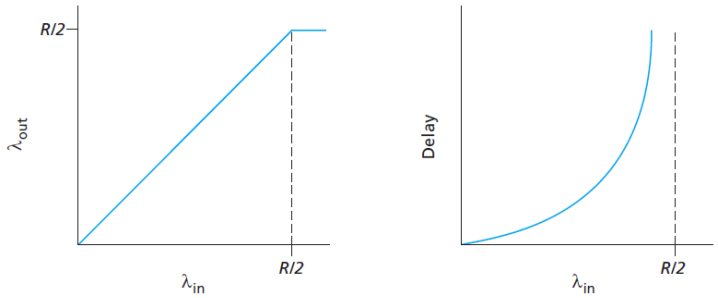
- İki host bir **router'ı (buffer=∞) paylaşarak** kullanmaktadır.
- **Retransmit, akış** ve **tıkanıklık denetimi** yapılmıyor.
- Host A ve Host B ağa λ_{in} **byte/s veri göndermektedir.**
- Alıcıya λ_{out} **byte/s** veri ulaşıyor.
- **Paylaşılan link R bps** bant genişliğine sahiptir.
- Host A ve Host B'nin **R/2 bps bant genişliği vardır.**



Tıkanıklığın nedenleri ve maliyeti

Senaryo 1: İki gönderici, bir router ve sınırsız buffer

- **Soldaki grafik** her bağlantı için **alıcıdaki throughput'u** gösterir.
- **Sağdaki grafik** λ_{in} değerine göre **gecikmeyi gösterir.**

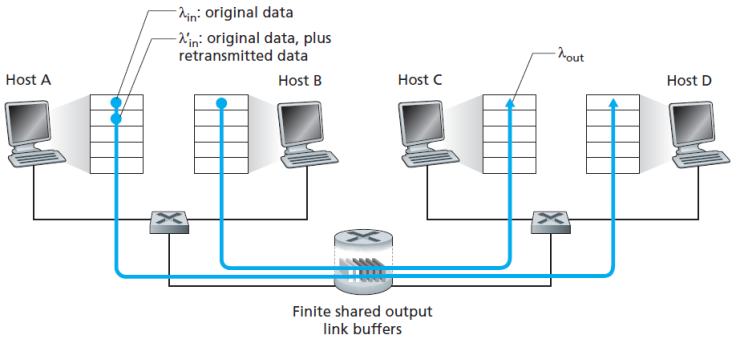


- Veri gönderim oranı **R/2'den fazla olursa buffer'a alınır.**
- Buffer'daki veri arttıkça gecikme süresi sonsuza doğru artar.
- **Tıkanıklığın maliyeti gecikme süresindeki artıştır.**

Tıkanıklığın nedenleri ve maliyeti

Senaryo 2: İki gönderici, bir router ve sınırlı buffer

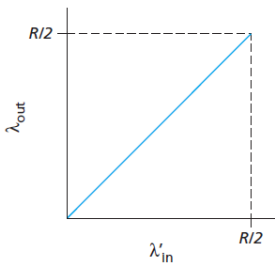
- **Router sınırlı buffer'a sahiptir.**
- Buffer dolu ise **paketler atılıyor**, gönderici retransmit yapıyor.
- Uygulama katmanı λ_{in} byte/s ulaşım katmanına gönderir.
- λ'_{in} **orijinal veri** ve **retransmit yapılan verinin toplamıdır.**



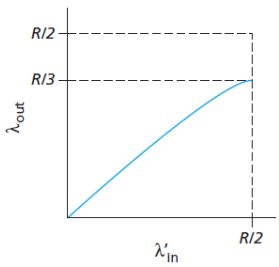
Tıkanıklığın nedenleri ve maliyeti

Senaryo 2: İki gönderici, bir router ve sınırlı buffer

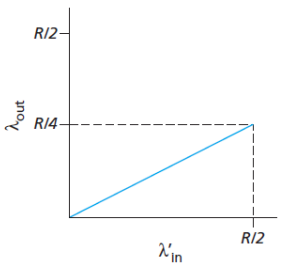
- **Kayıp** paket hiç **olmazsa** $\lambda_{in} = \lambda'_{in}$ **olur** (soldaki grafik).
- Kaybolduğu kesin olanlar tekrar gönderilebilir (ortadaki grafik).
- Retransmit oranı arttıkça λ_{out} azalır.
- **Burada tıkanıklığın maliyeti kayıpların retransmit edilmesidir.**



Kayıp yok



Kayıp var

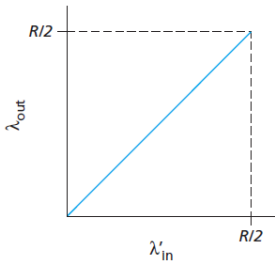


Tekrarlı gidenler var (Erken timeout)

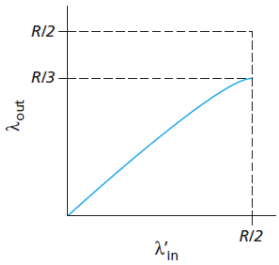
Tıkanıklığın nedenleri ve maliyeti

Senaryo 2: İki gönderici, bir router ve sınırlı buffer

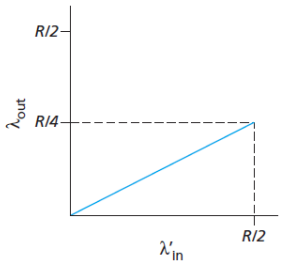
- **Erken timeout** ile **retransmit edilebilir** (sağdaki grafik).
- **Tekrar alınan paketler** alıcıda **atılır** ve λ_{out} **oranı azalır.**
- Router'da ve diğer bileşenlerde gereksiz iş yapılır.
- **Burada tıkanıklığın maliyeti** erken timeout'tan dolayı **gereksiz retransmit yapılmasıdır.**



Kayıp yok



Kayıp var

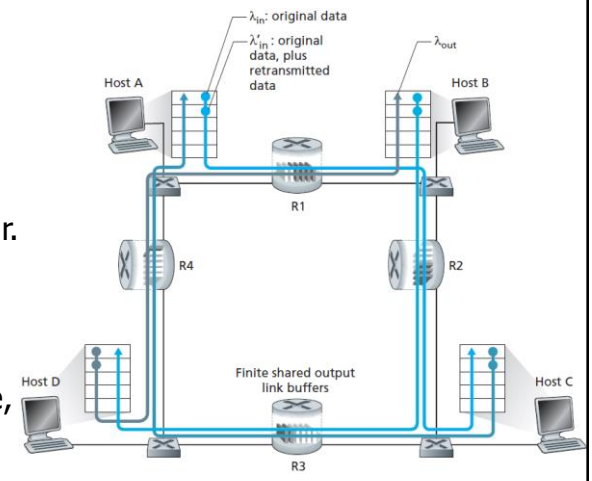


Tekrarlı gidenler var (Erken timeout)

Tıkanıklığın nedenleri ve maliyeti

Senaryo 3: Dört gönderici, router'lar (sınırlı buffer), multihop yol

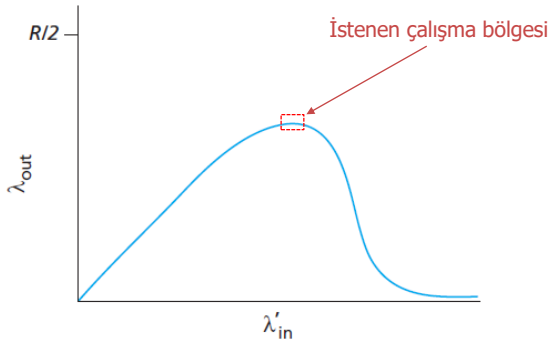
- Tüm host'lar iki hop yola sahiptir.
- Tüm hostlar λ_{in} byte/s veri göndermektedir.
- Tüm linkler R byte/s bant genişliğine sahiptir.
- Host A, Host C ile **R1** ve **R2** üzerinden haberleşmektedir.
- A-C, R1 linkini D->B ile, R2 linkini B->D ile paylaşmaktadır.



Tıkanıklığın nedenleri ve maliyeti

Senaryo 3: Dört gönderici, router'lar (sınırlı buffer), multihop yol

- Küçük λ_{in} değerleri için buffer taşması nadiren olur ($\lambda_{in} = \lambda_{out}$).
- λ_{in} arttıkça λ'_{in} artar.
- **λ_{in} ve λ'_{in} arttıkça atılan paket artar ve λ_{out} sıfıra doğru düşer.**



İçerik

- Tıkanıklık denetimi
- Tıkanıklığın nedenleri ve maliyeti
- Tıkanıklık denetimi yaklaşımları
- Ağ destekli tıkanıklık denetimi
- Uçtan uca tıkanıklık denetimi
- TCP tıkanıklık denetimi

Tıkanıklık denetimi yaklaşımları

- Tıkanıklık denetimi **ağdaki cihazlar tarafından geri bildirim** kullanılarak yapılırsa **ağ trafiğini artırır.**
- Tıkanıklık denetimi **uç birimlerde** yapılırsa **kayıp paket ve gecikmelere göre analiz yapılır.**
- Ağdaki tıkanıklık denetimi yaklaşımları temel olarak iki gruba ayrılır:
  - **Ağ destekli tıkanıklık denetimi**
  - **Uçtan uca tıkanıklık denetimi**

İçerik

- Tıkanıklık denetimi
- Tıkanıklığın nedenleri ve maliyeti
- Tıkanıklık denetimi yaklaşımları
- Ağ destekli tıkanıklık denetimi
- Uçtan uca tıkanıklık denetimi
- TCP tıkanıklık denetimi

Ağ destekli tıkanıklık denetimi

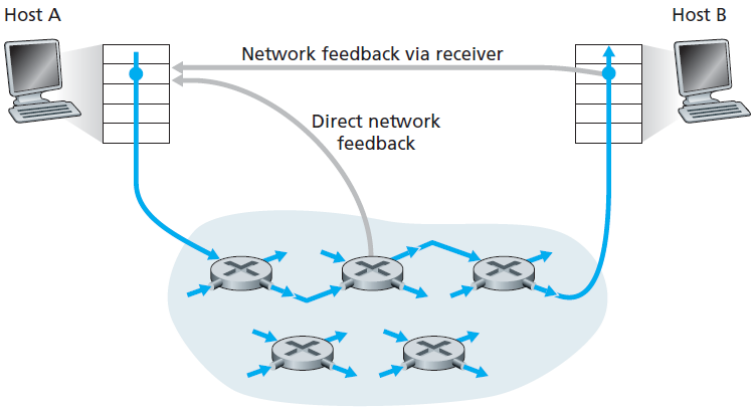
- **Ağ katmanı bileşenleri** (router) **göndericiye** doğrudan veya dolaylı bir şekilde **ağdaki tıkanıklığı bildirebilir.**
- Geri bildirim **bir bit kullanılarak yapılabilir.**
- Karmaşık yöntemlerde **router doğrudan göndericiye** kendisinin çıkış linkinin **desteklediği gönderim oranını da bildirebilir.**
- Günümüzde IBM SNA, DEC DECnet ve ATM ağlarda kullanılmaktadır.

Ağ destekli tıkanıklık denetimi

- Tıkanıklık bilgisi ağdaki **cihaz tarafından iki yolla** göndericiye iletilebilir.
  - **Doğrudan bildirim**
  - **Alıcı üzerinden bildirim**
- **Doğrudan bildirimde,** router doğrudan göndericiye bir paket ile bildirim yapar **(choke packet).**
- **Alıcı üzerinden bildirimde, router** üzerinden **geçen paketin tıkanıklık bitini işaretleyerek gönderir.**
- **Alıcı geri bildirimle** göndericiye tıkanıklık bildiriminde bulunur (akış denetimine benzer).

Ağ destekli tıkanıklık denetimi

- **Doğrudan router** tarafından bildirimde **geri bildirim süresi daha kısadır.**
- **Alıcı üzerinden bildirimde** en az **bir RTT süresi gerekir.**



İçerik

- Tıkanıklık denetimi
- Tıkanıklığın nedenleri ve maliyeti
- Tıkanıklık denetimi yaklaşımları
- Ağ destekli tıkanıklık denetimi
- Uçtan uca tıkanıklık denetimi
- TCP tıkanıklık denetimi

Uçtan uca tıkanıklık denetimi

- **Ağ katmanının ulaşım katmanına** doğrudan **destek sağlamadığı** durumlarda kullanılır.
- Ağdaki tıkanıklık **uç birimlerde paket kaybı** veya **gecikme** kullanılarak algılanır.
- **TCP uçtan uca tıkanıklık denetimi yapar.**
- **IP** uç sistemlere ağdaki **tıkanıklıkla ilgili geri bildirim yapmaz.**
- TCP **segment kaybı ağdaki tıkanıklığa işaret sayılır** ve gönderim hızı azaltılır.
- **RTT süresi arttıkça tıkanıklığa işaret sayılır** ve gönderim hızı azaltılır.

İçerik

- Tıkanıklık denetimi
- Tıkanıklığın nedenleri ve maliyeti
- Tıkanıklık denetimi yaklaşımları
- Ağ destekli tıkanıklık denetimi
- Uçtan uca tıkanıklık denetimi
- TCP tıkanıklık denetimi

TCP tıkanıklık denetimi

- Tıkanıklık denetimi TCP protokolünün temel özelliklerindendir.
- **IP katmanı** uç sistemlere **tıkanıklıkla ilgili geri bildirim göndermez.**
- Bu nedenle **TCP uçtan uca tıkanıklık denetimi yapar.**
- Ağdaki **tıkanıklık düzeyine göre her gönderici gönderme hızını sınırlar.**
- TCP tıkanıklık denetimi için tıkanıklık penceresi **(congestion window – `cwnd`)** kullanır.
- TCP **`cwnd` ile ağa veri gönderme hızını sınırlar.**
- ACK alınmadan gönderilecek **maksimum veri miktarı `min(cwnd, rwnd)` olmalıdır** (`rwnd` - receive window).

$$\texttt{LastByteSent} - \texttt{LastByteAcked} \leq \min\{\texttt{cwnd}, \texttt{rwnd}\}$$

TCP tıkanıklık denetimi

- TCP gönderici **timeout olduğunda** veya **3 kez art arda ACK** (negatif) **aldığında** tıkanıklık olduğuna karar verir.
- TCP aldığı ACK'lara göre **`cwnd`** boyutunu düzenler **(self-clocking)**:
  - Hiç **paket kaybı olmaksızın ACK almaya devam ederse,** RTT süresine göre **`cwnd`** boyutunu artırır.
  - **ACK alma süresi uzarsa,** **`cwnd`** boyutunu düşürür.
  - **ACK alma süresi aniden çok kısalırsa,** **`cwnd`** boyutunu hızla artırır; **aniden çok uzarsa** **`cwnd`** boyutunu hızla azaltır.
- TCP, **`cwnd`** boyutunu değiştirerek **veri gönderme hızını ayarlar.**
- TCP, ağda tıkanıklık olmayacak şekilde **maksimum veriyi göndermeye çalışır.**

TCP tıkanıklık denetimi

- Bir **segment kaybolduğunda** tıkanıklığı gösterir ve **TCP gönderme hızını düşürür.**
- TCP, **ACK aldığı sürece gönderme hızını sürekli artırır.**
- TCP tıkanıklık kontrol algoritması **üç bileşene sahiptir:**
  - **Yavaş başlatma** (slow start)
  - **Tıkanıklıktan kaçınma** (congestion avoidance)
  - **Hızlı toparlanma** (fast recovery)

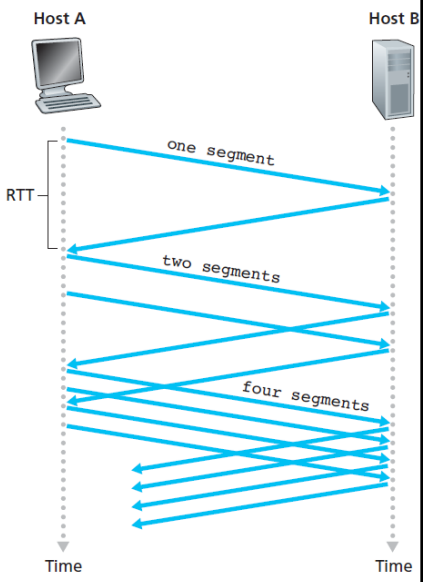
TCP tıkanıklık denetimi

Yavaş başlatma

- TCP **başladığında `cwnd`** değeri **1 MSS** (maximum segment size) **alınır.**

  MSS= MTU-TCPHeader-IPHeader
- MTU datalink layer payload boyutudur.
- **Hiç ACK almadan** veri gönderim oranı MSS/RTT olarak alınır.

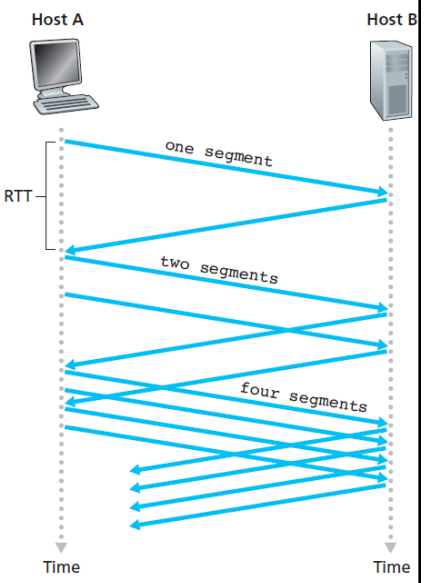
  MSS = 500 byte,
  RTT = 200 ms,
  `cwnd` = 20 kbps olur.



TCP tıkanıklık denetimi

Yavaş başlatma

- TCP, **her ACK aldığında `cwnd`** değerini 1 MSS artırır.
- Slow startta **ilk paket kaybında `cwnd` = 1*MSS** yaparak **slow start durumunu korur.**
- **`ssthresh`** (slow start threshold) **`cwnd/2`** yapılır.
- Eğer **`cwnd = ssthresh`** olursa **congestion avoidance moduna** geçer.
- Slow start modunde iken **3 kez negatif ACK alırsa,** fast retransmit yapar.
- Ardından **fast recovery moduna** geçerek **`ssthresh=cwnd/2`** **`cwnd=ssthresh+3*MSS`** yapar.



TCP tıkanıklık denetimi

Tıkanıklıktan kaçınma

- Congestion avoidance moduna geçildiğinde, **`cwnd`** değeri en son kayıp anındaki değerinin **yarısına eşitlenir.**
- TCP gönderici her ACK aldığında, **`cwnd`** değerini **`MSS/cwnd` oranında artırır (`cwnd=cwnd+MSS*(MSS/cwnd)`).**
- **MSS** = 1460 byte, **`cwnd`** = 14600 byte ise, 10 segment 1 RTT süresinde gönderilebilir.
- **Burada, her ACK** alındığında **`cwnd`** değeri **`MSS/cwnd`** **`(1/10)`** oranında artırılır.
- **Timeout olursa `cwnd=1`** ve **`ssthresh=cwnd/2`** yapılır, ardından **slow start** durumuna geçilir.
- TCP, art arda **üç negatif ACK aldığında** (fast retransmit yapar), **`cwnd=cwnd/2`** yapılır ve **fast recovery durumuna** geçer.

TCP tıkanıklık denetimi

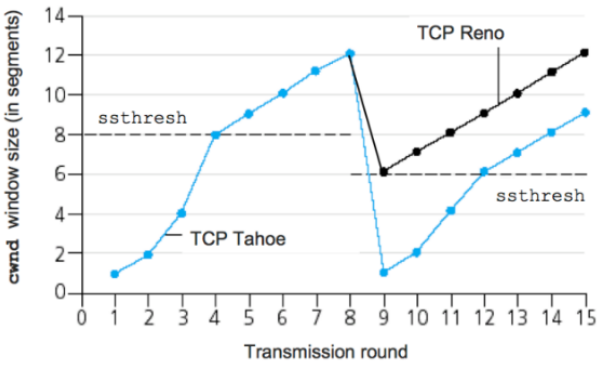
Hızlı toparlanma

- Fast recovery durumunda iken;
  - **Kayıp paket için negatif ACK** geldiğinde, **`cwnd`** değeri **1 MSS artırılır** ve **fast recovery** durumu devam eder.
  - **Yeni** bir **pozitif ACK** geldiğinde, **`cwnd=ssthresh`** yapılır ve **congestion avoidance** durumuna geçilir.
  - **Timeout olursa `ssthresh=cwnd/2`** ve **`cwnd=1`** yapılır, ardından **slow start** durumuna geçilir.
- TCP, **`cwnd`** recovery için **ilk versiyonlarında** ilk kayıp algılamada **`cwnd=1`** yapılmaktaydı **(Tahoe).**
- TCP **yeni versiyonda** ilk kayıp algılamada **`cwnd=cwnd/2`** yapılmaktadır **(Reno).**

TCP tıkanıklık denetimi

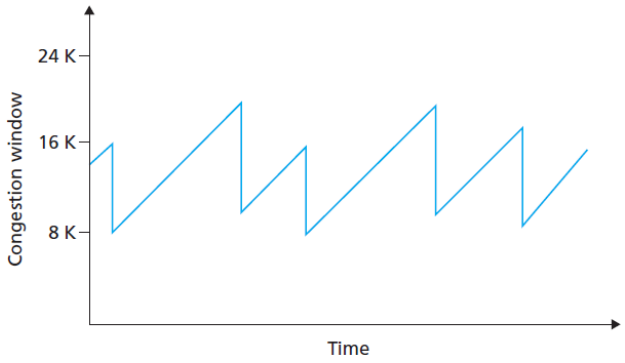
Hızlı toparlanma

- TCP Tahoe ve Reno'da **`ssthresh`** değerine kadar **`cwnd`** üstsel artar, **`ssthresh`** değerinden sonra doğrusal artar.
- İlk kayıp pakette, Tahoe **`cwnd`** değerini **`1`**, Reno ise **`cwnd/2`** yapar.



TCP tıkanıklık denetimi

- TCP, **`cwnd`** boyutunu çalışma süresince **toplama yaparak artırır,** her kayıp algılamasında **yarı yarıya düşürür (additive-increase, multiplicative decrease-AIMD).**



TCP tıkanıklık denetimi