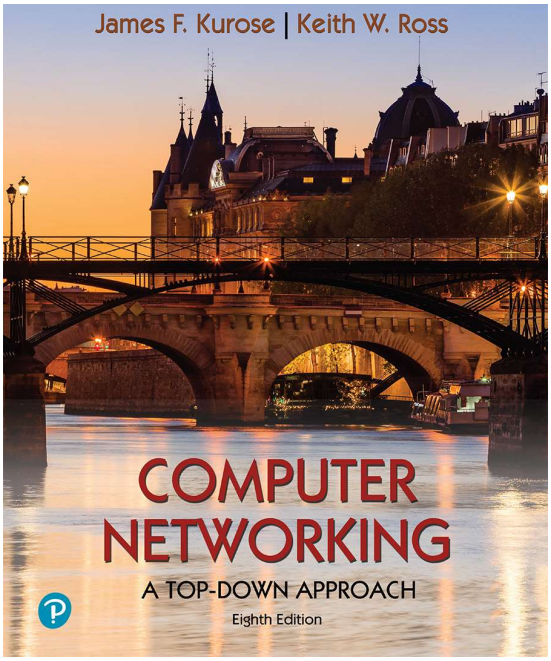


Bölüm 2
Uygulama Katmanı

Dr. Öğretim Üyesi A. Erhan AKKAYA

Bilgisayar ağları, ağ uygulamaları için var olur; bu uygulamalar olmasaydı ağ altyapısına ihtiyaç duyulmazdı. İnternetin yaygınlaşmasıyla birlikte e-posta, dosya transferi, haber grupları gibi klasik metin tabanlı uygulamalar ile **World Wide Web (WWW)** gibi yeni uygulamalar gelişti. 2000'lerden itibaren **Skype**, **YouTube**, **Netflix** gibi video uygulamaları ve **Facebook**, **Instagram**, **Twitter** gibi sosyal ağlar ön plana çıktı. Akıllı telefonların yaygınlaşmasıyla **WhatsApp** gibi konum tabanlı ve mesajlaşma uygulamaları popülerleşti.

Uygulama katmanı: Genel bakış/yol haritası

- Ağ uygulamalarının prensipleri
- Web ve HTTP
- E-mail, SMTP, IMAP
- Domain Name System DNS
- P2P uygulamalar
- video akışı ve içerik dağıtım ağları
- UDP ve TCP ile soket programlama

Uygulama katmanı: genel bakış

Hedeflerimiz :

- Uygulama katmanı protokollerinin kavramsal ve uygulama yönleri
  - taşıma katmanı servis modelleri
  - client-server paradigması
  - peer-to-peer paradigması
- popüler uygulama katmanı protokollerini inceleyerek protokoller hakkında bilgi edineceğiz
  - HTTP
  - SMTP, IMAP
  - DNS
- ağ uygulamalarının programlanması
  - socket API

Bu bölümde, ağ uygulamalarının kavramsal ve uygulama yönlerini inceleyeceğiz. Uygulama katmanı kavramlarını, uygulamaların gerektirdiği ağ servislerini, istemci-sunucu modellerini ve taşıma katmanı arayüzlerini tanımlayacağız. Web, e-posta, DNS, eşler arası dosya paylaşımı ve video akışı gibi popüler ağ uygulamalarını detaylı olarak ele alacağız. Ayrıca, **TCP** ve **UDP** üzerinden ağ uygulamaları geliştirip, **Python** ile basit istemci-sunucu uygulamalarını adım adım anlatacağız. Bölüm sonunda soket programlama ile ilgili ödevler de verilecektir.

Bazı ağ uygulamaları

- Sosyal ağ
- Web
- Metin mesajlaşma
- E-posta
- çok kullanıcılı ağ oyunları
- depolanmış video akışı (YouTube, Hulu, Netflix)
- P2P dosya paylaşımı
- voice over IP (e.g., Skype)
- gerçek zamanlı video konferans
- Internet arama
- uzak bağlantı
- …

Elinizde yeni bir ağ uygulaması fikri var ve bu fikir insanlığa katkıda bulunabilir ya da sadece eğlence amaçlı tasarlanmış olabilir. Şimdi, bu fikri gerçek dünyada nasıl hayata geçirebileceğinizi ele alalım.

Ağ uygulamaları geliştirirken, farklı uç sistemlerde çalışan ve ağ üzerinden haberleşen programlar yazmak temel bir adımdır. Örneğin, bir **Web uygulaması**, kullanıcının cihazındaki **tarayıcı** ve web sunucusunda çalışan **sunucu programı** ile etkileşime girer. Benzer şekilde, **Netflix** gibi bir **Talebe Bağlı Video uygulaması**, kullanıcının cihazında çalışan bir uygulama ve Netflix sunucusunda çalışan bir sunucu programından oluşur.

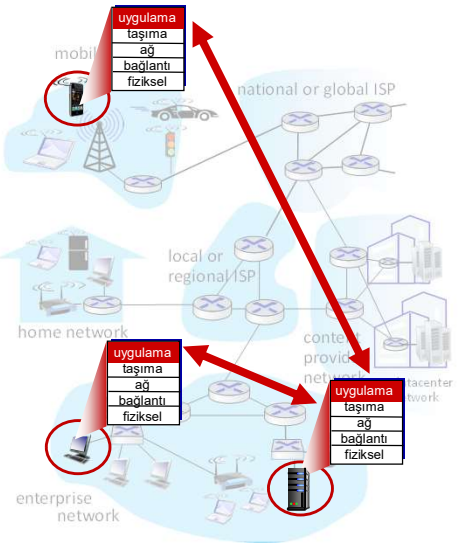
Bir ağ uygulaması oluşturma

programlar yazılmalı:

- (farklı) uç sistemler üzerinde çalıştırma
- ağ üzerinden iletişim
- örneğin, web sunucusu yazılımı tarayıcı yazılımı ile iletişim kurar

ağ çekirdekli cihazlar için yazılım yazmaya gerek yok

- ağ çekirdekli cihazlar kullanıcı uygulamalarını çalıştırmaz
- uç sistemlerdeki uygulamalar hızlı uygulama geliştirme, yayma



Uygulama Katmanı: 2-5

Sunucular genellikle veri merkezlerinde barındırılır, ancak bu her zaman böyle olmayabilir.

Yeni bir uygulama geliştirirken, birden fazla uç sistemde çalışacak yazılımlar yazmak gerekir. Bu yazılımlar **C**, **Java**, **Python** gibi dillerde yazılabilir. Çekirdek ağ cihazları için yazılım geliştirmenize gerek yoktur, çünkü bu cihazlar **ağ katmanında** çalışır. Uygulama yazılımının uç sistemlerle sınırlı olması, ağ uygulamalarının hızlıca geliştirilmesini kolaylaştırır.

Yazılım kodlamaya başlamadan önce, uygulamanın genel mimarisini planlamalıyız. **Uygulama mimarisi**, ağ mimarisinden farklıdır. Ağ mimarisi sabittir ve uygulamalara belirli servisler sağlar, ancak uygulama mimarisi geliştirici tarafından tasarlanır ve uygulamanın uç sistemler üzerinde nasıl yapılandırılacağını belirler. Uygulama geliştiricileri genellikle **istemci-sunucu** veya **eşler arası (P2P)** mimarilerinden birini tercih eder.

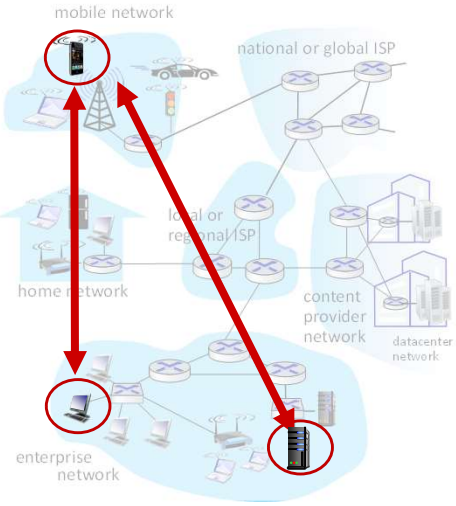
İstemci-sunucu paradigması

Server/sunucu:

- her zaman açık ana bilgisayar
- kalıcı IP adresi
- genellikle veri merkezlerinde, ölçeklendirme mevcut

Clients/istemci :

- iletişim, sunucu ile iletişim
- aralıklı olarak bağlanabilir
- dinamik IP adreslerine sahip olabilir
- birbirleriyle doğrudan iletişim *kurmazlar*
- örnekler: HTTP, IMAP, FTP



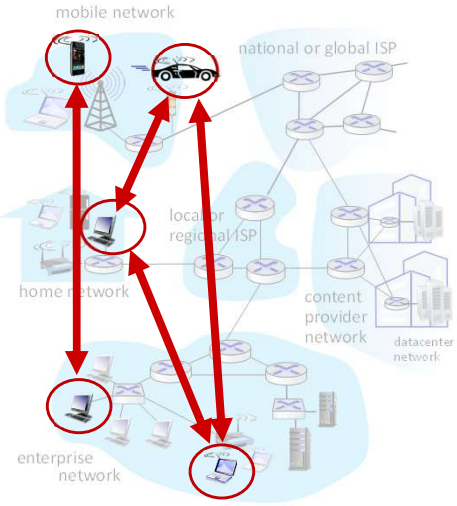
Uygulama Katmanı: 2-6

İstemci-sunucu mimarisinde, istemci cihazlardan gelen isteklere servis veren sürekli açık bir sunucu bulunur. Örneğin, bir **Web sunucusu**, tarayıcılardan gelen isteklere yanıt verir. Bu mimaride, istemciler birbirleriyle doğrudan iletişim kurmaz, sadece sunucuya bağlanır. Sunucunun sabit bir **IP adresi** vardır ve her zaman erişilebilir durumdadır. **Web, FTP, Telnet** ve **e-posta** bu mimariyi kullanan yaygın uygulamalardır.

Tek bir sunucu, yoğun trafiğe sahip uygulamalarda yetersiz kalabilir. Bu yüzden, büyük ölçekli uygulamalar, genellikle birden fazla sunucunun bulunduğu **veri merkezleri** kullanır. Örneğin, **Google**, **Amazon**, **Facebook** gibi servisler, dünya genelindeki veri merkezlerinde servis sunar. Bu veri merkezlerinde yüzbinlerce sunucu bulunabilir ve enerji, bakım, bant genişliği gibi maliyetler söz konusudur.

Peer-peer mimari

- *Her zaman açık sunucu yok*
- Rastgele uç sistemler doğrudan iletişim kurar
- Eşler diğer eşlerden servis talep eder, karşılığında diğer eşlere servis sağlar
  - *Kendi kendine ölçeklenebilirlik – yeni eşler, yeni servis taleplerinin yanı sıra yeni servis kapasitesi de getirir*
- eşler zaman zaman bağlanıyor ve IP adreslerini değiştiriyor
  - Kompleks Yönetimi
- Örnek: P2P dosya paylaşımı



P2P mimarisinde, veri merkezlerindeki özel sunuculara çok az bağımlılık vardır veya hiç yoktur. Bunun yerine, **eşler (peer)** adı verilen, aralıklı olarak bağlanan ve birbirleriyle doğrudan iletişim kuran ana bilgisayarlar kullanılır. Eşler, kullanıcılar tarafından kontrol edilen cihazlardır ve özelleştirilmiş bir sunucudan geçmedikleri için bu mimariye **eşler arası (P2P-peer to peer)** mimari denir. Örneğin, **BitTorrent** popüler bir P2P dosya paylaşım uygulamasıdır.

P2P mimarisinin en önemli özelliklerinden biri **kendi kendine ölçeklenebilmesi**dir. Bir P2P uygulamasında her eş dosya talep ettiğinde iş yükü oluşturur, ancak aynı zamanda dosya dağıtarak sisteme servis kapasitesi ekler. Sunucu altyapısı ve bant genişliği maliyeti düşük olsa da, P2P uygulamaları merkezi olmadığından güvenlik, performans ve güvenilirlik sorunları yaşayabilir.

Proses / İşlem iletişimi

İşlem/process: Bir ana bilgisayar içinde çalışan program

- aynı ana bilgisayar içinde, iki işlem, işlemler arası iletişimi (işletim sistemi tarafından tanımlanır) kullanarak iletişim kurar
- Farklı ana bilgisayarlardaki işlemler *mesaj* alışverişi yaparak iletişim kurar

İstemciler, sunucular

İstemci işlemi: İletişimi başlatan işlem

Sunucu işlemi: Kendisiyle iletişime geçilmesini bekleyen işlem

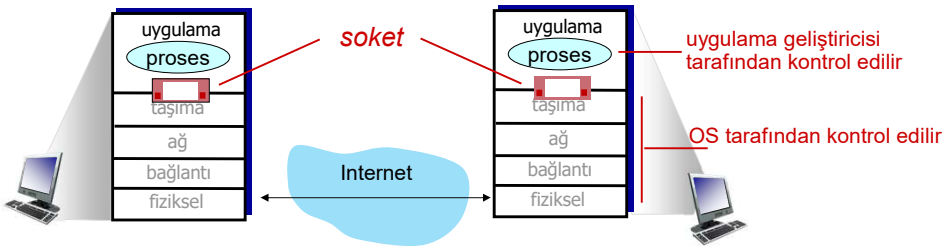
- Not: P2P mimarilerine sahip uygulamaların istemci işlemleri ve sunucu işlemleri vardır

Bir ağ uygulaması geliştirmeden önce, farklı uç sistemlerde çalışan programların nasıl iletişim kurduğunu bilmek önemlidir. Aslında iletişim, programlar değil, her bir uç sistemde çalışan **işlemler** (processes) arasında gerçekleşir. Farklı sistemlerdeki bu işlemler, ağ üzerinden **mesaj alışverişi** yaparak birbirleriyle iletişim kurarlar. Bir işlem mesaj gönderir, diğeri ise alır ve gerekirse yanıt verir.

İstemci ve Sunucu İşlemleri: Ağ uygulamaları, mesaj alışverişinde bulunan işlem çiftlerinden oluşur. Genelde bu işlemlerden biri **istemci** (client), diğeri **sunucu** (server) olarak adlandırılır. Örneğin, bir **Web tarayıcı** istemci işlemi, bir **Web sunucusuyla** iletişim kurar. **P2P dosya paylaşımı** gibi sistemlerde ise eşler hem istemci hem de sunucu olabilir. İletişimi başlatan işlem **istemci**, yanıt bekleyen işlem ise **sunucu** olarak adlandırılır. Örneğin, bir tarayıcı Web sunucusuyla iletişim kurduğunda tarayıcı istemci, sunucu ise sunucu işlemi olur.

Soket

- Süreç:: kendi soketine mesaj gönderir/kendi soketinden mesaj alır
- Soket kapıya benzetilebilir
  - Gönderme süreci mesajı kapıdan dışarı iter
  - Gönderen süreç, mesajı alıcı süreçteki sokete iletmek için kapının diğer tarafındaki taşıma altyapısına güvenir
  - Her iki tarafta birer tane olmak üzere iki soket söz konusudur



Uygulama Katmanı: 2-9

Çoğu ağ uygulaması, birbiriyle mesaj alışverişinde bulunan iki işlemden oluşur. Bu işlemler, ağ üzerinden mesaj göndermek ve almak için **soket** adı verilen bir yazılım arayüzü kullanır. Bir benzetme ile açıklarsak, bir işlem **eve**, soket ise **evin kapısına** benzer. Bir işlem başka bir işleme mesaj göndermek istediğinde, mesajı soketten geçirir ve bu mesaj, alıcı işlemin soketinden (kapısından) içeri girer.

Şekilde görüldüğü gibi, soket, bir ana bilgisayarın **uygulama katmanı** ile **taşıma katmanı** arasındaki bir arayüzdür. Soket, uygulama geliştiricisinin **API** (Uygulama Programlama Arayüzü) ile çalıştığı programlama arayüzüdür. Geliştirici, soketin uygulama katmanı tarafını kontrol eder, ancak taşıma katmanı üzerinde sadece sınırlı bir kontrolü vardır. Geliştirici, taşıma protokolünü (örneğin TCP) seçebilir ve bazı parametreleri (arabellek boyutu gibi) ayarlayabilir.

Adresleme İşlemleri

- mesajları almak için, süreç tanımlayıcıya sahip olmalıdır
- ana cihazın benzersiz 32 bit IP adresi vardır
- *S:* sürecin üzerinde çalıştığı ana bilgisayarın IP adresi süreci tanımlamak için yeterli midir?
  - *C:* Hayır, aynı ana bilgisayarda birçok işlem çalışıyor olabilir
- Tanımlayıcı, ana bilgisayardaki işlemle ilişkili hem IP adresini hem de port numarasını içerir.
- port numarası örnekleri:
  - HTTP sunucusu: 80
  - mail sunucusu: 25
- HTTP mesajlarını gaia.cs.umass.edu web sunucusuna göndermek için:
  - IP adres: 128.119.245.12
  - port numarası: 80

Adresleme İşlemleri

Bir işlemin, başka bir ana bilgisayarda çalışan bir işleme paket gönderebilmesi için alıcı işlemin adresini bilmesi gerekir. Bu adresleme iki temel bilgiyi içerir:
1. Ana bilgisayarın adresi
2. Hedef ana bilgisayardaki alıcı işlemi tanımlayan bir tanımlayıcı

İnternet'te, ana bilgisayarlar **IP adresleri** ile tanımlanır. IP adresi, her ana bilgisayara benzersiz bir kimlik sağlayan 32 bitlik bir numaradır. Ancak, bir ana bilgisayarda aynı anda birden fazla uygulama çalışabileceğinden, **bağlantı noktası (port) numaraları** kullanılarak hangi işlemin hedeflendiği belirlenir.

- **Bağlantı noktası numaraları:** Belirli uygulamalara atanmıştır.
- **80 numaralı bağlantı noktası:** Web sunucuları (HTTP)
- **25 numaralı bağlantı noktası**: Posta sunucuları (SMTP)

Bu standart port/bağlantı numaraları sayesinde, istemciler doğru hizmete ulaşabilir. Daha fazla ayrıntı ve port numarası listesine **www.iana.org** adresinden erişilebilir.

Uygulama katmanı protokolü aşağıdakileri tanımlar:

- Değişimi yapılan mesaj tipi,
  - örnek request, response
- Mesaja ait syntax:
  - mesajlardaki hangi alanlar ve alanların nasıl tanımlandığı
- Mesaja ait semantic:
  - alanlardaki bilginin anlamı
- süreçlerin mesajları ne zaman ve nasıl gönderip yanıtlayacağına ilişkin kurallar

Açık protokoller:

- RFC ile tanımlanırlar, herkesin protokol tanımına ve yapısına erişimi vardır
- birlikte çalışabilirliğe olanak sağlar
- örnek, HTTP, SMTP

tescilli protokoller:

- örneğin, Skype

Soket ve Taşıma Katmanı Protokolü Seçimi

Soket, **uygulama işlemi ile taşıma katmanı protokolü** arasındaki arayüzdür. Uygulama, mesajlarını soket üzerinden iletir ve taşıma katmanı protokolü, bu mesajların alıcı işlemin soketine eksiksiz ulaşmasını sağlar.

Ağlarda (örneğin, internet) **birden fazla taşıma katmanı protokolü** bulunur. Bir uygulama geliştirirken, bu protokoller arasından en uygun olanını seçmelisiniz. Seçim yaparken, mevcut protokollerin sağladığı hizmetleri değerlendirir ve uygulamanızın ihtiyaçlarına en uygun olanını tercih edersiniz.

Bu seçim, **tren** veya **uçak** arasında seyahat modu seçmeye benzetilebilir. Her ikisi farklı hizmetler sunar; tren, şehir merkezlerinde dururken uçak, daha hızlı ulaşım sağlar.

Bir uygulama hangi taşıma servisine ihtiyaç duyar?

Veri bütünlüğü

- bazı uygulamalar (örn. dosya aktarımı, web işlemleri) %100 güvenilir veri aktarımı gerektirir
- diğer uygulamalar (örn. ses) bir miktar kaybı tolere edebilir

Aktarım Hızı

- bazı uygulamalar (örneğin multimedya) "etkili" olmak için minimum miktarda iş hacmi gerektirir
- diğer uygulamalar ("elastik uygulamalar") elde ettikleri her türlü verimi kullanırlar

Taşıma Katmanı Hizmetleri

Taşıma katmanı protokollerinin sağladığı olası hizmetler dört ana başlıkta toplanır:

1. **Güvenilir Veri Aktarımı**: Verilerin eksiksiz ve hatasız iletilmesi.
2. **Aktarım Hızı**: Belirli bir hızda veri iletim garantisi.
3. **Zamanlama**: Verilerin gecikmesiz ve doğru sırayla ulaşması.
4. **Güvenlik**: Şifreleme, veri bütünlüğü ve kimlik doğrulama hizmetleri.

Her protokol bu hizmetleri farklı düzeyde sunar, dolayısıyla seçim yaparken uygulamanızın hangi hizmetlere öncelik verdiğini belirlemek önemlidir.

Güvenilir Veri Aktarımı

Ağ ortamında, paketler kaybolabilir veya bozulabilir. Örneğin, bir paket bir yönlendiricideki tampon belleğin dolması nedeniyle düşebilir veya hatalı bitler içeren bir paket atılabilir. Elektronik posta, dosya aktarımı, uzaktan erişim, web belge aktarımları ve finansal işlemler gibi birçok uygulama için veri kaybı ciddi sorunlara yol açar. Bu tür uygulamalar için, gönderilen verinin eksiksiz ve hatasız bir şekilde alıcıya ulaşması kritik önemdedir.

Güvenilir veri aktarımı, taşıma katmanı protokolünün sağladığı en önemli hizmetlerden biridir. Bu hizmet, verilerin soketten iletilmesinden sonra alıcıya doğru sırayla ve eksiksiz ulaşacağını garanti eder. Güvenilir bir protokol kullanıldığında, gönderen taraf verilerin alıcıya hatasız ulaşacağına güvenebilir.

Güvenilir olmayan veri aktarımı, Taşıma katmanı güvenilir veri aktarımı sağlamazsa, verilerin bir kısmı kaybolabilir. Ancak, konuşma sesi veya video gibi **kayba dayanıklı uygulamalar** için bu durum kabul edilebilir. Örneğin, ses veya görüntüdeki küçük veri kayıpları, kalitede hafif bir bozulmaya yol açar, ancak uygulamanın kullanılabilirliğini etkilemez.

Bir uygulama hangi taşıma servisine ihtiyaç duyar?

Veri bütünlüğü

- bazı uygulamalar (örn. dosya aktarımı, web işlemleri) %100 güvenilir veri aktarımı gerektirir
- diğer uygulamalar (örn. ses) bir miktar kaybı tolere edebilir

Aktarım Hızı

- bazı uygulamalar (örneğin multimedya) "etkili" olmak için minimum miktarda iş hacmi gerektirir
- diğer uygulamalar ("elastik uygulamalar") elde ettikleri her türlü verimi kullanırlar

Aktarım Hızı

Aktarım hızı, gönderici işlemin alıcıya bit gönderme hızıdır. Ağdaki diğer oturumlar bant genişliğini paylaştığı için bu hız zamanla dalgalanabilir. **Garantili aktarım hızı** hizmeti, taşıma katmanı protokolünün sunabileceği önemli bir özelliktir. Bu hizmette, uygulama belirli bir hızda (örneğin, 32 kbps) veri iletim garantisi talep edebilir ve taşıma protokolü, bu hızın altına düşülmemesini sağlar.

•**Bant genişliğine duyarlı uygulamalar**: Özellikle internet telefonları gibi uygulamalar belirli bir aktarım hızına ihtiyaç duyar. Eğer protokol bu hızı sağlayamazsa, uygulamanın daha düşük hızda kodlama yapması gerekebilir. Aktarım hızının yarısına veya daha azına düşülmesi durumunda, uygulama kullanılamaz hale gelebilir.

•**Adaptif kodlama**: Bazı multimedya uygulamaları, mevcut aktarım hızına göre ses veya video kalitesini ayarlamak için adaptif kodlama kullanır. Böylece, düşük bant genişliğinde bile çalışabilirler.

•**Elastik uygulamalar**: E-posta, dosya transferi ve web tarayıcıları gibi uygulamalar, mevcut aktarım hızına bağlı olmadan çalışabilir. Ancak, hız ne kadar yüksekse performans o kadar iyi olur. Bu uygulamalar için daha fazla hız her zaman avantaj sağlar.

Bir uygulama hangi taşıma servisine ihtiyaç duyar?

Zamanlama

- bazı uygulamalar (örn. internet telefonu, interaktif oyunlar) "etkili" olmak için düşük gecikme gerektirir

Güvenlik

- şifreleme, veri bütünlüğü, …

Zamanlama

Taşıma katmanı protokolü, veri iletimi için zamanlama garantileri sunabilir. Bu garantiler, farklı şekillerde olabilir. Örneğin, göndericiden çıkan her bitin, en geç 100 ms içinde alıcıya ulaşması bir zamanlama garantisidir. Bu tür hizmetler, **internet telefonları, sanal ortamlar, telekonferanslar** ve **çok oyunculu oyunlar** gibi **etkileşimli ve gerçek zamanlı** uygulamalar için kritiktir.

- **Zamanlama hassasiyeti:** Örneğin, internet telefonundaki uzun gecikmeler, konuşmada doğal olmayan duraklamalara yol açabilir. Aynı şekilde, bir oyunda veya sanal bir ortamda yapılan bir eylem ile karşıdan alınan tepki arasındaki uzun gecikme, kullanıcı deneyimini olumsuz etkileyebilir.

Gerçek zamanlı olmayan uygulamalar için, düşük gecikme genellikle tercih edilse de, bu uygulamalar için uçtan uca gecikmelerde sıkı bir sınır gerekmez. Ancak, zamanlama garantileri gerçek zamanlı uygulamaların performansını ve kullanıcı deneyimini artırmak için oldukça önemlidir.

Güvenlik

Taşıma protokolleri, uygulamalara çeşitli güvenlik hizmetleri sunabilir. Örneğin, bir taşıma protokolü, gönderen ana bilgisayarda veriyi şifreleyebilir ve alıcı tarafında veriyi şifresini çözüp iletebilir. Bu sayede, veriler aktarım sırasında gözlemlense bile iki işlem arasında **gizlilik** sağlanır. Taşıma protokolleri, gizliliğin yanı sıra şu güvenlik hizmetlerini de sunabilir:

•**Veri Bütünlüğü**: Verilerin aktarım sırasında değiştirilmediğini garanti eder.

•**Uç Nokta Kimlik Doğrulaması**: Verinin gerçekten doğru kaynaktan geldiğini doğrular.

Bu güvenlik özellikleri, taşıma protokollerinin temel fonksiyonlarına ek olarak uygulanabilir.

Taşıma servisi gereksinimleri : ortak uygulamalar

application	Veri kaybı	Aktarım hızı	Zaman hassasiyeti
Dosya aktarımı/download	kayıpsız	esnek	yok
e-posta	kayıpsız	esnek	yok
Web belgeleri	kayıpsız	esnek	yok
Gerçek zamanlı ses ve video	kayıp-toleranslı	ses: 5Kbps-1Mbps video:10Kbps-5Mbps	var, 10 ms
Kayıtlı ses/video	kayıp-toleranslı	Yukarı ile aynı	evet, birkaç sn.
Etkileşimli oyunlar	kayıp-toleranslı	birkaç kbps-10kbps	evet, 100 ms
Telefon mesajlaşması	kayıpsız	esnek	var ve yok

İnternet taşıma protokol servisleri

TCP servisi:

- *güvenilir veri taşıması:* gönderme ve alma işlemleri arasında
- *akış kontrolü:* gönderici alıcıyı bunaltamaz
- *tıkanıklık kontrolü:* ağ aşırı yüklendiğinde göndericiyi yavaşlatma
- *sağlamaz:* zamanlama, minimum iş gücü garantisi, güvenlik
- *bağlantı odaklı:* istemci ve sunucu işlemleri arasında kurulum gerekli

UDP servisi:

- *güvenilmez veri taşıması:* gönderme ve alma işlemleri arasında
- *sağlamaz:* güvenilirlik, akış kontrolü, tıkanıklık kontrolü, zamanlama, iş gücü garantisi, güvenlik veya bağlantı kurulumu.

S: UDP neden var?

TCP Servisleri

TCP servis modeli, iki temel servis sunar: bağlantıya yönelik servis ve güvenilir veri aktarımı.

- **Bağlantıya Yönelik servis:** TCP, veri akışı başlamadan önce istemci ve sunucunun taşıma katmanı bilgilerini birbirleriyle paylaşmasını sağlayan bir "el sıkışma" süreci yürütür. Bu aşama, iki tarafı veri alışverişine hazırlayarak bir TCP bağlantısının kurulmasını sağlar. Bu bağlantı, tam çift yönlüdür; yani, istemci ve sunucu aynı anda birbirine veri gönderebilir. Veri aktarımı tamamlandıktan sonra bağlantının kesilmesi gerekir.
- **Güvenilir Veri Aktarımı:** TCP, gönderilen verilerin hatasız ve doğru sırayla alıcıya ulaşmasını sağlar. Uygulama, bir bayt akışını bir soket üzerinden gönderdiğinde, TCP bu akışı alıcıya eksiksiz olarak iletir.

Ayrıca, TCP'nin internetin dengeli çalışması için kritik olan bir tıkanıklık kontrol mekanizması da vardır. Bu mekanizma, ağda tıkanıklık olduğunda veri gönderen tarafı yavaşlatarak ağın aşırı yüklenmesini önler. Tıkanıklık kontrolü, her bağlantının ağdaki bant genişliğinden adil bir pay almasını sağlamaya çalışır.

İnternet taşıma protokol servisleri

TCP servisi:

- *güvenilir veri taşıması:* gönderme ve alma işlemleri arasında
- *akış kontrolü:* gönderici alıcıyı bunaltamaz
- *tıkanıklık kontrolü:* ağ aşırı yüklendiğinde göndericiyi yavaşlatma
- *sağlamaz:* zamanlama, minimum iş gücü garantisi, güvenlik
- *bağlantı odaklı:* istemci ve sunucu işlemleri arasında kurulum gerekli

UDP servisi:

- *güvenilmez veri taşıması:* gönderme ve alma işlemleri arasında
- *sağlamaz:* güvenilirlik, akış kontrolü, tıkanıklık kontrolü, zamanlama, iş gücü garantisi, güvenlik veya bağlantı kurulumu.

S: UDP neden var?

UDP Servisleri

UDP, basit ve hafif bir taşıma protokolü olup, minimum düzeyde servis sunar. İşte temel özellikleri:

- **Bağlantısız servis**: UDP, veri iletimine başlamadan önce istemci ve sunucu arasında bir el sıkışma süreci gerektirmez. Bu nedenle, iki işlem arasında doğrudan veri gönderilebilir.

- **Güvenilir Olmayan Veri Aktarımı**: UDP, gönderilen mesajların alıcıya ulaşacağına dair bir garanti vermez. Gönderilen bir mesaj kaybolabilir ya da hasar görebilir. Ayrıca, alıcıya ulaşan mesajlar, gönderildiği sıradan farklı bir sırada gelebilir.

Bu özellikleriyle UDP, hız ve basitlik gerektiren uygulamalar için idealdir, ancak veri kaybının tolere edilemeyeceği durumlar için uygun değildir.

UDP, tıkanıklık kontrol mekanizmasına sahip değildir. Bu nedenle, UDP kullanan bir gönderici, verileri ağ katmanına dilediği hızda iletebilir. Ancak, bu hızın uçtan uca aktarım hızı olduğu anlamına gelmez. Araya giren ağ bağlantılarının sınırlı kapasitesi veya tıkanıklık gibi faktörler, veri iletim hızını düşürebilir. Bu nedenle, UDP her ne kadar hız açısından esnek olsa da, ağın durumu veri akışını etkileyebilir.

İnternet taşıma protokol servisleri

TCP servisi:

- *güvenilir veri taşıması:* gönderme ve alma işlemleri arasında
- *akış kontrolü:* gönderici alıcıyı bunaltamaz
- *tıkanıklık kontrolü:* ağ aşırı yüklendiğinde göndericiyi yavaşlatma
- *sağlamaz:* zamanlama, minimum iş gücü garantisi, güvenlik
- *bağlantı odaklı:* istemci ve sunucu işlemleri arasında kurulum gerekli

UDP servisi:

- *güvenilmez veri taşıması:* gönderme ve alma işlemleri arasında
- *sağlamaz:* güvenilirlik, akış kontrolü, tıkanıklık kontrolü, zamanlama, iş gücü garantisi, güvenlik veya bağlantı kurulumu.

S: UDP neden var?

UDP Servisleri devam…

Taşıma protokolü servislerini dört boyutta ele aldık: güvenilir veri aktarımı, aktarım hızı, zamanlama ve güvenlik. TCP, uçtan uca güvenilir veri aktarımı sağlar ve güvenliği artırmak için TLS ile desteklenebilir. Ancak, TCP ve UDP'nin aktarım hızı ve zamanlama garantisi sağlamaması dikkat çekicidir. Bu durum, zamana duyarlı uygulamaların (örneğin, internet telefonu) internet üzerinde çalışamayacağı anlamına gelmez. İnternet, zamana duyarlı uygulamaları destekleyecek şekilde tasarlandığı için bu uygulamalar genellikle sorunsuz çalışır. Ancak, yüksek gecikme veya düşük aktarım hızı gibi durumlarda sınırlamalar ortaya çıkar.

Bazı popüler internet uygulamaları TCP'yi kullanır, çünkü TCP'nin güvenilir veri aktarımı sağladığı ve verilerin hedefine ulaşacağını garanti ettiği bilinir. E-posta, uzak terminal erişimi, Web ve dosya aktarımı bu uygulamalara örnektir. Öte yandan, internet telefon uygulamaları (Skype gibi) bir miktar veri kaybını tolere edebilir ve belirli bir hız gereksinimi olduğundan, çoğunlukla UDP kullanmayı tercih eder. Bu sayede TCP'nin tıkanıklık kontrol mekanizmasından kaçınırlar. Ancak, birçok güvenlik duvarı UDP trafiğini engelleyebileceğinden, bu uygulamalar UDP başarısız olursa TCP'yi yedek olarak kullanmak üzere tasarlanır.

İnternet taşıma protokol servisleri

TCP servisi:

- *güvenilir veri taşıması:* gönderme ve alma işlemleri arasında
- *akış kontrolü:* gönderici alıcıyı bunaltamaz
- *tıkanıklık kontrolü:* ağ aşırı yüklendiğinde göndericiyi yavaşlatma
- *sağlamaz:* zamanlama, minimum iş gücü garantisi, güvenlik
- *bağlantı odaklı:* istemci ve sunucu işlemleri arasında kurulum gerekli

UDP servisi:

- *güvenilmez veri taşıması:* gönderme ve alma işlemleri arasında
- *sağlamaz:* güvenilirlik, akış kontrolü, tıkanıklık kontrolü, zamanlama, iş gücü garantisi, güvenlik veya bağlantı kurulumu.

S: UDP neden var?

UDP neden var?

Düşük Gecikme: UDP, TCP'nin aksine bağlantı kurma, paket sıralama veya hata düzeltme gibi ekstra adımlara ihtiyaç duymaz. Bu nedenle, veri aktarımı daha hızlı gerçekleşir. Bu durum, internet telefonları, canlı video akışı ve çevrimiçi oyunlar gibi gerçek zamanlı uygulamalar için kritik önem taşır. Bu tür uygulamalarda, bazı veri kayıpları yerine verinin hızlıca iletilmesi daha önemlidir.

Basitlik ve Hafiflik: UDP, veri paketlerini doğrudan gönderen ve alan, sade bir protokoldür. TCP'de bulunan tıkanıklık kontrolü, hata düzeltme ve bağlantı kurma gibi mekanizmalar UDP'de yoktur. Bu basit yapı, veri paketlerinin hızlı ve verimli bir şekilde iletilmesini sağlar. Ayrıca, UDP, TCP'ye göre daha az sistem kaynağı gerektirir, bu da belirli uygulamalarda avantaj sağlar.

Tıkanıklık Kontrolünden Kaçınma: TCP, ağ tıkanıklığını yönetmek için çeşitli mekanizmalara sahiptir, bu da veri aktarım hızını sınırlandırabilir. UDP, bu tür kontrollerden yoksundur, bu da belirli hızda veri akışı gereken uygulamalar için daha uygun hale getirir. Örneğin, video akışında, belirli paketlerin kaybolmasındansa akışın kesintisiz devam etmesi daha değerlidir.

Uygulamaya Özgü Hata Yönetimi: UDP, hata kontrolü veya veri sıralaması sağlamaz; bu sorumluluk, uygulamalara aittir. Bu esneklik, uygulamaların veri iletimini kendi gereksinimlerine göre optimize etmesine olanak tanır. Örneğin, bazı uygulamalar veri kaybını önemsemeyip, öncelikli olarak hızlı veri akışı isteyebilir.

Internet uygulamaları, uygulama katmanı protokolleri ve temel taşıma protokolleri

Uygulama	Uygulama katmanı protokolü	Taşıma protokolü
Dosya aktarımı/download	FTP [RFC 959]	TCP
Elektronik posta	SMTP [RFC 5321]	TCP
Web dökümanlar	HTTP 1.1 [RFC 7230]	TCP
Internet telefon	SIP [RFC 3261], RTP [RFC 3550], veya özel (Skype)	TCP veya UDP
Multimedya akışı	HTTP (YouTube), DASH	TCP
İnteraktif oyunlar	WOW, FPS (tescilli)	UDP veya TCP

Ağ işlemlerinde, soketler aracılığıyla mesaj gönderilerek iletişim kurulur. Ancak bu mesajların nasıl yapılandırıldığını anlamak önemlidir. Bu mesajlar, çeşitli alanlar içerir ve bu alanların belirli anlamları vardır. Ayrıca, işlemlerin mesajları ne zaman gönderdiği de önemli bir konudur. Bu noktada, uygulama katmanı protokolleri devreye girer. Uygulama katmanı protokolü, farklı sistemlerde çalışan uygulamaların mesaj alışverişini düzenler ve şunları belirler:

- Gönderilen mesaj türleri (istek/request ve yanıt/response mesajları gibi).
- Mesajların yapısı ve içerdikleri alanların düzeni.
- Mesajlardaki alanların taşıdığı bilgiler ve anlamları.
- Bir işlemin mesajları ne zaman ve nasıl göndereceği, bu mesajlara ne zaman ve nasıl yanıt vereceği ile ilgili kurallar.

Bazı uygulama katmanı protokolleri, RFC'lerde tanımlanmış ve kamuya açıkken, bazıları tescillidir ve gizlidir. Örneğin, HTTP, Web'in uygulama katmanı protokolü olarak RFC 7230'da tanımlıdır. HTTP kurallarına uyan bir tarayıcı, bu kurallara uyan herhangi bir web sunucusundan sayfa alabilir. Öte yandan, Skype tescilli protokoller kullanır.

Ağ uygulamaları ve uygulama katmanı protokolleri arasında fark vardır. Bir uygulama katmanı protokolü, ağ uygulamasının sadece bir parçasıdır. Örneğin, Web uygulaması HTML, tarayıcılar, web sunucuları ve HTTP gibi birçok bileşen içerir. HTTP, tarayıcı ve sunucu arasındaki mesaj alışverişini düzenler. Benzer şekilde, Netflix video servisi de DASH protokolü, video sunucuları ve istemci uygulamalar gibi bileşenlerden oluşur. DASH, Netflix sunucusu ve istemcisi arasındaki mesaj alışverişini tanımlar. Her iki protokol de uygulamanın önemli, ancak tek başına olmayan parçalarıdır.

TCP'nin Güvenliğini Sağlama

Vanilya TCP ve UDP soketleri:

- Şifreleme yok
- Sokete gönderilen açık metin parolaları açık metin olarak İnternet'te dolanır(!)

Transport Layer Security (TLS)

- şifreli TCP bağlantıları sağlar
- Veri bütünlüğü
- Uç nokta kimlik doğrulaması

Uygulama katmanında uygulanan TLS

- uygulamalar TLS kitaplıklarını kullanır ve bu kitaplıklar da sırayla TCP kullanır

TLS soket API

- sokete gönderilen açık metin interneti şifreli olarak geçer

"Vanilla TCP" ifadesi, TCP'nin **güvenlik eklemeleri veya özel uzantılar olmadan** en temel haliyle kullanıldığını ifade eder. "Vanilla" kelimesi burada, bir teknolojinin **orijinal, değiştirilmemiş ve standart** versiyonuna atıfta bulunur. Yani, vanilla TCP, sadece TCP'nin sağladığı **güvenilir veri iletimi ve tıkanıklık kontrolü** gibi temel işlevlerle çalışır. Bu haliyle vanilla TCP, **şifreleme veya ek güvenlik önlemleri** içermez. Güvenli veri aktarımı gerektiğinde, **TLS** gibi ek protokoller devreye girer. Örneğin, finansal işlemler veya hassas bilgilerin iletiminde, vanilla TCP yetersiz kalabilir. Bu tür durumlarda, TCP'nin üzerine güvenlik katmanları eklenerek veri aktarımı güvenli hale getirilir. Vanilla TCP ve UDP Soketlerinde, yani TCP ve UDP'nin temel sürümlerinde, veri şifrelemesi bulunmaz. Gönderilen veriler (örneğin, şifreler) internet üzerinde düz metin olarak (şifrelenmeden) taşınır. Bu, güvenlik açısından çok ciddi bir risk oluşturur.

Transport Layer Security (TLS)

•**Şifreli TCP bağlantıları sağlar**: TLS, TCP tabanlı iletişimlerde verileri şifreleyerek güvenli hale getirir.

•**Veri bütünlüğü**: TLS, verilerin aktarım sırasında bozulmadığını garanti eder.

•**Uç nokta kimlik doğrulaması**: İletişim kurulan tarafların kimliklerini doğrulamak için kullanılır.

TLS'in Uygulama Katmanında Kullanımı

•**Uygulamalar, TLS kütüphanelerini kullanır**: TLS, uygulama katmanında uygulanır ve arka planda TCP ile çalışır.

•**TLS Soket API'si**: Düz metin olarak gönderilen veriler, TLS aracılığıyla şifrelenerek internet üzerinde güvenli bir şekilde taşınır.

Uygulama katmanı: Genel bakış/yol haritası

- Ağ uygulamalarının prensipleri
- Web ve HTTP
- E-mail, SMTP, IMAP
- Domain Name System DNS
- P2P uygulamalar
- Video akışı ve içerik dağıtım ağları
- UDP ve TCP ile soket programlama

2.2 Web ve HTTP

1990'ların başlarına kadar internet, ağırlıklı olarak **araştırmacılar, akademisyenler** ve **üniversite öğrencileri** tarafından dosya transferi, haberleşme ve elektronik posta gönderimi için kullanılıyordu. Bu dönemde internet, akademik çevrelerin dışında pek yaygın değildi. Ancak, **World Wide Web**'in ortaya çıkışı (Tim Berners-Lee, 1994) internetin kullanımını kökten değiştirdi ve herkesin erişimine açtı. Web, interneti sadece birçok ağdan biri olmaktan çıkarıp, merkezi bir bilgi ve iletişim ağı haline getirdi.

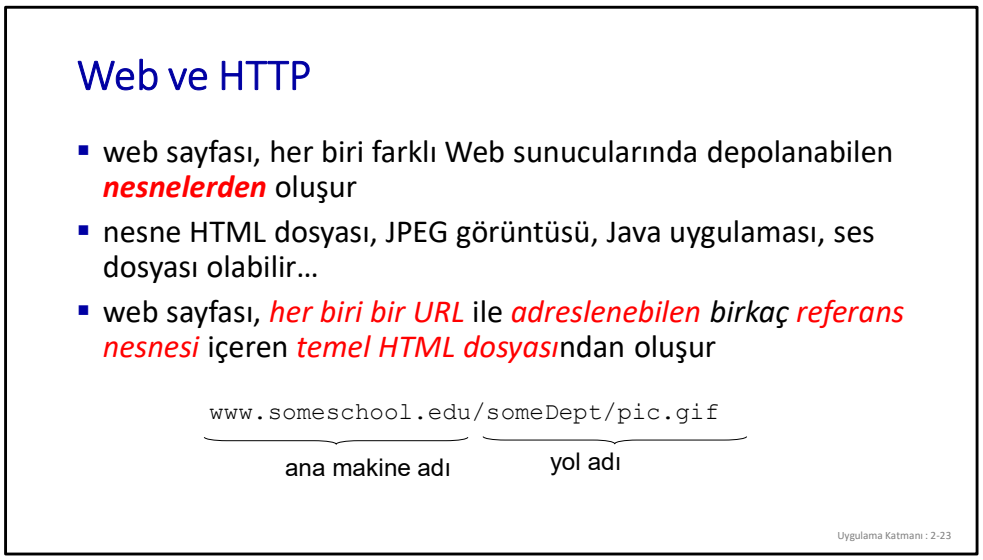
Web'in Çekici Yönleri

Web'in sunduğu en çekici özelliklerden biri, **talep üzerine erişim** sağlamasıydı. Kullanıcılar, istedikleri zaman bilgiye erişebiliyor ve web üzerinde gezinebiliyordu. Web, bu özelliğin yanı sıra başka birçok avantaj sundu:

- **Kolay İçerik Yayını:** Herkesin düşük maliyetle içerik yayınlayabilmesi.
- **Arama Motorları ve Bağlantılar:** Sonsuz bilgi denizinde arama ve bağlantılarla gezinmeyi mümkün kılar.
- **Etkileşimli İçerik:** Formlar, scriptler, videolar ve diğer multimedya içerikleriyle etkileşim sağlar.

Web Tabanlı Uygulamaların Gelişimi

Web, yalnızca bilgi paylaşımı değil, aynı zamanda platform olarak **Gmail, YouTube, Instagram, Google Haritalar** gibi web tabanlı veya mobil internet uygulamalarının temelini oluşturmuştur. Web ve HTTP protokolü, bu uygulamaların çalışması için gerekli altyapıyı sağlar ve interneti günlük hayatın vazgeçilmez bir parçası haline getirmiştir.



2.2.1 HTTP'ye Genel Bakış

HTTP (HyperText Transfer Protocol), **web'in temelini oluşturan uygulama katmanı protokolüdür** ve [RFC 1945], [RFC 7230] ve [RFC 7540] standartlarında tanımlanmıştır. HTTP, istemci ve sunucu arasında mesaj alışverişini düzenleyerek web üzerindeki iletişimi sağlar.

HTTP İstemci ve Sunucu Yapısı

•**İstemci tarafı**: Web tarayıcıları (ör. Chrome, Firefox, Internet Explorer) HTTP'nin istemci tarafında çalışır. Tarayıcılar, web sitelerine istek gönderir ve sunucudan yanıt alır.

•**Sunucu tarafı**: Web sunucuları (ör. Apache, Microsoft IIS), HTTP isteklerine yanıt verir ve ilgili web nesnelerini (HTML dosyaları, resimler, videolar gibi) sunar.

HTTP Mesajlaşma ve Yapısı

HTTP, **istemci ve sunucunun mesajları nasıl transfer edeceğini ve mesaj yapısını** tanımlar. İstemci, sunucuya bir istek (request) mesajı gönderir ve sunucu buna bir yanıt (response) mesajıyla karşılık verir.

Web Terminolojisi ve URL Yapısı

Bir web sayfası, farklı nesnelerden (HTML dosyaları, resimler, videolar vb.) oluşur ve bu nesneler bir **URL (Uniform Resource Locator)** ile adreslenir. URL, iki temel bileşenden oluşur:

1.Ana bilgisayar adı: Sunucuyu tanımlar (ör. www.someSchool.edu).

2.Yol adı: İlgili nesnenin sunucudaki konumunu gösterir (ör. /someDepartment/picture.gif).

Örneğin, http://www.someSchool.edu/someDepartment/picture.gif adresindeki:

•**Ana bilgisayar adı**: www.someSchool.edu

•**Yol**: /someDepartment/picture.gif

Popüler Web Sunucuları

HTTP'nin sunucu tarafında çalışan yaygın web sunucuları arasında **Apache** ve **Microsoft Internet Information Server (IIS)** bulunur. Bu sunucular, her biri bir URL ile adreslenebilen web nesnelerini barındırır ve HTTP isteklerine yanıt verir. HTTP, web'in işleyişini düzenleyen ve tarayıcı ile sunucu arasındaki iletişimi sağlayan temel protokoldür.

HTTP genel bakış

HTTP: hypertext transfer protocol

- Web'in uygulama katmanı protokolü
- İstemci/sunucu (client/server) modeli:
  - *istemci:* Web nesnelerini isteyen, alan (HTTP protokolünü kullanarak) ve "görüntüleyen" tarayıcı
  - *sunucu:* Web sunucusu isteklere yanıt olarak nesneler gönderir (HTTP protokolünü kullanarak)

HTTP'nin İşleyişine Genel Bakış

HTTP (HyperText Transfer Protocol), **web istemcilerinin sunuculardan nasıl sayfa talep edeceğini** ve sunucuların bu sayfaları nasıl ileteceğini tanımlar. İstemci (tarayıcı) ve sunucu arasındaki temel etkileşim şu şekilde özetlenebilir:

1.Kullanıcı talebi: Bir kullanıcı, örneğin bir bağlantıya tıkladığında, tarayıcı istenen web sayfasının nesneleri için HTTP istek mesajları gönderir.

2.İstek ve yanıt: Tarayıcı, sayfadaki tüm nesneler için sunucuya ayrı ayrı istekler gönderir (ör. HTML dosyası, resimler, CSS ve JavaScript dosyaları).

3.Sunucu yanıtı: Sunucu, her istek için bir **HTTP yanıt mesajı** gönderir ve istenen nesneyi tarayıcıya iletir.

Bu mesaj alışverişi, **istemci-tarayıcı ve sunucu** arasındaki temel iletişim modelini oluşturur. Her nesne için yapılan bu bireysel istek-yanıt yapısı, web'in modüler ve dinamik doğasını mümkün kılar.

HTTP, talep edilen içeriğin aktarımını nasıl yöneteceğini belirlerken, yanıtın türünü (ör. 200 OK, 404 Not Found) ve verinin hangi biçimde döndürüleceğini (HTML, JSON, vb.) de tanımlar. Tarayıcı, tüm bu nesneleri birleştirerek kullanıcının görüntülediği web sayfasını oluşturur.

HTTP genel bakış (devam)

HTTP, TCP kullanır:

- istemci sunucuya TCP bağlantısı başlatır (soket oluşturur), port 80
- sunucu istemciden gelen TCP bağlantısını kabul eder
- Tarayıcı (HTTP istemcisi) ve Web sunucusu (HTTP sunucusu) arasında HTTP mesajları değiş tokuş edilir (uygulama katmanı protokol mesajları)
- TCP bağlantısı kapatılır

HTTP "durumsuz"dur

- sunucu geçmiş *istemci* istekleri hakkında hiçbir bilgi tutmaz

Ayrıca

"durumu" koruyan protokoller karmaşıktır!

- geçmiş tarih (durum) muhafaza edilmelidir
- sunucu/istemci çökerse, "durum" hakkındaki bilgi tutarsız olabilir

HTTP ve TCP İlişkisi

HTTP, **temel aktarım protokolü olarak TCP'yi** kullanır. Bir HTTP istemcisi (tarayıcı), bir web sayfasına erişmek istediğinde, önce sunucuyla bir **TCP bağlantısı** başlatır. Bağlantı kurulduktan sonra, istemci ve sunucu arasındaki veri alışverişi, TCP soketleri üzerinden gerçekleşir.

•**İstemci tarafında**: Tarayıcı, HTTP istek mesajlarını soket arayüzüne gönderir ve yanıtları aynı arayüzden alır.

•**Sunucu tarafında**: Web sunucusu, HTTP istek mesajlarını soket arayüzünden alır ve yanıt mesajlarını soket arayüzüne iletir.

Bu sistem, **katmanlı mimarinin** faydasını gösterir: HTTP, **TCP'nin sağladığı güvenilir veri aktarımından** yararlanır ve ağdaki olası veri kaybı veya paketlerin sıralamasının bozulması gibi sorunlarla ilgilenmek zorunda kalmaz. TCP, bu sorunları arka planda çözer ve HTTP'ye **eksiksiz veri iletimi** sağlar.

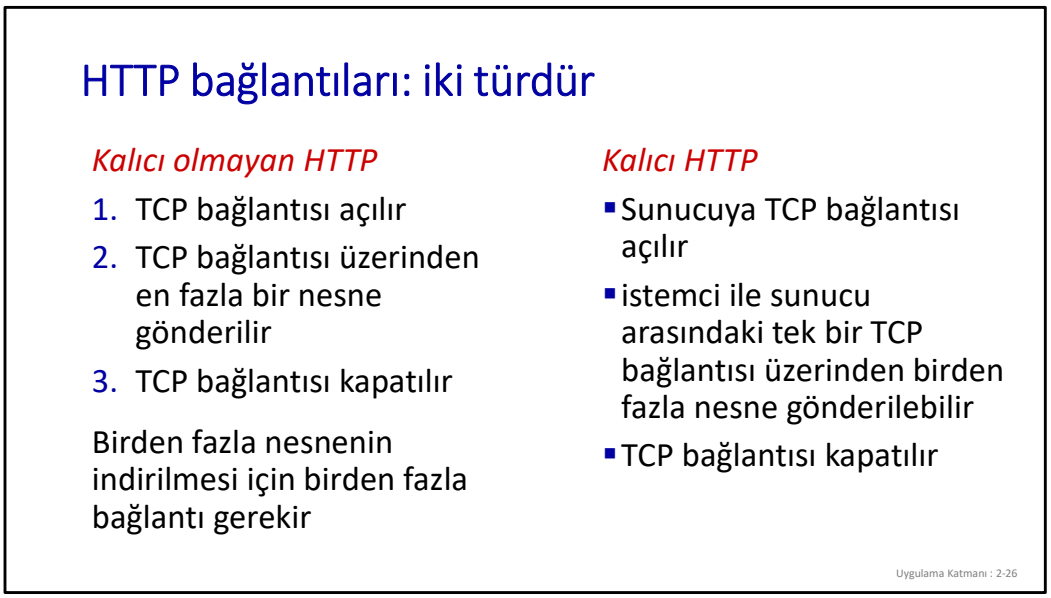
HTTP'nin Durumsuz (Stateless) Yapısı

HTTP, **stateless protocol (durumsuz protokol)** olarak çalışır. Bu, sunucunun istemci hakkında herhangi bir bilgi tutmadığı anlamına gelir. Örneğin, bir kullanıcı aynı dosyayı birkaç saniye içinde tekrar isterse, sunucu dosyayı daha önce sunduğunu hatırlamaz ve isteği yeniden işler. Bu yapı, sunucunun milyonlarca farklı tarayıcıdan gelen isteklere hizmet vermesini kolaylaştırır.

HTTP Sürümleri

HTTP'nin ilk sürümü olan **HTTP/1.0**, 1990'ların başında tanıtılmıştır ([RFC 1945]). **HTTP/1.1** ([RFC 7230]) sürümü, 2020 itibarıyla yaygın olarak kullanılmaktadır ve daha gelişmiş özellikler sunar. Bununla birlikte, **HTTP/2** ([RFC 7540]) desteği de hızla artmaktadır ve birçok tarayıcı ile sunucu bu yeni sürümü desteklemektedir. Bölüm sonunda, HTTP/2'ye yönelik bir giriş yapılacaktır.

Bu yapı sayesinde, HTTP internetin temel taşı olarak veri akışını yönetirken TCP güvenilirliği sağlar ve istemci-sunucu mimarisinin sorunsuz çalışmasını mümkün kılar.



2.2.2 Kalıcı ve Kalıcı Olmayan Bağlantılar

İnternet uygulamalarında istemci ve sunucu, uzun süreli veri alışverişi yapabilir. Bu alışveriş, istemcinin birden fazla istekte bulunup sunucunun her bir isteğe yanıt vermesi şeklinde gerçekleşir. İsteklerin sıklığı uygulamaya bağlı olarak **arka arkaya, düzenli aralıklarla veya düzensiz** şekilde olabilir. Bu durumda, geliştiricilerin iki seçenek arasında karar vermesi gerekir:

Kalıcı Olmayan Bağlantılar (Non-persistent Connections)

Her bir istek-yanıt çifti için **ayrı bir TCP bağlantısı** açılır ve işlem tamamlandıktan sonra bağlantı kapatılır.

Avantajları:

- Basit yapısı nedeniyle kaynakların etkin kullanımı.

Dezavantajları:

- Her bağlantı için yeniden el sıkışma işlemi (TCP handshaking) gerektiği için gecikmeler yaşanır.
- Aşırı bağlantı kurulumu ağ performansını düşürebilir.

Kalıcı Bağlantılar (Persistent Connections)

Tüm istekler ve yanıtlar **aynı TCP bağlantısı** üzerinden gerçekleştirilir. Bağlantı, işlemler arasında açık kalır ve birden fazla istek-yanıt için kullanılır.

Avantajları:

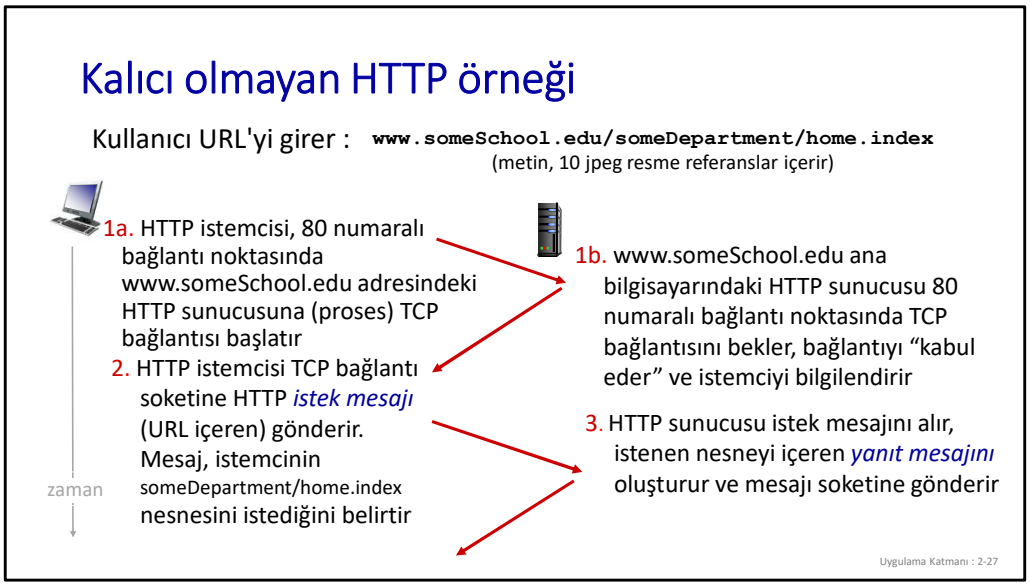
- Daha az gecikme, çünkü aynı bağlantı tekrar kullanılır.
- Daha az bağlantı kurulumu, dolayısıyla daha iyi performans.

Dezavantajları:

- Bağlantının uzun süre açık kalması, sunucunun kaynaklarını tüketebilir.

HTTP'deki Bağlantı Türleri

HTTP, varsayılan olarak **kalıcı bağlantılar** (persistent connections) kullanır. Ancak, sunucular ve tarayıcılar gerektiğinde **kalıcı olmayan bağlantılar** ile de yapılandırılabilir. Bu, uygulamanın performans gereksinimlerine göre esneklik sağlar. Kalıcı bağlantılar özellikle modern web sitelerinde **yüksek performans ve hızlı yanıt süreleri** için tercih edilir. Ancak, bazı durumlarda sunucular, kaynak yönetimi nedeniyle kalıcı olmayan bağlantılar kullanmayı tercih edebilir.



Kalıcı Olmayan Bağlantılarla HTTP'de Web Sayfası Aktarımı

Kalıcı olmayan bağlantılar kullanılarak bir web sayfasının istemciye nasıl aktarıldığını adım adım inceleyelim. Örneğimizde, bir HTML dosyası ve 10 JPEG görseli içeren bir sayfayı aktaracağız. Tüm bu nesnelerin aynı sunucuda barındırıldığını ve temel HTML dosyasının URL'sinin şu olduğunu varsayalım:
http://www.someSchool.edu/someDepartment/home.index.

Adım Adım İşleyiş:

i. TCP Bağlantısının Kurulması:

1. HTTP istemcisi, **80 numaralı port** üzerinden sunucuyla bağlantı kurmak için TCP üç yönlü el sıkışmasını başlatır.
2. Bağlantı kurulduğunda, istemci ve sunucuda **bir soket** oluşturulur.

ii.HTTP İsteğinin Gönderilmesi:

1. İstemci, kurulan soket aracılığıyla bir **HTTP istek mesajı** gönderir. Bu istek, /someDepartment/home.index nesnesine yöneliktir.

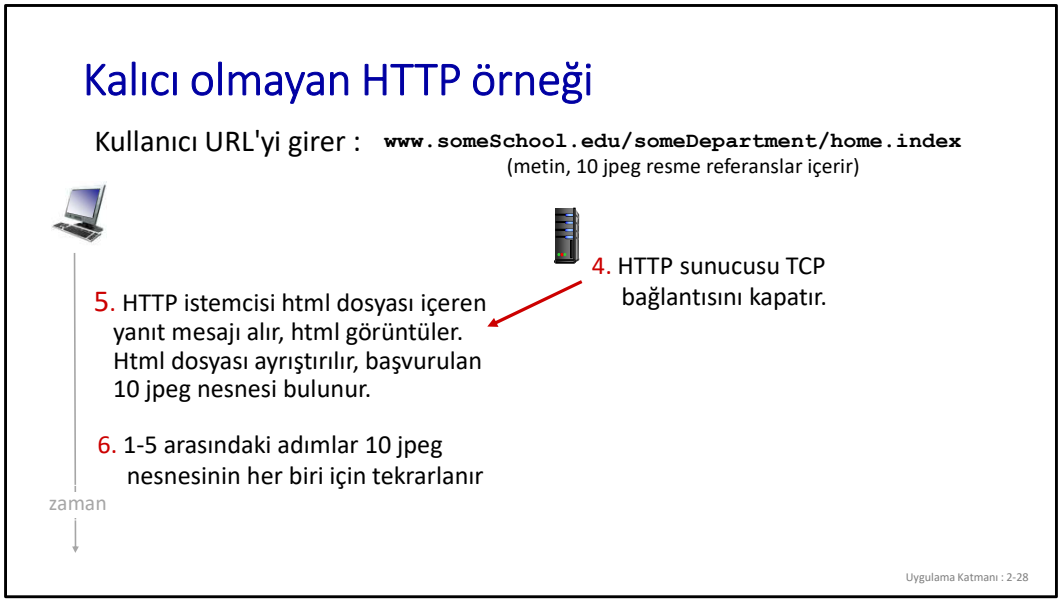
iii.HTTP Yanıtının Sunulması:

1. Sunucu, soket aracılığıyla bu isteği alır ve **/someDepartment/home.index** nesnesini **depolama alanından (RAM veya disk)** çıkarır.
2. HTML dosyası, bir **HTTP yanıt mesajına** eklenerek istemciye geri gönderilir.

Her Nesne İçin Yeni Bağlantı Gerekir

Kalıcı olmayan bağlantılar kullanıldığında, her nesne için **ayrı bir TCP bağlantısı** kurulmalıdır. Örneğin, bu sayfa 10 JPEG görüntüsü içerdiğinden:
•İstemci, her görüntü için yeni bir TCP bağlantısı başlatır.
•Her bağlantı, nesneyi talep etmek ve yanıtı almak için **iki RTT** süresinde tamamlanır.

Sonuç olarak kalıcı olmayan bağlantılar, her nesne için TCP bağlantısı açıp kapatılması gerektiğinden **ağda fazladan gecikmelere** neden olur. Ancak, bazı durumlarda basitliği nedeniyle tercih edilebilir. **Kalıcı bağlantılar** bu sorunu çözerek aynı bağlantı üzerinden birden çok nesne talebine izin verir ve performansı artırır.



Adım iv-vi: Sürecin Devamı

iv. **TCP Bağlantısının Kapatılması:**Sunucu, TCP'ye bağlantıyı kapatmasını söyler. Ancak, TCP, istemcinin yanıt mesajını **eksiksiz ve hatasız aldığını** doğrulamadan bağlantıyı kapatmaz.

v.Yanıtın Alınması ve İşlenmesi:İstemci, yanıt mesajını alır ve TCP bağlantısı sonlandırılır. Yanıtın içeriğinde bir **HTML dosyası** olduğu anlaşılır. Tarayıcı, dosyayı açıp inceledikten sonra **10 JPEG** referansını fark eder.

vi.Her JPEG için İşlemin Tekrarı: Her JPEG nesnesi için **yeniden TCP bağlantısı kurulması** gerekir. Bu, yukarıdaki ilk dört adımın **her görsel için** tekrarlanması anlamına gelir.

Kalıcı Olmayan Bağlantıların Dezavantajları

•**Her bağlantı** yalnızca **bir istek ve bir yanıt** içerir, bu da gereksiz bağlantı kurulumu ve kapanmasına neden olur.

•Örneğimizde, HTML dosyası ve 10 JPEG için toplamda **11 TCP bağlantısı** kurulur.

Paralel Bağlantı Desteği

•Tarayıcılar, **seri bağlantı** yerine **paralel TCP bağlantıları** açarak, aynı anda birden fazla nesne talep edebilir. Bu özellik, web sayfalarının daha hızlı yüklenmesini sağlar.

•Tarayıcılar, paralellik derecesini kontrol etmek için yapılandırılabilir. **Paralel bağlantılar**, özellikle büyük dosyalar veya çok sayıda nesne içeren sayfalarda **yanıt süresini önemli ölçüde azaltır**.

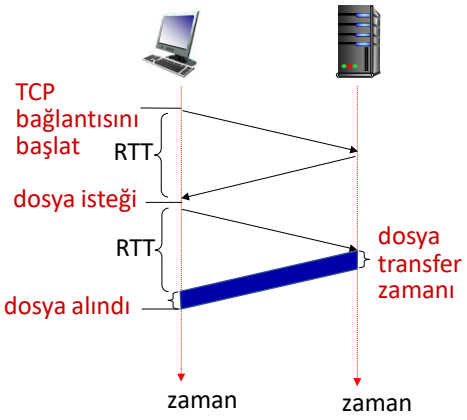
HTTP/1.0 kalıcı olmayan bağlantılar kullanır; her istek için ayrı TCP bağlantısı açılır ve kapatılır. HTTP/1.1 varsayılan olarak kalıcı bağlantıları destekler, böylece aynı bağlantı üzerinden birden çok istek-yanıt işlemi gerçekleştirilebilir. Kalıcı olmayan bağlantılar, fazladan **bağlantı gecikmelerine** neden olur ve verimsizdir. **Paralel bağlantılar** kullanılarak bu süre kısaltılabilir. Ancak, modern web tarayıcıları ve sunucular, bu sorunu çözmek için **kalıcı bağlantılar (persistent connections)** kullanır.

Kalıcı olmayan HTTP: yanıt süresi

RTT: Küçük bir paketin istemciden sunucuya gidip gelmesi için geçen süre

HTTP yanıt süresi (her bir nesne için):

- TCP bağlantısını başlatmak için bir RTT
- HTTP isteği için bir RTT ve HTTP yanıtının ilk birkaç baytının döndürülmesi
- nesne/dosya iletim süresi



Kalıcı olmayan HTTP yanıt süresi = 2RTT+ dosya transfer zamanı

Uygulama Katmanı : 2-29

Basit Yanıt Süresi Hesaplaması: HTML Dosyası Talebi

Bir istemcinin web sunucusundan temel bir **HTML dosyasını talep etmesi** ve dosyanın alınmasına kadar geçen süreyi kabaca tahmin etmek için **RTT (round-trip time)** kavramı kullanılır. RTT, **paketin istemciden sunucuya gitmesi ve geri dönmesi için geçen süreyi** içerir. Bu süre, paketlerin ağ üzerindeki gecikmelerini, ara yönlendiricilerdeki bekletme süresini ve işleme gecikmelerini kapsar.

Yanıt Süresini Etkileyen Aşamalar:

1.TCP Bağlantısı Kurulumu:

1. İstemci ve sunucu arasında TCP bağlantısı kurmak için **üç yönlü el sıkışma (three-way handshake)** gerçekleştirilir.
    1. İstemci, sunucuya küçük bir TCP segmenti gönderir.
    2. Sunucu, bu segmenti onaylar ve yanıt gönderir.
    3. İstemci, sunucuya nihai onayını iletir.
2. Bu **ilk iki adım** yaklaşık **bir RTT** sürer.

2.İstek ve Yanıt Süreci:

1. El sıkışmanın son adımında, **HTTP istek mesajı** sunucuya gönderilir.
2. Sunucu, HTTP isteğini aldıktan sonra **HTML dosyasını TCP bağlantısı üzerinden gönderir**.
3. Bu aşama da **bir RTT** sürer.

Toplam Yanıt Süresi Hesabı:

•**TCP bağlantısı kurulumu**: 1 RTT + **HTTP isteği ve yanıtı**: 1 RTT

•**Toplam yanıt süresi = 2 RTT + HTML dosyasının sunucudaki iletim süresi**.

Bu hesap, ideal koşullar içindir. Ağ tıkanıklığı, yönlendiricilerdeki gecikmeler veya TCP'nin yeniden iletim mekanizmaları gibi faktörler bu süreyi artırabilir.

Kalıcı HTTP (HTTP 1.1)

Kalıcı olmayan HTTP sorunları:

- nesne başına 2 RTT gerektirir
- Her TCP bağlantısı için işletim sistemine ek yükü vardır
- tarayıcılar genellikle referans verilen nesneleri paralel olarak almak için birden fazla paralel TCP bağlantısı açar

Kalıcı HTTP (HTTP1.1):

- sunucu yanıt gönderdikten sonra bağlantıyı açık bırakır
- aynı istemci/sunucu arasında açık bağlantı üzerinden gönderilen sonraki HTTP mesajları
- istemci, başvurulan bir nesneyle karşılaşır karşılaşmaz istek gönderir
- başvurulan tüm nesneler için bir RTT kadar az (yanıt süresini yarıya indirir)

Kalıcı Bağlantılar ile HTTP

Kalıcı olmayan bağlantılar, her nesne isteği için yeni bir TCP bağlantısı gerektirdiğinden bazı dezavantajlara sahiptir. Bu dezavantajlar şunlardır:

1.Yüksek Sunucu Yükü: Her yeni bağlantı, hem istemci hem de sunucu tarafında **TCP tamponlarının** tahsis edilmesini ve **TCP değişkenlerinin** tutulmasını gerektirir. Bir sunucu, aynı anda yüzlerce farklı istemciden gelen isteklere hizmet vermeye çalıştığında bu durum sunucu üzerinde önemli bir yük oluşturabilir.

2.Gecikme Süresi (RTT): Her nesne isteği, **iki RTT'lik gecikmeye** neden olur—bir RTT bağlantı kurulumu için, bir RTT ise nesnenin istenip alınması için harcanır.

HTTP/1.1 ve Kalıcı Bağlantılar

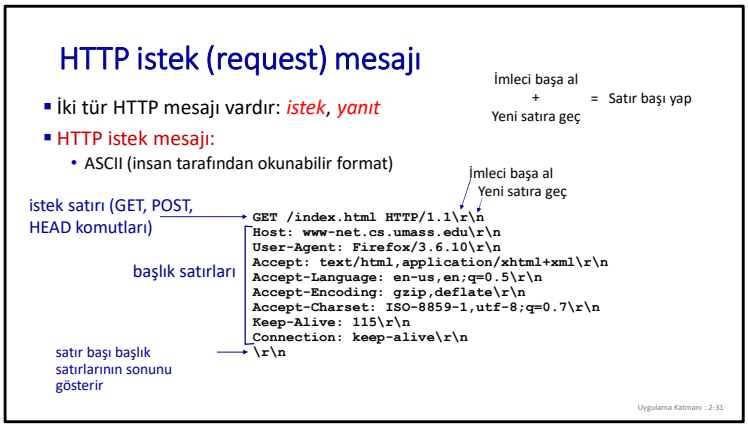
HTTP/1.1 ile birlikte kalıcı bağlantılar varsayılan hale gelmiştir. Bu bağlantılarda, sunucu bir yanıt gönderdikten sonra TCP bağlantısını açık bırakır. Böylece, aynı istemci-sunucu çifti arasında gerçekleşen sonraki istek ve yanıtlar **aynı bağlantı** üzerinden aktarılır.

•**Performans Artışı**: Örneğin, bir web sayfasının tamamı (HTML dosyası ve sayfadaki tüm resimler) **tek bir kalıcı TCP bağlantısı** üzerinden gönderilebilir.

•**Pipelining Desteği**: Kalıcı bağlantılar sayesinde, aynı bağlantı üzerinde **pipelinining** (arka arkaya istek gönderimi) yapılabilir. Bu sayede, bir nesne yanıtı beklenmeden yeni istekler gönderilebilir, bu da performansı artırır.

•**Zaman Aşımı ve Bağlantı Kapatma**: Bir HTTP sunucusu, belirli bir süre (zaman aşımı aralığı) etkinlik olmadığında bağlantıyı otomatik olarak kapatabilir. Ancak sunucuya üst üste gelen istekler, nesnelerin hızlıca arka arkaya gönderilmesini sağlar.

Kalıcı bağlantılar sayesinde, web tarayıcıları ve sunucular daha **verimli** çalışır ve daha az gecikme yaşanır. **HTTP/1.1** bu yaklaşımı benimsemiş olsa da, kalıcı olmayan bağlantılar belirli durumlarda hâlâ kullanılabilir. **HTTP/2** ([RFC 7540]) gibi yeni sürümler de bu kalıcı bağlantı kavramını geliştirmeye devam etmektedir.



HTTP İstek (Request) Mesajı Yapısı

HTTP, istemci ve sunucu arasındaki iletişimi düzenleyen protokol olarak iki tür mesaj içerir: **istek (request) mesajları** ve **yanıt (response) mesajları**.

1. Genel Yapı ve Format

•HTTP istek mesajları, **ASCII formatında** yazılır, bu da onların kolayca okunabilir olmasını sağlar.

•Mesaj birkaç satırdan oluşur ve her satırın sonunda satır başı ve satır sonu karakteri bulunur.

•İlk satır, **istek satırı (request line)** olarak adlandırılır ve **yöntem**, **URL** ve **HTTP sürümü** bilgilerini içerir.

•Sonraki satırlar **başlık (header) satırları** olarak bilinir ve istekle ilgili ek bilgileri sağlar.

2. İstek Satırı (Request Line)

İstek satırında üç alan yer alır:

•**Yöntem**: İstenen işlemi tanımlar. Örneğin: GET, POST, HEAD, PUT, DELETE. Örnekte **GET** yöntemi, tarayıcının /somedir/page.html nesnesini talep ettiğini belirtir.

•**URL**: İstenen kaynağın yolu.

•**HTTP Sürümü**: Kullanılan HTTP sürümünü belirtir. Örnekte, HTTP/1.1 sürümü kullanılmıştır.

3. Başlık Satırları (Header Lines)

Başlık satırları, istemcinin sunucuya gönderdiği ek bilgileri içerir:

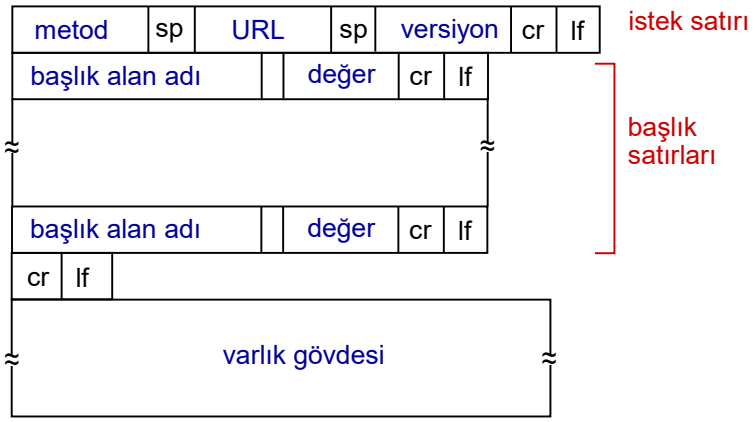
•**Host**: Host: www.someschool.edu satırı, nesnenin bulunduğu sunucuyu belirtir. Web proxy önbellekleri bu bilgiye ihtiyaç duyduğu için her istek mesajında yer alır.

•**Connection**: Connection: close, sunucuya **kalıcı bağlantı** (persistent connection) kullanılmayacağını ve nesne iletildikten sonra bağlantının kapatılmasını belirtir.

•**User-agent**: User-agent: Mozilla/5.0, isteğin hangi tarayıcıdan geldiğini belirtir. Bu bilgi, sunucunun aynı URL'nin farklı sürümlerini farklı tarayıcılara sunabilmesi açısından önemlidir.

•**Accept-language**: Accept-language: fr, istemcinin tercih ettiği dilin Fransızca olduğunu belirtir. Sunucu, uygun sürüm mevcutsa Fransızca içerik gönderir; yoksa varsayılan dili sunar.

HTTP istek mesajının genel formatı



Uygulama Katmanı : 2-32

HTTP İstek Mesajının Yapısı: Varlık Gövdesi (Entity Body)

Şekil bir HTTP istek mesajının daha önce gördüğümüz örnekle benzer bir yapıya sahip olduğunu gösterir, ancak ek olarak bir **"varlık gövdesi" (entity body)** içerebilir. Varlık gövdesi, özellikle **POST yöntemi** kullanıldığında aktif hale gelir.

Varlık Gövdesinin Kullanımı

•**GET yönteminde**: GET isteklerinde varlık gövdesi **boş** kalır, çünkü bu yöntem yalnızca sunucudan nesne talep etmek için kullanılır.

•**POST yönteminde**: POST istekleri sırasında, varlık gövdesi, kullanıcı tarafından form alanlarına girilen verileri içerir. Bu yöntem genellikle bir **form gönderildiğinde** veya bir **arama motoruna arama terimleri girildiğinde** kullanılır.

Örneğin, bir kullanıcı arama motoruna sorgu kelimeleri girdiğinde, POST mesajının varlık gövdesi, bu arama terimlerini içerir. Sunucu, kullanıcının gönderdiği bu verileri kullanarak **kişiselleştirilmiş** bir yanıt döner, örneğin, arama sonuçlarını sunar.

POST ve GET Yöntemlerinin Farkı

•**GET yöntemi**: Parametreler URL'nin parçası olarak gönderilir (örneğin, example.com/search?q=kitaplar), bu da veri boyutunu sınırlar ve verilerin gözlemlenebilir olmasına neden olur.

•**POST yöntemi**: Veriler **varlık gövdesinde** taşınır, bu da daha fazla veri iletimine ve hassas bilgilerin daha güvenli aktarılmasına olanak tanır.

Bu nedenle, **POST yöntemi**, kullanıcı etkileşimli formlarının işlendiği durumlarda tercih edilirken, **GET yöntemi** daha basit veri talepleri için kullanılır.

Diğer HTTP istek metotları

POST metodu:
- web sayfası genellikle form girişi içerir
- HTTP POST istek mesajının varlık gövdesinde istemciden sunucuya gönderilen kullanıcı girdisi

GET metodu: (sunucuya veri göndermek için):
- HTTP GET istek mesajının URL alanına kullanıcı verilerini eklenir ('?' işaretinden sonra):

```
www.somesite.com/animalsearch?monkeys&bananas
```

HEAD metodu:
- belirtilen URL bir HTTP GET yöntemiyle istendiğinde döndürülecek üstbilgileri (yalnızca) ister.

PUT metodu:
- sunucuya yeni dosya (nesne) yükler
- POST HTTP istek iletisinin varlık gövdesindeki içerikle belirtilen URL'de var olan dosyayı tamamen değiştirir

HTTP Yöntemleri: GET, POST, HEAD, PUT ve DELETE

HTML formları, yalnızca **POST** yöntemiyle değil, **GET** yöntemiyle de kullanılabilir. İşte HTTP yöntemlerinin kullanım amaçları ve farkları:

GET Yöntemi

•**GET** yöntemiyle form verileri, **URL'nin parçası** olarak gönderilir.

Örnek: Bir form iki alan içeriyorsa ve kullanıcı "monkeys" ve "bananas" girdilerini eklediyse, URL şu şekilde olur:

•www.somesite.com/animalsearch?monkeys&bananas

•GET yöntemi, verileri URL'ye eklediğinden, **veri boyutu sınırlıdır** ve veriler URL üzerinden görülebilir, bu da güvenlik açısından bazı sınırlamalar getirir.

POST Yöntemi

•**POST**, form verilerini **varlık gövdesi** aracılığıyla gönderir, bu da daha fazla veri taşımasına ve **gizliliğin korunmasına** olanak tanır.

•Özellikle **kullanıcı giriş formları** veya büyük veri gönderimleri için kullanılır.

HEAD Yöntemi

•**HEAD**, GET'e benzer, ancak yalnızca **başlık (header)** bilgilerini döndürür; istenen nesneyi içermez.

•Geliştiriciler tarafından genellikle **hata ayıklama** veya bağlantının mevcut olup olmadığını kontrol etmek için kullanılır.

PUT Yöntemi

•**PUT**, bir nesneyi sunucuda **belirli bir yola yüklemek** için kullanılır.

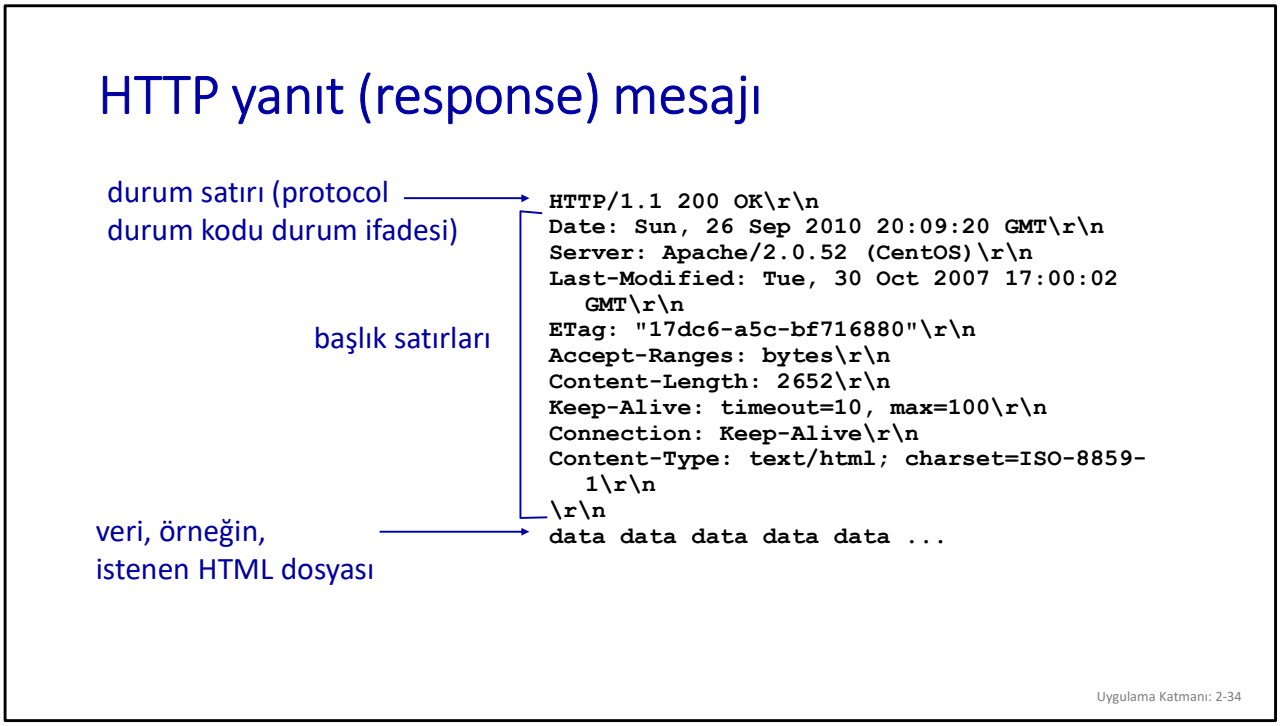
•Bu yöntem, özellikle **web yayınlama araçları** ve uygulamalar tarafından, sunucuya içerik yüklemek için kullanılır.

DELETE Yöntemi

•**DELETE**, sunucudaki belirli bir nesneyi **silmek** için kullanılır.

•Kullanıcı veya uygulamalar, gereksiz hale gelen nesneleri bu yöntemle kaldırabilir.

Her yöntem, farklı kullanım senaryolarına hitap eder. GET, POST gibi yaygın yöntemler dışında PUT ve DELETE gibi yöntemler, özellikle içerik yükleme veya yönetimi gerektiren uygulamalarda tercih edilir.



HTTP Yanıt Mesajı Yapısı

Bir **HTTP yanıt mesajı**, istemciden gelen isteğe karşılık sunucu tarafından gönderilen bilgileri içerir ve üç ana bölümden oluşur:

1.Başlangıç Durum Satırı (Status Line)

1. **Protokol versiyonu**, **durum kodu**, ve **durum mesajından** oluşur.
   Örnek: HTTP/1.1 200 OK
2. Bu örnekte sunucu, **HTTP/1.1** sürümünü kullandığını ve isteğin başarılı olduğunu belirtir.

2.Başlık Satırları (Header Lines)

1. **Connection: close**: Bağlantı kullanıldıktan sonra kapanacak.
2. **Date**: Yanıtın sunucu tarafından oluşturulduğu tarih ve saat.
   Örnek: Date: Tue, 18 Aug 2015 15:44:04 GMT
3. **Server**: Yanıtın hangi web sunucusu tarafından üretildiğini gösterir (örneğin: Apache/2.2.3 (CentOS)).
4. **Last-Modified**: İstenen nesnenin en son değiştirildiği tarih ve saat.
5. **Content-Length**: Yanıt gövdesindeki veri boyutunu bayt olarak gösterir.
6. **Content-Type**: Gönderilen verinin türünü belirtir (ör. text/html).

3.Varlık Gövdesi (Entity Body)

İstenen nesnenin kendisini içerir (örneğin, HTML verisi veya başka içerik).

HTTP yanıt durum kodları

- Durum kodu sunucudan istemciye yanıt mesajında 1. satırda görünür.
- Bazı örnek kodlar:

200 OK
- istek başarılı oldu, istenen nesne bu mesajın ilerleyen bölümlerinde

301 Moved Permanently
- istenen nesne kalıcı olarak başka bir yere taşındı

400 Bad Request
- istek mesajı sunucu tarafından anlaşılmadı

404 Not Found
- istenen belge bu sunucuda bulunamadı

505 HTTP Version Not Supported
- Sunucu istenen HTTP sürümünü desteklemiyor.

Yaygın HTTP Durum Kodları
•**200 OK**: İstek başarılı ve yanıt içerikle birlikte döndü.
•**301 Moved Permanently**: Nesne kalıcı olarak taşındı, nesneye ait yeni URL «**Location:»** başlığında belirtilir.
•**400 Bad Request**: İstek sunucu tarafından anlaşılamadı.
•**404 Not Found**: İstenen nesne bulunamadı.
•**505 HTTP Version Not Supported**: Sunucu istenen HTTP sürümünü desteklemiyor.