2.1 Principles of Network Applications (Ağ Uygulamalarının Temelleri)

2.1 Principles of Network Applications

2.1.1 Network Application Architectures

2.1.2 Processes Communicating

2.1.3 Transport Services Available to Applications

2.1.4 Transport Services Provided by the Internet

2.1 Principles of Network Applications

Network application geliştirmenin merkezinde farklı *end system'larda çalışan ve ağ üzeriden birbirleriyle iletişim kuran programlar yazmak bulunmaktadır.

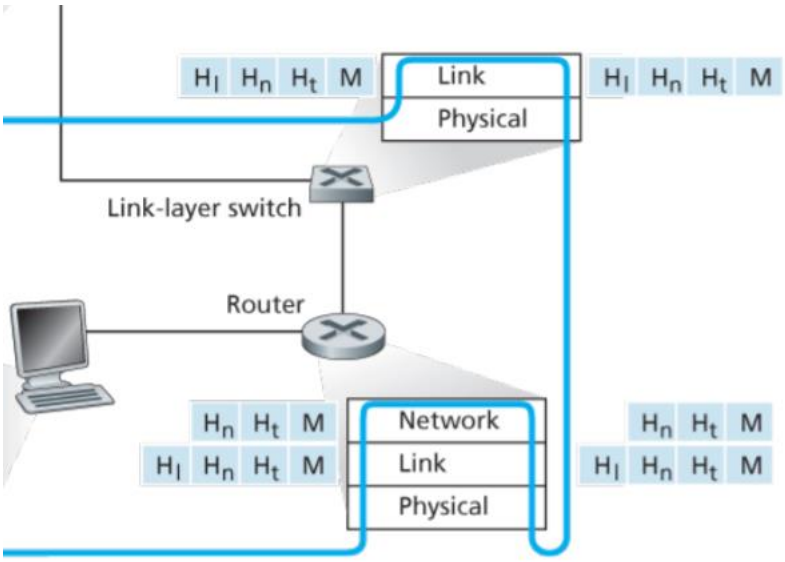
Örneğin : Web uygulamasında birbirleriyle iletişim kuran iki ayrı program vardır. Kullanıcının bilgisayarındaki çalışan browser ve Web sunucusu olan bilgisayardaki Server Host.

Başka bir örnek olarak bir P2P paylaşım sisteminde dosya paylaşımına katılan her bilgisayarda bir program bulunmaktadır.Bu durumda çeşitli bilgisayrdaki programlar benzer ya da özdeş olabilir.

Bu nedenle yeni bir uygulama geliştiriken birden fazla end system de çalışacak bir yazılım yazmamız gerekir.Bu yazılım herhangi bir dilde yazılmış olabilir.(c,java,python vs.)

Daha da önemlisi routers(yönlendirici) veya link layer(bağlantı katmanı) gibi network-core(ağ çekirdekli) aygıtlarda çalışan bir yazılım yazmamız gerekmez.Bunu yazmak istesek bile yapamayız.

Ağ çekirdekli(network-core) cihazlar application katmanında işlevsizdir. Bunun yerine alt katmanlarda Özellikle de network katmanında ya da daha altta çalışır.



Şekil 2.1 deki gibi temel tasarım yapmak çok sayıda network application'un hızlı bir şekilde geliştirilmesini ve yayılmasını,konuşlandırılmasını kolaylaştırmıştır.

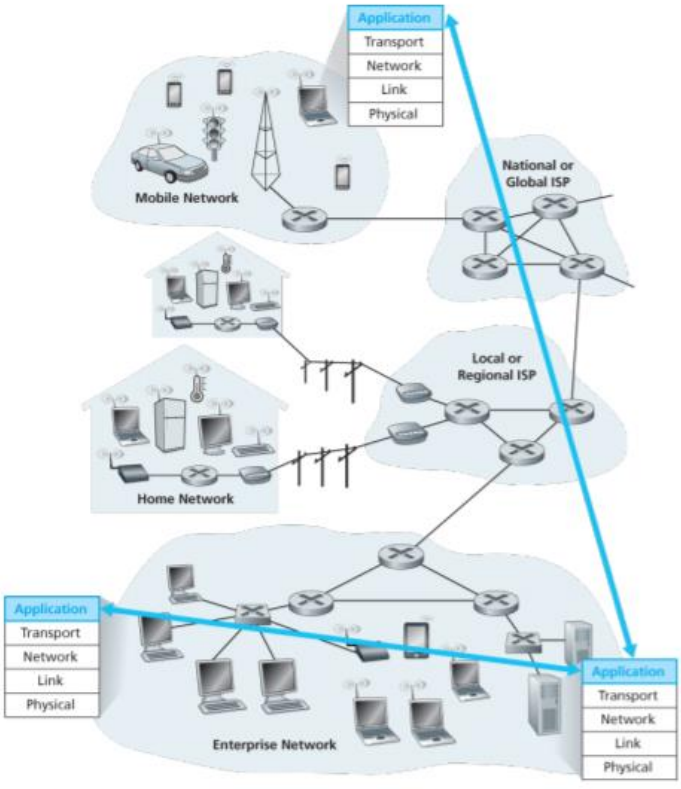


Figure 2.1 Communication for a network application takes place between end systems at the application layer

*End system : Ağ oluşturma jargonunda, bir bilgisayar ağına bağlı bilgisayarlara bazen son sistemler veya son istasyonlar denir. Ağın kenarında oturdukları için uç sistemler olarak etiketlenirler. Son kullanıcı her zaman son sistemlerle etkileşime girer. Son sistemler bilgi veya hizmet sağlayan cihazlardır.

2.1.1 Network Application Architectures (Ağ Uygulaması mimarisi)

Yazılım kodlamaya başlamadan önce uygulama için bir planımız olmalıdır.

Uygulama mimarisi 5 katmanlı Internet mimarisinden farklıdır.

Ağ mimarisi sabittir ve uygulamalara belirli bir set of services(hizmet kümesi) sağlar.

Uygulama mimarisinde ise uygulama geliştiricisi tarafından tasarlanmıştır ve uygulamanın çeşitli end system'lere göre nasıl yapılandırıldığını belirler.

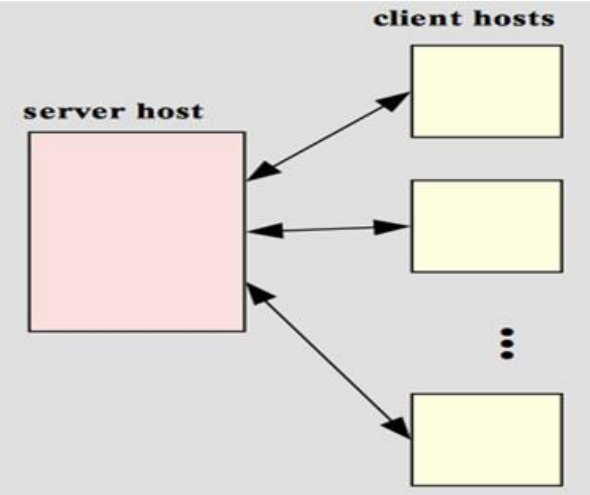
Uygulama mimarisi seçerken geliştirici 2 mimari paradigmadan birini seçer bunlar;

1) The Client-Server architecture
2) Peer-to-Peer (P2P) architecture

Bir Client-Server mimarisinde server(sunucu) ve clients(istemci) adı verilen bir çok bilgisayardan gelen isteklere hizmet veren ve her zaman açık bir ana bilgisayar vardır.

1) The Client-Server architecture

İstemci - sunucu mimarisi, ağ içindeki her bilgisayar ya da işlemin (process); ya istemci (client) ya da sunucu (server) olduğu mimaridir.



İstemciler (clients)

İstemciler, bilgisayarlar üzerinde çalıştırılan uygulamalardır.
İstemciler, sunuculardan aşağıdaki konularda istemde bulunabilirler:

- Dosyalar
- Cihazlar
- İşlemci gücü

Örneğin, en çok kullanılan istemcilerden biri e-posta istemcisidir. Size elektronik mektup gönderme ve alma olanağı sağlar.

Sunucular(server)

Sunucular, ağ (network) kaynaklarını yöneten bilgisayarlardır.
Fonksiyonlarına göre aşağıdaki isimleri alırlar:

- Dosya sunucuları
- Yazıcı sunucuları
- Ağ sunucuları

Örnek: Veritabanı sunucusu (database server): Veritabanı sorgularını işleyen bir bilgisayar sistemidir.

Haberleşme ağları istemci ve sunucuları birbirine bağlayan iletişim devreleridir.

Klasik bir örnek olarak her zaman açık olan bir web server hizmetlerinin client bilgisayarlarda çalışan tarayıcılardan istediği Web uygulamasıdır.
Web server , client host'tan bir nesne için istek aldığında istenen nesneyi client bilgisayara göndererek cevap verir.

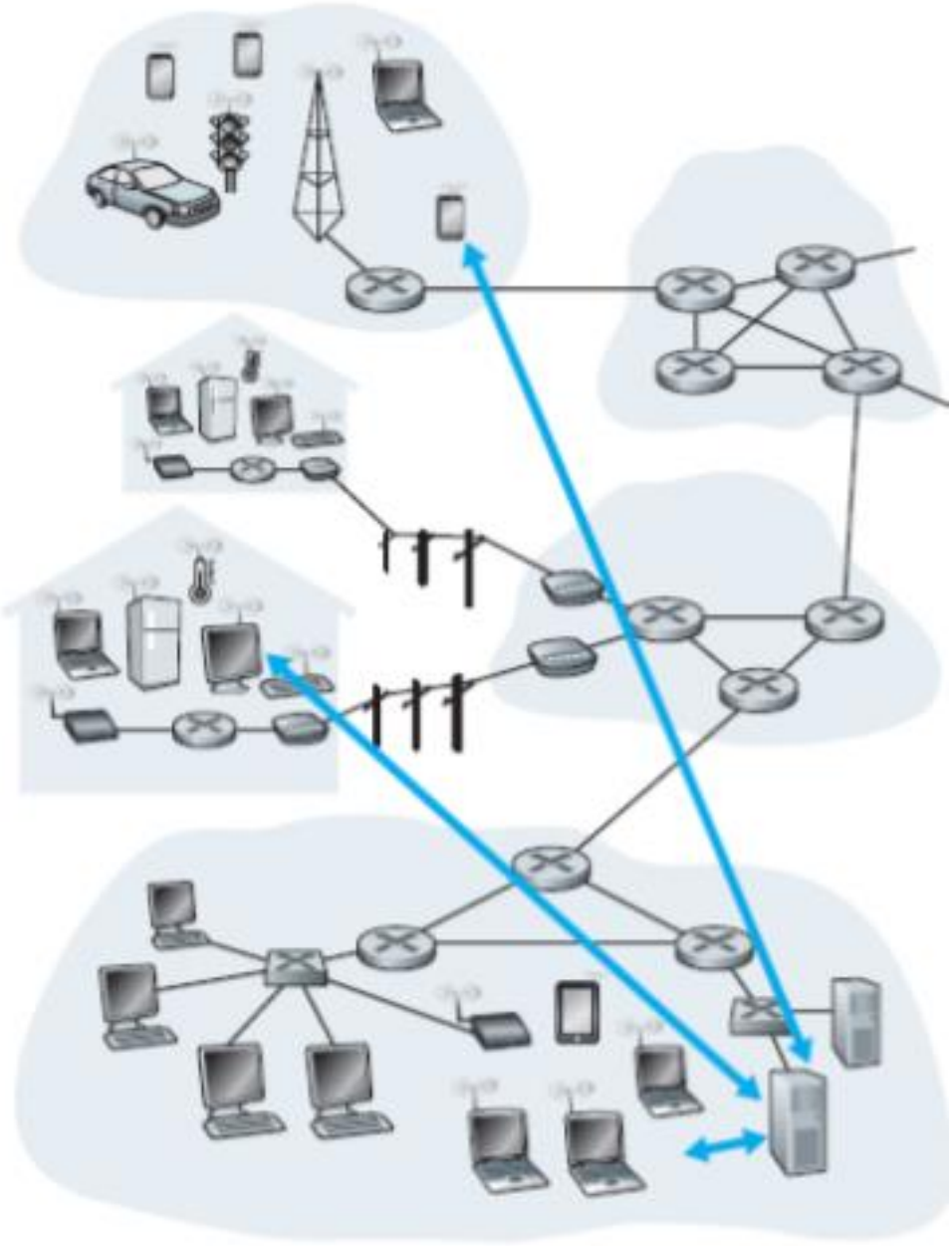
Client-server mimarisinde clientler birbirleriyle doğrudan iletişim kurmazlar unutma.

Örnek olarak bir web uygulamasında iki tarayıcı doğrudan iletişim kurmaz.

Client-Server mimarisinin bir diğer özelliği de sunucunun IP adresi fixed (sabit) ve well-known olmalıdır.

Sunucu fixed and well known bir adrese sahip olduğundan ve her zman açık olduğundan , client her zman sunucunun IP adresine bir paket göndererek sunucuyla contact kurabilir.

Bazı Client-Server uygulamaları ; Web , FTP , Telnet , E –mail..



a. Client-server architecture

Genellikle bir Client-Server uygulamasında tek bir server clientlerden gelen tüm talepleri karşılayamaz.

Örneğin popüler bir sosyal ağ sitesi tüm isteklerini yerine getiren bir server'a sahipse hızlı bir şekilde overwhelming(bunalma) olur.

Bu nedenle genellikle güçlü bir sanal server oluşturmak için çok sayıda host'u barındıran bir data centre(veri merkezi) kullanılır.

- Google-search engines
- Amazon-internet commerce
- Gmail-web-based e mail
- Facebook- social networking

gibi örneklerde hep birden daha fazla data centre kullanılır. Örnek google 30-50 arası data center vardır.

Bir data center'da yüzbinlerce server bulunabilir ve bunlar güçlü ve korunaklı olmalıdır.

Buna ek olarak servis sağlayıcıları data centre 'lardan veri göndermek için recurring interconnection(tekrar eden ara bağlantı) and bandwidth(bantgenişliği) maliyetlerini karşılaması gerekmektedir.

2) Peer-to-Peer (P2P) architecture

Peer-to-peer (P2P) mimarisinde, data center'lardaki özel serverlara çok az (ya da hiç) güven yoktur.

Bunun yerine uygulama, peers(eşler) olarak adlandırılan aralıklı olarak , bağlı host çiftleri arasındaki doğrudan iletişimi kullanır.

Peer'lar service provider (servis sağlayıcı) a ait değildir bunu yerine kullanıcılar tarafından konrol edilen masaüstü bilgisayarlar vs. dır.

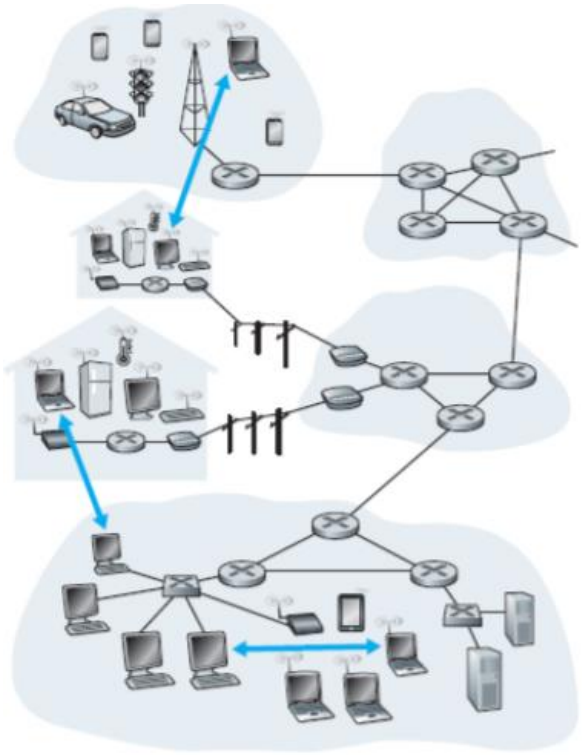
Peer'ların çoğu evlerde,üniversitelerde,ofislerde bulunur.

Peer'lar özel bir server'dan geçmeden iletişim kurdukları için mimariye peer-to-peer denir.

Günümüzde en popüler yoğun trafikli uygulamaların çoğu P2P mimarisine dayanmaktadır.

- BirTorrent – File sharing
- Xunlei- Peer-assisted dowland acceleration(eş destekli indirme hzılandırma)
- Skype- Video conference

gibi P2P mimarisini kullanan örnekler vardır.



b. Peer-to-peer architecture

Bazı uygulamalar hem client-server hem de P2P mimarilerini birleştiren hibrit mimarilere sahiptir.

Örneğin bir çok anlık mesajlaşma uygulaması için server'lar kullanıcıların IP adreslerni izler ancak kullanıcıdan kullanıcıya mesajlar doğrudan user hosts'lar arasında gönderilir(ara serverlardan geçmeden).

P2P mimarisinin en ilgi çekici özelliği self-scalability(kendi kendini ölçeklendirebilme) dir.

Örneğin bir P2P dosya paylaşım uygulamasında her peer dosya isteyerek iş yükü oluşursa da her eş aynı zamanda dosyaları diğer peer'lara dağıtarak da sisteme service capacity (hizmet kapasitesi) ekler.

P2P mimarileri de maliyetlidir. Normalde data centerlara sahip client-server tasarımlarının aksine önemli server altyapısı ve server bantwidth gerekirmezler.Ancak P2P uygulamaları merkezi olmayan yapıları (decentralized structure) nedeniyle güvenlik,performans,güvenilirlik zorluklarıyla karşı karşıyadır.

2.1.2 Processes Communicating (İletişim işlemleri)

Network application oluşturmadan önce birden fazla end systemde çalışan programların birbirleriyle nasıl iletişim kurduklarını anlamamız gerekir.

İşletim sistemleri jargonunda aslında programlar değil process'ler iletişim kurarlar.

Process, bir end system içinde çalışan bir program olarak düşünülebilir.

Process'ler aynı end system'da çalışırken end sytem'in işletim sistemi tarafından yönetilen kuralları kullanarak interprocess communication (processler arası iletişim) ile birbirleriyle iletişim kurabilirler.

Ancak biz aynı host daki işlemlerin nasıl iletişim kurduğuyla değil farklı hostlarda çalışan işlemlerin nasıl iletişim kurduklarıyla ilgileniyoruz.

İki farklı end system'daki işlemler computer network üzerinden mesaj alışverişi yaparak birbirleriyle iletişim kurarlar.

Bir gönderme işlemi mesaj oluşturur ve ağa gönderir. Bir mesaj alma işlemi bu mesajları alır ve geri göndererek yanıt verir. Figure 2.1

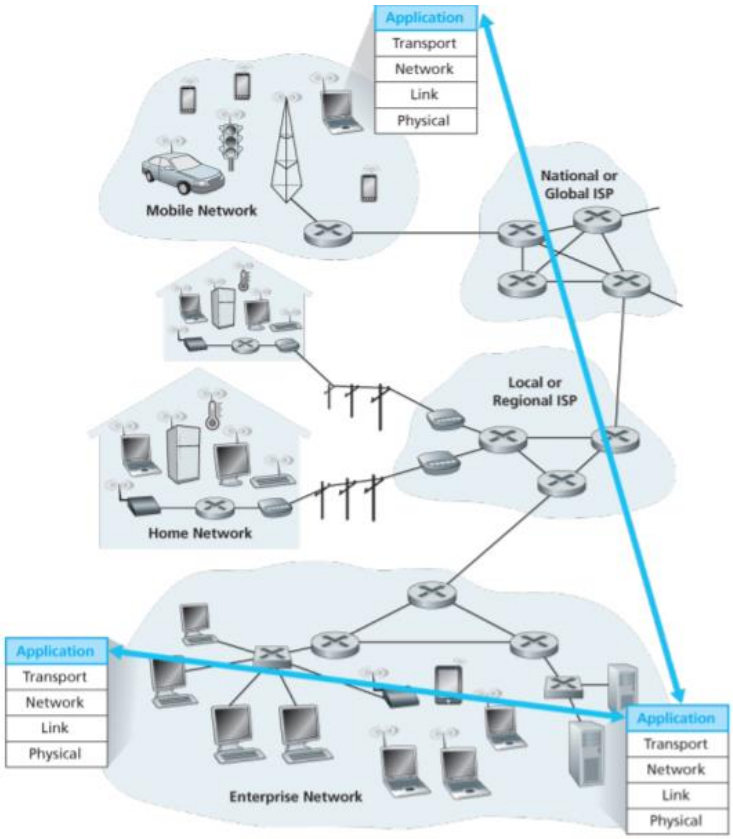


Figure 2.1 Communication for a network application takes place between end systems at the application layer

Birbirleriyle iletişim kuran 5 katmanlı protocol yığınının uygulama katmanında yer aldığını gösterir.

Client and Server Processes

Bir Network application bir ağ üzerinden birbirlerine mesaj gönderen işlem çiftlerinden oluşur.

Örneğin, web uygulamasında bir client tarayıcı işlemi bir web server işlemiyle mesaj alışverişi yapar.

Bir P2P dosya paylaşım sisteminde , dosya bir peer'deki işlemden başka bir peer deli işleme aktarılır.Her bir iletişim process'i için genellikle iki işlemden birini client ,diğer işlemi server olarak etiketleriz,sınıflandırırız.

Web tarayıcı(browser) bir client işlemidir ve web server bir server işlemidir.P2P dosya paylaşımı ile dosyayı indiren eş ,client olarak etiketlenir ve dosyayı yükleyen eş , server olarak etiketlenir.

Client işlemi : iletişimi başlatan process (işlem) Server işlemi: bağlantıyı bekleyen (iletişimi bekleyen) process (işlem)

P2P mimarisinde client ve server işlemleri vardır. P2P soya paylaşım sistemindel, bir process hem dosyaları yükleyebilir hem de indirebilir.

Bununla birlikte bu çift process arasındaki herhangi bir iletişimi biri process'i client diğerini server olarak etiketleyebiliriz.

Web'de, tarayıcı işlemi bir Web server işlemiyle başlatır bu nedenle tarayıcı işlemi Client ve web server işlemi server'dir.

P2P dosya paylaşımında PeerA ,PeerB den bir dosya göndermesini istediğinde PeerA client ve PeerB bu özel iletişimde Server(sunucu)dur.

Karışıklık olmadığında bazen "uygulamanın client tarafı ve server tarafı" terminolojisini de kullanırız.

The Interface Between the Process and the Computer Network (Bilgisayar ağı ve Process'ler arasındaki arayüz) ----SOKET-----

Çoğu uygulama işlem çiftlerinden oluşur ve her çiftteki iki işlem birbirine mesaj gönderir.Bir işlemden diğerine gönderilen tüm mesajlar temel ağdan geçmelidir.

Process Soket adı verilen bir yazılım arabirimi aracılığı ile ağa mesaj gönderir veya ağdan mesaj alır.

Proces'leri ve Soketleri anlamak için şu örneğe bakalım.

Bir process bir eve benzer ve Soket evin kapısına benzer.Bir process başka bir host'taki başka bir process'e mesaj göndermek istediğinde mesajı kapısından(soketten) çıkarır. Bu gönderme process'i kapısının diğer tarafında mesajı hedef process'in kapısına taşıyacak bir ulaşım altyapısı olduğunu varsayar.Mesaj hedef host'a ulaştığında mesaj alma process'in kapısından(soketinden) geçer ve alma process'i gerçekleşir.

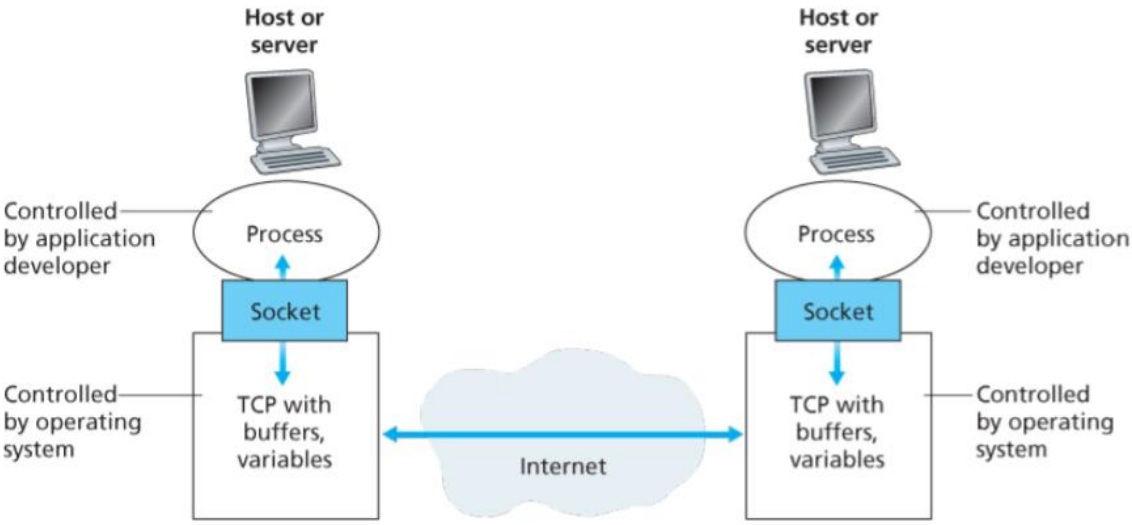


Figure 2.3 Application processes, sockets, and underlying transport protocol

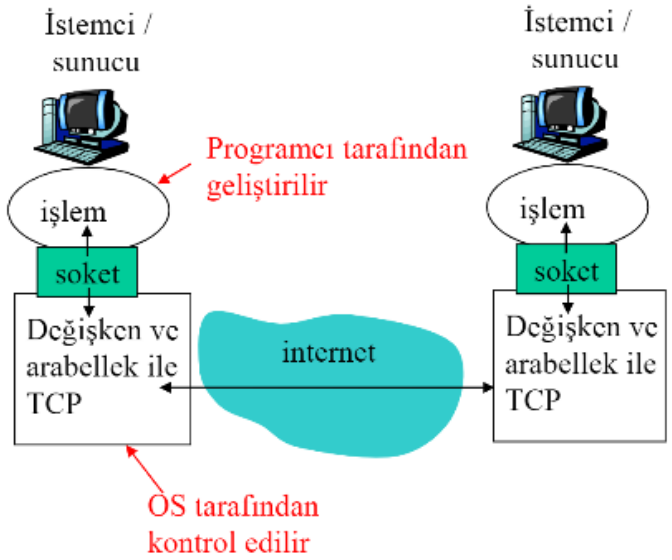


Figure 2.3 internet üzerinden iletişim kuran iki process arasındaki, soket process'i göstermektedir.Bu process tarafından kullanılan temel aktarım protokolünün Internet'in TCP protokolü olduğunu varsayar.

Bu şekilde gösterildiği gibi , bir soket application layer(uygulama katmanı) ile host içindeki transport layer (aktarım katmanı) arasındaki arabirimdir.

Soket ağ uygulamalarının oluşturduğu programlama arabirimi olduğundan Uygulama ile ağ arasındaki ApplicationProframmingInterface(API) olarak adlandırılır.

Uygulama geliştiricisi soketin (application layer) uygulama katmanı tarafındaki her şeyin kontrolüne sahiptir, ancak soketin Transport layer (taşıma katmanı) tarafı üzerinde çok az kontrole sahiptir.

Uygulama geliştiricisinin (Transport-Layer) taşıma katmanı tarafında sahip olduğu kontroller;

(1) Transport Protocol seçimi

(2) Maksimum tampon(max. Buffer) ve maksimum segment boyutları gibi birkaç Transport-layer parametresini düzeltme yeteneğidir.

Addressing Processes (İşlemleri adreslemek)

Belirli bir hedefe posta göndermek için hedefin vir adresinin olması gerekir.

Benzer şekilde bir host'ta çalışan bir process'in başka bir host'ta çalışan bir process'e paket gönderebilmesi için alıcı process'in de bir adresinin olması gerekir.

Alma (receiving) process'i için bilmemiz gereken 2 bilgi vardır.

1) Host'un adresi
2) Hedef hostta mesaj alma process'ini belirten bir tanımlayıcı (idetifier)

Internette Host IP adresi ile tanımlanır.

Host'ların 32-bitlik unique IP adresi vardır.

Bir mesajın hedeflendiği host'un adresini bilmesnin yanısıra gönderme process'inin host'ta çalışan alma process'ini tanımlaması gerekir.

Bu bilgi gereklidir çünkü genel olarak bir host bir çok ağ uygulaması çalıştırıyor olabilir.

Destination port number bu amaca hizmet eder.

Örneğin bir Web server port number 80 ile tanımlanır.

Bir mail server process'i (STMP protokolü kullanılarak) port number 25 tir.

Tüm listeye www.iana.com dan ulaşabiliriz.

Komut satırından ipconfig komutunu kullanarak IP adresini öğrenebilirsin(windows).

2.1.3 Transport Services Available to Applications(Taşıma hizmeti uygulamanın ihtiyacını karşılar)

Soketler application process ile transport-layer process arasındaki arabirimdir.Gönderen taraftaki uygulama mesajı soketten iter diğer taraftaki transport-layer protocol ise bu mesajları alır.

Birçok network birden fazla transport-layer protokolü sağlar.

Uygulama geliştirmek istediğin zaman mevcut transport-layer protokollerinden hangisini seçmelisin?

Bu seçimi nasıl yapmalısın?

Büyük olasılıkla mevcut transport layer protokollerinin hizmetlerin inceleyecek ve ardından uygulamanızın ihtiyacına uygun hizmetleri içeren protokolü seçeceksiniz.

Bu durum iki şehir arasında seyehat için tren ya da uçak ulaşımını seçmeye benzer.

Birini seçmelisiniz ve her ulaşım türü farklı hzimetler sunuyor.(Örneğin, trenşehir merkezinde karşılama ve bırakma imkanı sunarken uçak daha kısa seyahat süresi sunuyor.)

Transport-layer protokülünün uygulamalara sunabileceği hizmetler nelerdir?

Olası hizmetleri 4 boyutta sınıflandırabiliriz.

- Reliable data transfer or data integrity (Güvenilir veri aktrımı)
- Throughput (Verim)
- Timing (Zamanlama)
- Security (Güvenlik)

Reliable data transfer

Bildiğimiz gibi paketler bilgisayar network'ünün içinde kaybolabilir.

Örneğin bir paket yönlendirinin(router) bir arabelleğinde taşabilir (overflow) veya bazı bitleri bozulduktan sonrda bir host veya router tarafından atılabilir.

E posta , dosya aktarımı, web belgesi , finansal uygulamalar gibi bir çok uygulamam için veri kaybının yıkıcı sonuçları olabiilir.

Bu nedenle bu uyhulamaları desteklemek için uygulamanın bir ucu tarafından gönderilen verilerin uygulamanın diğer ucuna doğru tamamen teslim edilmesini sağlamak için birşeyler yapılmalıdır.

Bir protocol böyle bir garantili veri aktarım hizmeti sağlyorsa, güvenilir veri aktarımı sağladığı söylenir.

Bir tranport-layer protokolünün bir uygulamaya sağlayabileceği önemli hizmetlerden birisi processler arası güvenilir veri aktarımıdır.

Bir transport protokolü hizmeti sağlandığında gönderme process'i verilerini sokete aktarabilir ve verilerin receiving(alma) processinde hatasız olarak geleceğini bilir.

Bir transport-layer protokolü güvenli veri aktarımı sağlamadığında sending process tarafından gönderilen verilerin bir kısımı asla receiving process'e ulaşmayabilir.

Loss-tolerant uygulamalar özellikle video/ses gibi bir miktar veri kaybını tolere edebilen ugulamaları için kabul edilebilir.

Bu multimedya uygulamalarında kaybolan veriler önemli bir bozulma değil sadece ses/video da küçük bir aksaklığa neden olabilir.

Throughput (Verim)

Oturumlar(sessions) bant genişliğini paylaşacak ve bu sessions(oturumlar) gelip gideceğinden verim zamanla dalgalanabilir.

Bir transport-layer protokolü belirli bir oranda garantili kullanılabilir verim sağlayabilir.

Böyle bir hizmetle uygulama r bit/sec'lil garantili bir verim talep edebilir ve transport protokolü mevcut verimin daima en az r bit/sec olmasını sağlar.Böyle bir garantili throughput service bir çok uygulamaya hitap eder.

Örneğin bir internet telefonu uygulamsı sesi 32kbps hızında kodlarsa network'e veri göndermesi ve alıcı uygulamaya(receiving app.)bu hızda veri iletmesi gerekir.

Transport-layer protokolü bu verimi sağlayamıyorsa uygulamanın daha düşük bir hızda kodanması gerekir ya da gerekli verimin yarısının almaktan vazgeçmesi gerekebilir bu da Internet telefonu uygulaması için çok az ya da kullanışsızdır.

Verim gereksinimleri olan uygulamaların bant genişliğine duyarlı uygulamalar olduğu söylenir Mevcut uygulamaların çoğu bantgenişliğine duyarlıdır ancak bazı uygulamalar sijitalleştirilmiş ses veya videoyu suanda mevcut olan verim ile eş bir hızda

kodlamak(encode) için uyarlanabilir kodlama teknikleri(adaptive coding techniques) kullanır.

Bant genişliğine duyarlı uygulamaların özel Throughput requirement ları olsa da, elastik uygulamalar(elastic app.) mümkün olduğu kadar çok veya az verimden yararlanabilir. E-posta, dosya aktarımı ve Web aktarımlarının tümü elastik uygulamalardır.

Tabii ki daha fazla verim daha iyi.

Bir atasözü vardır.

cannot be too rich, too thin, or have too much throughput!

Timing (Zamanlama)

Bir transport-layer protokolü de timing (zamanlama) garantileri sağlayabilir.Throughput garantilerinde olduğu gibi zamanlama garantileri de bir çok şekilde olabilir.

Örnek olarak sender(gönderici) in sokete pompaladığı her bitin daha sonra 100ms olmayacak şekilde alıcı(receiving) nın soketine ulaşması olabilir.

Veri teslimatında sıkı zamanlama kısıtlamaları gerektiren Internet telefonu,sanal ortamlar,telekonferanslar, çok oyunculu oyunlar gibi etkleşimli gerçek zamanlı uygulamalara bu hizmetin etkili olması çekici gelecektir.

Örneğin internet telefon konuşamlarında doğal olmayan duraksamalar olabilir ve buna çok takılmayız.

Çok oyunculu bir oyunda veya sanal etkileşimli bir ortamda eylem sonucunda yanıtı alma uzun gecikmelere sebep olursa bu onu daha az gerçekçi yapar.

Gerçek zamanlı olmayan uygulamalar için düşük gecikme her zaman yüksek gecikmeye tercih edilir ancak end-to-end gecikmelere sıkı bir kısıt getirilmez.

Uçtan Uca Gecikme (End to End Delay)

Uçtan uca gecikme, kaynak ile hedef arasındaki yol üzerinde bulunan router sayısına bağlıdır. Ağda tıkanıklık olmadığı düşünüldüğünde (kuyruk gecikmesi ihmal edildiğinde) aşağıdaki gibi ifade edilir. Burada N yol üzerindeki router sayısını göstermektedir. Katmanlarda oluşan gecikmeler (modülasyon, kodlama, paket oluşturma süresi) uçtan uca gecikmeyi önemli oranda artırabilir.

Security (Güvenlik)

Son olarak bir transport layer protokolü bir veya daha fazla güvenlik hizmetine sahip bir uygulama sağlayabilir.

Örneğin sending host'te bir transport layer sending process(gönderme işlemi) tarafından iletilen tüm verileri şifreleyebilir ve receiving host'ta transport layer protokolü veri alma işlemine(receiving process) telim etmeden önce verilerin şifresini çözebilir.

Böyle bir hizmet sending ve receiving procesleri arasında bir şekilde veriler gözükse bile iki process arasında bir gizlilik sağlayacaktır.

Bir transport protocolü veri bütünlüğü(data integrity), uç nokta kmlik doğrulaması(end point authentication) ve Bölüm 8 de işlenecek konular dahil olmak üzere gizliliğin yanı sıra başka güvenlik hizmetleri de sağlayabilir.

Transport service requirements: common apps

application	data loss	throughput	time sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	
interactive games	loss-tolerant	few kbps up	yes, few secs
text messaging	no loss	elastic	yes, 100's msec yes and no

2.1.4 Transport Services Provided by the Internet

Bu noktaya kadar bir bilgisayr networkünün sağlayabileceği transport hizmetlerini düşündük.Şimdi daha spesifik olalım ve internet tarafından sağlanan transport hizmetlerinin türünü inceleyelim.

Internet(genel olarak TCP/IP networks) uygulamalar için iki tür aktarım protokolü sağlar.

1)UDP (User Datagram Protocol)

2)TCP (Transmission Control Protocol)

Bir uygulama geliştiricisi olarak yeni bir network uygulaması oluşturduğunuzda İnternet için vermeniz gereken ilk kararlardan biri UDP mi yoksa TCP mi kullanacağınızdır.

Her biri bu protokolleri çağıran uygulamalara farklı hizmetler sunar.

Aşağıdaki tablo bazı uygulamalar için servis gereksinimlerini gösterir

Application	Data Loss	Throughput	Time-Sensitive
File transfer/download	No loss	Elastic	No
E-mail	No loss	Elastic	No
Web documents	No loss	Elastic (few kbps)	No
Internet telephony/ Video conferencing	Loss-tolerant	Audio: few kbps–1Mbps Video: 10 kbps–5 Mbps	Yes: 100s of msec
Streaming stored audio/video	Loss-tolerant	Same as above	Yes: few seconds
Interactive games	Loss-tolerant	Few kbps–10 kbps	Yes: 100s of msec
Smartphone messaging	No loss	Elastic	Yes and no

Figure 2.4 Requirements of selected network applications

TCP (Transmission Control Protocol) Services

TCP hizmet modeli, bağlantıya yönelik bir hizmet(connection-oriented service) ve güvenilir bir veri aktarım hizmeti(reliable data transfer) içerir. Bir uygulama TCP transport protokolünü çağırdığında uygulama bu hizmetlerin ikisini de TCP'den alır.

- **Connection-Oriented service(Bağlantı-odaklı hizmet)**
  -TCP, uygulama düzeyindeki mesajlar akmaya başkamadan önce client(istemci) ve sever(sunucu) transport layer denetim bilgilerini birbiriyle değiştirir.
  -Bu handshaking(el sıkışma) prosedürü client ve server' ı uyarur ve paketlerin saldırıya hazırlanmasına izin verir.
  -Handshaking'den sonra iki process soketleri arasında bir TCP bağlantısı olur.Bağlantı iki processin bağlantı üzerinden aynı anda birbirlerine mesaj gönderebilmesi için tam çift yönlü(full-dublex) bir bağlantıdır.
  Uygulama mesaj göndereyi bitirdiğinde bağlantıyı koparmalıdır.
- **Reliable data transfer service (Güvenilir veri aktarım hizmeti)**
  -İletişim processleri gönderilen tüm verileri hatasız ve doğru sırada sunmak için TCP ye güvenebilir.Uygulamanın bir tarafı bir byte akışını sokete geçirdiğinde aynı byte akışını alıcı sokete eksik veya yinelenen byte olmadan iletmek için TCP ye güvenebilir.

TCP aynı zamanda iletişim processlerinin doğrudan yararından ziyade internetin genel refahı için tıkanıklığı kontrol mekanizması içerir.Ağ ,sender ve receiver arasında tıkandığında TCP tıkanıklığı kontrol mekanizması bir sender process'ini(client ya da server) kısıtlar.

UDP (User Datagram Protocol) Services

UDP asgari düzeyde hizmet sunan , kolay ve hafif (lightweight) bit transport protokolüdür.UDP bağlantısızdır(connectionless),bu nedenle iki process iletişim kurmaya bağlamadan önce handshaking olmaz.UDP güvenilir olmayan bir veri aktarım hizmeti sağlar;yani bir

işlem bir UDP soketine mesaj gönderdiğinde UDP mesajın hiçbir zaman alma processine ulaşacağını garanti etmez.Ayrıca alma(receiving) processine gelen mesajlar düzensiz gelebilir.

UDP bir tıkanıklık kontrol mekanizması içermez bu nedenle UDP nin gönderen tarafı verileri istediği hızda aşağıdaki katmana(network-layer) a pompalayabilir.

Bununla birlikte araya giren bağlantının sınırlı iletim kapasitesi veya tıkanıklık nedeniyle gerçek end-to-end verimin bu hızdan daha düşük olabileceğini unutmayın.

FOCUS ON SECURITY

SECURING TCP

Ne TCP ne de UDP herhangi bir şifreleme sağlamaz.Sending processinin yuvasına ilettiği veriler , network üzerinden hedef processe giden verilerle aynıdır.Bu nedenle örneğin sending process soketine açık metin halinde (yani şifrelenmemiş) bir parola gönderirse,açık metin parolası sender ve receiver arasındaki tüm bağlantıların üzerinden geçerek potansiyel olarak araya giren bağlantıardan sezilir ve keşfedilir.

Çünkü gizlilik ve diğer güvenlik sorunları birçok uygulama için kritik hale geldiğinden Internet topluluğu TCP için Secure Sockets Later(SSL) adı verilen bir gelştirme geliştirmiştir.

SSL ile geliştirilmiş TCP yalnızca geleneksel TCP nin yaptığı herşeyi yapmakla kalmaz aynı zamanda şifreleme, data integrity(veri bütünlüğü), end poin authentication da dahil olmak üzere kritik processler arası güvenlik hizmetleri sunar.

SSL'in TCP ve UDP ile aynı düzeyde bir internet transport protokolü olmadığını bunu yerine application layer da geliştirmelerin uygulandığı bir TCP geliştirmesi olduğunu vurguluyoruz.

Özellikle bir uygulama SSL hzimetlerini kullanmak istiyorsa uygulamanın hem client hem de server tarafında SSL kodu içermelidir.

SSL in geleneksel TCP soket API'ine benzer kendi soket API 'ı vardır.

Bir uygulama SSL kullanıldığında sending process'i açık metin verilerini SSL soketine iletir. Sending hosttaki SSL daha sonra verileri şifreler ve şifrelenmiş verileri TCP soketine iletir.

Şifrelenmiş veriler Internet üzerinden receiver(alıcı) processteki TCP soketine gider.Receiver soket şifrelenmiş verileri , verilerin şifresini çözen SSL 'ye geçirir. Son olarak SSL,açık metin verileri SSL soketinden receiving process'e geçirir.

Uygulama	Uygulama katmanı protokolü	Ulaşım protokolü
e-mail	SMTP	TCP
Uzaktan terminal erişimi	Telnet	TCP
Web	HTTP	TCP
dosya aktarımı	FTP	TCP
Multimedya akışı	HTTP (örnek,Youtube), RTP	TCP veya UDP
İnternet telefonu	SIP, RTP, özel (örnek, Skype)	Genellikle UDP