Yerel İletişim Ağları (lan) ve Ieee 802.x Protokolleride Başarımı

Mehmet Kavi

Yüksek Lisans Tezi

YILDIZ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YEREL İLETİŞİM AĞLARI (LAN) VE IEEE 802.X PROTOKOLLERİNDE BAŞARIMIN SİMULASYON YÖNTEMİYLE ANALİZİ

YÜKSEK LİSANS TEZİ

MÜH. MEHMET KAVİ

İSTANBUL 1991

YILDIZ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YEREL İLETİŞİM AĞLARI (LAN) VE IEEE 802.X PROTOKOLLERİNDE BAŞARIMIN SİMULASYON YÖNTEMİYLE ANALİZİ

YÜKSEK LİSANS TEZİ

MÜH. MEHMET KAVİ

İSTANBUL 1991

İÇİNDEKİLER

Bu çalışmayı hazırlama ve yönlendirme konusunda yardımlarını esirgemeyen danışman hocam Prof. Metin Yücel'e ve her türlü yardımlarını gördüğüm Bilmar A.Ş.'nin tüm çalışanlarına teşekkür ederim.

ÖZET

Bilgisayar teknolojisi geliştikçe bilgi başlıbaşına bir değer haline gelmiş ve bu değerin paylaşılması gereği daha fazla önem kazanmıştır. Her türlü alanda bilginin kapasite ve çeşit yönünden büyümesi bu bilginin işlenmesi ve paylaşılması için bilgisayarların kullanılmasını zorunlu hale getirmiştir.

Bilindiği gibi bilgisayarların en temel kullanma amaçları hesaplama, bilgi işleme/saklama ve haberleşmedir. Bilgisayarların birçok işlevleri daha çok bilgisayar mühendisliğinin ilgi alanına girmesine rağmen bilgisayar haberleşme konusunda elektronik mühendisliğinin ağırlığı daha fazladır.

İlk bilgisayar sistemleri Anaçatı (mainframe) bilgisayarlar ve bunlara bağlı akılsız terminallerden oluşuyordu. Merkeziyetçi bir felsefeye sahip olan bu sistemler artık yerlerini kendi başına işlem yapabilen bilgisayarların oluşturduğu haberleşme sistemlerine bırakmaktadır. Bunun sonucunda dağıtılmış bilgi işlem sistemleri ortaya çıkmaktadır. Bunun nedeni artık teknolojik gelişmelerin bu değişmeye olanak sağlamasıdır. Bu haberleşme sistemleri genel olarak Bilgisayar İletişim Ağı (Computer Network) olarak adlandırılır.

Bu tez çalışmasında bilgisayar haberleşmesinde son yıllarda yaygınlaşan iletişim ağları incelenmiş ve yerel iletişim ağlarında kullanılan temel protokoller olan IEEE 802 protokolleri arasında başarım farklılıkları araştırılmıştır. Bölüm 2'de bilgisayar ağlarının mimari yapıları araştırılmış ve aralarındaki ilişkiler incelenmiştir. Bölüm 3'te konunun ayrıntısına daha fazla girilmiş ve yerel iletişim ağları (LAN) incelenmiştir. Bu tip ağların X.25 protokolüyle birbirine bağlanması yoluyla oluşturulan geniş ağlar (WAN) da tartışılmıştır. Bölüm 4'te ise IEEE 802 protokollerinde yapılan başarım incelemesinde kullanılan yöntem açıklanmıştır. En son bölümde ise simulasyon sonuçlarından yola çıkılarak sonuçlar tartışılmıştır.

Tez çalışması içinde mümkün olduğu kadar türkçeleştirme yapılmıştır. Ancak bilgisayar ve bilgisayar haberleşmesi konusunda yerleşik Türkçe terimler olmadığından yanlış bir anlamaya meydan vermemek için şüpheli çevrilebilecek terimler aynen bırakılmıştır.

Teknolojide gelişme gözönüne alındığında bilgisayar haberleşmesinde ağ yaklaşımının değişmeyeceği, aksine daha da yaygınlaşarak diğer haberleşme sistemleriyle (ISDN gibi) bütünleşeceği tahmin edilebilir.

SUMMARY

Computer networks are become taking the place of the mainframes. Local Area Networks (LANs) are generally using in office environments for sharing common databases, data storages and peripheral devices. While this networks developed, the WANs has established with connecting several LANs each other. As a result, every computer on the network be able to communicate another computer and access the common knowledge.

In this thesis, some of the network architectures and the IEEE 802 protokols which used on the local area networks are investigated. To analyze performance levels between this protocols, a computer simulation program has been developed according to the general simulation/analytic approach. The results which got from the simulation program has interpreted and explained.

1.1 BILGISAYAR AGLARINA GIRIŞ

Data haberleşmesi konusu son yıllara kadar elektronik ve haberleşme mühendisliğinin özel bir dalıydı. Son 10 yılda bilgisayar teknolojisindeki hızlı gelişme mikrobilgisayarların birbirine bağlanması yoluyla kurulan bilgi işlem sistemlerinde büyük bir artışa yolaçtı. Bu gelişme farklı tipte bilgisayarlar arasında haberleşme sağlamak için kurulan iletişim ağlarının (network) yaygınlaşması ile sonuçlandı. Modern data haberleşmesinde bilgisayar ağları da elektronik mühendisliğinin ilgi alanına girdi.

TARIHSEL GELIŞIM:

Bilgisayar ağlarının evrimi teknolojik ilerlemeye paralel olarak gelişmiştir. İlk bilgisayar sistemleri karmaşık bir donanım ve yazılımdan oluşmaktaydı. Bilgisayar teknolojisi geliştikçe hızlı manyetik ortamlar ve multi-programming (time-shared) yazılımlar geliştirildi. Böylece bilgisayarın merkezi işlem birimi (CPU) zaman paylaşımlı (time-sharing) olarak çalışıyordu ve kullanıcılar programlarını terminallerinden interaktif olarak çalıştırabiliyorlardı. Bu terminaller -telex ağlarında hala kullanılan- elektromekanik TTY (teletypewriter) cihazlarıydı.

Bu bilgisayarlar işletmelerdeki terminallleri "on-line" olarak destekleyebilen "çoklu-erişim (multi-access)" sistemleri olarak biliniyordu. Terminaller başlangıçta bilgisayarın çevresinde bulunurken zamanla terminallerin uzaklıkları arttı ve bağlantı kurmak için telefon hatları ve modemler kullanılmaya başlandı. Haberleşme için gereken zaman ve maliyet çok fazla olduğundan bu nedenle çoğullayıcılar (multiplexer) ortaya çıktı. Bundan başka bilgisayarlara bağlanan terminal sayısının artmasının performansa etkisi büyük oldu ve bu durum ana bilgisayarın işlem yapma hızını oldukça düşürdü. "FEP-Front-End Processor" denilen ve ana bilgisayarın haberleşme işlemlerini yürüten cihazlar da belirli oranda yeterli kaldılar. FEP'ler bir anlamda dağıtılmış bilgi işleme kavramının habercisiydiler.

Kişisel bilgisayarların gelişmesiyle birlikte "**Bilgisayar Ağları (Computer Networks)**" kavramı ortaya çıktı. Bir bilgisayar ağı, tanım olarak farklı yerlerde bulunan ve birbirinden bağımsız çalışabilen bilgisayarların haberleşmesini sağlayan haberleşme sistemidir. Literatürde bilgisayar ağı ile dağıtılmış sistem **(Distributed System)** arasında bir kavram karışıklığı vardır. İletişim ağı bir haberleşme sistemidir, dağıtılmış sistem ise hesaplama işlemlerinin birkaç bilgisayar tarafından yapıldığı bir sistemdir. Dağıtılmış sistemde sistemin işletim sistemi gelen istekleri isimlere göre yerine getirir, iletişim ağında ise istekler fiziksel yerlere ulaşır. Fakat yine de dağıtılmış sisteme özel bir ağ olarak da bakılabilir.

İletişim ağlarının çeşitli kıstaslara göre sınıflandırmaları yapılmıştır. Şekil.1.1.'deki sınıflandırma bu ağların fiziksel büyüklüğüne göre yapılmıştır.

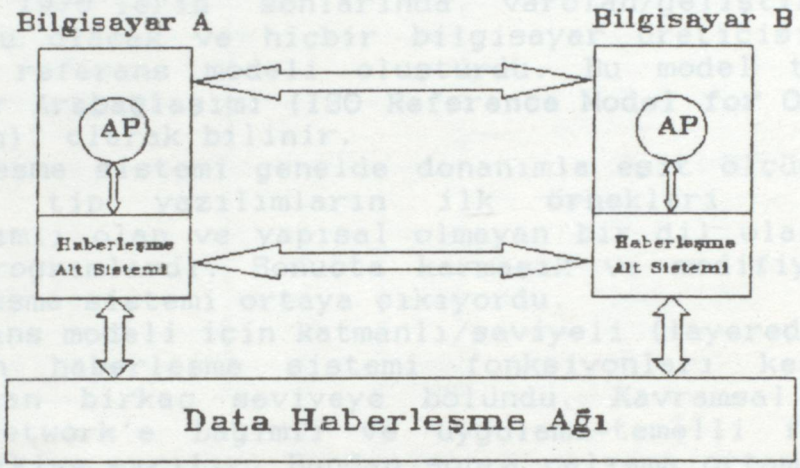
Uzaklık	Ortam	Örnek
0.1 m	Baskılı Devre	CPU kartı
1 m	Sistem	Multiprocessor Bus
10 m	Oda	Yerel İletişim ağları (LAN)
100 m	Bina	
1 km	Kampus	
10 km	Şehir	Uzun mesafeli ağlar (WAN)
100 km	Ülke	
1.000 km	Kıta	WAN'ların birbirine bağlanması
10.000 km	Gezegen	

Şekil.1.1: Ağların fiziksel genişliğine göre sınıflandırılması.

Bilgisayar ağlarının merkezi bilgisayar sistemlerinin yerini almasının iki nedeni vardır. Bunlardan birincisi işletmelerde birbirinden ayrı olarak çalışan bilgisayarların zaten varolmasıdır. Bu bilgisayarların haberleşme yeteneği kazanmasıyla bilgiye her uzaklıktan ulaşılması ve diğer kaynakların kullanılması çok kolaylaşacaktır. Diğer bir neden ise ağ kaynaklarının esnek olarak kullanılmasıdır. Tek başına çalışan bir bilgisayardaki donanım arızası işlerin durmasına neden olur ki bu da bankacılık, askeri uygulamalar, endüstriyel proses kontrolu gibi durumlarda kesinlikle istenmeyen bir durumdur. Ağ'da bir bilgisayarın bozulması daha az tehlikelidir, kullanıcılar bir başka bilgisayardan aynı bilgiye ulaşabilirler. İletişim ağlarının yaygınlaşmasının önemli diğer bir nedeni de bilgisayar ve haberleşme maliyetleri arasındaki değişmedir. 1970'lere kadar bilgisayarlar haberleşme cihazlarına göre görece olarak daha pahalıydılar. Bugün ise bunun tersi geçerlidir. Mainframe bilgisayarlar küçük bilgisayarlardan yaklaşık 10 kez daha hızlıdır, fakat fiyat olarak 1000 kez daha pahalıdır. Bunun sonucunda yerel iletişim ağları ortaya çıkmıştır ki hazırlanan tez çalışması bu konu üzerindeki araştırma sonucudur.

2.1 BILGISAYAR AĞ YAPILARI/MIMARILERI

Bilgisayar haberleşmesi üzerinde çalışan kurum ve kuruluşlar bu ağların geliştirilmesinde farklı yollar izlemişler ve bu nedenle farklı sistemler ortaya çıkmıştır. Uygulamadaki bu farklılıklara rağmen genelde bir haberleşme ağı Şekil.2.1.1'deki diagramla gösterilebilir.



Şekil.2.1.1: Bilgisayar haberleşmesi.

Bilgisayar haberleşmesinin ilk yıllarında bazı bilgisayar şirketleri, örneğin IBM Systems Network Architecture (SNA), DEC Digital Network Architecture (DNA) gibi özel mimariler geliştirdiler. Bu sistemler arasındaki haberleşmede sorunlar ortaya çıkmaya başlayınca Uluslararası Standartlar Örgütü (ISO- International Standards Organization) 1979'da daha önce geliştirilmiş olan standartları koordine etmek ve yeni gelişenlere uyum sağlamak için genel bir referans modeli oluşturdu. ISO'nun "**Referans Model for Open Systems Interconnection**" adıyla bilinen bu referans modeli bilgisayar haberleşme ağlarındaki özel uygulamalardan çok haberleşme yazılımlarını yapısallaştırıp üretici şirketlerden bağımsız hale getirmek için geliştirilmiştir. Bundan başka telefon haberleşmesinde kullanılan X.25, Xerox'un Xerox Network Systems (XNS) ve fiber optik teknolojisiyle yeni gelişen Fiber Distributed Data Interface (FDDI) gibi network mimarileri de en bilinen referans modelleridir. Yukarıda bahsedilen bütün modeller sırasıyla incelenecek ve aralarında bir karşılaştırma yapılacaktır.

2.2 OSI (Open Systems Interconnection) REFERANS MODELİ

2.2.1 GİRİŞ:

OSI referans modelinden önceki sistemler üretici firmaların kendi ağları üzerine kurulmuş "kapalı" sistemlerdi. SNA ve DNA (DECNET) bunlara tipik örneklerdir. Bu tip kapalı sistemlerin haberleşebilmesi için herkesin kabul edeceği açık bir sisteme ihtiyaç vardı. Bu sorunu çözmek için **ISO (International Standards Organization)** 1970'lerin sonlarında varolan/geliştirilen sistemlerle uyumlu olacak ve hiçbir bilgisayar üreticisine bağımlı olmayacak bir referans modeli oluşturdu. Bu model tam ismiyle **"Açık Sistemler Arabağlaşımı (ISO Reference Model for Open Systems Interconnection)"** olarak bilinir.

Bir haberleşme sistemi genelde donanımla eşit ölçüde yazılıma dayanır. Eski tip yazılımların ilk örnekleri haberleşme donanımına bağımlı olan ve yapısal olmayan bir dil olan assembler ile yazılan programlardı. Sonuçta karmaşık ve modifiye edilmesi zor bir haberleşme sistemi ortaya çıkıyordu.

ISO, referans modeli için katmanlı/seviyeli (layered) bir model getirdi. Bütün haberleşme sistemi fonksiyonları kesin olarak tanımlanmış olan birkaç seviyeye bölündü. Kavramsal olarak bu fonksiyonlar network'e bağımlı ve uygulama-temelli fonksiyonlar olarak kabaca ikiye ayrılır. Bundan sonra çalışma ortamlarına göre bir inceleme yapılabilir:

(1) **Ağ** (Network) Ortamı, çeşitli tipteki ağlarla ilişkili protokoller ve standartlarla ilgilidir.

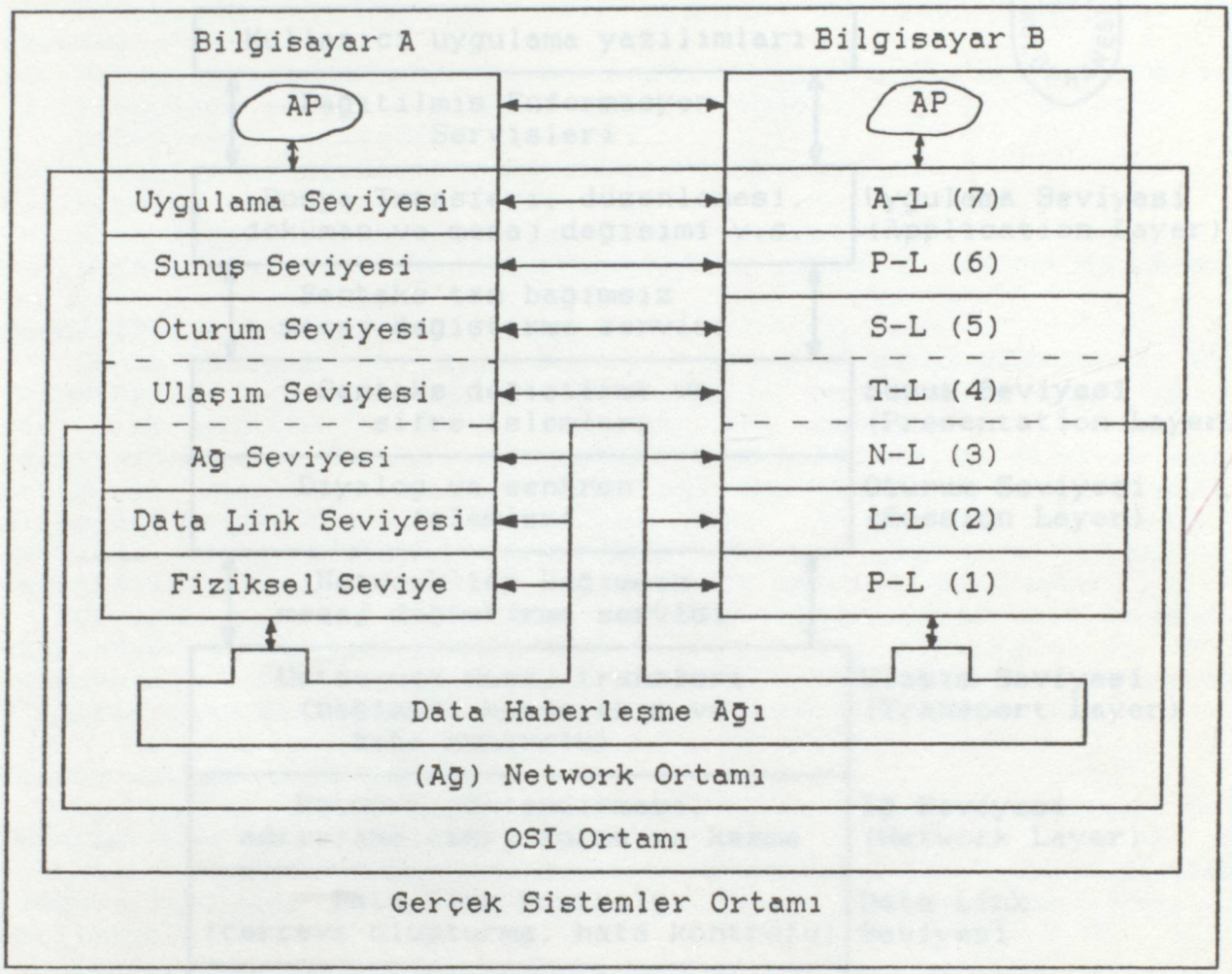
(2) OSI Ortamı, ağ ortamına uygulama-temelli protokolleri ekleyerek uç sistemlerin bir açık sistem üzerinden birbiriyle haberleşmesiyle ilgilidir.

(3) Gerçek Sistemler Ortamı, OSI ortamı üzerine üretici firmaların kendi sistemlerini oturtmasını ve kullanmasını sağlar.

ISO referans modelinin seviyeli yapısı ve çalışma ortamları **Şekil.2.2.1**'de gösterilmiştir. OSI'nin network'e bağımlı ve uygulama-temelli (network'ten bağımsız) bölümleri kendi içinde alt seviyelere bölünmüştür. Her protokol seviyesinin sınırları daha önceki standartlardan edinilen tecrübe ile belirlenmiştir.

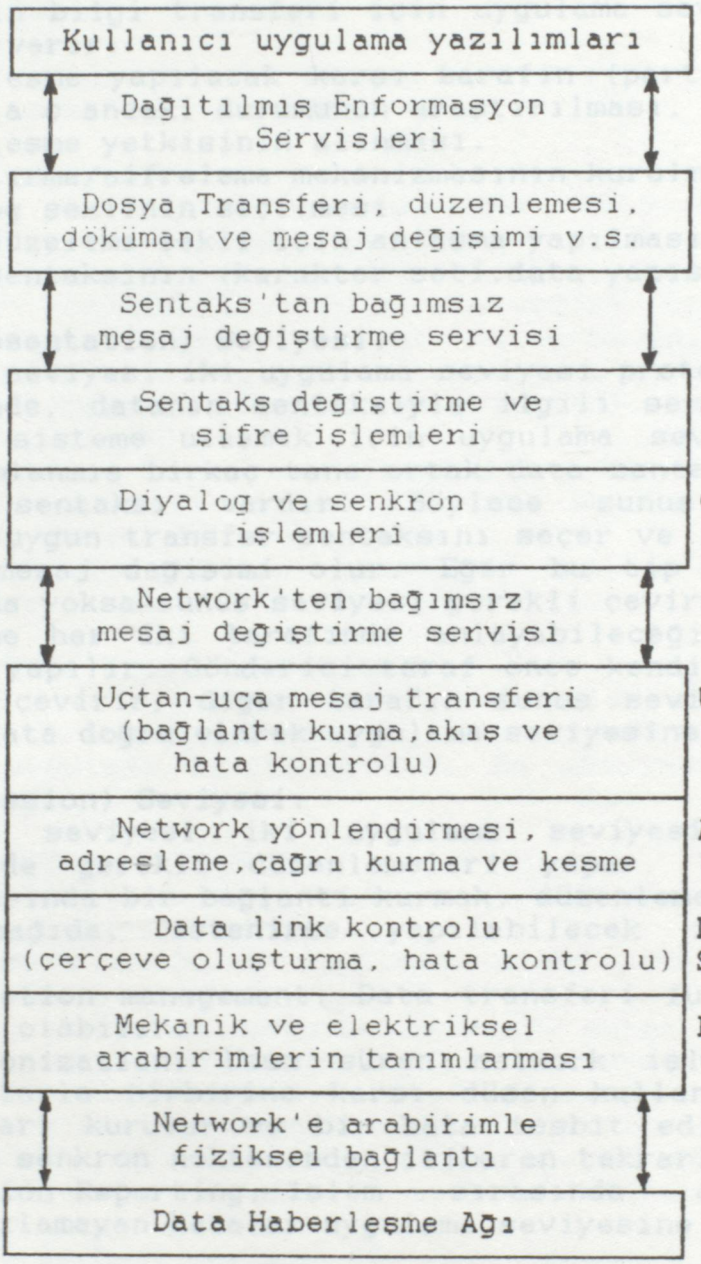
Her seviyenin kesin olarak belirlenmiş fonksiyonları vardır ve mesajların iletilmesinde tanımlanan protokollerle çalışarak diğer taraftaki sistemde kendi seviyesine eş servis verir. Her seviye kendisinin alt ve üstündeki seviyelerle yine kesin tanımlanmış arabirimlerle haberleşir, bunun sonucu olarak bir seviyede yapılan işlem diğerlerinden tamamen bağımsız olur.

ISO referans modelinin mantıksal yapısı **Şekil.2.2.1**'den görüleceği gibi yedi seviyeye bölünmüştür. En alttaki üç seviye network'e bağımlıdır ve bilgisayarları birbirine bağlayan network'ün kullandığı protokollerle ilgilidir. Benzer olarak en üst üç seviye de uygulama yazılımına dayanır ve normalde network işletim sisteminin servisleri ile bu seviyeye ulaşılır. Ortadaki ulaşım seviyesi ise bu iki grup seviyeyi birbirine bağlayan işlemleri yerine getirir.



Şekil.2.2.1: ISO Referans Modelinin Yapısı.

Her seviyenin fonksiyonu diğer sistemde kendisine karşı düşen seviyenin fonksiyonuyla aynı olmak zorundadır, böylece her seviye altındaki ve üstündeki seviyelere aynı servisleri verdiğinden bu seviyeler mantıksal olarak birbirine bağlı olur. Örneğin ulaşım seviyesi oturum seviyesine network'ten bağımsız mesajlar verir, aynı şekilde network seviyesine karşı tarafın ulaşım seviyesine gidecek mesajı uygun bir şekilde iletir. Mantıksal olarak her seviye diğer sisteminin aynı seviyesine bağlı olmasına rağmen pratikte mesajlar alt seviyelerden geçerek diğer sisteme ulaşır. Şekil.2.2.2'de her seviyenin fonksiyonu özetlenmiştir.



Şekil.2.2.2: Protokol seviyelerinin özeti.

UYGULAMA-TEMELLİ SEVİYELER

Uygulama (Application) Seviyesi:

Uygulama Seviyesi kullanıcıya network'deki servislere ulaşmak için bir arabirim sunar. Bunların içinde dosya transferi ve düzenlemesi, döküman ve mesaj alışverişi gibi servisler vardır.

Uygulama servislerine network işletim sistemi tarafından desteklenen protokoller ve fonksiyonlarla ulaşılır. Bu özellikler işletim sisteminin diğer fonksiyonları gibi çağrılır ve işletim sistemi haberleşme sisteminin yazılım ve donanımına erişerek istenen servisi verir. Haberleşme sisteminin ayrıntılı fonksiyonlarına erişmek de kullanıcıya açıktır.

Örneğin bilgi transferi için uygulama seviyesi aşağıdaki gibi servisler verir:

- Haberleşme yapılacak karşı tarafın (partner) adres veya isim yoluyla o andaki durumunun araştırılması.
- Haberleşme yetkisinin alınması.
- Sıkıştırma/şifreleme mekanizmasının kurulması.
- Diyalog şeklinin seçilmesi.
- Hata düzeltme şekli için anlaşma yapılması.
- Data sentaksının (karakter seti,data yapısı..) kabul edilmesi.

Sunuş (Presentation) Seviyesi:

Sunuş seviyesi iki uygulama seviyesi protokolü arasındaki data transferinde, datanın sentaksıyla ilgili servisler verir. Gerçek bir açık sisteme ulaşmak için uygulama seviyesinde kullanılmak için tanımlanmış birkaç tane ortak data sentaksı ve bunlara bağlı transfer sentaksı vardır. Böylece sunuş seviyesi transfer sırasında uygun transfer sentaksını seçer ve iki uygulama seviyesi arasında mesaj değişimi olur. Eğer bu tip gösterim ortak data sentaksında yoksa sunuş seviyesi gerekli çevirmeleri yapar.

Çevirme her iki tarafında anlayabileceği ortak bir protokol üzerinden yapılır. Gönderici taraf önce kendi sentaksını bu ortak protokole çevirir, diğer tarafın sunuş seviyesinde de ters çevirmeyle data doğru olarak uygulama seviyesine ulaşmış olur.

Oturum (Session) Seviyesi:

Oturum seviyesi iki uygulama seviyesi protokolünün data transferinde gerekli düzenlemeleri yapar. Yani bu seviye iki sistem arasında bir bağlantı kurmak, düzenlemek ve kesmekle yükümlüdür. Aşağıda, istenirse yapılabilecek işlemlerden bazıları verilmiştir.

- **Interaction management:** Data transferi full-duplex veya half-duplex olabilir.
- **Synchronization:** Uzun süren network işlemleri için belirli aralıklarla birbirine karşı düşen kullanıcılar için senkron noktaları kurulur ve bir hata tesbit edildiği zaman diyalog önceki senkron noktasından itibaren tekrarlanır.
- **Exception Reporting:** İşlem sırasında oluşan ve aradan çıkartılamayan hatalar uygulama seviyesine rapor edilir.

Ulaşım (Transport) Seviyesi:

Ulaşım seviyesi uygulama-temelli seviyeler ile network'e bağımlı protokoller arasında bağlantı servisi verir. Ulaşım seviyesi alttaki network'ün karmaşık operasyonlarını maskeleyerek oturum seviyesine network'ten bağımsız ve güvenli mesaj transferi sağlar.

Çeşitli network tiplerine göre ulaşım seviyesi birkaç tane servis sınıfı önerir. Buna alternatif olan servis sınıfı da Servis Kalitesi (QOS-Quality of service) farklı network tipleri için vardır.

NETWORK'E BAGIMLI SEVIYELER

Network seviyesi:

Network üzerinde iki ulaşım seviye protokolü arasında bağlantı kurmak ve kesmekle yükümlüdür.

Data Link seviyesi:

Network seviyesine güvenli bir data akışı sağlamak için fiziksel bağlantı kurar. Bu seviye hata kontrolu, transmisyon hatalarında mesajların yeniden gönderilmesi gibi fonksiyonlardan sorumludur. Normalde iki tip servis verir:

- **Bağlantısız (Connectionless)**, datagram da denilen bu servis her çerçevenin kendi bilgisini taşıdığını varsayarak gönderir.
- **Bağlantı-temelli (Connection oriented)**, Hatasız bilgi transferi için servis verir.

Fiziksel seviye: Kullanıcı cihazı ile network arasında elektriksel ve fiziksel arabirim sağlar. Fiziksel seviye data link seviyesine bilgiyi bit dizisi halinde iletir.

OSI referans modelini oluşturan yedi seviye ayrıntılı olarak bundan sonraki bölümlerde incelenecektir.

2.2.2 UYGULAMA TEMELLI SEVIYELER

Şekil.2.2.3'te uygulama-temelli seviyeler diagram halinde gösterilmiştir. Bunlar yukarıda da kısaca bahsedildiği gibi oturum, sunuş ve uygulama seviyeleridir. Bu seviyeler aşağıda sırasıyla incelenecektir.

OTURUM (SESSION) SEVIYESI

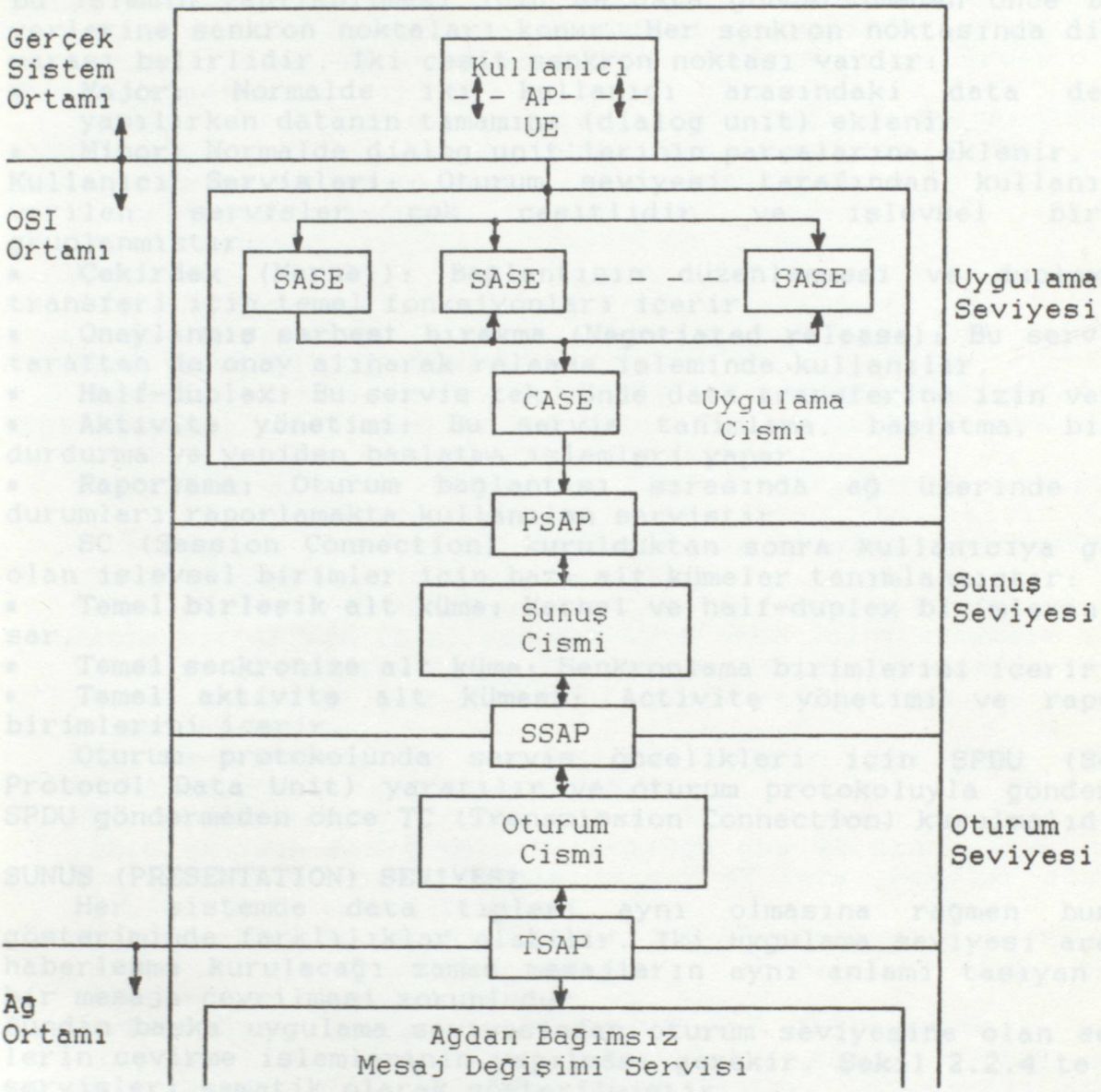
Oturum seviyesi, uygulama seviyesine sunuş seviyesi üzerinden aşağıdaki servisleri verir:

- Bir başka cihazın uygulama cismi (entity) ile mantıksal haberleşme yolu kurar;
- Diyalog sırasında senkron noktaları kurar ve hata olduğunda bir önceki senkron noktasından diyaloğu sürdürür;
- Önceden belirlenmiş bir noktada diyaloğu keser veya ara verir;
- Oturum sırasında network'teki olaylardan bilgi sahibi olur.

Token Kavramı: İki uygulama cismi oturum seviyesindeki bağlantı üzerinden diyaloğa girmek için aşağıdaki token'lerden yararlanır.

- Data tokeni,
- Token'i serbest bırakma (Release token),
- Senkronizasyon token'i,
- Aktivite token'i,

Bütün bu token'lar oturum-servisinin (SS-Session-Service) kullanıcılarından birinden diğer bir kullanıcıya karşı düşen servis için atanır. Token'in o anki sahibi ilişkili servisi eline alır. Örneğin data token iki kullanıcı arasında half-duplex data alışverişi için ve release token de bağlantıyı bitirmek için kullanılır.



AP : Application Process
UE : User Element
CASE: Common Application Service Element
SASE: Specific Application Service Element
PSAP: Presentation Service Access Point
SSAP: Session Service Access Point
TSAP: Transport Service Access Point

Şekil.2.2.3: Uygulama-temelli Seviyeler.

Senkronizasyon ve aktivite tokeni oturum sırasında senkronizasyon işlemleri için kullanılır. İki SS kullanıcısı büyük miktarda data alışverişi yapacaksa, data küçük parçalara bölünür ve haberleşmede bir hata olduğunda sadece en son parça zarar gördüğünden bu parçadan itibaren tekrar transfer yapılır.

Bu işlemin yapılabilmesi için de data gönderilmeden önce belirli yerlerine senkron noktaları konur. Her senkron noktasında dizi numarası belirlidir. İki çeşit senkron noktası vardır:

- **Major:** Normalde iki kullanıcı arasındaki data değişimi yapılırken datanın tamamına (dialog unit) eklenir.
- **Minor:** Normalde dialog unit'lerinin parçalarına eklenir.

Kullanıcı Servisleri: Oturum seviyesi tarafından kullanıcılara verilen servisler çok çeşitlidir ve işlevsel birimlere gruplanmıştır:

- **Çekirdek (Kernel):** Bağlantının düzenlenmesi ve duplex data transferi için temel fonksiyonları içerir.
- **Onaylanmış serbest bırakma (Negotiated release):** Bu servis iki taraftan da onay alınarak release işleminde kullanılır.
- **Half-duplex:** Bu servis tek yönde data transferine izin verir.
- **Aktivite yönetimi:** Bu servis tanımlama, başlatma, bitirme, durdurma ve yeniden başlatma işlemleri yapar.
- **Raporlama:** Oturum bağlantısı sırasında ağ üzerinde oluşan durumları raporlamakta kullanılan servistir.

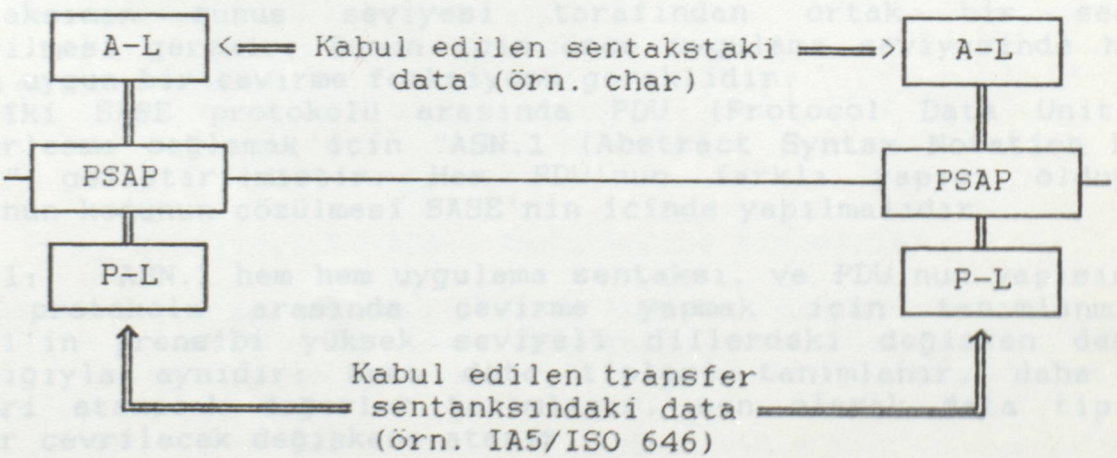
SC (Session Connection) kurulduktan sonra kullanıcıya gerekli olan işlevsel birimler için bazı alt kümeler tanımlanmıştır:

- **Temel birleşik alt küme:** Kernel ve half-duplex birimlerini kapsar.
- **Temel senkronize alt küme:** Senkronlama birimlerini içerir.
- **Temel aktivite alt kümesi:** Activite yönetimi ve raporlama birimlerini içerir.

Oturum protokolunda servis öncelikleri için **SPDU** (Session Protocol Data Unit) yaratılır ve oturum protokoluyla gönderilir. SPDU göndermeden önce **TC** (Transmission Connection) kurulmalıdır.

SUNUŞ (PRESENTATION) SEVİYESİ

Her sistemde data tipleri aynı olmasına rağmen bunların gösteriminde farklılıklar olabilir. İki uygulama seviyesi arasında haberleşme kurulacağı zaman mesajların aynı anlamı taşıyan ortak bir mesaja çevrilmesi zorunludur.
Bundan başka uygulama seviyesinden oturum seviyesine olan servislerin çevirme işlemlerinin yapılması gerekir. **Şekil.2.2.4**'te sunuş servisleri şematik olarak gösterilmiştir.



Şekil.2.2.4: Sunuş servisleri.

Sunuş Servisleri: Sunuş ve oturum seviyeleri uygulama seviyesinin altında bütün olarak servis verdiğinden sunuş servisi öncelikleri buna karşı düşen PPDU (Presentation Protocol Data Unit) biriminde gösterilir.

Bunlara ek olarak direkt olarak SS'e değişiklik yapmadan yani PPDU üretilmeden kullanılan sunuş servisleri de vardır. Bunlar normalde SS'de kullanıcı datası bölümünde bulunur.

- Senkronizasyon kontrol,
- Token kontrol,
- Raporlama,
- Aktivite yönetimi.

UYGULAMA (APPLICATION) SEVIYESI

Uygulama seviyesi herbiri bir **SASE (Specific Application Service Element)** tarafından yerine getirilen ve bazıları aşağıda listelenen bilgi işlem servislerini destekler:

- **FTAM** (File Transfer,Access and Management): Bu servis kullanıcıya UE (User Element) üzerinden kullanıcı AP'sine (Application Process) bir başka AP'nin elindeki dosya sistemine ulaşma olanağı sağlar.
- **JTM** (Job Transfer and Manipulation):Bu servis kullanıcı AP'ına başka bir AP'nin işini yapmaya veya onun işini izlemeye yarar.
- **VT** (Virtual Terminal): Bu servis lokal terminalle diyalog kurmak, bir AP ile standart yolla haberleşme kurmak gibi hizmetler verir.
- **MHS** (Message Handling Service):Bu servis iki AP arasında elektronik mesaj değişimi sağlar.
- **MMS** (Manufacturing Message Service):Bu servis örneğin otomasyonla çalışan bir üretim merkezinde ana AP'den kontrol edilen denetleyici ve robotlar gibi diğer AP'lere mesajlar göndermek ve almak gibi işlemler yapar.
- **DS** (Directory Services): Kullanıcı AP'si karşı taraftaki kullanıcı AP'sini sembolik adlar ve DS'nin sağladığı network adresleriyle ulaşabilir.

Herbiri belirli uygulama servisi veren SASE'ler ve bu servisleri toplayıp sunuş seviyesiyle haberleşme sağlayan CASE'ler iki uygulama seviyesininin haberleşmesini sağlar. OSI farklı bilgisayarlarda çalışan farklı AP'lere izin verdiğinden bütün bu AP'lerin sentaksının sunuş seviyesi tarafından ortak bir sentaksa çevrilmesi gerekir. Bunun için önce uygulama seviyesinde her AP için uygun bir çevirme fonksiyonu gereklidir.

İki SASE protokolü arasında PDU (Protocol Data Unit) ile haberleşme sağlamak için "**ASN.1 (Abstract Syntax Notation Number One)**" geliştirilmiştir. Her PDU'nun farklı yapısı olduğundan PDU'nun kodunun çözülmesi SASE'nin içinde yapılmalıdır.

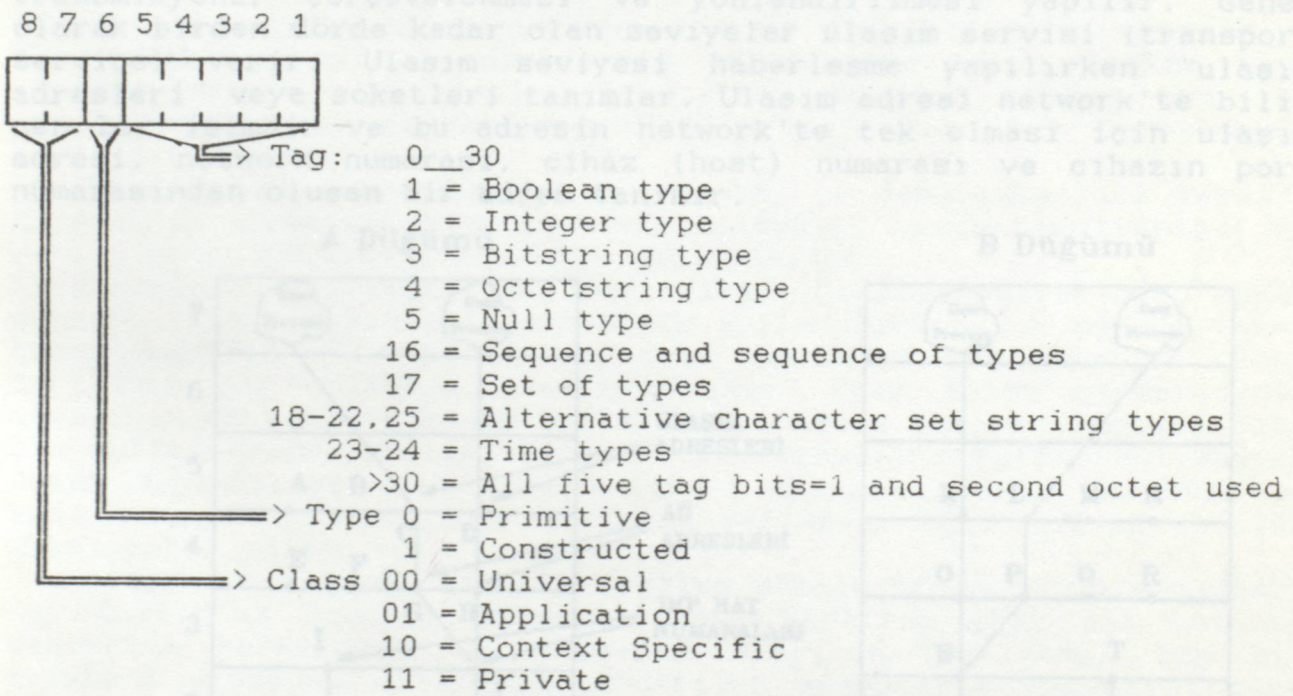
ASN.1: ASN.1 hem hem uygulama sentaksı, ve PDU'nun yapısına uygun protokolu arasında çevirme yapmak için tanımlanmıştır. ASN.1'in prensibi yüksek seviyeli dillerdeki değişken değişimi mantığıyla aynıdır: önce data tipleri tanımlanır, daha sonra değeri atanacak değerler tanımlanır, son olarak data tipindeki değer çevrilecek değişkene atanır.

Şekil.2.2.5'teki diagramda ASN.1 tip tanımlayıcı byte içeriği gösterilmiştir. ASN.1 aşağıdaki tip tanımlayıcılarını destekler:

- UNIVERSAL: Bunlar tamsayılar gibi genelleşmiş tiplerdir.
- CONTEXT_SPECIFIC: Kullanılan duruma özel tiplerdir.
- APPLICATION: Tamamen uygulama seviyesine özgü tiplerdir.
- PRIVATE: Kullanıcı tarafından tanımlanabilen tiplerdir.

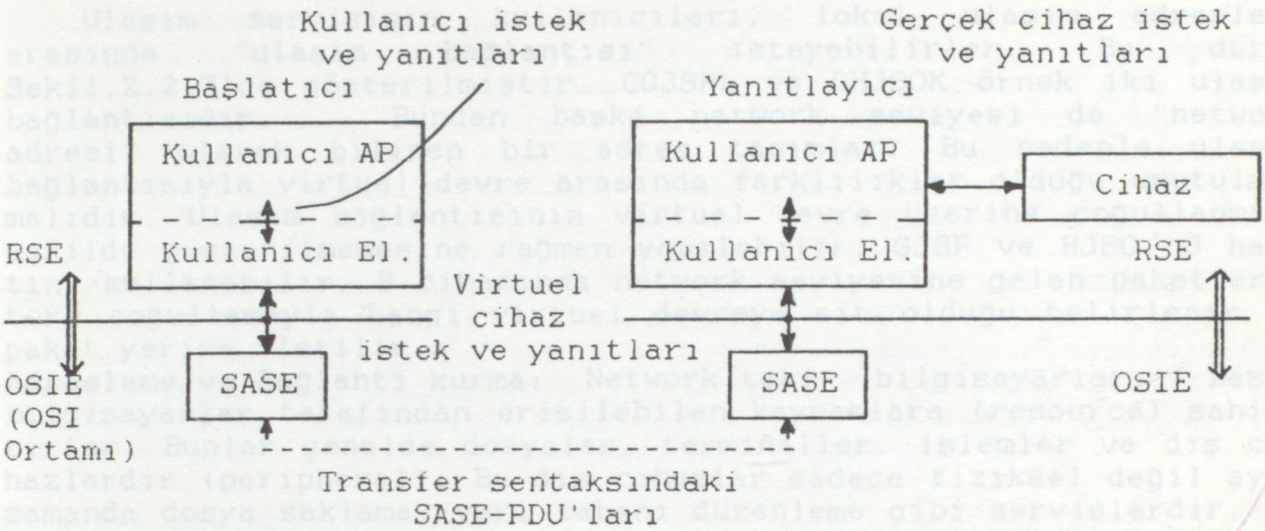
UE (USER ELEMENT): OSI ortamıyla (OSIE) gerçek sistemler ortamı (RSE) arasında bir tanılanan arabirimdir. Bu arabirim "kullanıcı arabirimi" adını alır ve uygulama yazılımı aracılığıyla OSI ortamına girişi sağlar. Genelde her uygulama servisi kullanıcı tarafından erişilecek servisler virtuel cihaz olarak tanımlar. Kullanıcı istekleri virtuel cihaz tarafından yerine getirilir.
CASE: Şekil.2.2.4.'den de görüldüğü gibi CASE sunuş seviyesi ile ilişki kurar. CASE'nin de kendi içinde değişik protokolleri vardır:

- İki SASE arasında mantıksal bağlantı kuran ve kesen **ACSE** (Association Control Service)
- Diğer SASE üzerinde işlemler yapabilen **ROSE** (Remote Operations Service Element)
- Küçük çaplı işlemler yapabilen **CCR** (Commitment, Concurrency and Recovery).



Şekil.2.2.5: ASN.1 tanımlayıcı bit dizisi.

Şekil.2.2.6'da virtuel cihaz kavramı şematik olarak gösterilmiştir. Pratikte UE (User Element) kullanıcı AP'sine eklenen hazır fonksiyonlardır. Normalde bir başlatıcı ve bir yanıtlayıcı AP vardır.

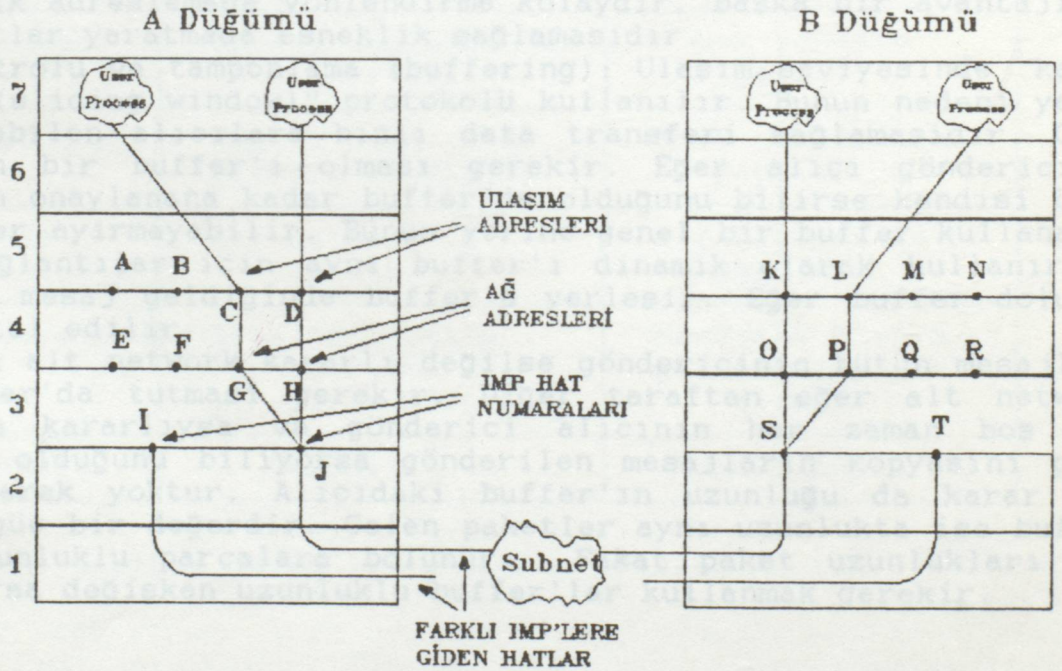


Şekil.2.2.6: Virtual device yaklaşımı.

2.2.3 NETWORK'E BAGIMLI SEVIYELER

ULAŞIM (TRANSPORT) SEVİYESİ

OSI'nin en alt üç seviyesinde iki cihaz arasında paketlerin transmisyonu, çerçevelenmesi ve yönlendirilmesi yapılır. Genel olarak birden dörde kadar olan seviyeler ulaşım servisi (transport service) verir. Ulaşım seviyesi haberleşme yapılırken "**ulaşım adresleri**" veya soketleri tanımlar. Ulaşım adresi network'te bilinen bir isimdir ve bu adresin network'te tek olması için ulaşım adresi, network numarası, cihaz (host) numarası ve cihazın port numarasından oluşan bir adres tanınır.



Şekil.2.2.7: Ulaşım ve Network adresleri.

Ulaşım servisinin kullanıcıları, lokal ulaşım adresleri arasında "**ulaşım bağlantısı**" isteyebilirler. Bu durum **Şekil.2.2.7**'de gösterilmiştir. CGJSPL ve DHJSOK örnek iki ulaşım bağlantısıdır. Bundan başka network seviyesi de "network adresi" olarak bilinen bir adres tanımlar. Bu nedenle ulaşım bağlantısıyla virtuel devre arasında farklılıklar olduğu unutulmamalıdır. Ulaşım bağlantısının virtuel devre üzerine çoğullanması şekilde gösterilmemesine rağmen yapılabilir. GJSP ve HJSO, J hattını kullanabilir. B cihazında network seviyesine gelen paketlerin ters çoğullamayla hangi virtuel devreye ait olduğu belirlenir ve paket yerine iletilir.

Adresleme ve Bağlantı kurma: Network'teki bilgisayarlar, başka bilgisayarlar tarafından erişilebilen kaynaklara (resource) sahiptirler. Bunlar genelde dosyalar, terminaller, işlemler ve dış cihazlardır (peripheral). Bu dış cihazlar sadece fiziksel değil aynı zamanda dosya saklama, veri tabanı düzenleme gibi servislerdir. Bu servisleri yerine getiren cihazlara da "**Server**" denir, bunların görevi terminallerin isteğine göre haberleşme servisleri vermektir.

Servis isteyen kullanıcıyla server arasında bağlantı kurmanın yolu istenen servise bir isim vermek ve server'a kullanıcı için gereken adresi vermektir. Adresleme bir isimle bir adres arasında yapılır.

Genelde ulaşım adresi vermek için iki yol vardır, **hiyerarşik** ve **düz (flat)** adresleme. Hiyerarşik adreslemede adres birkaç tane alandan oluşur, bu alanlar belli bir bölgeyi temsil eder ve adresin bilinmesi server'ın yerini de belirtir. Düz adreslemede böyle bir bağlantı yoktur.

Her iki tür adreslemenin de avantaj ve dezavantajları vardır. Hiyerarşik adreslemede yönlendirme kolaydır, başka bir avantajı da yeni portlar yaratmada esneklik sağlamasıdır.

Akış kontrolu ve tamponlama (buffering): Ulaşım seviyesinde "kayan pencere (sliding window)" protokolü kullanılır. Bunun nedeni yavaş haberleşebilen alıcılara hızlı data transferi sağlamasıdır. Göndericinin bir buffer'i olması gerekir. Eğer alıcı göndericinin mesajının onaylanana kadar buffer'da olduğunu bilirse kendisi özel bir buffer ayırmayabilir. Bunun yerine genel bir buffer kullanarak bütün bağlantılar için aynı buffer'ı dinamik olarak kullanır ve yeni bir mesaj geldiğinde buffer'a yerleşir. Eğer buffer doluysa mesaj iptal edilir.

Alıcı alt network kararlı değilse göndericinin bütün mesajları bir buffer'da tutması gerekir. Diğer taraftan eğer alt network gerçekten kararlıysa ve gönderici alıcının her zaman boş bir buffer'ı olduğunu biliyorsa gönderilen mesajların kopyasını geri almaya gerek yoktur. Alıcıdaki buffer'ın uzunluğu da karar verilmesi güç bir değerdir. Gelen paketler aynı uzunlukta ise buffer sabit uzunluklu parçalara bölünür. Fakat paket uzunlukları çok değişiyorsa değişken uzunluklu buffer'lar kullanmak gerekir.

Başka bir sorun da alt network'ün taşıma kapasitesidir. Eğer bilgisayarlar saniyede **x** çerçeve taşıyabiliyorsa ve **k** tane bağlantı çifti varsa bu bilgisayarların ne kadar buffer'ı olursa olsun saniyede **k.x** kadar mesajdan fazlasını taşıyamayacak demektir. Kayan pencere protokolü kullanıldığında eğer network saniyede **c** mesajı taşıyabiliyorsa ve bir haberleşme zamanı (transmisyon, sıraya sokma, alıcıdaki işlemler ve onaylama) da **r** ise göndericinin pencere uzunluğu **c.r** olmalıdır.

Çoğullama: Network mimarisinde bağlantılar, virtuel devreler ve fiziksel hatlar arasındaki haberleşmede birden fazla haberleşmenin aynı anda yapılması çoğullama ile sağlanır. Eğer üst seviyelerden gelen mesajlar çoğullanıp daha alt seviyelere ve diğer bilgisayara gidiyorsa buna "yukarı çoğullama (upward multiplexing)", tersi duruma ise "aşağı çoğullama (downward multiplexing)" denir.

AĞ (NETWORK) SEVİYESİ

Network'lerin bağlı bulunduğu alt network'lerle (subnet) haberleşmesi iki kavramsal model aracılığıyla olur. Bunlardan ilki "**Virtuel Devre**" modelidir ve bu modelde network seviyesi ulaşım seviyesine mükemmel bir kanal sağlar, yani ulaşım seviyesine iletilen mesajlar hatasızdır. İkinci model olan "**Datagram**" modelinde network seviyesi mesajları ulaşım seviyesinden tek tek ve birbirinden bağımsız olarak alır. **Şekil.2.2.8**'de virtuel devre ile datagram servisi arasındaki temel farklılıklar özetlenmiştir.

Virtuel devre ile datagram servisi arasındaki farkları incelerken kolaylık açısından ulaşım seviyesi yerine "**host**" ve network seviyesi yerine de "**subnet**" terimi kullanılacaktır.

İşlem	Virtuel Devre	Datagram
Varış adresi	Sadece başlangıçta gerekli	Her pakette gerekli
Hata saptama	Host'a açık (alt network'te yapılır	Host tarafından yapılır.
Uçtan-uca akış kontrolu	Alt network'te yapılır	Alt network'te yapılmaz.
Paket sıralama	Mesajlar host'a gönderilme sırasıyla alınır	Mesajlar host'a gelme sırasıyla alınır.
Başlangıç	Gerekli	Mümkün değil

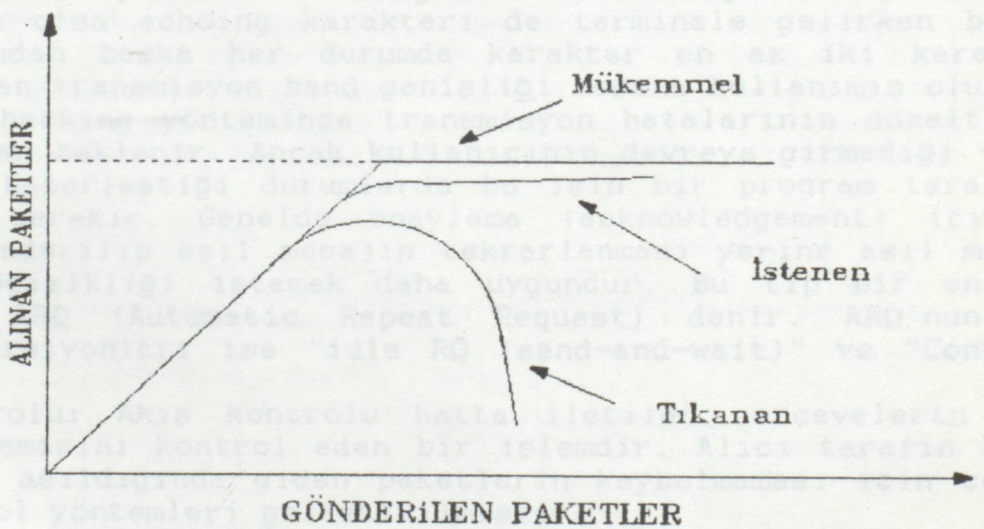
Şekil.2.2.8: Virtuel devre ile datagram arasındaki temel farklar.

Datagram servisi en basit servistir. Alt network'e datagram servisi verildiğinde host bu servise hata ve dizi kontrolu ekleyecektir. Virtuel devrelerin dizi kontrolu yapmak gibi bir sorunu yoktur. Fakat eğer kullanıcılar sıralamaya dikkat etmezlerse bu avantaj dezavantaja dönüşebilir. Bundan başka kullanıcılar hata ve akış kontrolu da yapmalıdır.

Yönlendirme algoritmaları: Yönlendirme algoritması network seviyesine gelen paketlerin alıcıya hangi yoldan gideceğine karar veren bir yazılımdır. Eğer alt network datagram servisiyle çalışıyorsa bu karar giden her pakette yapılmalıdır. Eğer alt network virtuel devre kuruyorsa yönlendirme kararı sadece virtuel devre kurulurken yapılır, bundan sonra data paketleri hep aynı yoldan iletilir. Bir sonraki aşama da "oturum yönlendirmesi"dir, burada yönlendirme oturum boyunca yapılır.

Yönlendirme algoritmaları adaptif olan ve adaptif olmayan yönlendirme olarak iki ana sınıfa ayrılır. Adaptif yönlendirme algoritmaları yönlendirme kararını trafik ve topolojinin o andaki durumuna göre verir, diğer tipte ise bu karar sabit yollar üzerinden verilir.

Tıkanıklık: Network seviyesinde iletilen paket sayısı çok arttığında "tıkanıklık (congestion)" denilen performans düşüklüğü durumu ortaya çıkar. Şekil.2.2.9'da bu durum gösterilmiştir.



Şekil.2.2.9: Trafik çok arttığında performans aniden düşer.

Tıkanıklık birkaç faktör değiştirilerek aşılabilir. Eğer haberleşme cihazının işlemcisi (CPU) yavaş ise, buffer kullanmak ve güncelleme (updating) tabloları yaratmak gibi çözümlerle sorun ortadan kalkar. Ancak CPU hızlı olsa bile giriş trafiği çıkış hatlarının kapasitesini aşarsa yine tıkanıklık olacaktır.

Tıkanıklığı aşmak için temel stratejiler aşağıda verilmiştir:

- Tıkanıklıktan kaçınmak için kaynakları verimli kullanmak,
- Haberleşme cihazlarına paketleri iptal etmek için izin vermek,
- Alt network'lerin paket sayısını kısıtlamak,
- Akış kontrolu yapmak,
- Tıkanıklık olduğunda network girişini kapatmak.

DATA LINK SEVİYESİ

Data Link ve Fiziksel seviye network'te haberleşmenin yapıldığı en alt seviyelerdir. Bilgisayarlar arasındaki haberleşmede haberleşmenin kurulması, bit dizilerinin anlamlı paketler haline getirilmesi ve çeşitli ortamlardan geçerek alıcıya ulaşması, alıcıda da hata saptama ve düzeltme işlemlerinin yapılması bu iki seviyede yapılır.

Hata kontrolu: Hata kontrolu iki şekilde yapılabilir: "**Echo Checking**" ve "**Automatic Repeat reQuest (ARQ)**". Echo checking normalde karakter-temelli protokollerde kullanılan bir yöntemdir. Terminaller lokal yada uzak modda çalışabilir. Lokal modda uzak bilgisayardan gelen karakterler aynı anda terminalin ekranında gösterilir. Uzak terminal modunda ise tuşa basıldığında karakter bilgisayara gider ve geriye gönderilir (echoing). Eğer geri gelen karakter gidene uymuyorsa kullanıcı bir transmisyon hatası olduğuna karar verip bilgisayara yanlış giden karakteri silmesi için **DEL**ete kontrol karakteri gönderir. Echo kontrolu görüldüğü gibi basit bir yöntemdir. Örneğin terminalin gönderdiği karakter doğru bile olsa echoing karakteri de terminale gelirken bozulabilir. Bundan başka her durumda karakter en az iki kere yollanacağından transmisyon band genişliği boşuna kullanımış olur.

Echo Checking yönteminde transmisyon hatalarının düzeltilmesi kullanıcıdan beklenir. Ancak kullanıcının devreye girmediği ve iki makinenin haberleştiği durumlarda bu işin bir program tarafından yapılması gerekir. Genelde onaylama (acknowledgement) için bir mesajın gönderilip asıl mesajın tekrarlanması yerine asıl mesajda gerekli değişikliği istemek daha uygundur. Bu tip bir onaylama yöntemine **ARQ (Automatic Repeat Request)** denir. ARQ'nun daha değişik versiyonları ise "**idle RQ (send-and-wait)**" ve "**Continuos RQ**"dur.

Akış Kontrolu: Akış kontrolu hatta iletilen çerçevelerin alıcı tarafa ulaşmasını kontrol eden bir işlemdir. Alıcı tarafın buffer kapasitesi aşıldığında giden paketlerin kaybolmaması için çeşitli akış kontrol yöntemleri geliştirilmiştir.

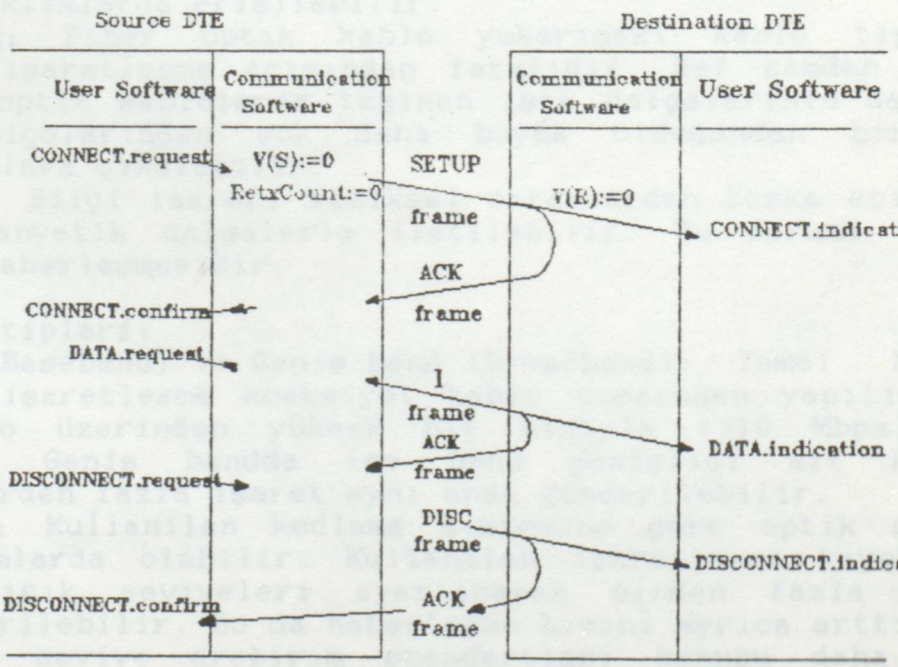
Bir ana bilgisayar ve terminallerden oluşan sistemlerde bilgisayara gelen istekler çok olduğunda bilgisayar paket göndermeyi kesmesi için terminallere X-OFF işareti gönderir. Terminal X-OFF işaretini aldığında klavyeden bilgi almayı ve göndermeyi keser ve bilgisayar tekrar X-ON işareti gönderene kadar bekler. Bu mekanizmaya **X-ON/X-OFF** protokolu denir.

Hat yönetimi: Haberleşme için hata ve akış kontrolundan başka hattın kontroluna da gerek vardır. Data haberleşmesi kurulmadan önce "Link Setup" denilen şartlandırma işlemi yapılır. Link kurulduktan sonra haberleşmeyi bitirmek için "**Link disconnection phase**" denilen işlem sırası yerine getirilir. Bu iki işlem arasındaki sürede datanın transferi ile ilgili olaylar "**Link management**"in içine girer.

Bir bilgisayarla terminaller arasında "**el sıkışma (handshake)**" denilen ve datanın iletilmesinde kullanılan ortak bir prosedür ve ayrı kontrol hatları kullanılır. Kullanıcı bilgisayarla haberleşeceği zaman terminalini açar, terminal bilgisayara hazır olduğunu belirten bir karakteri kontrol hattından gönderir ve bilgisayardan yanıt kodu bekler. Bu tip prosedürlere örnek olarak **RS-232C** protokolu verilebilir.

İki haberleşme sistemi paket haberleşmesi yapıyorsa link seviyesi protokollerinden "**control**" veya "**supervisory**" çerçevesi denilen özel kontrol paketleriyle hat bağlantısı kurulur.

Şekil.2.2.10'da kullanıcı yazılımının data link seviyesi aracılığıyla haberleşme prosedürü ve kullandığı mesajlar seti özetlenmektedir. Kullanıcı bağlantı isteğini (connection request) haberleşme yazılımına gönderir. Bundan sonra haberleşme yazılımı **RetxCount** ve **V(S)** durum değişkenlerini şartlar ve hat kurma isteğini bildiren çerçeveyi (SETUP.frame) alıcı DTE'ye gönderir. Alıcı DTE **V(R)** durum değişkenini şartlar ve bağlantı isteğini kullanıcı yazılımına gönderir. Aynı zamanda verici DTE'ye de onaylama paketi (ACK.frame) gönderilir. Böylece iki kullanıcı yazılımı arasında mantıksal haberleşme yolu kurulmuş olur. Daha sonra haberleşme yazılımı tarafından I-Frame'ler içinde data transferi yapılır. En son olarak, data transfer edildikten sonra mantıksal haberleşme yolu aynen kurulduğu biçimde kesilir (DISC.frame).



Şekil.2.2.10: Data Link Seviyesinde haberleşme kurma prosedürü.

FIZIKSEL SEVIYE

Fiziksel seviye OSI referans modelindeki en alt seviyedir. Bu seviyede çok değişik transmisyon ortamları ve protokollar kullanılabilir.Ayrıca transmisyon ortamlarına göre haberleşme şekilleri de baseband, broadband, mikrodalga, fiber optik gibi işaretleşme tipleri olabilir. Fiziksel seviyede data elektriksel sinyaller ile taşındığından dış etkilerden bu işaretlerdeki bozulmalar haberleşme performansına direkt olarak etki ederler.
Bu etkiler:

- Transmisyon ortamının tipine,
- Taşınan bilginin bit hızına,
- Iki haberleşme cihazının hızına bağlıdır.

TRANSMISYON ORTAMLARI:

Paralel Hatlar: Birbirine paralel iki kablodan oluşan bu ortam en basit transmisyon işlemlerinde kullanılır.Bu tip hatlar birbirine yakın (50 m'den küçük) ve düşük bit hızında (19 kbps) çalışan cihazlar arasında kullanılabilir. Bu hatlar iki kablo arasındaki kapasitif etkiden dolayı gürültüye karşı oldukça dayanıksızdır.
Twisted-Pair: Gürültünün etkisini azaltmak için paralel hatlar birbirinin üzerine burularak "twisted-pair" hatlar elde edilir. Twisted-pair hatlar uygun hat sürücüleri ve alıcı devreler ile küçük uzaklıklarda (<100 m) 1Mbps hıza kadar çıkabilir.Bazı twisted-pair kablolarda kabloların üzeri bir ekran ile kaplanır.Bu tip kablolara "**Shielded Twisted-Pair**" kablo denir.
Koaksiyel Kablo: Twisted-pair kablonun hızını sınırlayan temel özellik "**deri olayı**"dır. Bit hızı artınca telden akan elektrik akımı sadece telin dış yüzeyinden yol alır. Bu da yüksek frekanslarda hattın direncini arttırır ve sinyalin zayıflamasına neden olur.Bu nedenle 1Mbps'den büyük bit hızlarında "koaksiyel kablo" kullanılır.Kaoksiyel kabloda 10-20 Mbps hızlara birkaç yüz metrelik uzaklıklarda erişilebilir.
Fiber Optik: Fiber optik kablo yukarıdaki kablo tiplerinden malzeme ve işaretleşme açısından farklıdır. Saf camdan yapılmış olan fiber optik kablolarda taşınan ışık dalgalarının dalga boyu elektrik dalgalarından çok daha büyük olduğundan birkaç yüz Mbps'lik hızlara çıkılabilir.
Mikro Dalga: Bilgi işareti fiziksel ortamlardan başka açık havada da elektromanyetik dalgalarla iletilebilir. Bu duruma en tipik örnek uydu haberleşmesidir.

İşaretleşme tipleri:

Temel band (Baseband) ve **Geniş band** (Broadband): Temel band ve geniş band işaretleşme koaksiyel kablo üzerinden yapılır. Temel bandda kablo üzerinden yüksek bit hızıyla (>10 Mbps) iletim yapılabilir. Geniş bandda ise band genişliği alt kanallara bölünerek birden fazla işaret aynı anda gönderilebilir.
Fiber Optik: Kullanılan kodlama sistemine göre optik sinyaller değişik formlarda olabilir. Kullanılan işaretleşme kaynağı ışık olduğundan ışık seviyeleri ayarlanarak birden fazla seviyede sinyal gönderilebilir. Bu da haberleşme hızını ayrıca arttırır.

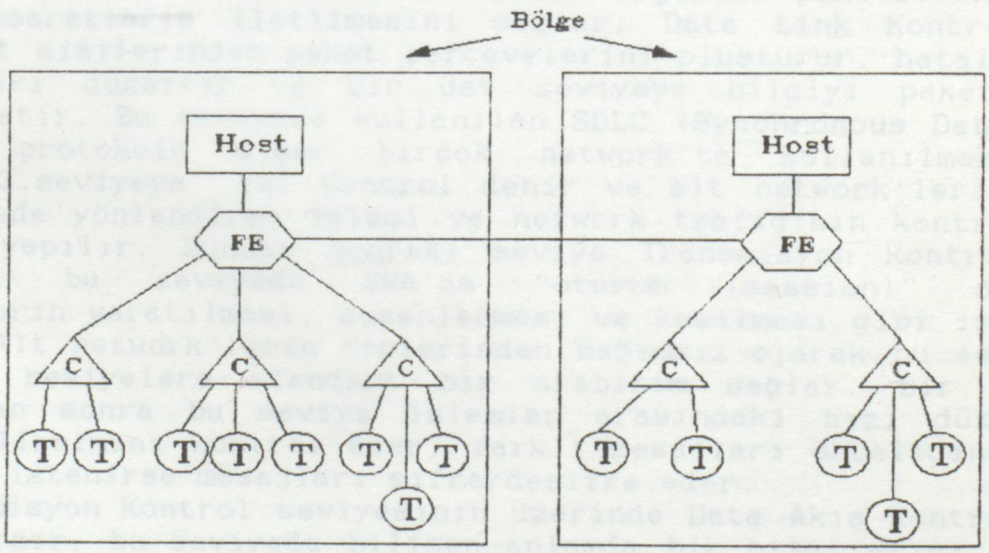
Fiziksel seviye arabirim standartları konusu daha sonraki bölümlerde ayrıntılı olarak inceleneceğinden burada bu konulara sadece tanıtım amacıyla yer verilmiştir.

2.3 SNA (Systems Network Architechture)

2.3.1 GİRİŞ

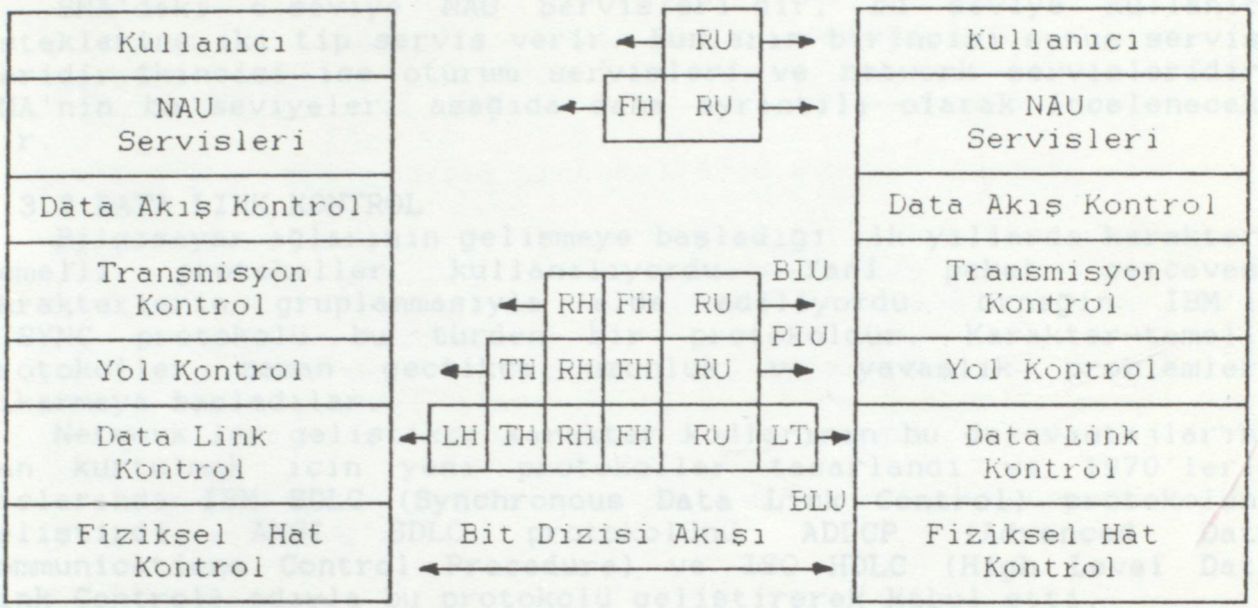
SNA, IBM firmasının kendi ürünlerinin haberleşmesini sağlamak amacıyla geliştirdiği bir network mimarisidir. SNA ilk olarak 1974 yılında geliştirildi. SNA'dan önce IBM'in ürettiği yüzlerce çeşit haberleşme cihazının kendi protokolleri vardı ve bu yüzden uyumsuzluk sorunları ortaya çıkıyordu. SNA'in geliştirilmesindeki temel amaç bu karmaşıklığın gidermek ve dağıtılmış işlem kavramına geçmekti. Bir SNA network'ü "**düğüm (node)**" denilen ve genel olarak 4 tipte bulunan cihazların birleşmesinden oluşur. 1.tipteki düğümler terminallerdir. 2.tipteki düğümler terminallerle diğer arabirimlerin kontrolunu yapan denetleyicilerdir. 4.tipteki düğümler "front-end" işlemcileridir ve data haberleşmesiyle ilgili interrupt kontrollarını yaparak ana bilgisayarın mikroislemcisini kontrol ederler. Her düğümün bir veya daha çok **NAU (Network Addressable Unit)** yazılımı vardır ve düğüm NAU sayesinde network'e erişebilir. Her NAU bir network adresi alır ve network'ü kullanmak için işlemci önce kendi NAU'suna bağlanır, bu NAU da bir başka NAU ile haberleşir. Yani NAU kullanıcı işlemleri için network'e giriş kanalıdır.

NAU'lar üç çeşittir. **LU (Logical Unit)**, kullanıcı işlemleriyle haberleşir. **PU (Physical Unit)**, her düğüm için özeldir ve düğümü on/off-line hale getirir, testler yapar ve düzenleyici işlemler yapar. En son tip NAU olan **SSCP (Systems Services Control Point)** bütün front-end işlemcilerini, denetleyicileri, ve host bilgisayara bağlanan terminalleri kontrol eder. SSCP tarafından düzenlenen yazılım ve donanıma "domain" denir. **Şekil.2.3.1**'de bir SNA ağındaki iki basit domain gösterilmiştir.



Şekil.2.3.1: SNA Network'ünde iki domain.

FE: Front End
C: Controller
T: Terminal



LH:Link Header
LT:Link Trailer
TH:Transmission Header
RH:Request/Response Header
BLU:Basic Link Unit (=frame)
PIU:Path Information Unit (=packet)
BIU:Basic Information Unit (=message)
RU :Request/Response Unit

Şekil.2.3.2: SNA'de protokol hiyerarşisi.

Şekil.2.3.2'de SNA'in protokol hiyerarşisi gösterilmiştir. Mimarinin en alt seviyesi bir cihazdan diğerine paketlerin elektriksel işaretlerle iletilmesini sağlar. Data Link Kontrol seviyesi bit dizilerinden paket çerçevelerini oluşturur, hatalı alınan bitleri düzeltir ve bir üst seviyeye bilgiyi paketlenmiş olarak iletir. Bu seviyede kullanılan SDLC (Synchronous Data Link Control) protokolü diğer birçok network'te kullanılmaktadır. SNA'daki 3.seviyeye Yol Control denir ve alt network'lerle olan haberleşmede yönlendirme işlemi ve network trafiğinin kontrolu bu seviyede yapılır. Bundan sonraki seviye Transmisyon Kontrol seviyesidir, bu seviyede SNA'da "oturum (session)" denilen bağlantıların yaratılması, düzenlenmesi ve kesilmesi gibi işlemler yapılır. Alt network'lerin tiplerinden bağımsız olarak bu seviyede daha üst seviyelere standart bir arabirim sağlar. Bir oturum kurulduktan sonra bu seviye işlemler arasındaki hızı düzenler, buffer kullanımını kontrol eder, farklı mesajları önceliğine göre yollar ve istenirse mesajları şifre/deşifre eder.

Transmisyon Kontrol seviyesinin üzerinde Data Akış Kontrol seviyesi vardır, bu seviyede bilinen anlamda bir bilgi akışı yapılmaz, onun yerine oturumların başlangıç ve bitişlerini izlenir ve hataların giderilmesi yapılır.

SNA'deki 6.seviye NAU Servisleri'dir, bu seviye kullanıcı isteklerine iki tip servis verir. Bunların birincisi sunuş servisleridir.İkincisi ise oturum servisleri ve network servisleridir. SNA'nin bu seviyeleri aşağıda daha ayrıntılı olarak incelenecektir.

2.3.2 DATA LINK KONTROL

Bilgisayar ağlarının gelişmeye başladığı ilk yıllarda karakter-temelli protokoller kullanılıyordu. Yani paket çerçevesi karakterlerin gruplanmasıyla elde ediliyordu. Örneğin IBM'in BISYNC protokolü bu türden bir protokoldür. Karakter-temelli protokoller zaman geçtikçe uzunluk ve yavaşlık problemleri çıkarmaya başladılar.

Network'ler geliştikçe karakter kodlarının bu dezavantajlarından kurtulmak için yeni protokoller tasarlandı ve 1970'lerin başlarında IBM SDLC (Synchronous Data Link Control) protokolünü geliştirdi. ANSI SDLC protokolünü ADDCP (Advanced Data Communications Control Procedure) ve ISO HDLC (High Level Data Link Control) adıyla bu protokolü geliştirerek kabul etti.

Bütün bu protokoller "bit-temelli (bit-oriented)" olarak tasarlandı. Bu protokollerin bit-temelli olarak tanımlanmasının nedeni çerçevelerin tamamen bit dizilerinden oluşması ve çerçevenin uzunluğunun değişken olabilmesidir.

Bu protokollerde datanın korunması için kullanılan yöntem "**Bit-Stuffing**" adıyla bilinir. Her çerçeve "01111110" şeklinde bir bit dizisiyle başlayıp biter. Verici taraftaki donanım bit dizisini gönderirken her beş bitte bir araya 0 ekler. Alıcı tarafta ise birbirini takip eden beş tane 1 ve bir 0 alınınca 0 biti otomatik olarak silinir. Kullanıcı datası "01111110" ise "011111010" olarak gönderilir, fakat alıcı tarafta tekrar "01111110"a dönüştürülüp belleğe konur. Şekil.2.3.3'te buna bir örnek verilmiştir.

(a) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0
(b) 0 1 1 0 1 1 1 1 1 **0** 1 1 1 1 1 **0** 1 1 1 1 1 **0** 1 0 0 1 0
(c) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

Şekil.2.3.3: Bit Stuffing. **(a)** Original data.
(b) Hatta iletilen data.
(c) Alınıp düzeltilen data

Bit temelli protokollerin çerçeve yapısı **Şekil.2.3.4**'teki gibidir. Address alanı terminallerin adreslerini belirtmek için kullanılır. Eğer uçtan-uca (point-to-point) haberleşme varsa bu alana bazı komutlar yerleştirilir. Control alanı sıra numarası, onaylama (acknowledgement) ve diğer amaçlar için kullanılır. Data alanı bilgi taşıma alanıdır ve diğer alanlara göre en uzun alandır. Bu alanın uzunluğunun seçimi iletişim hattının kalitesi nedeniyle oluşabilecek hataların olasılığı ile orantılıdır. Checksum alanı CRC-CCITT polinomu kullanarak CRC-(Cyclic Redundancy Code) ile kaybolan flag byte'larını yakalamak içindir.

Bits	8	8	8	>0	16	8
	01111110	Address	Control	Data	Checksum	01111110

Şekil.2.3.4: Bit-temelli protokollerin çerçeve formatı.

Çerçeve flag denilen bir bir bit dizisi (01111110) ile sınırlandırılır. Uçtan uca bağlı olan hatlarda flag dizileri ardışıl olarak iletilir. Minimum çerçeve (frame) uçlardaki flaglar hariç üç alan ve 32 bitten oluşur.

Üç tip çerçeve vardır: **Information, Supervisory** ve **Unnumbered**. Bu üç tip için Control alanı Şekil.2.3.5'te gösterilmiştir. Protokol üç bitlik dizi numarasıyla kayan-pencere (sliding-window) kullanır.

Şekil.2.3.5(a)'daki **Seq** alanı çerçeve dizi numarasıdır. **Next** alanı ise onaylama alanıdır.

Bits	1	3	1	3
(a)	0	Seq	P/F	Next
(b)	1	0	Type	P/F	Next
(c)	1	1	Type	P/F	Modifier

Şekil.2.3.5: Control alanı. (a) Information frame
(b) Supervisory frame
(c) Unnumbered frame.

P/F biti Poll/Final'ın kısaltmasıdır. Bir bilgisayar terminal grubuyla haberleşeceği zaman P/F bitini "1" yapar ve bir "poll" mesajı gönderir. Terminal de buna aynı çerçeve ile yanıt verir. Daha sonra bilgisayar data çerçevelerini gönderir.Son data çerçevesinin P/F biti 1 yapılarak bunun son çerçeve olduğu belirtilir. Bazı protokollerde P/F biti yanıt beklemeden diğer makineleri Supervisory çerçevesi göndermeye zorlar. Bu bit biraz değişiklikle Unnumbered çerçevede de kullanılır. Supervisory çerçevesinin Type alanına göre çeşitli şekilleri vardır:

- **Type 0** (RECEIVE READY), bir bir sonraki çerçevenin beklendiğini belirten bir onaylama çerçevesidir.
- **Type 1** (REJECT), transmisyon hatasının bulunduğunu belirten negatif bir onaylama çerçevesidir. Next alanı dizide doğru alınmayan ilk çerçeveyi gösterir. Verici Next alanında gösterilen çerçeveden itibaren datayı tekrar göndermelidir.
- **Type 2** (RECEIVE NOT READY), bütün çerçevelerin doğru alındığını onaylar, fakat vericiye göndermeyi kesmesini iletir. Yani alıcıda geçici haberleşme hatalarının olduğunu bildirir ve durum düzelince tekrar RECEIVE READY gönderir

- **Type 3** (SELECTIVE REJECT), sadece istenen çerçevenin yeniden transmisyonunu ister. Üçüncü tip çerçeve ise Unnumbered Frame'dir ve kontrol amacıyla kullanılır. Burada çeşitli tiplerdeki bit-temelli protokoller farklılıklar gösterir. Çerçeve tipi beş bitlik bir sayıyla belirlenir, fakat 32 olasılık geçerli değildir.

Bütün bu protokoller **DISC** (DISConnect) komutuna sahiptir ve komut sayesinde terminal haberleşme ağına haberleşmeyi keseceğini bildirir. Aynı komutla dizi numarası sıfırlanarak tekrar haberleşme kurulur. Bu komuta da **SNRM** (Set Normal Response Mode) denir. Üçüncü komut olan **FRMR** (FRaMe Reject) ise çerçevenin kontrol toplamının (CheckSum) doğru olduğunu, fakat mümkün olmayan bir semantik ile alındığını bildirir.
Kontrol çerçeveleri de data çerçeveleri gibi bozulabilir veya kaybolabilir, çerçevenin onaylanmadığını bildirmek için **UA** (Unnumbered Acknowledgement) denilen özel bir kontrol çerçevesi kullanılır. Diğer kontrol çerçeveleri de şartlandırma (initialization), toplama (polling) ve durum raporu vermekle ilgilidir. Burada UI (Unnumbered Information) denilen özel bir kontrol çerçevesi de vardır.

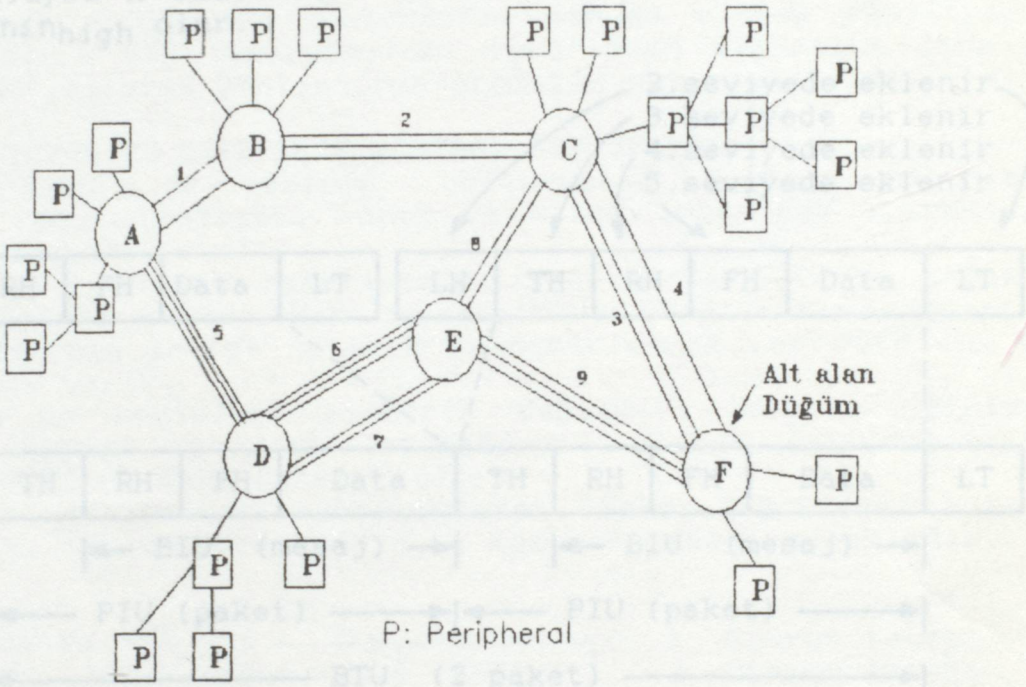
2.3.3 YOL (PATH) CONTROL

SNA da Yol Control seviyesi transmisyon kontrol seviyesine virtüel devre servisi verir. Bu seviyede oturum ve routing (yönlendirme) işlemleri yapılır.

Yönlendirmenin amacına göre SNA network'ü bir veya daha fazla NAU içeren "**alt alanlar (subarea)**"a bölünür. Her NAU'nun iki port adresi vardır: kendi alt alanı, ve alt alanın içinde kendi adresi. Her alt alan tip 4 ve tip 5 düğümünün kontrolu altındadır. **Şekil.2.3.6**'de görüldüğü gibi backbone oluşturan alt alanlara type 1 ve type 2 düğümleri bağlanır.

Oturum kurulduktan sonra kaynak düğümü virtüel bir rota seçer. Virtüel rota kaynak düğümünden alıcı düğüme olan alt alanların listesidir. Şekil.2.3.6'da ABCD, ADEF, ve ADECF, A alt alanından F alt alanına olan virtuel rotalardır. Bir virtüel rotadan diğerini seçmek için iki bilgiye ihtiyaç vardır: Çeşitli virtüel rotalara atanan o anki yük ve oturum için istenen servis sınıfıdır. Olası servis sınıfları, RJE (Remote Job Entry), file transferi, güvenlik gibi interaktif servislerdir. Bu 3 servis sınıfındaki 24 olasılıktan en fazla 8 alternatif virtuel rota seçilebilir. Aynı virtüel rotayı birbiriyle ilişkisiz birden fazla oturum kullanabilir. Alt alanlar arasında birden fazla hat olduğundan virtüel rota kavramı gereklidir. Bu hatlara "transmisyon grubu" denir. Örneğin **Şekil.2.3.6**'da D ile E arasında 5 hat vardır ve iki transmisyon grubuna bölünmüştür. Transmisyon grupları genelde homojen hatlar içerir. Her transmisyon grubunun tek bir "transmisyon kuyruğu (queue)" vardır. Birden fazla hat kullanılmasının nedeni band genişliğinin tek hattan daha fazla olmasıdır. Oturum kurmanın bir parçası olarak virtüel rota kaynak transmisyon grubunun dizisi olan "açık rotaya (explicit route)" atanır.

Örneğin Şekil.2.3.1'de 5683, 5783, 5783, 5684 ve 5784, ADECF virtüel rotası için olası açık rotalardır. Bir paket alt alana geldiğinde yönlendirme işlemi varış ve açık rota numarasını bulur ve kendi yönlendirme tablosuna baktıktan sonra paketi transmisyon grubuna koyar.

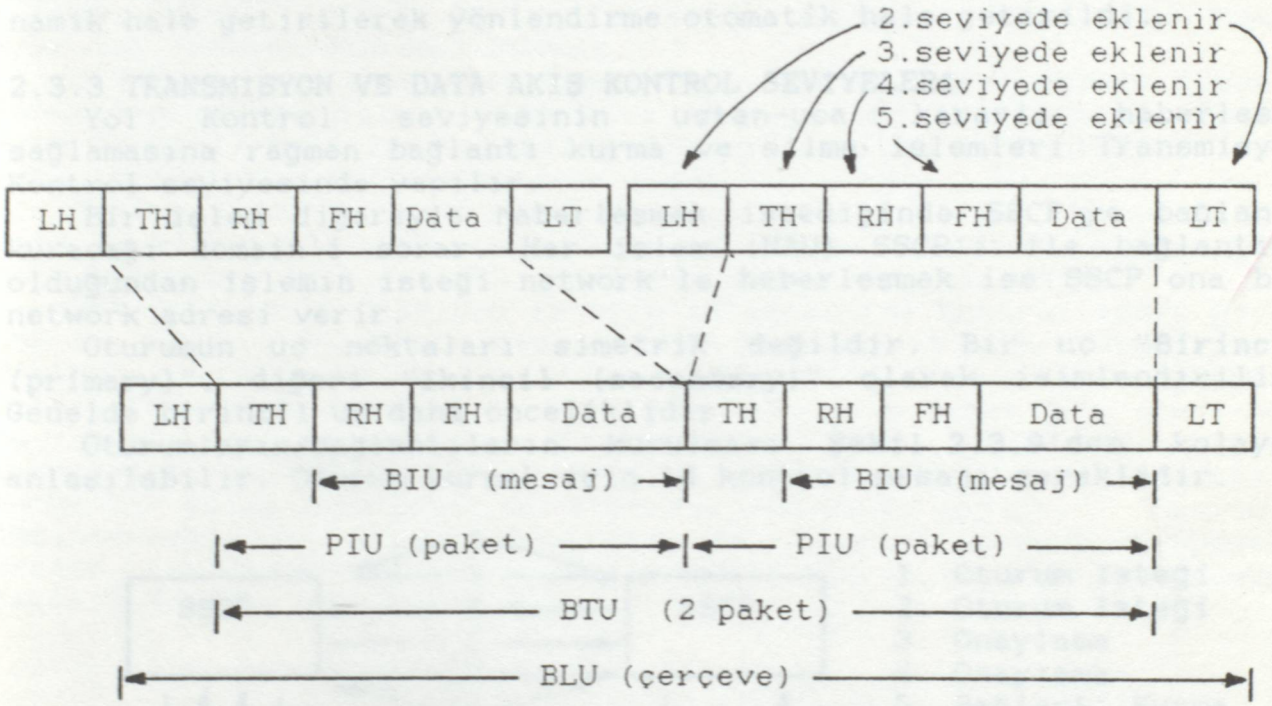


Şekil.2.3.6: Örnek bir SNA Network.

Eğer transmisyon grubundaki bütün hatlar arızalanırsa, grup tarafından kullanılan açık rotalar virtüel rota tarafından seçilen bir başka açık rotaya yönlendirilir. Eğer hiçbiri bulunamazsa bir başka virtüel rota seçilmelidir. Eğer hiçbir virtüel rota uygun değilse oturum iptal edilir ve kontrol paketleri network'e gönderilerek hangi alt alanın haberleşemediği belirtilir.

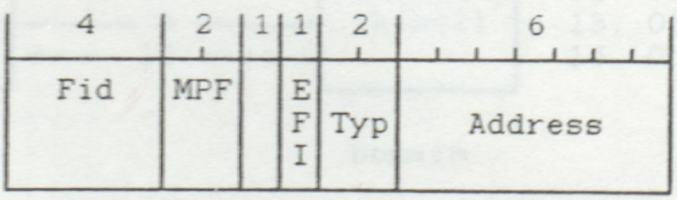
Transmisyon verimliliğini arttırmak için ilişkisiz paketler aynı transmisyon grubunda sıraya sokulurak Şekil.2.3.7'deki gibi "**temel transmisyon birimi**" oluşturulur. Çerçeve bir sonraki alt alana ulaştığı zaman bölünür ve bütün paketler transmisyon gruplarında sıraya girer. Bu özellik genelde ana bilgisayar (tip 5) ve front-end denetleyiciler (tip 4) arasında CPU interrupt'larını en aza indirmek için kullanılır. Alıcı taraftan gönderilen ve göndericiye hazır olduğunu belirten pakette n tane paketlik buffer olduğu belirtilir, bunun üzerine verici taraf n paketlik çerçeveyi gönderir. Alıcıda yeterli buffer varsa vericiye tekrar paketi göndermesi için yetki verir. Bu mekanizmaya "**pacing-tempoyu ayarlama**" denir, eğer alıcının **2n-1**'lik bufferi varsa ve sürekli yetki verirse sürekli data akışı sağlanır.

Trafik kontrolu pacing parametresi n ayarlanarak dinamik olarak yapılır. Her virtüel rotanın minimum uzunluğu n_{low} ve maximum uzunluğu n_{high}'dır. Genelde maximum pencere uzunluğu nlow'un birkaç katıdır ($n_{high} = 3 \times n_{low}$). Her paket kendi varış yerine vardığında paket header bitleri trafik yoğunluğunu gösterir. Yoğunluk fazlaysa n azaltılır, az ise arttırılır. Sonuç olarak n değeri $n_{low} \leq n \leq n_{high}$ olur.



Şekil.2.3.7: Iki paketin eklenerek bloklanması.

Yol Kontrolu seviyesi birkaç farklı TH formatı kullanır. Kısaltılmış header'lar kaynak ve varış alanlarında kullanılır. Terminal ve denetleyiciler arasında kullanılan basit bir header Şekil.2.3.8'de gösterilmiştir.



Şekil.2.3.8: Bir alt alanda kullanılan Transmisyon Header'ı.

Fid alanı,header tipini gösterir (Burada 0011).
MPF (MaPping Field), atama (mapping) alanıdır ve PIU veya BIU'nun parçalarından birini veya tümünün olduğunu belirtir.

EFI biti data akışının güvenli (expediated) olup olmadığını bildirir.

Type alanı oturum tipini gösterir: LU-LU, LU-SSCP, PU-LU veya PU-SSCP.

Address alanı alt alanın elemanını tanımlar.

SNA'daki yönlendirme algoritmaları ilk tasarlandığı şekliyle tamamen statik olduğundan yönlendirme manual olarak yapılıyordu, ancak daha sonra SNA tasarımcıları tarafından algoritma daha dinamik hale getirilerek yönlendirme otomatik hale getirildi.

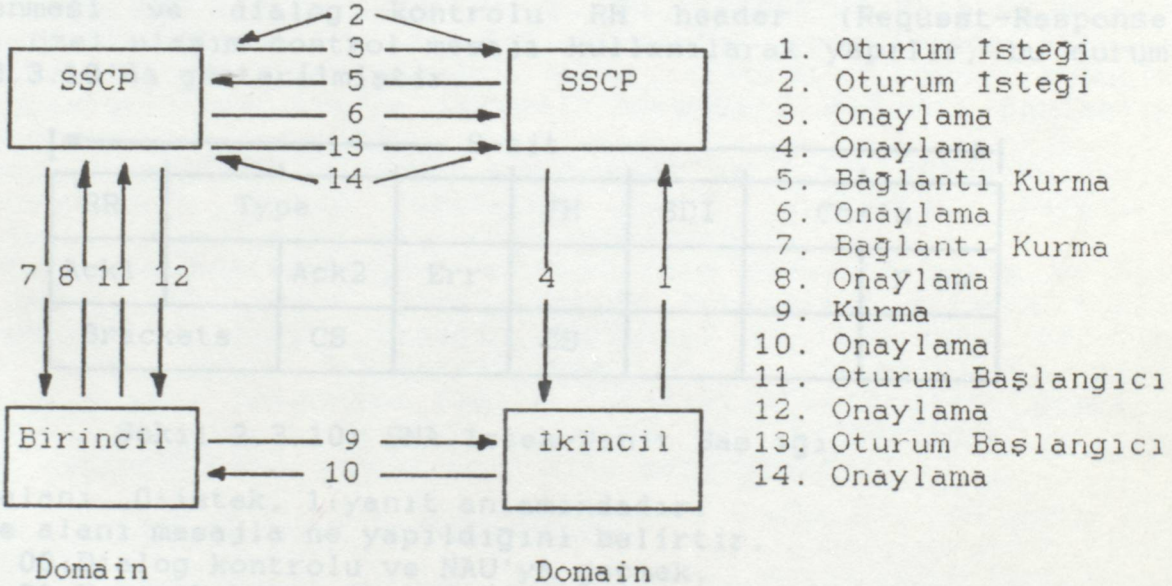
2.3.3 TRANSMISYON VE DATA AKIŞ KONTROL SEVIYELERI

Yol Kontrol seviyesinin uçtan-uca kararlı haberleşme sağlamasına rağmen bağlantı kurma ve silme işlemleri Transmisyon Kontrol seviyesinde yapılır.

Bir işlem diğeriyle haberleşmek istediğinde SSCP'ye bağlantı kuracağı domain'i sorar. Her işlem (NAU) SSCP'i ile bağlantıda olduğundan işlemin isteği network'le haberleşmek ise SSCP ona bir network adresi verir.

Oturumun uç noktaları simetrik değildir. Bir uç "**Birincil (primary)**", diğeri "**Ikincil (secondary)**" olarak isimlendirilir. Genelde birincil uç daha önceliklidir.

Oturumların/bağlantıların kurulması **Şekil.2.3.9**'den kolayca anlaşılabilir. Oturum kurmak için 14 kontrol mesajı gereklidir.



Şekil.2.3.9: İkincil'den istek geldiğinde oturum kurulması.

Önce ikincil kendi SSCP'sine oturum istediğini bildirir. SSCP de diğer taraftaki SSCP'ye kullanıcılarından birinin oturum kurmak istediğini belirtir ve kullanıcının şifresiyle yetkisini verir (mesaj 2). Eğer diğer SSCP oturum kuracaksa ve koşullar sağlanıyorsa SSCP'nin hazırlanmasını bildirir (mesaj 3). Onaylama alındıktan sonra ikincil'nin SSCP'si "bind" isteğini birincil SSCP'ye gönderir.

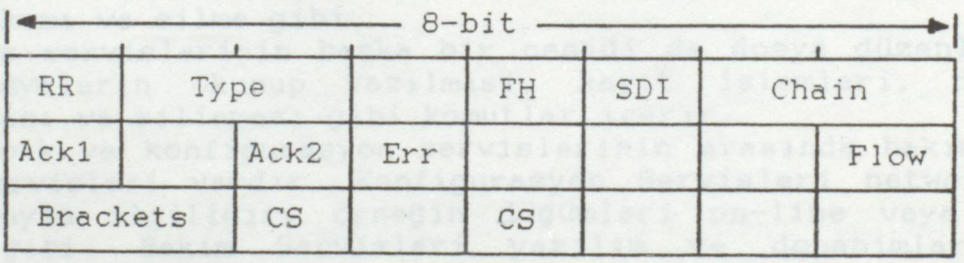
Binding (Bağlanma), SNA'nın önemli operasyonlarından biridir. Bind komutu (mesaj 9), herzaman birincil'den ikincil'e gönderilir, kullanılan protokoller akış kontrolu parametreleri gibi parametreleri içerir ve birkaç düzine byte'dan oluşur. Binding tamamlanınca birincil kendi SSCP'sine bağlantı kurulduğunu belirtir, bu SSCP'de diğerine aynı bilgiyi iletir. Bind mesajı gibi data mesajları da SSCP'yi atlayarak kaynaktan varışa gider.

Transmisyon kontrol seviyesinde kullanılan akış kontrol algoritması path kontrol seviyesindekinin aynıdır, ancak iki host arasında tamamen oturum kurulmaz.

Oturumda data akış hızı transmisyon kontrol seviyesinde düzenlenmesine rağmen akışın yönü data akış kontrol seviyesinde (Data Flow Control) belirlenir. Dialog Control denilen bu fonksiyon, OSI modelinde oturum seviyesine denk düşer.

Dialog Control, mesajları "**zincir (chain)**"lere gruplayarak hata düzeltme gibi işlemleri yapar. Zincirin ortasında oluşan bir hata bütün zincirin yeniden gönderilmesine neden olacaktır.

SNA isteklerin onaylama isteyip istemediğini kontrol eder. Kaynak ve alıcı onaylamanın semantiğini (=anlamını) kabul etmelidir. Dialog Control onaylamanın kullanıcıya istekle aynı anda gönderildiğinden emin olmalıdır. Dialog kontrolunun diğer bir özelliği de istek ve yanıtların "bracket" denilen birimlerde gruplanmasıdır. Eğer her oturum karmaşık işlemler yapacaksa "bracket" kurma bir işlem tamamlanmadan diğerine geçmemeyi sağlar. Oturum düzenlenmesi ve dialog kontrolu RH header (Request-Response Header) özel ulaşım control mesajı kullanılarak yapılır, bu durum **Şekil.2.3.10**'da gösterilmiştir.



Şekil.2.3.10: SNA İstek-Yanıt Başlığı.

RR alanı, 0:istek, 1:yanıt anlamındadır.
Type alanı mesajla ne yapıldığını belirtir.
00:Dialog kontrolu ve NAU'ya geçmek.
01:Network kontrol mesajı.
10:Normal işlem için dialog kontroluna geçmek.
11:Oturumla ilgili kontrol mesajı.
FH alanı, FH header'in data bölümü olup olmadığını belirtir. **SDI** alanı 4-byte "Sense data" alanının mesajda olup olmadığını belirtir. "Sense data" hata raporu için kullanılır.
Ack1, **Ack2** ve **Err** alanları, istenen onaylanmanın belirlenmesi ve hatayı verir. Bu alanların tam anlamı daha yüksek seviyelerdeki yazılımda belirlenir.

Flow alanı, vericiye yeni bir n mesajlık grubun istenip istenmediğini bildirmekte kullanılır.

Brackets alanı, bracket yapısını sınırlar, zincir alanına benzer.

Dialog kontrolu yarı-duplex hattı simule etmekte kullanılacaksa **CD** (Change directory) alanı yön değiştirmekte kullanılır.

Son olarak oturum sınırlandığında ikinci ucun aynı karakter kodunu kullanması için **CS** (Code Select) alanı kullanılır.

2.3.4 NAU (Network Addressable Unit)

SNA'da 6.seviye NAU servisleridir, bunlar yaklaşık üç farklı servisler verirler. Bunlar "sunuş (presentation)", "oturum (session)" ve network servisleridir. Presentation servislerinde file transfer protokolüyle dönüştürmeler yapılır. Oturum servisleri oturum kurmaya yarar. Network servisleri de network'ün çalışması için gerekli olan ileri düzeyde servisleri içerir.

Oturum sınırlı olduğu zaman **BIND** mesajının parametreleri hangi sunuş servislerinin olduğunu ve oturumda kullanacağını tanımlar. Sunuş servisleri kullanıldığı zaman, sunuş servisi protokolü FH header'ında saklanır. NAU servislerine örnek verilecek olursa data sıkıştırma (compression=karakter dizilerini karşı düşen tek bir karakterle iletme), bütünleştirme (compaction=birden fazla karakteri tek bir byte'a sıkıştırma) sayılabilir.

SNA virtüel terminal protokolüne benzer bir özelliğe sahiptir. Bu özelliğe **3270E** data akışı denir. Bu protokolde data ve kontrol karakterleri karışıktır, display bilgisini tanımlamada terminal bağımsızdır. Bu protokolde alan seçimi için komutlar vardır,gösterilecek alanın korumalı/korumasız olduğu, parlaklık, yazma, okuma ve silme gibi.

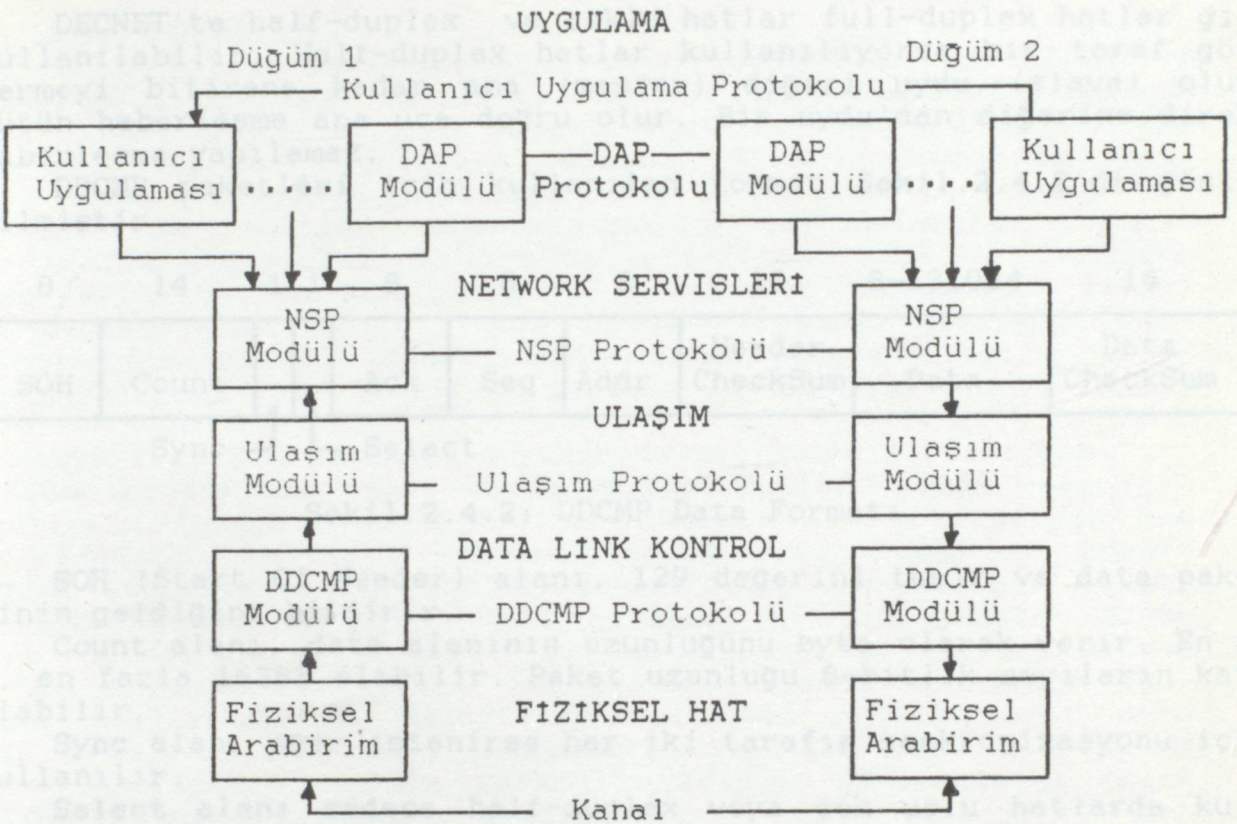
Sunuş servislerinin başka bir çeşidi de dosya düzenlemesidir. Uzak dosyaların okunup yazılması, kayıt işlemleri, dosyaların yaratılması ve silinmesi gibi komutlar içerir.

Network ve konfigurasyon servislerinin arasında bakım ve operatör servisleri vardır. Konfigurasyon Servisleri network konfigurasyonuyla ilgilidir, örneğin düğümleri on-line veya off-line yapmak gibi. Bakım Servisleri yazılım ve donanımların diagnostiğini yapar, networkle ilgili istatistikler toplar. Operator Servisleri operatörle network arasında bir arabirimdir ve operatörün network'ü kullanması için olan servislerdir.

2.4 DNA (Digital Network Architechture)

2.4.1 GİRİŞ

DECNET, DEC-Digital Equipment Corp. tarafından kendi bilgisayar sistemleri için tasarlanmış bir protokoller setidir. DECNET'in protokolleri DNA (Digital Network Architecture) mimarisi üzerine kurulur. DECNET'in amacı DEC'in kullanıcılarının kendi özel ağlarını kurmasına olanak sağlamaktır.



Şekil.2.4.1: DNA Protokol Mimarisi.

Şekil.2.4.1'den görüleceği gibi DECNET'te beş seviye vardır. Bunlar Fiziksel Seviye, Data Link Control Seviyesi, Ulaşım Seviyesi, Network Servisleri Seviyesi ve Uygulama Seviyesi'dir. Bu seviyeler aşağıda kapsamlı olarak incelenecektir.

2.4.2 FIZIKSEL VE DATA LINK SEVIYESI

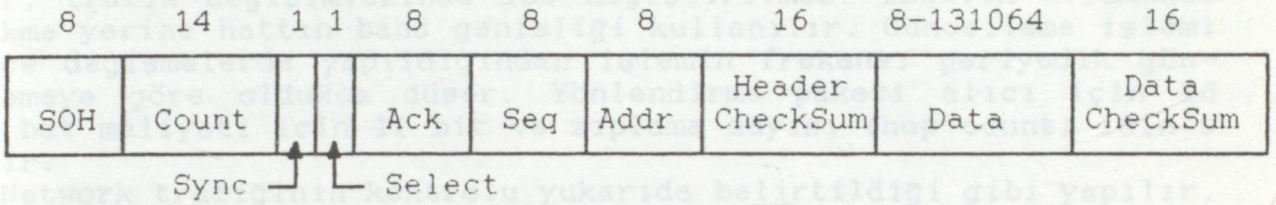
DECNET'te fiziksel seviye data kanalından taşınan fiziksel bilgiyi düzenler. Bu durum transmisyon ortamının özelliklerine, işaretleşme tekniklerine ve bilgisayar sistemi ile haberleşme servislerine bağlıdır. Fiziksel seviye işlemleri ya işletim sistemi içinde bir cihaz sürücüsü (device driver) olarak, ya da bir front-end işlemcisi içinde yapılır. Böylece kanaldan bağımsız bir arabirim elde edilmiş olur.

DECNET'te data link protokolüne **DDCMP (Digital Data Communications Message Protokol)** denir. DDCMP paketleri SDLC'de olduğu gibi bit patternleri ile sınırlanmamıştır, fakat header'de bir karakter sayısı saklanır. Eğer bu sayı transmisyon hataları nedeniyle bozulursa protokolün senkronizasyonu bozulduğundan çerçeve sınırları bozulur. Bu hatayı minimize etmek için çerçeve başlığının kendi kontrol toplamı (checksum) vardır.

DDCMP kayan pencere (sliding window) protokolü kullanır, en fazla 255 paket bir çerçeveye sığabilir. Maximum paket genişliği 16383 byte'dır.

DECNET'te half-duplex ve çoklu hatlar full-duplex hatlar gibi kullanılabilir. Half-duplex hatlar kullanılıyorsa bir taraf göndermeyi bitirene kadar ana (master) diğeri uydu (slave) olur. Bütün haberleşme ana uca doğru olur. Bir uydu'dan diğerine direkt haberleşme yapılamaz.

DDCMP paketleri için kullanılan format **Şekil.2.4.2**'de gösterilmiştir.



Şekil.2.4.2: DDCMP Data Formatı.

SOH (Start Of Header) alanı, 129 değerini taşır ve data paketinin geldiğini bildirir.

Count alanı, data alanının uzunluğunu byte olarak verir. En az 1, en fazla 16383 olabilir. Paket uzunluğu 8-bitlik sayıların katı olabilir.

Sync alanı eğer istenirse her iki tarafın senkronizasyonu için kullanılır.

Select alanı sadece half-duplex veya çok uçlu hatlarda kullanılır. Verici artık data göndermeyeceği zaman bu bit set olur. Half-duplex hatlarda bu bilgi diğer tarafın data göndermesine izin verir. Çok uçlu hatlarda bu bitle kontrol ana veya uyduya verilir.

Addr alanı çok uçlu hatlarda paketin adresleneceği terminalin adresini gösterir.

Header Checksum alanı, CRC-16 polinomu kullanarak header dosyalarında "**CRC-Cyclic Redundancy Checksum**" kontrolu yapar. Yukarıda bahsedildiği gibi datanın uzunluğunu saklayan Count alanı için header için ayrı kontrol toplamları gereklidir. Eğer binlerce byte'lık bir datanın içinde hatalı bir pakete raslanırsa alıcı bütün datayı yeniden almak zorundadır. Verici ara verip ya datayı tekrarlayacak, ya da doğrulama bilgisi gönderecektir. Birçok denemeden sonra alıcı bilgi beklerken verici alıcının bozulduğunu kabul edip data akışını kesecektir. Bu nedenle fazla büyük data bloklarının bu protokolle gönderilmesi sakıncalıdır.

Data alanı, bilgiyi içinde tutar. Datanın uzunluğu Count alanında saklandığından "byte stuffing" işlemine gerek yoktur.

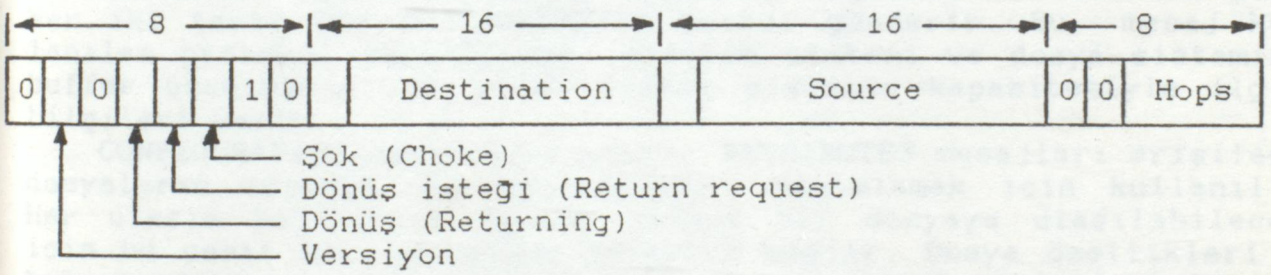
Data Checksum alanı, paketin data bölümünün kontrol toplamını saklar. Data paketlerinden başka kontrol paketleri de vardır. Bunların en önemlileri **ACK**, **NAK**, **REP**, **START** ve **STARTACK**'tır. ACK (Acknowledge), tersine bir trafik yoksa piggback tipi bir onaylama yapılmayacağından kullanılır. NAK (Negative Ack) paketin doğru alınmadığını bildirir. REP (Reply to Message Number) paketi vericinin mola (timeout) verdiğini bildirir. START ve STARTACK paketleri alıcı/verici çiftini hazırlamakta kullanılır.

2.4.3. ULAŞIM SEVİYESİ

DECNET'te ulaşım seviyesi datagram servisleri içerir. DECNET'in yönlendirme algoritması orijinal ARPANET algoritmasının tamamen aynısıdır. Her düğüm alıcıya endekslenmiş iki vektörü izler. Bir vektör alıcının yol uzunluğunu, diğeri ise hesaplanmış maliyeti saklar. Paket geldiğinde en az maliyetli ve en kısa yol seçilir.

Yönlendirme tabloları sadece topoloji değişmelerinde güncellenir, trafik değişimlerinde ise değiştirilmez. uzaklık ölçümünde gecikme yerine hattın band genişliği kullanılır. Güncelleme işlemi sadece değişmelerde yapıldığından işlemin frekansı periyodik güncellemeye göre oldukça düşer. Yönlendirme paketi alıcı için 16 bit, hat maliyeti için 11 bit ve zıplama sayısı (hop count) için 5 bittir.

Network trafiğinin kontrolu yukarıda belirtildiği gibi yapılır. Her düğüm iki vektörü izler ve günceller. Bunlar "**kredi (credit)**" ve "**kullanım (usage)**" vektörleridir. Kredi vektörü trafiğin kaç birim olduğunu, kullanım vektörü ise bunlardan kaçının gönderildiğini gösterir.Trafik birimleri sabittir, buna karşın paketlerin değeri değişir. Eğer kredi aşılırsa alıcıya gidecek bütün paketler iptal edilir. Trafiğin ne zaman fazla geleceğini bilmek için her düğümün gösterdiği kuyruk uzunluğuna ve hattın kullanımına bakılır ve bu değerlerin ağırlıklı ortalamasının daha önceki ortalamalarla karşılaştırması yapılır. Eğer ortalama aşılırsa "**şok paketi (choke packet)**" vericiye gönderilerek kredi ve kullanım değerleri yeniden ayarlanır.



Şekil.2.4.3: DECNET Transport Layer header.

Data paketleri için network header'ı **Şekil.2.4.3**'te gösterilmiştir. Versiyon biti paketin üretildiği protokolün veriyonunu verir. Dönüş İsteği biti vericiye bir paketin geri dönmesi gerektiğini bildirir. Paket geri geldiğinde Dönüş (Returning) biti 1 olur. Şok biti paketin sıradan bir paket olmayıp şok paketi olduğunu gösterir. Destination ve Source alanları paketin ağ üzerinde nereden gelip nereye gittiğini gösterir. Son olarak Hops sayıcısı eski paketleri silmekte kullanılır.

2.4.4 NETWORK SERVİSLERİ VE UYGULAMA SEVİYESİ

Network Servisleri Seviyesi kullanıcı arabirimi ile mantıksal haberleşme yolları kurar. İki işlem arasında mantıksal bağlantı kurulduktan sonra herikisi de aynı anda mesaj alışverişinde bulunurlar. Network Servisleri Seviyesi mesajları ardışıl olarak işleme koyar ve bu esnada mantıksal hattı kesmez. İletişimdeki herhangi bir değişme, örneğin kanalın veya düğümün bozulması gibi durumlarda bunlar kullanıcıya yansır.

Network servisleri seviyesi fonksiyonlarını **NSP** (Network Servisleri Protokolü) ile yerine getirir. NSP ulaşım seviyesinde sağlanan haberleşme servislerini kullanarak güvenli bir haberleşme sağlar. NSP'nin kullandığı teknikler DDCMP protokolüne çok benzer. NSP'de bulunmayan tek fonksiyon DDCMP'deki CRC kontroludur, çünkü bu kontrol daha alt seviyelerde zaten yapılmıştır.

DECNET bilinen anlamda uygulama seviyesi içermez, buna karşın kullanıcı programlarına dosyalara erişme imkanı sağlar. Bu işlemi yapabilmesi için **DAP** (Data Access Protocol) denilen bir protokol kullanır. DAP bir dosya transfer protokolü değildir, kullanıcıya dosyanın herhangi bir kaydına erişme olanağı verdiğinden daha genel bir protokol olarak kabul edilebilir. Dosyalar bilindiği gibi üç çeşittir: Sequential, Random ve Indexed. DEC sistemlerinde bu dosya tiplerine göre kayıtlar byte count ve stream olarak iki tipte saklanabilir. Her kayıtın içinde uzunluğunun ne kadar olduğunu belirten bir byte'lık bilgi vardır. DAP bütün bu durumları gözönüne alarak gerekli format değişikliklerini yapar.

Kullanıcı ile File Server arasında bağlantı kurulduktan sonra her iki taraf DAP CONFIGURATION mesajı gönderir. Bu mesaj kullanılan protokol versiyonunu, işletim sistemi ve dosya sistemini, buffer uzunluğunu, ve genel olarak sistemin kapasitesiyle ilgili bilgileri verir.

CONFIGURATION mesajından sonra, ATTRIBUTES mesajları erişilecek dosyaların içeriği ve özelliklerini tanımlamak için kullanılır. Her ulaşım bağlantısı içinde sadece bir dosyaya ulaşılabileceği için bu çeşit bir tanımlama kolaylık sağlar. Dosya özellikleri de belirlendikten sonra ACCESS mesajı yapılacak işlemi belirler: Dosya açma, yaratma, ismini değiştirme, silme, directory listeleme ve komut dosyasını çalıştırma gibi.

Son olarak CONTROL mesajı bilginin transfer edilmesi için kullanılır. CONTROL mesajının dosyaları küçük parçalara bölme gibi daha ileri fonksiyonları da vardır.

Yukarıdaki mesajlara ek olarak ayrıca DATA, ACKNOWLEDGEMENT ve STATUS gibi mesajlar da vardır.

2.5 XNS (XEROX Network Systems)

2.5.1 GİRİŞ

XNS (Xerox Network Systems) mimarisi, Xerox'un Palo Alto araştırma merkezinde PUP (PARC Universal Packet) üzerinde yapılan çalışmalar sırasında geliştirilmiştir. Novell, Ungermann-Bass ve 3Com gibi yerel iletişim ağları üzerine çalışan firmalar XNS'i kendi ürünleri için geliştirmişlerdir.

XNS'in internetworking için yaklaşımı her paketin "en iyi verim (best effort)" temeline göre yönlendirilmesidir. XNS ayrıca çabuk istek ve yanıt alma sağlayan protokollere, genişletilmiş haberleşme, hata raporlama özelliklerine, yönlendirme kontrolu ve sabit yönlendirme ve güncelleme tablolarına sahiptir.

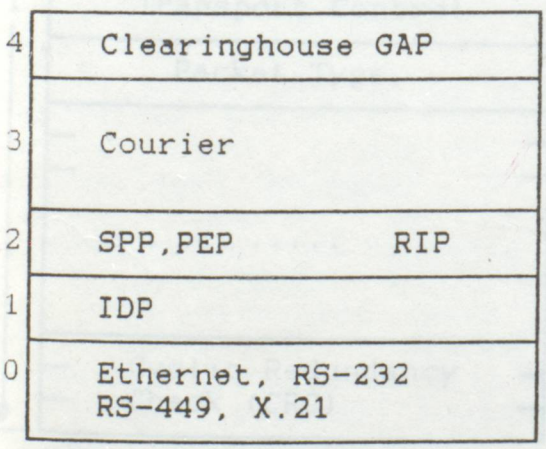
2.5.2 XNS MİMARİSİ

Şekil.2.5.1'den de görüleceği gibi XNS 0.seviyeden başlayan beş seviyeye bölünmüştür.

XNS'in 0.seviyesi OSI'nin fiziksel ve data link seviyelerine karşı düşer. 0.seviye protokolleri datayı transmisyon ortamında bir noktadan diğerine taşır. Bu seviye için en uygun isim "Ethernet"tir, fakat "Synchronous Point-To-Point Protocol" denilen bir başka protokol de vardır. Mimari ayrıca X.21, RS-232C, RS-449 gibi bilinen protokolleri de destekler.

XNS'in 1.seviyesi yönlendirme işlevinin merkezidir. XNS 1.seviye protokollerinde internet kaynak ve varış adresleri bulunur ve paketin nereye gideceği belirlenir. XNS bu seviyeda sadece IDP'yi (Internet Datagram Protocol) tanımlar.

XNS 2.seviye protokolleri OSI'nin transport seviyesine karşı düşer ve mesajın tamamlanması gibi servisler verir. XNS 2.seviyesi beş protokol ve bunlara karşı düşen beş paket tipi tanımlar. Bu protokoller IDP servislerinin üstünde yer alır ve retransmission, sequencing, akış kontrolu ve yönlendiricilerin yönlendirme aşamasında bilgiyi nasıl paylaşacağını belirler.



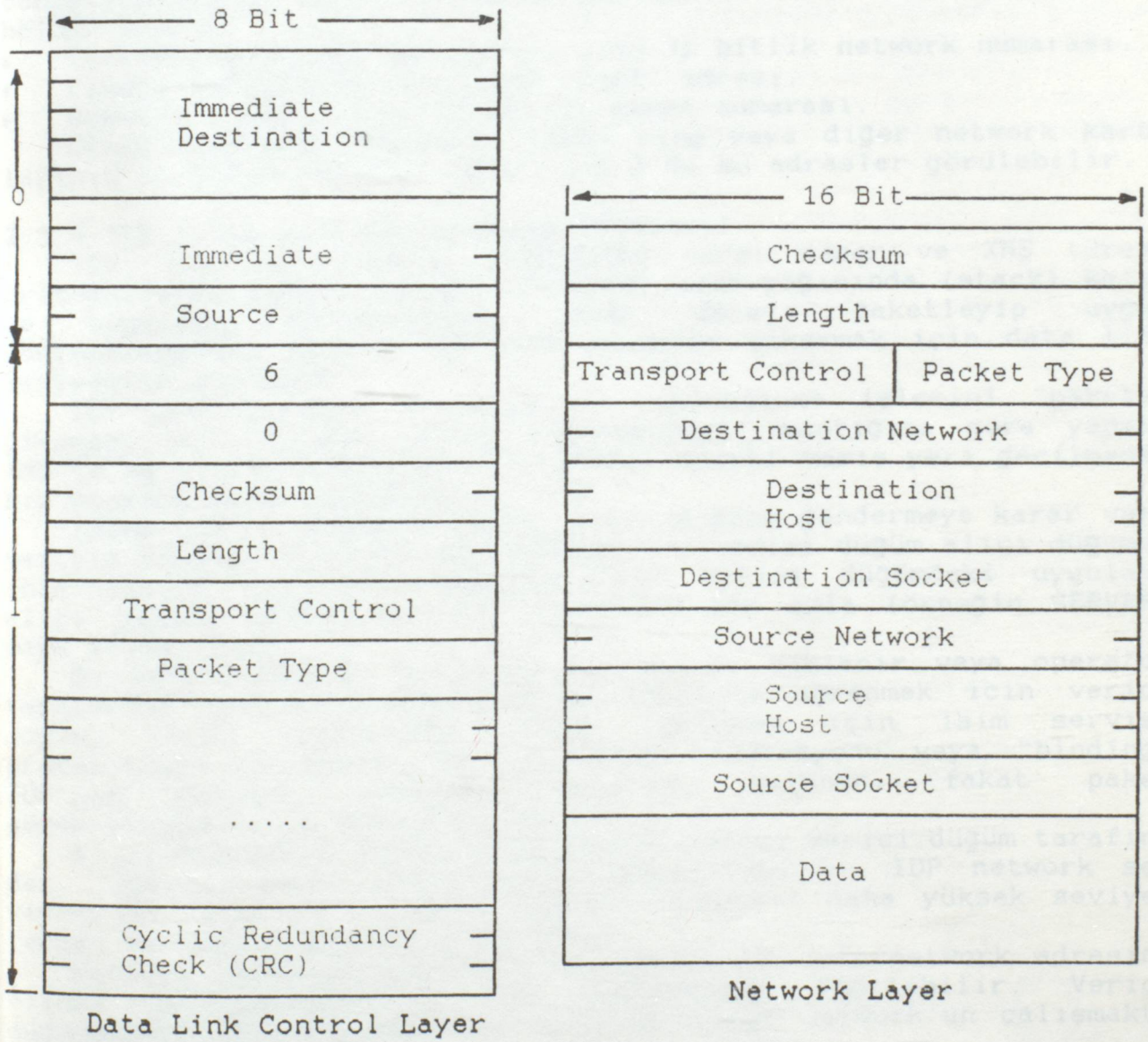
GAP: Gateway Access Protocol

SPP: Sequenced Packet Protocol
PEP: Packet Exchange Protocol
RIP: Routing Information Protocol
IDP: Internetwork Datagram Protocol

Şekil.2.5.1: XNS Protokol Modeli.

XNS'in 3.seviye protokolleri OSI'nin 5. ve 6. seviyelerinin, yani oturum (session) ve sunuş (presentation) seviyelerinin verdiği servislere benzer işleve sahiptir. Bu protokoller data yapısı oluşturur, kullanıcıların uzak kaynaklara ve dosyalara ulaşmasında, kaydetme ve print işlemleri ve çeşitli formatlardaki görüntü cihazlarıyla haberleşme yapar.

XNS'in 4.seviyesi OSI'nin uygulama (application) seviyesine karşı düşer ve özel uygulama protokolleri kullanır. Internetworking protokolunu kullanan birçok uygulama bu seviyede geliştirilir.



Şekil.2.5.2: IDP Paketleri.

2.5.3 ADRESLEME VE SOKETLER

XNS protokolu kullanan düğüm ve yönlendiricilerin çalışmasını anlamak için önce internetworking adresleme kavramını anlamak gerekir. Her XNS cihazı bir network adresi, cihaz adresi (host adress olarak bilinir) ve aynı düğümde çalışan uygulamaların ilişkili olduğu soket numarasını içeren bir adrese sahiptir. Bundan sonra "host" sözcüğü network'teki bir bilgisayar yerine kullanılacaktır.

İletişim ağındaki her cihazın içinde "soket" denilen birkaç yazılım modülü bulunur. Soket cihazın belleğinde bir alt adrestir ve uygulama soketi datayı gönderirken veya alırken kullanır. Soket dosya servisi veya print servisi gibi belirli network fonksiyonlarında düzenli olarak kullanılır. Her XNS adresinin üç bölümü vardır:

- Internetwork'teki LAN adresi olan 32 bitlik network numarası.
- Cihazı tanımlayan 48 bitlik "host" adresi.
- Soketleri tanımlayan 16 bitlik soket numarası.

Cihaz numarası ethernet, token ring veya diğer network kartlarının adresine eşittir. **Şekil.2.5.2**'de bu adresler görülebilir.

2.5.4 IDP (Internetwork Datagram Protocol)

IDP, OSI'nin network seviyesine karşı düşer ve XNS türevi protokollerin yükünü taşır. IDP haberleşme yığınında (stack) kalır ve uygulama seviyesinden gelen datayı paketleyip uygun yönlendirmeleri yaparak network ortamına çıkarmak için data link seviyesine gönderir.

IDP protokolünde yönlendirici yönlendirme işlemini "paketin gideceği bir sonraki yerin belirlenmesi" mantığına göre yapar. XNS'te ağ yönlendiricileri bir paketi önceki varış yeri geçilmeden bir sonraki yere göndermez.

İşlem verici düğümün paketi alıcı düğüme göndermeye karar vermesiyle başlar. Data gönderilmeden önce verici düğüm alıcı düğümün internetwork adresinin öğrenmelidir. Verici düğümdeki uygulama alıcı düğümü yüksek seviyeli sembolik bir adla (örneğin SERVER5 veya VAX23) tanır.

Bu adlar genelde uygulama tablosunda saklanır veya operator tarafından girilir. Internetwork adresini öğrenmek için verici düğümü adres tablosunda adresi öğrenmek için isim servisi protokolleri kullanır. Bu tip isim rezolüsyonu veya "binding" IDP'nin network seviyesi üzerinde bulunur, fakat paket gönderilmeden önce öğrenilmelidir.

Alıcı düğümünün bütün internetwork adresi verici düğüm tarafından öğrenildikten sonra paket gönderilebilir. IDP network seviyesinde isimleri anlamaz, sadece adresleri daha yüksek seviyelerden ve uygulamadan alıp kullanır.

Network seviyesi kullanıcı bilgisini ve internetwork adresini elinde bulundurduğundan ilk yönlendirme yapılabilir. Verici düğümündeki network seviyesi yazılımı hangi network'un çalışmakta olduğunu bilir.

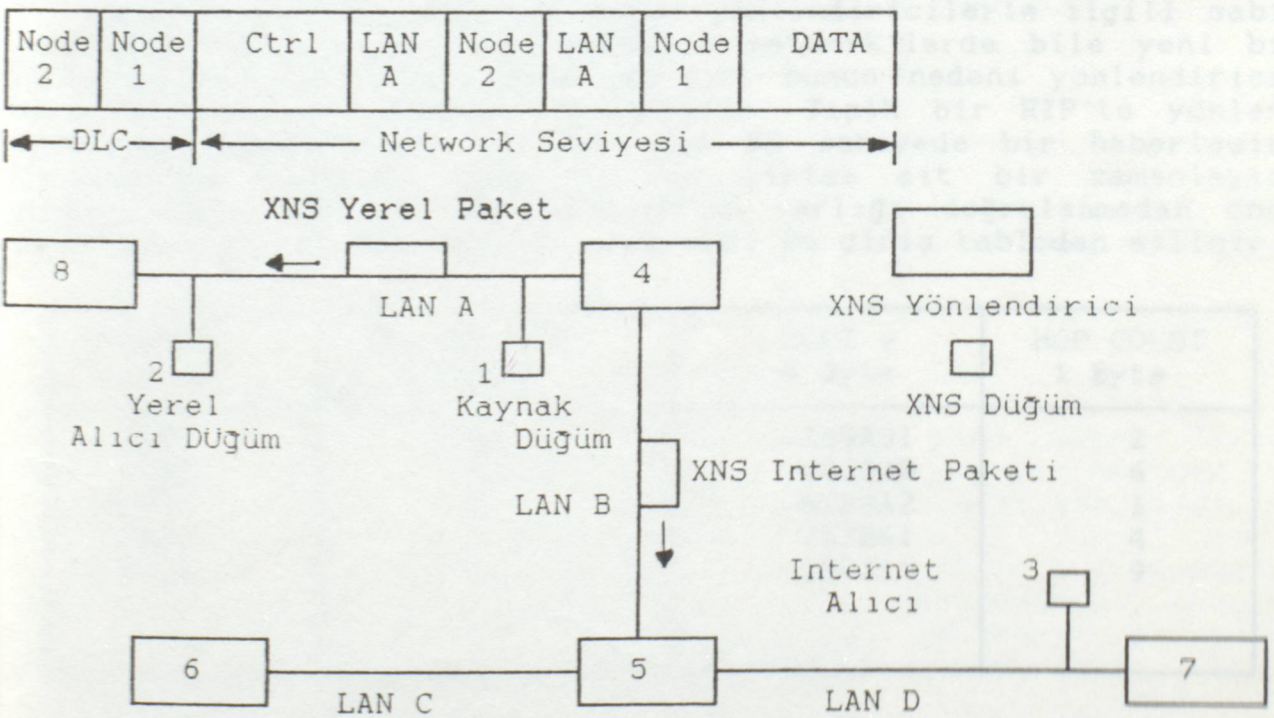
Eğer alıcı düğüm çalışan bir network'te ise kaynağın network seviyesi pakete kullanıcı datasını yerleştirip alıcı düğümün cihaz numarasını eklemesi için data link seviyesine gönderir. Bu olay **Şekil.2.5.3**'te lokal paket örneğinde gösterilmiştir. Basit bir düğümden yakın bir düğüme haberleşmenin yönlendirilmeye ihtiyacı yoktur.

Internetwork'un yönlendiricilerle bölündüğünü hatırlarsak, Şekil.2.5.3'te LAN A'daki data link seviyesi paketleri yönlendiricinin belirlediği LAN segmentinde yol alır. Köprülerin tersine, yönlendiriciler bu paketleri direkt olarak adresler.

Internetworking'in mekanizmasına gelince, verici düğümü bir başka LAN segmentindeki alıcı düğümünü bütün internetwork adresini içinde bulunduran bir paket oluşturur. önceki lokal örneğin tersine, network seviyesi yazılımı data link seviyesine adres olarak alıcı düğümünün adresini değil, yönlendiricinin cihaz adresini verir. Data link adresi XNS'teki adresi çağırır, hedef uç düğümünün adresine gerek yoktur. Bu durum Şekil.2.5.3'te internetworking örneğinde gösterilmiştir.

Yönlendirici, rota tablosundan alıcı düğümü için en yakın ve uygun olan bir cihaz adresini öğrenir. Sonraki yönlendiricininin cihaz adresi artık yeni adres olarak kabul edilir ve paket iletilmek üzere yeniden data link seviyesine gönderilir. Bu şekilde paket belki de yine bir yönlendiriciye gönderilecektir. Son olarak, aynı LAN segmentindeki bir yönlendirici alıcı düğümü ise paketi alır ve kendi cihaz adresini DLC adresi olarak kullanıp uç düğüme gönderir.

Bu noktada IDP internetwork haberleşmesi tamamlanır. Eğer bazı yüksek seviyeli yazılımlar haberleşmeyi garantileyecekse bir onaylama (acknowledge) verici düğüme geri gönderilebilir. Fakat bu garantiye gerek duyulmuyorsa IDP "en iyi verim (best effort)" kriterine göre yapar ve yeniden onaylamaya gerek kalmaz. IDP, paketin max. uzunluğunun normalde 576 byte olduğunu kabul eder.



Şekil.2.5.3: XNS'te Paket Haberleşmesi.

2.5.5 RIP (Routing Information Protocol)

XNS sistem yönetimi ve kontrol protokolleri şunlardır: **RIP (Routing Information Protocol)** Yönlendiriciler arası haberleşme için yönlendirme bilgisi protokolü, yönlendiriciler için paket bozulmalarını rapor eden hata protokolleri **(Error Protocol)** ve bilinmeyen cihazları ve yolu (path) test eden **Echo Protocol**.

XNS data transfer servisleri ise: **SPP** (Sequenced Packet Protocol) ve **PEP** (Packet Exchange Protocol)'dir.

XNS ve türevi LAN'lar için transport seviyesi protokollerinin en önemlileri datanın yönlendirilmesini sağlayan RIP, IDP'nin üzerinde garantilenmiş servisler sağlayan SPP, ve XNS SPP'nin çok benzeri olan ve XNS IDP'den türetilen Novell'in **IPX-Internet Packet Exchange**'dir.

Yönlendirici düğümden düğüme yönlendirme yapabilmesi ve adresleri hesaplayabilmesi için kendi yönlendirme tablosunu kullanır. XNS'te ve Novell'in IPX ve SPX'inde RIP yönlendiricilerinin kullandığı RIP protokolü yönlendirme tablosunu yönetir ve güncelleştirir. Internet yönlendiricileri RIP ve onun network topolojisine özel paket formatını kullanır.

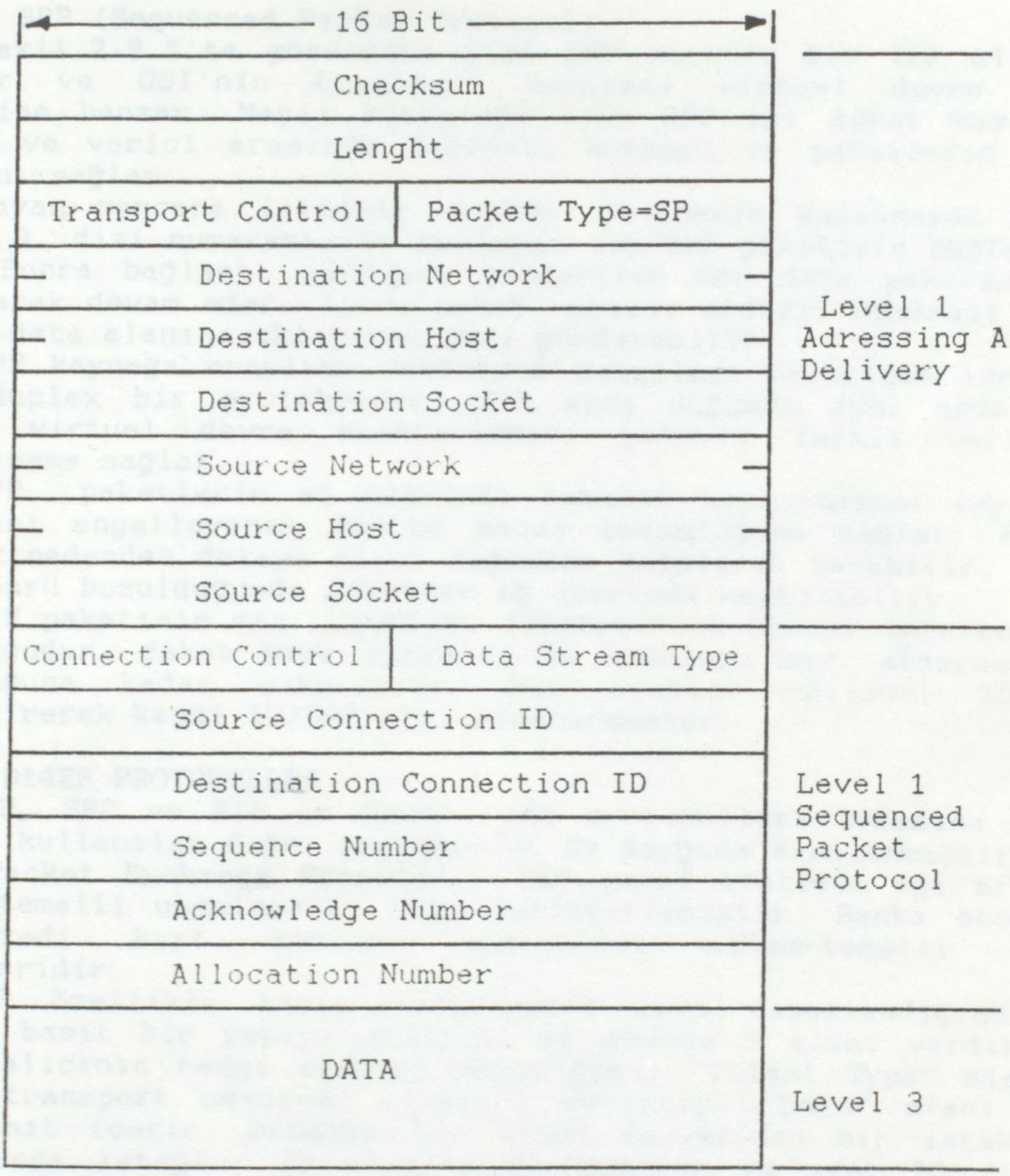
RIP, yönlendiricilerin network numaraları, yönlendirme adresleri, ve yönlendiriciye bağlı sıçrama (hop) sayısı, yönlendirme tablosu yaratmasına ve düzenlemesine izin verir.

Şekil.2.5.4'teki yönlendirme tablosu internetwork'teki her ağ için bir giriş içerir. Her ağın girişi için yönlendirme tablosu komşu (adjacent) yönlendiricinin cihaz adresini ve sonraki yönlendiricinin bulunduğu ağı verir. Yönlendirme tablosunda toplam yönlendirici sayısı da saklanır ve buna "hop count" denir.

Yönlendiriciler kendi ve diğer yönlendiricilerle ilgili sabit bilgileri paylaşırlar. Çok karmaşık network'lerde bile yeni bir yönlendirme bilgisi her yere yayılır, bunun nedeni yönlendiricilerin diğerlerini sürekli izlemesidir. Tipik bir RIP'te yönlendiriciler diğerleriyle her 30 veya 60 saniyede bir haberleşir. Yönlendirme tablosuna yapılan her girişe ait bir zamanlayıcı vardır. Eğer verilen bir network'un varlığı doğrulanmadan önce zamanlayıcının süresi dolarsa (run-out) bu giriş tablodan silinir.

SUBNET # 4 Byte	ROUTER PORT 1 Byte	NODE # 6 Byte	HOP COUNT 1 Byte
68A2	A	169A31	2
C43B	A	C6234B	6
1286	B	6789A2	1
D94C	C	7A7B61	4
672A	A	250311	9
.	.	.	.
.	.	.	.

Şekil.2.5.4: XNS Rota Tablosu.



Şekil.2.5.5: SPP Paketi.

Standart XNS'te yönlendirici bilgisini her yere iletir. Ayrıca yönlendırıcı kendi RIP paketlerini 60 saniye yerine her 30 saniyede bir iletir ve bu şekilde network band genişliği ve yönlendirme trafiği azaltılır.

Novell'in RIP paketinde fazladan eklenen 2-byte'lık parametre "Time-To-Net" adıyla bilinir ve uç istasyonunun tahmini gecikmesini saklar. Bu göndericinin doğru gecikmeyi bilmesini sağlar, bu da çok çabuk time-out olmasının veya paketi fazla beklemenin önüne geçer.

2.5.6 SPP (Sequenced Packet Protocol)

Şekil.2.5.5'te görüldüğü gibi SPP kararlı bir IDP alışverişi sağlar ve OSI'nin transport seviyesi virtuel devre servisi işlemine benzer. Mesaj bütünlüğü için SPP iki soket vasıtasıyla alıcı ve verici arasında bağlantı kurmayı ve paketlerin senkron akışını sağlar.

Kayan pencere (sliding window) protokolü kullanarak her iki soket 0. dizi numarası ile başlayan ilk SPP paketiyle bağlantı kurar. Sonra bağlantı sürdükçe gönderilen her data paketini numaralayarak devam eder. Alıcı soket, mesajı ardışıl numaralı paketin kendi data alanına ekleyerek geri gönderebilir.

SPP kaynağa onaylama istenecek paketleri belirleme izni veren full-duplex bir protokoldur. SPP aynı düğümde aynı anda birden fazla virtuel devre anahtarlaması yoluyla farklı vericilerle haberleşme sağlar.

SPP paketlerin ağ üzerinde tamamen kaybolmasını veya iptal olmasını engelleyerek XNS'te mesaj bütünlüğünü sağlar. Paketler birçok nedenden dolayı alıcı düğümüne bozularak varabilir, örneğin bir köprü bozulduğunda paketler ağ üzerinde kaybolabilir.

SPP paketinin max. uzunluğu 576-byte'lık normal internet paket uzunluğudur, fakat bazı firmalar bu uzunluğu max. ethernet paket uzunluğuna kadar çıkarırlar. Her üretici orijinal IDP/SPP'ı değiştirerek kendi ID/SPP'sini oluşturmuştur.

2.5.7 DİĞER PROTOKOLLER

IDP, SPP ve RIP en önemli XNS protokolleri olmasına rağmen, XNS'de kullanılan diğer protokoller de aşağıda sıralanmıştır:

PEP (Packet Exchange Protocol: PEP genel iletişim ağı ortamında işlem-temelli uygulamalar için geliştirilmiştir. Banka otomatları ve kredi kart kontrol sistemleri işlem-temelli çalışma örnekleridir.

PEP özellikle basit istek-yanıt için tasarlandığından PEP paketi basit bir yapıya sahiptir ve sadece 3 alanı vardır. "ID" alanı alıcının hangi servisi istediğini, "Client Type" alanı belirli transport seviyesi client'i belirtir. "Data" alanı isteğe bir yanıt içerir. Örneğin bir cihaz server'den bir istekte bulunduğunda isteğini ID alanına yerleştirir ve aynı ID ile Data alanınından alır.

Courier. (RPC-Remote Procedure Call protocol): XNS Courier, diğer mimarilerde oturum (session) seviyesi servislerine benzer ve uzaktan işlem yaptıran SPP ve IDP gibi alt seviyeleri kullanır.

Örneğin Courier RPC kullanan bir uygulama bir uzak printere lokal printer gibi ulaşabilir. Uygulamaya göre printer uzak bir fonksiyondur. Herhangi bir RPC protokoluyla Courier lokal fonksiyonu alıp uzak kaynağa transfer eder.

BDT-Bulk Data Transfer: Courier'de BDT-Bulk Data Transfer protokolu, genelde Courier'in alabileceği kadar büyük data bloklarının transferini sağlar. BDT protokolu dosya transferi ve mainframe-to-mainframe veritabanı transferi yaparken data bloğunu tek bir Courier datası gibi uzun bir dizi olarak algılar.
Error Protocol: Pakette bir hata (genellikle CheckSum hatası) sezen herhangi bir cihaz paketi iptal edip, vericiye hatanın cinsini ve bozulan paketi bildiren bir Hata Protokolu göndermelidir. Haberleşme donanımları gün geçtikçe kararlı ve hata kontrolu yapabilen bir hale geldiklerinden Hata Protokolunun artık XNS'de fazla önemi kalmamaktadır. Yönlendiriciler bir hata kontrol paketi gönderecekleri zaman, kendi hata soketini kullanırlar.
Diğer Uygulamalar: Xerox XNS sistemlerini diğer sistemlere bağlamak için GAP (Gateway Access Protocol) kullanır. Bu şekilde OSI'nin virtuel terminal protokolu gibi, 7.seviyede protokol çevirme işlemi yapılır. Dosyalama servisi, print servisi, doküman değişimi,elektronic mail gibi diğer uygulama seviyesi servisleri de burada yapılır.

2.6 BİLGİSAYAR AĞ MİMARİLERİNİN KARŞILAŞTIRILMASI

Bu bölümde daha önceki bölümlerde tanıtılan network mimarilerinin kısaca karşılaştırılması yapılacaktır. Bu karşılaştırmada önce mimarilerin protokol seviyeleri arasında karşılaştırmalar, en yaygın olan OSI referans modeli ile arasındaki farklılıklar, daha sonra gelecekte hangi mimarinin daha yaygın olabileceğine ilişkin görüşler verilecektir. **Şekil.2.6.1**'de bu mimarilerin protokol seviyeleri arasındaki farklılıklar görülmektedir. Tablodaki temel mimari OSI referans modelidir, diğer mimarilerin protokol seviyeleri buna en yakın seviyelere göre seçilmiştir.

SNA ile OSI arasında karşılaştırma yapılacak olursa özellikle 3,4 ve 5 nolu seviyeler başta olmak üzere bu mimarilerin birbirine pek uyumlu olmadığı görülür. Her iki mimaride de en alt iki seviye fiziksel ve data link seviyeleridir. 3. seviye olan "path control" seviyesi OSI'nin network seviyesine benzer işlemler yapar. SNA'da ulaşım seviyesi path control ve transmisyon kontrol seviyeleri ile eşit seviyededir. Path control seviyesi uçtan-uca kararlılık ve güven sağlar, fakat ulaşım bağlantılarının kurulması ve kaldırılması transmisyon kontrol seviyesi tarafından yapılır. Oturum seviyesi fonksiyonlarının bir kısmı transmisyon kontrol seviyesi tarafından, bir kısmı da data akış kontrol seviyesi tarafından yapılır. Sunuş seviyesini SNA'da NAU (Network Accessible Unit) servisleri yerine getirir ve birbirinden bağımsız üç servis verir: Sunuş servisleri, oturum servisleri ve network servisleri. Uygulama seviyeleri de her iki mimaride aynı işlevleri yaparlar.

S	ISO	SNA	DNA	XNS
7	Uygulama	Kullanıcı	Uygulama	Clearing-house GAP
6	Sunuş	NAU Servisleri		Courier
5	Oturum	Data Akış Kontrol	(Yok)	
4	Ulaşım	Transmisyon Kontrol	Network Servisleri	SPP, PEP, RIP
3	Ağ	Yol Kontrol	Ulaşım	IDP
2	Data Link	Data Link Kontrol	Data Link Kontrol	Ethernet RS-232 RS-449 X.21
1	Fiziksel	Fiziksel	Fiziksel	

NAU: Network Accessable Unit
GAP: Gateway Access Protocol
RIP: Routing Information Protocol
IDP: Internetwork Datagram Protocol
SPP: Sequenced Packet Protocol
PEP: Packet Exchange Protocol

Şekil.2.6.1: Çeşitli network mimarilerinin karşılaştırılması.

OSI ile DNA karşılaştırılacak olursa, SNA'e göre ikisi mimari arasında biraz daha fazla benzerlik olduğu görülebilir. DECNET'te beş seviyeden fiziksel, data link, ulaşım ve network servisleri seviyeleri aşağı yukarı OSI modeliyle aynıdır. Ancak DECNET'te oturum seviyesi olmadığından bu benzerlik bozulur, 5.seviye olan uygulama seviyesi OSI'nin sunuş ve uygulama seviyesini kapsar. Bundan başka DECNET'te üçüncü seviye ulaşım seviyesidir, OSI'de ise ulaşım seviyesi 4.sıradadır.

Son olarak XNS'le OSI modelleri karşılaştırıldığında görüleceği gibi XNS OSI'ye göre daha tümleşik bir yapıya sahiptir. XNS'in 0.seviyesi OSI'nin fiziksel ve data link seviyelerine karşı düşer. 1.seviye OSI'nin network seviyesiyle aynı servisleri verir, paketlerin network boyunca yönlendirme işlemleri bu seviyede yapılır. XNS'in 2.seviye protokolleri OSI'nin ulaşım seviyesi protokollerine benzer. 3.seviye OSI'nin oturum ve sunuş seviyelerine karşı düşer. 4.seviyede OSI'nin uygulama seviyesine yaklaşık işlemler yapılır.

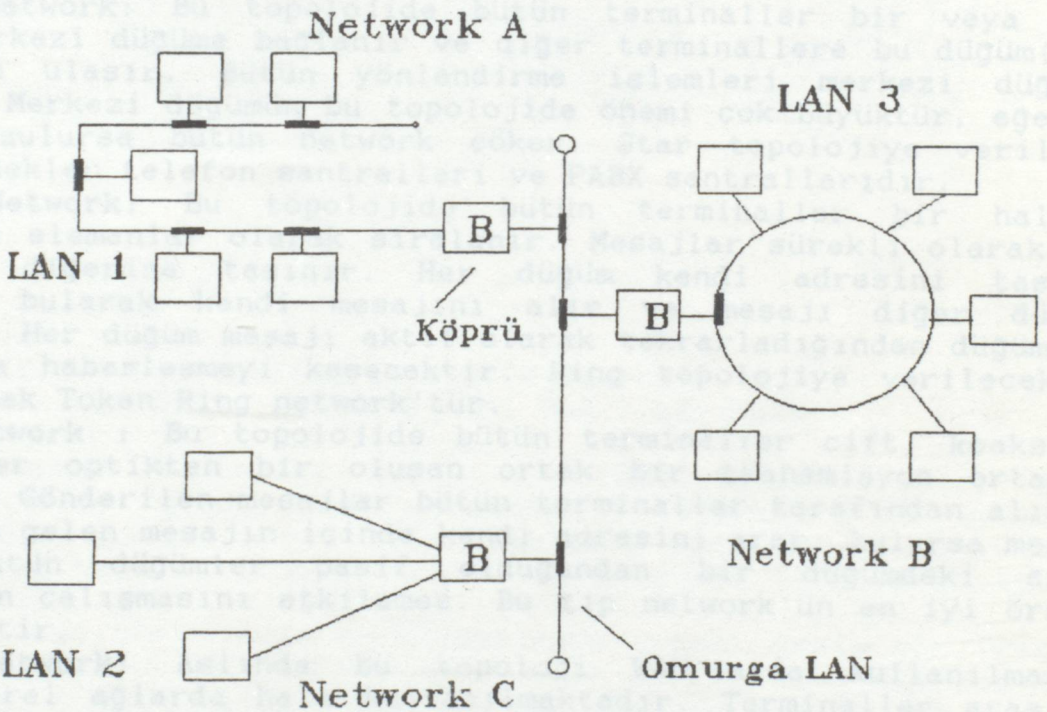
Protokol mimarileri arasında yapılan bu inceleme sonucunda OSI referans modelinin bundan sonraki dönemlerde yine en genel mimari olarak kalacağı söylenebilir. özellikle XNS'in yerel iletişim ağlarında daha fazla yaygınlaşacağı beklenebilir.

3.1 YEREL İLETİŞİM AGLARI

3.1.1 GİRİŞ

Yerel İletişim Ağları (LAN-Local Area Network) bir bina veya binalar grubu içinde yayılmış olan bilgisayar ve veya bilgisayar temelli cihazların oluşturduğu bir iletişim ağıdır. Bu ağda her bilgisayar birbirinden bağımsız çalışır. Yerel iletişim ağlarında cihazlar arası uzaklık az, haberleşme hızı çok yüksek ve hata oranı çok düşüktür. Şekil.3.1'de örnek bir LAN görülmektedir. Genelde bir yerel ağı belirleyen temel özellikler aşağıda sıralanmıştır:

- Çapı maksimum birkaç kilometre olabilir,
- Data hızı minimum 1 Mbps'dır,
- Sadece tek bir organizasyona aittir. Bu özelliği nedeniyle LAN'lara "Özel Ağlar (Private Data Networks)" de denir.



Şekil.3.1: LAN temelli dağıtılmış sistem örnekleri.

Yukarıdaki özellikleri nedeniyle LAN'larda kullanılan protokoller Geniş Alan Ağlarında (WAN-Wide Area Network) kullanılamaz. PTT hatlarını kullanan WAN'ların tersine LAN'ların kendi haberleşme ortamı vardır. Genel olarak bir yerel iletişim ağının temel parametreleri aşağıdadır:

- Transmisyon ortamları,
- Topolojiler,
- İşaretleşme protokolleri.

3.2 TOPOLOJİLER

Topoloji iletişim ağındaki elemanların fiziksel yerlerine göre kullanılacak bağlantı sistemidir. Bu bağlantı sistemi kablo veya diğer haberleşme sistemleri (radyo, mikrodalga gibi..) üzerinden yapılabilir.

Bilgisayar ağlarında topolojinin önemi performans, maliyet ve güvenlik gibi kriterler düşünüldüğü zaman ortaya çıkar. Bilgisayar ağı tasarlanırken eldeki veriler bilgisayar ve terminallerin yerleri, trafik yoğunluğu ve uzaklık nedeniyle oluşan maliyettir. Performans düşünüldüğünde kararlılık ve hız (gecikme) önem kazanacaktır. Üzerinde çalışılacak değişkenler ise topoloji, hat kapasiteleri, ve akış kontroludur. Topoloji işaretleşme şekliyle de ilgilidir ve bazan protokolün ismiyle anıldığı da olur.

Yerel iletişim ağlarının en yaygın topolojileri aşağıda sıralanmıştır:

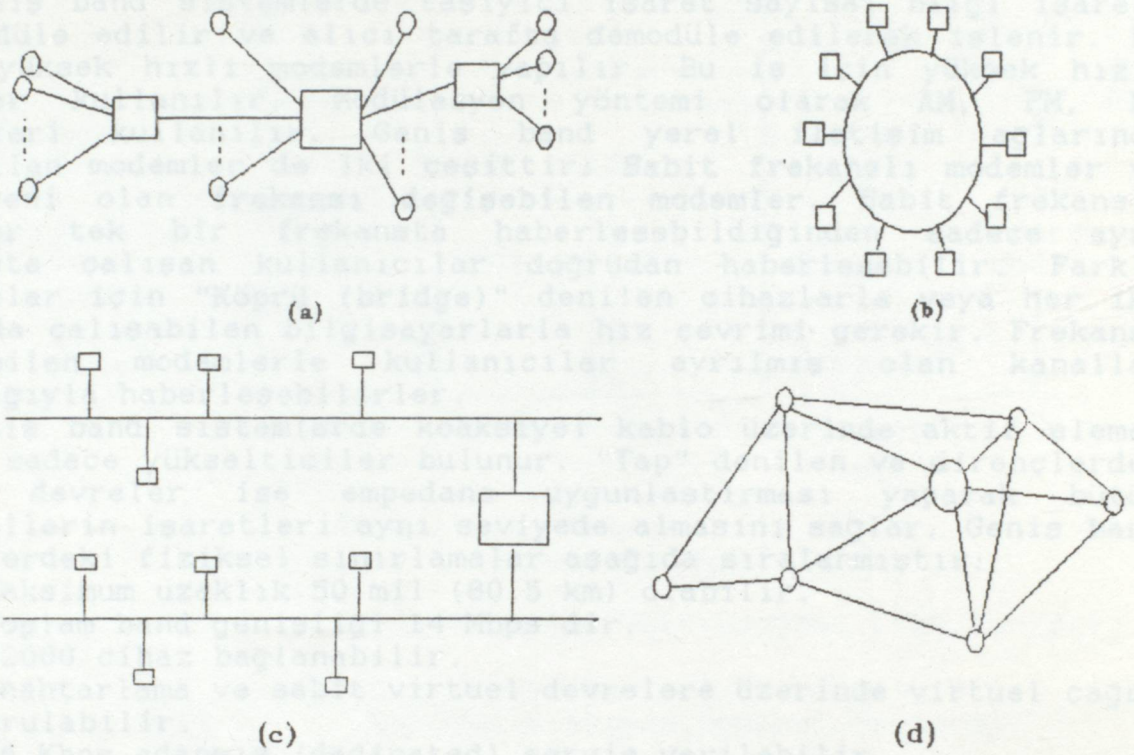
1-Star Network: Bu topolojide bütün terminaller bir veya daha fazla merkezi düğüme bağlanır ve diğer terminallere bu düğüm(ler) üzerinden ulaşır. Bütün yönlendirme işlemleri merkezi düğümde yapılır. Merkezi düğümün bu topolojide önemi çok büyüktür, eğer bu düğüm bozulursa bütün network çöker. Star topolojiye verilecek tipik örnekler telefon santralleri ve PABX santrallarıdır.

2-Ring Network: Bu topolojide bütün terminaller bir halkayı oluşturan elemanlar olarak sıralanır. Mesajlar sürekli olarak bir düğümden diğerine taşınır. Her düğüm kendi adresini taşınan mesajdan bularak kendi mesajını alır ve mesajı diğer düğüme gönderir. Her düğüm mesajı aktif olarak tekrarladığından düğümdeki bir arıza haberleşmeyi kesecektir. Ring topolojiye verilecek en tipik örnek Token Ring network'tür.

3-Bus Network : Bu topolojide bütün terminaller çift, koaksiyel veya fiber optikten bir oluşan ortak bir transmisyon ortamına bağlanır. Gönderilen mesajlar bütün terminaller tarafından alınır. Her düğüm gelen mesajın içinde kendi adresini arar, bulursa mesajı alır. Bütün düğümler pasif olduğundan bir düğümdeki arıza network'ün çalışmasını etkilemez. Bu tip network'ün en iyi örneği Ethernet'tir.

4-Mesh Network: Aslında bu topoloji WAN'larda kullanılmasına rağmen yerel ağlarda hala kullanılmaktadır. Terminaller arasında bağlantılar uçtan uca yapılır, genelde bütün terminaller birbirine bağlanmaz. Önemli noktalardaki terminallerin yönlendirme kapasitesi vardır ve trafik yönetimini üstlenirler.

Şekil.3.2'de yukarıdaki network topolojileri basit olarak gösterilmiştir.



Şekil.3.2: Yerel İletişim Ağ topolojileri:
(a) Star; (b) Ring (c) Bus; (d) Mesh.

İşaretleşme tiplerine yerel iletişim ağları "temel band (baseband)" ve "geniş band (broadband)" olarak ikiye ayrılır. Temel bandda çalışan yerel ağlar hakkında geniş bilgi daha sonra geniş olarak verileceğinden burada sadece geniş band yerel ağları incelenecektir.

Geniş band yerel ağlarda transmisyon ortamı geniş bir frekans bandına sahiptir; bu nedenle aynı anda birçok kanaldan iletişim yapılabilir. Temel band ağlarda ise taşınan işaretler sayısal olduğundan aynı anda sadece tek bir bilgi taşınabilir. Transmisyon ortamı olarak kullanılan koaksiyel kablonun frekans bandı 300-400 MHz'dir, fiber optik kablo kullanıldığında çok daha yüksek frekanslara çıkılabilir. Geniş band yerel ağlarda cok çeşitli işaret aynı anda gönderilebilir:

- Bilgisayarlar arası haberleşme.
- Terminal-bilgisayar arası haberleşme.
- Kablolu TV yayını.
- Güvenlik sistemleri için kapalı devre TV işareti.
- Telefon haberleşmesi.
- Yangın alarmları.
- Proses kontrol ve düzenleme işaretleşmesi.

Geniş band sistemlerde taşıyıcı işaret sayısal bilgi işareti ile modüle edilir ve alıcı tarafta demodüle edilerek işlenir. Bu işlem yüksek hızlı modemlerle yapılır. Bu iş için yüksek hızlı modemler kullanılır. Modülasyon yöntemi olarak AM, FM, PM yöntemleri kullanılır. Geniş band yerel iletişim ağlarında kullanılan modemler de iki çeşittir: **Sabit frekanslı modemler** ve daha yeni olan **frekansı değişebilen modemler**. Sabit frekanslı modemler tek bir frekansta haberleşebildiğinden sadece aynı frekansta çalışan kullanıcılar doğrudan haberleşebilir. Farklı frekanslar için "**Köprü (bridge)**" denilen cihazlarla veya her iki hızda da çalışabilen bilgisayarlarla hız çevrimi gerekir. Frekansı değişebilen modemlerle kullanıcılar ayrılmış olan kanallar aracılığıyla haberleşebilirler.

Geniş band sistemlerde koaksiyel kablo üzerinde aktif eleman olarak sadece yükselticiler bulunur. "Tap" denilen ve dirençlerden oluşan devreler ise empedans uygunlaştırması yaparak bütün terminallerin işaretleri aynı seviyede almasını sağlar. Geniş band sistemlerdeki fiziksel sınırlamalar aşağıda sıralanmıştır:

- Maksimum uzaklık 50 mil (80.5 km) olabilir.
- Toplam band genişliği 14 Mbps'dir.
- 32000 cihaz bağlanabilir.
- Anahtarlama ve sabit virtuel devrelere üzerinde virtuel çağrı kurulabilir.
- 56 Kbps adanmış (dedicated) servis verilebilir.
- Genel uygulamalar için 50 kanal ayrılmıştır.
- Anahtarlanmış hatlar üzerinde 19.2 Kbps ve adanmış (dedicated) hatlar üzerinde 56 Kbps hıza çıkabilir.

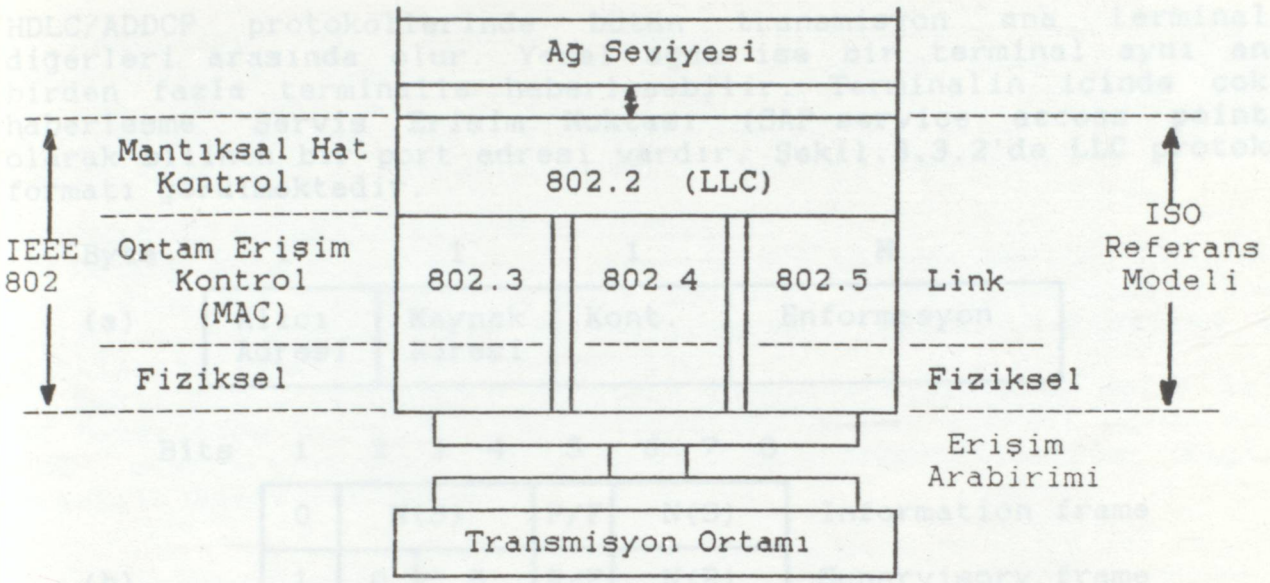
3.3 LAN STANDARTLARI

3.3.1 GİRİŞ

Bu bölümde yerel iletişim ağları için belirlenen standartlar ve OSI referans modelindeki yerleri incelenecektir. Yerel iletişim ağları için belirlenen standartlar sadece LAN içindeki protokolleri değil, fiziksel arabirimler ve network işletim sistemlerinin kullandığı protokolleri de kapsar.

IEEE (Institute of Electric and Electronics Engineers) 1980'de yerel iletişim ağlarında kablolama, elektriksel arabirimler, protokoller ve ağ ürünlerine erişim planı konularında **IEEE 802.X** standartlarını belirledi. IEEE genel kurulu yerel ağ standartları üzerine 802 kez toplandığından standartlara bu isim verilmiştir. IEEE 802.X standartları OSI referans modelinin en alt iki seviyesini kapsar. OSI'nin data link seviyesi HDLC'ye benzeyen "**Mantıksal Hat Kontrolu (LLC-Logical Link Control)**" ve Ethernet gibi "**Ortam Erişim Kontrol (MAC-Medium Access Control)**" seviyesi olarak iki alt seviyeye bölünmüştür. Link seviyesi ile fiziksel işaretleşme ve transmisyon ortamına bağlanan fiziksel bağlantı arasında "**Erişim birimi (AUI-Access Unit Interface)**" tanımlanmıştır.

Şekil.3.3.1'de IEEE 802 protokol modeli görülmektedir.



802.2 - Mantıksal Hat Kontrol Protokol (LLC-Logical Link Control)
802.3 - CSMA/CD
802.4 - Token Bus
802.5 - Token Ring
Ortam Erişim Kontrol Protokolleri (MAC-Media Access Control Protocol)

Şekil.3.3.1: IEEE 802.X protokol modeli.

IEEE 802 protokolleri aslında bir standartlar ailesidir. **Şekil.3.3.1**'den de görüleceği gibi standart, mantıksal hat kontrolu alt seviyesinde 802.X gibi sıra numaralarıyla belirtilen ve topolojilere özel olarak tanımlanan erişim protokollerini de belirler.

3.3.2 IEEE 802.2 Mantıksal Hat Kontrol Seviyesi (LLC-Logical Link Control)

Mantıksal hat kontrol seviyesi, ağ (network) seviyesine ve ortam erişim kontrolu seviyesine servisler verir. Bu servisleri kendi içinde sırasıyla "**arabirim servis özellikleri (interface service specification)**", "**LLC prosedürleri**" ve "**LLC/MAC arabirim servis özellikleri**" gibi alt servislerle yerine getirir. Arabirim servis özellikleri ağ seviyesiyle datagram tipi ve onaylama gerektirmeyen bağlantılar kurar. Bağlantı-temelli servis de verilebilir. LLC prosedürü HDLC ve ADDCCP (ANSI X3.66) protokollerinin asenkron dengeli moduna dayanır. Bu protokoller yerel iletişim ağları için birden fazla terminal ve çoklu erişim ihtiyacına göre genişletilmiştir.

HDLC/ADDCP protokollerinde bütün transmisyon ana terminalle diğerleri arasında olur. Yerel ağda ise bir terminal aynı anda birden fazla terminalle haberleşebilir. Terminalin içinde çoklu haberleşme **"Servis Erişim Noktası (SAP-service access point)"** olarak bilinen bir port adresi vardır. **Şekil.3.3.2**'de LLC protokol formatı görülmektedir.

(a)

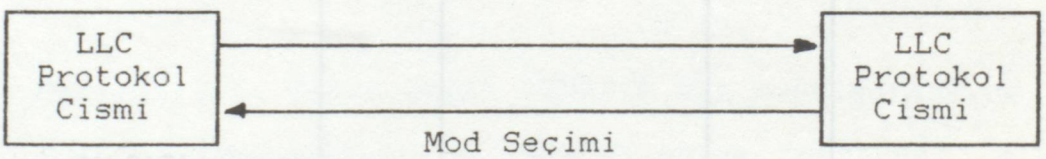
Byte	1	1	1	M
	Alıcı Adresi	Kaynak Adresi	Kont.	Enformasyon

(b)

Bits	1	2	3	4	5	6	7	8	
	0	N(S)			P/F	N(S)			Information frame
	1	0	S		P/F	N(R)			Supervisory frame
	1	1	M		P/F	M			Unnumbered frame

N(S), N(R): Send and receive sequence
S: Supervisory function definition
M: Modifier function definition
P/F: Poll/Final bit

(c)



Şekil.3.3.2: Mantıksal hat kontrolu (LLC) çerçeve formatı.
(a) LLC-PDU formatı.
(b) Kontrol alanı bit tanımlamaları.
(c) Data link kontrol fonksiyonları.

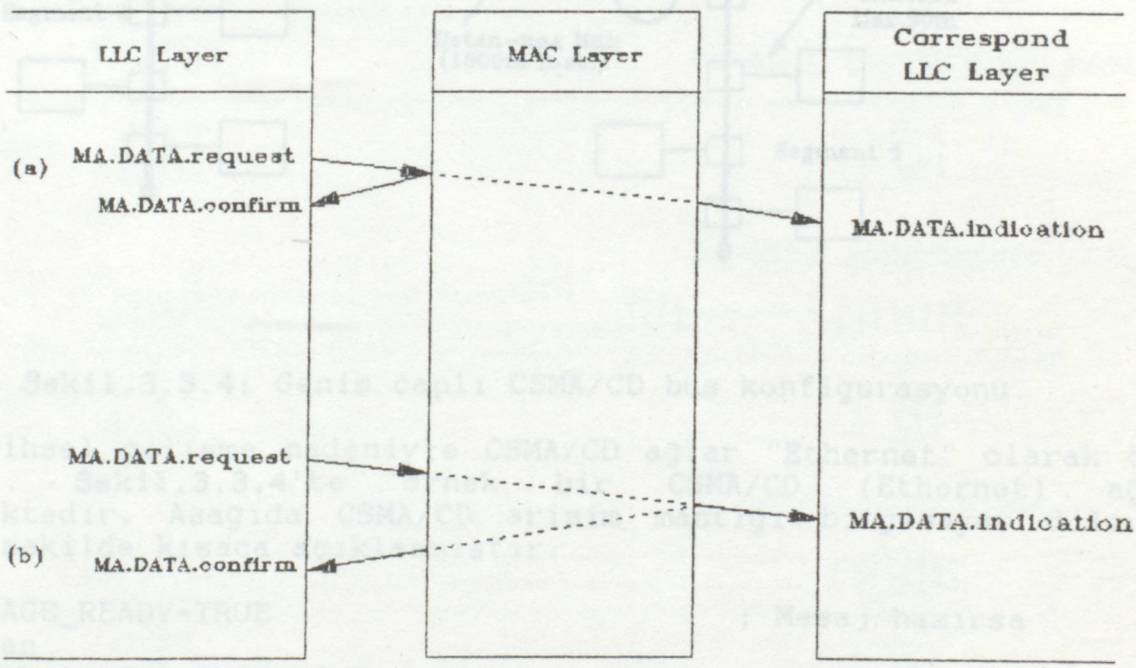
DSAP (Destination Service Access Point) ve **SSAP (Source Service Access Point)** daha önce belirtildiği gibi ayrı ayrı adres bildirir. DSAP ayrıca grup adresi olarak da kullanılabilir. **Kontrol** alanına yerleştirilen bilgiler **Şekil.3.3.2(a)**'da gösterilmiştir. LLC prosedüründe iki tip çalışma şekli vardır. **1.tip** çalışma bağlantısız moddur, mantıksal data hattı kurulmasına, onaylama işlemine, akış kontroluna ve hata düzeltme işlemine gerek yoktur. **2.tip** ise bağlantı-temelli moddur, mantıksal data hattı kurulur, onaylama ve adresleme gereklidir.

Mantıksal hat kontrol (MAC) prosedürünün iki sınıfı tanımlanmıştır: **Class I** sadece 1.tip çalışmadan ibarettir, **Class II** ise her iki tip çalışmaya da izin verir. Böylece bütün terminaller bağlantısız çalışmayı destekler, Class II ise hem bağlantısız, hem de bağlantı temelli çalışmayı destekler.

TRANSMİSYON ORTAMINA ERİŞİM PROTOKOLLERİ

Ortama erişim kontrolu alt seviyesinin çalışma modları CSMA/CD, Token Ring/Bus yöntemlerine göre değişir. Bu seviyede desteklenen kullanıcı servisleri ise **MA.DATA.request**, **MA.DATA.indication** ve **MA.DATA.confirmation** olarak verilir.

Şekil.3.3.3'te CSMA/CD ve token ring/bus tipi yerel ağlar için MAC haberleşme prosedürü gösterilmiştir. Bundan sonraki bölümlerde IEEE standartlarıyla belirlenmiş ağ protokolleriyle ilgili ayrıntılı bilgi verilecektir.

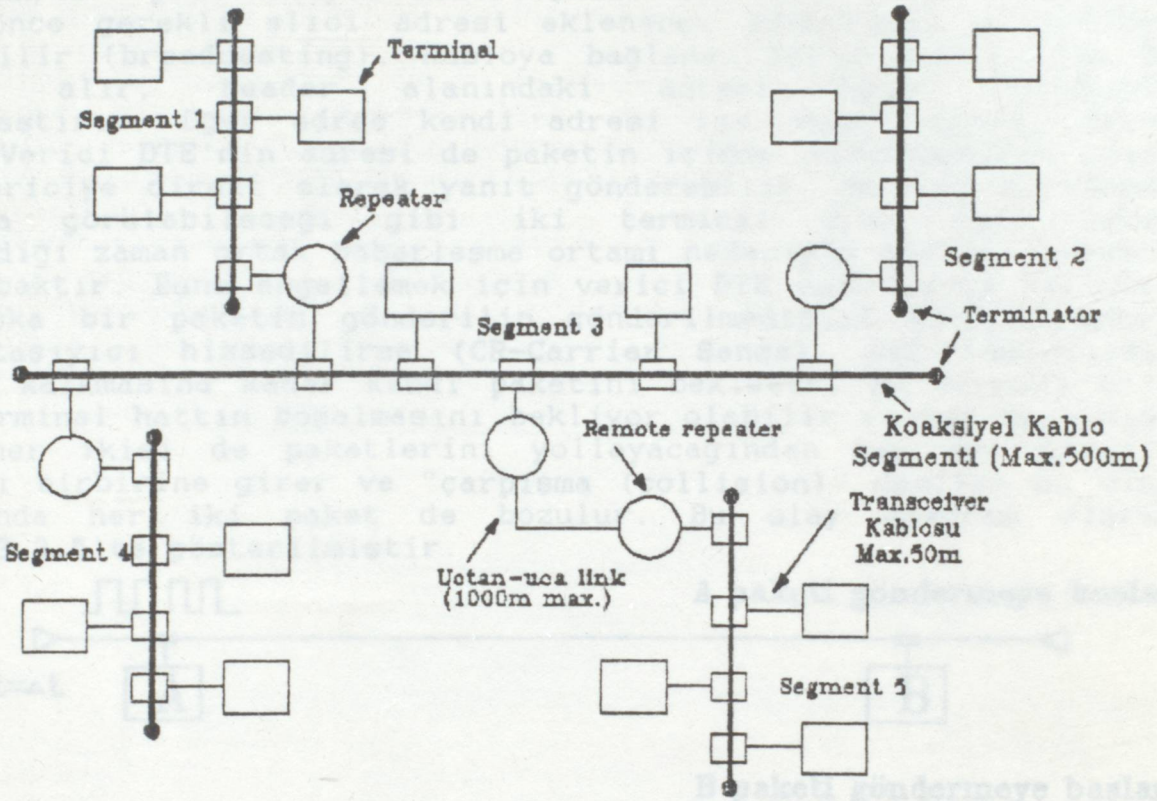


Şekil.3.3.3: MAC kullanıcı servisleri. (a) CSMA/CD
(b) Token Ring/Bus

3.3.3 IEEE 802.3: CSMA/CD STANDARDI
(Carrier Sense Multiple Access with Collision Detection)

3.3.3.1 GİRİŞ

CSMA/CD erişim metodu bus topolojisine sahip yerel iletişim ağlarında kullanılır. Bu topolojide bütün DTE'ler aynı haberleşme ortamını kullanırlar ve haberleşmenin gerçekleştiği bu ortam "**çoklu ulaşım (MA-multiple access)**" modunda çalışır.

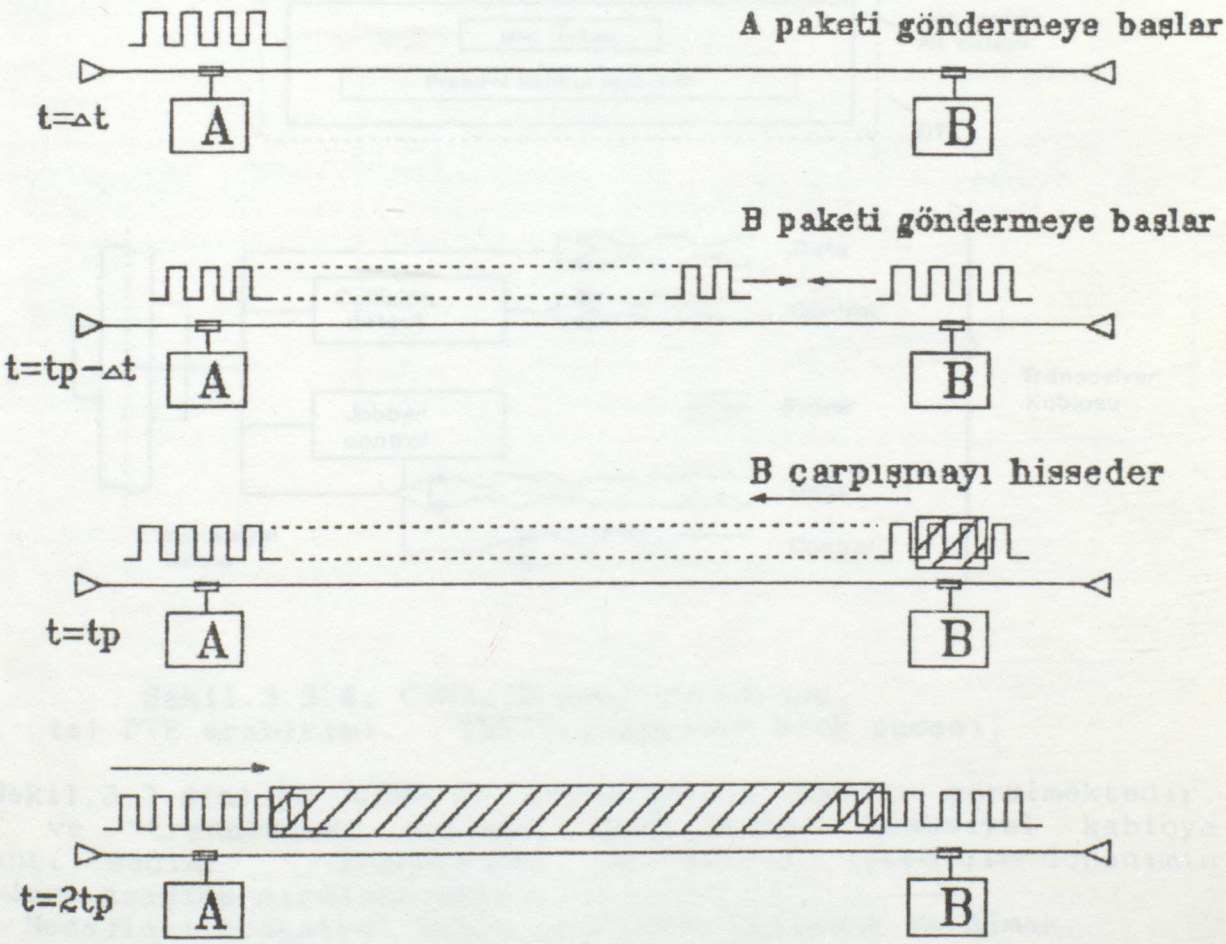


Şekil.3.3.4: Geniş çaplı CSMA/CD bus konfigurasyonu.

Tarihsel gelişme nedeniyle CSMA/CD ağlar "**Ethernet**" olarak da bilinir. **Şekil.3.3.4**'te örnek bir CSMA/CD (Ethernet) ağı görülmektedir. Aşağıda CSMA/CD erişim mantığı bilgisayar diline benzer şekilde kısaca açıklanmıştır:

```
if MESSAGE_READY=TRUE                          ; Mesaj hazırsa
     then
       while BUS_BUSY=TRUE then                ; Hattı kontrol et
          DEFER                                ; İptal et
       end while
       while MESSAGE_BUFFER_FULL=TRUE          ; Mesajı gönder
          OUTPUT_MESSAGE_BUFFER
          if COLLISION=TRUE then               ; Çarpışma olursa
                        wait RANDOM_TIME       ; Bekle
          end if                               ; Tekrar gönder
       end while
end if
```

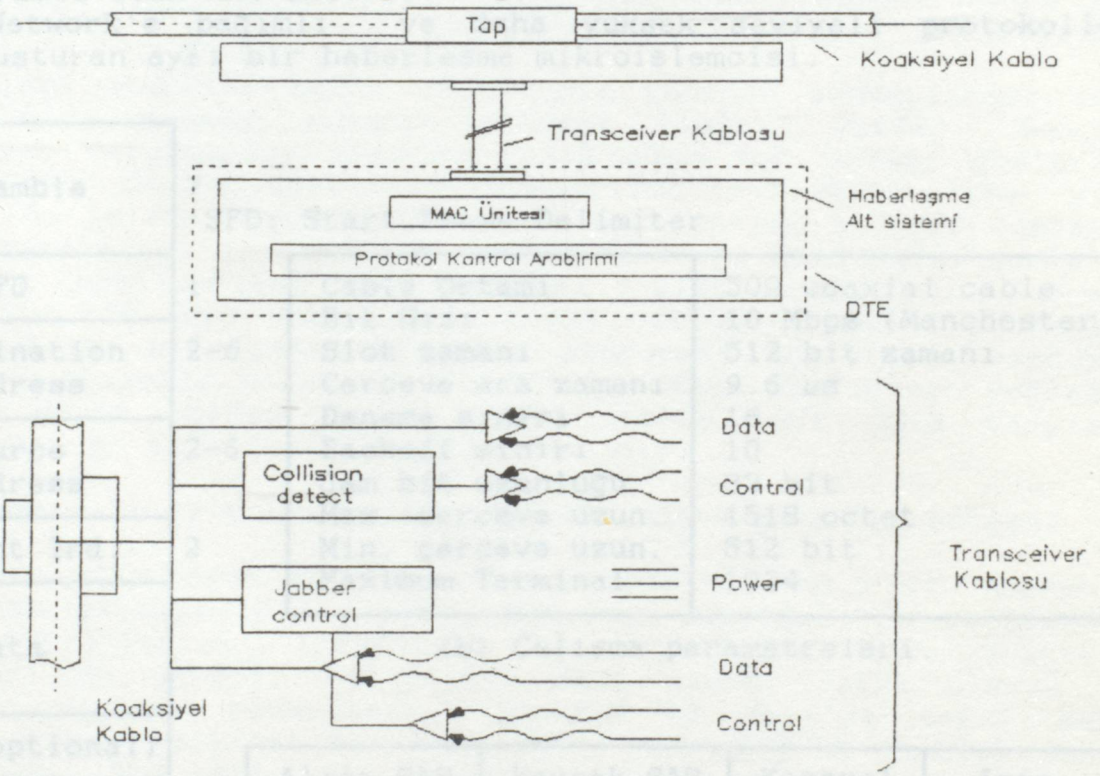
CSMA/CD protokolüyle haberleşmede DTE tarafından gönderilen data önce gerekli alıcı adresi eklenerek paketlenir ve kabloya gönderilir (broadcasting). Kabloya bağlanan bütün terminaller bu paketi alır, header alanındaki adresi kendi adresiyle karşılaştırır. Eğer adres kendi adresi ise paketi almaya devam eder. Verici DTE'nin adresi de paketin içinde bulunduğundan alıcı DTE vericiye direkt olarak yanıt gönderebilir. Bu tip çalışmada kolayca görülebileceği gibi iki terminal aynı anda paket gönderdiği zaman ortak haberleşme ortamı nedeniyle her iki pakette bozulacaktır. Bunu engellemek için verici DTE önce hattı "dinler" ve başka bir paketin gönderilip gönderilmediğini kontrol eder. Eğer taşıyıcı hissedilirse (CR-Carrier Sense), DTE taşıyıcının hattan kalkmasına kadar kendi paketini bekletir. Bu durumda bile iki terminal hattın boşalmasını bekliyor olabilir ve hat boşaldığı anda her ikisi de paketlerini yollayacağından her iki paketin içeriği birbirine girer ve "çarpışma (collision)" denilen bu olay sonucunda her iki paket de bozulur. Bu olay diagram olarak Şekil.3.3.5'te gösterilmiştir.



Şekil.3.3.5: CSMA/CD'de çarpışma olayı.

Çarpışma olasılığını azaltmak için DTE sürekli olarak hattı izler ve gönderdiği paketle okuduğunu bilgiyi karşılaştırır. Eğer farklılık varsa çarpışma detekte edilmiş (**CD-collision detection**) olur. Diğer terminallerin de bu çarpışmayı izlediğinden emin olmak için kısa bir süre rasgele bit dizileri göndererek datanın bozulmasını sağlar. Buna "**Jam sequence**" denir. Bundan sonra iki ya da fazla terminal rasgele aralıklarla tekrar paketleri göndermeye çalışırlar.

CSMA/CD yönteminde haberleşmenin tamamen olasılığa ve network (kablo) trafiğinin yoğunluğuna bağlı olduğu kolaylıkla görülebilir. Haberleşme hızının oldukça yüksek olduğu (10 Mbps) düşünülürse network yoğunluğunun, dolayısıyle çarpışmaların olasılığının çok düşük olduğu da gözönüne alınmalıdır.



Şekil.3.3.6: CSMA/CD konfigurasyonu.
(a) DTE arabirimi. (b) Transceiver blok şeması.

Şekil.3.3.6(a)'da CSMA/CD arabiriminin yapısı görülmektedir. "Tap" ve "**transceiver unitesi (MAC unit)**" koaksiyel kabloya bağlantı sağlar. Transceiver unitesinin içerdiği donanımın görevleri aşağıda sıralanmıştır:

- Mesajları koaksiyel kablo üzerinden yollamak ve almak,
- Kabloda oluşan çarpışmaları hissetmek,
- Koaksiyel kablo ile arabirim arasında elektriksel izolasyon sağlamak,
- DTE'de veya arabirimde oluşacak arızanın kabloyu etkilemesini engellemek.

Yapılması gereken bir başka fonksiyon da "**Jabber control**" denilen ve bozuk bir DTE'nin gönderdiği rasgele işaretlerin asıl haberleşme işaretlerini bozmasını hissetmektir. Belirli bir süre maksimum uzunlukta paketlerin haberleşmesi bozuluyorsa bu kontrol ile haberleşme gerekirse durdurulur.

Şekil.3.3.6(b)'den de görüleceği gibi transceiver unitesi DTE'ye beş çift kabloyla bağlanır. Bu kablonun uzunluğu max. 50m olabilir. Haberleşme kartı bilgisayarın slotlarından birinde takılıdır ve özel bir programla network işletim sistemiyle data alışverişi yapar. Haberleşme kartında aşağıdaki bloklar bulunur:

- Çerçevelerin oluşturulması ve ayrıştırılması, hata saptama ve düzeltme işlemleri ve MAC algoritmasının kurulmasından sorumlu olan MAC unitesi.
- Network'e bağımlı ve daha yüksek seviyeli protokolleri oluşturan ayrı bir haberleşme mikroişlemcisi.

Preamble	7
SFD	1
Destination Address	2-6
Source Address	2-6
Lenght Ind.	2
Data	
Pad (optional)	
Frame Check Sequence	4

SFD: Start Frame Delimiter

(a) Çerçeve formatı

Cable Ortamı	50Ω coaxial cable
Bit Hızı	10 Mbps (Manchester)
Slot zamanı	512 bit zamanı
Çerçeve ara zamanı	9.6 μs
Deneme sınırı	16
Backoff sınırı	10
Jam bit uzunluğu	32 bit
Max. çerçeve uzun.	1518 octet
Min. çerçeve uzun.	512 bit
Maximum Terminal	1024

(b) Çalışma parametreleri.

Alıcı SAP	Kaynak SAP	Kontrol	Info
1	1	1	0-M

(b) LLC Control çerçevesi.

Şekil.3.3.7: CSMA/CD Bus karakteristikleri.

Çerçeveler iletileceği zaman çerçeve içerikleri MAC unitesi tarafından **Şekil.3.3.7(a)**'daki gibi düzenlenir. **Preamble** 7 octet'lik bir alandır ve **PLS (physical level signalling)** devreleri arasında senkronizasyon sağlamak için kullanılır.

Haberleşme ortamındaki diğer paketlerle çarpışmanın engellenmesi için MAC unitesinin ortama erişim bölümü önce taşıyıcı işareti hisseder ve gerekliyse çerçeveyi bekletir. Kısa bir süre **(Çerçeve arası boşluğu (Interframe gap) olarak da bilinir)** bekledikten sonra çerçeve adreslenen alıcı DTE'ye gönderilir. Manchester kodu ile kodlanmış bit dizisi önce bağlantı kablosuyla transceiver unitesine gönderilir, daha sonra kablo üzerinde taşımak için uygun formata sokulup gönderilir.

Bit dizisi kablo üzerinden gönderilirken transceiver sürekli olarak kabloda giden işareti izler ve çarpışmanın olup olmadığını hisseder, FCS alanı da yerine vardıktan sonra yeni bir çerçevenin iletilmesi için beklenir. Eğer çarpışma olmuşsa transceiver "çarpışma hissetme işareti"ni hemen set eder. Bundan sonra MAC unitesi diğer DTE'lerin de çarpışmayı hissetmesini garantilemek için rasgele bit dizisi gönderir.

Çarpışma olduktan sonra çerçevenin yeniden gönderilmesi için belirli bir **"deneme sınırı (attempt limit)"** vardır. Sürekli tekrarlanan çarpışmalar trafiği fazla olan bir ortamı gösterir, MAC unitesi de çerçevelerin gönderilme sıklığını buna göre ayarlar. Bu işleme **"truncated binary exponential backoff (delay)"** denir. Rasgele bit dizileriyle deneme sınırına daha erişilmemiş ve haberleşme kesilmemişse MAC unitesi **"slot zamanı"**nın rasgele birkaç katı bir süre bekledikten sonra çerçeveyi tekrar gönderir. DTE haberleşme sırasında **"çarpışma penceresi (collision window)"** denilen ve preamble alanının ilk bitlik zaman aralığının yaklaşık iki katı olan bölümü izler. Slot zamanı, DTE için çarpışma olasılığının en az olduğu maksimum zamandır:

Slot zamanı = (2 x transmisyon gecikmesi) + güvenlik bölgesi

Burada transmisyon gecikmesi çerçevenin bir vericiden herhangi bir alıcıya gitmesi için gereken maksimum zamandır. Bu gecikmenin içinde tekrarlayıcı (repeater) ve yönlendiriciler (router) gecikmeleri de vardır. Kullanılan slot zamanı octet olarak en yakın bit hızına yuvarlanır. Örneğin 10 Mbps'da temel band koaksiyel kablo kullanan bir network'te max. 2.5 km uzaklık varsa slot zamanı 512 bit (64 octet) olacaktır. n'inci gönderme denemesinde slot zamanı uniform dağılımlı rasgele bir sayıdır (R) ve $0 \leq R \leq 2^k$ aralığındadır. k fonksiyonu ise **k=min (n, gecikme sınırı)** olarak verilir. Bu algoritma aşağıda kısaca açıklanmıştır:

1) İlk çarpışmada terminal 0 veya 1 sayılarından birini seçer, bu sayı kadat slot zamanı bekledikten sonra yeniden gönderir.
2) İkinci çarpışma olmuşsa rasgele sayının aralığı ikiye katlanır (k=2) ve slot zamanı 0,1,2,3 sayılarından biriyle çarpılarak bekleme süresi bulunur.
3) Her ardışıl çağırmada rasgele sayı bir öncekinin iki katı alınarak bu aralık içinden seçilir.

Bu yöntem trafik yoğunluğunun değişmelerine hemen adapte olabildiğinden standart olarak kullanılmaktadır.

Network kablosuna bağlı her DTE'de MAC unitesi içindeki alıcı elektronik devre önce göndericiden gelen işarete bakarak "taşıyıcı hissetme (carrier sense)" işaretini "on" yapar. Gelen preamble alanı senkronizasyonu sağlamakta kullanılır. Senkron sağlandıktan sonra Manchester koduyla kodlanmış data normal binary forma dönüştürülür. Daha sonra preamble ve start-of-frame sınırlayıcı alanı çerçeveden atılır. Alıcı ve verici adresleri de daha sonra kullanmak üzere çerçeve tampon belleğine (frame buffer) yüklenir. MAC unitesi tarafından hesaplanan FCS ile paketin içindeki FCS karşılaştırılır. Eğer bir farklılık yoksa alınan çerçevenin tampon bellekteki başlangıç adresi bir üst protokol seviyesine gönderilir. Bu kontrollardan başka çerçevenin 8 bitlik octet'lerin tam katları uzunlukta olup olmadığı ve uzunluk sınırlarının aşılıp aşılmadığıgibi kontrollar da yapılır. Eğer çerçeve bu kontrollardan herhangi birinden geçmiyorsa paket iptal edilir ve hata mesajı network seviyesine iletilir.

MAC Frame şeklinde görülmektedir. "**Preamble**" 7-octet'lik bir alandır, ve "**PLS (Physical Level Signalling)**" devreleri anında senkronizasyon sağlamakta kullanılır. "**Start Frame Delimiter**" alanı **10101011** bit dizisinden alınan bir alandır ve geçerli çerçevenin başladığını belirtir. Bundan sonra alıcı ve verici adres alanları vardır. Her iki adres de 2 veya 6 octet'tir. Fakat her iki adreste aynı uzunlukta olmalıdır. **Destination** adresi tek bir alıcı veya grubu temsil edebilir. İki çeşit grup adresi vardır. Multicast-group adresi mantıksal olarak bağlantılı terminaller ve broadcast adres, bütün terminalleri grup oluşturan adrestir ve bu adresler tektir.

Lenght alanı 2 octet'liktir ve LLC data uzunluğunu gösterir. Data alanı kendi LLC protocol data unitesini kapsar. Eğer data alanı uzunluğu alıcı SAP, verici SAP ve data protocol için gerekli minimum uzaklıktan az ise data alanı "**PAD**" denilen ekstra octet'lerle genişletilir. Örneğin 10 Mbps temel band bir sistemde çerçeve uzunluğu 512 byte'dir. Son olarak FCS alanı alıcı ve verici adreslerine, LLC data ve pad alanlarına göre 32-bit CRC hesabını yapar.

MAC çerçeve yapısı orijinal Ethernet spesifikasyonlarına çok benzer. Ethernet spesifikasyonlarında adres alanı mutlaka 6 octet olmalıdır. Ayrıca lenght alanı yerine type alanı vardır ve yüksek seviyeli protokoller tarafından anlaşılması için 2 octet'lik bilgi vardır.

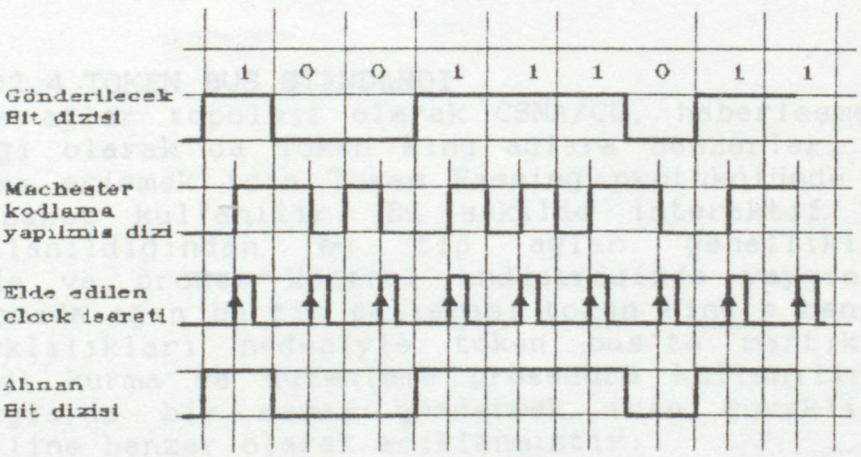
MAC alt seviyesinin fonksiyonları data link seviyesi protokolündeki fonksiyonlarla aynıdır. MAC alt seviyesi datanın çerçevelenmesi, adreslenmesi ve hata saptama işlemleri yapar. Bundan başka haberleşme ortamına erişimin düzenlenmesi ve çarpışmaları önleyici mekanizmalar kurar.

CSMA/CD protokolünde MAC alt seviyesi LLC'den gelen datayı alıp çerçeveyi oluşturur. Fiziksel seviyeden taşıyıcı hissetme (Carrier Sense) sinyali gelir. Bu sinyal "on" durumundaysa çerçeve bekletilir. Fiziksel ortam boşalmışsa belli bir gecikmeden sonra (interframe gap:10 Mbps için 9.6 μs) çerçevenin iletilmesi gerçekleşir.

Haberleşme sırasında da fiziksel seviye ortamı izler ve eğer çarpışma olmuşsa çarpışma hissetme (Collision Detection) işareti "on" yapılır. Eğer birden fazla terminal aynı anda çerçevelerini göndermeye çelışmışsa çarpışma (collision) olur. Terminal transmisyonu belli bir süre (collision window) izleyebilir.

MAC alt seviyesi **PLS servis spesifikasyonları** aracılığıyla PLS alt seviyesiyle haberleşir. DTE'nin data link ve daha yüksek seviyeli protokolleri transmisyon ortamına **MAU (Medium Attachment Unit)** aracılığıyla bağlanır. DTE ile MAU ise **AUI (Attachment Unit Interface)** standardında belirlenen kablo ile birbirlerine bağlanırlar. AUI, DTE'nin transmisyon ortamının temel band veya geniş band olmasından bağımsız çalışmasını sağlar. AUI 1,5,10 veya 20 Mbps gibi data hızlarını destekleme yeteneğine sahiptir, AUI kablosu 50 metre uzunluğa çıkabilir ve bağlantı 15 bacaklı standart konnektorlerle yapılır.

AUI üzerinden data akışı Manchester kodlama yöntemiyle yapılır. Manchester kodlama data ve clock işaretlerini tek bir bit dizisinde birleştiren bir kod sistemidir. Manchester kodlama mantığı **Şekil.3.3.8**'de görülmektedir.



Şekil.3.3.8: Manchester Kodlama yöntemi.

Manchester kodlamada data ve clock işaretleri faz kodlamasıyla birleştirilir. Her bit için ayrılan zamanın ortasındaki pozitif ve negatif geçişlerde clock işareti elde edilir, bit zamanının ilk yarısında darbenin durumu ise datayı gösterir.

MAC (Medium Attachment Unit) transmisyon ortamına göre seçilir. Temel bandda çalışan MAC orijinal Ethernet spesifikasyonlarına uygundur. İşaretleşme hızı 10 Mbps'dir ve bir segmentin boyu 500 metreye kadar çıkabilir. Segmentler tekrarlayıcılarla (repeater) birbirine bağlanabilirler. Kullanılan kablolar işaretlerin yansımasını önlemek için 2.5 metrenin katları olmalıdır. Bir segmentteki MAU sayısı 100'ü geçemez. Koaksiyel kablonun bir ucundan topraklanması gerekir.

Geniş band MAU için bir standartlaşma şimdiye kadar tamamlanamamıştır. Geniş band yerel iletişim ağları maksimum 300 Mhz band genişliğine sahip **CATV** (Cable TV) koaksiyel kablosu üzerine kurulurlar.

Geniş band ağlarda segmentler ağaç yapısına benzer topolojiler kullanır. Ağacın köküne "**head-end**" denir. İşaretler kabloya direkt olarak "tap"lardan uygulanır. Transmisyon yönü için frekans bandının alt ve üst yarı bandları kullanılır. Mid-split sistemlerde alt yarı band gelen işaretler, üst yarı band giden işaretler için kullanılır. "**Remodulator**" head-end'lerde bulunur ve gelen sinyalleri frekans bandında kaydırır. Böylece bir MAU'nun alıcı ve vericisi ayrı frekanslarda çalışabilir.

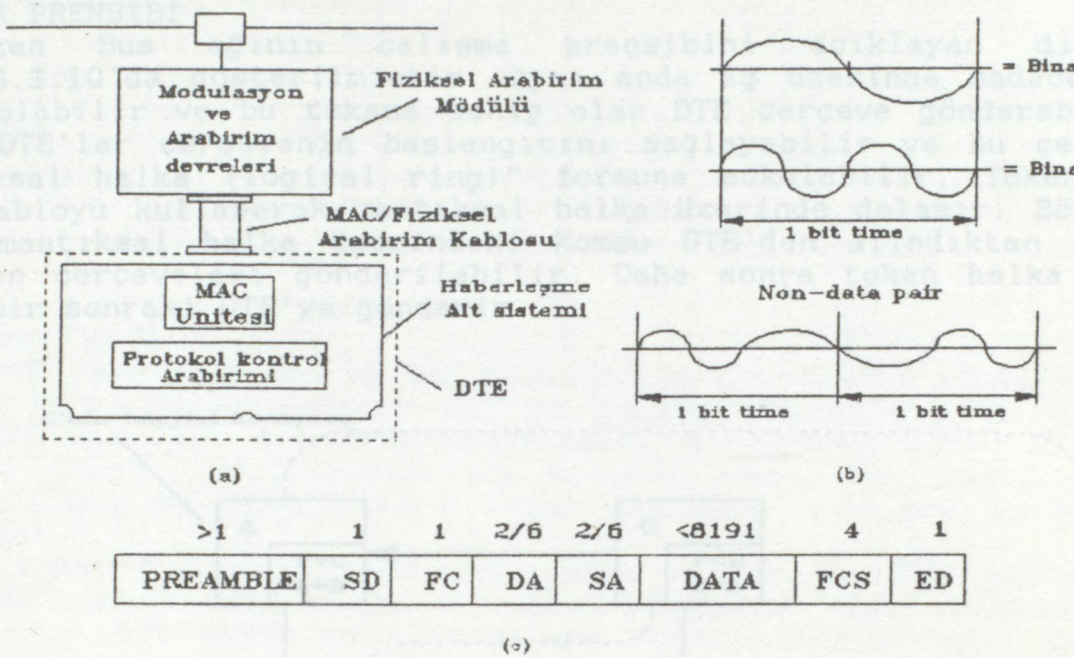
Geniş band sistemlerde 75 Ohm koaksiyel kablo kullanılır. Bir sistem tasarlanırken işaret seviyeleri kablo ve tap'lardaki zayıflamaya göre hesaplanır ve gerekirse periyodik olarak yükselticiler eklenir. Geniş band sistemlerde işaret seviyeleri hattın her yerinde farklı olduğundan çarpışma deteksiyonu işaretin seviyesi ölçülerek değil, bit karşılaştırılması yapılarak elde edilir. Bu yönteme "**LWT-CSMA (Listen While Talk CSMA)**" denir. Gönderici MAU gönderdiği çerçevenin bir kopyasını bellekte tutar ve aldığı çerçevelerle karşılaştırır. Eğer bir farklılık yoksa başka bir DTE'nin göndermediği anlaşılır. Geniş band sistemlerde data hızları 1-10 Mbps arasında değişmektedir ve frekans bandı 6MHz'dir.

3.3.4 IEEE 802.4 TOKEN BUS STANDARDI

Token Bus ağlar topoloji olarak CSMA/CD, haberleşme ortamına erişim mantığı olarak da Token Ring ağlara benzerler. Token bus ağlarda ortama erişmek için Token Passing protokolünde token denilen özel paket kullanılır. Bu şekilde interaktif haberleşme mantığı kullanıldığından bu tip ağlar genellikle üretim teknolojisinde ve proses kontrol endüstrisinde yaygınlaşmıştır. Normal çalışmada ağın bu tip çalışması token ring'e benzer. Fakat topoloji farklılıkları nedeniyle token bus'ta mantıksal halka (logical ring) kurma ve düzenleme prosedürü kullanılır. Aşağıda Token Bus ağlarda bir mesaj göndermek için gerekli prosedür bilgisayar diline benzer olarak açıklanmıştır:

```
while TOKEN=TRUE and TOKEN_HOLD_TIMEOUT=FALSE
   while MESSAGE_BUFFER_FULL=TRUE
      OUTPUT MESSAGE                    ; mesajı gönder
   end while
   OUTPUT TOKEN                         ; token'i sıradaki başka
end while                               ; bir DTE'ye gönder
```

Şekil.3.3.9'da token bus ağların çalışması için gerekli elemanlar, dalga şekilleri ve çerçeve formatları görülmektedir. Haberleşme ortamı olarak normalde koaksiyel kablo kullanılır, geniş band ve "**Carrierband**" denilen düzeltilmiş temel band kullanılır.



Şekil.3.3.9: Token Bus ağ elemanları. (a) DTE arabirim devresi
(b) "Carrierband" kodlama
(c) Çerçeve formatı

Şekil.3.3.9(a)'da görülen modülasyon ve arabirim devresi aşağıdaki fonksiyonları yerine getirir:

- Gönderilen bilgiyi kodlama (modülasyon),
- Alınan bilginin kodunun çözülmesi (demodülasyon),
- Clock işaretinin üretilmesi.

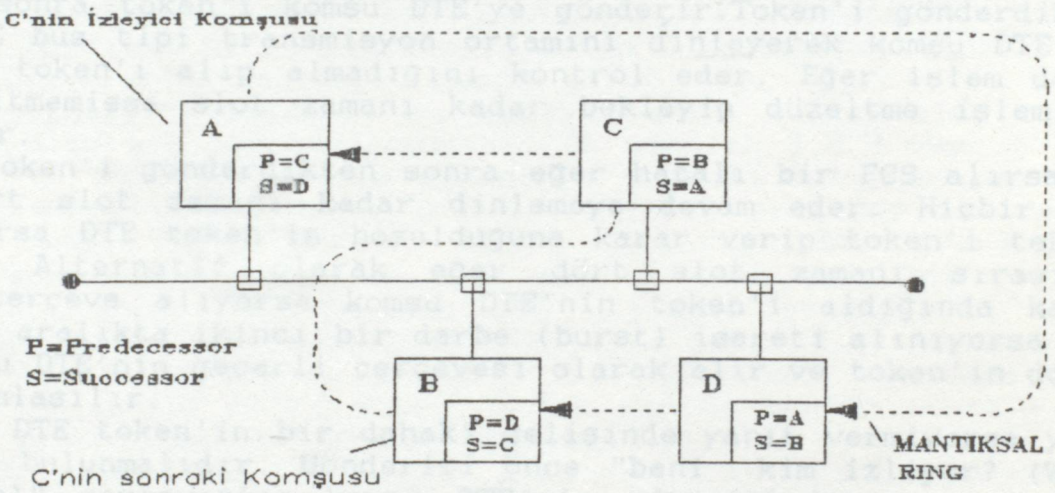
DTE ile haberleşme ortamı arasında "**Fiziksel arabirim modülü (PIM-Physical Interface Modul)**" denilen standart bir arabirim vardır. Bu arabirim DTE'ye bağlanan bir haberleşme kartı şeklindedir.

Carrierband modunda çalışma prensibi **Şekil.3.3.9(b)**'de görülmektedir. Temel bandda olduğu gibi carrierband'da kablonun bütün bandgenişliği tek bir haberleşme için harcanır, fakat gönderilmeden önce **Frekans Kaymalı Anahtarlama "FSK-(Frequency Shift Keying)** yöntemiyle modüle edilir. Şekilden de görüldüğü gibi binary 1 sinusoidal bir işaretle (normalde 1 ve 5Mbps), binary 0 ise aynı frekansın iki katı frekansta gönderilir. Bit sınırlarında faz değişimi olmadığına dikkat edilmelidir, bu duruma "**faz coherent**" denir. Temel band işaretinin birçok harmoniği olmasına karşın carrierband işaretinin iki bileşeni vardır. Gürültü işaretlerini ayırmak için filtre kullanılarak bu iki bileşen temiz olarak alınabilir ve sistemin gürültü bağışıklığı oldukça arttırılır.

Token bus için çerçeve formatı **Şekil.3.3.9(c)**'de görülmektedir. Çerçeve formatı hemen hemen token ring'le aynıdır. Fakat token ring'de SD ve ED alanlarında bulunan J ve K bitleri özel işaretlerle (non-data pair) yer değiştirmiştir.

ÇALIŞMA PRENSİBİ

Token Bus ağının çalışma prensibini açıklayan diagram Şekil.3.3.10'da gösterilmiştir. Aynı anda ağ üzerinde sadece bir token olabilir ve bu tokene sahip olan DTE çerçeve gönderebilir. Bütün DTE'ler çerçevenin başlangıcını sağlayabilir ve bu çerçeve "mantıksal halka (logical ring)" formuna sokulabilir. Token bus tipi kabloyu kullanarak mantıksal halka üzerinde dolaşır. Böylece token mantıksal halka üzerindeki komşu DTE'den alındıktan sonra bekleyen çerçeveleri gönderilebilir. Daha sonra token halka üzerinde bir sonraki DTE'ye gönderir.



Şekil.3.3.10: Token Bus ağlarda çalışma prensibi.

Halkayı düzenleyen prosedürleri daha ayrıntılı incelemeden önce token bus ağların iki temel özelliği incelenecektir. Bunlardan birincisi bir DTE çerçeve gönderdiği zaman, bütün DTE'ler aynı transmisyon ortamını paylaştıklarından çerçevenin bütün DTE'ler tarafından alınmasıdır. İkincisi ise, DTE'nin gönderdiği çerçevenin yanıtını beklemesi için gerekli sürenin bir sınırı olmasıdır. Bunun nedeni alıcı terminal bozulabilmesi veya transmisyon ortamından dolayı çerçeve kaybolması gibi durumlarda ağın kilitlenmesinin önüne geçmektir. Bu zaman "slot zamanı" olarak bilinir ve aşağıdaki gibi tanımlanır:

Slot zamanı= 2 x (transmisyon hattı gecikmesi + işlem gecikmesi)

Burada transmisyon hattı gecikmesi ağ üzerindeki herhangi bir vericiden gönderilen çerçevenin herhangi bir alıcıya gitmesi için gereken maksimum süredir, işlem gecikmesi ise MAC unitesi için alınan çerçevenin yanıtının hazırlanması için geçen süredir. Bunlara bir de güvenlik süresi de eklenerek bit sayısı 8 bitin katlarına yuvarlanır.

Normal çalışma koşullarında token bir DTE'den diğerine mantıksal yol üzerinden kısa bir token çerçevesiyle iletilir. Her DTE halkada kendinden sonra gelen DTE'nin adresini bilmelidir. Eğer DTE token'i alamazsa gönderici DTE tokeni kurtarma ve başka bir komşu DTE bulma prosedürlerine sahiptir. Bundan başka halkanın doğru çalışması için gerekli düzeltmeler, halkaya yeni DTE ekleyip çıkarmak için prosedürler vardır. Token ring'teki gibi token'e öncelikler atanabilir, fakat normalde her token'in önceliği tektir.

Token Passing Mantığı

Token'i aldıktan sonra, DTE bekleyen çerçevelerini gönderebilir ve daha sonra token'i komşu DTE'ye gönderir.Token'i gönderdikten sonra DTE bus tipi transmisyon ortamını dinleyerek komşu DTE'nin aktif ve token'i alıp almadığını kontrol eder. Eğer işlem doğru olarak bitmemişse slot zamanı kadar bekleyip düzeltme işlemleri yapmalıdır.

DTE token'i gönderdikten sonra eğer hatalı bir FCS alırsa en fazla dört slot zamanı kadar dinlemeye devam eder. Hiçbir şey duyulmuyorsa DTE token'in bozulduğuna karar verip token'i tekrar gönderir. Alternatif olarak eğer dört slot zamanı sırasında geçerli çerçeve alıyorsa komşu DTE'nin token'i aldığında karar verir. Bu aralıkta ikinci bir darbe (burst) işareti alınıyorsa DTE bunu komşu DTE'nin geçerli çerçevesi olarak alır ve token'in doğru geçtiği anlaşılır.

Komşu DTE token'in bir dahaki gelişinde yanıt vermiyorsa yeni bir komşu bulunmalıdır. Gönderici önce **"beni kim izliyor? (Who-follows-me)"** çerçevesine komşu DTE'nin adresini koyar ve gönderir. Bu çerçeveyi alan bütün DTE'ler çerçevenin data alanındaki adresle kendisine komşu olan DTE'lerin adresleriyle karşılaştırır. Daha sonra adresi aynı olan DTE kendi adresini **set-successor** çerçevesine koyarak yanıt verir. Böylece bu DTE token'i alır ve bozulan DTE köprülenmiş olur.

Gönderici DTE who-follows-me çerçevesine yanıt alamıyorsa çerçeveyi tekrar gönderir. Eğer hala yanıt yoksa DTE, DA alanında kendi adresi bulunan bir **"bekleyen-aday (solicit-successor)"** çerçevesi gönderir. Eğer herhangi bir DTE buna yanıt verirse **"yanıt penceresi (response window)"** denilen bir prosedürle mantıksal halka yeniden kurulur. Alternatif olarak eğer hiçbir yanıt alınamıyorsa diğer bütün DTE'lerin, haberleşme ortamının bozuk veya DTE'nin kendi alıcısının bozulduğuna karar verilir. Bu koşullarda DTE haberleşme düzelene kadar dinleme konumunda kalır.

YANIT PENCERESİ (Response Window)

Bu prosedür yeni DTE'lerin mantıksal halkaya katılması için rasgele bir zaman sağlar. Yanıt zamanı aslında DTE'nin çerçeveyi gönderdikten sonra beklediği zamandır ve slot zamanına karşı düşer. DTE tarafından gönderilen her bekleyen çerçeve bir SA ve DA belirler ve halkaya girmek isteyen DTE'nin adresi bu iki adres arasındadır. Her DTE token'in sahibi olduğu zaman **solicit-successor** çerçevesini gönderir.

DTE bu çerçeveyi gönderdiği zaman "açık" bir yanıt penceresi olduğu söylenir ve DTE yanıt penceresi periyodu kadar bir süre yanıt bekler. Eğer bir DTE'nin adresi SA-DA adresleri arasında ise verici DTE'ye yeni aday olduğunu belirten bir istek çerçevesi gönderir. Gönderici bu yanıt çerçevesini alırsa yeni DTE'yi aday yapıp token'i bu DTE'ye gönderir. Adres bölgesi birden fazla DTE'yi içerdiğinden bunlardan geri dönen yanıt çerçeveleri de bozulabilir. Bu durumda DTE tek bir yanıtı kabul eder.

ŞARTLANDIRMA (Initialization)

Şartlandırma prosedürü aslında yanıt penceresi prosedürü ile başlar. Ağdaki her DTE hattaki transmisyonu gösterir ve yeni bir haberleşme olduğu zaman bir zamanlayıcıyı **(inactivity timer)** başlatır. Eğer DTE normal çalışmada token'i kaybederse zamanlayıcı durur ve DTE yeniden başlangıç fazına girer.

Her potansiyel başlatıcı enformasyon alanı birkaç slot zamanı uzunluklu olan bir çerçeve gönderir. Bu uzunluk 0,2 4 veya 6 katı olabilir, ve DTE'nin network adresinin ilk iki bitinden seçilir. Çerçeve gönderildikten sonra DTE transmisyon ortamını dinlemeden önce bu zaman kadar bekler. Hattı dinlerken bir transmisyon duyarsa başka bir DTE'nin de aynı tür ve daha uzun bir çerçeve gönderdiğini anlar. Bu durumda DTE başlatma prosedürünü iptal eder. Eğer hiçbir transmisyon duyulmuyorsa adres alanının sonraki iki biti kullanılıp işlem tekrarlanır, yukarıdaki olaylara göre DTE token'in sahibi olur. Token'in tek sahibi bundan sonra yanıt penceresi prosedürünü kullanarak başlatma işlemine devam eder.

DTE'nin mantıksal halkadan çıkması için token kendisine geldiğinde yanıt vermemesi yeterlidir, ayrıca DTE'nin ayrılması için diğer bir yöntem de DTE'nin token geldiğinde bir "set-successor" çerçevesi göndererek kendi komşusunu halkada kendi yerine yerleştirmesidir.

ÖNCELİK (PRIORITY) İŞLEMİ

Token Ring ağlarda olduğu gibi Token Bus ağlarda da öncelik mekanizması kurmak mümkündür. Token Bus'taki öncelik seviyeleri **"erişim sınıfları (access classes)"** denilen ve 0, 2, 4, ve 6 (en üst) olarak adlandırılan öncelik sıralarıdır.

Daha önce de bahsedildiği gibi token bus ağlar üretim otomasyonu ve proses kontrolu gibi uygulamalarda kullanıldığından bu seviyelerin tipik kullanımı aşağıdaki gibidir:

- **6.sınıf**:Kritik alarm ve buna bağlı kontrol durumlarında,
- **4.sınıf**:Halkanın düzenli çalışması için gerekli normal işlemler,
- **2.sınıf**:Data toplama ve işlemeyle ilgili mesajlar,
- **0.sınıf**:Program yükleme ve dosya transferiyle ilgili mesajlar, düşük öncelikli uzun mesajlar.

Gönderilmeyi bekleyen çerçevelerden en yüksek önceliğe sahip olan çerçeve DTE'ler tarafından paylaşılan halka kapasitesi ile kontrol edilir. DTE token'i aldığında bekleyen yüksek öncelikli çerçevelerini **"yüksek öncelikli token-tutma zamanı (high-priority token-hold time)"** denilen bir zaman aralığında gönderir. Daha sonra bu zaman geçtikten sonra token bir sonraki DTE'ye gönderilir. Bu DTE'de kendi yüksek öncelikli çerçevesini gönderir, yukarıdaki zaman aralığı dolmamışsa bir alt seviyede önceliğe sahip olan çerçeveyi de gönderir.

Mantıksal halkadaki her DTE en son token'i aldığı zamandan itibaren çalışan zamanlayıcıyla bu süreyi belirler. Bu değer **"token dönme zamanlayıcısı (TRT-token rotation timer)"** de tutulur. DTE token'i aldığında önce bekleyen çerçevesini gönderir, daha sonra TRT değerini bir arttırır ve **"sabit token dönme zamanı (TTRT-Target TRT)"** ile karşılaştırır. Eğer TRT-TTRT pozitifse DTE bir çerçeve daha gönderebilir, bu fark negatifse DTE token'i komşu DTE'ye göndermek zorundadır. Böylece DTE'lerin mesaj yoğunluğuyla değişen dinamik bir erişim mantığı kurulmuş olur.

3.3.4 IEEE 802.5 TOKEN RING STANDARDI

Token Ring yerel iletişim ağları IBM tarafından öncelikle teknik amaçlarla ve ofis ortamlarında kullanılmak için geliştirilmiştir. Aşağıda Token Ring protokol mantığı bilgisayar diline yakın biçimde gösterilmiştir.

```
while MESSAGE_BUFFER_FULL=TRUE
  input MESSAGE                              ; Mesajı al
  if MESSAGE=TOKEN then                      ; Mesaj token ise
          CONVERT TOKEN TO CONNECTOR         ; Token'i konnektor'e
  end if                                     ; dönüştürme işlemini yap
  output MESSAGE
end while
output TOKEN                                 ; Token'i gönder.
```

Bir DTE çerçeve göndereceği zaman önce token'i bekler. Token'i aldıktan sonra içinde istenen alıcının adresi bulunan çerçeveyi oluşturur. Daha sonra çerçeve bir sonraki DTE'ye tekrarlanır. yeni gelen her yeni bit alınıp aynen gönderilir. Çerçeve bütün DTE'leri dolaştıktan sonra gönderici DTE'de bu çerçeve halkadan atılır. Çerçeve gönderilirken bir kopyası da DTE'de saklanır ve bu çerçevenin arkasında **"yanıt biti (response bit)"** ile gösterilir. Çerçevenin en son biti de gönderildikten sonra onu ilk gönderen DTE yeni bir boş token yaratıp hatta gönderir.

Tipik bir token ring ağı **Şekil.3.3.12**'da görülmektedir. Burada ayrıca DTE' nin haberleşme ortamına bağlanması için gerekli çeşitli elemanlarda gösterilmiştir. (Trunk) kablosu olarak genelde "shilded twisted pair" kablo kullanılır ve 1 ile 4 Mbps hızında farksal olarak sürülür.

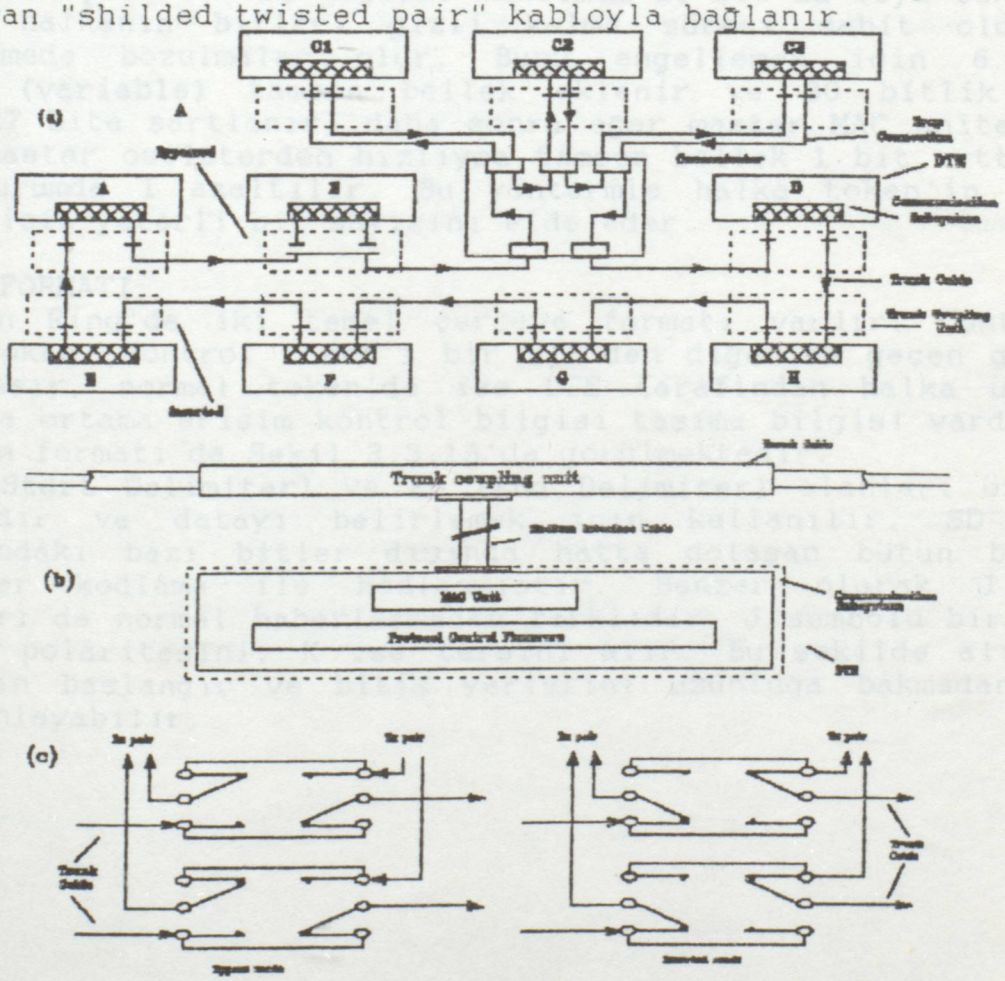
Şekilden de görüleceği gibi DTE direkt olarak halkaya bağlanmak zorunda değildir, **"Toplayıcı (Concentrator)"** denilen bir cihaz aracılığıyla da bağlanabilir. Bu durumda trunk kablosu toplayıcıdan DTE'lere dağılır ve **"Wiring Concentrator"** diye anıldığı da olur.

RING ARABİRİMİ

Transmisyon ortamına bağlantı **TCU (trunk coupling unit)** denilen standart fiziksel arabirimle yapılır. TCU' nun içinde kablodan data alışverişi yapmak için röle ve elektronik devreler bulunur. Röleler öyle ayarlanmıştır ki DTE bağlantısı kopuk bile olsa TCU **"bypass"** duruma geçerek ring haberleşmeye devam eder.

DTE'nin halkaya sokulması DTE içindeki haberleşme denetleyicisindeki MAC unitesi tarafından yapılır. Şekil.3.3.12'den de görüleceği gibi MAC unitesi her iki röleyi de çektirerek DTE' yi ringe dahil eder ve bu alınan bütün işaretlerin MAC unitesine gitmesini sağlar. Eğer DTE gelen işaretin sahibi değilse sadece işaret tekrarlanır.

İki çift röle kullanılması MAC'ın **gönderme (transmit)** ve **alma (receiver)** kablolarında kısa devre veya açık devre hatalarını kesinlikle yakalayabilmesini sağlar. Bundan başka bypass durumunda MAC unit kendi kendini test eder. DTE TCU' ya biri gönderme (transmit) diğeri alma (receiver) için "twisted pair" kablo bulunduran "shilded twisted pair" kabloyla bağlanır.



Şekil.3.3.12: Token Ring ağ elemanları: (a) Halka konfigurasyonu
(b) DTE Arabirimi
(c) TCU iç şeması.

MAC unitesi çerçeve oluşturma ve paketlere ayırma, FCS hesaplama ve hata hesaplama, MAC algoritması kurma gibi fonksiyonlardan sorumludur. Ayrıca eğer DTE aktif halka monitorü ise bu durumda halkanın clock işaretini üretmek de MAC'ın görevidir. Hattı dolaşan bit dizisi aktif monitorde **Manchester** koduyla kodlanır ve halkadaki diğer DTE'ler bu bit dizisini **Faz kilitlemeli çevrim (PLL- Phase-Locked-Loop)** devresiyle elde eder.

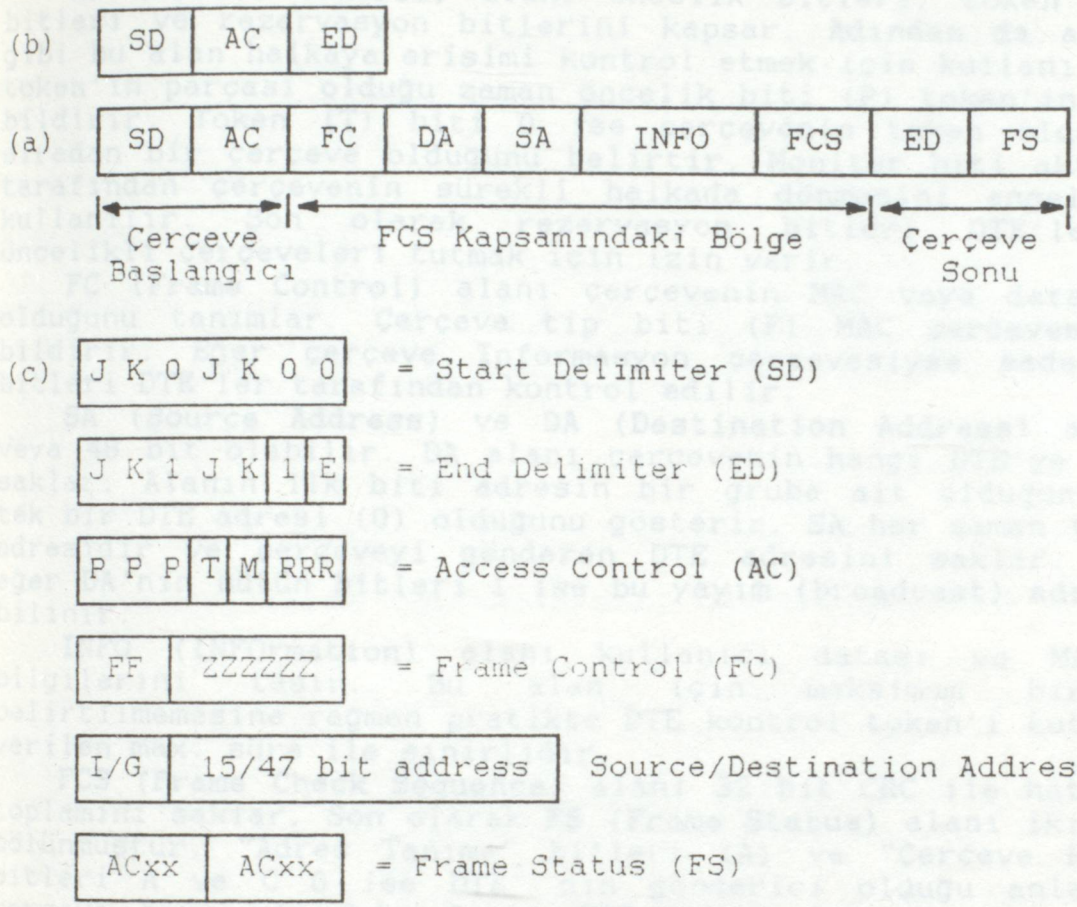
Ek olarak eğer DTE aktif monitor ise, halkanın en az **"varolma zamanı (minimum latency time)"** olduğuna emin olmalıdır, yani bu zaman bit dizisinin halkayı dolaşması için gerekli zamandır. Bu zamana her MAC unitesinin gecikme zamanları da dahildir. Bütün DTE'ler bypass modundayken kontrol token'inin bütün halkayı dolaşması için halkanın minimum varolma zamanı en az token'i oluşturan bitler kadar olmalıdır. Böylece bu zaman yüzünden token'in bozulması engellenmiş olur.

Daha sonra da görüleceği gibi, token'in uzunluğu 24 bittir, bunun için DTE active monitor olduğu zaman MAC unitesi sabit 24 bitlik tampon bellek ayırır ve bütün koşullar altında doğru çalışmayı garantileyerek halkaya katılır. Halkanın bit hızı aktif monitordeki clock işareti ile ayarlanmasına rağmen, her MAC unitesindeki PLL'ler nedeniyle ring üzerindeki bit hızı biraz değişebilir. En kötü durum analizi yapılırsa ve max. DTE (250) olduğu varsayılırsa bu değişme maksimum üç bit az veya çoktur. Bu durumda halkanın bitleri gizli kalma süresi sabit olduğundan haberleşmede bozulmalar olur. Bunu engellemek için 6 bitlik **elastik (variable)** tampon bellek eklenir ve 30 bitlik tampon bellek 27 bite şartlanır, daha sonra eğer master MAC unitesindeki işaret master osilatorden hızlıysa tampon bellek 1 bit arttırılır, tersi durumda 1 azaltılır. Bu yöntermle halka token'in sürekli dönmesi için yeterli bit sayısını elde eder.

ÇERÇEVE FORMATI

Token Ring'de iki temel çerçeve formatı vardır: **Control** ve **Normal** token. Kontrol token'i bir DTE'den diğerine geçen gönderme hakkı taşır, normal token'de ise DTE tarafından halka üzerinde data veya ortama erişim kontrol bilgisi taşıma bilgisi vardır. Her iki tipin formatı da **Şekil.3.3.13**'de görülmektedir.

SD (Start Delimiter) ve **ED (End Delimiter)** alanları özel bit dizileridir ve datayı belirlemek için kullanılır. SD ve ED alanlarındaki bazı bitler dışında hatta dolaşan bütün bilgiler Manchester kodlama ile kodlanmıştır. Benzer olarak J ve K sembolleri de normal haberleşmeden farklıdır. J sembolü bir önceki sembolün polaritesini, K ise tersini alır. Bu şekilde alıcı DTE token'inin başlangıç ve bitiş yerlerini uzunluğa bakmadan kesin olarak anlayabilir.



Şekil.3.3.13: Token Ring çerçeve formatları ve alan tanımlamaları.
(a) Token formatı
(b) Çerçeve formatı
(c) Alan tanımlamaları

Fakat **Şekil.3.3.13**'den de görüleceği gibi sadece ilk 6 sembol geçerli çerçeve için gereklidir. I ve E iki bitinin diğer fonksiyonları vardır:

- Token'de I ve E bitleri 0'dır.
- Normal çerçevede I=1 ise çerçeve dizisinin ilk sırasında, I=0 ise tek veya sonuncu olduğunu gösterir.
- E biti hata saptama amacıyla kullanılır. Ilk DTE tarafından set edilir, fakat herhangi bir DTE hata bulduğunda bu biti 1 yapar ve gönderici DTE bu hatadan haberdar olur.

AC (Access Control) alanı öncelik bitleri, token ve monitor bitleri ve rezervasyon bitlerini kapsar. Adından da anlaşılacağı gibi bu alan halkaya erişimi kontrol etmek için kullanılır. AC bir token'in parçası olduğu zaman öncelik biti (P) token'in önceliğini bildirir. Token (T) biti 0 ise çerçevenin token olduğu, 1 ise sıradan bir çerçeve olduğunu belirtir. Monitor biti aktif monitor tarafından çerçevenin sürekli halkada dönmesini engellemek için kullanılır. Son olarak rezervasyon bitleri DTE'lere yüksek öncelikli çerçeveleri tutmak için izin verir.

FC (Frame Control) alanı çerçevenin MAC veya data çerçevesi olduğunu tanımlar. Çerçeve tip biti (F) MAC çerçevesi olduğunu bildirir. Eğer çerçeve Informasyon çerçevesiyse sadece kontrol bitleri DTE'ler tarafından kontrol edilir.

SA (Source Address) ve **DA (Destination Address)** alanları 16 veya 48 bit olabilir. DA alanı çerçevenin hangi DTE'ye gideceğini saklar. Alanın ilk biti adresin bir gruba ait olduğunu (1) veya tek bir DTE adresi (0) olduğunu gösterir. SA her zaman tek bir DTE adresidir ve çerçeveyi gönderen DTE adresini saklar. Ek olarak eğer DA'nın bütün bitleri 1 ise bu yayım (broadcast) adresi olarak bilinir.

INFO (INFOrmation) alanı kullanıcı datası ve MAC kontrol bilgilerini taşır. Bu alan için maksimum bir uzunluk belirtilmemesine rağmen pratikte DTE kontrol token'i tutarken izin verilen max. süre ile sınırlıdır.

FCS (Frame Check Sequence) alanı 32 bit CRC ile hata kontrolu toplamını saklar. Son olarak **FS (Frame Status)** alanı iki alt alana bölünmüştür. "Adres Tanıma" bitleri (A) ve "Çerçeve Kopyalandı" bitleri A ve C 0 ise DTE' nin gönderici olduğu anlaşılır.Eğer çerçeve bir yada daha fazla DTE tarafından tanınırsa A biti 1 yapılır.bu öekilde gönderici DTE, alıcı(ların) olup olmadığını, kapalı olduğunu, açık olduğunu fakat çerçeveyi kopyalamadığını veya çerçeveyi kopyaladığını anlayabilir.

ÇERÇEVE TRANSMİSYONU

Data gönderileceği zaman MAC unitesi tarafından çerçeve haline getirilir. MAC unitesi daha sonra kendi çerçevesiyle aynı veya daha az öncelikli token bekler. Çerçeve düzenlendikten ve token beklendikten sonra MAC unitesi token' in veya paketin önceliğini MAC alanındaki rezervasyon bitlerinden öğrenir. Eğer öncelik sırası gelen paketten düşükse bunu değiştirmeden geri yollar. Eğer öncelik sırası büyükse yeni öncelik sırasını çerçeveye yerleştirir. Daha sonra daha yüksek öncelikli çerçeve olmadığı varsayılarak token o andaki kullanıcı tarafından gönderilir.

MAC unitesi token'in önceliğinin göndermek için beklettiği çerçeveyle aynı olduğunu anlar ve AC alanının token bitini 1 yapark token'i kabul eder. Böylece token normal çerçevenin bir "start-of-frame" bit dizisi olur. MAC unitesi gelen işareti durdurup kendi hazırladığı çerçeveyi gönderir. Çerçeve gönderilirken bir yandan FCS hesaplanıp çerçevenin sonuna eklenir.

Çerçeve bir kere dolaşmaya çıktıktan sonra MAC unitesi tekrarlamayı bırakır ve çerçeve bütün halkayı dolaştıktan sonra çerçeveyi kaldırır. Ek olarak FS alanındaki A ve C bitleriyle çerçevenin kopyalandığı veya iptal olduğu bilgisi saklanır. Daha sonra yeni bir token yaratıp bir başka DTE'nin kullanması için halkaya yollar.

ÇERÇEVENİN ALINMASI

Gelen işaretin tekrarlanmasına ek olarak MAC unitesi her çerçevenin başangıcını tanır ve eğer F bitlerine göre bir MAC çerçevesi olduğunu belirtiyorsa çerçeve kopyalanır, ve eğer çerçeve normal data taşıyan bir çerçeveyse ilgili DTE veya grup adresine kopyalanır.

ÖNCELİK İŞLEMİ

MAC tarafından token'a atanan öncelik aşağıdaki koşullara bağlıdır:

1) Halkanın o andaki önceliğinden daha yüksek seviyeli önceliğe sahip çerçeveler,
2) Bütün DTE'lerin çerçevelerinin halkaya erişmede aynı haklara sahip olması.

Her çerçevenin AC alanındaki P ve R bitleri ile öncelik işlemleri yapılır. Bunun için her MAC unitesi iki değişken kümesi kullanır: **Pm, Pr** ve **Rr**. Pm o anda DTE'den o anda iletilen çerçevelerin en yüksek öncelik sırasını gösterir. Pr ve Rr'ye **"öncelik sayaçları (priority registers)"** denir ve sırasıyla en son alınan token veya çerçevenin AC alanındaki öncelik ve rezervasyon değerlerini saklar. Değerlerin saklandığı iki yığına (stack) da Sr ve Sx denir. DTE tarafından iletilen bütün çerçevelerin AC alanında o andaki halka servis önceliği Pr ve rezervasyon değeri sıfırdır. Öncelik sırası daha büyük olan bütün çerçeveler iletildikten sonra MAC unitesi **P=P** ve **R=Rr Ve Pm'den büyük** olan yeni bir token yaratır. Eğer DTE'nin beklediği yeni bir çerçeve yok ve önceliği o andaki önceliğe eşit/büyükse öncelik **P=Rr ve Pm'den büyük** ve **R=0** olacaktır. DTE halkanın önceliğini aldığından ve daha önceki servis önceliğini (Pr) Sr yığınında ve yeni servis önceliğini Sx yığınında sakladığından **"Yığın terminali (stacking station)"** adını alır. Yeni token **P=Rr** ve **R=0** değerleriyle gönderilir.

HALKA YÖNETİMİ

Bundan önceki bölümlerde halkanın normal çalışması sırasında çerçeve ve tokenlerin iletilmesiyle ilgiliydi. Bundan başka DTE normal çalışan bir halkaya eklenmek istediğinde halkanın çalışmasını bozmamak için bir başlangıç prosedürüne ihtiyaç vardır. Ek olarak normal çalışmada her aktif DTE hata olduğunda hatayı düzeltme işlemleri yapar. Bütün bu işlemler "halka yönetimi" olarak bilinir.

Bir DTE halkanın parçası olacağı zaman başlangıç (initialization) prosedüründe önce DTE açılır, halkada aynı adrese sahip olan başka bir DTE olup olmadığı araştırılır ve yanındaki DTE'ye halkaya girildiği bildirilir. Başlangıç prosedürü **"aynı adres test (DAT-Duplicate Address Test)"** çerçevesinin gönderilmesiyle başlar. Eğer DAT çerçevesinin A biti "1" ise DTE kendi ağ seviyesine bypass konumuna geçmesini bildirir. DAT çerçevesinde bütün A bitleri "0" ise DTE başlangıç prosedürüne **SMP (Standby Monitor Present)** MAC çerçevesi göndererek devam eder. SMP çerçevesini A ve C bitleri "0" olarak alan bir DTE kendinden sonra gelen DTE'nin adresini alır ve bunu **"UNA-Upstream Neighbours Address"** olarak SA alanında saklar. Bu işlem hata saptama ve monitoring fonksiyonları için gereklidir.

STANDBY VE AKTİF MONITOR

Başlangıç prosedürü tamamlandıktan sonra DTE normal çerçeve ve tokenleri iletebilir. Buna ek olarak DTE "standby monitor" moduna geçerek halkanın çalışmasını izleyebilir. Monitor DTE halkada **"AMP-Active Monitor Present"** MAC çerçevesini periyodik olarak dolaştırarak monitoring işlemlerini yapar.

Bir DTE yeni aktif monitor olduğu zaman kendi varolma (latency) tampon belleğini halkaya aktarır ve kendi clock işaretini aktif hale (enable) getirir. Halkada aynı anda sadece bir tane aktif monitor olabilir. DTE daha sonra PRG (purge) MAC çerçevesini göndererek halkada başka token veya çerçeve olmadığına emin olur. DTE, PRG çerçevesindeki SA alanında kendi adresini bulursa bu işlemin başarılı olduğu anlar.

Aktif monitor olan DTE AMP (active monitor present) çerçevesini düzenli aralıklarla halkada dolaştırır ve her DTE bu çerçeveyi tekrarlar. Bu şekilde halkadaki her DTE tokenlerin halkada sürekli dolaşması (jabbering) gibi hataları bulabilir ve **"Beaconing"** denilen hata düzeltme prosedürü işleme konulur.

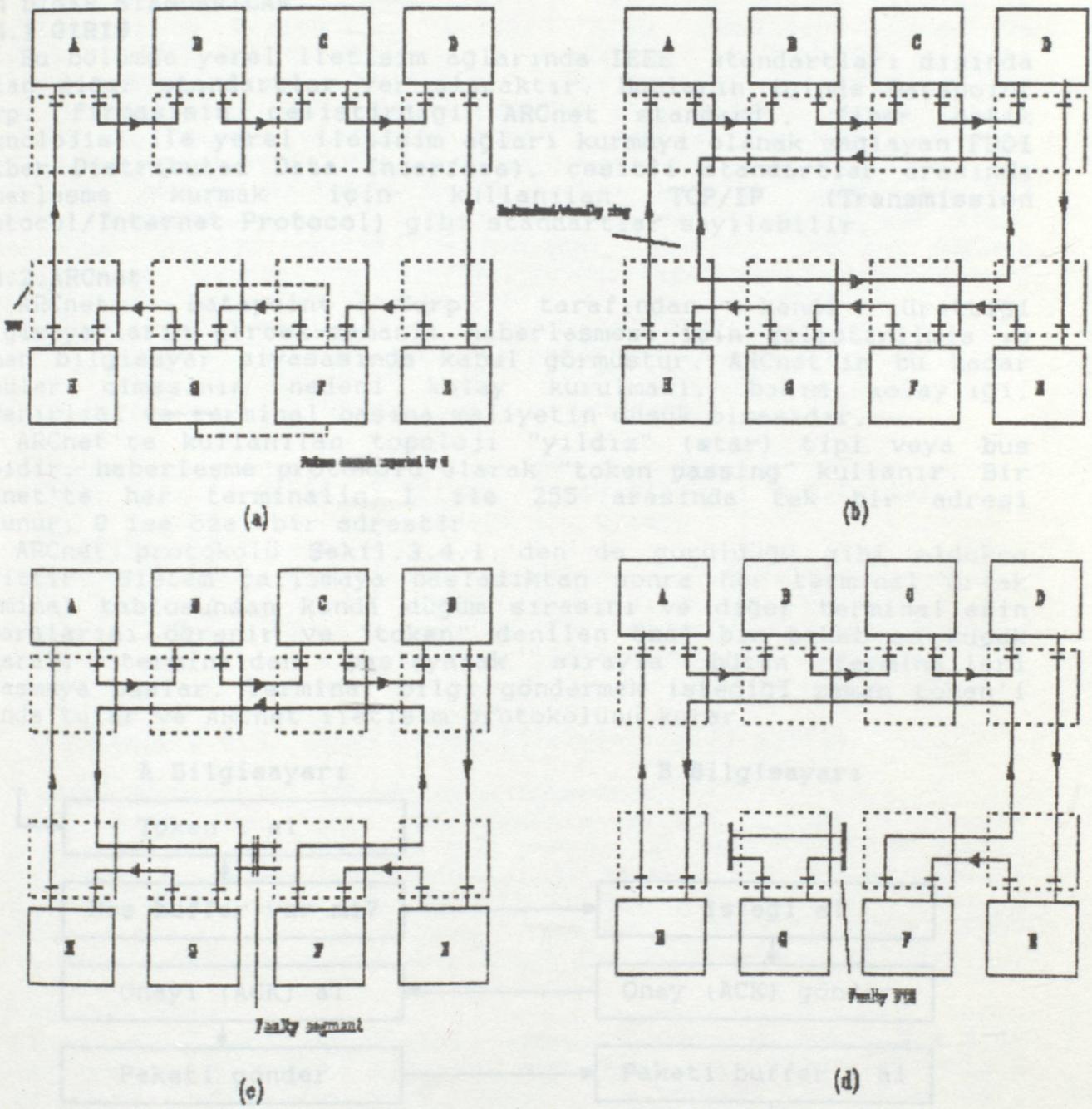
BEACONING (İşaretleme)

Eğer kablo kopması gibi ciddi bir hata varsa **token-passing** protokolü duracağından **"beaconing"** denilen bir prosedür ile her DTE bundan haberdar olur. Bozulabilecek ortam aşağıdakiler olabilir:

- Hatayı rapor eden "uyarıcı terminal" (beaconing station),
- Uyarıcı terminalin yukarısındaki DTE,
- Bu iki DTE arasındaki haberleşme ortamı.

Şekil.3.3.14'de F ve G DTE'leri arasındaki bozuk ortam nedeniyle haberleşme kesilmesi örnek olarak gösterilmiştir. Bu örnekte G uyarıcı terminal ve F bunun üst komşusudur. Hata rapor edildiği zaman halka standby moda geçirilir **(b)**. Daha sonra halkanın sağlam kısmı bu geçici halka ile birleştirilerek bozuk segment ayırılır **(c)**. **(d)**'de ise bozuk bir terminal nedeniyle yapılan düzeltme görülmektedir.

Bundan sonraki bölümlerde yukarıda bahsedilen IEEE 802.X standartları gerçek boyutlarıyla modellenerek bilgisayar simulasyonu yöntemiyle incelenecektir.



Şekil.3.3.14: Halkada hata saptama ve düzeltme.
(a) Hata saptama (b) standby mod'da çalışma.
(c) Segment ayırma (d) Terminali ayırma.

3.4 DİĞER STANDARTLAR

3.4.1 GİRİŞ

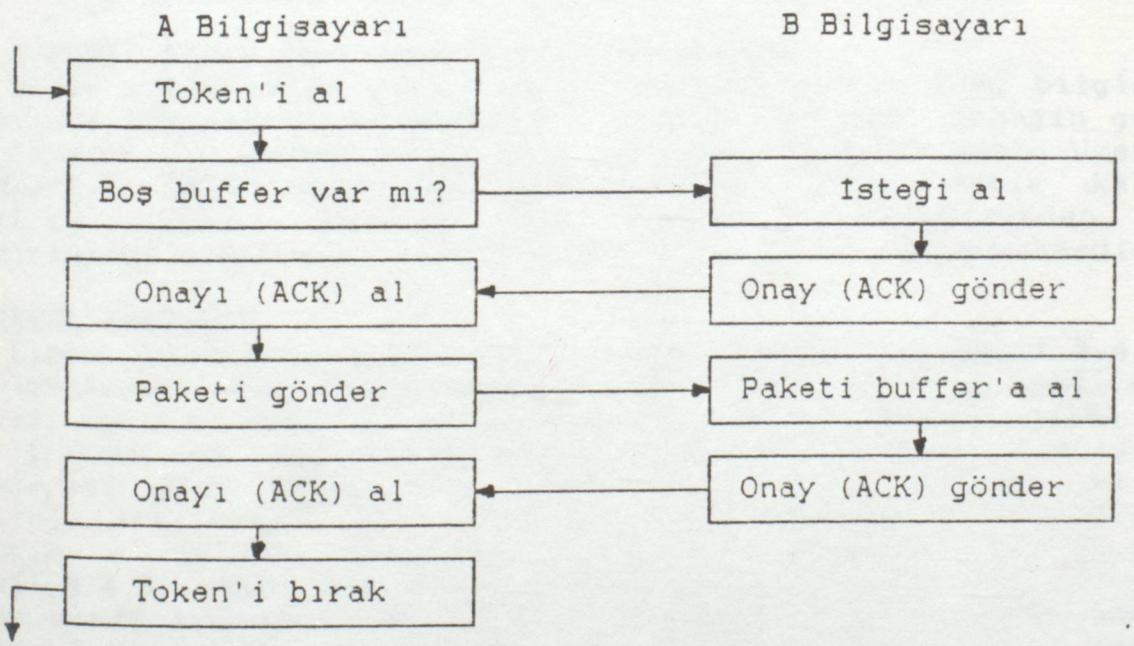
Bu bölümde yerel iletişim ağlarında IEEE standartları dışında kalan diğer standartlar yer alacaktır. Bunların içinde Datapoint Corp. firmasının geliştirdiği **ARCnet** standardı, fiber optik teknolojisi ile yerel iletişim ağları kurmaya olanak sağlayan **FDDI (Fiber Distributed Data Interface)**, çeşitli standartlar arasında haberleşme kurmak için kullanılan **TCP/IP (Transmission Protocol/Internet Protocol)** gibi standartlar sayılabilir.

3.4.2.ARCnet

ARCnet, Datapoint Corp. tarafından kendi ürettiği bilgisayarların gerçek-zamanlı haberleşmesi için geliştirilmiş ve hemen bilgisayar piyasasında kabul görmüştür. ARCnet'in bu kadar popüler olmasının nedeni kolay kurulması, bakım kolaylığı, güvenirliği ve terminal başına maliyetin düşük olmasıdır.

ARCnet'te kullanılan topoloji **"yıldız" (star)** tipi veya **bus** tipidir, haberleşme protokolü olarak **"token passing"** kullanır. Bir ARCnet'te her terminalin 1 ile 255 arasında tek bir adresi bulunur. 0 ise özel bir adrestir.

ARCnet protokolü Şekil.3.4.1.'den de görüldüğü gibi oldukça basittir. Sistem çalışmaya başladıktan sonra her terminal ortak terminal tablosundan kendi düğüm sırasını ve diğer terminallerin numaralarını öğrenir ve **"token"** denilen özel bir paket en küçük numaralı terminalden başlayarak sırayla bütün terminalleri dolaşmaya başlar. Terminal bilgi göndermek istediği zaman token'i elinde tutar ve ARCnet iletişim protokolünü kurar.



Şekil.3.4.1: ARCnet protokol mantığı.

Mesajı göndermeden önce verici terminal alıcı terminalin yeterli tampon belleğinin (buffer) olup olmadığını sorar ve onaylama (ACK-Acknowledgement) işaretini aldığında paketi gönderir. Eğer alıcı terminalin tampon belleği yetersiz ise bu sefer negatif onay (NACK-negative ACK) geri gönderilir ve verici terminal bu işareti aldığında token'i serbest bırakır, token'in bir dahaki gelişinde tekrar göndermeyi dener.

Birçok ARCnet kartında 2 KB tampon belleği vardır ve 512 byte'lık 4 paketi alabilir. Tampon bellekteki bu kısıtlama yoğun haberleşme trafiği varsa performansı etkiler. Ancak ARCnet kartlarındaki hybrit devrelerin güvenirliği sonucu haberleşme diğer LAN'lara göre daha kararlıdır.

ARCnet'te sisteme yeni bir terminal eklemek veya çıkarmak çok kolaydır. Yeni bir terminal eklendiğinde veya çıkarıldığında network data transferini keser, yeni terminal yapısına göre yeniden konfigure eder ve çalışmasına devam eder.

Protokol Tekniği	Data Hızı (Mbps)	Paket Uzunluğu (byte)	Erişim tekniği	İletişim Ortamı	Özellikler
ARCnet	2.5	512	Token passing	Koaksiyel, Twisted-pair fiber-optik	Ucuzluk, Kararlılık, Esneklik

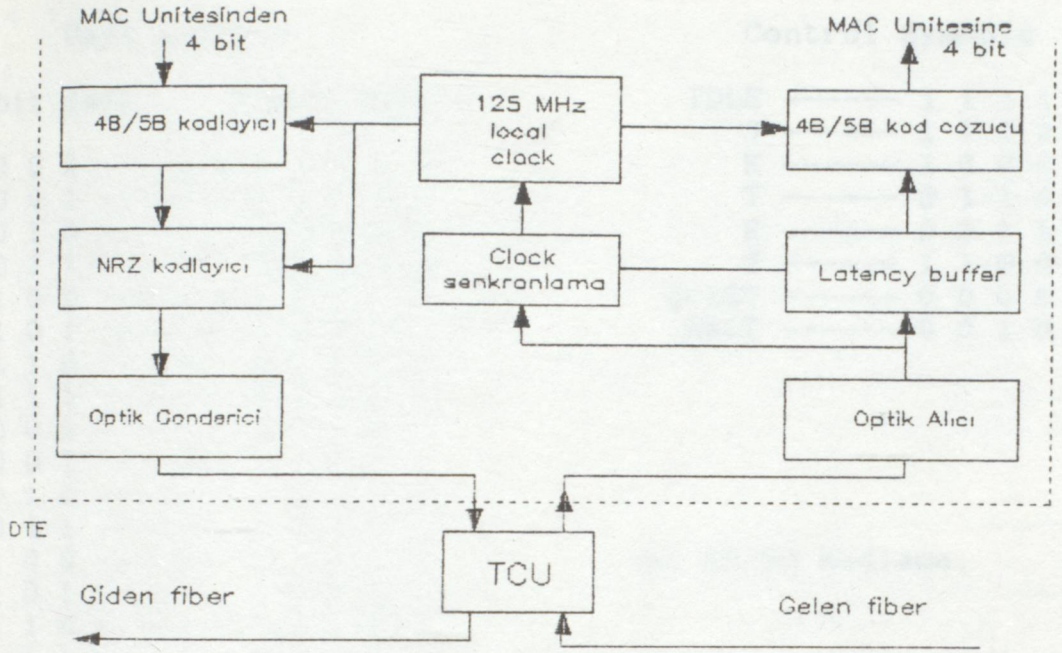
Şekil.3.4.2: ARCnet topolojisiyle ilgili özetler.

3.4.2 FDDI (Fiber Distributed Data Interface)

Büyük miktarlarda data transferinin gerekli olduğu bilgisayar ağlarında bilinen yöntemlerle hız sorunu aşılamaz. Örneğin grafik uygulamalarının gerektirdiği hıza ancak fiber optik kablo üzerinde 100Mps'lık FDDI ağlarında erişilebilir. Fiber optik kabloda elektrik işareti yerine ışık ışınları iletildiğinden FDDI arabirimleri elektronik olarak oldukça farklılık göstermektedir.

FİZİKSEL ARABİRİM

Fiber optik kablo üzerindeki fiziksel arabirim Şekil.3.4.3'de görülmektedir. FDDI halka tipi bir ağda her arabirimin kendi clock devresi vardır. Data bu clock işareti ile gönderilir, gelen data ise frekans ve faz kilitleme yöntemi ile alınır. İleride de görüleceği gibi bütün data gönderilmeden önce en az iki bit periyodu aralıklarla geçiş (transition) sağlamak için kodlanır. Datanın 4 bitlik grupları 5 bitlik gruplara dönüştürülür (Şekil.3.4.5). Yani her 4 bit kendisine karşı düşen kod sözcüğüne (code word) kodlanır. Bu işlemi yapan kodlayıcıya **4B/5B encoder** denir. Şekilden de görüleceği gibi en fazla iki 0 biti yanyana gelebilir. Daha sonra bu işaret **NRZI** kodlayıcıyla işaret geçişleri yaratılır. Böylece en az iki bitte bir işaret geçişi garantilenmiş olur.



Şekil.3.4.3: FDDI fiziksel arabirim şeması.

5 bitlik paketlerin 4 biti göstermesi nedeniyle geriye kalan 16 kombinasyon iletilen çerçevelerin veya tokenlerin başlangıç ve bitişleri gibi diğer link kontrol fonksiyonlarında kullanılır.

PA alanı 16 veya daha fazla **IDLE** sembollerini içerir ve beş bit sayesinde kablo işaretini maksimum frekansa çıkarır. Böylece alıcıda clock senkronizasyonu kurulmuş olur.

SD alanı J ve K gibi iki kontrol sembolünden oluşur, bunlar alıcının çerçeve içerikleri ve sınırlarını doğru tayin etmesini sağlar.

FC, **DA** ve **SA** alanları diğer protokollerdeki anlamlarını korur, INFORMATION alanı ise FDDI'da 4500 octet'e çıkabilir. **ED** (End Delimiter) alanı bir veya daha fazla kontrol sembolünden (T) oluşur. Son olarak **FS** (Frame Status) alanı R ve S kontrol sembollerinin kombinasyonunu içerir.

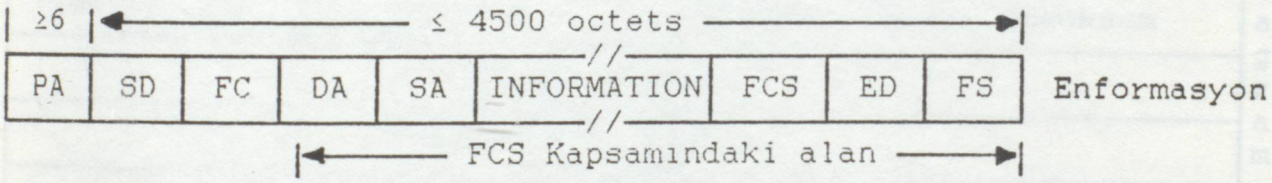
Saat (clock) işaretinin frekansı 125 MHz'dir, fakat 4B/5B operasyonu nedeniyle data hızı 100 Mbps olur. Bütün transmisyon 5 bitlik sembollerle yapıldığından beş bitlik her sembol kodlanmadan önce tampon bellekte saklanmalıdır. SD alanındaki J ve K sembolleri alıcıya 10-bit tampon bellek sağlar. Buna "**latency (elastic) buffer**" denir.

Protokol Tekniği	Data Hızı	Maksimum Uzaklık	Topoloji	Erişim tekniği	Max. Node	İletişim Ortamı
FDDI	100 Mbps	2 km	Dual Ring	Token passing	500	fiber-optik kablo

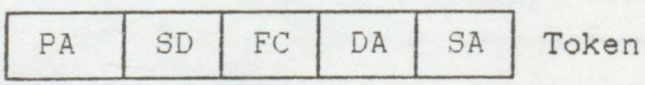
Şekil.3.4.4: FDDI standardıyla ilgili özet bilgiler.

Data Symbols		Control Symbols	
4-bit data	5-bit symbol		
0 0 0 0	1 1 1 1 0	IDLE	1 1 1 1 1
0 0 0 1	0 1 0 0 1	J	1 1 0 0 0
0 0 1 0	1 0 1 0 0	K	1 0 0 0 1
0 0 1 1	1 0 1 0 1	T	0 1 1 0 1
0 1 0 0	0 1 0 1 0	R	0 0 1 1 1
0 1 0 1	0 1 0 1 1	S	1 1 0 0 1
0 1 1 0	0 1 1 1 0	QUIET	0 0 0 0 0
0 1 1 1	0 1 1 1 1	HALT	0 0 1 0 0
1 0 0 0	1 1 1 1 0		
1 0 0 1	1 0 0 1 1		
1 0 1 0	1 0 1 1 0		
1 0 1 1	1 0 1 1 1		
1 1 0 0	1 1 0 1 0		
1 1 0 1	1 1 0 1 1		
1 1 1 0	1 1 1 0 0		
1 1 1 1	1 1 1 0 1		

(a) 4B/5B kodlama.



PA : Preamble (16>)
SD : Start Delimiter (2)
FC : Frame Control (2)
DA : Destination Address (4 or 12)
SA : Source Address (4 or 12)
FCS: Frame Check Sequence (8)
ED : End Delimiter (1 or 2)
FS : Frame Status (3)



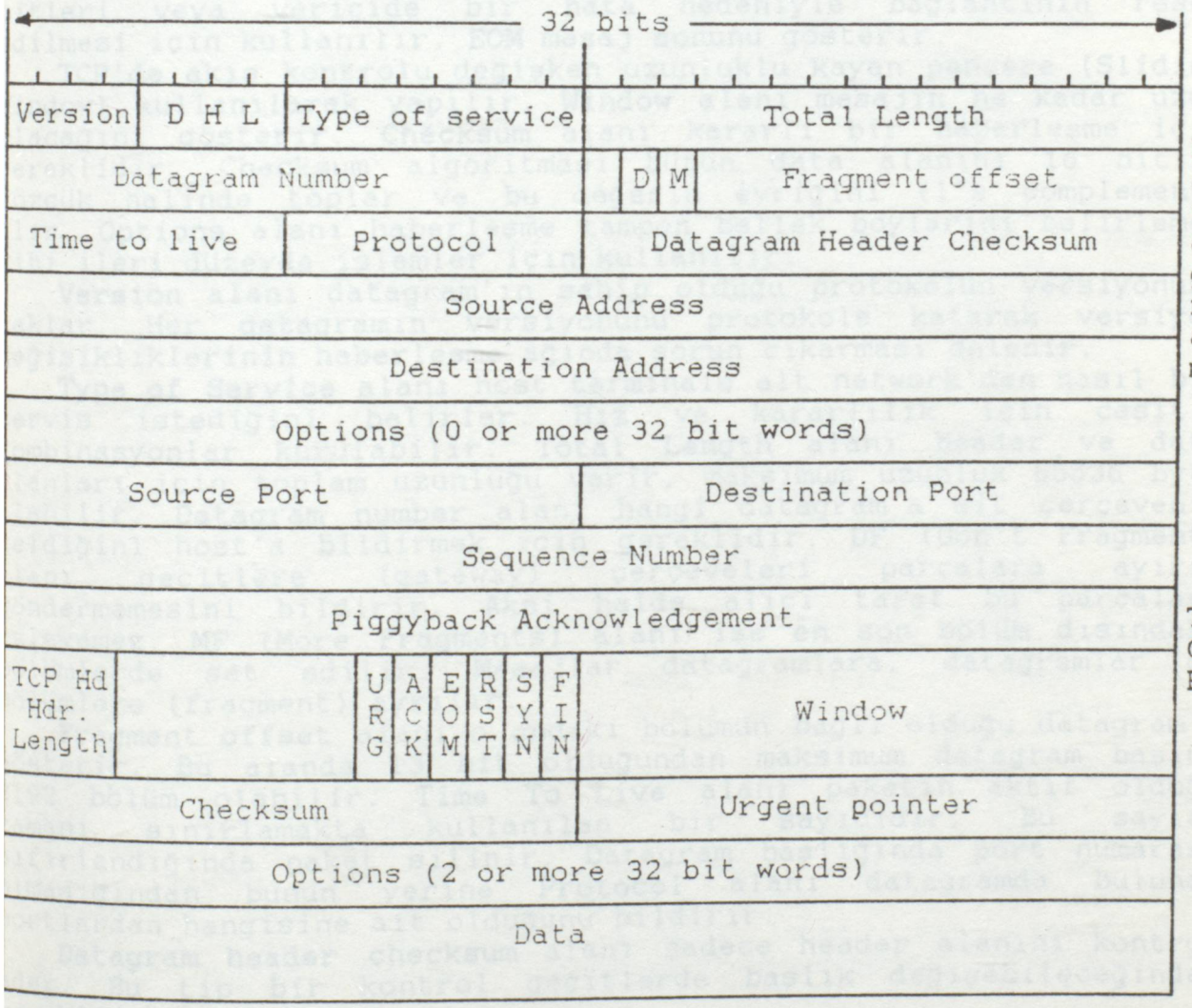
(b) Çerçeve formatı.

Şekil.3.4.5: FDDI kodlaması ve çerçeve formatları.

3.4.3 TCP/IP (Transmission Control Protocol/Internet Protocol)

TCP protokolu Daha çok haberleşmede işaret kayıpları fazla olan alt network'larda kullanılmak için tasarlanmıştır. Bir TCP ulaşım terminali kullanıcılardan uzun mesajları alır ve bunları 65 Kbyte'ı aşmayan parçalara ayırdıktan sonra her parçaya ayrı datagram olarak gönderir. Ağ seviyesi datagramların yerine ulaşmasını garanti etmediğinden bu kontrol TCP'ye bağlıdır ve TCP gerekirse paketi yeniden gönderir. Sırasına uygun gitmeyen datagramları TCP yeniden uygun sıraya koyarak düzeltir.

TCP tarafından gönderilen her datanın kendi sıra numarası vardır. Dizi numarası 32 bittir ve eski kopyaların kaybolması yüzünden sıra numaraları karışabilir. TCP bu nedenle geciken kopyaların neden olduğu problemi üç yollu el sıkışma (handshake) ile çözer.



Şekil.3.4.6: TCP ve internet datagram başlıkları.

Şekil.3.4.6'de TCP tarafından kullanılan **header** ve **datagram** protokolu görülmektedir. Minimum TCP+Datagram protokolünün header alanı 40 byte'tır. **Source Port** ve **Destination Port** alanları kullanıcı işlemleri arasındaki haberleşmeyi tanımlar. **Sequence Number** ve **Piggyback Acknowledgement** alanları bilinen işlevleri yapar ve uzunlukları 32 bittir. **TCP Header Length** TCP header'da kaç tane 32 bitlik sözcük olduğunu belirtir. Bu seçenek **Option** alanı değişken uzunluk alabildiğinden gereklidir. **URG** bayrağı (flag) 1 ise **Urgent** (ivedi) gösterici (pointer) kullanıldığını gösterir. Urgent pointer ise o andaki dizi numarasından uzaklığı göstermek için kullanılır. **SYN** biti haberleşme kurmak için kullanılır. **SYN1** de denilen bağlantı isteğinde SYN=1 ve ACK=0 olur. Bağlantı yanıtında ise **(SYN2)** SYN=1 ve ACK=1'dir. ACK biti aslında SYN1 ile SYN2'yi ayırmak için kullanılır. **FIN** biti bağlantının bitirilmesi için kullanılır. **RST** biti gecikmiş SYN bitleri veya vericide bir hata nedeniyle bağlantının reset edilmesi için kullanılır. **EOM** mesaj sonunu gösterir.

TCP'de akış kontrolu değişken uzunluklu kayan pencere (Sliding Window) kullanılarak yapılır. **Window** alanı mesajın ne kadar uzun olacağını gösterir. **Checksum** alanı kararlı bir haberleşme için gereklidir. Checksum algoritması bütün data alanını 16 bitlik sözcük halinde toplar ve bu değerin evriğini (1's complement) alır. **Options** alanı haberleşme tampon bellek boylarını belirlemek gibi ileri düzeyde işlemler için kullanılır.

Version alanı datagram'ın sahip olduğu protokolün versiyonunu saklar. Her datagramın versiyonunu protokole katarak versiyon değişikliklerinin haberleşme ağında sorun çıkarması önlenir.

Type of Service alanı host terminale alt network'den nasıl bir servis istediğini belirler. Hız ve kararlılık için çeşitli kombinasyonlar kurulabilir. **Total Length** alanı header ve data alanları için toplam uzunluğu verir, maksimum uzunluk 65536 byte olabilir. **Datagram number** alanı hangi datagram'a ait çerçevenin geldiğini host'a bildirmek için gereklidir. **DF** (Don't Fragment) alanı geçitlere (gateway) çerçeveleri parçalara ayırıp göndermemesini bildirir. Aksi halde alıcı taraf bu parçaları izleyemez. **MF** (More Fragments) alanı ise en son bölüm dışındaki bölümlerde set edilir. Mesajlar datagramlara, datagramlar da bölümlere (fragment) ayrılır.

Fragment offset alanı o andaki bölümün bağlı olduğu datagram'ı gösterir. Bu alanda 13 bit olduğundan maksimum datagram başına 8192 bölüm olabilir. **Time To Live** alanı paketin aktif olduğu zamanı sınırlamakta kullanılan bir sayıcıdır. Bu sayıcı sıfırlandığında paket silinir. Datagram başlığında port numarası olmadığından bunun yerine **Protocol** alanı datagramda bulunan portlardan hangisine ait olduğunu bildirir.

Datagram header checksum alanı sadece header alanını kontrol eder. Bu tip bir kontrol geçitlerde başlık değişebileceğinden gereklidir.

Source ve **Destination address** network (8 bit) ve host (24 bit) numarasını verir. **Options** alanı güvenlik, kaynak yönlendirmesi, hata raporlama ve diğer bilgiler için kullanılır.

3.5 CCITT X.25 PROTOKOLU

3.5.1 GİRİŞ

Public Data Network'leri (PDN) çalışma prensiplerine göre genelde paket-anahtarlamalı (PSDN) ve devre-anahtarlamalı (CSDN) data network'leri olarak ikiye ayrılır. Genelde daha yaygın olması açısından standartlar PSDN'ler için belirlenmiştir ve bu standartlar OSI referans modelinin en alt üç seviyesine karşı düşer. Paket anahtarlamalı network'lerde haberleşme DCE'lerle (Data Circuit Terminating Equipment), DTE'ler (Data Terminating Equipment) arasında yapılır, burada haberleşme yapılırken network boyunca DCE ile DTE arasında fiziksel bağlantı kurulmaz, PSE (Packet Switching Exchange) aracılığıyla iletişim kurulur. Paket haberleşmesinde her paketin içinde varış adresi saklıdır. Paket DTE'ye geldiğinde DTE paketin içindeki adresin kendi adresi olup olmadığına bakar. Her PSE içinde bir "Routing Directory" bulunduğundan PSE bu paketi belirli transmisyon yollarından uygun hızda yollar. Bu çeşit çalışmaya "Sakla-Gönder" (Store-and-Forward Mode) denir.

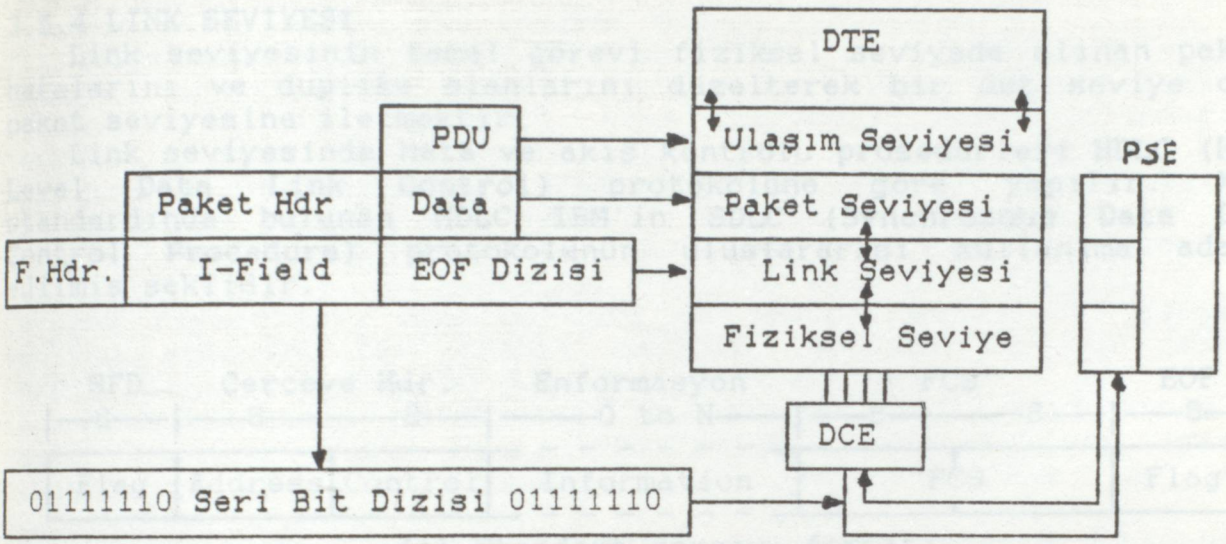
PSDN ile iki çeşit servis desteklenir: Datagram ve Virtual Call (circuit). Datagram servisinde her paket diğerlerinden bağımsız olarak gönderilir. Eğer mesajda birden fazla paket varsa, virtuel çağrı servisi seçilir. Bu tip çalışmada önce iki taraftaki DTE'ler arasında bağlantı özel paketlerle kurulur. Daha sonra iki DTE arasında iki yönlü haberleşme yapılır. Bazı durumlarda kullanıcı iki DTE arasında devre-anahtarlamalı network'lerdeki gibi sürekli bağlantı istediğinde "Permanent (kalıcı) Virtuel Circuit" denilen özellik kullanılır.

3.5.2 PAKET-ANAHTARLAMALI DATA AĞLARI

CCITT X.25 tavsiyeleri, PSDN'ler (Public Switching Data Network) üzerinde bilgisayar veya terminal (DTE) ile network haberleşme cihazı (DCE) arasındaki haberleşme için standartlar koyar. X.25 protokolu OSI (Open Systems Interconnection) referans modelinde alttan ilk üç seviye protokollerine karşı düşer.

CCITT (Consultative Committe on International Telephony and Telegraphy), PDN'lere 1974 yılında yayınladığı gri kitapta X.25'in temel yapısını belirlemiştir. 1977'de link kontrol protokollerinde yapılan değişikliklerle ISO'nun HDLC protokolüne benzer protokoller oluşturulmuştur. Şekil.3.5.1'de görüldüğü gibi X.25 protokolü birkaç seviyeden oluşur.

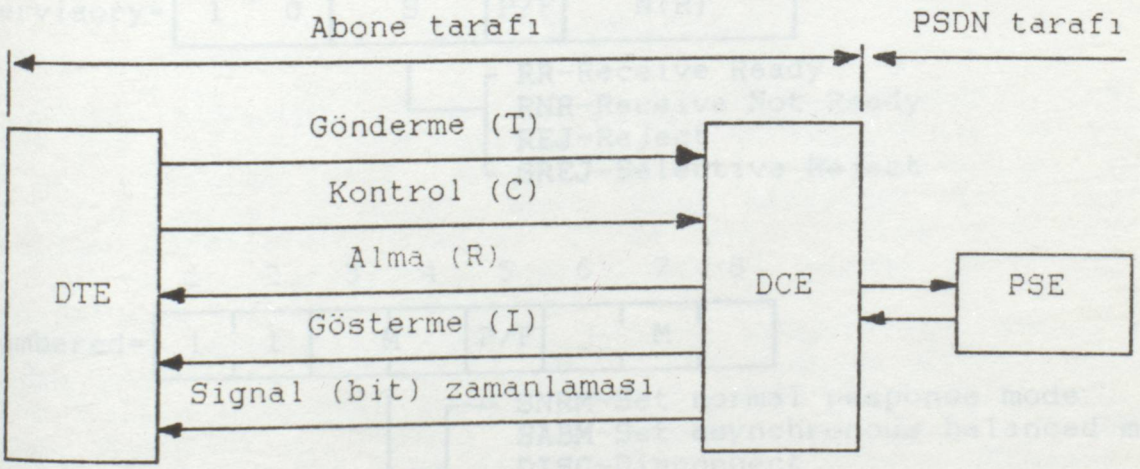
Fiziksel seviyede DCE ile DTE arasında bir elektriksel bağlantı vardır. Bu seviyede full-duplex senkron haberleşme için X.21 standardı kullanılır. X.25'in ikinci seviyesi DCE ile DTE arasında bağlantı kuran link erişim prosedürünü (LAP) içerir. X.25 standardının üçüncü seviyesi public data network'ü üzerinden virtüel çağrıların yapılmasını kontrol eder. Son olarak X.25 standardı PAD'ın (Packet Assembler/Dissassembler) fonksiyonunu açıklar.



Şekil.3.5.1: X.25 mesaj birimleri ve seviye arası haberleşmeler.

3.5.3 FİZİKSEL SEVİYE

DTE ile PTT'nin kullandığı DCE arasındaki fiziksel arabirim CCITT'nin **X.21** tavsiyelerinde tanımlanmıştır. DCE'nin fiziksel seviyede fonksiyonu DTE ile lokal PSE arasında full-duplex, seri ve senkron haberleşme sağlayan senkron bir modem'den pek farklı değildir. Hızı 600 bps ile 48 Kbps arasında değişebilir. Bundan başka kullanıcı cihazlarının varolan analog network'lerle RS-232C aracılığıyla çalışması için **X.21 bis** adlı ikinci bir standart da vardır. **Şekil.3.5.2**'de X.21 ile bağlanan iki DCE görülmektedir.

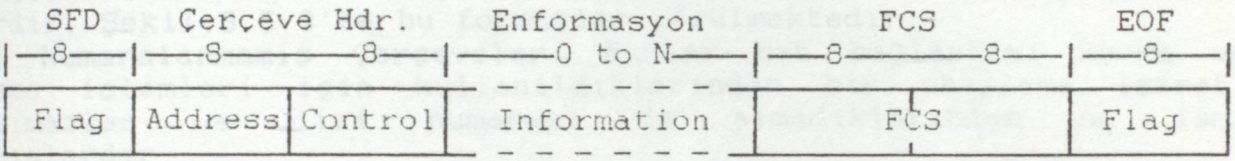


Şekil.3.5.2: X.21 fiziksel Seviye Devresi.

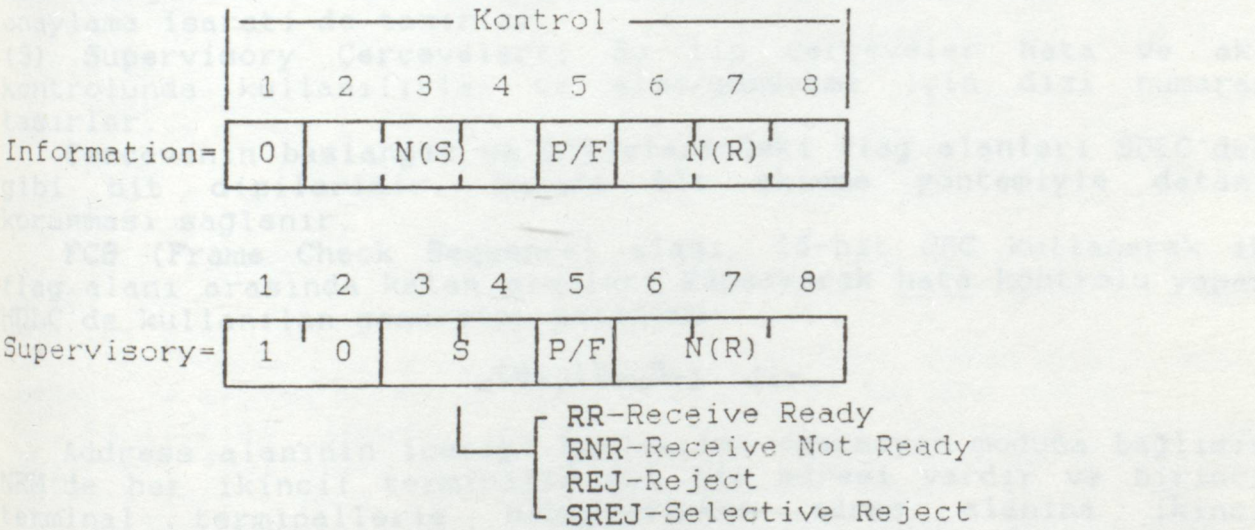
3.5.4 LINK SEVIYESİ

Link seviyesinin temel görevi fiziksel seviyede alınan paketi hatalarını ve duplike olanlarını düzelterek bir üst seviye olan paket seviyesine iletmektir.

Link seviyesinde hata ve akış kontrolu prosedürleri **HDLC (High Level Data Link Control)** protokolüne göre yapılır. X.25 standardında bulunan HDLC IBM'in **SDLC (Synchronous Data Link Control Procedure)** protokolünün uluslararası kullanıma adapte edilmiş şeklidir.



(a) Standart çerçeve formatı.



(b) Kontrol alanları.

Şekil.3.5.3: HDLC çerçeve formatları.

HDLC temel olarak iki çalışma moduna sahiptir:
(1) **Dengelenmemiş Normal Yanıt Modu (NRM)**: Bu tip çalışma terminal temelli network'lerde ikincil veya uydu (slave) diye tanımlanan terminallerin ana (master) veya birincil diye adlandırılan terminalle olan haberleşmesi yapılırken seçilir. Network hattı eğer ana terminal izin veriyorsa uçtan-uca veya çok-uçlu olabilir.
(2) **Asenkron Dengeli Mod (ABM)**: Bu çalışma bilgisayarlar veya istatistiksel çoğullayıcılar arasında direkt haberleşme yapılırken seçilir. Bu modda her terminal eşit önceliktedir.
Çerçeve Formatları: HDLC'de data ve kontrol mesajları "**çerçeve**" denilen standart bir blokla taşınır. HDLC'de üç tip çerçeve vardır. **Şekil.3.5.3**'te bu formatlar görülmektedir.
(1) **Numaralanmamış Çerçeveler**: Bunlar hat bağlantısı kurma ve kesme işlemleri için kullanıldıklarından bir onaylama işareti taşımazlar ve dizi numarası da almadıklarından bu ismi almışlardır.
(2) **Bilgi Çerçeveleri**: Bunlar gerçek datanın taşındığı çerçevelerdir ve normalde "**I-frame**" olarak bilinirler. Eğer hat ABM (dengelenmiş mod) ile kurulmuşsa bu çerçeveler ters yönde bir onaylama işareti de taşırlar.
(3) **Supervisory Çerçeveleri**: Bu tip çerçeveler hata ve akış kontrolunda kullanılırlar ve alma/gönderme için dizi numarası taşırlar.

Çerçevenin başlangıç ve bitişlerindeki flag alanları SDLC'deki gibi bit dizileridir, burada bit ekleme yöntemiyle datanın korunması sağlanır.

FCS (Frame Check Sequence) alanı, 16-bit CRC kullanarak iki flag alanı arasında kalan alanları kapsayarak hata kontrolu yapar. HDLC'de kullanılan generator polinomu:

$$x^{16}+x^{12}+x^{5}+1 \text{ 'dir.}$$

Address alanının içeriği kullanılan operasyon moduna bağlıdır. NRM'de her ikincil terminalin tek bir adresi vardır ve birincil terminal terminallerle haberleşirken adres alanına ikincil terminalin adresini koyar. Buna **grup adresleri** denir ve grup adresiyle adreslenen paketler gruba ait terminallere iletilir. Bütün adresleri kapsayan adrese **broadcast adresi** denir ve paket bütün ikincil terminallere gider.

İkincil terminaller yanıt mesajı göndereceği zaman adres alanına yine kendi adresini koyar. ABM modunda direkt uçtan-uca hatlar kullanıldığından adres alanı bu şekilde kullanılmaz, onun yerine komutların yönünü ve ilgili yanıtları göstermekte kullanılır. **Şekil.3.5.3**'te çeşitli kontrol alanları görülmektedir. Supervisory çerçevesindeki S-alanı ve Numaralanmamış çerçevedeki M-alanı özel çerçeve tiplerini belirler. Gönderme ve alma dizi numaraları (**N(S), N(R)**) akış ve hata kontrolu için kullanılır. **P/F-biti (Poll-Final)** Birincil terminal içinde Poll biti olan bir çerçeve gönderdiğinde bunun komut çerçevesi olduğunu bildirir ve bir yanıt çerçevesi bekler. Alıcı da içinde Final biti bulunan bir paket gönderir.

3.5.5 PAKET (NETWORK) SEVİYESİ

Paket seviyesinde, her çerçevenin içerdiği datanın üst seviyelere taşınması, yönlendirme ve virtüel devre düzenlenmesi işlemleri yapılır.

Bir DTE diğer bir DTE ile haberleşeceği zaman önce virtüel devre kurar. Bunun için DTE **CALL REQUEST** paketini düzenleyip diğer DTE'ye yollar. Eğer alıcı DTE çağrıyı kabul ederse **CALL ACCEPTED** paketi gönderir. Verici DTE bu paketi aldıktan sonra çağrı kurulmuş olur ve DTE'ler bu full-duplex virtüel devreyi kullanabilirler. Bir tarafın buffer'ı dolduğunda **CLEAR REQUEST** paketini gönderir, daha sonra onaylama olarak **CLEAR CONFIRMATION** paketini gönderir.

Verici DTE görüşecek boş bir virtüel devre seçebilir. Eğer bu virtüel devre alıcı DTE tarafından kullanılıyorsa, alıcı DCE paketi almadan önce bunun yerine boş biriyle değiştirmelidir. Böylece giden çağrılar DTE, gelenler ise DCE'ler tarafından tanımlanır. Bazen aynı virtüel devre her ikisi tarafından seçilir ki bu duruma "**Çağrı çarpışması-Call Collision**" denir. X.25 bu durumda gelen çağrının iptalini ve giden çağrının gözönüne alınmasını belirtir. Bu virtüel çağrılara ek olarak X.25 kalıcı virtüel devre de ayırabilir. Bunlar kiralanmış hatlar gibi hizmet verir ve bir çağrı kurma işlemine gerek kalmaz.

CALL REQUEST paketinin formatı **Şekil.3.5.4(a)**'da görülmektedir. Bu pakette diğer X.25 paketleri gibi 3-byte header ile başlar.

Grup ve **Channel** alanları 12-bit virtüel devre numarasını saklar. 0 numaralı virtüel devre rezervedir, prensipte 4095 virtuel devre aynı anda açılabilir.

CALL REQUEST paketindeki ve diğer kontrol paketlerindeki **Type** alanı paket tipini tanımlar. **Control** biti data paketlerinde 0, kontrol paketlerinde 1 değerini alır.

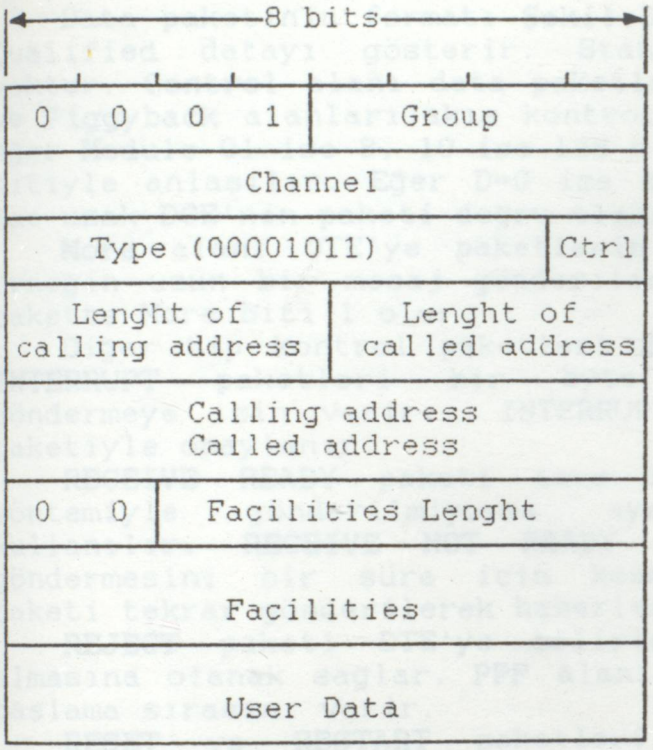
İlk 3-byte'lık header'den sonra kontrol paketindeki alanlar aranan ve arayan adresleri desimal digitler halinde saklar.

Facilities Lenght alanı, Facilities alanının uzunluğunu belirler. **Facilities** alanı ise virtüel devre için gerekli olan ve network'lere göre değişen özellikler için kullanılır. Bu özellikler arasında haberleşme önceliği, paket uzunluğunun 128 byte'dan büyük olabilmesi, haberleşme hızı ve diğer özellikler sayılabilir. ,

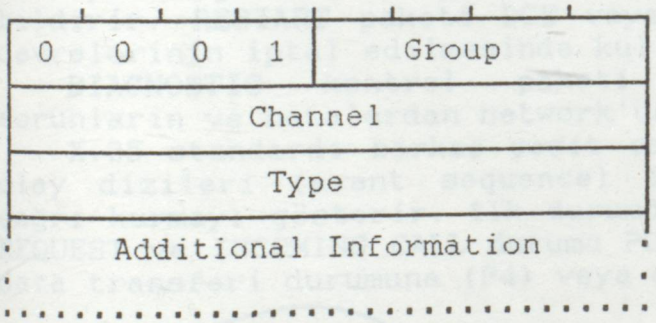
User data alanı, CALL REQUEST paketiyle beraber 16-byte'lık datanın gönderilmesine izin verir.

Diğer kontrol paketlerinin formatı **Şekil.3.5.4(b)**'de görülmektedir. Bazı durumlarda bu paketler sadece header'den oluşur veya birkaç byte eklenir. Örneğin CLEAR REQUEST paketinin 4.byte'ı virtüel devrenin neden kesildiğini bildirir.

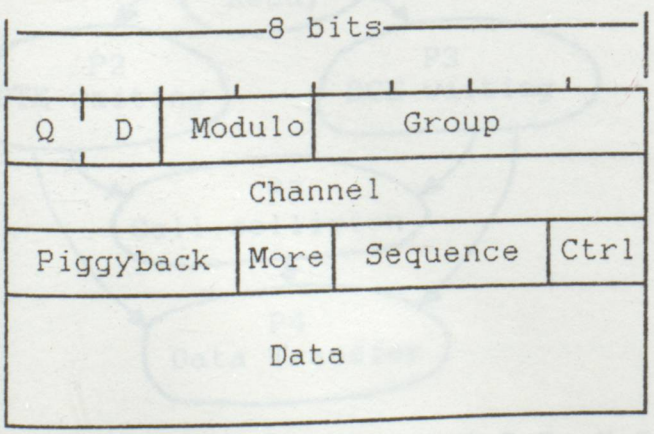
X.25'de CLEAR REQUEST ile CLEAR CONFIRMATION arasında farklılık olduğundan dolayı "**Kesme çarpışması-Clear Collision**" olasılığı vardır. Ancak bu durum fazla önemli değildir, virtüel devre fazla gecikmeden kesilir.



(a) Çagrı İsteği formatı.



(b) Control paket formatı.



Type	3.byte
Data	PPPMSSS0
Call Request	00001011
Call Accepted	00001111
Clear Request	00010011
Clear Confirmation	00010111
Interrupt	00100011
Interrupt Confir.	00100111
Receive Ready	PPP00001
Receive Not Ready	PPP00101
Reject	PPP01001
Reset Request	11111011
Reset Confirmation	00011011
Restart Request	11111011
Restart Confir.	11111111
Diagnostics	11110001

(d) Type alanı P:Piggyback
S:Sequence
M:More

(e) Data paketi formatı

Şekil.3.5.4: X.25 paket formatları.

Data paketinin formatı **Şekil.3.5.4(c)**'de görülmektedir. **Q** biti **Qualified** datayı gösterir. Standartta bu konuda bir açıklık yoktur. **Control** alanı data paketlerinde herzaman 0 olur. **Sequence** ve **Piggyback** alanları akış kontrolu için kullanılır. Dizi numarası eğer **Modulo** 01 ise 8, 10 ise 128 olur. Piggyback alanının anlamı **D** bitiyle anlaşılır. Eğer D=0 ise paketi lokal DCE'nin aldığı, D=1 ise uzak DCE'nin paketi doğru olarak aldığı anlaşılır.

More alanı DTE'ye paketlerin oluşturduğu grupları belirler. örneğin uzun bir mesaj gönderilirken en son paket dışındaki her pakette More biti 1 olur.

Diğer tip kontrol paketleri **Şekil.3.5.4(d)**'de listelenmiştir. **INTERRUPT** paketleri bir byte'lık sinyali sıranın dışında göndermeye izin verir. INTERRUPT paketi **INTERRUPT CONFIRMATION** paketiyle onaylanır.

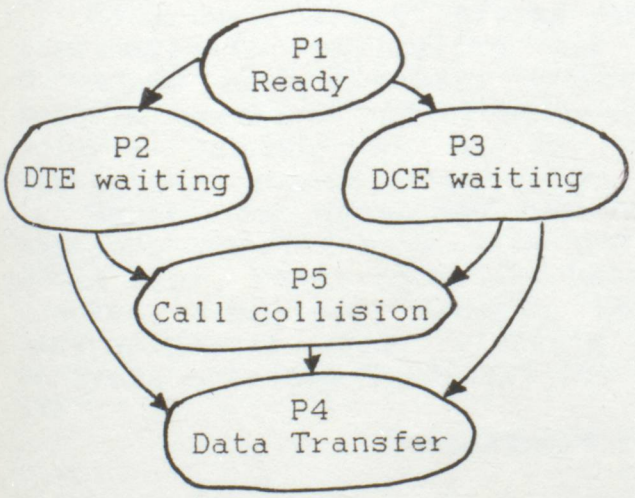
RECEIVE READY paketi ters bir onaylama işareti piggyback yöntemiyle gönderilmiyorsa ayrı onaylama göndermek için kullanılır. **RECEIVE NOT READY** paketi de diğer tarafa paket göndermesini bir süre için kesmesini bildirir. RECEIVE READY paketi tekrar gönderilerek haberleşmeye devam edilebilir.

REJECT paketi DTE'ye belirlenmiş bir seri paketi yeniden almasına olanak sağlar. **PPP** alanı yeniden gönderilecek paketlerin başlama sırasını verir.

RESET ve **RESTART** paketleri çeşitli bozuklukları ortadan kaldırmak için kullanılır. **RESET REQUEST** belirli bir virtüel devreye uygulanarak pencere parametresinin 0'a ayarlanmasını bildirir. **RESTART** paketi DCE veya DTE bozulduğunda bütün virtüel devrelerinin iptal edilmesinde kullanılır.

DIAGNOSTIC kontrol paketi haberleşmede oluşan çeşitli sorunların ve hatalardan network'ün bilgi sahibi olmasını sağlar.

X.25 standardı birkaç çeşit çağrı kurma ve kesme gibi çeşitli olay dizileri (event sequence) içerir. **Şekil.3.5.5**'deki diagram çağrı kurmayı gösterir. İlk durumda arabirim P1 durumundadır. CALL REQUEST ve INCOMING CALL durumu P2 ve P3'e çevirir. Bu durumlardan data transferi durumuna (P4) veya direkt olarak P5'e geçilir.



Geçiş	Yapan	Gönderilen Paket
1	DTE	Call Request
2	DCE	Call Connected
3	DCE	Incoming Call
4	DTE	Call Accepted
5	DCE	Incoming Call
6	DTE	Call Request
7	DCE	Call Connected

Şekil.3.5.5: X.25'te Çağrı kurma.

3.5.6 X.25 ve LAN'lar

X.25 standartları yakın zamanlara kadar yerel iletişim ağlarından farklı alanlarda kullanılıyordu. Fakat yerel iletişim ağları yaygınlaştıkça bu ağların birbiriyle yüksek hızlı haberleşmesi gündeme gelmiş ve bu şekilde geniş iletişim ağları (WAN) yerel ağların X.25 şebekesi üzerinden bağlanmasıyla oluşturulmuştur. LAN konusu üzerinde çalışan firmalar bu ağların X.25 bağlantısı için standart arabirimler geliştirmişlerdir. LAN üzerinde çalışan bir geçit aracılığıyla X.25 bağlantısı özel donanımla olarak yapılır ve bu geçitte çalışan bir yazılım X.25 protokollerini yerel ağda kullanılan protokole çevirir. Bu şekilde X.25 ile LAN kombinasyonu sayesinde geniş ağlar kurulabilmektedir.

4.1 YEREL İLETİŞİM AĞI PROTOKOLLERİNDE BAŞARIMIN SİMULASYON/ANALİTİK YÖNTEMLE İNCELENMESİ

4.1.1 GİRİŞ

Yerel iletişim ağları (LAN), yüksek data hızları ve düşük hata olasılığı özellikleriyle büyük ölçekli ağlardan (WAN) ayrılır. Genel olarak iletişim ağlarında başarım kavramını anlamak için aşağıdaki belirtilen parametreleri incelemek gereklidir:

1- Data Hızı: Genelde data hızı ve performans direkt olarak ilişkilidir. Data hızındaki artış başarımı aynı ölçüde arttırır. Ancak yüksek data hızında hata oranının da artacağı unutulmamalıdır.

2-Trafik yükü: İletişim ağındaki ortak kaynakları kullanan terminallerin sayısı topoloji ile bağlantılı olarak iletişim ağının başarımını etkiler. Terminal sayısı arttıkça özellikle CSMA/CD gibi çarpışma olasılığına dayalı ve Token Ring gibi paketin bütün ağı dolaşması gereken protokollerde başarımda düşüşler olur.

3-Erişim Yöntemi: Bu parametre daha çok diğer parametrelere bağımlıdır.

4-Topoloji: Bazı topolojilerin diğerlerine göre avantajları vardır. özellikle uzun mesafeli ağlarda veya yüksek bit hızında daha fazla transmisyon hatası oluşabileceğinden bu durumda kısa kablo mesafesine sahip ağların ve uygun bit hızının daha fazla başarım gegtirebileceği olacağı söylenebilir.

5-Hata oranı: İletişim ağında başarımın hesabının temel değişkeni hata oranıdır. Hataları saptama ve temizleme algoritmalarının çalışması için geçen zaman başarımı etkiler.

6-Protokoller: Mesajları almak ve göndermek için kullanılan protokoller uçtan-uca olan haberleşmedeki hızı etkiler.

Gönderilen paketlerin net data hızları ortak fiziksel kanalı paylaşma nedeniyle ve erişim metoduna göre performansta etkili olur. Data hızının en fazla etkili olacağı konu ağ üzerinde aynı kanalı paylaşan terminalleri ne kadar arttırılabileceğinin belirlenmesidir.

Başarımı direkt olarak etkileyen en önemli etken bütün terminallerin paylaşılan bir kaynağı aynı anda kullanmaları durumunda ortaya çıkar. Network'te bir kullanıcı açısından net başarım hesabı o andaki kullanıcı sayısının data hızına bölünmesi yoluyla bulunur ki bu da erişim metodu, protokol ve diğer faktörlerin gözardı etmek olur.

Haberleşme esnasında bir hata meydana geldiğinde bu hatanın saptanıp düzeltilmesi için gereken zaman performansta düşmelere neden olur. Yerel ağlardaki hata oranı bit hızına göre düşünülürse oldukça düşüktür. örneğin Ethernet'te bir hata oranı 10E-9 seviyesindedir. Eğer iletişim ağı %30 kapasite ile çalışıyorsa saniyede ortalama 3.000.000 bit iletilir. Ortalama hata ise

Hata/Saniye = bit/saniye x hata/bit
= 3E6 x 10E-9 hata/saniye
= 3E-3 hata/saniye

İki hata arasındaki zaman = 1/(Hata/saniye)
= 1/3E-3
= 333 saniye

Böylece 333 saniyede bir bitlik hata olduğu, yani kabaca 5½ dakikada bir hata olacağı söylenebilir.

Bütün yerel iletişim ağlarında hata oranları aşağı yukarı aynıdır. Aslında kurulacak ağın coğrafi durumuna göre yapılacak iyi bir tasarım ile çok daha düşük hata oranlarına erişilebilir. Hataların nedenleri çok çeşitlidir. Bunlardan bazıları:

1- Alıcı terminalde haberleşme için ayrılmış olan bellek sınırlı olduğundan bu bellek dolduğunda gelen mesajların alınması imkansız olacaktır. Bu durumda IEEE 802 protokolleri değişik önlemler alır.

2- İletişim ağındaki bir veya bir başka elemanın arızalanması.

3- Ağ'daki geçici durumların mesajların kaybolmasına neden olması. Token Ring ve Token Bus protokollerinde sıradaki terminallerin kaybolması nedeniyle yeniden düzenleme için bir miktar zaman gerekir.

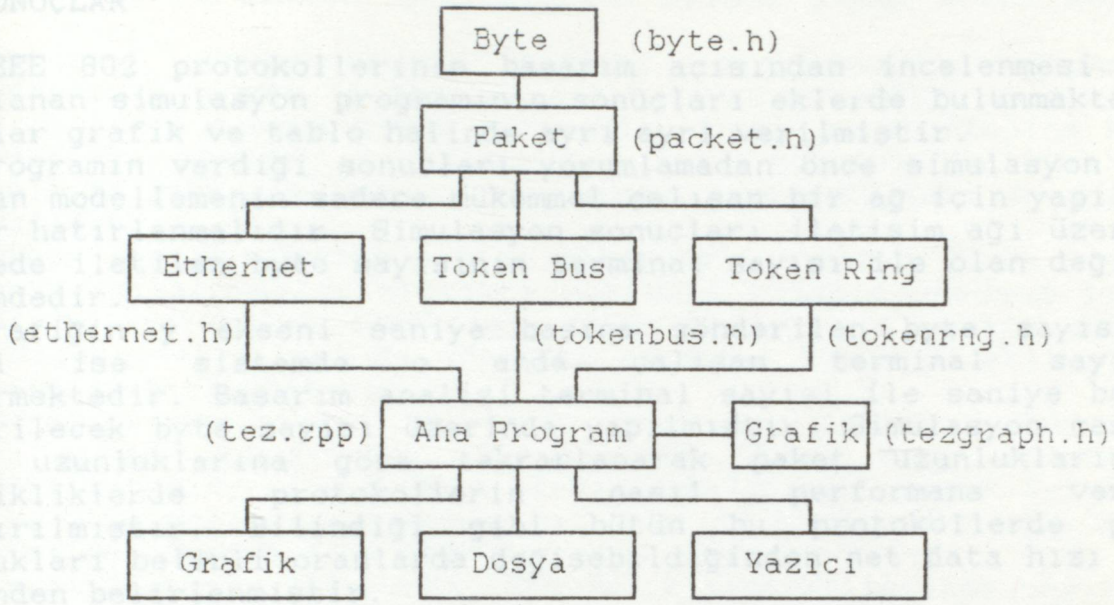
4- Elektrikli cihazlardan yayılan gürültü nedeniyle oluşan haberleşme hataları.

Yerel iletişim ağlarında istenen sadece başarımın yüksek değil, ağın çalışmasında süreklilik ve güvenliğin maksimum olması ve maliyettir. Bazı durumlarda güvenlik ön plana çıkar ve hız parametresi ikinci sıraya düşebilir. Bu durumda topoloji ve protokol seçiminin önemi çok daha fazla olacaktır. Bu çalışmada yerel iletişim ağlarında sadece başarım analizi yapıldığından ortaya çıkan sonuçları buna göre değerlendirmek daha doğru olacaktır. Data güvenliğinin ön planda olduğu veya maliyetin daha önemli olduğu incelemeler başka bir çalışmanın konusu olabilir.

Simulasyon Programı:

IEEE 802.X protokolleri arasındaki başarım özelliklerini ölçmek için hazırlanan simulasyon programında sadece protokol mantığı incelenmiş, hata olasılığı, iletişim ağındaki fiziksel arıza olasılığı, transmisyon gecikmeleri ve ağdaki bilgisayarların kendi hızı gibi parametreler hesaba katılmamıştır. Bundan başka ağlar arasında (internetworking) ve farklı sistemler arasındaki haberleşme (gateway) gibi uygulamalar da simulasyon dışı bırakılmıştır. Sonuç olarak hazırlanan simulasyon programı mükemmel çalıştığı farzedilen bir iletişim ağı üzerinde çalışan bu protokoller arasında sadece başarım analizi yapmak için yazılmıştır.

Simulasyon yönteminin seçilmesinin nedeni yerel iletişim ağlarındaki çalışmanın kolayca modellenmesi, gerçek ortamdaki değişmelerin simulasyonda gösterilebilmesi ve karmaşık hesaplara girmeden sonuç alınabilmesidir.



Şekil.4.1.1: Simulasyon Programının Blok Yapısı.

Simulasyon programının blok yapısı Şekil.4.1.1'de gösterilmiştir. Program Nesneye Yönelik (Object Oriented) bir programlama dili olan **Turbo C++** ile yazılmıştır. Bilindiği gibi C++ dili AT&T laboratuarlarında yine AT&T tarafından üretilen sayısal santralların programlanması için geliştirilmiştir. Simulasyon programı geliştirilirken C++ dilinin yapısal özelliğinden yararlanılarak önce **Byte** ve **Packet** nesneleri tanımlanmış, daha sonra bu nesnelerden **Ethernet**, **Token Bus** ve **Token Ring** paketlerini temsil eden nesneler türetilmiştir. C++'ın doğası gereği protokollerin çalışma mantığı bu modüllerde kolayca modellenmiştir. Bu modüllerde paketlerin oluşturulması, gönderilmesi ve oluşabilecek hatalar tanımlanmıştır. IEEE 802.X protokollerinin tanımlandığı nesneler **Ana Program'a** bağımlı olarak çalışmaktadır. Ana program modülünde kullanıcı tarafından protokol tipi, paket uzunluğu,terminal sayısı, test sayısı gibi simulasyon parametreleri ve çıkış ünitesi seçimi yapılır. Ana program bu parametrelere göre ilgili modülü çağırıp dönen sonuçları listeler. Kullanıcının seçimine göre sonuçlar ekrana, yazıcıya, "TEZRAPOR" adlı dosyaya veya grafik ekrana (aynı zamanda dosyaya) alınabilir.

Simulasyon programının akış diagramları ve program listeleri eklerde bulunmaktadır. Program listeleri tamamen bu tez çalışması için hazırlanmış olup ticari amaçla kullanılmamalıdır.

5.1 SONUÇLAR

IEEE 802 protokollerinin başarım açısından incelenmesi için hazırlanan simulasyon programının sonuçları eklerde bulunmaktadır. Sonuçlar grafik ve tablo halinde ayrı ayrı verilmiştir.

Programın verdiği sonuçları yorumlamadan önce simulasyon için yapılan modellemenin sadece mükemmel çalışan bir ağ için yapıldığı tekrar hatırlanmalıdır. Simulasyon sonuçları iletişim ağı üzerinde saniyede iletilen byte sayısının terminal sayısı ile olan değişimi şeklindedir.

Grafiğin y ekseni saniye başına gönderilen byte sayısı, x ekseni ise sistemde o anda çalışan terminal sayısını göstermektedir. Başarım analizi terminal sayısı ile saniye başına gönderilecek byte sayısı üzerinde yapılmıştır. Simulasyon çeşitli paket uzunluklarına göre tekrarlanarak paket uzunluklarındaki değişikliklerde protokollerin nasıl performans verdiği araştırılmıştır. Bilindiği gibi bütün bu protokollerde paket uzunlukları belirli oranlarda değişebildiğinden net data hızı byte cinsinden belirlenmiştir.

Grafikler incelenecek olursa sonuçların teorik hesaplamalara hemen hemen yaklaştığı görülür. Bilindiği gibi istatistikte deney sayısı ne kadar fazla olursa sonucun doğruluğu o kadar fazla olur. Bu programda yapılan deneyler 1000-10000 arasında tekrarlanmıştır. Çeşitli paket uzunluklarına göre yapılan incelemeler aşağıda analiz edilmiştir.

1. grafik 1512 byte'lık paket uzunluğunda 1000 test yapılarak elde edilmiştir. Grafikten de görülebileceği gibi terminal sayısı beşten az iken saniyede gönderilen byte sayısında büyük farklılıklar olmaktadır. Terminal sayısı arttıkça başarımdaki en fazla düşüş Token Ring protokolünde olmuştur. Ethernet'te de benzer bir düşüş vardır, ancak yine de byte sayısı diğer iki protokolden daha fazladır. Token Bus protokolü hız açısından Ethernet ve Token Ring arasında bulunmasına rağmen terminal sayısına pek bağımlı değildir.

2. grafik 4096 byte'lık paket uzunluğunda 5000 test yapılarak elde edilmiştir. Ancak Ethernet standardında maksimum paket uzunluğu 1546 byte olduğundan Ethernet grafiği bu paket uzunluğuna göre çizilmiştir. Bu grafik 1. grafikle genel olarak karşılaştırılırsa bütün protokollerde byte sayısında küçük düşüşler görülmektedir. Bu grafikteki Token Bus eğrisinde terminal sayısı ile data hızı arasındaki eğim daha fazla olmuştur. Token Ring eğrisinin ise 1. grafikle aynı çıkması Token Ring protokolünde hızın paket uzunluğuyla değişmemesi gibi ilginç bir sonuç vermektedir.

3. grafik 512 byte'lık paketlerde 2000 test ile elde edilmiştir. Bu paket uzunluğundaki başarım hemen hemen aynıdır. Bu durum büyük ve küçük paket paketlerde net data hızında benzer düşüşlerin olduğu şeklinde açıklanabilir.

Yukarıda yorumlanan grafiklerden ortaya çıkan ortak sonuçlar ise:

Ethernet protokolünün terminal sayısındaki değişmelerle etkilenmesine rağmen 10 Mbs'lik bit hızının getirdiği avantajla en hızlı protokol olmaya devam etmesidir. Ancak bu avantajına karşın Ethernet'in mesafe kısıtlaması, bus topolojinin getirdiği güvensizlik gibi simulasyona katılmayan dezavantajları da gözönüne alınmalıdır.

Token Bus protokolünün hızı Ethernet'e nazaran düşük olmasına rağmen terminal sayısı arttıkça net data hızının düşüşün daha az olması bu protokolün esnekliği, güvenirliliği ve mesafe avantajı gibi diğer özellikleri de gözönüne alınırsa büyük ölçekli ağlarda seçilebileceğini gösterir.

Terminal sayısından en fazla etkilenen protokol Token Ring olarak ortaya çıkmaktadır. Simulasyonda Token Ring protokolünde standart bit hızı olan 4 Mbps olarak seçilmiştir, ancak Token Ring protokolünün yeni versiyonunun bit hızı 16 Mbps olduğundan bu hız tabii ki net data hızına doğrudan etki etkileyecektir.

Yerel İletişim Ağlarında IEEE 802.X protokollerinde yapılan bu incelemenin kapsamı sınırlı tutulmuştur. Zamanla bu protokollere yenileri de eklenecektir. Örneğin FDDI protokolünün standartlaştırılma çalışmaları yapılmaktadır.

Bir iletişim ağı tasarlanırken sadece hız açısından yola çıkmak yanlış olacaktır. İdeal tasarım kurulacak sistemin gereksinimlerini iyi saptayıp; güvenlik, büyüyebilirlik, hız, maliyet ve esneklik gibi ölçütlere yanıt verecek en iyi sistemin seçilmesiyle yapılabilir.

6.1 EKLER

6.1.1 PROGRAM LİSTELERİ

```
// BYTE.H  18/12/90   (C) Mehmet Kavi 881601
////////////////////////////////////////////////////////////////////////////
//                          CLASS BYTE                                    //
// Constants : None                                                       //
//                                                                        //
// Variables : Loc, Byte                                                  //
//                                                                        //
// Functions :                                                            //
//                                                                        //
//      Byte(byte c)                                                      //
//      Byte()                                                            //
//      SetLoc(byte i)                                                    //
//      GetLoc()                                                          //
//      SetBit(byte i)                                                    //
//      ResetBit(byte i)                                                  //
//      GetBit(byte i)                                                    //
//      SetByte(byte c)                                                   //
//      *ByteToStr(void)                                                  //
//      StrToByte(char *)                                                 //
////////////////////////////////////////////////////////////////////////////

class Byte
{
 protected:
  byte Loc;
public:
  byte Byte;
  Byte(byte c)          { SetLoc(0); SetByte(c); };
  Byte()                { SetByte(0); SetLoc(0); };
  byte GetBit(byte);
  void SetLoc(byte i)   { Loc=i; };
  byte GetLoc()         { return Loc; };
  void SetBit(byte i)   { byte c; c=pow(2,i);      Byte |=c; };
  void ResetBit(byte i) { byte c; c=0xFF-pow(2,i); Byte &=c; };
  void SetByte(byte c)  { Byte=c; };
  char * ByteToStr(void);
  void StrToByte(char *);
};

byte Byte::GetBit(byte i)
{ byte c;
  c=pow(2,i); if (Byte & c) return 1; else return 0;
};

void Byte::StrToByte(char *str)
{ byte i;
  Byte=0;
  for (i=0;i<8;i++) if (str[i]=='1') Byte+=pow(2,i);
};
```

```
// PACKET.H  13/01/91 (C) Mehmet Kavi 881601
////////////////////////////////////////////////////////////////////////////
//                  CLASS PACKET  (Derived from BYTE)                     //
//                                                                        //
// Constants: MaxPacketSize, MaxFields                                    //
//                                                                        //
// Variables: PacketReady,InputBufferEmpty,OutputBufferEmpty,FatalError   //
//            Timer, PacketSize, Field, FieldOffs[] FieldSize             //
// Functions:                                                             //
//                                                                        //
// Packet(field)                                                          //
// Packet(field, fieldoffs[], fieldsize[])                                //
// ~Packet()                                                              //
// SetPacketReady()                                                       //
// isPacketReady()                                                        //
// isPacketReceived()                                                     //
// GetField()                                                             //
// SetField(field)                                                        //
// GetFieldOffs(field)                                                    //
// SetFieldOffs(field, offs)                                              //
// GetFieldSize(field)                                                    //
// SetFieldSize(field, size)                                              //
// GetPacketSize()                                                        //
// SetPacketSize(PackLen)                                                 //
// SetPacketFormat(field, fieldoffs[], fieldsize[])                       //
// SetOffsByte(field, offset, byte c)                                     //
// GetOffsByte(field, offset)                                             //
// SetOffsBit(field, offset)                                              //
// ResetOffsBit(field, offset)                                            //
// GetOffsBit(field, offset)                                              //
// *OffsByteToStr(field, offset)                                          //
// OffsStrToByte(field, offset)                                           //
// ResetTimer(void)                                                       //
////////////////////////////////////////////////////////////////////////////

class Packet
{
protected:
  Boolean PacketReady;
  Boolean InputBufferEmpty;
  Boolean OutputBufferEmpty;
  Boolean FatalError;
  float Timer;
  byte Field;
  word FieldOffs[MaxFields];
  word FieldSize[MaxFields];
  Byte *Packet;
```

```
public:
  word PacketSize;
  Packet(byte field);
  Packet(byte field, word fieldoffs[], word fieldsize[]);
  ~Packet()                    { delete Packet; };
  void SetPacketReady()        { OutputBufferEmpty=false; PacketReady=true; };
  Boolean isPacketReady()      { return PacketReady;                        };
  Boolean isPacketReceived();
  void ClearInputBuffer()      { InputBufferEmpty=true; };
  byte GetField(void)          { return Field;                              };
  void SetField(byte field)    { if (field<MaxFields) Field=field;          };
  word GetFieldOffs(byte field) { return FieldOffs[field];                  };
  void SetFieldOffs(byte field, word offs) { FieldOffs[field]=offs;         };
  word GetFieldSize(byte field)            { return FieldSize[field];       };
  void SetFieldSize(byte field, word size) { FieldSize[field]=size;         };
  word GetPacketSize(void)                 { return PacketSize;             };
  void SetPacketSize(word PackLen);
  void SetPacketFormat(byte field, word fieldoffs[], word fieldsize[]);
  void SetOffsByte(byte field, word offset, byte c);
  byte GetOffsByte(byte field, word offset);
  void SetOffsBit(byte field, word offset);
  void ResetOffsBit(byte field, word offset);
  byte GetOffsBit(byte field, word offset);
  char * OffsByteToStr(byte field, word offset);
  void OffsStrToByte(byte field, word offset, char *str);
  void ResetTimer(void) { Timer=0; };
};

Packet::Packet(byte field, word fieldoffs[], word fieldsize[])
{ Packet=new Byte[MaxPacketSize];
  SetPacketFormat(field, fieldoffs, fieldsize);
  PacketReady=false;
};

Packet::Packet(byte field)
{ Packet=new Byte[MaxPacketSize];
  if (field==0) Field=MaxFields; else Field=field;
  PacketReady=false;
};
Boolean Packet::isPacketReceived()
{ InputBufferEmpty=false;
  if (InputBufferEmpty) return false;
  return true;
};

void Packet::SetPacketSize(word PackLen)
{ if (PackLen==0) PacketSize=MaxPacketSize;
    else PacketSize=PackLen;
};
```

```
void Packet::SetPacketFormat(byte field, word fieldoffs[], word fieldsize[])
{ word i;
  Field=field;
  for (i=0;i<Field;i++)
    { SetFieldOffs(i,fieldoffs[i]);
      SetFieldSize(i,fieldsize[i]);
    }
};

void Packet::SetOffsByte(byte field, word offset, byte c)
{ Packet[FieldOffs[field]+offset].SetByte(c); };

byte Packet::GetOffsByte(byte field, word offset)
{ return Packet[FieldOffs[field]+offset].Byte; };

void Packet::SetOffsBit(byte field, word offset)
{ div_t x;
  x=div(offset,8);
  if (offset<FieldSize[field]) Packet[FieldOffs[field]+x.quot].SetBit(x.rem);
};

void Packet::ResetOffsBit(byte field, word offset)
{ div_t x;
  x=div(offset,8);
  if (offset<FieldSize[field]) Packet[FieldOffs[field]+x.quot].ResetBit(x.rem);
};

byte Packet::GetOffsBit(byte field, word offset)
{ div_t x;
  x=div(offset,8);
  if (offset<FieldSize[field])
      return Packet[FieldOffs[field]+x.quot].GetBit(x.rem);
};

char * Packet::OffsByteToStr(byte field, word offset)
{ return Packet[FieldOffs[field]+offset].ByteToStr(); };

void Packet::OffsStrToByte(byte field, word offset, char *str)
{ Packet[FieldOffs[field]+offset].StrToByte(str); };
```

```
// ETHERNET.H   27/01/91 (C) Mehmet Kavi 881601
////////////////////////////////////////////////////////////////////////////
//                CLASS ETHERNET (Derived from PACKET)                    //
//                                                                        //
// Constants: EthBitRate, EthInterframeGap, EthMaxFrSize, EthMinFrSize    //
//            EthMaxStations, AttemptLimit                                //
//            EthPrSize, EthPrOffs, EthSFDSize, EthSFDOffs, EthDASize     //
//            EthDAOffs, EthSASize, EthSAOffs, EthLenSize, EthLenOffs     //
//            EthDataOffs, EthFCSSize                                     //
// Variables: DataSize, PacketSize, BusFree, NoCollision, Pad             //
// Functions:                                                             //
//  isBusFree(CurrTerminal)                                               //
//  isNoCollision(CurrTerminal)                                           //
//  Ethernet(datalen, Ethfield, Ethfieldoffs[],Ethfieldsize[])            //
//  SetPreamble()                                                         //
//  SetSFD()                                                              //
//  SetDestAddr(local, group)                                             //
//  SetSourceAddr()                                                       //
//  SetLen()                                                              //
//  SetData()                                                             //
//  CalcFCS()                                                             //
//  MakeEthernetPacket()                                                  //
//  SendEthernetPacket()                                                  //
//  ResetTimer()                                                          //
//  SendJamBits()                                                         //
//  WaitInterFrameGap()                                                   //
//  WaitRndTime()                                                         //
////////////////////////////////////////////////////////////////////////////

class Ethernet : public Packet
{protected:
  word DataSize;
  Boolean BusFree;
  Boolean NoCollision;
public:
  Boolean isBusFree();
  Boolean isNoCollision();
  Ethernet(word datalen, byte Ethfield, word Ethfieldoffs[], word Ethfieldsize[]);
  void SetPreamble(void);
  void SetSFD(void) { SetOffsByte(1,0,0xAB); };
  void SetDestAddr(byte local, byte group);
  void SetSourceAddr(void);
  void SetLen(void);
  void SetData(void);
  void CalcFCS(void);
  void MakeEthernetPacket(void);
  void SendEthernetPacket(void) { Timer+=PacketSize*EthBitTime; };
  void ShowEthernetPacket(void);
  void SendJamBits(void)        { Timer+=EthJamSize*EthBitTime; };
  void WaitInterFrameGap(void)  { Timer+=EthInterFrameGap;};
  byte WaitRndTime(void);
  float FrameTransmission(word Test);
};
```

```
Ethernet::Ethernet(word datalen, byte Ethfield, word Ethfieldoffs[], word Ethfieldsize[])
        : Packet(Ethfield, Ethfieldoffs, Ethfieldsize)
{ word i;
  EthDataSize=datalen;
  if (EthDataSize>EthMaxPacketSize) EthDataSize=EthMaxPacketSize;
  PacketSize=0; for(i=0;i<Ethfield;i++) PacketSize+=Ethfieldsize[i];
  if (PacketSize<EthMinPacketSize) { EthDataSize=486; EthPad=EthDataSize-datalen; }
  Ethfieldsize[5]=EthDataSize; FatalError=false;
  EthFCSOffs=EthDataSize+EthDataOffs;
  PacketSize+=EthDataSize; PacketReady=false;
  SetPacketFormat(Ethfield, Ethfieldoffs, Ethfieldsize);
};

void Ethernet::SetPreamble()
{ word i; for(i=0;i<EthPrSize;i++) SetOffsByte(0,i,0xAA); };

void Ethernet::SetDestAddr(byte local, byte group)
{ word i,same=0;
l1:
  for (i=0;i<EthDASize;i++)
    { SetOffsByte(2,i,Uniform(0,255));
      if (GetOffsByte(2,i)==GetOffsByte(3,i)) same++;
    }
  if (local) SetOffsBit(2,0); else ResetOffsBit(2,0);
  if (group) SetOffsBit(2,1); else ResetOffsBit(2,1);
  if (same==EthSASize) goto l1;
};

void Ethernet::SetSourceAddr()
{ word i; for (i=0; i<EthSASize; i++) SetOffsByte(3,i,Uniform(0,255)); };

void Ethernet::SetLen(void)
{ word tmplen=0;
  byte i;
  for(i=0;i<Ethfield;i++) tmplen+=Ethfieldsize[i];
  SetOffsByte(4,0,tmplen & 0xFF); SetOffsByte(4,1,tmplen >> 8);
};

void Ethernet::SetData()
{ word i;
  while(i<EthDataSize)
   { if (i<(EthDataSize-EthPad)) SetOffsByte(5,i,Uniform(0,255));
       else SetOffsByte(5,i,0xAA);
     i++;
   }
};

void Ethernet::CalcFCS()
{ byte i; for(i=0;i<EthFCSSize;i++) SetOffsByte(6,i,0xAA); };
```

```
void Ethernet::MakeEthernetPacket()
{ SetPreamble();
  SetSFD();
  SetSourceAddr();
  SetDestAddr(0,0);
  SetLen();
  SetData();
  CalcFCS();
  SetPacketReady();
};

void Ethernet::ShowEthernetPacket(void)
{ word i,j,k=0;
  for (j=0; j<Ethfield;j++)
   { for (i=0; i<Ethfieldsize[j]; i++) { printf("%d %X   %s\n",k,GetOffsByte(j,i),OffsByteToStr(j,i)); k++; }
     getch();
     printf("\n\n");
   }
};

Boolean Ethernet::isBusFree()
{ word Busy;
  Busy=Uniform(0,CurrTerminal);
  if (Busy<CurrTerminal) BusFree=true; else BusFree=false;
  return BusFree;
};

Boolean Ethernet::isNoCollision(void)
{ word Collision;
  Collision=Poisson(CurrTerminal);
  if (Collision<CurrTerminal) NoCollision=false; else NoCollision=true;
  if (Collision) Timer+=SlotTime;
  return NoCollision;
};

byte Ethernet::WaitRndTime()
{ byte Slot,R;
  R=pow(2,Attempt); Slot=Uniform(0,R);
   if (Attempt==1) Slot=Uniform(0,1);
   if (Slot>10) Slot=10;
  Timer+=Slot*SlotTime;
  return Slot;
};
```

```
float Ethernet::FrameTransmission(word Test)
{ word i=0;
  Attempt=0; ResetTimer();
  while (i<Test)
   { while (isBusFree())
      { WaitInterFrameGap();
        SendEthernetPacket();
        if (isNoCollision()) { Attempt=0; i++; }
           else
            { SendJamBits();
              Attempt++;
              if (Attempt>EthAttemptLimit)
                { FatalError=true;
                  textattr(REV);
                  gotoxy(2,23); cputs("***** Network Error !  *****");
                  gotoxy(2,24); cputs(" Herhangi Bir Tuşa Basın... ");
                  getch();
                  exit(1);
                }
              WaitRndTime();
            }
      }
   }
 return((Test/Timer)*PacketSize/8);
};
```

```
// TOKENBUS.H   20/01/91 (C) Mehmet Kavi 881601
////////////////////////////////////////////////////////////////////////////
//                CLASS TOKEN BUS (Derived from PACKET)                   //
//                                                                        //
// Constants: ToBBitRate, ToBMaxFrSize, ToBMinFrSize, ToBfield           //
//            ToBPrOffs, ToBSFDOffs, ToBDAOffs, ToBFrContOffs,            //
//            ToBSAOffs,                                                  //
//            ToBPrSize, ToBSFDSize, ToBDASize, ToBFrContSize,            //
//            ToBSASize, ToBInfoOffs, ToBFCSSize, ToBEFDSize              //
// Variables: InfoSize, ToBInfoSize, ToBFCSOffs, ToBEFDOffs               //
//            PacketSize, TokenAvail, PacketIsToken, Priority             //
// Functions:                                                             //
//                                                                        //
// TokenBus(datalen, ToBfield, ToBfieldoffs[],ToBfieldsize[])             //
// SetPreamble()                                                          //
// SetSFD()                                                               //
// SetFrameCont()                                                         //
// SetDestAddr()                                                          //
// SetSourceAddr()                                                        //
// SetInfo()                                                              //
// CalcFCS()                                                              //
// SetEFD()                                                               //
// void MakeTokenBusPacket()                                              //
// void ShowTokenBusPacket()                                              //
// void SendTokenBusPacket()                                              //
// float FrameTransmission()                                              //
////////////////////////////////////////////////////////////////////////////

class TokenBus : public Packet
{
protected:
  Boolean TokenAvail;
  Boolean PacketIsToken;
  Boolean Priority;
  word DataSize;
public:
  Boolean isPriorityOK();
  Boolean isTokenAvail(void);
  Boolean isPacketToken();
  TokenBus(word datalen, byte ToBfield, word ToBfieldoffs[], word ToBfieldsize[]);
  void SetPreamble(void);
  void SetSFD(void)          { SetOffsByte(1,0,0x11); };
  void SetFrameCont(void)    { SetOffsByte(2,0,0x22); };
  void SetDestAddr(void);
  void SetSourceAddr(void);
  void SetInfo(void);
  void CalcFCS(void);
  void SetEFD(void)              { SetOffsByte(7,0,0xEE); };
  void MakeTokenBusPacket(void);
  void ShowTokenBusPacket(void);
  void SendTokenBusPacket(void)  { Timer+=PacketSize*ToBBitTime; };
  float FrameTransmission(word Test);
};
```

```
TokenBus:: TokenBus(word datalen, byte ToBfield, word ToBfieldoffs[], word ToBfieldsize[])
    : Packet(ToBfield, ToBfieldoffs, ToBfieldsize)
{ word  i;
  ToBInfoSize=datalen;
  if (ToBInfoSize>ToBMaxPacketSize) ToBInfoSize=ToBMaxPacketSize;
  ToBfieldsize[5]=ToBInfoSize;
  ToBfieldoffs[6]=ToBfieldsize[0]+14+ToBInfoSize;
  ToBfieldoffs[7]=ToBfieldoffs[6]+4;
  PacketSize=0; for(i=0;i<ToBfield;i++) PacketSize+=ToBfieldsize[i];
  PacketReady=false;
  SetPacketFormat(ToBfield, ToBfieldoffs, ToBfieldsize);
};

void TokenBus::SetPreamble(void)
 { word i; for (i=0; i<ToBPrSize; i++) SetOffsByte(0,i,0x00); };

void TokenBus::SetDestAddr()
{ word i,same=1;
l1:
  for (i=0;i<ToBDASize;i++)
     { SetOffsByte(3,i,Uniform(0,0xff));
       if (GetOffsByte(3,i)==GetOffsByte(4,i)) same++;
     }
  if (same==ToBSASize) goto l1;
};

void TokenBus::SetSourceAddr()
{ word i; for (i=0; i<ToBSASize; i++) SetOffsByte(4,i,Uniform(0,255)); };

void TokenBus::SetInfo()
{ word i; for(i=0;i<ToBInfoSize;i++) SetOffsByte(5,i,Uniform(0,255)); };

void TokenBus::CalcFCS(void)
{ byte i; for(i=0;i<ToBFCSSize;i++) SetOffsByte(6,i,0xAA); };

void TokenBus::MakeTokenBusPacket()
{ SetPreamble();
  SetSFD();
  SetFrameCont();
  SetDestAddr();
  SetSourceAddr();
  SetInfo();
  CalcFCS();
  SetEFD();
  SetPacketReady();
};
```

```
void TokenBus::ShowTokenBusPacket(void)
{ word i,j,k=0;
  for (j=0; j<ToBfield;j++)
   { for (i=0;i<ToBfieldsize[j];i++)
       { printf("%d %X %s\n",k,GetOffsByte(j,i),OffsByteToStr(j,i)); k++; }
     getch();
     printf("\n\n");
   }
};

Boolean TokenBus::isTokenAvail(void)
{ if (CurrTerminal<Uniform(1,CurrTerminal)) TokenAvail=true;
  Timer+=ToBBitTime*Uniform(1,CurrTerminal);
  return TokenAvail;
};

Boolean TokenBus::isPriorityOK()
{ Priority=false;
  if (Uniform(0,2)==1) Priority=true;
  return Priority;
};

Boolean TokenBus::isPacketToken()
{ PacketIsToken=false;
  if (Uniform(0,2)==1) PacketIsToken=true;
  if (PacketIsToken==false) Timer+=CurrTerminal*ToBBitTime;
  return PacketIsToken;
};

float TokenBus::FrameTransmission(word Test)
{ word i=0;
  ResetTimer();
  while (i<Test)
   { if (isPacketReceived())
       { if (isPacketToken() && isPriorityOK())
           { SendTokenBusPacket();
             i++;
           }
         else SendTokenBusPacket();
       }
   }
 return((Test/(Timer))*PacketSize);
};
```

```
// TOKENRNG.H   27/01/91 (C) Mehmet Kavi 881601
////////////////////////////////////////////////////////////////////////////
//          CLASS TOKEN RING (Derived from PACKET)                        //
//                                                                        //
// Constants: ToKSDOffs, ToKAccContOffs, ToKDAOffs, ToKSAOffs,           //
//            ToKInfoOffs,                                                //
//            ToKSDSize, ToKAccContSize, ToKDASize, ToKSASize,            //
//            ToKFCSSize, ToKEFDSize                                      //
//                                                                        //
// Variables: ToKInfoSize, ToKFCSOffs, ToKEFDOffs, PacketSize             //
//            TokenAvail, PacketIsToken                                   //
// Functions:                                                             //
//                                                                        //
// TokenRing(datalen, ToKfield, ToKfieldoffs[],ToKfieldsize[])            //
// SetSD()                                                                //
// SetAccCont()                                                           //
// SetDestAddr()                                                          //
// SetSourceAddr()                                                        //
// SetInfo()                                                              //
// CalcFCS()                                                              //
// SetEFD()                                                               //
// MakeTokenRingPacket()                                                  //
// ShowTokenRingPacket()                                                  //
// isTokenAvail()                                                         //
// isPriorityOK()                                                         //
////////////////////////////////////////////////////////////////////////////

class TokenRing : public Packet
{
protected:
  Boolean TokenAvail;
  Boolean PacketIsToken;
  Boolean Priority;
  word InfoSize;
public:
  TokenRing(word datalen, byte ToKfield, word *ToKfieldoffs, word *ToKfieldsize);
  void SetSD(void);
  void SetAccCont(void)      { SetOffsByte(1,0,0x11); };
  void SetDestAddr(void);
  void SetSourceAddr(void);
  void SetInfo(void);
  void CalcFCS(void);
  void SetEFD(void);
  void MakeTokenRingPacket(void);
  void ShowTokenRingPacket(void);
  Boolean isTokenAvail(void);
  Boolean isPriorityOK(void);
  void SendTokenRingPacket(void) { Timer+=PacketSize*ToKBitTime; };
  float FrameTransmission(word Test);
};
```

```
TokenRing:: TokenRing(word datalen, byte  ToKfield, word *ToKfieldoffs, word  *ToKfieldsize)
          : Packet(ToKfield. ToKfieldoffs. ToKfieldsize)
{ word i;
  ToKInfoSize=datalen;
  if (ToKInfoSize>ToKMaxPacketSize) ToKInfoSize=ToKMaxPacketSize;
    ToKfieldsize[4]=ToKInfoSize;
    ToKfieldoffs[5]=ToKfieldsize[0]+15+ToKInfoSize;
    ToKfieldoffs[6]=ToKfieldoffs[5]+4;
  PacketSize=0; for(i=0;i<ToKfield;i++) PacketSize+=ToKfieldsize[i];
  PacketReady=false; OutputBufferEmpty=true;
  PacketIsToken=true;
  SetPacketFormat(ToKfield, ToKfieldoffs, ToKfieldsize);
};

void TokenRing::SetSD()
{ word i; for (i=0;i<ToKSDSize;i++) SetOffsByte(0,i,0x00); };

void TokenRing::SetDestAddr()
{ word i,same=1;
l1:
  for (i=0;i<ToKDASize;i++)
    { SetOffsByte(2,i,Uniform(0,255));
      if (GetOffsByte(2,i)==GetOffsByte(3,i)) same++;
    }
  if (same==ToKSASize) goto l1;
};

void TokenRing::SetSourceAddr()
{ word i; for (i=0; i<ToKSASize; i++) SetOffsByte(3,i,Uniform(0,255)); };

void TokenRing::SetInfo()
{ word i; for(i=0;i<ToKInfoSize;i++) SetOffsByte(4,i,Uniform(0,255)); };

void TokenRing::CalcFCS(void)
{ byte i; for(i=0;i<ToKFCSSize;i++) SetOffsByte(5,i,0xFC); };

void TokenRing::SetEFD()
{ word i; for (i=0;i<ToKEFDSize;i++) SetOffsByte(7,0,0xFD); };

void TokenRing::MakeTokenRingPacket()
{ SetSD();
  SetAccCont();
  SetDestAddr();
  SetSourceAddr();
  SetInfo();
  CalcFCS();
  SetEFD();
  SetPacketReady();
};
```

```
void TokenRing::ShowTokenRingPacket(void)
{ word i,j,k=0;
  for (j=0; j<ToKfield;j++)
   { for (i=0; i<ToKfieldsize[j]; i++)
        { printf("%d %X %s\n",k,GetOffsByte(j,i),OffsByteToStr(j,i)); k++; }
     getch();
     printf("\n\n");
   }
};

Boolean TokenRing::isTokenAvail(void)
{ Timer+=ToKBitTime*Uniform(1,CurrTerminal);
  TokenAvail=true;
  return TokenAvail;
};

Boolean TokenRing::isPriorityOK()
{ Priority=false;
  if (Uniform(0,1)==1) Priority=true;
  return Priority;
};

float TokenRing::FrameTransmission(word Test)
{ word i=0,j=0;
  ResetTimer();
  while (i<Test)
   { if (isPacketReceived())
      { if (PacketIsToken && isPriorityOK())
         { for (j=0;j<CurrTerminal;j++) SendTokenRingPacket();
           i++;
         }
         else for (j=0;j<CurrTerminal;j++) SendTokenRingPacket();
      }
   }
 return((Test/(Timer))*PacketSize);
 };
```

```
// TEZGLOB.H   26/01/91 (C) Mehmet Kavi 881601
////////////////////////////////////////////////////////////////////////////
//                                                                        //
//  Global constants, functions and variables...                          //
//                                                                        //
////////////////////////////////////////////////////////////////////////////
typedef unsigned char byte;
typedef unsigned int  word;

enum Boolean {false=0,true=1};

const byte MaxFields = 16;
word       CurrTerminal;
word       MaxPacketSize=8192;
word       PacketSize;
word       Test;
float      Result[3][51];
byte     *ScrBuff[4096];
char *PacketType[3]={ " IEEE 802.3 (Ethernet )  ",
                      " IEEE 802.4 (Token Bus)  ",
                      " IEEE 802.5 (Token Ring) " };

char *packtype[3]={ " Ethernet   ",
                    " Token Bus  ",
                    " Token Ring " };

char *PackSize[6]={ "    512  ","   1024  ","   1512  ",
                    "   2048  ","   4096  ","   8192  " };

byte packsize[6]={ 512,1024,1512,2048,4096,8192 };

char *TermCnt[10]={ "    5   ","   10   ","   15   ",
                    "   20   ","   25   ","   30   ",
                    "   35   ","   40   ","   45   ",
                    "   50   " };

byte termcnt[10]={ 5,10,15,20,25,30,35,40,45,50 };

char *TestCnt[9]={ "     10  ","     50  ","    100  ",
                   "    250  ","    500  ","   1000  ",
                   "   2000  ","   5000  ","  10000  "
                 };

word testcnt[9]={ 10,50,100,250,500,1000,2000,5000,10000 };

char *Output[4]={ " Ekran   "," Yazıcı  "," Dosya   "," Grafik  " };

byte output[4]={ 0,1,2,3 };
```

```
#define NORM 0x07
#define REV  0x070
#define HIGH 0x0F
#define BLNK 0x8F
#define BS 8
#define FORMFEED 12
#define CR 13
#define ESC 27
#define HOME 327
#define END 335
#define UP 328
#define DOWN 336
#define PGUP 329
#define PGDN 337
#define LEFT 331
#define INS 338
#define RIGHT 333
#define DEL 339
#define F1 315
#define F2 316
#define F3 317
#define F4 318
#define F5 319
#define F6 320
#define F7 321
#define F8 322
#define F9 323
#define F10 324

char *SingleFrame="┌┐└┘│─";
char *DoubleFrame="╔╗╚╝║═";

// Statistical functions ...

float Uniform(float lower, float upper)
{ float tmp;
  tmp=random(upper-lower+1);
  if (upper>=lower) return lower+tmp; else return 0;
}

unsigned int Poisson(long int mean)
{ unsigned int i=0;
  double x=1,y;
  y=exp(-mean);
  while(x>y) { y=exp(-mean); x=x*Uniform(0, mean)/mean; i++; }
  return i;
};
```

```
void Draw(byte x,byte y,byte ch,byte attr)
{ gotoxy(x,y);
  _AH=9;_BH=0;_CX=1;_AL=ch;_BL=attr;
  geninterrupt(0x10);
};

void DrawFrame(byte X1,byte Y1,byte X2,byte Y2,char *hdr,byte Attr,char *Fr)
{ byte i;
 textattr(Attr);
 for (i=X1;i<=X2;i++) { Draw(i,Y1,Fr[1],Attr); Draw(i,Y2,Fr[1],Attr); }
 for (i=Y1;i<=Y2;i++) { Draw(X1,i,Fr[3],Attr); Draw(X2,i,Fr[3],Attr); }
 Draw(X1,Y1,Fr[0],Attr); Draw(X2,Y1,Fr[2],Attr);
 Draw(X2,Y2,Fr[4],Attr); Draw(X1,Y2,Fr[6],Attr);
 if (strcmp("\0",hdr)!=0)
   { gotoxy(X1+2,Y1); if (Attr==HIGH) textattr(REV); cputs(hdr); }
}

void DrawHorLine(byte X1,byte X2,byte Y,byte Attr,char *Fr)
{ byte i;
  for (i=X1;i<=X2;i++) Draw(i,Y,Fr[1],Attr);
};

void DrawVertLine(byte Y1,byte Y2,byte X,byte Attr,char *Fr)
{ byte i;
  for (i=Y1;i<=Y2;i++) Draw(X,i,Fr[3],Attr);
};

void SetupScreen()
{ byte i,j=0;
  _setcursortype(_NOCURSOR);
  DrawHorLine(30,80,17,NORM,SingleFrame);
  DrawHorLine(30,80,20,NORM,SingleFrame);
  DrawHorLine(30,80,22,NORM,SingleFrame);
  DrawVertLine(16,25,30,NORM,DoubleFrame);
  for (i=0;i<100;i+=10)
    { if (i<90) DrawVertLine(16,25,30+i,NORM,SingleFrame);
      textattr(REV);
      if (j<5) gotoxy(32+i,16); else gotoxy(32+i-50,21);
      cputs(TermCnt[j]); j++;
    }
  DrawHorLine(30,20,80,NORM,SingleFrame);
  DrawFrame(1,15,80,25," Sonuçlar ",HIGH,DoubleFrame);
  DrawFrame(1,1,80,25," IEEE 802.x Standartları ",HIGH,DoubleFrame);
  textattr(NORM);
  gotoxy(3,17); cputs("Paket Tipi    :");
  gotoxy(3,18); cputs("Paket Uzunluğu:");
  gotoxy(3,19); cputs("Max. Terminal :");
  gotoxy(3,20); cputs("Test  Sayısı  :");
  gotoxy(3,21); cputs("Çıkış         :");
  textattr(REV);
  gotoxy(2,23); cputs(" F10 : Simulasyon          ");
  gotoxy(2,24); cputs(" ESC : İptal               ");
};
```

```
void VersionScreen()
{ byte i;
  window(35,3,76,14);
  textattr(NORM);
  cputs("\n\n\r  Tez Konusu:\n\r");
  cputs("\n     IEEE 802.x Protokollerinin\n\r");
  cputs("     Bilgisayar Simulasyonu ile\n\r");
  cputs("     Performans Analizi\n\r");
  cputs("\n           Mehmet Kavi 881601\n\r");
  cputs("  Y.Ü. Elektronik ve Hab. Müh. Bölümü\n\n\r");
  DrawFrame(1,1,41,12," Info ",HIGH,SingleFrame);
  delay(1000);
  for(i=1;i<15;i++) { delay(20);insline(); }
  window(1,1,80,25);
};

word GetKey(void)
{ word key,lo,hi;
  key=bioskey(0);
  lo=key & 0X00FF;
  hi=(key & 0XFF00) >> 8;
  return((lo==0) ? hi+256 : lo);
}

int CtrlBreak(void)
{ printf("\nProgram kullanıcı tarafından kesildi...\n");
  return (0);
};

FILE *Report;

char *Mess_Header[5]={ "\t\t          Mehmet Kavi 881601\n\r",
                       "\t\t   Y.Ü. Elektronik ve Hab. Müh. Bölümü\n\r",
                       "\t\t      IEEE 802.x Protokollerinin\n\r",
                       "\t\t      Bilgisayar Simulasyonu ile\n\r",
                       "\t\t           Başarım Analizi\n\n\r",
                     };

char *Params[4]={ "\t\t\tPaket Tipi    : ",
                  "\t\t\tPaket Uzunluğu: ",
                  "\t\t\tMax. Terminal : ",
                  "\t\t\tTest  Sayısı  : ",
                };
char *Table_Header[2]=
       { " Terminal      Ethernet      Token Bus      Token Ring ",
         "==========    ==========    ===========    ============"
       };
```

```
void WriteToFile(byte M2, byte M3, byte M4, byte M5)
{ word i;
  switch(M5)
   { case 0: Report=NULL; break;
     case 1: Report=stdprn; break;
     case 2: Report=fopen("TEZRAPOR","wt"); break;
     case 3: if (Result[0][1]==0 || Result[1][1]==0 || Result[2][1]==0)    Report=NULL;
              else Report=fopen("TEZRAPOR","wt");
              break;
   }
  if (M5!=0)
   {  for(i=0;i<5;i++) fprintf(Report,Mess_Header[i]);
      fprintf(Report,"%s%s\n\r",Params[1],PackSize[M2]);
      fprintf(Report,"%s  %s\n\r",Params[2],TermCnt[M3]);
      fprintf(Report,"%s%s\n\n\r",Params[3],TestCnt[M4]);
      for(i=0;i<2;i++) fprintf(Report,"\t%s\n\r",Table_Header[i]);
      for(i=1;i<51;i++)
          fprintf(Report,"\t     %d\t\t%7.0f\t\t%7.0f\t\t%7.0f\n\r", \
              i,Result[0][i],Result[1][i],Result[2][i]);
     if (M5==1) fclose(Report);
   }
};

////////////////////////////// CLASS MENU ////////////////////////////////////

class Menu
{
protected:
   char WinBuff[512];
   Boolean ActWin;
public:
   char Name[80];
   byte X1;
   byte Y1;
   byte X2;
   byte Y2;
   byte Item;
   Menu(char *name, byte x1, byte y1, byte x2, byte y2);
   void DrawMenu(byte Attr, char *Fr);
   word GetMenuID(char **menuitem, byte maxitem, byte item);
   void SaveMenu(void) { gettext(1,1,80,25,ScrBuff); }
   void RestoreMenu(void) { puttext(1,1,80,25,ScrBuff); }
};

Menu::Menu(char *name, byte x1, byte y1, byte x2, byte y2)
{ strcpy(Name,name);
  X1=x1; Y1=y1; X2=x2; Y2=y2;
  Item=0;
};
```

```
void Menu::DrawMenu(byte Attr, char *Fr)
{ byte i;
 textattr(Attr);
 for (i=X1;i<=X2;i++) { Draw(i,Y1,Fr[1],Attr); Draw(i,Y2,Fr[1],Attr); }
 for (i=Y1;i<=Y2;i++) { Draw(X1,i,Fr[3],Attr); Draw(X2,i,Fr[3],Attr); }
   Draw(X1,Y1,Fr[0],Attr); Draw(X2,Y1,Fr[2],Attr);
   Draw(X2,Y2,Fr[4],Attr); Draw(X1,Y2,Fr[6],Attr);
 if (strcmp("\0",Name)!=0)
     { gotoxy(X1+2,Y1); if (Attr==HIGH) textattr(REV); cputs(Name); }
}

word Menu::GetMenuID(char **menuitem, byte maxitem, byte item)
{ byte i=0;
  word k=0;
  _setcursortype(_NOCURSOR);
  if (item==0) item=Item; else Item=item;
  for (i=0;i<maxitem;i++)
      { gotoxy(X1+1,Y1+i+1); textattr(NORM);
        if (i==item) textattr(REV);
        cputs(menuitem[i]);
      }
  i=Item;
  while(k!=ESC &&  k!=F10 && k!=CR && k!=LEFT && k!=RIGHT)
    { k=GetKey();
      if (k==UP)   { gotoxy(X1+1,Y1+i+1); textattr(NORM); cputs(menuitem[i]);
                     if (i>0) i--; else i=maxitem-1; }
      if (k==DOWN) { gotoxy(X1+1,Y1+i+1); textattr(NORM); cputs(menuitem[i]);
                     if (i<maxitem-1) i++; else i=0; }
      gotoxy(X1+1,Y1+i+1); textattr(REV); cputs(menuitem[i]);
    }
  switch (k)
    { case   ESC: item=Item;   break;
      case   F10: item=Item;   break;
      case  LEFT: item=Item=i; break;
      case RIGHT: item=Item=i; break;
      case    CR: item=Item=i; break;
    }
  return k;
};

//////////////////////////// END OF CLASS MENU ////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////
//                                                                        //
// IEEE 802.3 (Ethernet v2.0) Definitions...                              //
//                                                                        //
////////////////////////////////////////////////////////////////////////////

const float EthBitTime=1e-7;
const float EthInterFrameGap=9.6e-6;
const float SlotTime=51.2e-6;
const float EthJamTime=3.2e-6;

const word  EthMaxPacketSize=1518;
const word  EthMinPacketSize=512;
const byte  EthJamSize=32;
const word  EthMaxStations=1024;
const word  EthAttemptLimit=16;

////////////////////////////////////////////////////////////////////////////
//  CSMA/CD Field Definitions...                                          //
//                                                                        //
//                        Fields,            Size,   Offset              //
////////////////////////////-----------////////////------///--------/////////

const    word EthPrSize=7;  // Preamble :              7        0
const    word EthPrOffs=0;
const    word EthSFDSize=1; // Start Frame Delimiter:  1        7
const    word EthSFDOffs=7;
const    word EthDASize=6;  // Destination Address :   6        8
const    word EthDAOffs=8;
const    word EthSASize=6;  // Source Address:         6        14
const    word EthSAOffs=14;
const    word EthLenSize=2; // Len :                   2        20
const    word EthLenOffs=20;
const    word EthDataOffs=22;
const    word EthFCSSize=4; // Frame Check Sequence :  4    22+DataSize
word EthFCSOffs;
word EthDataSize;           // Ethernet Data field size...
word EthPad=0;              // Pad field size (optional)

byte Ethfield=7;
word Ethfieldoffs[7]={ EthPrOffs, EthSFDOffs, EthDAOffs, EthSAOffs,
               EthLenOffs, EthDataOffs, EthFCSOffs},
     Ethfieldsize[7]={ EthPrSize, EthSFDSize, EthDASize, EthSASize,
               EthLenSize, EthDataSize, EthFCSSize};
```

```
////////////////////////////////////////////////////////////////////////////
//                                                                        //
// IEEE 802.4 (Token Bus)  Definitions ...                                //
//                                                                        //
////////////////////////////////////////////////////////////////////////////

float ToBBitTime=4e-7;       //  1, 4, 5, 10 MBps..
const word   ToBMaxPacketSize=8192;
const word   ToBMinPacketSize=20;

////////////////////////////////////////////////////////////////////////////
//   Token Bus Field Definitions...                                       //
//                                                                        //
//                          Fields,          Size,    Offset             //
////////////////////////////----------/////////////-------//--------/////////
const    word ToBPrSize=1;     // Preamble :          1         0
const    word ToBPrOffs=0;
const    word ToBSFDSize=1;    // Start Frame Delimiter: 1       1
const    word ToBSFDOffs=1;
const    word ToBFrContSize=1; // Frame Control :     1         2
const    word ToBFrContOffs=2;
const    word ToBDASize=6;     // Destination Address :  6      2
const    word ToBDAOffs=3;
const    word ToBSASize=6;     // Source Address:        6      8
const    word ToBSAOffs=9;
         word ToBInfoSize;     // MAC Info Field       0-8191
const    word ToBInfoOffs=15;
const    word ToBFCSSize=4;    // Frame Check Sequence : 4
         word ToBFCSOffs;
const    word ToBEFDSize=1;    // Frame Delimiter :      1
         word ToBEFDOffs;

byte ToBfield=8;
word ToBfieldoffs[8]={ ToBPrOffs, ToBSFDOffs,  ToBFrContOffs, ToBDAOffs,
                ToBSAOffs, ToBInfoOffs, ToBFCSOffs,    ToBEFDOffs },
     ToBfieldsize[8]={ ToBPrSize, ToBSFDSize,  ToBFrContSize, ToBDASize,
                ToBSASize, ToBInfoSize, ToBFCSSize,    ToBEFDSize };
```

```
////////////////////////////////////////////////////////////////////////////
//                                                                        //
// IEEE 802.5 (Token Ring)  Definitions ...                               //
//                                                                        //
////////////////////////////////////////////////////////////////////////////

float ToKBitTime=2e-7;    // 4, 16 MBps.. (6.25e-8 sec.)
const word   ToKMaxPacketSize=4100;
const word   ToKMinPacketSize=20;

////////////////////////////////////////////////////////////////////////////
//   Token Ring Field Definitions...                                      //
//                                                                        //
//                                  Fields,          Size,    Offset     //
////////////////////////////////////////----------/////////-------////--------//
const    word ToKSDSize=2;      // Start Delimiter       1         0
const    word ToKSDOffs=0;
const    word ToKAccContSize=1; // Start Frame Delimiter 1         1
const    word ToKAccContOffs=2;
const    word ToKDASize=6;      // Destination Address   6         2
const    word ToKDAOffs=3;
const    word ToKSASize=6;      // Source Address:       6         8
const    word ToKSAOffs=9;
         word ToKInfoSize;      // MAC Info Field     0-4099
const    word ToKInfoOffs=15;
const    word ToKFCSSize=4;     // Frame Check Sequence  4
         word ToKFCSOffs;
const    word ToKEFDSize=2;     // Frame Delimiter       2
         word ToKEFDOffs;

byte ToKfield=7;
word ToKfieldoffs[7]={ ToKSDOffs, ToKAccContOffs, ToKDAOffs, ToKSAOffs,
                       ToKInfoOffs, ToKFCSOffs, ToKEFDOffs},
     ToKfieldsize[7]={ ToKSDSize, ToKAccContSize, ToKDASize, ToKSASize,
                       ToKInfoSize, ToKFCSSize, ToKEFDSize};
```

```
// TEZGRAPH.H 26/01/91 (C) Mehaet Kavi 881601
////////////////////////////////////////////////////////////////////////
//                                                                    //
//  Graphics Functions, Variables....                                 //
//                                                                    //
////////////////////////////////////////////////////////////////////////

word MaxX,MaxY,XOffs=50,YOffs=30;
byte ScrX,ScrY;

void DrawCoords()
{  word i,j;
   char *GrMessage=new char[80];
   char *Term[10]={ " 5","10","15","20","25","30","35","40","45","50" };
   char *Res[6]={ "1000"," 800"," 600"," 400"," 200","   0" };
   char *Lbl[6]={ "BYTE/SN (x1000)",
                  "TERMINAL SAYISI",
                  "IEEE 802.x PROTOKOLLERINDE BASARIM ANALIZI",
                  "Ethernet","Token Bus","Token Ring"
                };
  MaxX=getmaxx(); MaxY=getmaxy();
  setlinestyle(SOLID_LINE,0,THICK_WIDTH);
  cleardevice();
  line(XOffs,YOffs-15,XOffs,MaxY-YOffs);
    for(i=MaxY-YOffs;i<MaxY;i-=50) outtextxy(XOffs-3,i-2,"-");
    j=-7; for(i=0;i<6;i++) { j+=51; outtextxy(XOffs-40,j,Res[i]);}
    outtextxy(XOffs-40,YOffs-30,Lbl[0]);
    setlinestyle(SOLID_LINE,0,THICK_WIDTH);
  line(XOffs,MaxY-YOffs,MaxX-XOffs,MaxY-YOffs);
    for(i=100;i<MaxX-XOffs;i+=50) outtextxy(i-4,MaxY-YOffs-2,"|");
    j=39; for(i=0;i<10;i++) { j+=50; outtextxy(j,MaxY-YOffs+13,Term[i]);}
    outtextxy(j-20,MaxY-YOffs+27,Lbl[1]);
  strcpy(GrMessage,Lbl[2]);
    outtextxy(XOffs+100,YOffs-40, GrMessage);
  strcpy(GrMessage,Lbl[3]);
    setlinestyle(SOLID_LINE,0,NORM_WIDTH);  line(XOffs+450,YOffs,XOffs+480,YOffs);
    outtextxy(XOffs+500,YOffs-2,GrMessage);
  strcpy(GrMessage,Lbl[4]);
    setlinestyle(DOTTED_LINE,0,NORM_WIDTH); line(XOffs+450,YOffs+20,XOffs+480,YOffs+20);
    outtextxy(XOffs+500,YOffs+16,GrMessage);
  strcpy(GrMessage,Lbl[5]);
    setlinestyle(USERBIT_LINE,0xFF0F,NORM_WIDTH); line(XOffs+450,YOffs+40,XOffs+480,YOffs+40);
    outtextxy(XOffs+500,YOffs+36,GrMessage);
};
```

```
void GraphicsMode()
{ int gdriver = DETECT, gmode, errorcode;
  gettext(1,1,80,25,ScrBuff);
  ScrX=wherex(); ScrY=wherey();
  initgraph(&gdriver, &gmode, "");
  errorcode = graphresult();
  if (errorcode != grOk)
    { printf("Graphics Error...: %s\n",grapherrormsg(errorcode));
      getch(); exit(1);
    }
};

void RestoreScreen()
{ restorecrtmode();
  puttext(1,1,80,25,ScrBuff);
  gotoxy(ScrX,ScrY);
};

void DrawGraphics()
{ byte i,j,k=0;
  word x,y;
  word key=0;
  GraphicsMode();
  DrawCoords();
  for(i=0;i<3;i++)
   { k=0;
     switch (i)
      { case 0: setlinestyle(SOLID_LINE,0,NORM_WIDTH);  break;
        case 1: setlinestyle(DOTTED_LINE,0,NORM_WIDTH); break;
        case 2: setlinestyle(USERBIT_LINE,0xFF0F,NORM_WIDTH); break;
      }
     moveto(XOffs,(MaxY-YOffs+5)-(MaxY-YOffs)*Result[i][1]/11800000);
     for(j=1;j<51;j++)
     { x=k*10+XOffs;
       y=(MaxY-YOffs)-(MaxY-YOffs)*Result[i][j]/1180000;
       if (i==2 && j<4) y=(MaxY-YOffs)-(MaxY-YOffs)*Result[2][3]/1180000;
       lineto(x,y);
       k++;
     }
   }
  while (key!=ESC) key=GetKey();
  closegraph();
  RestoreScreen();
};
```

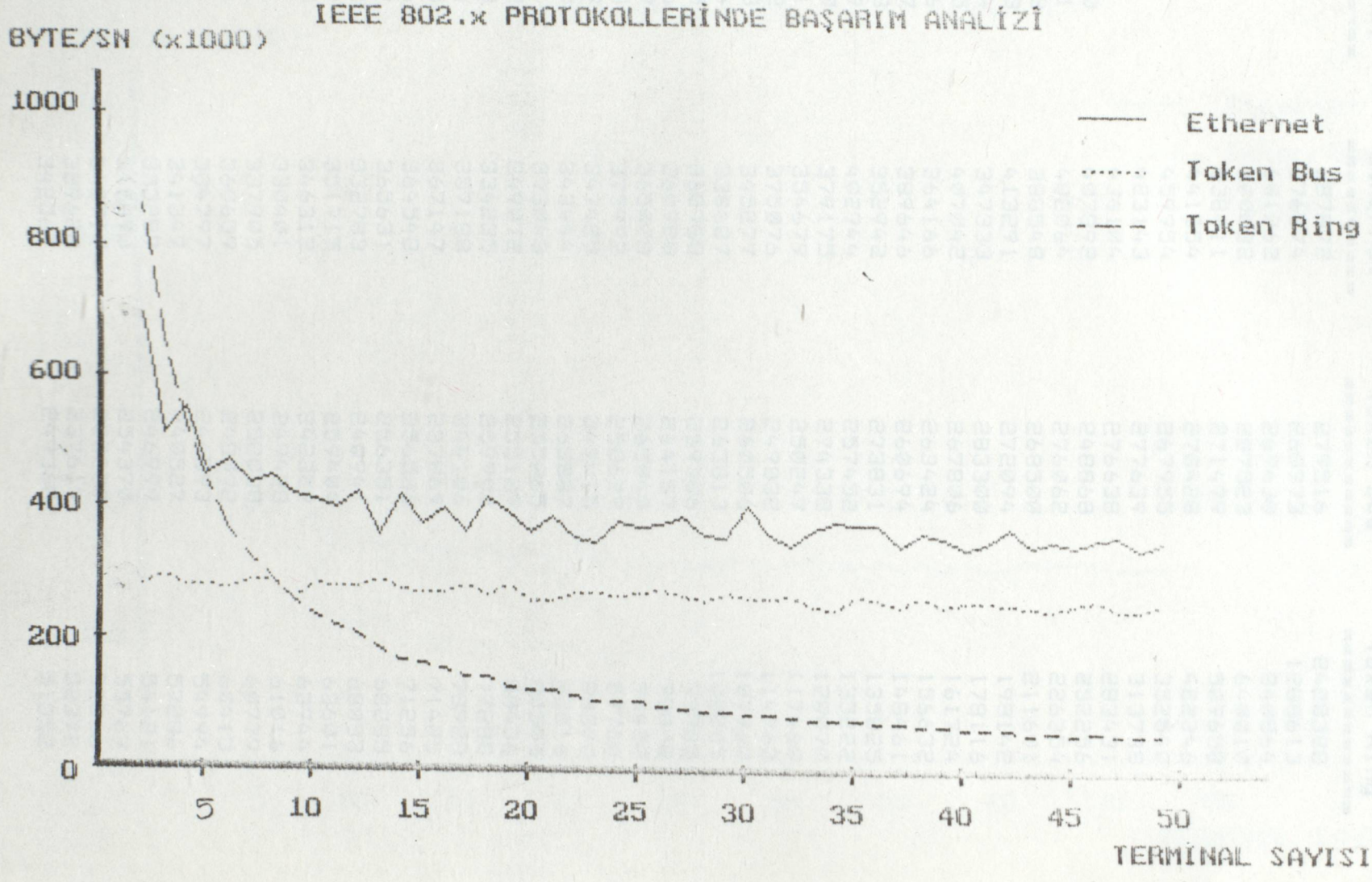
```
////////////////////////////////////////////////////////////////////////
//                                                                    //
//          TEZ.CPP 08/02/91    (C) Mehmet Kavi 881601                //
//                                                                    //
//                      MAIN PROGRAM                                  //
//                                                                    //
////////////////////////////////////////////////////////////////////////
#include <time.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include <dos.h>
#include <bios.h>
#include <graphics.h>

#include "tezglob.h"
#include "tezgraph.h"
#include "byte.h"
#include "packet.h"
#include "ethernet.h"
#include "tokenbus.h"
#include "tokenrng.h"

main()
{ byte i,j;
  word mnu=0,kkey=0;
  for(i=0;i<3;i++) for (j=0;j<51;j++) Result[i][j]=0;
  setcbrk(1);
  ctrlbrk(CtrlBreak);
  randomize();
  clrscr();
  SetupScreen();
  VersionScreen();
  textattr(HIGH); gotoxy(33,3); cprintf("Paket       Term      Test        Çıkış");
   Menu M1(" IEEE 802.x ",3,3,29,7);
   Menu M2(" Paket ",31,3,41,10);
   Menu M3(" Term ",43,3,52,14);
   Menu M4(" Test ",54,3,64,13);
   Menu M5(" Çıkış ",66,3,76,8);
l1:
  SetupScreen();
while (kkey!=ESC && kkey!=F10)
{ switch (mnu)
   { case 0: M1.DrawMenu(HIGH,DoubleFrame);
             kkey=M1.GetMenuID(PacketType,3,0);
             if (kkey==LEFT) mnu=4;
             if (kkey==RIGHT || kkey==CR) mnu=1;
             gotoxy(18,17); textattr(HIGH); cputs(packtype[M1.Item]);
             M1.DrawMenu(NORM,SingleFrame);
            break;
```

```
    case 1: M2.DrawMenu(HIGH,DoubleFrame);
            kkey=M2.GetMenuID(PackSize,6,0);
            if (kkey==LEFT) mnu=0;
            if (kkey==RIGHT || kkey==CR) mnu=2;
            if (M1.Item==0 && M2.Item>2) { M2.Item=2; mnu=1; }
            if (M1.Item==2 && M2.Item>4) { M2.Item=4; mnu=1; }
            gotoxy(18,18); textattr(HIGH); cputs(PackSize[M2.Item]);
            M2.DrawMenu(NORM,SingleFrame);
           break;
    case 2: M3.DrawMenu(HIGH,DoubleFrame);
            kkey=M3.GetMenuID(TermCnt,10,0);
            if (kkey==LEFT) mnu=1;
            if (kkey==RIGHT || kkey==CR) mnu=3;
            gotoxy(20,19); textattr(HIGH); cputs(TermCnt[M3.Item]);
            M3.DrawMenu(NORM,SingleFrame);
           break;
    case 3: M4.DrawMenu(HIGH,DoubleFrame);
            kkey=M4.GetMenuID(TestCnt,9,0);
            if (kkey==LEFT) mnu=2;
            if (kkey==RIGHT || kkey==CR) mnu=4;
            gotoxy(18,20); textattr(HIGH); cputs(TestCnt[M4.Item]);
            M4.DrawMenu(NORM,SingleFrame);
           break;
    case 4: M5.DrawMenu(HIGH,DoubleFrame);
            kkey=M5.GetMenuID(Output,4,0);
            if (kkey==LEFT)  mnu=3;
            if (kkey==RIGHT || kkey==CR) mnu=0;
            gotoxy(18,21); textattr(HIGH); cputs(Output[M5.Item]);
            M5.DrawMenu(NORM,SingleFrame);
           break;
  }
}
if (kkey==F10)
 { textattr(BLNK);
   gotoxy(2,23); cputs(" Simulasyon . . .          ");
   gotoxy(2,24); cputs("                           ");
   PacketSize=packsize[M2.Item];
   Test=testcnt[M4.Item];
   switch (M1.Item)
     { case 0: Ethernet Eth(PacketSize, Ethfield, Ethfieldoffs, Ethfieldsize);
               Eth.MakeEthernetPacket();
               textattr(HIGH);
               for (i=1;i<=termcnt[M3.Item];i++)
                 { CurrTerminal=i;
                   Result[0][i]=Eth.FrameTransmission(Test);
                   if ((i%5==0) || (i==5))
                     { if (i<=25) gotoxy(22+2*i,18);
                          else gotoxy(22+2*i-50,23);
                       cprintf("%7.0f",Result[0][i]);
                     }
                 }
               break;
```

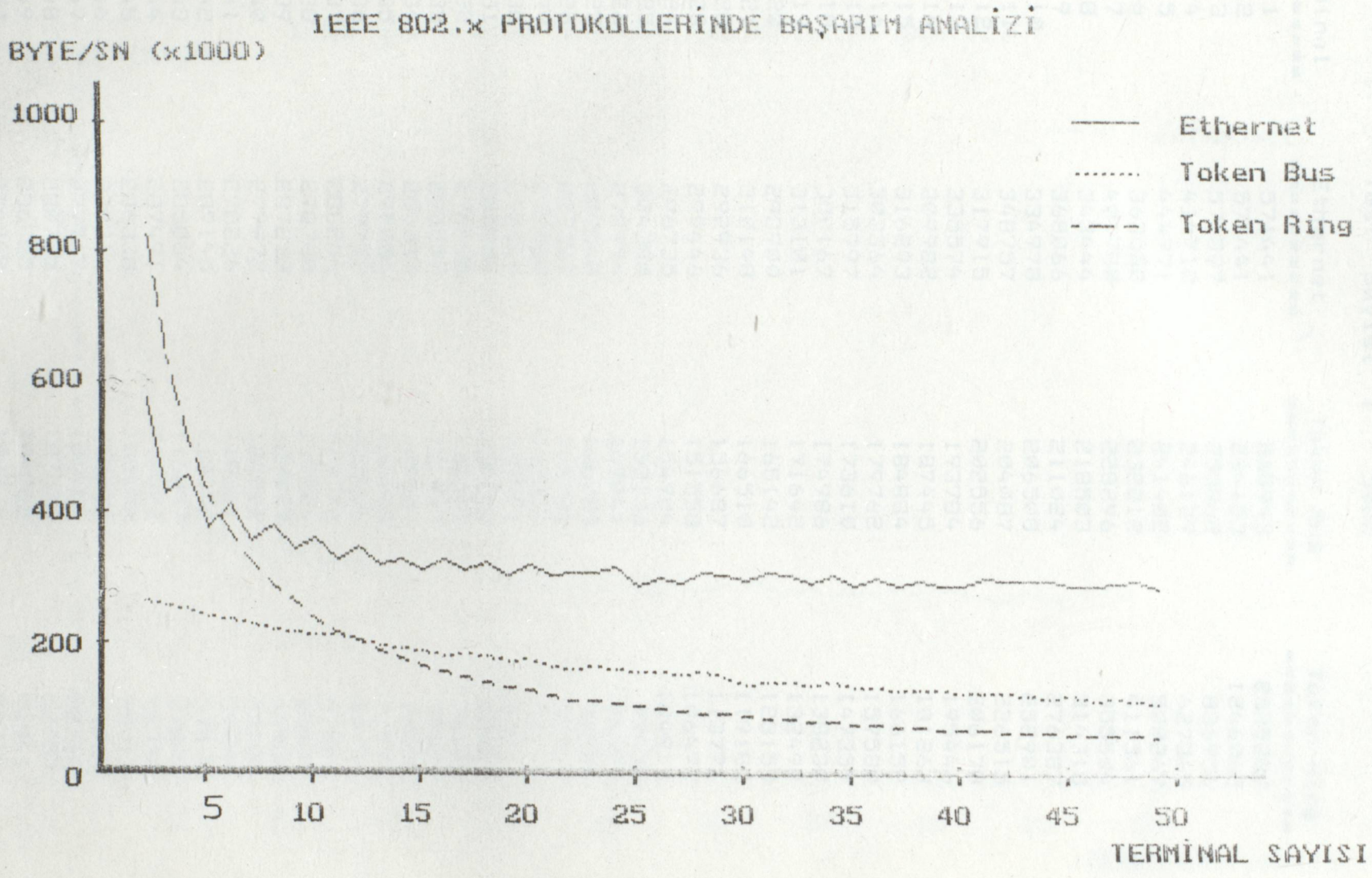
```
      case 1: TokenBus ToB(PacketSize, ToBfield, ToBfieldoffs, ToBfieldsize);
              ToB.MakeTokenBusPacket();
              textattr(HIGH);
              for (i=1;i<=termcnt[M3.Item];i++)
                { CurrTerminal=i;
                  Result[1][i]=ToB.FrameTransmission(Test);
                  if ((i%5==0) || (i==5))
                    { if (i<=25) gotoxy(22+2*i,18); else gotoxy(22+2*i-50,23);
                      cprintf("%7.0f",Result[1][i]);
                    }
                }
              break;
      case 2: TokenRing ToK(PacketSize, ToBfield, ToBfieldoffs, ToBfieldsize);
              ToK.MakeTokenRingPacket();
              textattr(HIGH);
              for (i=1;i<=termcnt[M3.Item];i++)
                { CurrTerminal=i;
                  Result[2][i]=ToK.FrameTransmission(Test);
                  if ((i%5==0) || (i==5))
                    { if (i<=25) gotoxy(22+2*i,18); else gotoxy(22+2*i-50,23);
                      cprintf("%7.0f",Result[2][i]);
                    }
                }
              break;
    }
 }
gotoxy(2,23); cputs(" Herhangi Bir Tuşa Basın... ");
gotoxy(2,24); cputs(" ESC : Iptal                ");
kkey=GetKey();
textattr(NORM);
for (i=0;i<=M3.Item;i++)
  { if (i<5) gotoxy(31+i*10,18);
      else gotoxy(31+i*10-50,23);
    cputs("        ");
  }
if (M5.Item==3)
  { if (Result[0][1]!=0 && Result[1][1]!=0 && Result[2][1]!=0)
    WriteToFile(M2.Item, M3.Item, M4.Item, M5.Item);
    DrawGraphics();
  }
  else WriteToFile(M2.Item, M3.Item, M4.Item, M5.Item);
if (kkey!=ESC) goto 11;
if (M3.Item==3) closegraph();
clrscr();
_setcursortype(_NORMALCURSOR);
};
```


IEEE 802.x PROTOKOLLERİNDE BAŞARIM ANALİZİ
BYTE/SN (x1000)
1000
800
600
400
200
0
5
10
15
20
25
30
35
40
45
50
TERMİNAL SAYISI
Ethernet
Token Bus
Token Ring

Mehmet Kavi 881601
Y.Ü. Elektronik ve Hab. Müh. Bölümü
IEEE 802.x Protokollerinin
Bilgisayar Simulasyonu ile
Performans Analizi

Paket Uzunluğu: 1512
Max. Terminal : 50
Test Sayısı : 1000

Terminal	Ethernet	Token Bus	Token Ring
1	687872	279516	2428328
2	676374	260973	1288613
3	681062	269630	840864
4	500082	287523	640010
5	538011	271499	507658
6	441554	270428	422346
7	459954	267955	352540
8	423143	277639	313738
9	436104	276638	283431
10	407892	248868	252256
11	402064	276062	226354
12	388548	268500	211601
13	413291	272094	198142
14	347333	283300	178116
15	407042	267806	161724
16	364166	263424	156632
17	389646	260694	148461
18	352942	273831	135825
19	402944	257452	135222
20	379175	274338	120070
21	354679	250247	117460
22	375076	249832	114963
23	345277	260506	107960
24	338687	263813	102205
25	368788	259386	99985
26	360780	264157	98808
27	365008	265040	94430
28	375992	258629	87760
29	347425	249415	83390
30	343444	263887	83516
31	393043	255065	80505
32	349072	255129	80436
33	334235	259902	77288
34	357188	244186	73928
35	367147	237069	71425
36	364548	256536	71256
37	365631	256351	68285
38	332783	240941	68033
39	351915	259404	63601
40	346312	242329	63944
41	330401	249428	61016
42	337905	252038	60779
43	360659	248092	60413
44	334797	244593	54994
45	341347	240527	57286
46	333289	246999	54121
47	345890	254370	55769
48	348710	238239	52866
49	329622	237619	52372
50	342367	249368	51352

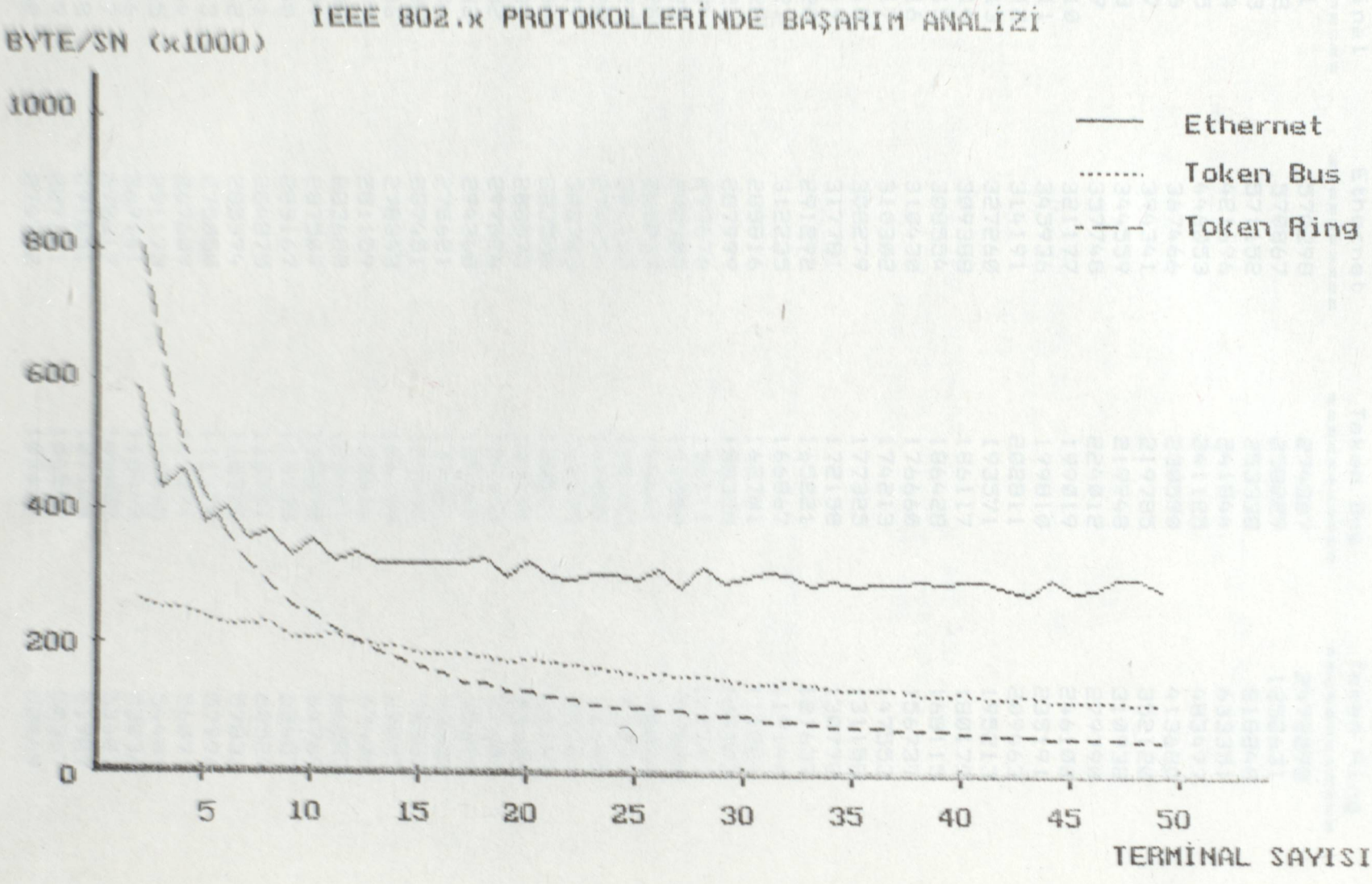

IEEE 802.x PROTOKOLLERİNDE BAŞARIM ANALİZİ
BYTE/SN (x1000)
1000
800
600
400
200
0
5
10
15
20
25
30
35
40
45
50
TERMİNAL SAYISI
Ethernet
Token Bus
Token Ring

Mehmet Kavi 881601
Y.Ü. Elektronik ve Hab. Müh. Bölümü

IEEE 802.x Protokollerinin Bilgisayar Simulasyonu ile Performans Analizi

Paket Uzunluğu: 4096
Max. Terminal : 50
Test Sayısı : 5000

Terminal	Ethernet	Token Bus	Token Ring
1	571441	268943	2525280
2	571441	264153	1246062
3	571327	255862	836956
4	414712	246139	627365
5	444771	241402	500267
6	363582	232212	411361
7	401780	228296	355504
8	343444	218503	314318
9	368066	211024	276387
10	334978	206508	252901
11	348757	204087	226515
12	317915	202556	206178
13	335574	193784	195446
14	309982	187445	181263
15	316293	184884	166158
16	303364	179762	157580
17	315997	173610	147359
18	300167	174986	138230
19	313101	171642	132498
20	293900	165142	123155
21	310148	166510	119182
22	292936	156687	113776
23	299446	151558	106472
24	296735	154904	104819
25	304380	153136	99435
26	278994	147041	94799
27	289540	146085	92280
28	283357	145738	88931
29	297560	147035	86114
30	292360	139991	83611
31	287554	130181	80889
32	296495	132222	76406
33	293505	131907	75623
34	283549	127887	72458
35	295084	131113	71774
36	279927	123396	67751
37	280244	123772	67804
38	278728	119704	65457
39	287223	120379	64938
40	277072	122202	63108
41	278226	118361	59984
42	287145	115352	59545
43	285064	116097	57805
44	287471	110602	56388
45	286338	109492	55713
46	279314	113832	54027
47	277767	106451	52918
48	280735	107830	52753
49	286725	104060	51941
50	274349	102236	50316


IEEE 802.x PROTOKOLLERİNDE BAŞARIM ANALİZİ
BYTE/SN (x1000)
1000
800
600
400
200
0
5
10
15
20
25
30
35
40
45
50
TERMİNAL SAYISI
Ethernet
Token Bus
Token Ring

Mehmet Kavi 881601
Y.ü. Elektronik ve Hab. Müh. Bölümü

IEEE 802.x Protokollerinin Bilgisayar Simulasyonu ile Performans Analizi

Paket Uzunluğu: 512
Max. Terminal : 50
Test Sayısı : 2000

Terminal	Ethernet	Token Bus	Token Ring
1	570298	274367	2472298
2	570867	258229	1252631
3	571152	253338	818248
4	421696	241844	633351
5	446053	241125	483477
6	367466	230590	413680
7	394341	219785	352120
8	344559	219948	310932
9	357748	224012	279790
10	321177	199019	246102
11	345936	199810	235191
12	314191	202811	209463
13	327260	193571	195213
14	309388	186117	180170
15	308554	186428	168115
16	310430	176660	156737
17	310302	174213	147553
18	308279	177325	131158
19	317781	172158	130772
20	291292	165221	121633
21	312235	166867	119146
22	288816	162781	111826
23	287999	158328	108703
24	293676	156111	101261
25	292703	151850	97924
26	286057	144661	98048
27	301163	148105	93017
28	274654	142809	91441
29	305783	144199	85569
30	283520	138511	83989
31	288573	133799	83102
32	297826	134153	77605
33	294368	134469	75328
34	278621	131504	73680
35	287451	124457	70882
36	278843	126189	69033
37	281159	123616	67402
38	283688	123421	66223
39	287509	118370	64769
40	283167	114733	63487
41	284875	118951	60524
42	285394	112318	57833
43	275050	111470	57996
44	267609	117371	56091
45	271173	110375	54401
46	269141	110468	53036
47	272617	108089	53407
48	291651	107148	51929
49	289121	104551	50367
50	274752	102669	50469

4.6.1 KAYNAKÇA

Kitaplar:

- ADVANCED NETWARE V2.XX-THEORY OF APPLICATIONS.
  Novell Inc. 1986, Utah, USA.

- ANALOG AND DIGITAL COMMUNICATION.
  W. David Gregg, The University of Texas at Austin.
  John Wiley & Sons, NewYork, 1977, USA.

- BASICS OF DATA COMMUNICATIONS.
  Editor-In-Chief: Harry R. Karp.
  Mc Graw-Hill Publications Co., 1976, USA.

- COMMUNICATIONS AND NETWORKING FOR THE IBM PC.
  Larry E. Jordan, Bruce Churchill.
  Prentice/Hall Inc., 1983

- COMPUTER COMMUNICATION, NETWORK DESIGN AND ANALYSIS.
  Mischa Schwartz, Columbia University, NewYork.
  Prentice/Hall Inc., New Jersey, 1977, USA.

- COMPUTER NETWORKS.
  Andrew S.Tanenbaum.
  Vrije Universiteit,Amsterdam, The Netherlands.
  Prentice/Hall International Inc., 1981, USA.

- DATA COMMUNICATIONS, COMPUTER NETWORKS, AND OSI.
  Fred Halsall, University Of Sussex.
  Addison-Wesley Publishing Co. ,1988, Great Britain.

- DATA COMMUNICATIONS, NETWORKS AND SYSTEMS.
  Editor-In-Chief : Thomas C.Bartee.
  Howard W.Sams & Co. (McMillan Inc.), 1987, USA

- MICROCOMPUTER HANDBOOK.
  Editor: J.A. McCrindle
  Collins Professional and Technical Books, 1985, Great Britain.

- LAN EVALUATION REPORT.
  Novell Inc. 1986, Utah, USA.

- PRACTICAL APPLICATIONS OF DATA COMMUNICATIONS.
  Editor-In-Chief: Harry R. Karp.
  McGraw-Hill Publications Co., 1976, USA.

Makaleler:

- AN INTRODUCTION TO NETWORK ARCHITECTURE AND PROTOCOLS.
  Paul E. Green Jr.
  IEEE Transcations on Communications, April, 1980.

- BIT ORIENTED DATA LINK CONTROL PROCEDURES.
  David E. Carlson.
  IEEE Transactions on Communications, April 1980.

- CHARACTER ORIENTED DATA LINK CONTROL PROTOCOLS.
  James W. Conard.
  IEEE Transactions on Communications, April 1980.

- COMPARATIVE PERFORMANCE OF VOICE/DATA LANs.
  Timothy A. Gonsalves, Fouad A. Tobagi.
  IEEE Journal on Selected Areas in Communications,June 1989

- COMPUTER SIMULATION.
  Richard Bronson.
  BYTE Magazine, March 1984.

- DNA: THE DIGITAL NETWORK ARCHITECTURE.
  Stuart Wecker.
  IEEE Transactions on Communications, April, 1980.

- ESTIMATION OF RELIABILITY FOR COMMUNICATION/COMPUTER NETWORKS:Simulation/Analytic Approach.
  Peter Kubat.
  IEEE Transactions on Communications, Septembert 1989.

- GETTING A HANDLE ON FDDI.
  Mary Rose Swastek, David J. Vereeke, Darrel R. Scherbarth.
  Data Communications Magazine, June 1989.

- INSIDE TOKEN RING VERSION II, ACCORDING TO BIG BLUE.
  Norm Strobe, IBM Corp., Research Triangle.
  Data Communications Magazine, January 1989.

- MODERN ÜRETİM TEKNOLOJİSİNDE İLETİŞİMİN ÖNEMİ,
  YÖRESEL AĞLAR, ETHERNET.
  Doç.Dr. Atilla Gönder, Aselsan.
  Bilgisayar Dergisi, 3.Türkiye Bilgisayar Kongresi, Nisan 1986.

- MULTIACCESS PROTOCOLS IN PACKET COMMUNICATION SYSTEMS.
  Fouad A. Tobagi.
  IEEE Transactions on Communications, April 1980.

- NETWORKING OF NETWORKS:Internetworking According to OSI.
  Fred M. Burg, Nicola Di Iorio.
  IEEE Journal on Selected Areas in Communications,
  September 1989.

- OSI REFERENCE MODEL: The ISO Model of Architecture for Open Systems Interconnection.
  Hubert Zimmermann.
  IEEE Transactions on Communications, April,1980.

- PHYSICAL LEVEL PROTOCOLS.
  H.V. Bertine.
  IEEE Transactions on Communications, April, 1980.

- QUEUE SIMULATION.
  E. Hart Rasmussen.
  BYTE Magazine, March 1984.

- SIMULATION AND GRAPHICS ON MICROCOMPUTERS.
  Ronald R Miller.
  BYTE Magazine, March 1984.

- YEREL BİLGİSAYAR AĞLARI VE UYGULAMA ALANLARI.
  Prof.Dr. Oğuz Manas. Ege Üniversitesi.
  Bilgisayar Dergisi, Aralık 1988.

ÖZGEÇMİŞ

1964 Uşak doğumluyum. Orta öğrenimimi Maçka Teknik Lisesinde tamamladım. İki yıl Marmara Ün. Teknik Eğitim Fakültesi'nde öğrenim gördükten sonra Yıldız Üniversitesi Elektronik ve Haberleşme Mühendisliği bölümüne girdim. 1988 yılında mezun olduktan sonra aynı yıl Haberleşme Ana Bilim dalında yüksek lisans eğitimine başladım. Halen Bilmar A.Ş.'de görev yapmaktayım.