TORUS AĞI BENZETİMİNİN ÇOK İŞLEMCİLİ MİMARİLER İÇİN GERÇEKLEŞTİRİMİ

IMPLEMENTATION OF TORUS NETWORK SIMULATION FOR MULTIPROCESSOR ARCHITECTURES

Hacettepe Üniversitesi

Lisansüstü Eğitim – Öğretim ve Sınav Yönetmeliğinin

BİLGİSAYAR Mühendisliği Anabilim Dalı İçin Öngördüğü

YÜKSEK LİSANS TEZİ

olarak hazırlanmıştır.

2011

Fen Bilimleri Enstitüsü Müdürlüğü'ne,

Bu çalışma jürimiz tarafından **BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI'nda YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Başkan	: .....
	(Doç.Dr. Murat Caner Testik)
Üye (Danışman)	: .....
	(Yrd.Doç.Dr. Kayhan M. İmre)
Üye	: .....
	(Yrd.Doç.Dr Harun Artuner)
Üye	: .....
	(Yrd.Doç.Dr. Mustafa Ege)
Üye	: .....
	(Dr. Ahmet Burak Can)

ONAY

Bu tez Hacettepe Üniversitesi Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliği'nin ilgili maddeleri uyarınca yukarıdaki jüri üyeleri tarafından ...../...../..... tarihinde uygun görülmüş ve Enstitü Yönetim Kurulunca ...../...../..... tarihinde kabul edilmiştir.

...../...../.....

Prof.Dr. Adil Denizli

FEN BİLİMLERİ ENSTİTÜSÜ MÜDÜRÜ

TORUS AĞI BENZETİMİNİN ÇOK İŞLEMCİLİ MİMARİLER İÇİN GERÇEKLEŞTİRİMİ

ÖZ

İşlemci hızlarının doğal sınırlara ulaşması ve sektörlerde her gün artan işlem kapasitesi ihtiyaçları, işlemci mimarilerinde koşut sistemleri zorunlu kılmıştır. Günümüzde bilgisayar sistemlerinin bütünü büyük ve küçük ölçekli çok işlemcili sistemlere dönüşmüşlerdir. Çok işlemcili mimarilerin yaygınlaşması, bu mimarilerin ve mimariler üzerinde yapılan çalışma ve tasarımların önemini artırmıştır.

Bilgisayar sistemleri gibi fiziksel testlerin çok maliyetli ve uzun sürelere mal olduğu alanlar, sistemleri modelleyen benzetimlerin gereksinimini artırır. Benzetimler bilimsel ve ticari alanlarının bütününde yoğun biçimde kullanılırlar.

Tez kapsamında, çok işlemcili sistemlerde kullanılan bir ağın benzetimi yapılmıştır. Benzetimi yapılan ağ *torus* topolojisine sahiptir ve w*ormhole* anahtarlama modelini kullanmaktadır. Benzetimi yapılan ağ *wormhole* anahtarlama üzerinde yapılmış durağan ve kaynak kararlı yönlendirme algoritmalarını desteklemektedir. Benzetim koşut benzetim olarak kesikli olay benzetimi kapsamında gerçekleştirilmiştir. Benzetim java programlama diliyle geliştirilmiştir. Geliştirilen benzetimin tasarım detayları anlatılmış ve üzerinde yapılan testlerin sonuçları tartışılmıştır.

Anahtar Kelimeler: Çok İşlemcili Ağlar, Koşut ve Dağıtılmış Benzetimler,Kesikli Olay Benzetimi, *Wormhole* Anahtarlama, *Torus* Topolojisi,

Danışman: Yrd.Doç.Dr. KAYHAN M. İMRE, Hacettepe Üniversitesi Bilgisayar Mühendisliği Bölümü

IMPLEMENTATION OF TORUS NETWORK SIMULATION FOR MULTIPROCESSOR ARCHITECTURES

ABSTRACT

Processor speeds reaching the natural limits and the increasing processing needs have made the parallel systems be necessary in the processor architectures. At the present time, all of the computer systems have transformed into multi-processor systems of large or small scales. The spread of architecture with multiple processors has increased the importance of both these architectures and the studies on them.

Fields like computer systems, in which physical tests are costly and long-term increase the need for simulations which model systems. Simulations are massively used in all scientific and commercial areas.

In the scope of thesis, the simulation of a multi-computer network has been implemented. The simulated network has *torus* topology and uses *wormhole* switching model. The simulated network supports deterministic and source-based routing algorithms. Simulation has been implemented as a parallel simulation and in the scope of discrete event simulation. The simulation has been developed with Java programming language. The details of developed simulation design have been explained and the results of tests have been discussed.

Keywords: Multiprocessor Networks, Parallel and Distributed Simulations, Discrete Event Simulation, *Wormhole* Switching, *Torus* Topology

Advisor: Yrd.Doç.Dr. KAYHAN M. İMRE, Hacettepe Üniversitesi Bilgisayar Mühendisliği Bölümü

TEŞEKKÜR

Tez konusunun belirlenmesini sağlayan, tez çalışmasının hazırlanmasında ve tez metninin yazılmasında yardımcı olan Sayın Yrd.Doç.Dr. Kayhan M. İmre' ye;

Tez metnini inceleyerek biçim ve içerik bakımından son halini almasına yardım eden değerli hocalarıma;

Tez çalışması boyunca desteklerini ve değerli fikirlerini benden esirgemeyen Aydın Kaya, Yiğitcan Aksarı ve Ali Seydi Keçeli'ye;

Hayatımdaki her dönemde olduğu gibi, yüksek lisans dönemim boyunca değerli desteği ve sabrıyla hep yanımda olan -müstakbel- eşim Özge Koç'a;

Tez sürecimde akademik tecrübelerini benimle paylaşan ve çalışmalarımı destekleyen dayım Abdullah Güllü'ye;

Hayatım boyunca her konuda yanımda olan fedakar aileme içtenlikle teşekkür ederim.

İÇİNDEKİLER DİZİNİ

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

SÖZLÜK

İnterconnection Network	Bileşen Bağlantılı Ağlar
Direct Networks	Dolaysız Ağlar
Indirect Networks	Dolaylı Ağlar
Hybrid Networks	Melez Ağlar
Switching	Anahtarlama
Routing	Yönlendirme
Deterministic Routing	Durağan Yönlendirme
Adaptive Routing	Uyarlamalı Yönlendirme
Flow Control	Akış kontrolü
Deadlock	Kilitlenme
Discrete Event Simulation	Kesikli Olay Benzetimi
Message Passing Programming	İleti Göndermeli Programlama
Interface	Arayüz

1. GİRİŞ

İşlemci teknolojilerinde saat sıklığının fiziksel sınırlara yaklaşması, küçük ve büyük ölçekli bilgisayar mimarilerinin bütününde koşut sistemleri zorunlu kılmış ve koşut sistemlerinin kullanımlarını yaygınlaştırmıştır. Çok işlemcili koşut sistemlerin yaygınlaşmasıyla işlemci üretici firmalar düşük maliyetli ve yüksek başarımlı koşut sistemlere ve yonga düzeyinde çok işlemcili ağ sistemlerinin tasarımlarına yönelmişlerdir. Düşük maliyetli ve yüksek başarımlı koşut sistemlere olan ihtiyaç, bu konularda yapılacak akademik ve endüstriyel çalışmalara olan ihtiyacı artırmıştır.

Fiziksel bir sistemin gerçekleştiriminin ekonomik açıdan ve zaman açısından maliyetli olduğu veya sistemin gözlenmesinin olanaksız olduğu durumlarda, fiziksel sistemin benzetiminin yapılması anlamlıdır. Benzetim, bir sistemin, nesnenin veya olayın davranışlarının zamana bağlı olarak taklit edilmesidir. Gereksinimleri yüksek olan fiziksel sistemler koşut benzetimlerle gerçekleştirilirler. Koşut benzetimlerde, modellenen fiziksel sistem alt modellere ayrılır ve bu modeller çok işlemcili sistemdeki işlemcilere dağıtılırlar. Kullanıcı veya sistem kaynaklarının coğrafi olarak dağıtılmış olduğu koşut sistemler, koşut ve dağıtılmış olarak gerçekleştirilirler.

Tez kapsamında yapılan çalışmada, çok işlemcili bir ağın benzetimi gerçekleştirilmiştir. Gerçekleştirilen ağ benzetimindeki bileşenler birbirlerine *torus* topolojisiyle bağlanmıştır ve dolaysız ağ özelliklerini içermektedir. Ağ benzetimi koşut benzetim biçiminde, java programlama dili kullanılarak yapılmıştır. Gerçekleştirimde java programlama dilinin koşut programlama kütüphaneleri ve java işletim dizileri yoğun biçimde kullanılmıştır.

Tez metninin ilk kesiminde koşut programlama ve dolaysız ağların temeli olan bileşen bağlantılı ağlar hakkında bilgilendirme verilecektir. Sonraki kesimde dolaysız ağlar ve özellikleri anlatılacak, dolaysız ağları oluştururken ve çalıştırırken kullanılan yöntemler detaylandırılacaktır. Kesim 4'te benzetimin temelleri ele alınacaktır. Bu kesimde benzetimlerin sınıflandırılması, benzetimlerde zaman yönetimi ve koşut ve dağıtılmış konuları anlatılacaktır. Kesim 5'te ağ benzetimi gerçekleştirimi anlatılacaktır. Bu kesimde önce benzetim altyapısı gerçekleştirimi daha sonra bu benzetim altyapısı üzerine kurulan çok işlemcili ağ benzetiminin gerçekleştirimleri ve çözüm yaklaşımları anlatılmıştır. Kesim 6'da gerçekleştirilen ağ benzetimi üzerinde

yapılan durum çalışmaları ve sonuçları incelenmiştir. Son kesimde ise yapılan çalışmanın sonuçları tartışılmış ve devamında yapılabilecek çalışmalar ortaya konulmuştur.

1.1. Benzer Çalışmalar

Günümüzde en çok kullanılan benzetim altyapısı HLA benzetim altyapısıdır (DMSO, 2000, Perumalla, 2006). HLA altyapısı üst düzey zaman ve görev yönetimi sağlayan genel amaçlı bir benzetim sistemidir.

Tez kapsamında yapılan koşut ağ benzetimi çalışmasına benzer biçimde genel geçer bir benzetim altyapısı örnekleri bulunmaktadır. Çok işlemcili benzetim ağları üzerinde yapılan çalışmalarda, probleme yönelik benzetim gerçekleştirimleri kullanılmış ve sonuçları paylaşılmıştır. Bu çalışmalarda benzetim gerçekleştirim detayları verilmemiştir.

MultiSim'in çalışma ilkesi, tez kapsamında ele alınan çalışmaya benzer ancak detaylı bir çalışmadır (McKinley, 1993). MultiSim, *wormhole* anahtarlama modelini desteklemekte, durağan ve uyarlamalı yönlendirme yaklaşımlarını desteklemektedir. MultiSim benzetimi; *hypercube*, veri yolu tabanlı hypercube, ağaç gibi topolojileri desteklemektir. MultiSim benzetim sistemi *torus* topolojisini desteklememektedir. MultiSim benzetim sistemi, CSIM benzetim tanımlama diliyle gerçekleştirilmiştir (Schwetman,1986, CSIM, 2011).

FlexSim, *torus* topolojisine sahip *wormhole* anahtarlama ağı üzerinde tam uyarlamalı yönlendirme modeli öneren, benzetim sonuçlarını paylaşmıştır (Pinkston, 1999). Flexsim çalışmasında 16x16 *torus* kullanılmıştır.

Swaminathan Rawany yaptığı doktora çalışmasında, *wormhole* anahtarlama üzerinde uyarlamalı yönlendirme modellerinin benzetimini gerçekleştirmiştir (Rawany, 1995).

2. KOŞUT PROGRAMLAMA VE BİLEŞEN BAĞLANTILI AĞLAR

Moore Yasası, donanım karmaşıklığının belli bir kuralla arttırılabileceğini, sistem başarımının her 18 ayda iki katına çıkacağını belirtmektedir. Günümüz işlemci teknolojilerinde yakın zamana kadar bu yasayla paralellik göstermiştir ve 1980'lerden beri 2 ila 3 yıl arasında, tek bir çekirdek yongası içine yerleştirilebilen transistor sığası yaklaşık iki katına çıkmıştır (Intel, 2005). Ancak bu yasada belirtilen artışın günümüzde sınırlarına ulaşılmıştır. Son yıllarda transistor boyutları ve işlemci karmaşıklıkları önceki yıllardaki artışını gösterememektedir. Bununla beraber hâlihazırdaki işlemci gereksinimlerinin artışıyla, mimariler aşırı derecede karmaşıklaşmaktadır. İşlemci üreticileri istenilen gereksinimleri sağlayacak hızda işlemciler üretememektedir.

Özellikle işlemci teknolojilerinde fiziksel sınırlara erişilmesine rağmen; bilim, finans, askeri alanlarda sürekli yeni problemler ortaya çıkmakta ve dolayısıyla ihtiyaç duyulan işlem gücü artmaktadır. Günümüzde bu sektörlerdeki problemleri çözmek için ihtiyaç duyulan işlem kapasitesi *teraflops* (Saniyede 10^{12} kayan noktalı işlem) seviyelerine ulaşmış, basit problemler bile *gigaflops* (Saniyede 10^{9} kayan noktalı işlem) seviyesinde işlem kapasitelerine ihtiyaç duyar hale gelmektedir.

Yükselen işlem kapasitesi ihtiyaçlarına karşılık veremeyen donanım üreticileri için, koşut sistemleri kullanmak zorunlu hale gelmiştir. Bu alternatif mimari yaklaşımında, sistem içindeki birden fazla işlemci, bir problemi çözmek için birlikte çalışarak sistemin toplam işlem kapasitesini yükseltirler. Bununla birlikte, bellekler sistem üzerindeki işlemcilere dağıtılarak, bellek erişim sığası sistemdeki işlemci sayısıyla ölçekli biçimde artırılır. Bu sistemlerde, ihtiyaç doğrultusunda, bellek ve işlemcilere benzer biçimde, çevresel aygıtlarda da birden fazla bileşen eşgüdümlü biçimde kullanılarak koşut çalışması sağlanmaktadır.

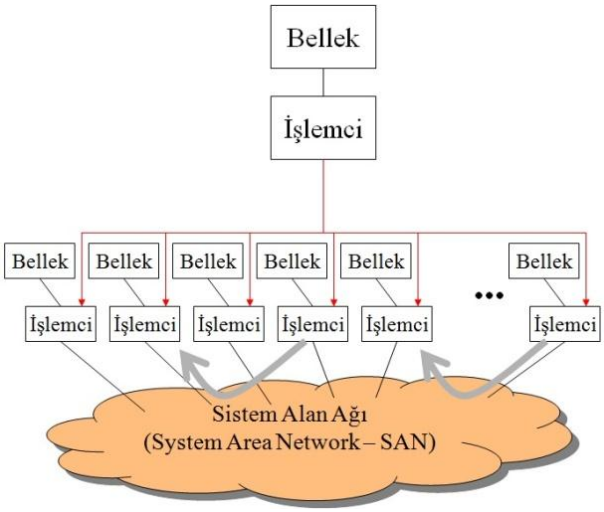
Koşut bilgisayarların programlanması, seri programlama yöntemlerine göre daha karmaşık ve hataya açıktır. Üzerinde program koşturulan işlemcilerin iletişimleri ve işlemcilerin bellekler üzerinde eriştiği verilerin bütünlüğü ve tutarlılığı sağlanmalıdır. Bu problemlere getirilen programlama yaklaşımları *koşut programlama* yöntemlerini oluşturur. Koşut programlama yöntemleri, ileti göndermeli programlama ve paylaşımlı bellek üzerinde programlama olarak iki sınıfa ayrılır. İleti göndermeli

programlamada, sistem üzerinde koşut çalışan işlemciler arasındaki veri alışverişi, işlemcilerin bağlı olduğu iletişim ortamı üzerinde birbirlerine ileti göndermeleriyle sağlanır (Athas, 1988). Paylaşımlı bellek için geliştirilen programlama modelinde ise, sistem üzerindeki işlemciler aynı adres alanını paylaşırlar ve veri alışverişlerini aynı bellek üzerinde gerçekleştirirler. İleti göndermeli programlama, paylaşımlı bellek üzerinde programlama özelliklerine göre daha zordur ve üzerinde program koşturulan koşut sistem hakkında daha alt seviyede sistem bilgisine ihtiyaç duyar. Bununla birlikte paylaşımlı bellek üzerinde programlamada veri tutarlılığı ve bütünlüğü problemi daha önemlidir ve donanımsal ve yazılımsal yöntemlerle sağlanır.

2.1. Koşut Bilgisayar Mimarileri

Bilgisayar mimarileri için en bilinen sınıflandırma, 1966 yılında Michael J.Flynn tarafından yapılmıştır (Flynn, 1972). Flynn bilgisayar mimarilerini dört temel sınıfta toplar:

- *SISD(Single Instruction, Single Data):* Bilinen geleneksel işlemcileri kapsar. Bu sınıfa dahil olan mimarilerde, tek bir işlemci tek bir bellek alanındaki verileri tekil komut akışlarıyla işler.



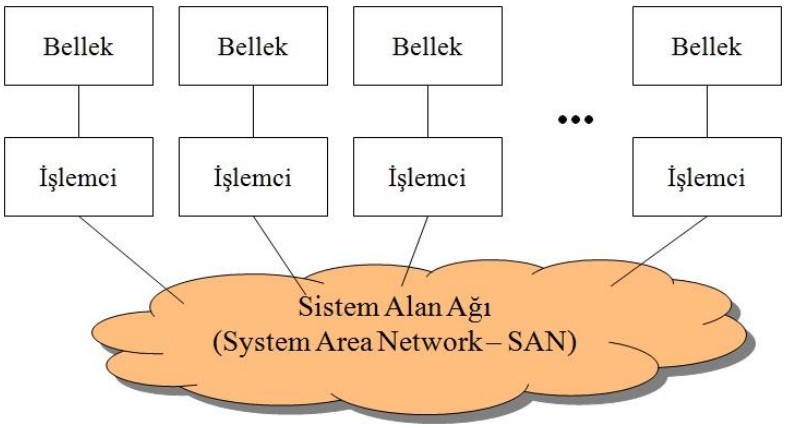
Şekil 2-1. SIMD modeli

- *SIMD(Single Instruction, Multiple Data):*Aynı işlem bir grup veri üzerinde koşut olarak gerçekleştirildiği mimarilerdir. Bu mimarilerde, denetleyici işlemci olarak da tanımlanan bir komut işlemcisi ve komut istemcisinin gönderdiği komutu işleyen çok sayıda işlemci bulunur. Denetleyici işlemci komutların akış

kontrolünden ve diğer işlemcilere iletilmesinden sorumludur. Diğer işlemciler seçilen komutu kendi veri grupları üzerinde işletirler. SIMD modeli Şekil 2-1'de gösterilmiştir. Anlaşılacağı üzere SIMD mimarisindeki işlemciler komut düzeyinde zaman uyumlu olarak çalışırlar. SIMD modelleri ilk koşut mimarilerin tasarımında kullanılmıştır. Bununla birlikte vektör bilgisayarlar ve günümüz modern grafik işleyiciler SIMD modelini de temel alırlar.

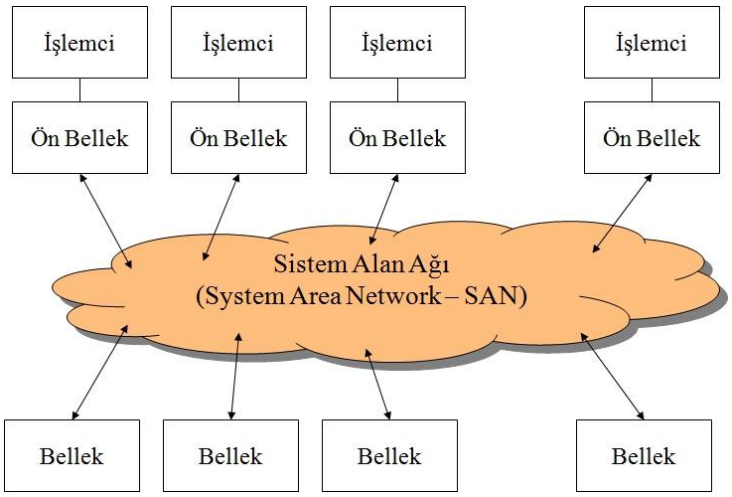
- *MISD(Multiple Instruction Single Data):* Birden fazla birimin aynı veri üzerinde çalıştığı mimari sınıfıdır. Bu mimarileri kullanarak ticari olarak gerçekleştirilmiş ve başarılı olan koşut sistemler bulunmaz. Bununla birlikte MISD mimari sınıfı ardıl düzen işleme (*pipelining*) kullanan bilgisayarları içerir.
- *MIMD(Multiple Instruction Multiple Data):* MIMD mimarisindeki bilgisayarların belli bir sayıda işlemcisi olur. Bu işlemciler, SIMD mimarisinden farklı olarak, komut düzeyinde zaman uyumsuz; bununla birlikte daha üst düzeyde zaman uyumlu veya zaman uyumsuz çalışabilirler. MIMD mimarisi içindeki işlemcilerin kendilerine ait komut yönetim birimleri bulunur ve işlemciler kendilerine atanmış veri grupları üzerinde komutlarını işletirler. Büyük ölçekli koşut sistemlerin büyük bir çoğunluğu ve bu tez kapsamında incelenen sistemler, MIMD mimari sınıfındadırlar.

MIMD mimarisi, bellek alanı gerçekleştirimlerine göre *Dağıtılmış ve Paylaşımlı Bellekli MIMD* sınıflarına ayrılır. Dağıtılmış bellekli MIMD koşut sistemlerde, her işlemci kendi yerel belleğine sahiptir ve işlemciler arası iletişim sistem alan ağı üzerinden ileti göndererek sağlanır. Bu sistemler *ileti göndermeli koşut sistemler* olarak da adlandırılırlar (Duato, 2002). Dağıtılmış bellekli MIMD yapısı Şekil 2-2'de gösterilmiştir.



Şekil 2-2. Dağıtılmış bellekli MIMD yapısı

Paylaşımlı Bellekli MIMD koşut sistemlerde bellek bileşenleri sistem alan ağı üzerinden erişilebilir durumdadır ve bütün işlemciler aynı adres alanını paylaşırlar. Bu sistemlerde işlemcilerin veri alışverişi için birbirlerine ileti göndermesi gerekmez. Veri alış verişi işlemleri bellek elemanları üzerinden gerçekleştirilir. Bu sistemlerde veri işlemlerinde bellek gecikmeleri sistem alan ağı beklemelerinden kaynaklanır. Dolayısıyla, sistem alan ağının işlemcilere göreli yavaş olduğu sistemlerde, bellek gecikmeleri başarımı düşürecektir. Bununla birlikte, birden fazla işleyicinin aynı bellek bileşenine istek göndermesi durumunda ağ darboğazları oluşacaktır. Bu soruna çözüm olarak paylaşımlı MIMD sistemlerde işlemcilerin ön bellek kullanması getirilmiştir. Bu sistemlerde, işlemcilerin sık kullandığı veriler ön bellekte saklanır ve ön bellekler işlemcilerin yerel belleği işlevini görür. Ön bellek kullanılan sistemlerde özellikle ön bellek, genel bellek ve diğer ön bellekler arasında veri tutarlılığı ve bütünlüğünün sağlanması gerekir. Şekil 2-3'te paylaşımlı MIMD yapısı gösterilmiştir.



Şekil 2-3. Paylaşımlı Bellekli MIMD

2.2. Bileşen Bağlantılı Ağlar ve Sınıflandırılması

Sayısal bir sistem üç temel bileşenden oluşur: *mantık, bellek* ve *iletişim* (Dally, 2004). Mantık verinin işlenerek dönüşümünü sağlar. Örnek olarak işlemcinin iki sayıyı toplaması verilebilir. Bellek işlenen ve işlenecek verinin saklanması ve zamanı geldiğinde tekrar çağrılmasından sorumludur. İletişim ise verinin birimler arasında taşınmasını belirtir.

Bir sistemin toplam hızını, o sistemdeki en yavaş birimin hızı belirler. Bu durum darboğaz olarak tanımlanır. Günümüzde işlemci ve bellek sığaları artmalarına rağmen, veri iletişiminde ışık hala en hızlı yol olarak görülmektedir ve ışığın hızı sabittir. Bu durum iletişim sığasının artırılması ve işlemci – bellek sığalarına yaklaşması için farklı veri iletişimi yaklaşımlarını zorunlu kılmıştır.

İletişim darboğazlarına çözüm olarak *bileşen bağlantılı ağlar (interconnection networks)* önerilmiştir. Bileşen bağlantılı ağlar, en basit yaklaşımla uçbirimler arasında veri iletişimi sağlayan programlanabilir sistemler olarak tanımlanırlar (Dally, 2004). Bileşen bağlantılı ağlar, günümüzde iletişim protokollerinin çoğunu kapsamaktadır ve Bu sistemler yonga düzeyinde işlemci ile bellek arasında iletişimi sağlayan altyapıyı oluştururken, yerel ve geniş alan ağlarında ayrık sistemlerin bütünleşmesini sağlarlar.

Bileşen bağlantılı ağların ilk kullanım yıllarında her düzeydeki veri iletişimi altyapısı veri yolları (*bus*) ile sağlanmıştır. Fakat günümüzde etkinliğin önemli olduğu sistemlerin bütününde noktadan noktaya bağlantı destekleyen bileşen bağlantılı ağlar tercih edilmektedir.

Koşut bilgisayar sistemlerinde, etkinliğin temel parametre olması, bu sistemlerde kullanılacak iletişim altyapısının hız ve başarımının önemini artırmıştır. Koşut sistem kullanımının ilk dönemlerinde iletişim alt sistemi için en basit çözüm yaklaşımı olarak, işlemcileri, bellekleri ve diğer çevresel aygıtları, sistemde tek bir işlemci varmış gibi, sistem ve giriş/çıkış veri yollarına bağlamak önerilmiştir. Sistemdeki dağıtılmış işlemciler de, yerel alan ağı (*Local Area Network, LAN*) üzerinden birbirleriyle iletişim kurmuşlardır. Fakat önerilen bu iletişim altyapısında bellek ve işlemci arasındaki ile işlemciler arasındaki veri iletişim hızı yerel alan ağı hızına eşit olacaktır. Dolayısıyla bu sistemler üzerinde gerçekleştirilen çoğu uygulamada iletişim darboğazı meydana gelmiştir.

Koşut bilgisayar mimarilerindeki iletişim problemlerine çözüm olarak, bu sistemlerde iletişim alt sistemi olarak bileşen bağlantılı ağların kullanılması önerilmiştir. Bileşen bağlantılı ağ teknolojisi ilk olarak dağıtılmış bellekli çoklu bilgisayarlara uygulanarak bu sistemlerin ölçeklenebilirliğinin artırılması amaçlanmıştır. Dağıtılmış bellekli çoklu bilgisayarların, diğer bileşen bağlantılı ağ uygulama alanlarından daha fazla etkinliğe ihtiyaç duyması, bileşen bağlantılı ağların çoklu bilgisayarlar için geliştirilmesi çalışmalarını artırmıştır. 90'lı yılların sonunda, bileşen bağlantılı ağlar, Yerel Alan Ağlarında (LAN) uygulanmaya başlanmış ve “*backplane*” veri yollarının yerini almaya başlamıştır (Duato, 2002).

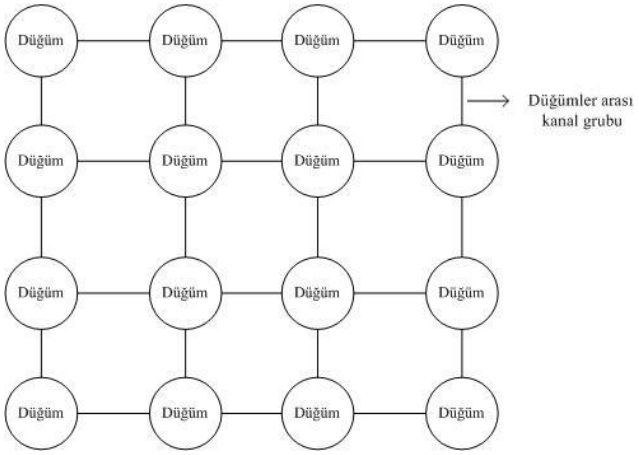
Bileşen bağlantılı ağların, iletişim protokollerinde yaygın kullanımı birçok bileşen bağlantılı ağ modelinin ortaya çıkmasına neden olmuştur. Önerilen bu ağ modelleri belirli kıstaslara göre sınıflandırılmaktadır. Bileşen bağlantılı ağlar, en temel biçimde çalışma moduna göre zaman uyumlu (*synchronous*) ya da zaman uyumsuz olarak ve ağ kontrolüne göre merkezi (*centralized*) veya dağıtılmış (*distributed*) olarak sınıflandırılır. Günümüzde, çoklu işleyiciler, çoklu bilgisayarlar ve iş istasyonları koşut sistem piyasasına egemen durumda olup ve bu sistemler genel olarak dağıtılmış zaman uyumsuz biçimde çalışmaktadırlar.

Çizelge 2-1'de, en çok bilinen ve kullanılan bileşen bağlantılı ağlar dört temel sınıfta toplanmıştır. Tez kapsamında bileşen bağlantılı ağ sınıflarından dolaysız ağların benzetimi gerçekleştirilmiştir. Dolaysız ağların özellikleri Bölüm 3'te anlatılacaktır.

- Bileşen Bağlantılı Ağlar (*Interconnection Networks*)
  - Paylaşımlı ortam ağları (*Shared Medium Networks*)
    - Yerel Alan Ağları (*Local Area Network*)
      - Contention Bus
      - Token Bus
      - Token Ring
    - Backplane Bus
  - Dolaysız Ağlar (*Direct Networks*)
    - Katı Dikey Topolojiler (*Strictly Orthogonal*)
      - Hasır Doku (*Mesh*)
      - *Torus* (*k -ary n-cube*)
      - *Hypercube*
    - Diğer Topolojiler
  - Dolaylı Ağlar (*Indirect Networks*)
    - Düzenli Topolojiler (*Regular Topologies*)
      - Çapraz Bağlantılı Ağlar
      - Çok düzeyli Bileşen Bağlantılı Ağlar (*Multistage Interconnection Network, MIN*)
        - *Blocking Networks*
          - Tek yönlü MIN
          - Çift yönlü MIN
        - *Nonblocking Networks*
    - Düzensiz Topolojiler (*Irregular Topologies*)
  - Melez Ağlar (*Hybrid Networks*)
    - *Multiple-Backplane Buses*
    - Hiyerarşik Ağlar

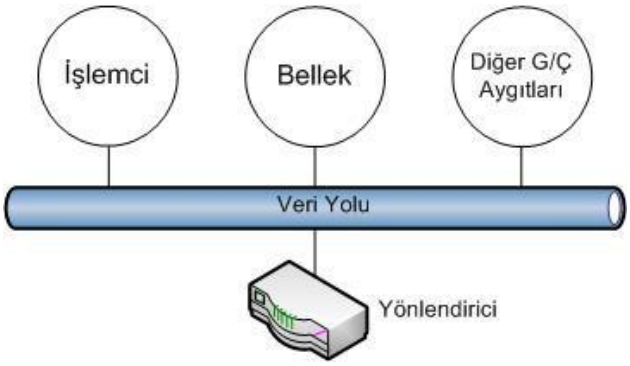
Çizelge 2-1. Bileşen Bağlantılı Ağların Sınıflandırılması (Duato, 2002)

3. DOLAYSIZ AĞLAR VE ÖZELLİKLERİ



Şekil 3-1. 4x4 hasır doku topolojisindeki dolaysız ağ modeli

Dolaysız Ağlar veya *Noktadan Noktaya Ağlar,* yüksek sayıda işlemciyi barındıran ağ modelleridir. Dolaysız ağ tabanlı koşut sistemler, düğüm olarak tanımlanan alt bileşenlerinin birbirlerine bağlanmasından oluşur. Dolaysız ağ mimarisinde, her düğümün, etrafındaki bir grup düğüme noktadan noktaya, ya da doğrudan, bağlantısı bulunur. Düğümün etrafındaki bu düğümlere *komşu düğümler* denir. Dolaysız ağlar koşut sistemlerde yoğun şekilde tercih edilir. Çünkü düğüm sayısı arttıkça sistemin iletişim ve bellek erişim sığası ile işlem kapasitesi de paralelinde artacaktır (Ni, 1993). Örnek 4x4 hasır doku dolaysız ağ modeli Şekil 3-1'de gösterilmiştir.



Şekil 3-2. Genel düğüm yapısı

Şekil 3-2'de genel bir dolaysız düğümü yapısı gösterilmiştir. Dolaysız ağlardaki düğümlerin her biri kendi işlemcisi, yerel belleği ve çevresel aygıtları bulunan programlanabilir bir bilgisayardır. Bir dolaysız ağdaki düğümler sistem içinde veya

farklı sistemler içindeki düğümler karşılaştırıldığında farklı işlevlere sahip olabilirler. Örneğin sistem içinde bir grup işleyici vektör işleyici, grafik işleyici veya giriş/çıkış işleyiciler olabilir.

Dolaysız Ağ mimarisinde gerçekleştirilen koşut bilgisayarlar, ileti göndermeli koşut programlama modeliyle programlanırlar. Bu sistemlerde bulunan düğümler, birbirleriyle veri alış verişi yapmak için bağlı oldukları sistem alan ağına ileti gönderirler. Dolaysız ağlarda komşu düğümlerin aralarında doğrudan bağlantıları oldukları için, dolaysız olarak birbirlerine ileti gönderebilirler. Birbirlerine komşulukları olmayan düğümlerin iletilerinin, kaynak düğümden hedef düğüme iletilmesi için, bu düğümler arasındaki bir dizi düğümden geçmesi gerekir (Ni, 1993).

Dolaysız ağlarda düğümler arasındaki iletilerin yönlendirme işlevi *yönlendiriciler* tarafından gerçekleştirilir. Bu sebeple bu ağlar *Yönlendirici Tabanlı Ağlar* olarak da isimlendirilirler (Ni, 1993) .

Bir dolaysız ağ içindeki iki düğümün komşuluğu, bu iki düğümün yönlendiricileri arasında çift yönlü bir iletişim bağlantısının kurulmasıyla gerçekleştirilir. Bu iletişim protokolü kapsamında komşuluklar, yönlendiricilerinin her ikisinin giriş kanalları, komşu düğüme ait yönlendiricinin çıkış kanallarına karşılıklı olarak bağlanılarak gerçekleştirilir.

Dolaysız ağlardaki düğümler arasındaki iletişim sağlayan bağlantılar *kanal* olarak isimlendirilirler. Her yönlendirici değişken sayıda giriş ve çıkış kanalını destekler. Yönlendiriciler üzerindeki *yerel kanallar,* yönlendiricinin kendi düğümü içindeki yerel bellek ve işlemcisi ile iletişimini sağlar. Düğümlerin birbirleriyle iletişimini sağlayan yönlendiriciler arasındaki kanallar *dış kanallar* olarak isimlendirilirler. Aksi belirtilmediği sürece, bir dolaysız ağ kapsamında kanal terimi, varsayılan olarak bir harici kanalı betimler (Ni, 1993). Metnin ilerleyen kısımlarında bu yaklaşım sürdürülecektir.

Dolaysız ağlar, G(N,C) biçiminde bir çizgeyle tanımlanabilirler. Tanımlanan bu çizgede köşeler (*vertices*) işleyici düğümler kümesini, kenarlar ise bu düğümler arasındaki iletişim kanalları kümesini betimler (Ni, 1993). Bu model, gerçekleştirimde ortaya çıkacak problemleri ele almayan basit bir modeldir. Bazı çalışmalarda çift

yönlü kanallar köşe gibi ele alınsa da, tez kapsamında çift ve tek yönlü kanalların tümü kenar olarak ele alınacaktır.

Ağ özelliklerinin çizge modelinde gösteriminden bazı temel özellikler elde edilmiştir:

- *Düğüm derecesi: Bir* düğüme komşularından bağlanan kanal sayısını gösterir.
- *Çap: Ağ* içindeki iki düğüm arasındaki en büyük uzaklığı gösterir.
- *Düzenlilik: Tüm* düğümleri aynı derecede olan ağ *düzenli ağ* olarak tanımlanır.
- *Simetri: Tüm* düğümlerinden benzer gözüken ağ *simetrik ağ* olarak tanımlanır.

Dolaysız ağlar; *topoloji, yönlendirme, akış kontrolü* ve *anahtarlama* olmak üzere dört temel başlıkta ele alınırlar (Dally, 1990b).

3.1. Dolaysız Ağlarda Topoloji

Topoloji, bileşen bağlantılı ağ içindeki düğümlerin birbirleriyle nasıl bağlanacağını tanımlar (Duato, 2002). Dolaysız ağlar için, ideal topoloji, bütün düğümlerin birbirlerine bağlandığı ağ modelidir. Bu ağ modelinde, gönderilen herhangi bir iletinin bir ara düğümden geçmesine gerek kalmaz. Düğümler arasında tam iletişim sağlayan bu model, *N* düğüm içeren bir topolojide, her düğümün N bağlantı destekleyen bir yönlendirici barındırmasını gerektirir (kendi işlemcisine olan yerel bağlantısıyla birlikte). Bu durumda ağ maliyeti makul seviyelerden yüksek seviyelere çıkar. Bununla birlikte bir düğümün destekleyeceği maksimum bağlantı sayısı, donanım kısıtlamalarıyla sınırlanmıştır. Dolayısıyla çok düğümlü koşut sistemlerin bu topolojiyle gerçekleştirimi olanaksızdır. Tam bağlantılı ağ topolojilerinin gerçekleşebilir olmaması, dolaysız ağlar için birçok topolojinin önerilmesini sağlamıştır. Önerilen topolojilerde, belirtildiği gibi, komşu olmayan düğümler arasındaki iletilerin iletimi, kaynak ve hedef düğüm arasındaki bir dizi düğümden geçmesiyle sağlanır.

Tez kapsamında gerçekleştirilen dolaysız ağ benzetiminde *torus* topolojisi kullanılmıştır. *Torus* topolojisi izleyen bölümde anlatılmaktadır. Dolaysız ağlarda kullanılan topoloji tanımları ve hasır doku topolojisi (Ek-2)'de anlatılmıştır.

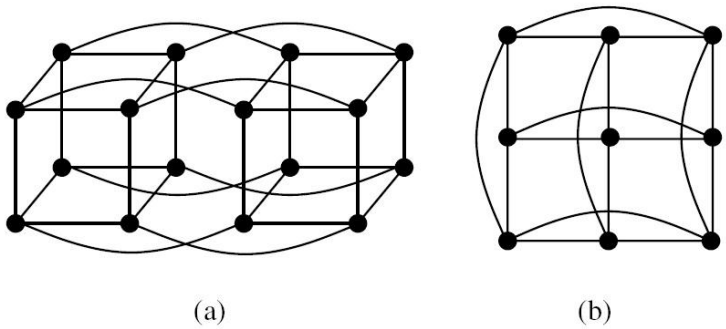
3.1.1. k-ary n-küp topolojileri (Torus, Hypercube)

k-ary n-küp topolojilerinde, bütün düğümler eşit sayıda komşuya sahip olurlar. k-ary n-küp topolojileri, genel olarak n boyutlu hasır dokulara benzer olarak tanımlanırlar (Ek-2). Fakat k-ary n-küp topolojilerinde her boyutta eşit sayıda düğüm bulunur. Dolayısıyla her $k_i = k$ olur. Bununla birlikte k-ary n-küp topolojileri için komşu düğüm tanımı farklıdır. k-ary n-küp topolojisinde, herhangi *x* ve *y* düğümleri için, denklem (3.1)'deki koşullar sağlanıyorsa, x ve y düğümleri *komşu düğümlerdir* denir (Ni, 1993).

$$\forall i \; için \; \sigma_i(x) = \; \sigma_i(y) \; ve \;\; bir \; değer \; hariç \; \forall j \; değeri \; için$$

$$\sigma_j(y) = \left(\sigma_j(x) \; \pm 1\right) mod \; k \qquad (3.1)$$

Tanımdaki modül aritmetiği tanımı, hasır dokuda bulunmayan *sarmal kanallar (wraparound channels)* sayesinde sağlanır. Bu kanallar her boyuttaki en küçük ve en yüksek koordinat değerli düğümler arasında komşuluk bağlantısı kurarlar. Bir k-ary n-küp topolojisinde k^n düğüm bulunur. *k*=2 için tüm düğümler her boyutta bir tane olmak üzere n tane komşuya sahip olurlar. k> 2 için tüm düğümler her boyutta iki tane olmak üzere 2n tane komşuya sahip olurlar. Şekil 3-3'de bir 2-ary 4-küp (Şekil 3-3a) ve 3-ary 2-küp (Şekil 3-3b) gösterilmiştir. Şekil 3-3'de 2-ary 4-küp bir *Hypercube* (Dally, 2004,Bhuyan, 1984) ve 3-ary 2-küp bir *torus (*Duato, 2002, Dally, 1990a) betimler.



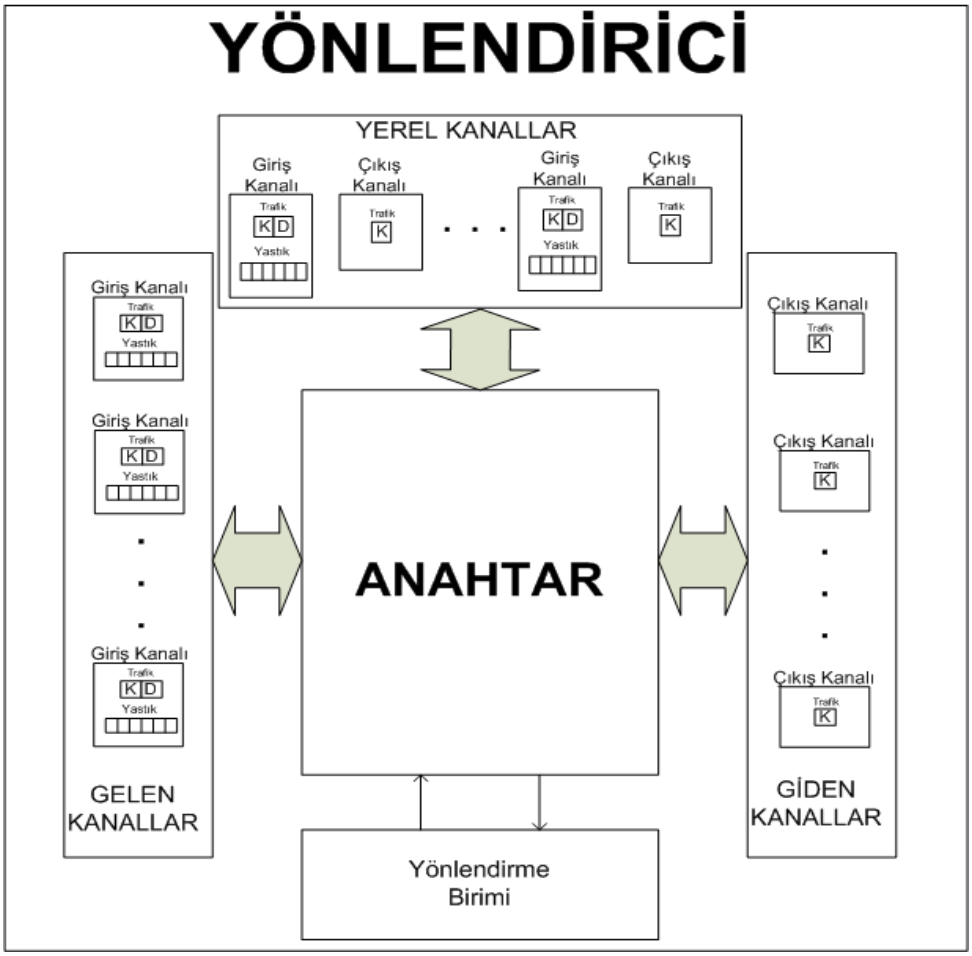
Şekil 3-3.k-ary n-küp topolojileri.

(a) 2-ary 4-küp, (b) 3-ary 2-küp (Duato, 2002)

k-ary n-küp topolojilerinden biri olan *torus* topolojisinde, sahip olduğu sıkı dikey topoloji özellikleri, topoloji içindeki bütün düğümlerin homojen olmasını sağlar ve topoloji üzerinde genel geçer yönlendirme algoritmaların tanımlanmasını kolaylaştırır. Bu özelliklerinden dolayı *torus* topolojisi gerçekleştirilmiş süper bilgisayarlarda tercih edilmiştir. MIT tarafından geliştirilmiş CrayT3Dsüper bilgisayarlarında (Oed, 1993, Kessler, 1993, CISL, 2009) *torus* topolojisi kullanılmıştır.

3.2. Dolaysız Ağlar İçin Genel Yönlendirici Modeli

Dolaysız ağlar için genel bir yönlendirici modeli Şekil 3-4'te verilmiştir.



Şekil 3-4. Genel Yönlendirici Modeli

Şekilde belirtildiği gibi, yönlendirici modeli genel olarak aşağıdaki bileşenleri içerir:

- *Anahtar: Bu* bileşen, yönlendirici içindeki gelen yastıklarının, giden yastıklarına anahtarlamasını sağlar. Yüksek hızlı yönlendiriciler bu bileşen için çapraz (*crossbar*) bağlantılı anahtarları kullanırlar.

- *Yönlendirme Birimi: Bu* bileşen gelen bir ileti için giden kanalı seçer ve iletim için yönlendirici anahtarını kurar. Eğer birden fazla gelen kanal, aynı giden kanal için istekte bulunursa, bu birimin belli bir protokol kapsamında seçim (hakimlik) yapması gerekir. Eğer istenilen kanal kullanımda ise, ileti gelen yastığında bekletilir. Yönlendirme algoritmaları Bölüm 3.5'te açıklanacaktır.
- *Gelen (Sanal) Kanal*[1]*(Gelen Kapısı):* Yönlendirici içindeki giriş kapılarıdır. Yönlendirici her komşusu için bir sanal kanal içerir. Sanal kanallar alıcı iletilerinin saklandığı yastık, çıkış kanal/kapı adresi ve kanal durumu bilgilerini içerir. Sanal kanallar üç bileşen içerirler:
    - *Kanal Durumu:* Sanal kanalın durum bilgisini içerir. Sanal kanallar *Etkin, Beklemede, Boş* durumlarında olurlar. Sanal kanal bir çıkış kapısı trafiğini elde ettiğinde Etkin, bir kanal trafiğini beklediğinde Beklemede ve Trafik olmadığında Boş durumlarında olurlar.
    - *Çıkış Kanalı: Sanal* kanalın trafiği olduğu durumda, trafiğin hangi kanala iletildiği bilgisini içerir.
    - *Yastıklar: Kuyruk* veri yapısındaki yastıklardır. Bu yastıklarda, giden kanallar için iletilecek paketler/iletiler saklanır. Yastıklar kanallara dağıtılmış biçimde veya yönlendirici içindeki merkezi bir yastık üzerinden yönetilebilirler (Kotapati, 1999).
- *Giden Kanal (Giden Kapısı):* Ağ birimleri arasında veri iletişiminin yapıldığı hatlardır. Yönlendiriciler ile işlemci arasındaki kanallar yerel giden kanallar, yönlendiriciler arasındaki kanallar dış giden kanallardır. Dış giden kanallar yönlendiricilerdeki çıkış kapılarına (*port*) karşılık gelmektedirler.
- *Yerel Kanallar: Y*önlendiricinin bağlı olduğu düğümdeki işleyici için bir iletişim arayüzü sağlar. Bu bileşen işleyici tarafından gelen yerel çıkış kanalları ile işleyici tarafına giden yerel giriş kanallarının yönetimini sağlar. Bu kanalların sayıları gerçekleştirim seçimlerine göre değişkenlik gösterir. Yerel çıkış kapısının, işleyici üzerinde sadece tek bir giden – gelen kanal çifti sağladığı yönlendiricilere *tek kapılı* (*single port*) *yönlendiriciler* denir. Bu yönlendiricilerde, işlemci yönünde sadece tek bir gidiş ve geliş trafiğine izin verilir. İşlemcinin yönlendirici üzerindeki her kapı için, ters yönde bir kapı içerdiği yönlendiriciler ise *tam kapılı (all-port) yönlendiriciler* olarak

[1] Sanal Kanal terimi, birden fazla mantıksal kanalın tek bir fiziksel kanalla eşleştiği iletişim protokolü için de kullanılmaktadır. Bu terim Ek-3'teki*Sanal Kanallar* bölümünde açıklanacaktır.

tanımlanırlar. Bu yönlendiricilerde, işlemci yönlendiricideki gelen-giden kapısı kadar trafiği eş zamanlı olarak destekler (Yih-Jia Tsai, 1996).

Yönlendirici biriminin başarımı iki parametre belirler. Bir ileti yönlendiriciye ulaştığında, iletinin hangi çıkış kanalına yönlendirileceğine karar verilmesi gerekir. Bu kararın verilmesi için geçen süre *yönlendirme gecikmesi* olarak tanımlanır. Genel olarak, yönlendiricinin kanal seçiminden sonra anahtarı kurması için gereken süre de yönlendirme gecikmesine eklenir. Yönlendirici içi yönlendirme yolu seçildikten sonra, iletinin gelen kanal yastığından, anahtar boyunca geçerek giden kanalına aktarılması gerekir. Bu aktarım bir yayılım beklemesine (*propagation delay*) sebep olur ve geçen süre *yerel akış kontrolü beklemesi (internal flow control latency)* olarak tanımlanır (Duato, 2002). Benzer şekilde, iletinin giden kanal yastıklarından fiziksel kanallara dolayısıyla ağ ortamına aktarımı için geçen süre *dış akış kontrolü beklemesi (external flow control latency)* olarak tanımlanır. Yönlendirme beklemesi, akış kontrol beklemeleriyle birlikte ileti beklemesini tanımlar (Duato, 2002).

Anahtarlama katmanı, genel bağlamda gerçekleştirim ve göreli akış kontrol işlemlerine göre sınıflandırılabilirler. Bazı anahtarlama yaklaşımlarında, bu süreler yönlendirme karar süresiyle üst üste bindirilebilir.

Anahtarlama teknikleri, akış kontrolü oranlarının kanal veya ileti düzeyinde akış kontrol birimlerinin büyüklüğüne göre değişiklik gösterirler. Genel bağlamda, bir ileti sabit boydaki *paketlere,* paketler ise daha alt birimler olan *flit (flow control unit)* iletilerine bölünebilir. Bir sistem ağında, kullanılan kanalların aktarım sığasına göre, bir saat vuruşunda aktarılabilecek kadar ikili içeren akış kontrol birimi *phit* olarak tanımlanır. *flit* mantıksal bir bilgi birimini gösterirken, *phit* (*physical flit*) fiziksel bir bilgi birimini gösterir (Duato, 2002). Şekil 3-5'te akış kontrol birimi yaklaşımları gösterilmiştir.

Şekil 3-5. Akış kontrol birimleri

3.3. Dolaysız Ağlarda Anahtarlama

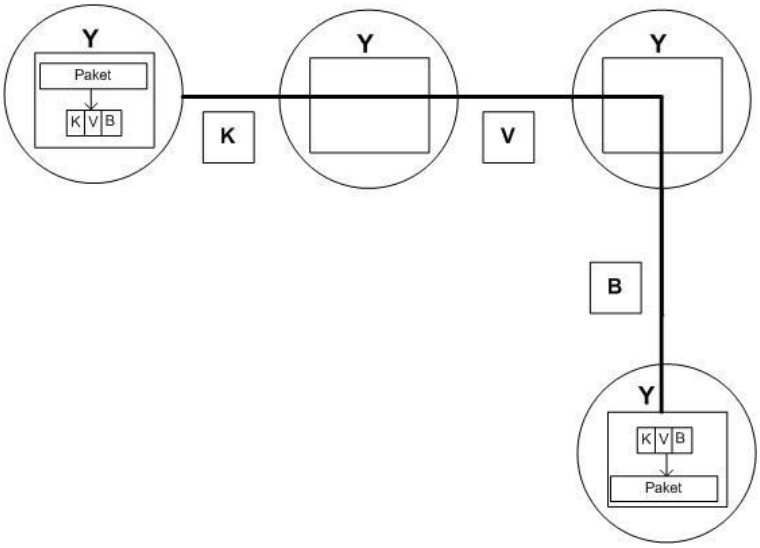
Yerel ve geniş alan ağlarında, işleyiciler arası iletişim, belli bir hiyerarşide sıralanmış bir servisler bütünü olarak görülebilir. Bu servisler ikili akışlarının kontrol edildiği fiziksel katmandan başlayıp, paketlenme, veri sıkıştırma ve veri şifreleme gibi işlemlerin uygulandığı yukarı seviye protokollere kadar devam eder. Benzer biçimde, dolaysız ağlar için iletişim katmanları üç temel katmanda toplanabilir, *yönlendirme katmanı, anahtarlama katmanı ve fiziksel katman* (Duato, 2002). Fiziksel katman, bağlantı seviyesindeki protokolleri içerir ve komşu yönlendiriciler arasındaki bağlantıların yönetimini sağlar. Anahtarlama katmanı, iletilerin ağa iletimi için bir takım düzenekler gerçekleştirerek, fiziksel katmandaki protokollerin daha etkin çalışmasını sağlar. Yönlendirme katmanı ise ara düğümlerde hangi çıkış kanalının seçileceğine karar vererek iletinin takip edeceği yolun kurulmasını sağlar. Yönlendirme protokolünün özellikleri ve başarımı, seçilen anahtarlama katmanı ile birebir ilişkilidir.

Bir ileti veya paket başlığı bir ara düğüme eriştiğinde, anahtarlama katmanı yönlendirici anahtarının nasıl ve ne zaman kurulacağına karar verir. Bu kurulum işlemi, gelen paket kanalında veya bu kanalla eşleşmiş yastıkta, bulunan paketin yönlendirme algoritmasıyla seçilen giden kanalına aktarılmasını kapsar (Duato, 2002).

3.3.1. Wormhole Anahtarlama

Tez kapsamında gerçekleştirilen ağ benzetiminde *wormhole* anahtarlama modeli kullanılmıştır. Dolaysız ağlarda kullanılan diğer anahtarlama türleri (Ek-3)'te anlatılacaktır.

Bir paketi bütünüyle yönlendiricide yastıklama gereği, Yönlendirici içinde yüksek sığalı yastıklara gereksinim duyar. Bu durum küçük ve hızlı yönlendiricilerin yapılmasını zorlaştırır.



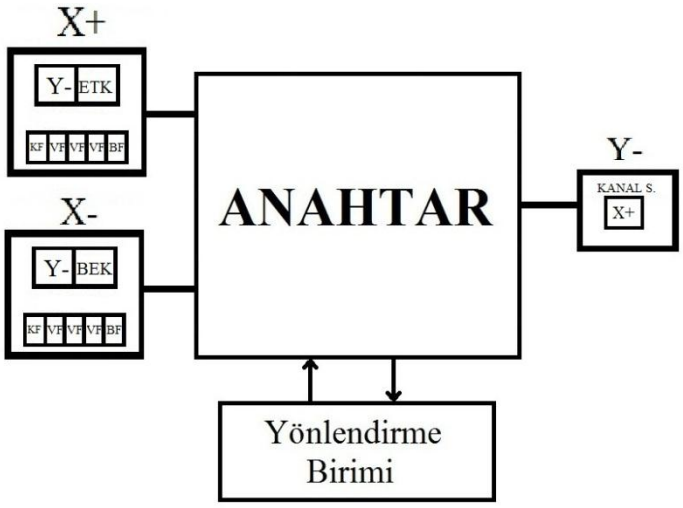
Şekil3-6.*Wormhole* Anahtarlama

(K=Kuyruk *flit*, V=Veri *flit*, B=Başlık *flit* Y=Yönlendirici)

Wormhole anahtarlama yaklaşımında, ileti paketleri de alt paketlere parçalanır ve ağ içinde ardıl bir düzende (*pipelining*) iletilir (Şekil3-6).

Wormhole anahtarlamada yönlendirici için ihtiyaç duyulan yastık boyları büyük ölçüde azdır. Genel olarak yönlendiriciler içinde bulunan giden ve gelen kanal yastıkları, bir kaç *flit* saklayabilecek kapasitededir. Örneğin, CrayT3D sisteminde yönlendirici yastıkları 1 *flit* saklayabilir ve her *flit* 16-ikililik 8 *phit* biriminin birleşmesinden oluşur. *Wormhole* anahtarlama küçük ve hızlı yönlendiricilerin yapılabilmesine olanak verir. Bunun yanında, *wormhole* yönlendirmelerde paket çoğaltma ile bir alt paket birden fazla kanalın yastığına kopyalanarak çoklu kanaldan koşut olarak yollanabilir. Bu yaklaşım çoğa gönderim (*multicast*) ve yayım (*broadcast*) gerçekleştirimlerinde faydalı olur.

Bir pakete ait başlık *flit* bir düğüme ulaştığında, bu *flit* iletisinin, iletilmeden önce yönlendirici içindeki üç kaynağı ele geçirmesi gerekir: *flit* yastığı, gelen kapı (*inport*) durumu ve istenilen çıkış kapısı/kanalı. Başlık *flit* paketi, istenilen kaynakları ele geçirdikten sonra paket trafiği iletilmeye başlar. Pakete ait diğer *flit* iletiler (veri *flit* iletileri ve kuyruk *flit* iletisi), başlık *flit* paketinin açtığı kanal boyunca iletilirler. Kuyruk *flit* dosyası iletilirken giriş ve çıkış kapıları ile bu kapılara atanmış kaynaklar serbest bırakılırlar.



Şekil 3-7.*Wormhole* anahtarlama çakışma durumu

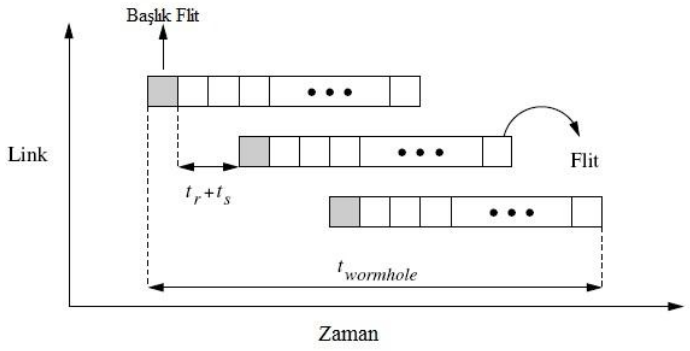
(ETK = Etkin, BEK = Beklemede, BF=Başlık *flit* VF= Veri *flit* KF= Kuyruk *flit*)

Bir yönlendirici üzerinde iki farklı *wormhole* trafiği Şekil 3-7'de gösterilmiştir. Çizim sadeliği açısından, şekilde sadece ilgili kanallar gösterilmiştir. Şekildeki durumda X+ ve X- gelen kapılarının her ikisi de Y- çıkış kanalına ihtiyaç duymaktadırlar. Fakat yönlendiricinin seçimi bağlamında X+ giriş kapısı kanalı ele geçirmiştir ve kapıya ait trafik sona erene kadar X- kanalı ve Y- kanalını kullanmak isteyecek diğer giriş kapıları bekleme durumunda kalacaklardır. X- kanalı, yastık kapasitesi kadar *flit* paketini saklayabilir. Fakat X- kanalı yastığı dolunca, akış kontrolünün sağladığı yöntem bağlamında gönderen bilgilendirilecek ve gönderen *flit* göndermeyi X- kanalı kaynağı ele geçirene kadar durduracaktır.

Wormhole anahtarlamada, yönlendirme ve kontrol bilgileri sadece başlık *flit* iletilerde bulunur. Başlık *flit* iletileri dışındaki *flit* iletiler herhangi bir adres bilgisi içermezler. Dolayısıyla bir kanal, bir ileti trafiğine atandığında, aktarılan iletiye ait son *flit* atanan kanaldan iletilmeden önce, kanalın başka bir trafiğe atanması veya başka bir deyişle ileti trafikleri arasında paylaşılması olanaksızdır. Bu bağlamda *wormhole* anahtarlama devre anahtarlamaya (Ek-3) yakın bir yaklaşım gösterir.

Wormhole anahtarlama engellemelerin olmadığı durumda, Sanal Kısa yol anahtarlama (Ek-3) gibi davranır ve *flit* iletiler ardıl bir düzende iletilirler. Bununla birlikte, *wormhole* anahtarlamada iletimde bir engelleme oluştuğunda, iletim yolu boyunca ara düğümlerde bulunan *flit* iletiler, engelleme süresi boyunca bulunduğu ara düğümde saklanır. *Wormhole* anahtarlamada engellemelerin bu şekilde ele

alınışı, ağ içinde kilitlenmelerin (Bölüm.3.4.4) oluşmasını kolaylaştırır. Çünkü bekleme süresince flit iletileri saklayan kanalların başka bir ileti trafiğine atanması olanaksızdır. Bu duruma çözüm yaklaşımı olarak bazı yaklaşımlar geliştirilmiştir. Sanal Kanallar (Ek-3) bu yaklaşımlardan biridir. .Şekil 3-8'de*wormhole* anahtarlama modelinin zaman ekseni üzerinde davranışı gösterilmiştir.



Şekil 3-8.*Wormhole* modelinin zaman ekseninde davranışı (Duato, 2002)

Wormhole anahtarlama modelinin geniş ölçekli gerçekleştirimlerinde, yayılım beklemeleri sebebiyle, düğümler arası yüksek hızlı zaman uyumlu saat vuruşu kullanmak zorlaşır. Bu durum düğümler arası veri alışverişlerinde akış kontrol birimlerinin etkin yöntemler kullanmasını zorunlu kılar. Bölüm 3.3.1 'te, çok işlemcili dolaysız ağlar için önerilen akış kontrolü, geri bildirim yöntemleri açıklanacaktır.

Wormhole anahtarlama yaklaşımıyla bir paketin iletim gecikmesi Denklem (3.2) ile açıklanabilir (Duato, 2002).

$$t_{wormhole} = D(t_r + t_s + t_w) + max(t_s, t_w)\left\lceil \frac{L}{W} \right\rceil \quad (3.2)$$

Denklem (3.2) ifadesinde; iletinin *L* ikiliden oluştuğu ve *W* ikiliden oluşan bir başlık *flit* içerdiği düşünülmüştür. Denklemde t_r *yönlendirme gecikmesi,* t_s *anahtar yayılım gecikmesi ve* t_w *iletim gecikmesini* göstermektedir. İletim yapılan kanalların her saat vuruşunda *W* ikili iletebildiği varsayılmıştır.

Wormhole anahtarlama MIT tarafından geliştirilen J-Machine süper bilgisayarında anahtarlama yöntemi olarak kullanılmıştır (Nuth, 1992).

3.4. Dolaysız Ağlarda Akış Kontrolü

Bir paketin bulunduğu düğümden bir sonraki düğüme iletilmesi için o düğümle iletişimi sağlayan kanal aktarım sığası ve yastık kaynaklarını ele geçirmesi gerekir. Akış kontrolü, ağ boyunca iletilecek bir iletiye, ağ kaynaklarının hangi bağlamda atanacağını tanımlar (Dally, 2004). Ağ içindeki kaynaklar kanal aktarım sığası, yastıklar ve kontrol durum değişkenlerini kapsar. Kontrol durum değişkenleri, paketlere atanmış kaynakları ve paketlerin ağdan iletimini izlerler. Akış kontrolü yönteminin amacı, gönderen ve alıcı düğümler arasındaki iletişim protokolünü belirleyerek, bu kaynakları etkin biçimde yönetmek ve alıcı düğümün işleyebileceğinden fazla iletiyle (yastık sığasından fazla iletiye) boğulmasını engellemektir.

Dolaysız ağlarda akış kontrolü iki seviyede gerçekleşir. Alıcı tarafında yeterli yastık alanı olmadığı durumda, alıcı iletimi durdurur. Bu yaklaşıma *ileti düzeyinde akış kontrolü* denir. Bununla birlikte yönlendiriciler arasındaki kanalın aktarım sığasının seçilen paket boyundan düşük olduğu durumda iletinin aktarımı birden fazla saat vuruşu alacaktır. Bu durumda saat vuruşlarının her birinin kontrol edilmesi ve ileti bölümlerin düzgün biçimde alınması gerekir. Bu akış kontrolü ise *kanal akış kontrolü* olarak tanımlanır (Duato, 2002).

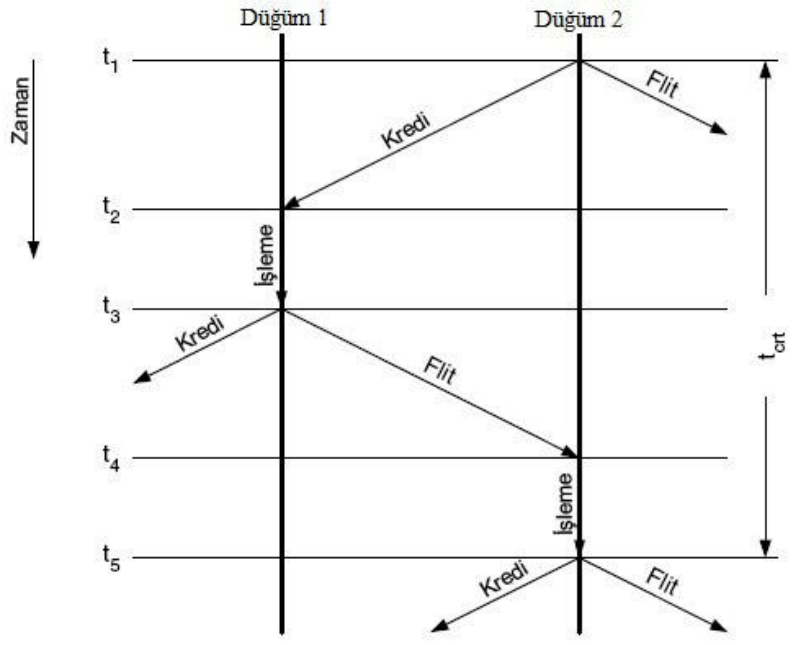
Akış kontrol yöntemleri, genel olarak kaynakların atanması (bir paketin iletişim için çıkış kanalına atanması) ve çakışmaların giderilmesi (aynı anda iki paketin aynı kaynağa ihtiyaç duyması) koşullarını ele alırlar. Çözüm yaklaşımlarına göre akış kontrolleri *Yastıklı Akış Kontrolü* ve *Yastıksız Akış Kontrolü* olmak üzere iki temel grupta toplanır. Akış kontrol türleri (Ek-4)'te anlatılmıştır. Bölüm 3.4.1'de tez kapsamında gerçekleştirilen akış kontrol yöntemi olan *Kredi Tabanlı Akış Kontrolü* anlatılacaktır.

3.4.1. Kredi Tabanlı Akış Kontrolü

Kredi tabanlı akış kontrollerinde, her kanal, karşı taraftaki düğümün kendisine atanmış giriş kapısındaki (sanal kanal) boş olan yastık alanı sayısını tutar (Dally, 2004). Kanaldaki boş yastık sayısı bilgisi *kredi sayısı* olarak isimlendirilir. Başlangıç durumunda, bütün kanallar karşı taraftaki yastık sığası kadar krediye sahip olurlar. Düğümler, alıcı düğümlere gönderdiği her iletiden sonra ilgili çıkış kanallarındaki

kredilerini bir azaltır ve kredileri olmadığında alıcılara iletimi durdururlar. Alıcı düğümler de giriş kapılarındaki yastıklarından her bir *flit* gönderildiğinde (boşaldığında), gönderen düğümlerine yeni bir ileti gönderimine onay veren alındı paketi (ACK paket) gönderirler. Bu alındı paketi *kredi* olarak isimlendirilir. Söz konusu protokol, her bir düğüm tarafından uygulandığında, gönderen düğümden alıcı düğüme kadar bütün düğümlere bir geri bildirim mekanizması kurulmuş olur. Kredi tabanlı akış kontrolü *flit* tabanlı bir akış kontrolü yöntemidir.

Şekil 3-9'da, iki düğüm arasındaki kredi tabanlı akış kontrolü zaman ekseni üzerinde gösterilmiştir. t_1 anında Düğüm$_2$ içinde Düğüm$_1$'e atanmış yastık doludur, Düğüm$_2$ içindeki ilgili sanal kanal (giriş kapısı) karşı tarafa gönderim trafiğini elde edince karşıya bir *flit* gönderir ve yastıkta yer açılmış olur. Aynı anda Düğüm$_1$'e bir kredi yollar. Kredinin t_2 anında Düğüm$_1$'e ulaşmasından ve işlenmesinden sonra, Düğüm$_1$ t_3 anındaDüğüm$_2$'ye bir sonraki *flit* iletisini ve trafik aldığı düğüme de boşalan yastık alanı bilgilendirmesi amaçlı bir kredi yollar. Yollanan *flit* iletisinin t_4 anında Düğüm$_2$'ye ulaşmasından ve işlenmesinden sonra, Düğüm$_2$ içindeki giriş kapısı elde ettiği trafik üzerinden t_5 anında gelen *flit* ve bir sonraki alım istemi için Düğüm$_1$'e bir kredi yollar.



Şekil 3-9. Kredi tabanlı akış kontrolü zaman ekseninde örneklenmesi (Dally, 2004)

Bir düğümün trafik aldığı düğüme (yastığa) kredi yollamasından, aynı düğüme bir sonraki *flit* alımı için kredi yollamasına kadar geçen süre kredi gidiş-geliş süresi

(*credit round trip time*[2]) denir (Dally, 2004). Şekil 3-9'da bu süre t_{crt} olarak gösterilmiştir ve (t_5-t_1) olarak hesaplanmıştır.

Ağ içindeki *flit* boyunun L_f ikili olduğu ve aktarım sığası *b* olduğunu varsayalım. Her yönlendiricinin sadece tek *flit* saklayabilen yastıklara sahip olduğu senaryoda, her düğüm iletim öncesinde bir kredi beklemek durumunda kalır. Bu durumda bir *flit* iletisinin maksimum iletim süresi t_{crt} olacaktır. Bu durumda kanal üzerindeki ikili sıklığı (*bit rate*) $\frac{L_f}{t_{crt}}$ olur. Sanal kanal üzerinde F *flit* saklayabilecek yastık bulunduğu senaryoda ise, düğüm krediyi beklemeden F *flit* yollayabilir. Bu durumda düğüm t_{crt} sürede F *flit* yollayabilir veya saniyede $\frac{F*L_f}{t_{crt}}$ikili gönderebilir. Dolayısıyla, akış kontrol yönteminin veri iletim hacmini düşürmemesi için, çıkış kanalıyla eşleşmiş yastık için gereken yastık boyu Denklem (3.3) ile gösterilmiştir.

$$F \geq \frac{t_{crt}\, b}{L_f} \tag{3.3}$$

3.5. Dolaysız Ağlarda Yönlendirme

Yazılımcının bakış açısından, düğümler arasındaki veri alış verişi birimi *iletilerdir.* Koşut uygulamalardaki ileti büyüklüğü, uygulamalara göre çeşitlilik gösterebilir. Ağ kaynaklarının etkin ve adil kullanımı için, genellikle iletiler alt ağ paketlerine bölünürler. *Paket,* ağ ortamındaki en küçük iletişim birimidir. Paketin hedef adresi gibi sistem alan ağı içinde iletimi için gereken bilgiler paket başlığı içinde tutulur. İletinin hedef düğüme erişmesi için ara düğümlerden geçmesi gereken topolojilerde, *yönlendirme algoritması* iletinin takip edeceği yolu seçer. İleti bir ara düğüme vardığında, ara düğüm yönlendirme algoritması bağlamında bir sonraki kullanılacak kanala karar verir. Seçilen kanalların hepsinin meşgul olduğu durumda ileti engellenir. Sistem alan ağlarında dolaşan iletilerin takip edeceği yolun seçilmesi ve seçilen yol kapsamında hangi kanalları kullanacağı kararlarının verilmesini kapsayan işlemlerin bütünü *yönlendirme* olarak tanımlanır (Duato, 2002). Bileşen bağlantılı ağların başarımında, etkin yönlendirme önemli bir rol oynar.

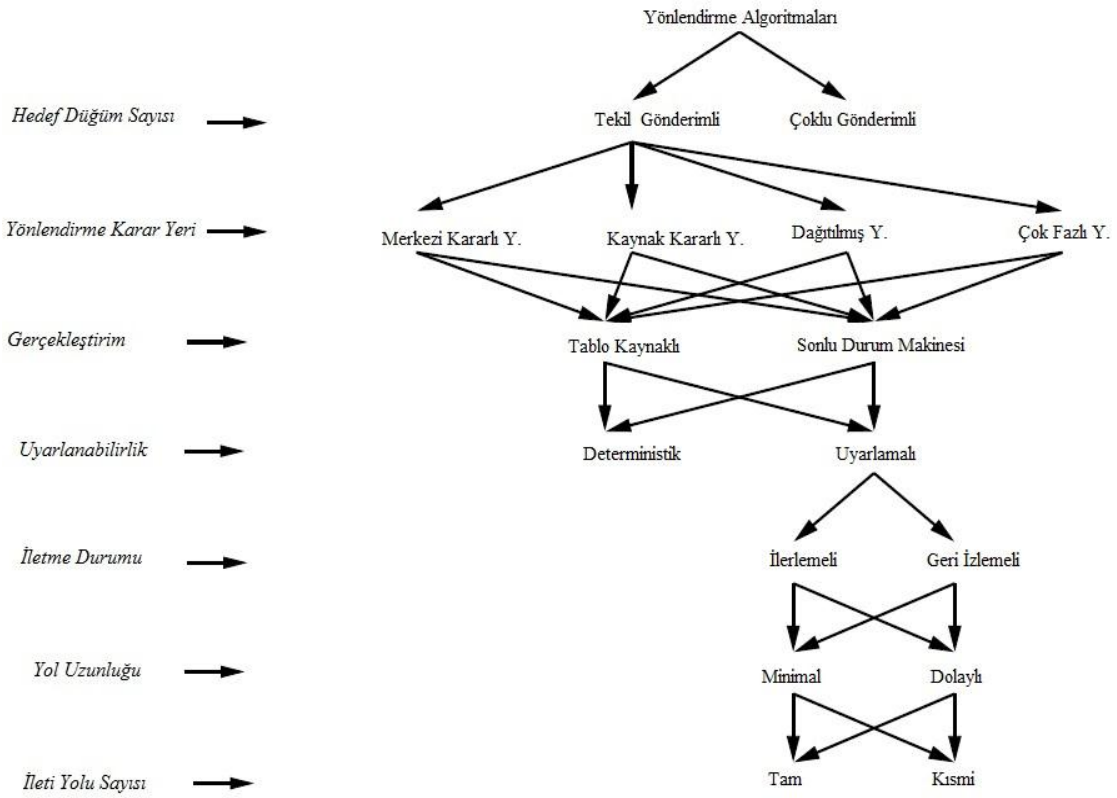
[2]Geliş Gidiş Süresi (Round Trip Time, RTT) Bir imin gönderilmesinden, bu ime ait alındı iminin ulaşmasına kadarki geçen toplam süre

Tez kapsamında *wormhole* anahtarlama ele alındığı için, *wormhole* anahtarlamada kullanılan yönlendirme algoritmaları incelenecektir.

Bileşen bağlantılı ağların aşağıdaki özellikleri, kullandığı yönlendirme algoritmasına bağlı olarak belirlenir (Duato, 2002):

- *Bağlanabilirlik (Connectivity):* Ağ içindeki herhangi bir kaynak düğümün herhangi bir hedef düğüme ileti gönderebilmesidir. Başka bir deyişle, ağ içindeki düğümler arasında tam iletişim sağlanmasıdır.
- *Uyarlanabilirlik (Adaptivity):* Düğümler içinde hatalar olduğunda veya trafikte tıkanıklık oluştuğunda, alternatif yollara yönlendirme yapılabilmesidir.
- *Kilitlenmelerden Bağımsız (Deadlock and Livelock Freedom):* Bu özellik ağ içindeki herhangi bir iletinin veya paketin sonsuza kadar engellenmeyeceğinin veya iletiye seçilen yönlendirme yolu boyunca sonsuz döngülere girmeyeceğini garanti eder.
- *Hata Hoşgörüsü (Fault Tolerance): Ağ* içindeki hatalı bileşenler olduğunda dahi ileti yönlendirmesinin yapılabilmesidir. Hata toleranslı yönlendirme işlemleri için uyarlanabilir yönlendirme algoritmaları zorunlu değildir. Söz konusu iletiyi bir ya da daha fazla fazda yönlendirerek ve ara düğümlerde saklayarak da sağlanabilir.

3.5.1. Yönlendirme Algoritmalarının Sınıflandırılması



Şekil 3-10. Yönlendirme algoritmalarının sınıflandırılması (Duato, 2002)

Şekil 3-10'da yönlendirme algoritmalarının sınıflandırma ölçütleri verilmiştir. Yönlendirme ölçütleri Şekil 3-10'da italik olarak verilmiştir ve her satırda belirtilen ölçüte alternatif yaklaşımlar belirtilmiştir. Ölçütler arası ilişkiler ok geçişleri ile gösterilmiştir. Yönlendirme algoritmalarının sınıflandırılması (Ek-5)'te anlatılmıştır.

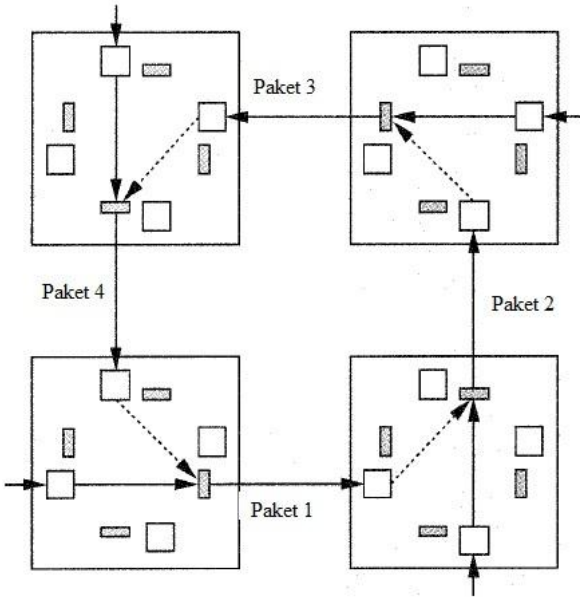
İleti yönlendirilirken, bir sonraki düğümün her iletilen düğümde karar verildiği yönlendirme yaklaşımı *dağıtılmış yönlendirme;* yönlendirme kararlarının bütününü kaynak düğümün belirleyerek iletim yoluna karar verdiği yaklaşım ise *Kaynak Kararlı Yönlendirme* olarak tanımlanırlar.

Yönlendirme algoritmaları uyarlanabilir durumlarına göre *Uyarlamalı Yönlendirme* ve *Durağan Yönlendirme* olmak üzere iki gruba ayrılırlar. Durağan yönlendirmede herhangi bir kaynak/hedef çifti için her zaman aynı iletim yolu kurulur. Yönlendirme algoritmasının iletim yönü seçimi her zaman aynıdır. Uyarlanabilir yönlendirmede ise

iletim yönü seçilirken, seçim anındaki trafik durumu da dikkate alınır. Seçilen ileti yönü kullanımda olduğunda ileti boştaki başka bir kanala yönlendirilir.

3.5.2. Dolaysız Ağlarda Kilitlenmeler

Genel bağlamda, bir iletim trafiğinin herhangi bir sebeple kendi veya başka iletim trafiklerini engelleyip, sonuçlanmayacak döngülere soktuğu durumlar kilitlenmeler olarak tanımlanır. Dolaysız ağlarda, iki çeşit kilitlenme bulunur: kilitlenme (*deadlock*) ve canlı kilitlenme (*livelock*).



Şekil 3-11. Kilitlenme örneği (Ni, 1993)

Bir iletinin, ihtiyaç duyduğu kaynağı elinde tutan iletinin beklediği kaynağı elinde tuttuğu durum, kilitlenme (*deadlock*) olarak tanımlanır. Başka bir deyişle, birbirlerinin bekledikleri kaynakları karşılıklı olarak elinde tutan iletiler arasında karşılıklı kilitlenme olur. Şekil 3-11'de bir ölü kilit örneklenmiştir. Şekil 3-11'de gösterilen dört paketten, paket 1 paket 2'nin, paket 2 paket 3'ün, paket 3 paket 4'ün ve paket 4 paket 1'in elinde tuttuğu kanalı beklemekte; sonuçta dört paket birbirlerini kilitlemektedir.

Bir diğer kilitlenme türü olan canlı kilitlenmede ise iletilerin sonsuz döngüde birbirlerini beklemeleri durumu yoktur. Canlı kilitlenme durumunda ileti ağ ortamında döngüsel bir harekette dolaşır ve iletilmesi gereken hedef düğüme erişemez.

Anlaşılacağı üzere, kilitlenmelerin ağ kaynaklarını elinde tutan iletilerin, ihtiyaç duydukları kaynaklar için döngüsel biçimde birbirlerini beklemelerinden kaynaklanır.

Paket anahtarlama ile sanal kısa yol anahtarlama (Ek-3) yaklaşımlarında kaynaklar, yastıklardır. Devre anahtarlamalı ve *wormhole* anahtarlamada ise kaynaklar kanallardır; çünkü iletilen paketler kanalları ve bu kanallara ayrılmış alt paket yastıklarını tutarlar.

Ağlarda karşılıklı kilitlenmeleri çözmek için geliştirilen yaklaşımlardan biri, karşılıklı kilitlenmeye sebep olabilecek paketlerin düşürülmesi veya tekrar yönlendirilmesidir. Bu yaklaşımlardan; paketin tekrar başka bir kanala yönlendirilmesi en kısa olmayan yönlendirmelere neden olur. Bu paketlerin düşürülmesi ise, kaynak düğümden paketin tekrar istenmesine ve yollanmasına; dolayısıyla ağ trafiğinin yükselmesine sebep olur. Bu sebeplerden dolayı, bu yaklaşım dolaysız ağlarda kullanılmaz.

Karşılıklı kilitlenmeler yönlendirme algoritması sayesinde engellenebilir. Ağ kaynaklarını yönetirken, bu kaynakları kullanacak paketlerin isteklerini tekdüze bir sırada atayarak döngüsel beklemeler (karşılıklı kilitlenmelerin temel sebebi) engellenebilir (Dally, 1987). Böylece bu kaynakları içerecek karşılıklı kilitlenmeler engellenmiş olur.

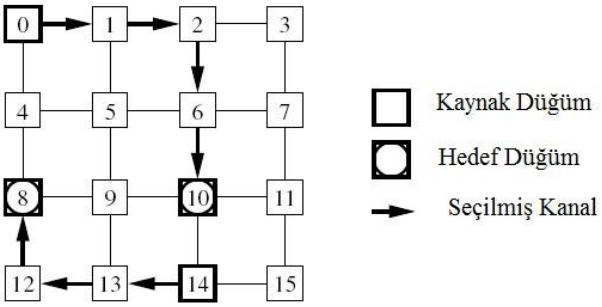
Wormhole Ağlarda kritik kaynaklar kanallardır. Kanallar paketlere atanırken, döngüsel beklemeleri engellemek için, kanallar arası tek yönlü bir D=G(C,E) bağımlılık çizgesi kullanılabilir. Burada C(D) kümesi ağ üzerindeki bütün tek yönlü kanalları içerirken, E(D) kümesi, ağda tanımlı bütün bağlı kanal çiftleri kümesini içerir. Bir başka deyişle, eğer (c_i, c_j) çifti bir E(D) kümesi elemanıysa, bu kanallar bir düğüme ait giriş ve çıkış kanal çiftini tanımlar. Eğer bağlılık çizgesinde hiçbir döngü yoksa yönlendirme algoritması kilitlenmeden bağımsızdır.

Dolaysız ağlarda önerilen yönlendirme algoritmaların temel işlevi kilitlenmelerden bağımsız olması gereğidir. Herhangi bir koşut sistemde, yönlendirme algoritmalarının ürettiği iletim trafiklerinin birbirlerinin veya diğerlerinin çalışmalarını engellememesi gerekir. Dolaysız ağlar için kullanılan anahtarlama yöntemleri kilitlenmelere eğilimlidir. Özellikle *wormhole* anahtarlama modeli, daha önce belirtildiği gibi, istenilen kanal kullanımda olduğu bekleme durumundaki davranışı kilitlenmeleri kolaylaştırır.

3.5.3. Yönlendirme Algoritmaları

3.5.3.1. Boyut Sıralı Yönlendirme (*Dimension Ordered Routing*)

Boyut sıralı yönlendirmede; bir paket, bir boyutta ulaşması gereken koordinat için uygun düğüme erişmeden, bir sonraki boyut üzerinde ilerlemez. Örneğin, iki boyutlu bir *torus* topolojisinde, iletilecek paket için hedef koordinat (3,5) ise, paket önce gönderenin x eksenindeki koordinatı 3 olan düğüme erişecek, daha sonra erişilen düğümün y ekseni boyunca ilerleyip koordinatı 5 olan düğüme, hedef düğüme, erişecektir. Anlaşılacağı üzere paket önce x ekseni boyunca, daha sonra y ekseni boyunca ilerleyecektir. Boyut sıralı yönlendirmede, her zaman sadece artan boyut sıralı geçişlere izin verilen tekdüze yollar üretileceği için, algoritmaların kilitlenmelerden bağımsız olacağı açıktır. Boyut sıralı yönlendirme Şekil 3-12'de gösterilmiştir.



Şekil 3-12. İki boyutu hasır doku üzerinde boyut sıralı yönlendirme (Duato, 2002)

İki boyutlu hasır doku topolojisindeki ağlarda, her düğüm (*x,y*) olmak üzere iki koordinat bilgisiyle tanımlanır. İletiler, önce x ekseni boyunca, daha sonra y ekseni boyunca ilerler. Başka bir deyişle; ileti gönderiminde sadece x ekseninden y eksenine dönüşe izin verilir. (s_x,s_y) ve (d_x, d_y) koordinatlarının sırasıyla kaynak ve hedef düğümler olduklarını düşünelim. İki düğüm arasındaki koordinat farkları sırasıylag_x = (d_x - s_x) ve g_y = (d_y - s_y) olur. (g_x, g_y) koordinatı, birinci *flit* iletisine ait başlığa eklenerek, XY yönlendirme gerçekleştirilebilir. Başka bir yaklaşım g_x birinci *flit*, g_y ikinci *flit* başlığı içinde bulunur. Paketi alan ara düğümler, ilk fazda, g_x negatif ise artırır, pozitif ise azaltır ve x ekseni boyunca paketi yönlendirir. g_x 0 olduğunda, paket x ekseni boyunca yolunu tamamlamış olur ve paket y eksenine yönlendirilir. g_y Sıfır olduğunda, hedef düğüm, paket içindeki iletiyi yerel işlemciye yönlendirir (Ni, 1993).

4. BENZETİM

Benzetim, modellenen bir olayın, nesnenin veya sistemin davranışlarının zamana bağlı olarak taklit edilmesi olarak tanımlanır. Modelleme ise bir olayın, nesnenin veya sistemin ilgilenilen özelliklerinin simgelenmesidir. Bir fiziksel nesnenin veya sistemin aşağıda verilen sebeplerden ötürü benzetimin yapılması anlamlıdır(İmre, 2010):

- Gerçeğine erişilememesi,
- Gerçeğinin kullanımının tehlikeli olması,
- Gerçeğinin kullanımının ekonomik olmaması,
- İzlenmesinin / çözümlenmesinin / yönetilmesinin zor ya da imkansız olması,
- Benzetiminin yapılarak incelenmesinin zaman kazandırması

Benzetimler; bilimsel, finansal ve ticari birçok alanda, farklı amaçlarla kullanıldığı çok geniş bir uygulama alanına sahiptirler. Uygulama alanlarında askeri, tıp, acil yardımda kullanılan eğitim sistemleri veya bilgisayar oyunları gibi kullanıcı etkileşimli örnekleri gösterilebilir. Bununla birlikte bir bitkinin yıllar süren gelişiminin daha kısa sürede gözlemlenmesi, birden fazla hava alanı arasındaki hava trafiğinin modellenmesi, bir sistem üzerindeki ağ trafiğinin modellenmesi gibi gözlem ve planlama amaçlı örnekleri de mevcuttur.

Bir bilgisayar benzetimi, fiziksel bir sistemin veya nesnenin zaman ekseninde davranışını modelleyen bir bilgisayar programı olarak tanımlanabilir. Bu programlarda bir modelle eşleşmiş program değişkenleri *durum değişkenleri* olarak tanımlanırlar ve benzetim modelini betimlerler. Benzetim programı durum değişkenlerini günleyerek, fiziksel sistemin zaman ekseninde değişiminin modellenmesini sağlar (Fujimoto, 2000). İlerleyen kısımlarda benzetim terimine karşılık olarak bilgisayar benzetimleri kullanılacaktır.

Benzetimler modellerin zaman ekseni üzerinde davranışlarını inceledikleri için, benzetim programlarındaki zaman bilgisi ve zaman yönetimi önemlidir. Benzetimler zamanı üç temel kavram üzerinden yönetirler:

- *Fiziksel Zaman:* Benzetimi yapılan fiziksel sistemin zamanına karşılık gelir.

- *Benzetim Zamanı:* Fiziksel zamanın benzetimde soyutlanmış halidir. Başka bir deyişle fiziksel sistemde geçen zamanın benzetim üzerinde modellenmiş halidir.
- *Duvar Saati Zamanı:* Benzetimin çalışması boyunca gerçek dünyada geçen zamandır.

Benzetimler zaman yönetimlerine göre üç gruba ayrılırlar:

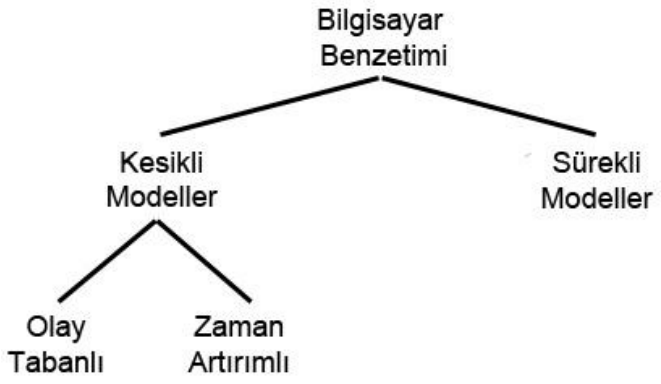
- *Gerçek Zamanlı Benzetim:* Benzetim zamanıyla duvar saati zamanının zaman uyumlu olarak işletildiği benzetimlerdir. Bu benzetimlerde gerçek dünyada geçen zaman ile benzetimde geçen zaman birbirlerine eşittir.
- *Ölçekli Gerçek Zamanlı Benzetim:* Benzetim zamanındaki ilerlemelerin, duvar saati ilerlemeleriyle ölçekli olarak yapıldığı benzetimlerdir. Bu benzetimlerde benzetim zamanı ve duvar saati zamanları arasında, aynı zaman birimlerini kullandığı varsayıldığında, Denklem (4.1) ile ifade edilen ilişki vardır.

$$Benzetim\ Zamanı = sabit * Duvar\ Saati\ Zamanı \tag{4.1}$$

- *Olabildiğince Hızlı Benzetim:* Benzetim zamanıyla duvar saati zamanı arasında bir ilişki bulunmaz.

4.1. Benzetimlerin Sınıflandırılması

Benzetimler durum değişimlerine ve zaman akış biçimlerine göre Şekil 4-1'de gösterildiği gibi sınıflandırılırlar.



Şekil 4-1. Benzetimlerin sınıflandırılması (Fujimoto, 2000)

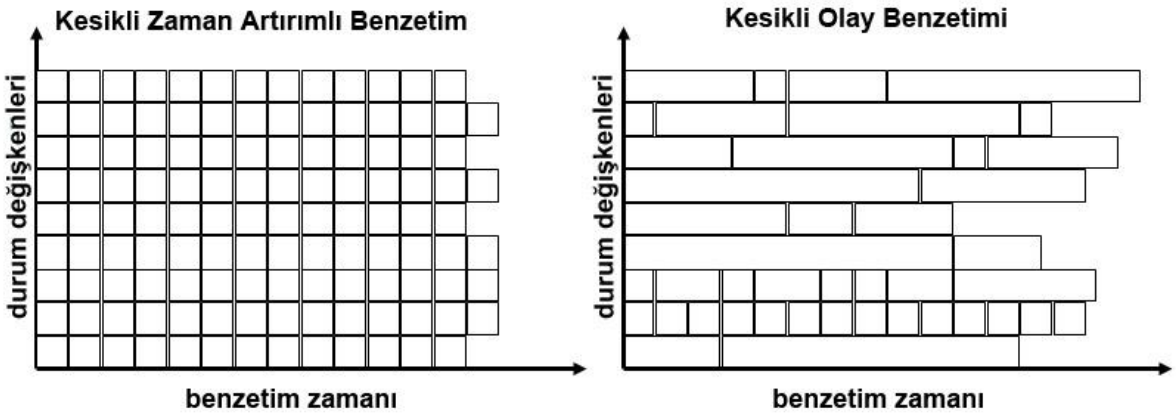
Sürekli Benzetimler, durum değişimlerinin benzetim zamanı boyunca sürekli olduğu benzetimlerdir. Bu benzetim türlerinde genellikle davranışlar diferansiyel denklemlerle tanımlanır (Fujimoto, 2005).

Kesikli Benzetimler, durum değişiminin sürekli olmadığı, kesikli benzetim anlarında olduğu benzetimlerdir. Kesikli benzetimler zamanın ilerleyiş biçemine bağlı olarak kendi içine ikiye ayrılırlar: *Zaman Artırımlı Benzetimler* ve *Kesikli Olay Benzetimleri* (Fujimoto, 2005).

Zaman Artırımlı benzetimlerde, benzetim zamanı sabit zaman artırımlarıyla ilerler. Bu benzetimlerde, benzetim sistemi çalışma zamanı boyunca benzetim zamanında periyodik zaman artırımları uygular ve her zaman artırımı sonunda sistem için bir durum hesaplar (Fujimoto, 2000).

4.1.1. Kesikli Olay Benzetimi

Benzetim için her zaman artırımı sonunda durum hesaplamak yerine, sadece ele alınması anlamlı bir olay olduğunda durum değişkenleri günlemek daha anlamlıdır. Kesikli olay benzetimlerinde, durumlar arası geçişlere sebep olan etkiler *olay* olarak tanımlanır (Fujimoto, 2000). Dolayısıyla benzetim zamanı, zaman artırımlı benzetimlerdeki gibi sabit zaman adımları biçiminde değil, durum değişimlerine sebep olacak olaylar meydana geldikçe ilerler. Kesikli benzetimlerde durum değişkenlerin değişimleri zaman ekseninde Şekil 4-2'de gösterilmiştir.



Şekil 4-2. Kesikli benzetimlerde zamana bağlı durum değişimleri (Fujimoto, 2000)

Kesikli olay benzetimlerinde durum geçişlerini betimleyen olayların hepsi bir zaman etiketi içerirler. Olayın zaman etiketi, bu olayın benzetim zamanı olarak ne zaman

oluşacağı bilgisini betimler. Örneğin bir hava trafiği benzetiminde, bir uçağın bir havaalanına ulaşması olayının zaman etiketi, bu uçağın ne zaman havaalanına ulaşacağı bilgisini gösterir.

Benzetim sistemi, oluşan olayları zaman sıralı olarak işleyerek, durum değişkenlerini günler ve benzetim zamanını ilerletir. Bu bağlamda kesikli olay benzetimleri, olay kuyruğunda zaman sıralı olarak tutulan bir dizi olayın zaman sıralı işlenmesi olarak tanımlanabilir. İşlenen olaylar durum değişkenlerinin günlenmesine sebep olabileceği gibi, yeni olayların oluşmasına da sebep olabilir. Oluşan yeni olaylar zamanı geldiğinde işlenmek üzere olay kuyruğuna atılır. Kesikli olay benzetimleri, benzetimin yaşam süreleri boyunca Çizelge 4-1'de verilen algoritmayı çalıştırırlar.

Olay kuyruğu boş olmadığı sürece

{

Zaman etiketi en küçük olan olayı olay kuyruğundan çıkar;

Benzetim zamanını bu olayın zaman etiketine eşitle;

Olayla ilgili hesaplamaları yap, durum değişkenlerini güncelle;

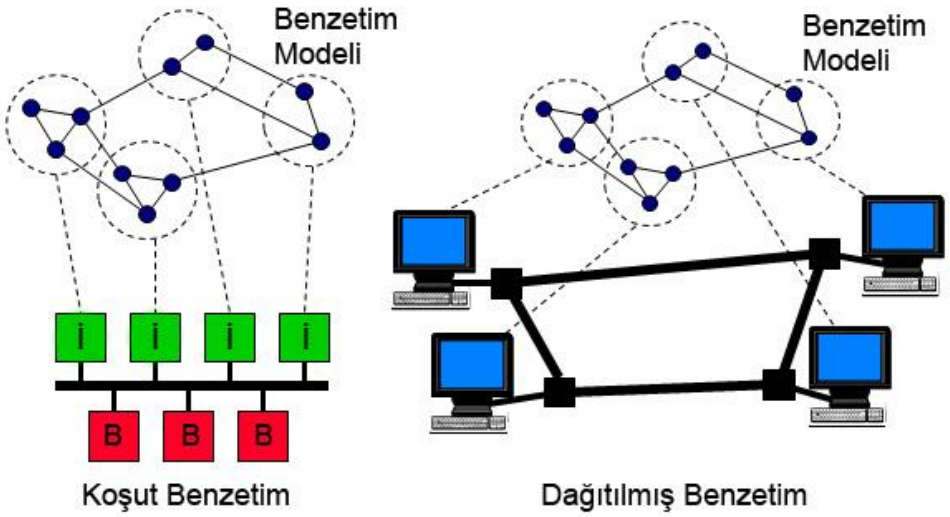
Varsa yeni olayları olay kuyruğuna ekle*;

}

* Yeni olayın zaman etiketi ≥ Benzetim Zamanı

Çizelge 4-1. Kesikli olay benzetimi olay arttırma algoritması (İmre, 2010)

4.2. Koşut ve Dağıtılmış Benzetim



Şekil 4-3. Koşut ve dağıtılmış benzetim yapıları (Fujimoto, 2005)

(İ= İşlemci B = Bellek)

Koşut benzetim, benzetim programının birden fazla işlemci içeren sistemler üzerinde gerçekleştirilmesi ve çalıştırılmasıdır. Benzetimin üzerinde çalıştırıldığı sistem çok işlemcili bir bilgisayar, iş istasyonları veya bir coğrafyada dağıtılmış bir bilgisayar kümesi olabilir. Benzetimin üzerinde çalıştırıldığı bilgisayar sistemi işlemcilerin birbirlerine çok yakın biçimde bağlanmadığı, coğrafik olarak farklı konumlarda bulunduğu durumda, benzetim *dağıtılmış benzetim* olarak tanımlanır. Koşut ve dağıtılmış benzetimlerin yapıları Şekil 4-3'te gösterilmiştir.

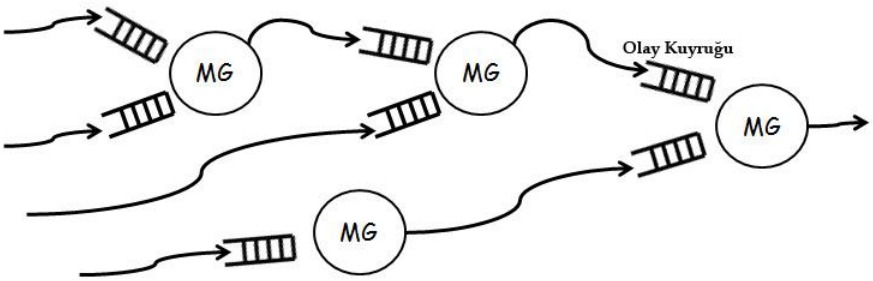
Koşut benzetimler benzetimin etkinliğinin önemli olduğu veya modellenecek sistem için gereksinimlerin (bellek ve/veya işlemci gücü) tek işlemcili bilgisayarlar tarafından karşılanamadığı durumlarda tercih edilirler. Bununla birlikte kullanıcıların veya sistem kaynakların coğrafik olarak dağıtılmış olduğu veya farklı benzetim uygulamalarının, tek bir senaryo altında çalışmasının gerektiği durumlarda; benzetimin koşut ve dağıtılmış gerçekleştirilmesi gerekir.

Koşut benzetimlerde benzetim, üzerinde gerçekleştirildiği çok işlemcili sistem üzerinde çalışan bir görev grubu tarafından işletilir. Benzetimi işleten bu görevlere *mantıksal görev* denir. Mantıksal görevler farklı işlemciler üzerinde çalıştırılırlar ve her bir görev gerçekleştirilen fiziksel modelin bir alt modelinden sorumludur. Fiziksel sistemin her bir alt modeli *fiziksel görev* olarak tanımlanır (Chandy, 1978). Örneğin

kesikli olay benzetimi biçiminde gerçekleştirilen bir hava trafiği benzetiminde, her bir mantıksal görev bir havaalanının benzetiminden sorumludur ve havaalanları arasındaki uçuşlar olaylarla benzetilirler. Koşut benzetimde her bir mantıksal görev bir fiziksel görevi modeller. Koşut benzetimlerde her bir mantıksal görevin kendi olaylar kuyruğu veya kuyrukları bulunur. Görevlere atanan benzetim modelleri arasındaki etkileşim, birbirlerine olay göndermeleriyle sağlanır. Modellerin olay alış verişinin gerçek dünyadaki karşılığı, modelleri gerçekleştiren mantıksal görevlerin birbirlerine veri alış verişi olacaktır. Veri alış verişinin nasıl olacağı benzetimin üzerinde çalıştığı fiziksel sistemin mimarisine bağlı biçimde değişiklik gösterir.

Koşut benzetimlerde gerçekleştirilen fiziksel sistem bağlamında, benzetim süresince etkileşecek alt modellerin birbirleriyle komşulukları olur. Kesikli olay benzetimi olarak gerçekleştirilen bir benzetimde, işletim süresince olay üreten alt modeller, ilgili komşu alt modellere uygun zamanlı olarak olaylar gönderirler.

Kurulan model bağlamında, birbirleriyle etkileşerek birbirlerine olay üreten görevler *komşu mantıksal görevler* olarak tanımlanırlar. Her mantıksal görev kendi ürettiği olaylarla birlikte komşu mantıksal görevlerden gelecek olayları, zaman sıralı olarak işlemekten sorumludur. Mantıksal görevin komşu görevlerden gelen ve dâhili olarak oluşan olaylarının bütününü tek bir kuyruk üzerinden yönetmesi olanaksızdır. Çünkü olay kuyruğundan seçtiği herhangi bir olayı işlemeden önce, komşularından daha erken bir olay gelmeyeceğine emin olmalıdır. Dolayısıyla mantıksal görev bütün komşularından en az bir olay almadan en küçük zamanlı olayın seçilip işlenmesine karar veremez. Bu soruna çözüm olarak mantıksal görevler her komşusu için ayrı bir olay kuyruğu tutar ve mantıksal görev olay kuyruklarından seçtiği olayı işler. Bahsedilen benzetim yapısı Şekil 4-4'te gösterilmiştir.



Şekil 4-4. Koşut/dağıtılmış benzetim yapısı (İmre, 2010)

(MG=Mantıksal Görev)

4.2.1. Koşut Benzetimlerde Zaman Uyumlama

Bir benzetim programı, seri program biçiminde gerçekleştirilip çalıştırıldığında, gerçekleştirim tek bir görev tarafından yürütüleceği için, tek bir olay kuyruğunda görevleri zaman etiketi sıralı olarak tutarak, olayları kuyruktaki sırasıyla işleyebilir. Fakat koşut benzetim gerçekleştirimlerinde benzetim bir mantıksal görev grubuna eşlendiği için, mantıksal görevler arası zaman uyumlamayı sağlayacak yöntemlere ihtiyaç duyulur. Söz konusu zaman uyumlama yöntemlerinin, bütün mantıksal görevlerde olayların zaman sıralı olarak işlenmesini sağlaması gerekmez. Zaman uyumlama yönteminin amacı, bir benzetimin koşut ve seri gerçekleştirimlerinin aynı sonuçları üretmesini sağlamaktır (Fujimoto, 2000).

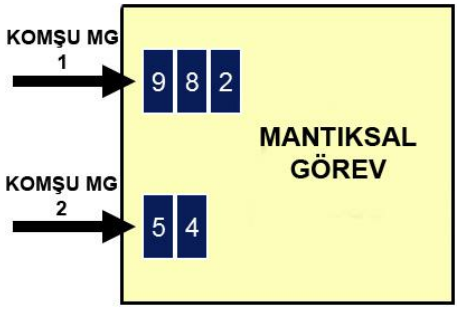
Mantıksal görevler, olay kuyrukları arasından seçip işledikleri olaylar arasında zaman uyumluluğunu sağlarken, olay kuyruklarına olay gönderen komşu mantıksal görevler de olayları zaman sıralı göndermek zorundadırlar. Dolayısıyla mantıksal görevlerin birbirlerine geçmiş zamanlı olay üretmemeleri ve her mantıksal görevin olayları zaman sıralı olarak işlemeleri gerekir. Bu kural koşut ve dağıtık benzetimlerin temel kurallarından biridir ve *yerel nedensellik kısıtlaması (local causality constraint)* olarak tanımlanır (Fujimoto, 2000). Yerel nedensellik kısıtlaması, mantıksal görevlerin kendi içlerinde zaman sırası bütünlüğünün sağlanmasını sağlar. Bununla birlikte, mantıksal görevlere dışarıdan veya birbirleri tarafından müdahale edilmemesi gerekir. Bu sebeple mantıksal görevle eşlenmiş benzetim alt modellerin durumlarını betimleyen durum değişkenlerinin, mantıksal görevler arasında paylaşılmaması gerekir (Fujimoto, 2005, Chandy, 1978).

Yerel nedensellik kısıtlaması, benzetimin seri ve koşut gerçekleştirimlerinin aynı sonuçları üretmesini sağlar. Bununla birlikte benzetimin anlamlı sonuçları üretmesini garanti edeceği anlamına gelmez. Benzetim sonuçlarının anlamlı olması, benzetim uygulamasının algoritmasına bağlı bir durumdur.

Bahsedilen koşut/dağıtılmış benzetim özelliklerine göre, Çizelge 4-1'de verilen kesikli olay yönetim algoritması, Çizelge 4-2'deki biçimde günlenir. Bu algoritma her bir mantıksal görev tarafından uygulanır.

```
Benzetim çalıştığı sürece
{
    Olay kuyruklarımın hepsi en az bir öğe içerene kadar bekle;
    Zaman etiketi en küçük olan olayı kuyruğundan çıkar;
    Benzetim zamanımı bu olayın zaman etiketine eşitle;
    Olayla ilgili hesaplamaları yap, durum değişkenlerini güncelle;
    Varsa yeni olayları alıcılarına birer iletiyle gönder*;
}
* Yeni olayın zaman etiketi ≥ Benzetim Zamanım
```

Çizelge 4-2. Zaman uyumlu koşut kesikli olay benzetimi için olay yönetim algoritması (İmre, 2010)



Şekil 4-5. Koşut benzetimlerde zaman sıralı olay işletimi örneği (Fujimoto, 2000)

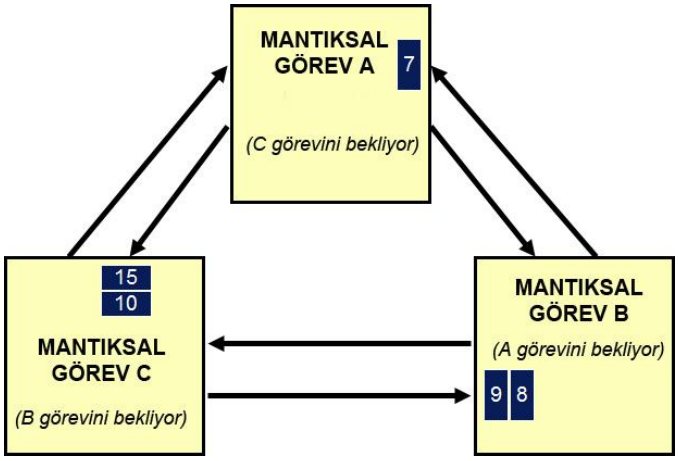
Şekil 4-5'te bahsedilen mantıksal görev yapısı örneklenmiştir. Bu mantıksal görevin iki komşu mantıksal görevi vardır ve her komşu mantıksal görev bir olay kuyruğuyla eşleşmiştir. Olay kuyruklarında belirtilen numaralar, kuyruklardaki olaylara ait zaman etiketleridir.

Çalışma anında mantıksal görev önce 1. komşu mantıksal görevden (MG) gelen ve zaman etiketi 2 olan olayı işler. Daha sonra 2.komşu MG kuyruğundan, sırasıyla 4 ve

5 zaman etiketli olayları işler. Mantıksal görev 1.komşu MG kuyruğundaki 8 etiketli olayın işlenmesine geçmeden önce, 2.komşu MG'den bir olay gelmesini beklemelidir. Çünkü gelecek olay 8 zaman etiketli olaydan daha küçük zaman etiketine sahip olabilir. 2.komşu MG'den bir olay geldikten sonra, mantıksal görev işletimine devam eder.

4.2.2. Koşut Benzetimlerde Karşılıklı Kilitlenme

Mantıksal görevler, yerel olay kuyrukları dışında bütün kuyruklarında en az bir olay bulunmadan, bir sonraki olayı seçip işleyemezler. Dolayısıyla kuyruklarından en az biri boş olduğunda, mantıksal görev komşu mantıksal görevlerinin hepsinden en az bir olay gönderimi almadan işletime devam edemezler.



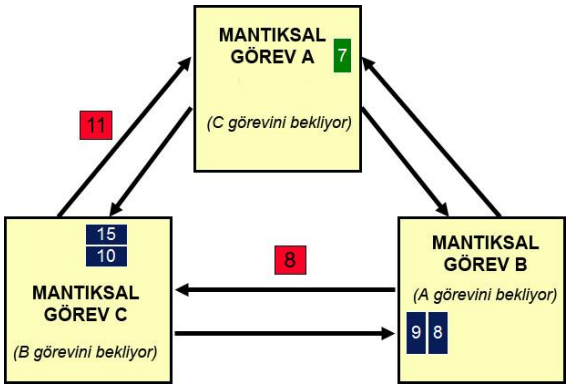
Şekil 4-6. Koşut benzetimlerde Karşılıklı Kilitlenme örneği (Fujimoto, 2000)

Şekil 4-6'da, her mantıksal görevin Çizelge 4-2'de verilen algoritmayı işlettiği varsayılan ve her birinin diğer iki mantıksal görevle komşuluğunun bulunduğu üç mantıksal görev gösterilmiştir. Verilen örnekte mantıksal görevler işletimlerine devam etmek için birbirlerini bekledikleri için, üç mantıksal görev de sonsuz bir bekleme konumunda kalır ve benzetimin ilerlemesi olanaksızdır. Bu durum *karşılıklı kilitlenme* olarak tanımlanır (Fujimoto, 2000). Karşılıklı kilitlenme koşut yazılımların hepsinde karşılaşılan bir durumdur. Koşut benzetimlerde karşılıklı kilitlenmelerin engellenmesi ve benzetimin çalışmaya devam etmesi için birtakım yöntemler geliştirilmiştir. Bir sonraki bölümde bu yöntemlerden biri olan *"Boş ileti algoritması"* anlatılacaktır.

4.2.2.1. Boş İleti Algoritması

Chandy/Misra/Bryant Boş İleti Algoritmasında (Chandy, 1978), mantıksal görevler karşılıklı kilitlenmeleri engellemek için, komşularına sadece ilgili olay üretildiğinde değil, benzetim zamanlarını her ilerlettiklerinde olay gönderirler. Mantıksal görevler benzetim zamanlarını ilerlettiklerinde, o zamanla ilgili göndereceği olayı olmadığı komşularına, zaman etiketli boş bir olay gönderirler. Bu olay, bekleyen mantıksal görevin, zamanını gönderilen boş olay zamanına kadar ilerletebilmesini sağlar. Zaman uyumlama sağlayan boş olaylara *boş ileti* denir (Fujimoto, 2000).

Mantıksal görevlerin kendi zamanlarında boş iletiler göndermeleri, mantıksal görevlerde benzetim zamanlarının ilerleyememesine ve benzetimin kilitlenmesine sebep olabilir. Örneğin mantıksal görevlerin benzetim uygulama bölümlerinin olay üretmediği bir durumda, bütün mantıksal görevler 0 zaman etiketli boş iletiler gönderirler. Bu durumda benzetim zamanının ilerlemesi olanaksızdır. Boş iletilerin benzetim zamanını ilerletmesini sağlamak için, benzetimlerde olay oluşma eşik değeri ($t_{eşik}$) belirlenir. Bu değer, bir mantıksal görevin t benzetim anında bir komşusuna ileteceği bir olayı olmadığında, komşusunun zamanını ($t+t_{eşik)}$ benzetim anına kadar ilerletebileceği anlamına gelir. Örneğin bir tren benzetiminde, A istasyonundan B istasyonuna bir trenin minimum yolculuk süresinin 5 birim benzetim zamanı olduğunu varsayalım. A istasyonundan t benzetim anında B istasyonuna doğru yola çıkacak bir tren olmadığı koşulda, B istasyonuna A istasyonundan $t+5$ anına kadar ulaşacak bir tren yoktur. Bu durumda A istasyonunu modelleyen mantıksal görev, B istasyonunu modelleyen mantıksal göreve t benzetim anında $(t+5)$ zaman etiketli bir boş ileti yollar ve mantıksal görevin zamanının ilerlemesini sağlamış olur. Belirtilen $(t_{eşik)}$ değeri, boş ileti algoritmasında *ilerletme değeri (look ahead value)* olarak tanımlanır (Fujimoto, 2000).



Şekil 4-7. Koşut Benzetimlerde boş ileti algoritması örneği (Fujimoto, 2000)

Şekil 4-7'de, boş ileti algoritmasının uygulanması örneklenmiştir. Verilen örnekte ilerletme değerinin 3 birim benzetim zamanı olduğunu varsayalım. Çalışma anında B görevinin benzetim zamanının 5 olduğu anı düşünelim. B görevi, C görevine gönderecek olayı olmadığı için, bu göreve 8 zaman etiketli bir boş ileti yollar. Bu iletiyi alan C görevi, benzetim zamanını 8 zamanına kadar ilerletir ve benzer şekilde A görevine 11 zaman etiketli bir boş ileti yollar. A görevi de tüm kuyrukları dolu olduğu için 7 zaman etiketli görevi işleyebilir. Böylece kilitlenme çözülmüş olur.

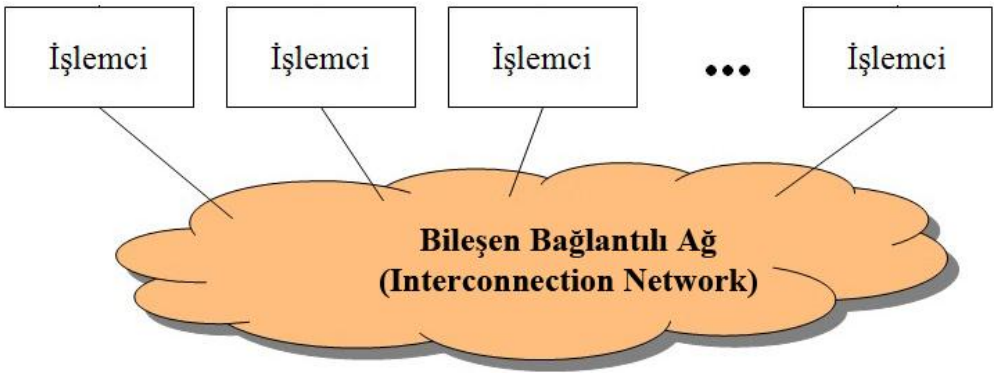
Çizelge 4-2'de verilen koşut benzetim algoritmasının boş ileti algoritmasıyla zaman uyumu sağlanmış biçimi Çizelge 4-3'te verilmiştir.

```
Benzetim çalıştığı sürece {

    Olay kuyruklarımın hepsi en az bir öğe içerene kadar bekle;

    Zaman etiketi en küçük olan olayı kuyruğundan çıkar;

    Benzetim zamanımı bu olayın zaman etiketine eşitle;

    Olayla ilgili hesaplamaları yap, durum değişkenlerini güncelle;

    Varsa yeni olayları alıcılarına birer iletiyle gönder*;

     Komşu olan her alıcıya zamanı (benzetim zamanım+ ilerletme zamanı) olan boş bir ileti gönder.

}

* Yeni olayın zaman etiketi ≥ Benzetim Zamanım
```

Çizelge 4-3. Zaman uyumlu koşut benzetim olay yönetim algoritması (İmre, 2010)

5. GERÇEKLEŞTİRİM

Tez kapsamında, çok işlemcili bir ağ mimarisinin benzetimi gerçekleştirilmiştir. Benzetim, koşut benzetim olarak gerçekleştirilmiştir. Benzetimde birden fazla mantıksal görev eş zamanlı çalışmaktadır. Gerçekleştirilen koşut benzetimde, mantıksal görevler arasında zaman uyumlama sağlamak için boş ileti algoritması kullanılmaktadır.



Şekil 5-1. Benzetim mimarisi

Benzetimi yapılan ağ, dağıtılmış çok işlemcili bir ağdır ve birimler bağlı oldukları bileşen bağlantılı ağ üzerinden birbirlerine ileti göndererek iletişim sağlamaktadırlar. Benzetimi yapılan mimari Şekil 5-1'de örneklenmiştir.

Benzetimi yapılan ağdaki işlemciler birbirlerine *torus* topolojisiyle bağlıdırlar. Gerçekleştirilen *torus* topolojisi iki boyutlu bir *torus* topolojisidir. Benzetimdeki işlemci(düğüm) sayısı ve *torus* topolojisinin boyutları kullanıcıdan benzetim başlangıcında parametre olarak alınmaktadır.

Benzetim ağındaki işlemciler birbirlerine dolaysız ağ mimarisinde bağlanmışlardır. Dolaysız ağlar bağlamında her işlemcinin kendisiyle eşleşmiş olduğu bir yönlendiricisi bulunmaktadır. Gerçekleştirimde işlemciler ve yönlendiriciler modellenmiş, yönlendiricilerin bileşen bağlantılı ağ içinde komşulukları sağlanmıştır. Modellenen işlemcilerin birbirlerine ileti göndermesi, işlemcilerin bağlı oldukları yönlendiriciler boyunca iletilerek hedef işlemciye ulaşmasıyla olmaktadır. Yönlendiriciler arasında çift yönlü (*duplex*) kanallar ile bağlantılar kurulmuştur.

Benzetimi yapılan ağ, *wormhole* anahtarlama modelini kullanmaktadır. Ağ içindeki ileti birimi *flit* iletileridir. Düğümlerin birbirlerine gönderdikleri iletiler, *flit* iletilerine bölünerek *wormhole* anahtarlama yöntemiyle iletilirler.

Benzetimi yapılan ağ içinde kredi tabanlı akış kontrolü kullanılmıştır. Çok işlemcili mimarilerde, yastık boylarının küçük olduğu durumlarda kredi tabanlı akış kontrolü yöntemi tercih edilmektedir. Tez kapsamında küçük boyutta ve kolay gerçekleştirilebilir yönlendiricilerin amaçlanması sebebiyle, kredi tabanlı akış kontrol yöntemi tercih edilmiştir.

Gerçekleştirilen benzetimde, durağan ve kaynak kararlı yönlendirme algoritmaları ile toplu iletişim yöntemlerinden bazıları gerçekleştirilmiş, hedefe göre kullanıcının istediği yönlendirme algoritmasını seçmesi amaçlanmıştır. Bununla birlikte, diğer bileşenler gibi, kullanıcının kendi yönlendirme algoritmasını da gerçekleştirmesine olanak sağlanmıştır. Yönlendirme algoritması benzetim çalışmasının başında kullanıcı tarafından seçilmektedir.

Benzetimde işlemcilerin modellenmesi basit düzeyde olmuş, seçilen iletişim örüntüsünün testlerinde kullanılmıştır. Bununla birlikte kullanıcının daha karmaşık işlemciler tasarlayıp benzetimi bu işlemci yapısıyla çalıştırmasına da olanak verilmiştir.

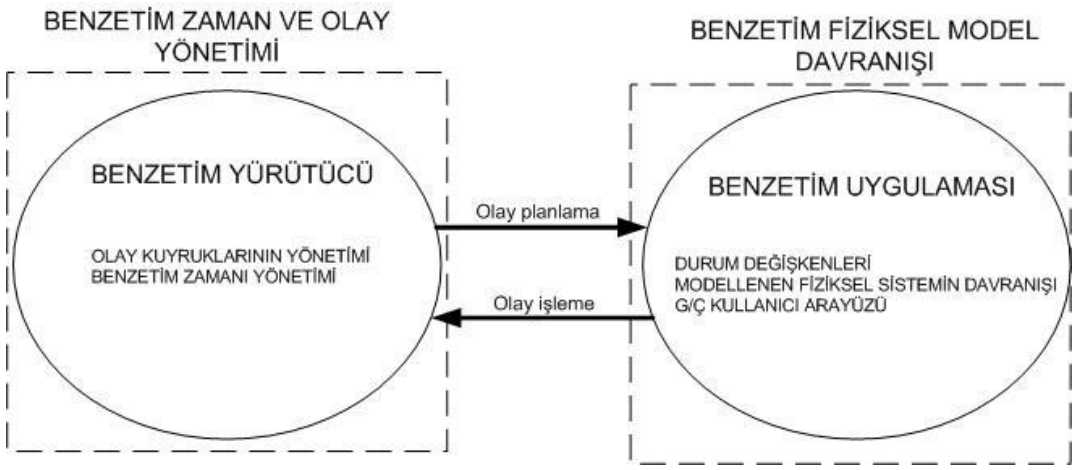
Benzetim gerçekleştirimi Java programlama dilinde yapılmıştır (Schildt, 2002). Benzetim gerçekleştiriminde java dilinin koşut bileşenleri (Goetz, 2006, Oracle, 2011a) yoğun biçimde kullanılmıştır. Gerçekleştirim tek bir uygulama biçiminde tasarlanmıştır. Koşut benzetim içindeki mantıksal görevler, java İşletim dizileriyle (Schildt, 2002, Oracle, 2011b) gerçekleştirilmiştir. Benzetim gerçekleştirimi nesneye yönelik olarak tasarlanmış ve gerçekleştirilmiştir. İlerleyen kısımlarda getirilen nesneye yönelik çözümler detaylandırılacaktır.

Gerçekleştirim, benzetim altyapısı ve ağ altyapısı olmak üzere iki bölüme ayrılmıştır. Tez çalışması olarak ilk önce benzetim altyapısı gerçekleştirilmiş ve ilgili testler ile doğruluğu gösterilmiştir. Daha sonra bu benzetim altyapısı üzerinde ağ benzetimi kurulmuş ve çalıştırılmıştır.

5.1. Benzetim Altyapısının Gerçekleştirimi

Benzetim altyapısı olarak Çizelge 4-3'teki algoritma gerçekleştirilmiştir. Bu algoritmaya bakıldığında, mantıksal görevlerde, seçilen olayın işlenmesi kısmı dışında yapılan işlemlerin aynı olduğu görülür. Bu bağlamda benzetim alt modellerini betimleyen mantıksal görevlerin iki bölüm halinde ele alınabilir: *Benzetim Uygulaması* ve *Benzetim Yürütücü.*

Benzetim yürütücüsü, çalışma süresi boyunca yerel olaylar ve komşularından gelen olayların zaman etiketine bağıl olarak seçilmesinden, mantıksal görevler arası olayların iletilmesinden ve mantıksal görevin zamanının yönetilmesinden sorumludur. Bu birim mantıksal görevlerin temel yapısını oluşturur. Benzetim uygulaması ise, modellenen fiziksel sistemi gerçekleştirir. Benzetimin uygulama kısmı benzetimin durum değişkenlerinden, olayların işlenmesinden ve yeni olayların üretilmesinden sorumludur. Bu birim olay ürettiğinde benzetim yürütücü birimine olayları zamanı gelince iletilmesi için planlar (iletir). Benzetim yürütücü de, seçtiği olayları işlemesi için benzetim uygulamasına iletir. Önerilen benzetim altyapısı tasarımı Şekil 5-2'de gösterilmiştir.



Şekil 5-2. Gerçekleştirilen benzetim altyapısı mimarisi

Önerilen benzetim altyapısı tasarımında, mantıksal görevler temel olarak benzetim yürütücü işlevlerini yürütmekte, benzetim yürütücünün seçtiği olayları işlemek üzere kendilerine bağlı benzetim uygulaması nesnesine iletmektedir. Mantıksal görevin yüklendiği modelin zaman yönetiminden, benzetim yürütücü nesne sorumludur. Bu bağlamda her mantıksal görev bir benzetim yürütücü nesne ile eşleşmiştir.

```
Benzetim_Yürütücü

{

    Olay kuyruklarımın hepsi en az bir öğe içerene kadar bekle;

en_kucuk_olay = en_kucuk_zamanlı_olayi_sec(olay_kuyruklari)

    benzetim_zaman = en_kucuk_olay.zaman

    Olay zamanı = benzetim_zamani bütün olaylar için tekrar et{

        Benzetim_Uygulama.olayi_isle(en_kucuk_olay)

        En_kucuk_olay = eş_zamanli_olay_sec(en_kucuk_olay,
benzetim_zamani)

}

    Zaman uyumlama için boş ileti yolla

}
```

* Yeni olayın zaman etiketi ≥ Benzetim Zamanım

Çizelge 5-1. Benzetim yürütücü tarafından gerçekleştirilen zaman uyumlu olay yönetim algoritması

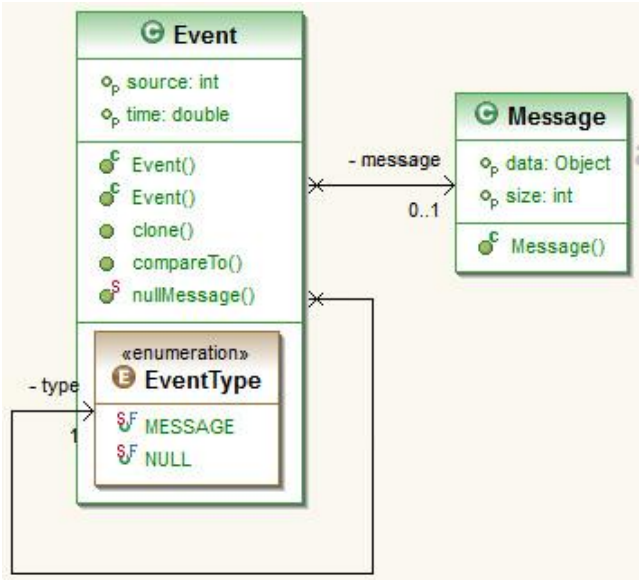
Benzetim uygulaması tasarım örüntülerinden Strateji örüntüsünü gerçekleştirir (Gamma, 1994). Benzetim yürütücü, benzetim uygulaması detaylarını bilmeden, benzetim uygulamasının olayları işleme davranışını çağırır ve benzetim uygulamasının ürettiği olayları zaman sıralı olarak iletir.

Benzetim altyapısı *olay ve olay yönetimi, benzetim görevleri, benzetim uygulamaları* ve *benzetim ağı* bileşenlerinden oluşmaktadır. Bununla birlikte topoloji ve benzetim günlük oluşturma birimleri benzetim altyapısında kullanılmaktadır. İlerleyen kısımda bu bileşenler ve gerçekleştirim yaklaşımları anlatılacaktır.

5.1.1. Olay ve Olay Yönetimi

Olay (Event sınıfı), benzetimdeki temel iletişim birimidir. Benzetim içindeki bütün görev ve birimler, birbirleriyle olaylar üzerinden iletişim kurmaktadırlar. Olaylar zaman, olay türü, gönderen bilgisi ve ileti niteliklerine sahiptirler. Zaman niteliği olayın ne zaman oluşacağı bilgisini, olay türü ise olayın boş veya ileti içerikli olduğu bilgisini tutar. İleti (*Message sınıfı*), olayın gönderildiği birime iletilmek istenilen bilgiyi içerir. İleti nesneleri ileti boyu ve ileti verisi bilgilerini içerirler. İleti sınıflarının, fiziksel benzetimdeki iletime göre genişletilmesi beklenir.

Olay ve ileti sınıflarının UML çizeneği Şekil 5-3'de gösterilmiştir.



Şekil 5-3. Olay ve İleti sınıflarının UML çizeneği

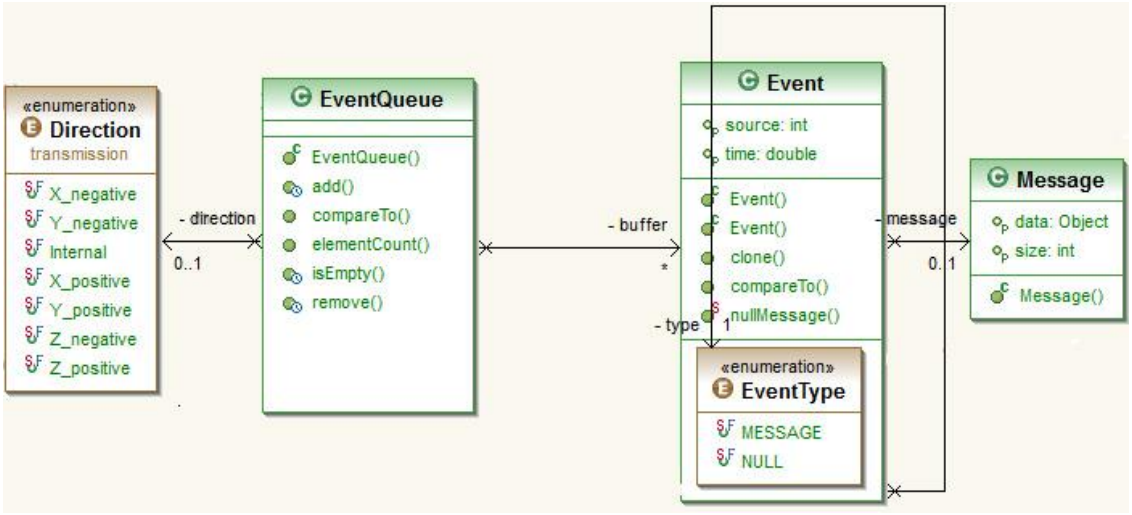
Olay sınıfı Java *Comparable* arayüzünü (Schildt, 2002) gerçekleştirerek, olayların zaman etiketlerine göre karşılaştırılabilmelerini sağlar. Bu yaklaşım olay nesnelerinin zaman etiketlerine göre sıralanabilmelerini kolaylaştırır.

Benzetim yürütücülerinin olay kuyrukları, olay kuyruğu sınıflarında (*EventQueue sınıfı*) gerçekleştirilmişlerdir. Olay kuyrukları, sakladıkları olaylarla birlikte kuyruktaki ilk olayın, hangi yöndeki komşuya ait oldukları bilgisini ve kuyruk boş ise kuyruktan çıkan son olayın zaman bilgilerini tutarlar. Son olay bilgisi olay kuyruklarının en küçük zamanlı olayını seçerken kullanılır. Gerçekleştirilen sınıfta, kuyruklar eklenen olayları azalmayan zaman sıralı olarak tutarlar. Bu özellik, olay kuyruklarının tek kuyruk içeren seri benzetim gerçekleştirimlerinde de kullanılmasına imkan sağlar.

Bir mantıksal görevin bir komşusuna olay planlaması, gönderdiği komşusunun kendisine atadığı olay kuyruğuna yeni bir olay eklemesine karşılık gelir.

Dolayısıyla olay kuyrukları, koşut benzetimlerdeki görevler arasında paylaşılan kaynak durumundadırlar. Olay kuyruklarının bütünlüğünü korumak için, olay kuyruklarındaki işlemlerinin bütünü, monitör kavramının java programlama dilindeki gerçekleştirimi olan zaman uyumlu metotlar (*synchronized methods*) (Schildt, 2002, Oracle, 2011a) ile gerçekleştirilmiştir. Bu sayede aynı anda herhangi iki işletim dizisinin, bir olay kuyruğu nesnesi üzerinde işlem yapması engellenmiştir.

Olay kuyruğunun UML çizeneği Şekil 5-4'te gösterilmiştir.



Şekil 5-4. Olay kuyruğu sınıfının UML çizeneği

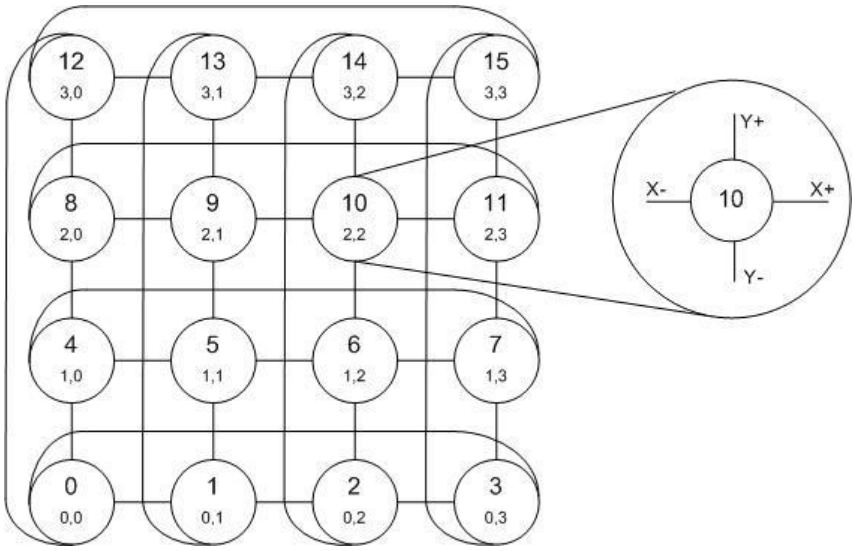
5.1.2. Yön, Koordinat ve Topoloji

Benzetimlerde mantıksal görevlerin betimlediği alt modellere göre, mantıksal görevler arasında komşuluklar olur. Bu komşulukların nasıl olacağı seçilen topoloji ile belirlenir. Bir ağ topolojisindeki her düğümün ağ içindeki konumu iki nitelikle belirlenir: *ağ koordinatı* ve *ağ kimliği.*

Ağ koordinatı, ağ topolojisindeki düğümün, topolojideki konumunu adresler. Her düğüm, topolojinin her boyutu için bir koordinat bilgisine sahiptir. Örneğin; iki boyutlu bir topolojide her düğümün, x ve y, üç boyutlu bir topolojide x, y ve z koordinatları bulunur. Düğümlerin x koordinatları satır ofsetlerini, y koordinatları sütun ofsetlerini

verirler. Bilgisayar benzetimlerinde koordinatlara sıfırdan başlanarak değer atamaları yapılır.

Ağ kimliği, düğüme ağ içinde verilmiş bir numaradır ve bu numara ağ içinde biriciktir. Ağ kimliği numaraları, düğümlere en düşük koordinattaki düğümden başlanarak koordinat sıralı olarak ilerleyen sırada verilir. Bu atama yöntemi, dikey topolojilerde ağ kimliği ve ağ koordinatları arasındaki çevrimlere imkan sağlar. İki boyutlu 4x4 bir *torus* topolojisi üzerindeki düğümlerin koordinatları ve ağ kimlikleri Şekil 5-5'te örneklenmiştir.



Şekil 5-5. 4x4 iki boyutlu *torus* üzerinde koordinat ve ağ kimlikleri

Gerçekleştirilen benzetim altyapısında, mantıksal görevlere belirlenen komşulukların durağan olduğu ve çalışma zamanında değişmediği varsayılmıştır. Bu yaklaşımla benzetim ağı seçilen topolojiye göre mantıksal görevlerin yaratılması işleminden sonra komşulukları belirler.

Topoloji içindeki düğümler, her komşusunu iki nitelik kullanarak belirler: *Yön* ve *ağ kimliği.* Yön bilgisi, mantıksal görevin komşusunun, düğümler koordinat düzlemine yerleştirildiklerinde hangi koordinat düzeyinde ve hangi koordinat konumunda bulunduğunu belirler. Gerçekleştirimde her koordinat düzeyi (x, y ve z koordinat düzeyleri) pozitif ve negatif olmak üzere iki ayrı bölgeye ayrılmıştır. Her koordinat düzeyinde koordinatın arttığı yön pozitif, azaldığı yön negatif yön olarak seçilmiştir. Şekil 5-5'teki 10 ağ kimlikli düğümün, x pozitif yönündeki komşusu 11, y negatif yönündeki komşusu 6 ağ kimlikli düğümlerdir.

Benzetim ağı sınıfı (*Bölüm 5.1.5*) mantıksal görevlerin komşuluklarını belirlerken, Topoloji Kurucu arayüzünün (*TopologyBuilder arayüzü*) davranışlarını kullanır. Topoloji Kurucu arayüzünü gerçekleştiren sınıflar, gerçekleştirdiği topolojiye göre mantıksal görevin komşularının belirlenmesi yeteneğini sağlar. Topoloji kurucu nesneler, ayrıca çalışma zamanında ağ kimliğini parametre olarak aldığı düğümün belirtilen yöndeki komşusunun ağ kimliğini bulma davranışını da kapsar. Böylece mantıksal görevlerin komşu adreslerini tutması ve bu adreslere göre işlem yapmasına gerek kalmaz. Mantıksal görev dışındaki birimlerin de topoloji kurucuları kullanmalarını sağlamak için, alt düzeyde komşu kimliklerini de belirlemektedir.

Topolojiye bağıl adres ve konum bilgilerinin bütünü topoloji kurucu arayüz davranışlarıyla sağlanarak, mantıksal görev ve benzetim uygulamalarının çalışmalarının olabildiğince topolojik yapıdan soyutlanması amaçlanmıştır.

Benzetim sistemindeki topoloji davranışları, topoloji kurucu arayüzde toplanmıştır. Diğer topolojilerin gerçekleştirilmesi için, benzetim programcısının seçtiği topolojiyi topoloji kurucu nesne sınıfı davranışlarıyla gerçekleştirmesi gerekir. Topoloji kurucu arayüzde, topoloji davranışları iki temel metotla sağlanır: *getNeighbour()* ve *setNeighbours(). getNeighbour() metodu,* ağ kimliği verilen bir görev için, parametre yöndeki komşu görevin ağ kimliğini hesaplama davranışını gerçekleştirir. Bu hesaplama topolojiye bağlıdır. *setNeighbour()* metodu ise parametre olarak gönderilen mantıksal görev için komşuları belirleme davranışını gerçekleştirir. Topoloji gerçekleştiriminin örneklenmesi için, tez kapsamında gerçekleştirilen iki boyutlu torus topolojisi kurucu sınıfı (Ek-8)'de anlatılmıştır.

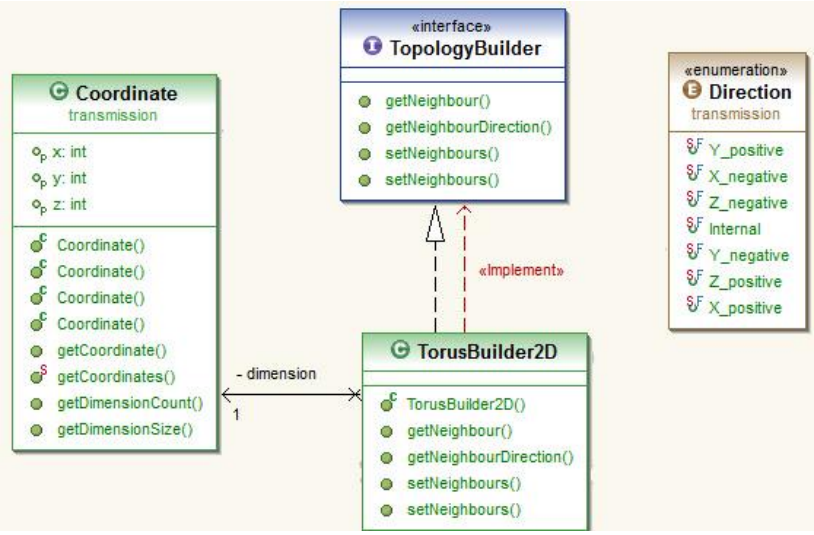
Tez kapsamında sadece iki boyutlu *torus* topolojisi gerçekleştirilmiştir. Bu amaçla Topoloji Kurucu arayüz, *torus* özelliklerine göre gerçekleştirilmiştir. *Torus* topolojisinde komşuluklar belirlenirken mantıksal görevlere oluşturulma sırasında atanan ağ kimlikleri kullanılmıştır. Görevlerin koordinatları ve komşulukları, kendilerine atanan ağ kimliklerine göre belirlenmiştir. *Torus* topolojisinin desteklediği sarmal kanalları (*wraparound channels*) kullanarak her koordinat düzeyindeki en küçük ve en büyük koordinatlı düğümler de komşu olarak bağlanmışlardır. Bu sayede sıkı dikey topolojiler kapsamında, bütün düğümlerin her yönde bir komşuya sahip olmaları sağlanmıştır.

Torus topolojisinde komşulukların belirlenmesinde kullanılan algoritması Çizelge 5-2'de verilmiştir.

```
Komsu_Belirle(Yön y, kimlik) {

koordinat = koordinat_belirle(kimlik, boyut);

X_boyut = boyut.X();

Y_boyut = boyut.Y();



komsu_kimlik = 0;

eğer( yon = X_negatif )

komsu_kimlik := (koordinat.X() = 0 ? (kimlik+(X_boyut-1)) : kimlik-1);



eğer( yon = X_pozitif )

komsu_kimlik := (koordinat.X() = (x_boyut-1) ? (id-(X_boyut-1)) : id+1);



eğer( yon =  Y_negatif)

komsu_kimlik = (koordinat.getY() = 0 ?(kimlik+(Y_boyut-1)*X_boyut):kimlik-X_boyut



eğer( yon = Y_pozitif )

komsu_kimlik = (koordinat.getY() == (Y_boyut-1) ? (id-(Y_boyut-1)*X_boyut):
kimlik+X_boyut);



       return komsu_kimlik;

}
```

Çizelge 5-2. İki boyutlu *torus* için komşu belirleme algoritması

Topoloji Kurucu arayüzü ve İki boyutlu *Torus* Kurucu (*TorusBuilder2D*) sınıflar için UML çizenekleri Şekil 5-6'da gösterilmiştir.



Şekil 5-6. Topoloji Kurucu arayüz ve *Torus* Kurucu sınıf UML çizenekleri

5.1.3. Benzetim Görevleri

Benzetim görevleri, benzetimin çalışması süresince koşut olarak çalışan mantıksal görevlerdir. Mantıksal görevler benzetim içinde fiziksel sisteme ait alt modelleri gerçekleştirirler.

Tez kapsamında gerçekleştirilen benzetim altyapısında, koşut olarak çalışması gereken görevlerin bütünü mantıksal görevler olarak gerçekleştirilmiştir. Her bir mantıksal görev, java programlama dilindeki *Runnable* arayüzünü (Schildt, 2002) gerçekleştirmiş, her bir mantıksal görev bir java işletim dizisiyle eşleştirilmiştir. Böylece çalışma zamanında bütün mantıksal görevler koşut olarak işletilmişlerdir. Gerçekleştirimde iki benzetim görevi tanımlanmıştır: *Benzetim Görevi* ve *Mantıksal Görev*.

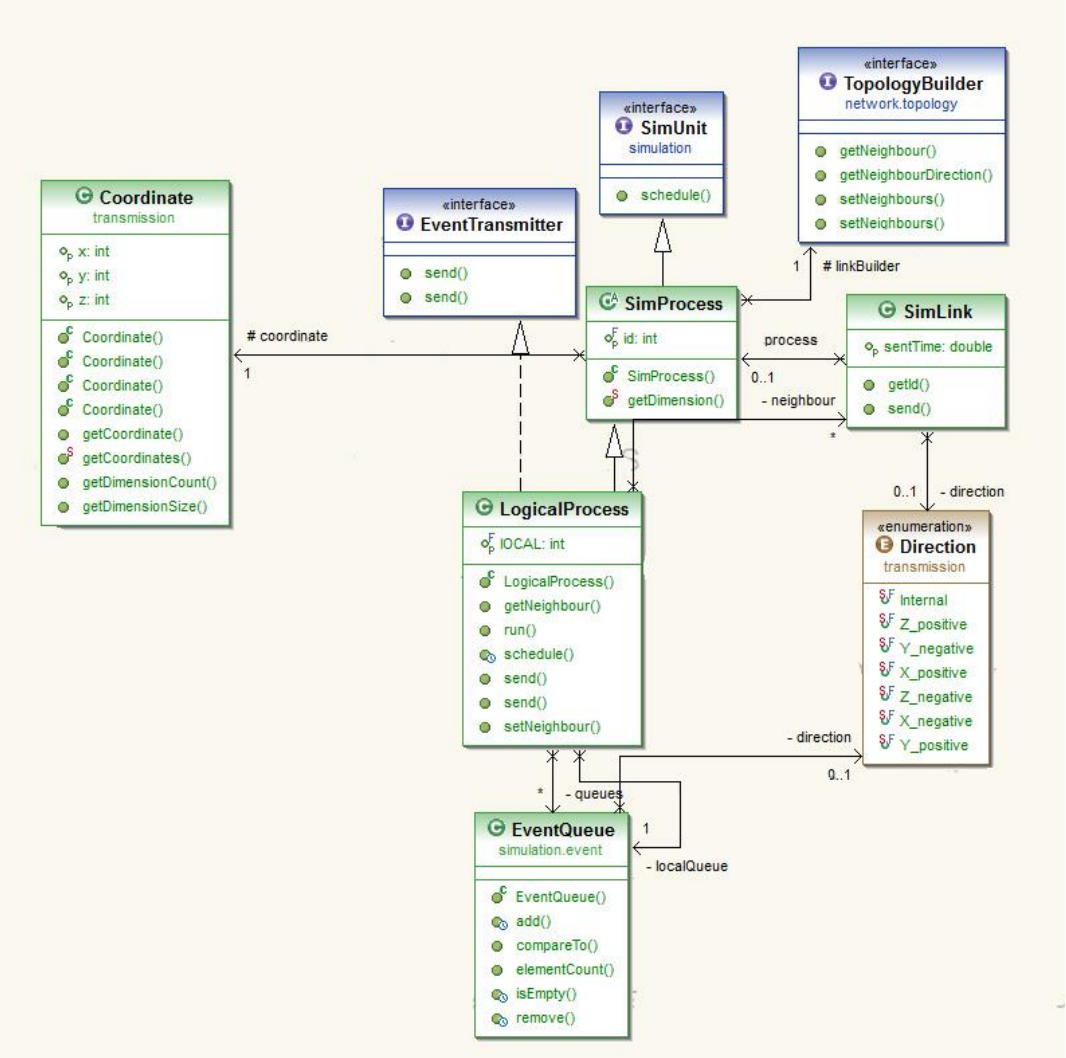
Benzetim görevleri (*SimProcess sınıfı*), benzetim içinde bulunan görevlerin genel yapısını tanımlayan ve temel niteliklerini içeren soyut bir sınıftır. Benzetim görevlerinin bütünü bir benzetim ağı içinde yönetilmişlerdir ve bütün benzetim görevlerinin ağ içindeki konum bilgileri, ağ kimlikleri ve ağ topoloji boyut bilgileri nitelik olarak bulunmaktadır. Bununla birlikte benzetim görevinin benzetim zamanının

tutulduğu zaman bilgisi ve benzetim ağı topolojisinin kurulmasını sağlayan topoloji kurucu nesne (*TopologyBuilder sınıfı*) bulunur. Bu sınıf kesikli olay benzetimi dışında benzetim görevleri içeren benzetim ağlarının tanımlanmasına imkan sağlamak için gerçekleştirilmiştir. Benzetim görevleri benzetim altyapısında tanımlanmış SimUnit arayüzünü gerçekleştirir. Bu sınıf benzetim içindeki bütün birimlerde olması beklenen olay kabul etme davranışını gerçekleştirir. Benzetim görevi sınıfı, kendi türünden bir komşular dizisi içerir. Komşularıyla bağlantısının nasıl ve ağdaki hangi görevlerle olacağı, seçilen topolojiye bağlıdır. Benzetim görevlerinde her komşusu için planladığı son olayın zaman bilgisi tutulur. Bu bilgi zaman uyumlamada boş ileti yollarken, ilgili komşuya geçmiş zamanlı bir olay iletimini engellemek için kullanılır. Benzetim görevlerinde komşu görev ve bu komşuya yollanan son ileti zamanı komşu bağlantısı sınıfı (*SimLink sınıfı*) içinde sarmalanmışlardır.

Mantıksal görev sınıfı (*LogicalProcess sınıfı*), benzetim görevleri sınıfını genişleterek kesikli olay yönetimi davranışını sağlar. Bu sınıfta olay yönetim ve olay iletimi yetenekleri bulunur. Mantıksal görev sınıfı, komşularından gelen olayların saklandığı olay kuyrukları içerir ve bu kuyruklar üzerinden Çizelge 5-1‘de verilen benzetim yürütücü algoritmasını gerçekleştirir. Mantıksal görev nesneleri seçilen olayların benzetim uygulaması tarafından işletilmesi ve ilgili olayların komşu mantıksal görevlere iletilmesinden sorumludurlar. Benzetim altyapısı, zaman bütünlüğünü korumak için mantıksal görev nesneleri dışındaki benzetim birimlerinin olay iletmesini engeller.

Mantıksal görev sınıfı, zaman uyumlama için boş ileti algoritmasını kullanır. Mantıksal görev sınıfından türetilmiş bütün nesneler, en küçük zamanlı olay ve bu olayla eş-zamanlı olayları işledikten sonra, bütün komşularına ileri zaman etiketli bir olay gönderir. Boş iletiler yollanırken; son ileti gönderim zamanı, gönderilecek boş ileti zamanından daha ileri olan komşularına, bu komşu zaten gönderilecek ileti zamanına ilerlediği için, boş ileti göndermez.

Benzetim görevlerine ait UML çizenekleri Şekil 5-7'de gösterilmiştir.



Şekil 5-7. Benzetim görevleri sınıfları UML çizenekleri

Boş iletiler sadece zaman uyumlama için zaman bilgisi içeren ve iletisi olmayan olaylardır. Dolayısıyla bu olayların benzetim uygulamasında işlenmesi anlamsızdır. Bu bağlamda benzetim yürütücü görev boş ileti olaylarını sadece zaman günlemede kullanmıştır.

Mantıksal görevler, kendilerine gelen olaylar ve işlenen olaylar için günlükler tutarlar. Bu günlükler Olay Günlük sınıfı (*EventLogger sınıfı*) yaratılan nesnelerdir. Bu sınıflar parametre olarak verilen kütük sistemi alanlarında oluşturdukları dosyalarda, kaydedilen olayları virgülle ayrılmış değer (*comma separated values, csv*) biçiminde tutarlar. Tutulan dosyalarda, günlüğü oluşturulan olayın kaynak görevi, türü, zamanı ve işlenme / göreve gelme sırası bilgileri bulunur. Bu günlükler benzetimin tutarlığı,

ne kadar süre sürdüğü gibi konularda benzetim programcısına bilgi sağlaması için gerçekleştirilmiştir.

Benzetim görevleri, gerçekleştirilen java uygulamasında eş-zamanlı çalışan işletim dizileridir. Fakat her komşu ayrı bir olay kuyruğuyla eşleşmiştir ve mantıksal görevler arası veri paylaşımı benzetim tutarlılığını sağlamak için engellenmiştir. Dolayısıyla mantıksal görevlerde, işletim dizileri arası paylaşılan bir kritik kaynak oluşmamıştır. Fakat en küçük zamanlı olayın seçimi sırasında yeni bir olayın gelmesini engellemek ve karmaşık komşuluk ilişkilerine sahip mantıksal görevlerde bütünlüğü korumak için, olay kuyrukları işlemleri ve en küçük zamanlı olayın seçilmesi işlemleri java monitör yapısı olan zaman uyumlu yöntemler (*synchronized methods*) (Schildt, 2002 ,Oracle, 2011a) ile gerçekleştirilmişlerdir.

5.1.4. Benzetim Uygulamaları

Benzetim uygulamaları, benzetimlerde modellenen fiziksel sistemi gerçekleştirir ve bu gerçekleştirimde gereken durum değişkenlerini içerir. Gerçekleştirilen benzetim altyapısında, benzetim uygulaması davranışları benzetim uygulaması arayüzünde (*SimApplication* arayüzü) tanımlanmıştır. Mantıksal görev bu arayüzde tanımlı davranışları kullanarak olayların işlenmesini ve yönetilmesini sağlar. Tanımlanan arayüzde benzetim uygulaması aşağıdaki davranışlara sahiptir:

- *processEvent(Event e):* Mantıksal görevin seçtiği olayın işlenmesi davranışıdır. Benzetim uygulamasının temel işlevidir.
- *initSimulation():* Benzetim başlangıcında, mantıksal görev tarafından çağrılır. Benzetim uygulamasının benzetim başlangıcında yapacağı işlemleri ve göndereceği iletileri tanımlar.
- *getLookAhead():* Benzetim uygulaması için ilerletme değeri (*look ahead value*) bilgisini gönderir. Açıklandığı üzere, bu bilgi benzetim uygulamasına bağıl bir değerdir ve benzetim programcısı tarafından belirlenir.
- *isSimulationEnd():* Benzetimin bitiş koşulunu tanımlar. Mantıksal görev bu yöntem yanlış (*false*) değer döndüğü süre boyunca çalışır. Bu tanımlama erişilecek belli bir zaman veya benzetim uygulamasında oluşacak bir koşul olabilir.

Çizelge 5-3'te, gelen ileti zamanlarına göre zamanı ilerleterek, benzetim zamanı 20 olunca sonlanan bir benzetim uygulaması örneklenmiştir.

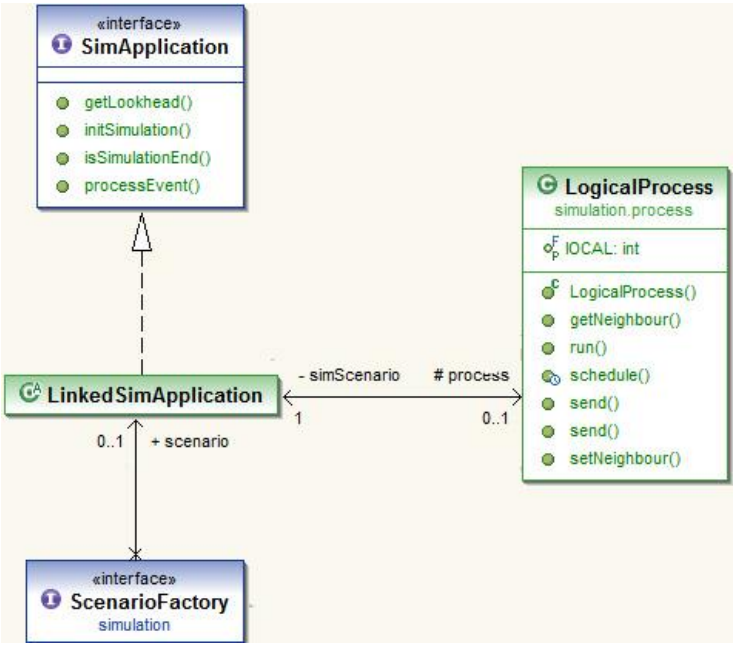
```
public class TestScenario implements SimApplication{

        privateLogicalProcessprocess;



        public TestScenario(LogicalProcess process) {

                this.process = 0;

        }

        @Override

        public void processEvent(Event e) {

                return;

        }

        @Override

        public double getLookhead() {

                return 1;

        }

        @Override

        public void initSimulation() {

                return;

        }

        @Override

        public boolean isSimulationEnd() {

                return (process.getTime()>= 20);

        }

}
```

Çizelge 5-3. Benzetim uygulaması örneği

Bağlı benzetim uygulaması sınıfı (*LinkedSimApplication sınıfı*), benzetim uygulaması arayüzünü gerçekleştirerek, uygulamayı bir mantıksal görev ile eşleyen soyut bir sınıftır.

Benzetim uygulamaları mantıksal görevlerin nitelikleridir. Fakat mantıksal görevler benzetim uygulamasının yapısını ve niteliklerini bilmeden, sadece davranışlarını kullanır. Dolayısıyla benzetim uygulamasını benzetim başlangıcında durağan bir yöntemle yaratması olanaksızdır. Bu problemi çözmek için *AbstractFactory* tasarım örüntüsü (Gamma, 1994) kullanılmıştır. Mantıksal göreve parametre olarak aldığı fabrika nesnesinin, çalışma zamanında bir benzetim uygulaması nesnesi yaratması sağlanır.

Benzetim uygulamaları sınıflarına ait UML çizenekleri, Şekil 5-8'de gösterilmiştir.



Şekil 5-8. Benzetim uygulamaları sınıfları UML çizeneği

5.1.5. Benzetim Ağı

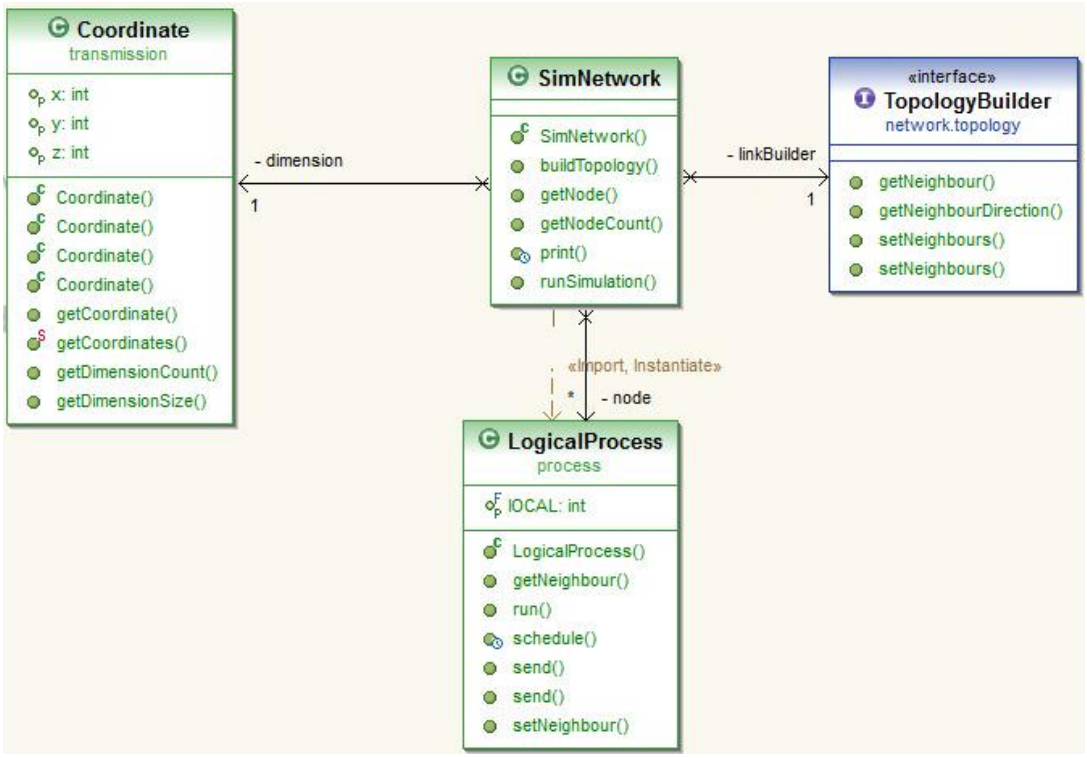
Benzetim ağı son kullanıcının etkileştiği seviyedir ve en üst düzeyde benzetim denetimi ve kullanımı sağlar. Benzetim ağı, benzetim görevleri ve niteliklerinin yaratılıp yönetildiği katmandır. Benzetimin yönetildiği temel sistemdir. Bu sistem Benzetim Ağı sınıfı (*SimNetwork sınıfı*) içinde gerçekleştirilmiştir.

Benzetim ağı, oluşturulacak benzetimin kurulumu için gerekli nitelikler olan benzetim uygulaması fabrika nesnesi, benzetim topoloji kurucusu nesne ve benzetim boyutları, benzetim ağı nesnesine başlangıç parametreleri olarak aktarılırlar. Benzetim ağı nesnesi bu parametreleri kullanarak benzetim başlatılması ve çalıştırılması için gerekli temel görevleri yerine getirir. Bu görevler tüm mantıksal görevlerin oluşturulması, topoloji bağlamında komşulukların kurulması ve mantıksal görevlerin işletim dizileriyle eşleştirilip çalıştırılmasını kapsar.

Benzetim ağı, benzetimin kurulup çalışması görevlerini iki temel davranışı üzerinden gerçekleştirmektedir:

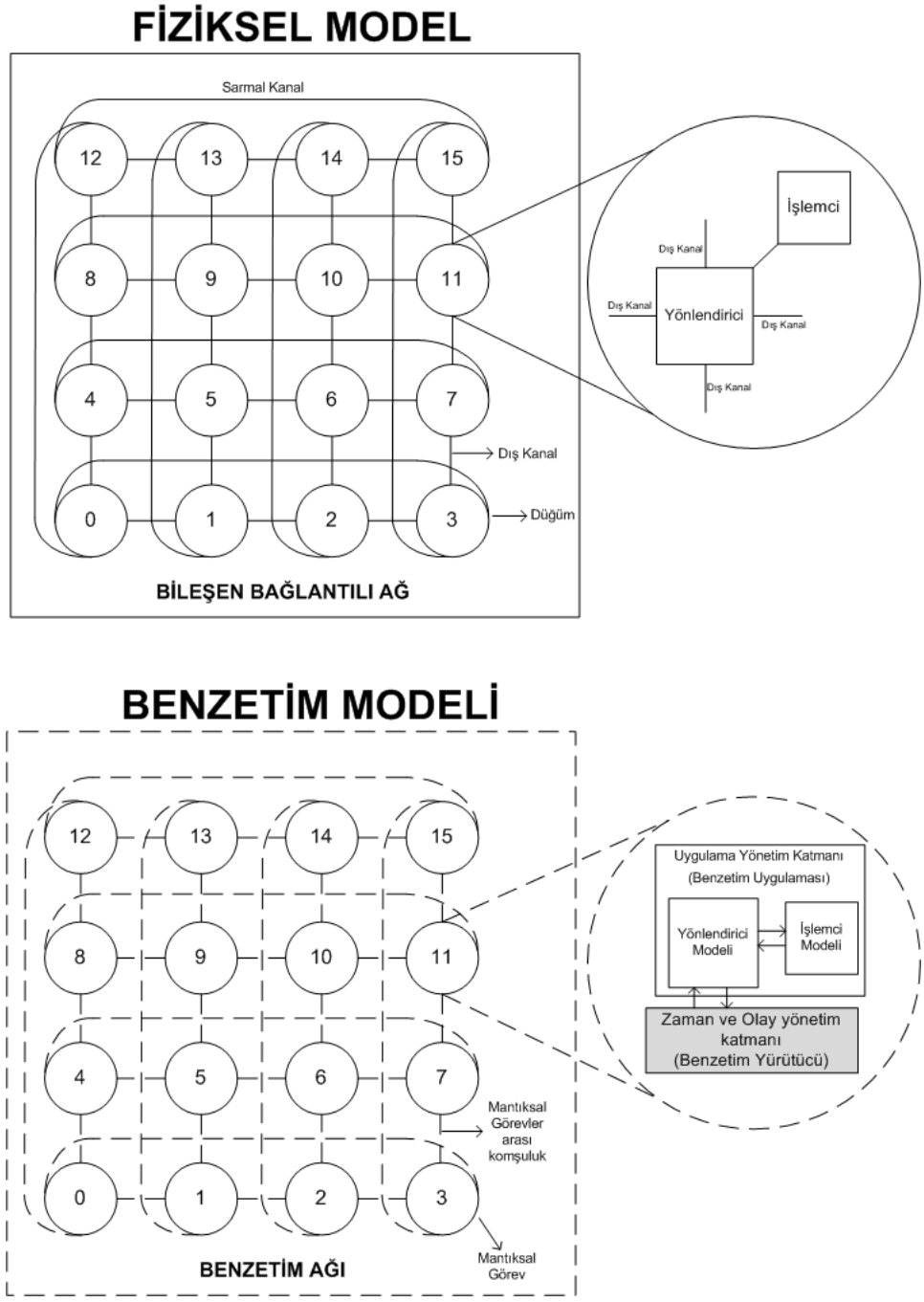
- *buildTopology():* Mantıksal görevlerin benzetim uygulamalarıyla birlikte oluşturulmalarını sağlar. Daha sonra oluşturulan mantıksal görevler arasında, belirtilen boyutlarda ve topolojide benzetim ağının kurulmasını sağlar. Topolojiye bağıl biçimde komşuluklar kurulurken, Topoloji kurucu nesne (Bölüm 5.1.2) kullanılır.
- *runSimulation():*Oluşturulan her mantıksal görevi bir java işletim dizisi ile eşleştirerek koşut olarak çalıştırır. Her işletim dizisi, çalışma süresince mantıksal görevlerin işlevlerini (benzetim yürütücü ve benzetim uygulaması) yerine getirirler.

Benzetim ağının varsayılan gerçekleştiriminde; bütün mantıksal görevler aynı benzetim, aynı topoloji ve ağ içinde konumlandırırlar. Benzetim ağı sınıfının UML çizeneği Şekil 5-9'da gösterilmiştir.



Şekil 5-9. Benzetim ağı sınıfı için UML çizeneği

5.2. Çok işlemcili ağ benzetiminin gerçekleştirimi



Şekil 5-10. Gerçekleştirilen benzetim modeli

Dolaysız ağlarda, her işlemci, iletişim işlemleri için kendisiyle eşleşmiş bir yönlendiriciye sahiptir. Bu sebeple dolaysız ağlara yönlendirici tabanlı ağlar da denir. Dolaysız ağlarda her düğümün bir veya birden fazla düğüme doğrudan komşuluğu bulunur. Komşu olmayan iki düğüm arasındaki ileti, düğümler arasında belirlenen yol boyunca bulunan yönlendiricilerden geçerek iletilir.

Dolaysız ağ benzetiminde temel problem, düğümler arası iletişimdeki ileti trafiğinin ve iletilerin hedef düğümlere ulaşmasının zaman tutarlılığıyla birlikte modellenmesidir. Dolaysız ağ iletişiminin yönlendiriciler üzerinden olduğu düşünüldüğünde, benzetimin temel birimi yönlendiriciler olacaktır. Bu bağlamda ağ benzetimi yönlendiricilerin tutarlı benzetimi üzerine kurulmuştur. Gerçekleştirimde, temel olarak yönlendiricilerin gerçeğe yakın ve tutarlı iletişim kurmaları üzerinde çalışılmıştır.

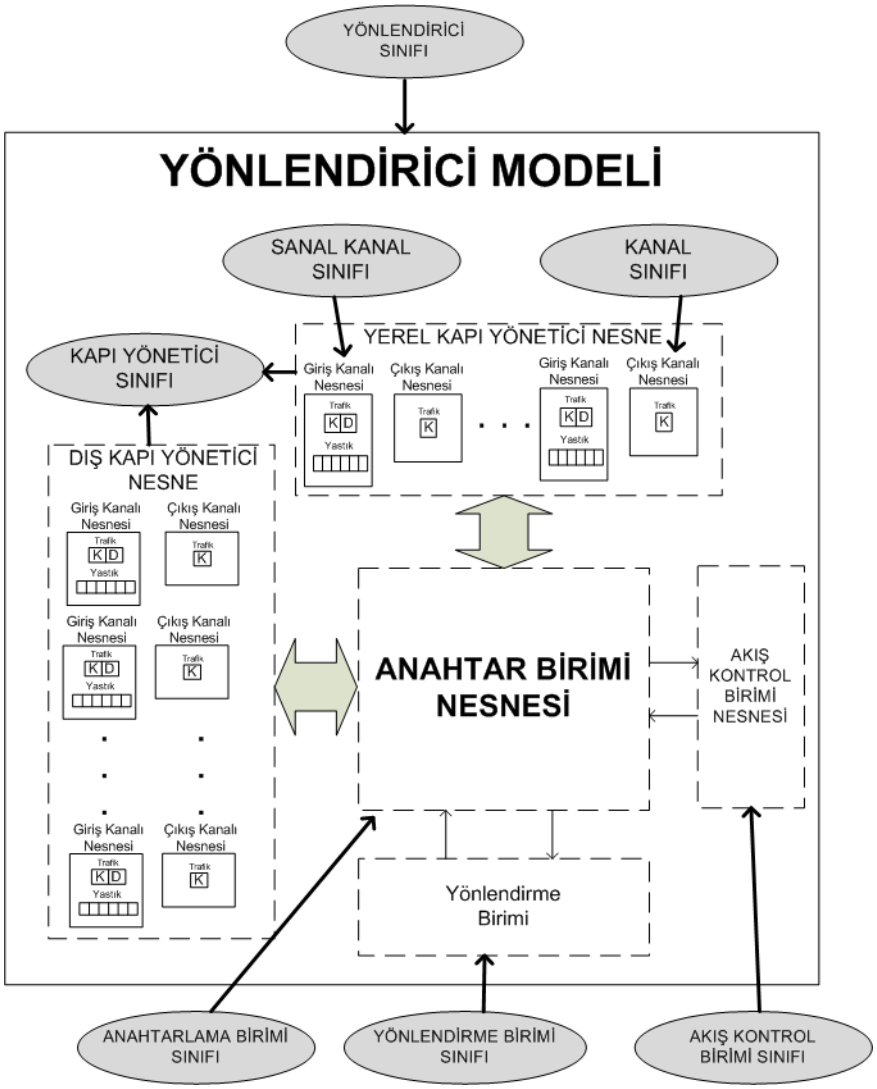
Benzetimi gerçekleştirilen dolaysız ağ içindeki yönlendiricilerde anahtarlama modeli olarak *wormhole* anahtarlama kullanılmıştır. Ağ içindeki yönlendiriciler arasındaki iletiler, *wormhole* anahtarlama yaklaşımında iletilmiş ve ağ içindeki temel ileti birimi *flit* olmuştur.

Benzetim gerçekleştiriminde, Bölüm 5.1'de anlatılan benzetim altyapısı kullanılmıştır ve ağ modeli ile benzetim modeli birbirleriyle tam olarak eşleştirilmişlerdir. Benzetimde her bir mantıksal görevin bir yönlendirici işlemci çiftini modellemesi sağlanmış ve mantıksal görevler kullanıcı tarafından seçilen topolojiyle birbirlerine bağlanmışlardır. Benzetim içinde zaman kaymalarını ve tutarsızlıkları engellemek için, benzetim uygulamalarının birbirlerine doğrudan ileti göndermeleri engellenmiştir. Gerçekleştirilen benzetim modelinde; herhangi bir yönlendirici bir iletim yapacağı zaman, iletinin veya paketin iletim yönünü belirler. Daha sonra o yöndeki göreve iletiyi göndermesi için, kendisine eşleştirilmiş mantıksal göreve bir olay planlar. Planlanan olay, mantıksal görevin benzetim zamanı iletim zamanına erişince, mantıksal görev komşu mantıksal göreve iletinin ulaşacağı zamanda bir olay gönderir. İleti trafiğinde bir çakışma olduğunda, ileti çakışmanın olduğu yönlendirici içinde bekletilir. İleti, trafiği ele geçirdiğinde ilgili mantıksal göreve o anki benzetim zamanı ve iletim maliyetlerini zaman bilgisi olarak içeren bir olay ile iletilir. Olay tabanlı bu iletim benzetimi, ileti hedef yönlendirici modele ulaşıncaya kadar aynı biçimde devam eder. Gerçekleştirilen benzetim modeli ve fiziksel model Şekil 5-10'da gösterilmiştir. Fiziksel modeldeki düğümler birbirlerine *torus* topolojisiyle bağlanmışlarıdır.

Benzetim gerçekleştiriminde öncelikle ağ benzetimi ve temel iletişim protokolleri; daha sonra kurulan iletişim protokolleri üzerinde seçilen örnek toplu iletişim algoritmaları gerçekleştirilmiştir. Test sonuçları Bölüm 6'da gösterilmiş ve tartışılmıştır.

Dolaysız ağ benzetimi, Bölüm 3'te anlatılan topoloji, anahtarlama, akış kontrolü ve yönlendirme bileşenlerine ayrılmış ve her bileşen ayrı olarak gerçekleştirilmiştir. Böylece farklı bileşenlerin seçilmesiyle oluşturulacak dolaysız ağ birleşimlerine olanak sağlanmıştır.

5.2.1. Yönlendirici Modelli ve Özellikleri



Şekil 5-11. Gerçekleştirilen Yönlendirici Modeli

Dolaysız ağ benzetimi, gerçekleştirilen benzetim altyapısını kullanmaktadır ve yönlendirici sınıfı (*Router sınıfı*) bir benzetim uygulaması gerçekleştirimidir. Gerçekleştirimde Bölüm 3.2'deki yönlendirici modeli temel alınmıştır. Yönlendirici içinde ileti trafiğini yönetmekte kullandığı üç birim bulunur. Yönlendirici, gelen iletinin hangi yöne (kanala) iletileceğini belirlemek için yönlendirme birimini kullanır. Yönlendirme işlemi yapılmış iletilerin, iletilmelerinde veya bekletilmelerinde gerekli

kaynakların, iletilere atanması işlemleri akış kontrol birimi tarafından yönetilir. Gelen kanal yastıklarındaki iletilerin, çıkış kanallarına aktarılması işlemleri de anahtar birimi tarafından yönetilir. Gerçekleştirilen yönlendirici modeli Şekil 5-11'de verilmiştir.

Her yönlendiricide giriş ve çıkış kanallarının yönetiminden sorumlu yerel ve dış kapı yönetim birimleri bulunur. Kapı yönetim birimleri Kapı Yöneticisi arayüzünü (*PortManager arayüzü*) gerçekleştirir. Gerçekleştirimde iki çeşit kapı yöneticisi birim gerçekleştirimi bulunur: *Tek kapılı* ve *Tam kapılı* kapı yönetim birimleri. Tek kapılı yönetim birimlerinde tüm trafik yönleri için sadece bir giriş çıkış kanal çifti bulunur. Tam kapılı kapı yönetim biriminde ise her yön için bir giriş çıkış kanal çifti bulunur. Yönlendirici içindeki yerel kapı yönetim birimi seçime bağlı olarak tek kapılı veya tam kapılı olabilir. Dış kapı yönetim birimi her zaman tam kapılı kapı yönetim birimidir. Kapı yönetim birimleri yönlendirici içindeki giriş ve çıkış kanalları içerir ve yönetirler.

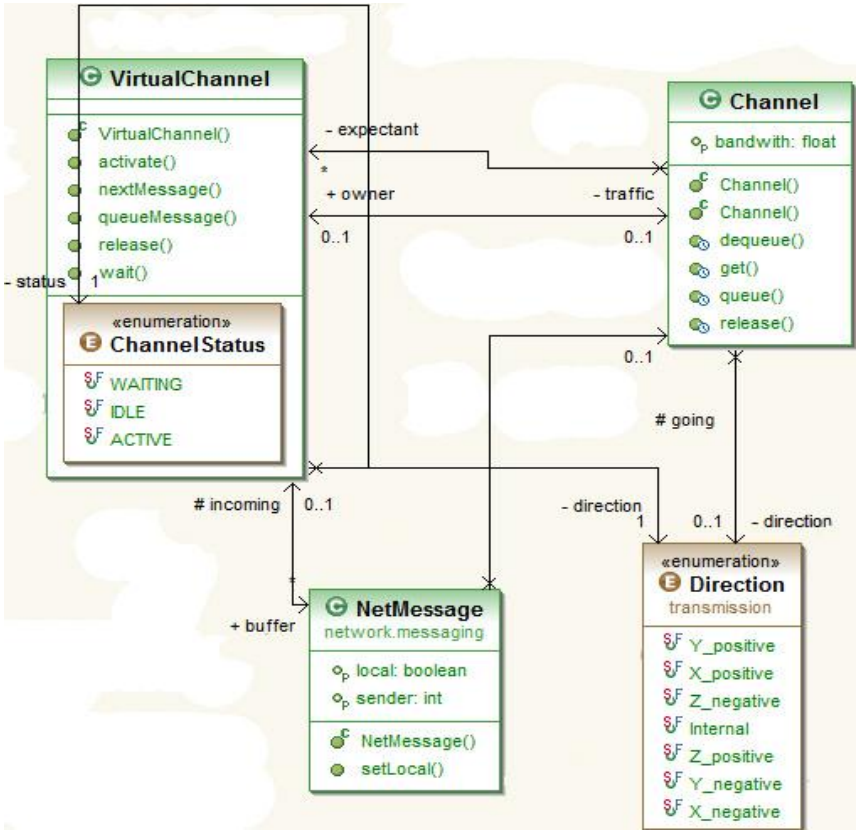
Giriş kanalları (*VirtualChannel* sınıfı), Bölüm3.2'de açıklanan giriş kapısı modeline benzer biçimde gerçekleştirilmişlerdir. Her giriş kanalı, kanal durumu, giriş kanalına atanmış yastık ve ileti trafiğinin atandığı çıkış kanalı bilgilerini tutar. Girişi kanalına atanmış bir trafik olmadığı durumda, çıkış kanalı herhangi bir çıkış kanalını göstermez. Giriş kanalının trafiği ilettiği çıkış kanalını göstermesi, giriş kanalından bütün bir ileti aktarılana kadar giriş kanalının başka bir iletiye atanmadığı, *wormhole* ve sanal kısa yol gibi anahtarlama yöntemlerinde veri *flit* paketlerinin iletiminde yönlendirme bilgisi olarak kullanılır. Yerel giriş kanalları, dış giriş kanallarıyla aynı özellikleri taşırlar. Fakat yerel giriş kanallarında, yastık boyu sınırlandırılmamıştır ve kanal aktarım sığası diğer fiziksel kanallara göre daha hızlıdır.

Benzetimdeki çıkış kanalları (*Channel* sınıfı), birimler arasındaki fiziksel iletim kanallarını modellerler. Yerel çıkış kanalları yönlendirici ve işlemci arasında, dış çıkış kanalları ise yönlendiriciler arasında iletişimi sağlar. Her kanal kendisine ait aktarım sığası bilgisini barındırarak, bir paketin kanal üzerinden iletim süresinin hesaplanabilmesini sağlar. Çıkış kanalları, giriş kanallarındaki yastıklarla birlikte ağ içindeki kaynakları oluştururlar.

Aynı anda iki ileti trafiğinin aynı çıkış kanalına ihtiyaç duyduğunda oluşan duruma *çakışma* denir. Çakışma durumu yönlendirici gerçekleştirimindeki anahtarlama ve akış kontrolü yöntemlerine bağlı biçimde çözülerek, çakışan kanallardan biri seçilir.

Diğer ileti trafiğine sahip giriş kanalı, seçilen ileti trafiği bitene kadar bekleme durumuna geçer. Modellenen çıkış kanallarında, kanalı bekleyen giriş kanallarının tutulduğu bir bekleyen kuyruğu bulunur. Bir ileti trafiği sona erdiğinde, kanal trafiği bekleyen kuyruğundaki ilk giriş kanalına atanır. Kuyruk boş ise, kanal "Boş" durumuna geçer.

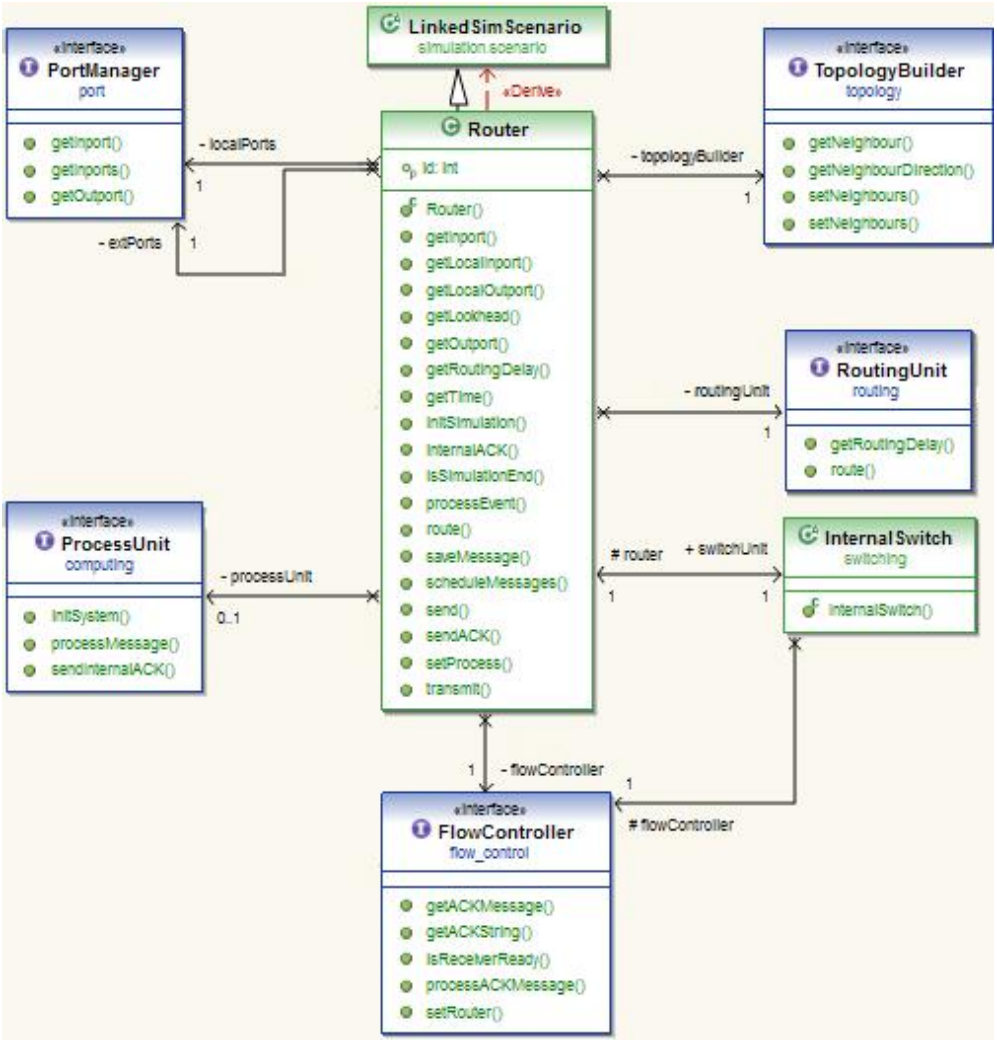
Benzetimdeki giriş ve çıkış kanalları sınıflarının UML çizenekleri Şekil 5-12'de gösterilmiştir.



Şekil 5-12. G/Ç kanalları sınıfları UML çizenekleri

Gerçekleştirilen benzetimde, işlemci yönlendirici üzerindeki trafikten bağımsız biçimde paket iletimi isteği göndermektedir. Gönderilen paketin ihtiyaç duyduğu yerel giriş kapısının (giden kanalın) kullanımda olduğu durumda, paket giriş kapısı üzerindeki bekleme kuyruğunda bekletilmektedir. Giriş kapısı üzerindeki trafik sona erdiğinde, giden kanal trafiği bekleyen kuyruğundaki pakete atanmaktadır. Eğer bekleyen paket yoksa kanal "Boş" durumuna geçmektedir.

Yönlendirici içindeki ileti trafiğinin yönetimi, temel olarak yönlendiricide seçilen anahtarlama biçimine bağlıdır. Gelen iletilerin yastıkta saklanma süreleri, hangi iletilere yönlendirme işlemi uygulanacağı, iletilere *flit* veya paket düzeyinde kaynak ataması yapılacağı seçimlerinin bütünü anahtarlama yöntemi tarafından belirlenir. Gerçekleştirilen yönlendirici modelinde, anahtarlama işlemleri anahtar birimi tarafından yürütülmektedir. Bu bağlamda yönlendiricinin temel işlem birimi anahtar birimi olmaktadır. Akış kontrol ve yönlendirme birimleri, yönlendirici içinde gerçekleştirilmiş olsalar bile, anahtar birimine bağlı biçimde çalışmaktadırlar. Tez kapsamında gerçekleştirilen yönlendirici modelinde, akış kontrol birimi anahtar birimi içinde konumlanmıştır. Fakat yönlendirme biriminin anahtar birimine bağlı olması gerçek modele uymayacağı için, yönlendirme birimi yönlendirici içinde alt birim olarak gerçekleştirilmiştir. Yönlendirici modeli sınıfının UML çizeneği Şekil 5-13'te gösterilmiştir.



Şekil 5-13.Yönlendirici modeli sınıfının UML çizeneği

Ağ benzetiminde, yönlendirici dışındaki birimlerin doğrudan olay üretmeleri engellenmiştir. Gerçekleştirilen çözüm yaklaşımında, anahtar birimi gelen iletilerin işlenmesini ve gerekli çıkış kanalına veya gerekli kanal kuyruğuna yönlendirilmesi işlemlerini sağlar. Yönlendirici, her benzetim zamanı döngüsünde, bulunulan benzetim zamanı anında etkin olan her giriş kanalından, bu giriş kanalına atanmış çıkış kanalına, kanal uygun ise, bir ileti gönderimi yapar. Yönlendirici her çıkış kanalı için bir serbest kalma zamanı bilgisi tutar ve ileti gönderiminden sonra, çıkış kanalının serbest kalma zamanını iletim süresi kadar artırır. Giriş kanalın ihtiyaç duyduğu çıkış kanalının, işlem yapılan benzetim anında uygun olmadığı durumda, iletim yapılmaz. İleti, çıkış kanalının serbest bırakıldığı benzetim anına kadar bekletilir. Bu yöntemle, her iletinin gereken benzetim anında gönderilmesi sağlanarak, akış kontrol birimlerinde oluşacak tutarsızlıklar engellenir. Alt birimler alındı paketi (*ACK packet*) iletimi için yönlendiriciye istekte bulunabilirler. Alındı paketleri akış kontrol biriminin yönettiği kaynaklardan bağımsız olduğu için, bu paketlerin gelecek zamanlı gönderimine izin verilmiştir.

Gerçekleştirilen ağ benzetiminde, her yönlendiriciye bir işlem birimi (işlemci) bağlıdır. Yönlendirici, kendisine atanmış işlemcisine iki temel hizmet sağlar: işlemcisine gelen iletileri işlemesi için işlemcisine iletir ve işlemcisinin iletim için yolladığı paketin iletimi bittiğinde, işlemcisine geri bildirimde bulunur. Bu bağlamda işlemci, benzetim uygulamasına benzer çalışır. İşlemci gerçekleştirimi Bölüm 5.2.6'da anlatılmıştır.

5.2.2. Topoloji Altyapısının Gerçekleştirimi

Gerçekleştirilen ağ benzetimde, topoloji bağlamında yönlendiriciler arası bağlantılar mantıksal görevler düzeyinde yapılmıştır. Uygulama düzeyinde topoloji bağlantıları yapılmamıştır. Mantıksal görevlerin bağlantıları topoloji kurucu nesneler kullanılarak yapılmıştır. Ayrıca yönlendirme ve iletim işlemlerinde, topoloji kurucu nesneler mantıksal görevlere çalışma zamanında belirtilen yöndeki komşu adres bilgisi sağlaması için de kullanılmıştır. Yönlendirici dışındaki birimler de topolojik bilgiye ihtiyaç duyduklarında, topoloji kurucu nesneleri kullanmışlardır.

5.2.3. Anahtarlama Altyapısının Gerçekleştirimi

Çok işlemcili ağlarda seçilen anahtarlama yöntemi, ağ içindeki iletişim yaklaşımının yöntemini belirler. Ağ kaynaklarının yönetimi, iletim süresi ve maliyetlerinin bütünü anahtarlama modeline bağlı olarak değişen parametrelerdir. Anahtar birimlerinin temel işlevi, giriş ve çıkış kanalları arasındaki eşgüdümü sağlamak, gelen iletilerin uygun çıkış kanallarına yönlendirmektedir.

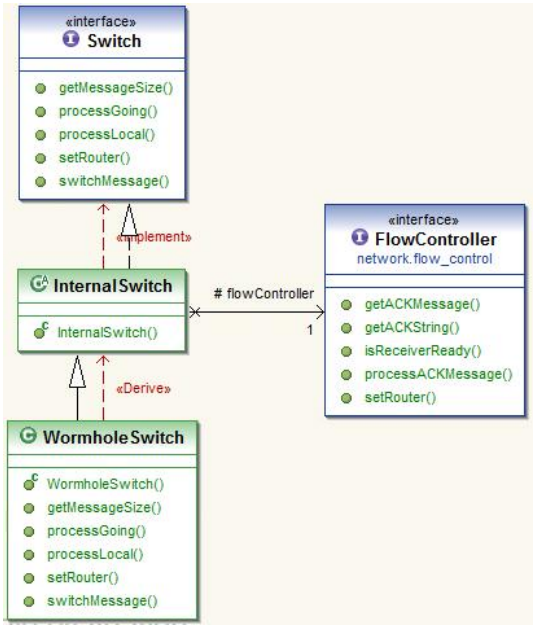
Gerçekleştirilen ağ benzetimindeki anahtar birimi gelen iletileri ilk ele alan birimdir. Anahtar birimi gelen ileti için, yönlendirme birimini kullanarak gerekli yönlendirme işlemlerini yerine getirdikten sonra, seçilen çıkış kanalının uygun olduğu durumda iletiyi doğrudan hedef yönlendiriciye iletir. İleti için seçilen çıkış kanalının başka bir ileti trafiğine atandığı durumda iletiler gelen kanal yastığında bekletilir. Bekletilen iletiler, iletinin beklediği çıkış kanalı trafiği boşalınca karşı tarafa gönderilirler. İletinin nasıl ve hangi maliyetle yönlendireceği, gerçekleştirilen anahtarlama yöntemine göre değişiklik gösterir. Anahtarlama davranışlarının bütünü anahtarlama birimi üzerinde toplanarak yönlendirici nesnenin anahtarlama yöntemi detaylarından soyutlanması amaçlanmıştır. Böylece yönlendirici anahtarlama modelinden bağımsız biçimde çalışması sağlanmıştır.

Yapılan benzetim yaklaşımında, yönlendirici nesne gelen iletiyi anahtar birime iletir. Anahtar birimi gelen ileti için gerekli yönlendirme işlevlerini yapar ve çıkış kanalını belirler. Seçtiği çıkış kanalı durumuna göre, yönlendirici gereken benzetim anında iletiyi yönlendirir veya bekletir. Anahtar birimi, iletim için seçtiği iletileri ilgili düğüme yönlendirmesi için yönlendirici nesneyi kullanır.

Gerçekleştirimde anahtar biriminin, iletinin kapılar arası yönlendirilmesi dışında iki görevi daha bulunmaktadır. Bu birim temel işlevi yanında, giden iletilerin ve yerel iletimlerin işlenmesi görevlerini yerine getirir. Yönlendirici yerel işlemciye veya başka yönlendiricilere gönderilerek iletileri öncelikle anahtar birimine yönlendirir. Anahtar birimi, giden iletiyi işlerken, bu iletiye atanmış kaynaklar varsa bunları serbest bırakır veya başka trafiğe atar. Bununla birlikte anahtar birim yerel iletileri işlenip paket haline getirilmesinden ve işlemciye yönlendirilmesinden de sorumludur. Örneğin *wormhole* anahtarlama modelinde, anahtar birim yerel *flit* iletileri birleştirerek paket haline getirir ve işlemciye iletmesi için yönlendiriciye yönlendirir.

Bir gelen iletinin ilgili çıkış kanalına iletilmesi, bu ileti trafiğine bulunduğu yönlendiricideki kanal kaynağının ve alıcı tarafındaki yastık kaynağının atanması anlamına gelir. Dolayısıyla yönlendirici içindeki yerel anahtar birimi gelen iletileri çıkış kanallarına iletirken, yönlendirici kaynaklarının ve alıcı tarafındaki kaynakların anlık durumlarına göre karar verir. Yönlendirici içindeki kaynakları akış denetim birimleri yönetirler. Bu durum akış kontrol ve anahtar birimlerinin eşgüdümlü olarak çalışmasını zorunlu kılar. Benzetim gerçekleştiriminde, Akış denetim biriminin anahtar biriminin niteliği olmasının, anahtar tasarımını sadeleştireceği ve iki birim arasındaki işlemleri kolaylaştıracağı düşünülmüştür. Bu yaklaşımla, akış kontrol birimi niteliği olan bir soyut anahtar sınıfı (*InternalSwitch sınıfı*) gerçekleştirilmiştir. Akış denetim birimini yoğun biçimde kullanan anahtarlama yöntemlerinin, bu soyut sınıfı genişletmeleri hedeflenmektedir.

Tez kapsamında *wormhole* anahtarlama yöntemi gerçekleştirilmiştir. *Wormhole* anahtarlama yöntemi soyut anahtar birimi genişletilerek gerçekleştirilmiştir. *Wormhole* anahtarlamada iletiler *flit* paketleri düzeyinde yönlendirilir. İleti trafiklerine kanal kaynakları paket düzeyinde atanır. Dolayısıyla bir paket kanal üzerinden tamamen iletilmeden, başka bir ileti trafiğine atanmaz. Bu anahtarlama yaklaşımında yastık kaynakları *flit* düzeyinde atanır ve bir çakışma durumunda her *flit* bulunduğu yönlendiricideki yastıkta saklanır. Anahtarlama arayüzü ve sınıflarının UML çizeneği Şekil 5-14'te gösterilmiştir.



Şekil 5-14. Anahtarlama arayüzü ve sınıflarının UML çizeneği

Gerçekleştirilen *wormhole* anahtarlama algoritmasında, iletiye yapılacak işlemler *flit* paketi türüne göre değişir. *Wormhole* anahtarlamada bağlamında, dört çeşit *flit* paket türü tanımlanmıştır: *Başlık, Veri, Kuyruk* ve *Alındı flit* paketleri.

Başlık *flit* paketleri, *flit* paketlerine bölünmüş iletinin ilk *flit* paketi veya paketleridir. Sadece başlık *flit* paketlerinde iletinin yönlendirmesinde kullanılacak bilgiler bulunur. Bu sebeple iletinin yönlendirilmesi ve yönlendirme sonucunda ihtiyaç duyduğu çıkış kanalının belirlenerek atanması, başlık *flit* paketi yönlendiriciye ulaştığında yapılır. Bir iletiye atanan trafik hakkı, bu iletinin bütünü kanaldan iletilmeden başka kanala atanmaz. Dolayısıyla iletinin ihtiyaç duyduğu kanal başka bir trafik tarafından kullanılıyorsa, gelen giriş kanalı çıkış kanalına yönlendirilir ve giriş kanalı bekleme durumuna geçer. Gelen *flit* paketleri, giriş kanalındaki yastıkta saklanır. Çıkış kanalı uygun olduğunda, giriş kanalı aktif duruma geçer ve *flit* paketleri karşı tarafa gönderilmeye başlanır. Başlık *flit* paketleri, ilk kaynak atamaların yapıldığı paketler oldukları için önemlidirler.

Veri *flit* paketleri herhangi bir yönlendirme bilgisi içermezler. Bu paketler yönlendiriciye ulaştığında, kendisinden önceki başlık *flit* paketlerinin yönlendiriciye erişmiş olduğu ve ilgili yönlendirme işlemlerinden sonra giriş kanalının ilgili çıkış kanalına yönlendirildiği düşünülür. Bu sebeple yönlendirme bilgisi için giriş kanalındaki çıkış kanalı bilgisi kullanılır. Veri *flit* paketleri, yönlendirme işlemi yapılmadan bu kanala yönlendirilir.

Kuyruk *flit* paketleri, veri *flit* paketleriyle aynı şekilde ele alınır. Fakat kuyruk *flit* paketinin iletiminden sonra, çıkış kanalı serbest kalır. Bu sebeple kuyruk *flit* paketinin iletiminden sonra, iletildiği çıkış kanalı, varsa bekleyen giriş kanalına atanır. Çıkış kanalını bekleyen giriş kanalı yoksa kanal serbest durumda kalır.

Alındı *flit* paketleri, veri ve başlık *flit* paketlerinden farklı olarak alıcı veya gönderen yönlendiricilerden bilgilendirme amacıyla yollanırlar. Gerçekleştirilen yönlendirici benzetiminde alındı paketleri akış denetim birimlerine aktarılmıştır. Alındı paketinin işlenmesi sonrasında bir iletinin gönderilmesi gerektiğinde, anahtar birimi gerekli paketi yönlendiriciye iletmiştir. *Wormhole* anahtarlama algoritması Çizelge 5-4'te gösterilmiştir.

```
İleti_anahtarla(İleti i)
{
    Flit f = (Flit)f;

   Eğer f BAŞLIK türü flit paketi ise {
       F flit paketini yönlendir.
       İlet(f);
}

   Eğer f VERİ türü flit paketi ise {
f flit paketini gelen kanal trafiğine yönlendir.
       İlet(f);
}

   Eğer f KUYRUK türü flit paketi ise {
f flit paketini gelen kanal trafiğine yönlendir.
       İlet(f);
}
}
İlet(Flit f) {
   Eğer f paketinin iletileceği yöndeki çıkış kanalı UYGUN ise {
f paketini iletmesi için yönlendiriciye ilet.
        Eğer f flit paketi KUYRUK türü flit paketi ise {
            Çıkış kanalını varsa bekleyen giriş kanalına ata, yoksa serbest bırak
            return;
}
}
   Giriş_kanal[f.gelen_kanal].sakla(f);
}
```

Çizelge 5-4.*Wormhole* anahtarlama algoritması

5.2.4. Akış Kontrolü Altyapısının Gerçekleştirimi

Akış kontrolü bir iletiye ağ içindeki kaynakların hangi bağlamda atanacağını tanımlar. Bu bağlamda akış kontrol birimi, yönlendirici içindeki ağ kaynaklarının yönetiminden sorumlu birimdir. Akış kontrol birimi, yastıklardaki paketlerin ağ ortamındaki kanallara aktarılıp hedef düğümlere doğru ve tam biçimde iletilmesi için gerekli kaynak atamalarını yönetir.

Bir paket alıcı tarafına başarıyla iletildiğinde, alıcı yönlendirici tarafında pakete kaynak ataması yapıldığı anlamına gelir. Bu durumda pakete yönlendirici içinde atanmış kaynaklar serbest bırakılır veya başka bir pakete atanır. Alıcı tarafında yeterli kaynak olmadığı bilgisi ulaştığında, yönlendiricinin iletimi durdurması ve iletilere kendi içinde atadığı kaynakları tutması gerekir.

İki yönlendirici arasında bir iletim trafiği olduğunda, paket iletimlerinin doğruluğu ve başarımı hakkında alıcı düğümün gönderen tarafına, bir şekilde durum bilgilendirmesi yapması gerekir. Ağ ortamında alıcı ve gönderenler arasındaki bilgilendirmeler alındı paketleriyle sağlanır. İletimin başarılı olduğu durumda pozitif alındı paketi, iletim hataları sebebiyle paketin iletilememesi durumunda negatif alındı paketleri yollanır. Alındı paketini alan yönlendirici bu paketleri işleyerek, ileti trafiğinin devam etmesi, durdurulması veya tekrarlanmasına karar verir. Alındı paketlerinin biçimini, ağ gerçekleştiriminde kullanılan akış kontrol yöntemi belirler.

Ağ iletimleri ilgili kaynakların iletiye atanmasına eşdeğer olduğu için, iletim trafiğine atanmış kaynakların durumları hakkında bilgilendirme sağlarlar. Dolayısıyla alındı paketlerinin, yönlendirici içindeki kaynakların yönetiminden sorumlu olan akış kontrol birimleri tarafından işlenmesi gerekir. Benzer şekilde bir paket iletimine başlamadan önce, alıcı tarafında pakete atanmış yastık kaynaklarının bulunmasının gerekmesi, iletimlere başlamadan önce alındı paketlerini işleyen akış kontrol biriminin karşı tarafa iletime onay vermesini gerektirir.

Ağ benzetiminde, akış kontrol biriminin bahsedilen sorumluluklarını kapsayan bir arayüz tasarlanmıştır (*FlowController arayüzü*). Akış kontrol birimi arayüzü, aşağıda belirtilen davranışları kapsar:

- Gelen alındı paketlerinin işlenmesi
- Yeni iletimden önce, karşı alıcı tarafındaki duruma göre iletimin başlamasına karar verilmesi
- Alıcılar için gerekli alındı paketlerinin üretilmesi.

Gerçekleştirilen akış kontrol birimi alındı paketlerini işlenmesi ve üretilmesi ile iletim başlamalarına karar verilmesi yeteneklerine sahiptir. Her yönlendirici, karşı düğüme başlattığı bir iletim içini gelen tarafına da akış kontrol biriminin ürettiği bir alındı paketi yollar.

Yapılan çok işlemcili ağ benzetiminde, kredi tabanlı akış kontrol yöntemi kullanılmıştır ve bu yöntemi gerçekleştiren Kredi tabanlı akış kontrolü sınıfı (*CreditBasedFlowController sınıfı*) tasarlanmıştır. Kredi tabanlı akış kontrol sınıfı akış kontrol birimi arayüzünü gerçekleştirmiştir.

Kredi tabanlı akış kontrolü gerçekleştiriminde, akış kontrol birimleri her yöndeki komşuları için bir kredi miktarı tutmaktadır. Benzetim başlangıcında her yöne yastık boyları kadar kredi bulunmaktadır. Yönlendiricinin her yeni ileti gönderme onayı istediğinde, akış kontrol birimi ilgili yön için tuttuğu krediyi bir azaltır ve gönderimi onaylar. Kredi sayısının sıfıra ulaştığı yönler için akış kontrol birimi iletime onay vermez ve ileti yönlendirici yastığında bekletilir.

Gerçekleştirilen yönlendirici davranışında, her iletim için akış kontrol biriminden, bir alındı paketi üretmesini ister. Bu alındı paketi, trafiğin geldiği yönlendiriciye geri bildirim için gönderilir. Kredi tabanlı akış kontrolünde akış kontrol birimi, gelen yönlendiricilere yeni bir yastık alanının boşaldığı anlamına gelen bir alındı paketi üretir.

5.2.5. Yönlendirme Altyapısının Gerçekleştirimi

Yönlendirme birimi, gelen iletinin bir sonraki iletimde hangi yöne iletileceğine karar verir. Dolaysız ağlarda iletim, iletinin bir sonraki yönlendiriciye aktarılması veya ağdan çekilerek işlemciye iletilmesi anlamına gelir. Bir iletinin yönlendirilebilmesi için, gerekli hedef düğüm adresi veya önceden hesaplanmış yönlendirilme bilgilerini içermesi gerekir.

İletileri oluşturan paketlerin hangilerine ve hangi yönlendiricilerde yönlendirme işlemi uygulanacağı, ağ gerçekleştiriminin kıstaslarına bağlıdır. Paket anahtarlama yönteminde, her paket yönlendirme bilgisi içerir yönlendiriciye ulaşan her pakete yönlendirme işlemi uygulanır. Sanal kısa yol ve *wormhole* anahtarlama yöntemlerinde ise sadece başlık paketlerine yönlendirme uygulanır ve sonraki paketler başlık paketlerin açtığı yolu takip ederler. Bununla birlikte; kaynak kararlı yönlendirme algoritmalarında, ileti için bütün iletim yolu kaynak tarafından hesaplanır. Dağıtılmış yönlendirme algoritmalarında ise ileti paketlerime her ara düğümde yönlendirme işlemi uygulanır.

Yönlendirme birimleri gelen iletilerin bir sonraki hedefine karar vermekten sorumludur. Gerçekleştirilen benzetim ağında, yönlendirme birimi sadece gelen iletilerin yönlendirilmesi davranışını sağlar. Bu davranış Yönlendirme birimi arayüzünde (*RoutingUnit arayüzü*) tanımlanmıştır. Bir iletinin veya paketin, yönlendirme birimi tarafından yönlendirilebilmesi için, yönlendirilebilir ileti (*Routable*) olması gerekir.

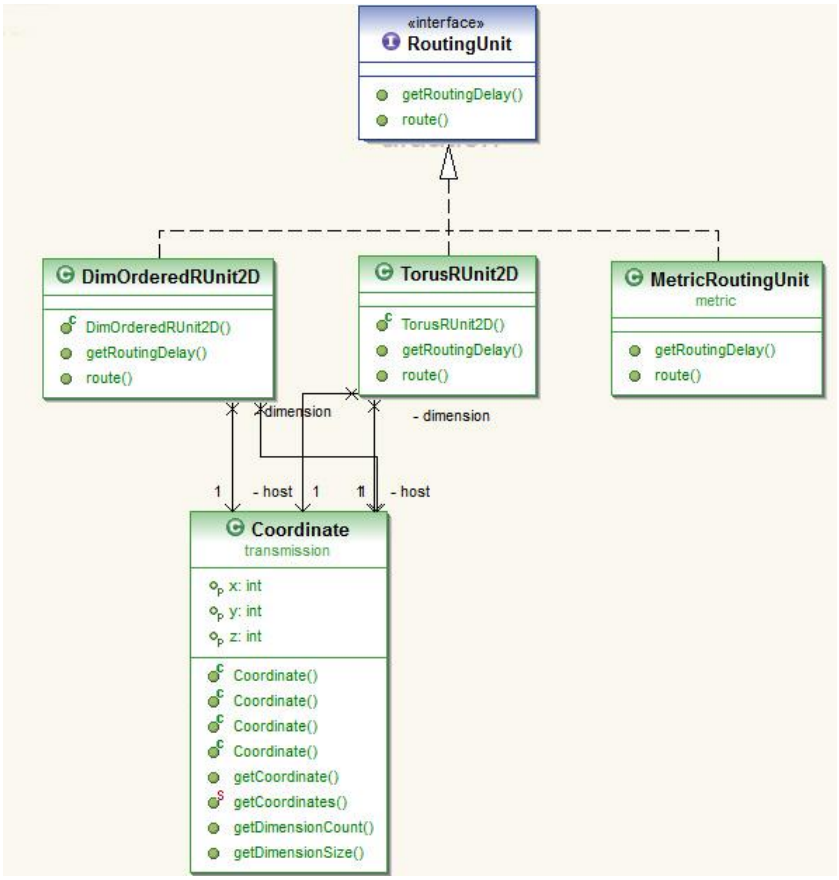
Günümüz gelişen teknolojilerinde işlemcilerin yonga üzerinde oluşturulup koşut biçimde çalıştığı çok işlemcili mimariler önem kazanmıştır. Bu sistemlerde işlemciler, veri iletişimi için yonga üzerindeki ağ sistemini (*network on chip*) kullanmaktadırlar. Yönlendiricilerin yonga düzeyinde üretilmesi, basit ve hızlı yönlendirici tasarımlarını zorunlu kılmıştır.

Boyut sıralı durağan yönlendirme algoritması olarak Bölüm 3.5.3.1'te anlatılan yönlendirme algoritması gerçekleştirilmiştir. Bu algoritma bağlamında iki boyutlu *torus* üzerinde iletinin önce x düzlemi, daha sonra da y düzlemi üzerinde ilerlemesi sağlanmıştır.

Kaynak kararlı yönlendirme algoritması olarak metrik tabanlı yönlendirme algoritması tanımlanmıştır. Bu yönlendirme yaklaşımında kaynak düğüm metriklerle bütün iletim yolunu hesaplar ve bu iletim yolu paket başlığında taşınır. İleti ara düğümlerde hesaplanan iletim yolunu, kaynağın belirlediği sırada uygulayarak iletimi sağlarlar.

Gerçekleştirilen diğer yönlendirme algoritmasında, *torus* topolojisindeki sarmal kanalları kullanan bir *torus* yönlendirme algoritması tanımlanmıştır. Bu algoritmada iletiler boyut sıralı yönlendirme algoritmasına benzer biçimde x, y ve z koordinat düzeylerinde sırayla yönlendirilmektedir. Fakat koordinat düzeylerinde, erişmek istedikleri koordinat ile bulundukları koordinat arasındaki fark, topolojisinin yarısından fazla ise negatif (ters) yönde yönlendirme uygulanarak, iletinin *sarmal* kanal üzerinden daha az adımda hedef ulaşması sağlanmıştır.

Yönlendirme birimlerine ait UML çizenekleri Şekil 5-15'te gösterilmiştir.



Şekil 5-15. Yönlendirme birimi için UML çizeneği

5.2.6. İşlemci Modelinin Gerçekleştirimi

Gerçekleştirilen ağ benzetiminde, işlemci yönlendiriciye bağlı işlem birimi olarak gerçekleştirilmiş ve işlemciye gelen iletiler yönlendirici tarafından işlenmesi için işlemciye aktarılmıştır. İşlemci ağ içindeki diğer işlemcilere ileti göndermek için yönlendiriciyi kullanmaktadır.

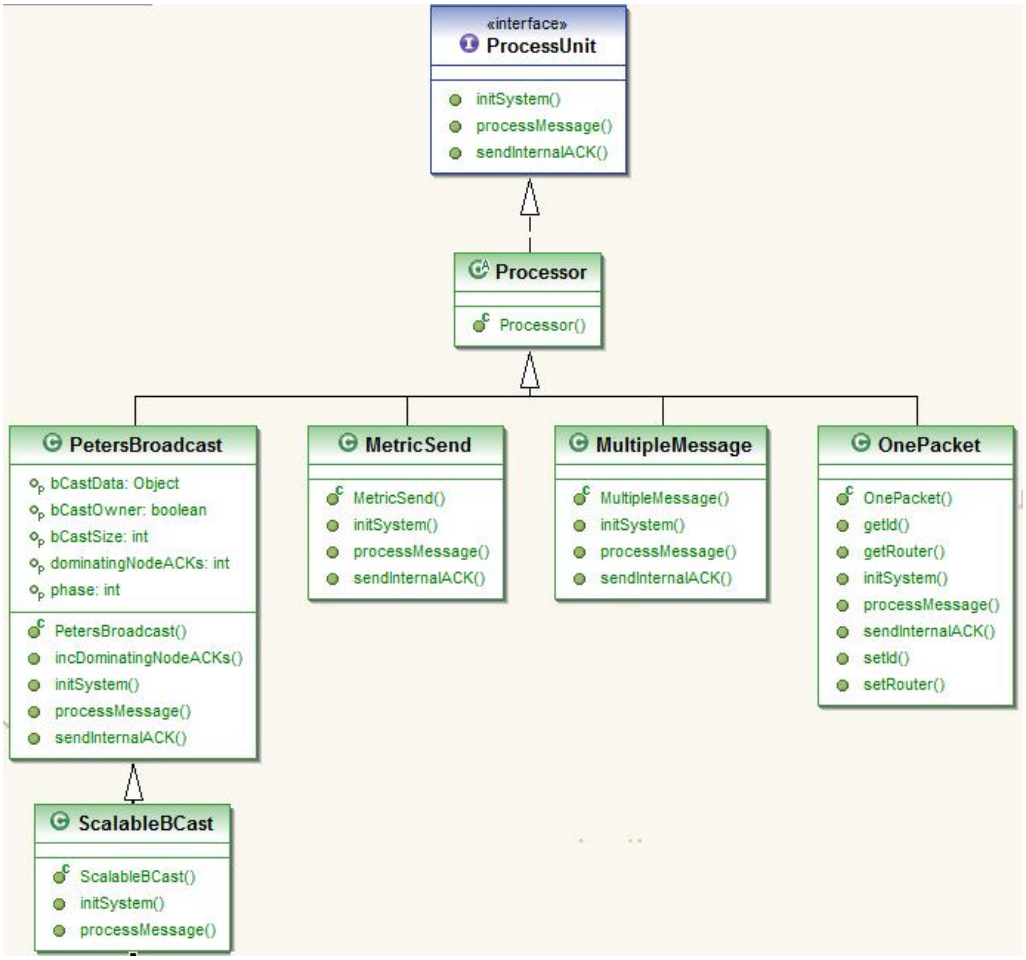
İşlemci, ağ benzetimi üzerinde gerçekleştirilen algoritmayı modeller. Gerçekleştirilen algoritmaya bağlı olarak gerektiğinde diğer işlemcilere ileti gönderir ve gelen iletileri işler. İşlemci üzerinde benzetim zamanı yönetimi yapılmamıştır.

Benzetim içinde kullanılacak algoritmanın, işleme birimi arayüzündeki (*ProcessingUnit interface*), *initSystem()* ve *processMessage()* metotlarını gerçekleştirmesi gerekir. *initSystem()* metodu yönlendirici sınıf tarafından, benzetim başlangıcında çağrılır. Her yönlendirici nesnesi kendisiyle eşleşmiş işlemci nesnesinin metodunu çağırır. Bu metot içinde benzetim başlangıcındaki işlemler tanımlanır. Sistem başlangıç maliyetleri bu metot ile hesaplanır. *processMessage()* metodu, yönlendiriciye kendi işlemcisine ileti geldiğinde, yönlendirici tarafından çağrılır. Gelen iletinin, işlenmesi için işlemciye iletilmesi davranışını gerçekleştirir.

Yönlendirici, kendisine atanmış işlemcinin ilettiği bir iletim sona erdiğinde, işlemciye iletiminin tamamlandığını belirtilen bir im gönderir. İm gönderim davranışı, işlemci birimi arayüzündeki *sendInternalACK()* metodu içinde gerçekleştirilir. Bu imler işlemcilerin iletişim protokollerinde, işlemciler arası zaman uyumlaması için kullanılmıştır.

Yönlendirici iletişimi kolaylaştırmak için işlemci soyut sınıfı (*Processor sınıfı*) gerçekleştirilmiştir. Bu soyut sınıf işlemci birimi davranışı gerçekleştirir ve bir yönlendirici birimi içerir.

Tez kapsamında yapılan testlerin ve gerçekleştirilen toplu iletişim algoritmalarının her biri ayrı bir işlemci olarak gerçekleştirilmiş ve ağ üzerinde koşturulacak işlemci birimi çalışma zamanında seçilmiştir. . İşlemci modelinin gerçekleştiriminin örneklenmesi için, Bölüm 6.2'de verilen Peters-Syska algoritmasının gerçekleştirimi (Ek-7)'de verilmiştir. İşlemcilere ait UML çizeneği Şekil 5-16'te gösterilmiştir



Şekil 5-16. İşlem birimi UML çizeneği

6. DURUM ÇALIŞMASI (*Case Study*)

Bu bölümde tez kapsamında gerçekleştirilen *torus* ağı benzetimi üzerinde yapılan testler anlatılacaktır. Durum çalışması kapsamında ilk etapta benzetimin tutarlılığı ve doğru çalışması hakkında testler yapılmıştır. Bu testler çakışma içeren trafik örüntüleri, tek kapılı ve tam kapılı yönlendiricilerde denenmiştir. Bu testlerde *torus* topolojileri için gerçekleştirilen boyut sıralı yönlendirme algoritması kullanılmıştır.

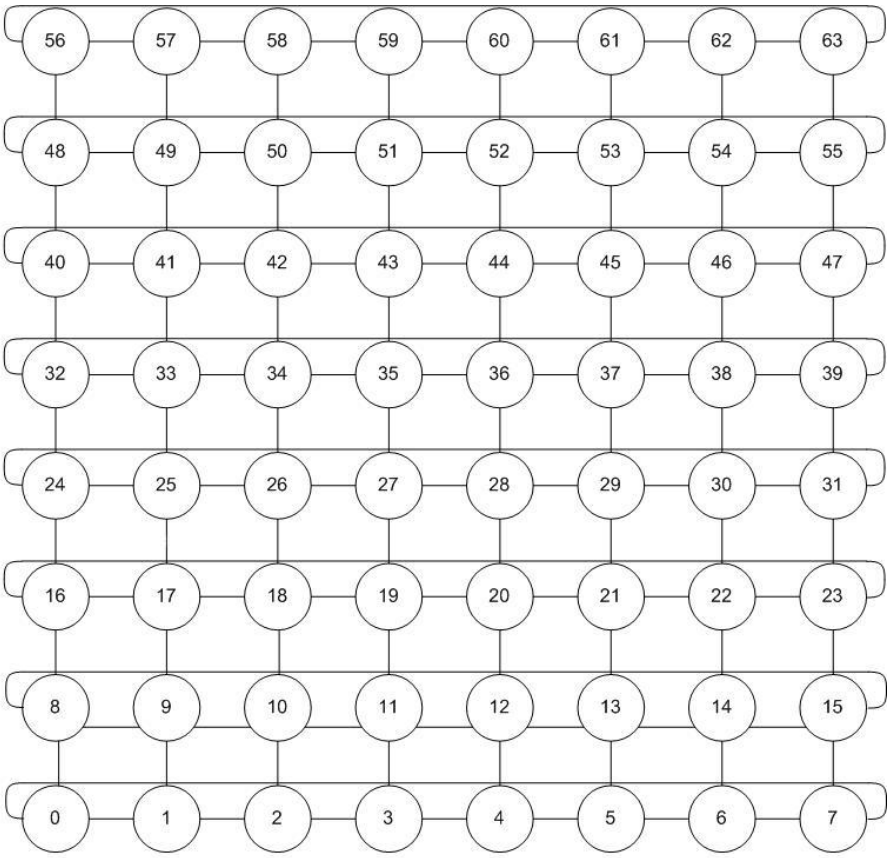
Daha sonra gerçekleştirilen ağ benzetimi üzerinde 5x5 *torus* için iki yayım algoritması gerçekleştirilmiş, bu yayım algoritmaları değişik ileti boyları için test edilmiştir. Test sonuçları yorumlanmıştır.

Durum çalışmalarında yapılan testlerin bütününde *flit* boyu 4 sekizli olarak seçilmiş ve alındı paketlerinin bir *flit* olduğu varsayılmıştır. Anlaşılırlığı artırmak için yönlendiriciler arasındaki kanalların her saat vuruşunda (*cycle*) 4 sekizli ilettiği dolayısıyla her *flit* iletisinin bir saat vuruşunda iletildiği düşünülmüştür.

6.1. Tek ve Tam Kapılı Yönlendiriciler üzerinde çakışma içeren örüntülerin gerçekleştirimi

Gerçekleştirilen ağ benzetiminin tutarlı çalıştığını görmek için, 8x8 *torus* üzerinde, iki ve dört ileti içeren çakışmalı trafiklerin benzetimi yapılmıştır. Testlerde Bölüm 5.2.5'te anlatılan *torus* ağlar için boyut sıralı durağan yönlendirme kullanılmıştır.

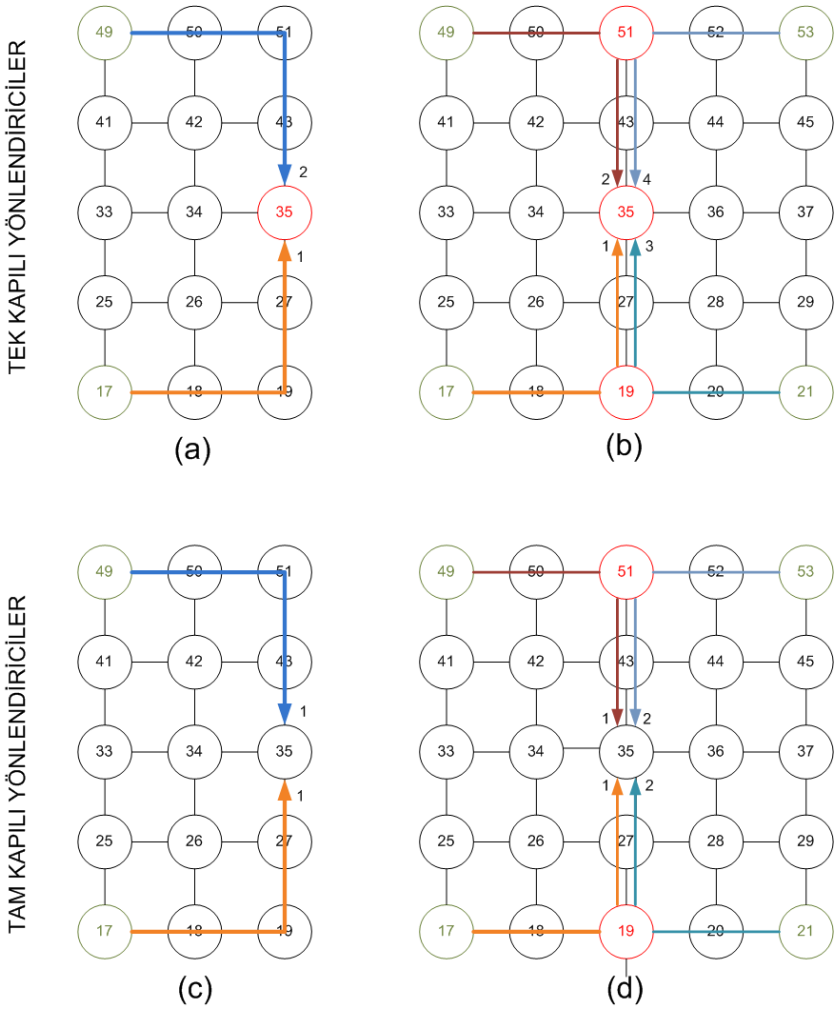
Wormhole anahtarlamada, bir iletiye ait trafik sona ermeden bu trafiğe atanmış kaynaklar başka bir trafiğe atanamaz. Dolayısıyla bir işlemci veya bir yönlendirici bir çıkış kanalı üzerinden aynı anda iki ileti kabul edemez. Tek kapılı yönlendiricide, işlemci yerel veya harici birer trafiği desteklediği için, bir düğüme iki ayrı düğüm veya dört ayrı düğüm ileti gönderdiğinde, iletimler sıralı olarak gerçekleşmektedir. Bu durumda iletimin toplam süresi, ileti sayısıyla doğru orantılı olarak artar.



Şekil 6-1. 8x8 *Torus* çizimi

Tam kapılı yönlendiricide işlemciye eş zamanlı olarak yönlendiricideki çıkış kanalı kadar iletim yapılabilir. Bu durumda birden fazla iletimin aynı işlemciye olduğu durumda, iletiler ara düğümlerde çakışmadıkça aynı anda varırlar.

Test olarak 8x8 *torus* üzerinde iletilerin çakıştıkları senaryolar tek ve tam kapılı yönlendiriciler üzerinde benzetilmiştir. Test uygulanan *torus* Şekil 6-1'de gösterilmiştir.



Şekil 6-2. 8x8 Torus üzerinde çakışmalı iletim örüntüleri

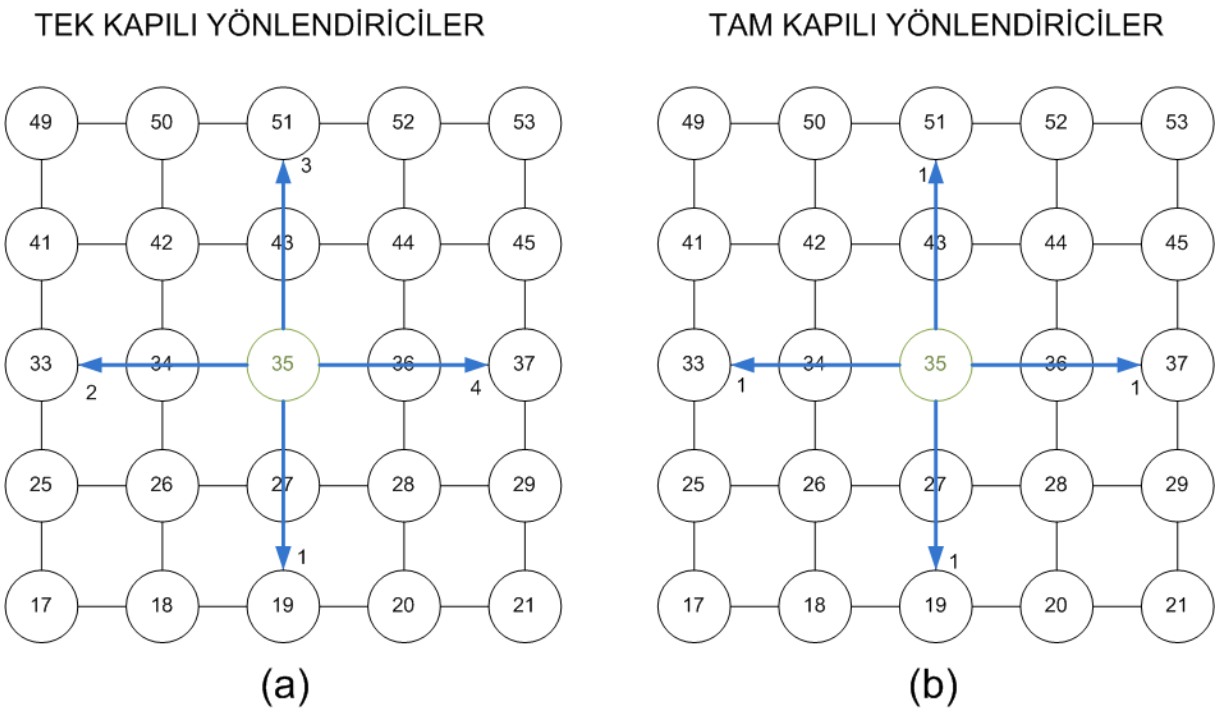
Gerçekleştirilen çakışma senaryoları Şekil 6-2'de gösterilmiştir. Şekil 6-2'de kaynak düğümler yeşil, çakışmanın oluştuğu düğümler kırımızı renkte gösterilmiştir. Her iletim yolu farklı bir renkte gösterilmiş, iletimlerin hedef düğüme erişim sıraları iletim oklarının yanında gösterilmiştir.

Senaryoların ilkinde 17 ve 49 ağ kimlikli düğümler eş zamanlı olarak 35 ağ kimlik düğüme ileti göndermektedir (Şekil 6-2-a, Şekil 6-2-c). Bu iletilerin hedef düğüm üzerinde çakışmaları ve iletimin gözlenmesi amaçlanmıştır. Bu senaryo, tek kapılı yönlendirici gerçekleştiriminde çakışmalara sebep olur (Şekil 6-2-a).

İkinci senaryoda 17, 21, 49 ve 53 ağ kimliklerinin eş zamanlı olarak 35 ağ kimliklerine ileti gönderdiği durum gerçekleştirilmiştir (Şekil 6-2-b, Şekil 6-2-d). Bu durumda, ilk etapta 17 ve 21 ağ kimlikli düğümlerin iletileri 19; 53 ve 49 ağ kimlikli düğümlerin iletileri de 51 ağ kimlikli düğüm üzerinde çakışırlar. Bu iki düğümden

seçilen iki trafik, hedef düğüm olan 35 ağ kimlikli düğüm üzerinde çakışırlar. Bu senaryoda; tam kapılı gerçekleştirimde hedef düğüm üzerinde çakışma olmaz (Şekil 6-2-d); fakat tek kapılı yönlendiricide hem hedef düğüm hem de geçiş düğümleri (51 ve 49) üzerinde çakışmalar olur (Şekil 6-2-b).

Gerçekleştirilen üçüncü senaryo tek ve tam kapılı yönlendiriciler için Şekil 6-3'te gösterilmiştir. Senaryoda, 35 ağ kimlikli düğüm 33, 37, 51 ve 19 ağ kimlikli düğümlere çoklu gönderme yapar. Bu senaryoda ise tek kapılı yönlendiricide dört ileti çakışır ve sıralanır (Şekil 6-3-a); fakat tam kapılı yönlendiricide tüm iletiler eş zamanlı iletilirler (Şekil 6-3-b).



Şekil 6-3. 8x8 *Torus* üzerinde çoklu gönderme

Wormhole anahtarlama modeli için ileti aktarım süresi Denklem (3.2) ile ifade edilmiştir. Denklemde iletisi L ikiliden ve bir başlık *flit* iletisinden oluşan paketin aktarımı hesaplanmıştır. Denklemde t_r yönlendirme gecikmesi, t_s anahtar yayılım gecikmesi ve t_w kanal yayılım gecikmesidir.

Yapılan testlerde anlaşılırlığı kolaylaştırmak için; anahtar içi ve yönlendirme gecikmeleri göz ardı edilmiştir. *Flit* boyunun kanal sığasına eşit olduğu ve her bir saat vuruşunda bir *flit* aktarımının tamamlandığı varsayılmıştır. Bu durumda L

ikiliden oluşan iletinin, aralarında *D* kanal olan iki düğüm arasındaki aktarım maliyeti Denklem (6.1) ile ifade edilir:

$$t_{wormhole} = D + n_{flit}$$
$$n_{flit} = \left\lceil \frac{L}{W} \right\rceil \quad (6.1)$$

Bu bölümdeki testlerin bütününde 44 sekizliden, yani 11 *flit* iletisinden oluşan bir ileti kullanılmıştır. İletiyi içeren paket bir başlık *flit* içermektedir. İletiler *"İleti-[Hedef Düğüm Ağ kimliği]"* içermektedir. Test sonuçları Çizelge 6-1'de gösterilmiştir.

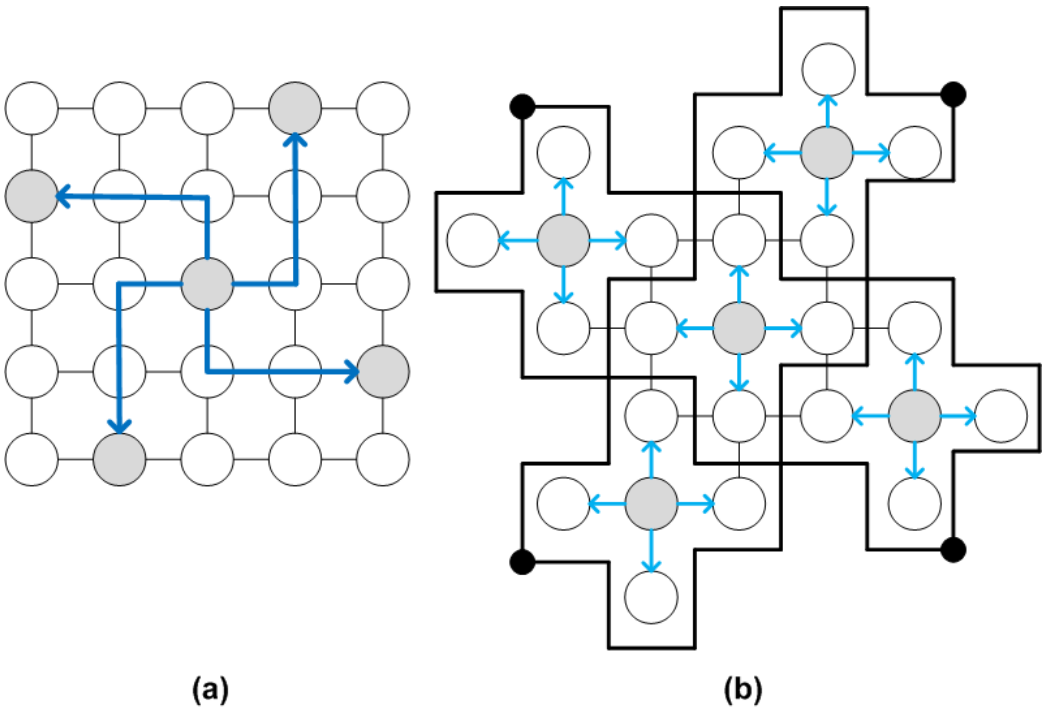
TEK KAPILI YÖNLENDİRİCİ				**TAM KAPILI YÖNLENDİRİCİ**			
İki ileti içeren trafik için iletim zamanları				İki ileti içeren trafik için iletim zamanları			
Hedef Adresi	**İleti**	**Kaynak Adresi**	**Zaman(Saat vuruşu)**	**Hedef Adresi**	**İleti**	**Kaynak Adresi**	**Zaman(Saat vuruşu)**
35	İleti-17	17	15.0	35	İleti-17	17	15.0
35	İleti-49	49	23.0	35	İleti-49	49	15.0
Dört ileti içeren trafik için iletim zamanları				Dört ileti içeren trafik için iletim zamanları			
Hedef Adresi	**İleti**	**Kaynak Adresi**	**Zaman(Saat vuruşu)**	**Hedef Adresi**	**İleti**	**Kaynak Adresi**	**Zaman (Saat vuruşu)**
35	İleti-17	17	15.0	35	İleti-17	17	15.0
35	İleti-49	49	23.0	35	İleti-49	49	15.0
35	İleti-21	21	32.0	35	İleti-21	21	23.0
35	İleti-53	53	40.0	35	İleti-53	53	23.0
Çoklu Gönderme				Çoklu Gönderme			
Hedef Adresi	**İleti**	**Kaynak Adresi**	**Zaman(Saat vuruşu)**	**Hedef Adresi**	**İleti**	**Kaynak Adresi**	**Zaman(Saat vuruşu)**
19	İleti-35	35	13.0	37	İleti-35	35	13.0
33	İleti-35	35	24.0	33	İleti-35	35	13.0
51	İleti-35	35	35.0	51	İleti-35	35	13.0
37	İleti-35	35	46.0	19	İleti-35	35	13.0

Çizelge 6-1: İletim örüntüleri için test sonuçları

Çakışmayla karşılaşmayan iletilerin iletim maliyetlerinin, Denklem (6.1) ile uyuştuğu görülmektedir. Fakat çakışmadan dolayı bekleyen iletilerin, trafiği ele geçirdikten sonra ilk iletiden daha kısa sürede iletildiği görülmektedir. Bunun sebebi çakışan iletime ait *flit* iletilerini, ara düğümlerin yastık sığaları kadarını kabul ederek çakışan ve çakışmayan iletimlerinin bir kısmının eş zamanlı yürütülmesidir. Yönlendiricilerdeki yastık boyları arttıkça, bekleyen ara düğümlerdeki yastıklarda saklanan ileti sayısı artacak ve çakışan iletiler hedef düğüme daha çok yaklaşacaktır. Bu durum çakışan iletilerin iletim sürelerini azaltır. Bu bölümde yapılan testlerde dış giriş kanalları için yastık boyunun 5 olduğu varsayılmıştır.

6.2. Peters - Syska (1-Hepsi) Yayım Algoritmasının Gerçekleştirimi

Peters- Syska, devre anahtarlamalı *torus* ağları için etkin bir yayım algoritması önermiştir (Peters, 1996). Bu algoritma 5^kx5^k*torus* ağları üzerinde çalışmaktadır. Algoritma iki adımda çalışmaktadır. İlk adımda yayım yapan düğüm baskın düğümlere iletiyi göndermektedir (Şekil 6-4-a). İkinci adımda her baskın düğüm komşu düğümlerine iletiyi göndermektedir (Şekil 6-4-b). Bahsedilen iletim örüntüsü Şekil 6-4'te gösterilmiştir.



Şekil 6-4. Peters-Syska yayım algoritması iletim örüntüsü

Yayım yapılacak iletinin boyunun *L* olduğu durumda, N = 5^kx5^k*torus* üzerinde yayım süresi Denklem (6.2) ile ifade edilir (Peters, 1996).

$$(log_5 N)\alpha + (\sqrt{N} - 1)\delta + (log_5 N)L\tau \quad (6.2)$$

Burada α iletinin başlangıç (kurulum) maliyeti, δ bir ara düğüme, bir *flit* anahtarlama maliyeti, $\frac{1}{\tau}$ ise kanal aktarım sığasıdır (Peters, 1996). *k*=1 için 5x5 *torus* üzerinde yayım maliyeti Denklem (6.3) ile ifade edilir.

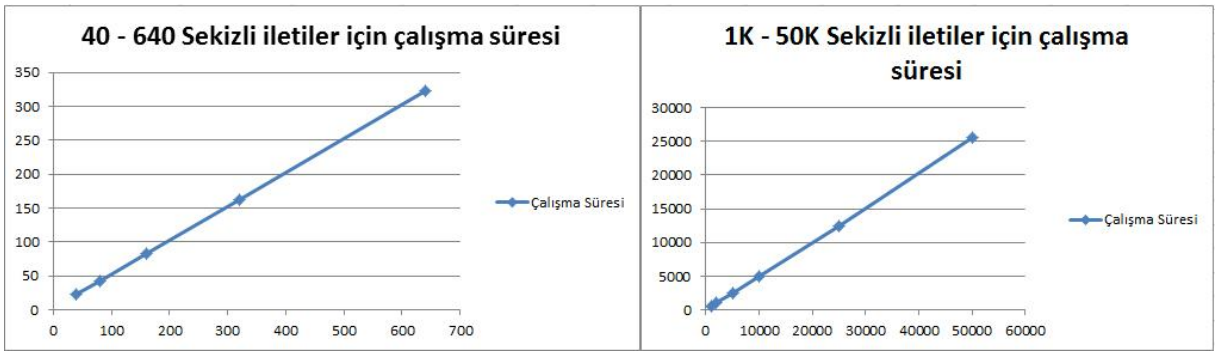
$$2\alpha + 4\delta + 2L\tau \quad (6.3)$$

Durum çalışması olarak Peters – Syska algoritmasının 5x5 *torus* üzerinde *wormhole* anahtarlama ve kredi tabanlı akış kontrolü kullanarak benzetimi gerçekleştirilmiştir. Test senaryosunda α başlangıç maliyetiyle, δ iletim maliyetinde anahtar için beklemeler göz ardı edilmiş ve sadece *flit* iletim maliyeti dikkate alınmıştır. *Flit* iletim süresi bir saat vuruşu ve dış giriş kanallarındaki yastık boyları 5 olarak seçilmiştir.

Gerçekleştirilen yayım algoritması, farklı ileti boyları için test edilmiştir. Testlerde bir başlık *flit* iletisi içeren paketler kullanılmıştır. Çalışma süresi olarak iletiyi en geç alan düğüme iletinin ulaşma süresi kullanılmıştır. Farklı boylardaki paketlerin toplam yayım iletim zamanları Çizelge 6-2'de gösterilmiştir. Paket boyu – iletim zamanı dağılımı grafiği Grafik 6-1'de gösterilmiştir.

PETERS - SYSKA YAYIM ALGORİTMASI			
Paket Boyu (Sekizli)	Çalışma Süresi (Saat Vuruşu)	Paket Boyu (Sekizli)	Çalışma Süresi (Saat Vuruşu)
40	23	2000	1003
80	43	5000	2503
160	83	10000	5003
320	163	25000	12503
640	323	50000	25503
1000	503		

Çizelge 6-2. Peters - Syska yayım algoritması ileti boyları için iletim zamanları



Grafik 6-1. Peters - Syska yayım algoritması ileti boyu iletim zamanı dağılımı grafiği

Yapılan testlerdeki çıkan sonuçların, denklem (6.3) ile verilen teorik değerden bir saat vuruşu (iletim süresi) fazla olduğu görülmektedir. Bunun sebebi baskın düğümlerin yayım sahibi düğümden aldıkları ileti yolundaki komşu düğümlerine, birinci adımın sonunda yolladıkları alındı paketleriyle, bu düğümlere ikinci adımda yolladıkları veri *flit* iletilerinin çakışmasıdır.

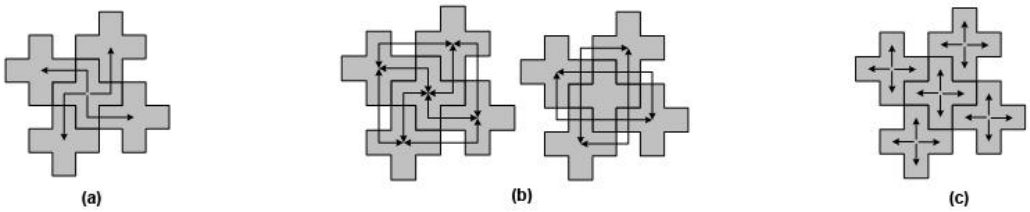
Peters-Syska algoritmasının algoritma düzeyinde gerçekleştirimi (Ek-7)'de anlatılmıştır.

6.3. Ölçeklenebilir (1- Hepsi) Yayım Algoritmasının Gerçekleştirimi

Bu durum çalışmasında, ölçeklenebilir yayım algoritmasının 5x5 *torus* için olan yayım algoritması gerçekleştirilmiştir (İmre, 2011).

Ölçeklenebilir yayım algoritması, üç adımdan oluşmaktadır: *Baskın düğümlere çoklu gönderme, baskın düğümler arası değişim* ve *komşu düğümlere çoklu gönderme*. 5x5 *torus* topolojisindeki bir ağda, baskın düğümler arası çoklu gönderme çakışmalardan dolayı iki adımda gerçekleşmektedir.

Ölçeklenebilir yayım algoritmasının alt iletişim adımları Şekil 6-5'te gösterilmiştir.



Şekil 6-5. Ölçeklenebilir gönderimler için alt iletişim adımları

(a= Baskın düğümlere çoklu gönderme, b= Baskın düğümler arası değişim, c= komşu düğümlere çoklu gönderme)

Gerçekleştirilen algoritmada ileti boyu *L* olan bir iletinin, aktarım sığası *(1/τ)* olan kanallarda iletimini ele alalım. Algoritmanın ilk adımında, yayım sahibi düğüm iletiyi 5 parçaya böler ve her baskın düğüme (*L/5)* boyunda ileti parçasını iletir (Şekil 6-5-a). Baskın düğümler arası çoklu gönderme sonunda her düğümde iletinin (*L/5*) boyundaki kısmı bulunur. Bu adım 5x5 *torus* topolojisindeki gerçekleştiriminde Bölüm 6.3'te anlatılan algoritma ile benzerlik gösterir. İlk adım bitince, baskın düğümlerin her biri diğer baskın düğümlerle veri değişimi yaparlar. Böylece iletinin tamamı her bir baskın düğümde toplanmış olur. 5x5 *torus* topolojisinde değişim adımı, çakışmalardan dolayı tek adımda gerçekleştirilemez. Bu adımın ilk kısmında yakın baskın düğümler değişim yapar, ikinci kısımda uzak düğümler değişim yapar(Şekil 6-5-b). Son adımda her baskın düğüm, komşu düğümlerine iletinin tamamını iletir (Şekil 6-5-c).

Yayım yapılacak iletinin boyunun *L* olduğu ve α ile δ değeri için ara düğüm anahtarlama maliyetinin göz ardı edildiği durumda, N = 5x5 *torus* üzerinde bu algoritma için yayım süresi Denklem (6.4) ile ifade edilir. Burada d iletim örüntüsünde geçilmesi gereken kanal sayısını, $L\tau$ ise L ikiliden oluşan iletinin iletim maliyetini göstermektedir. Denklem (6.4)'teki *d* değişkenleri başlık *flit* iletisinin iletim süresini göstermektedir. d_{bg} başlık *flit* iletisinin baskın düğümlere iletim süresi, d_{d1} baskın düğümler arasındaki değişim adımının ilk adımdaki iletim süresi,

d_{d2} baskın düğümler arasındaki değişim adımının ikinci adımdaki iletim süresi ve d_{kg} baskın düğümlerin komşu düğümlerine iletim süresidir.

$$t_y = (d_{bg} + \frac{L\tau}{5}) + (d_{d1} + \frac{L\tau}{5}) + (d_{d2} + \frac{L\tau}{5}) + (d_{kg} + L\tau) \qquad (6.4)$$

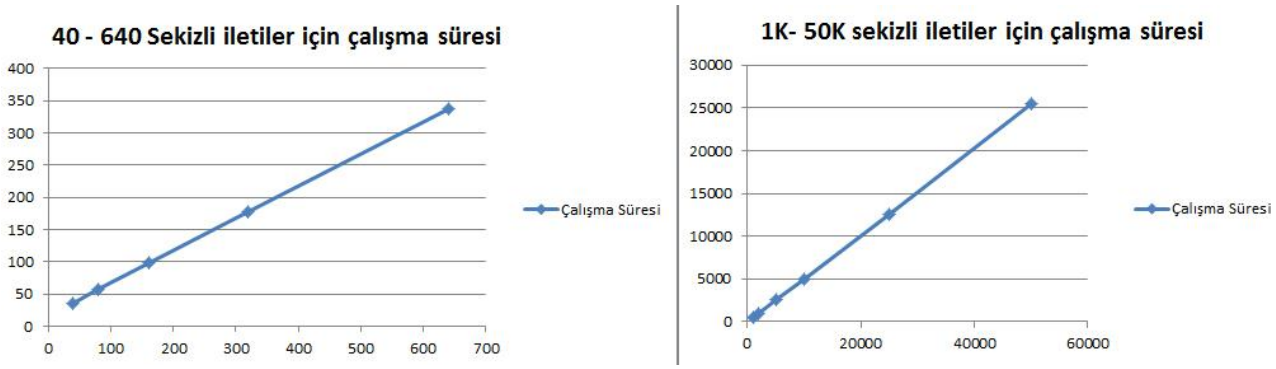
Durum çalışması olarak ölçeklenebilir yayım algoritmasının 5x5 *torus* üzerinde *wormhole* anahtarlama ve kredi tabanlı akış kontrolü kullanarak benzetimi gerçekleştirilmiştir. Test senaryosunda α değeri ve δ*flit* iletim maliyeti içindeki ara düğüm anahtarlama maliyeti göz ardı edilmiştir. Yönlendiriciler içindeki yastık boyları 5 olarak seçilmiştir.

Farklı ileti boyları için gerçekleştirilen yayım algoritması test edilmiştir. İletim süresi olarak, iletiyi son alan düğüme iletinin ulaşma zamanı dikkate alınmıştır. Testlerde farklı boylardaki iletiler için bir başlık *flit* iletisi içeren paketler kullanılmıştır. Çalışma süresi olarak iletiyi en geç alan düğüme iletinin ulaşma süresi kullanılmıştır. Farklı ileti boyları için yayım toplam iletim zamanları Çizelge 6-3'te gösterilmiştir.

ÖLÇEKLENEBİLİR YAYIM ALGORİTMASI			
Paket Boyu (Sekizli)	Çalışma Süresi (Saat Vuruşu)	Paket Boyu (Sekizli)	Çalışma Süresi (Saat Vuruşu)
40	36	2000	1017
80	58	5000	2510
160	98	10000	5018
320	178	25000	12512
640	337	50000	25512
1000	518		

Çizelge 6-3. Ölçeklenebilir yayım algoritması ileti boyları için iletim zamanları

İleti boyu – iletim zamanı dağılımı grafiği Grafik 6-2'de gösterilmiştir.



Grafik 6-2. Ölçeklenebilir yayım algoritması ileti boyu iletim zamanı dağılımı grafiği

Denklem (6.4)'deki ifadeye göre; her flit iletisinin bir saat vuruşunda gönderildiği ağda, *L* ikiliden oluşan iletinin yayım süresi Denklem (6.5) ile gösterilmiştir.

$$t_y = \frac{3L\tau}{5} + L\tau + 14 \qquad (6.5)$$

Benzetim sonuçlarında, beklenen teorik değerden 6 saat vuruşundan başlayan ve doğrusal biçimde artan bir sapma görülmektedir. Bunun sebebi, baskın düğümler arası değişim adımlarında, kredi alındı paketlerinin karşılıklı trafiklerde darboğaza sebep olmasıdır. Bu deney gerçekleştirilen yayım algoritması için, akış kontrol yöntemi olarak kredi tabanlı akış kontrolünün başka bir yaklaşımını veya farklı bir akış kontrolü yapısının tercih edilmesi gerektiğini göstermektedir.

7. SONUÇLAR

Tez kapsamında çok işlemcili ağ benzetimi gerçekleştirilmiştir. Gerçekleştirilen ağ benzetimi *torus* topolojisindedir ve dolaysız ağ özelliklerini içermektedir. Benzetim koşut olarak gerçekleştirilmiştir. Benzetim içinde görevler arası zaman uyumlama için boş ileti algoritması kullanılmıştır. Yapılan durum çalışmalarıyla ağ benzetiminin çalışması test edilmiş ve sonuçları incelenmiştir. Sonuçlar ağ benzetimin doğru çalıştığını göstermiştir.

Tez kapsamında yapılan ağ benzetimi, koşut biçimde gerçekleştirilmiş ve bütün mantıksal görevler işletim dizileriyle çalıştırılmıştır. Benzetim java programlama diliyle gerçekleştirilmiştir ve java dilinin koşut kütüphaneleri ve java işletim dizileri yoğun biçimde kullanılmıştır.

Yapılan tez çalışması, çok işlemcili mimariler için etkin bir benzetim aracını ortaya çıkarmıştır. Bu araç çok işlemcili mimariler üzerinde koşut algoritmaların ve iletim örüntüsü yaklaşımlarının test edilmesini ve sonuçlarının incelenmesini kolaylaştırmıştır. Bu araç özellikle analitik olarak sonuçlarının çıkarılmasının zor olduğu çoklu gönderim algoritmalarında etkin sonuçlar sağlayabilmektedir. Ağ benzetiminde, üst düzey ve alt düzeydeki ağ birimlerinin davranışları analiz edilmiş ve arayüzlerle tanımlanmıştır. Benzetim uygulamasındaki birimler arasındaki ilişkiler arayüzler seviyesinde kurulmuş ve bütün birimler diğerlerinin davranışlarını kullanmıştır. Böylece ağ benzetiminin ilerde geliştirilmesi ve yeni birimlerin eklenmesi kolaylaştırılmıştır.

Ağ benzetimi, daha önceden gerçekleştirilen benzetim altyapısı üzerine inşa edilmiştir. Sağlanan altyapı, çok işlemcili ağ benzetimleri dışındaki diğer benzetim uygulamalarının da boş ileti zaman uyumlu koşut benzetim altyapısı kullanılarak gerçekleştirilmesine imkan sağlar.

Tez çalışmasında, her bir benzetim uygulaması bir yönlendirici ve işlemci çiftiyle eşleşmiştir. Bu yaklaşım sadece tek işlemcili ağ düğümü örüntüsünü destekler. İlerleyen çalışmalarda, kullanıcının tanımlayacağı daha değişik ağ düğümü örüntülerini desteklemek üzere benzetim uygulaması katmanının geliştirilmesi gerekmektedir.

Ağ benzetiminde 8x8 iki boyutlu *torus* topolojisindeki ağların benzetimlerine kadar test edilmiştir. Yapılacak çalışmalarla, gerekli topoloji kurucu sınıfın geliştirilmesiyle, benzetim ağının üç boyutlu *torus* başta olmak üzere çok işlemcili ağlarda tercih edilen diğer dikey topolojileri de desteklemesi sağlanmalıdır.

Gerçekleştirilen ağ mimarisinde modern koşut sistemlerde en çok tercih edilen anahtarlama yöntemlerinden biri olan *wormhole* anahtarlama modeli tercih edilmiştir. Sistem *wormhole* anahtarlama yöntemleriyle uyumlu durağan ve kaynak kararlı yönlendirme modelleri ile kredi tabanlı akış kontrol sistemlerini desteklemektedir. Ağ benzetimine özellikle sistem etkinliklerini karşılaştırılması için; paket anahtarlama yönteminin de eklenmesi, bununla birlikte daha karmaşık yönlendirme yaklaşımları ile büyük yastık boyunun tercih edildiği sistemlerde kullanılan açık/kapalı akış kontrol yaklaşımının gerçekleştirimleri sağlanmalıdır.

Gerçekleştirilen ağ benzetiminde, ağ içinde oluşacak hatalar göz ardı edilmiş ve sistemin tam doğru ve etkin çalıştığı düşünülmüştür. İleriki çalışmalarda hata toleranslı ağlar için benzetim çalışmaları yapılmalıdır.

KAYNAKLAR

[1] Athas, W.C., Seitz, C.L.,1988, Multicomputers: Message-passing concurrent computers, IEEE Computer, vol. 21, no. 8, pp. 9–24.

[2] Bhuyan, L.M., Agrawal D.P., 1984, Generalized hypercube and hyperbus structures for a computer network, IEEE Transactions on Computers, vol. C-33, no. 4, pp. 323–333.

[3] Chandy, K.M., Misra, J., 1978, Distributed Simulation: A Case Study in Design and Verification of Distributed Programs, IEEE Transactions on Software Engineering SE-5(5): 440-452.

[4] CISL, 2009, Cray T3D süper bilgisayar mimarisi, http://www.cisl.ucar.edu/computers/gallery/cray/t3d.jsp.

[5] Close, P., 1986, The iPSC/2 node architecture, In Proc. of the Conference on Hypercube Concurrent Computers and Applications, p. 43–55.

[6] CSIM, 2011, http://www.csim.com/.

[7] Dally, W.J., Seitz, C.L.,1987, Deadlock free message routing in multiprocessor interconnection networks, IEEE Transactions on Computers, 36(5):547–553.

[8] Dally, W.J., 1990, Performance analysis of k-ary n-cube interconnection networks, IEEE TC, 39(6):775–785.

[9] Dally, W.J., 1990, Virtual-channel flow control, In Proc. of the International, Symposium on Computer Architecture (ISCA), pages 60–68.

[10] Dally, W.J., Aoki, H., 1993, Deadlock-free adaptive routing in multicomputer networks using virtual channels, IEEE Transactions on Parallel and Distributed Systems, 4(4):466–475.

[11] Dally, W.J., Towles, B.P., 2004, Principles and Practices of Interconnection Networks, Morgan Kaufmann, 550s.

[12] Dao, B.V., Duato, J., Yalamanchili, S., 1995, Configurable flow control mechanisms for fault-tolerant routing, Proceedings of the 22nd International Symposium on Computer Architecture, pp. 220–229.

[13] DMSO, 2000, High Level Architecture Run-Time Infrastructure RTI 1.3-Next Generation Programmer’s Guide.

[14] Duato, J., 1996, A necessary and sufficient condition for deadlock-free routing in cut-through and store-and-forward networks, IEEE Transactions on Parallel and Distributed Systems, 7(6):841–854.

[15] Duato, J., 1997, A Theory of Fault-Tolerant Routing in Wormhole Networks, IEEE Trans.on Parallel and Distributed Systems, Vol. 8,No. 8.

[16] Duato, J., Yalamanchili, S., Ni L.M., 2002, Interconnection networks, Morgan Kaufmann, 624s.

[17] Flynn, M., 1972, Some Computer Organizations and Their Effectiveness, IEEE Trans. Comput. C-21: 948.

[18] Fujimoto, R.M., 2000, Parallel and Distributed Simulation Systems, Wiley-Interscience, 320s.

[19] Fujimoto, R.M., 2005, Parallel & Distributed Simulation Systems: From Chandy/Misra to the High Level Architecture and Beyond, http://homepages.laas.fr/alex/fujimoto.ppt.

[20] Gamma, E., Helm, R., 1994, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Professional, 416s.

[21] Glass, C.J., Ni, L.M., 1992, The turn model for adaptive routing, In Proc. of the International Symposium on Computer Architecture (ISCA), pages 278–287.

[22] Goetz, B., Peierls, T.,2006, Java Concurrency in Practice, Addison-Wesley Professional, 384p.

[23] Intel, 1985, iPSC User's Guide, Intel 17455-03, Portland, Oregon.

[24] Intel, 2005, Moore Yasası, http://download.intel.com/museum/Moores_Law/Printed_Materials/Moores_Law_2pg.pdf.

[25] İmre, K.M., 2010, Hacettepe Üniversitesi Bilgisayar Mühendisliği Bölümü BIL710 Koşut ve Dağıtılmış Benzetim Sistemleri ders notları.

[26] İmre, K.M., Baransel C., Artuner H., 2011, Efficient and Scalable Routing Algorithms for Routing Algorithms for Collective Communication Operations on 2D All-Port Torus Networks, International Journal of Parallel Programming.

[27] Kermani, P., Kleinrock, P., 1979, Virtual-cut through: a new computer communications switching technique, Computer Networks, 3(4):267–286.

[28] Kessler, R.E., Schwarzmeier, J.L., 1993, Cray T3D: a new dimension for Cray Research, In Proc. of the IEEE Computer Society International Conferrence (COMPCON), pages 176–182.

[29] Kotapati, K., Dandamudi, S.P., 1999, Buffer Management in wormhole routed torus multicomputer networks, Future generation Computer Systems, pp. 483-491.

[30] Linder, D.H., Harden J.C., 1991, An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes, IEEE Transactions on Computers, vol. C-40, no. 1, pp. 2–12.

[31] McKinley, P.K., Trefftz, C., 1993, MultiSim: A simulation tool for the study of largescale multiprocessors, Proceedings of the International Workshop on Modeling, Analysis, Simulation of Computer and Telecommunication Systems, pp. 57–62.,

[32] Ni, L.M., McKinley, P.K., 1993, A survey of wormhole routing techniques in direct networks, IEEE Computer, vol. 26, no. 2, pp. 62–7.

[33] Nuth, P.R., Dally, W.J., 1992, The J-machine network. In Proc. of the International Conference on Computer Design, pages 420–423.

[34] Oed, W., 1993, The Cray Research Massively Parallel Processing System: Cray T3D, Cray Research.

[35] Oracle, 2011, Java Koşut Programlama Kütüphanesi dokümanı, http://download.oracle.com/javase/tutorial/essential/concurrency/

[36] Oracle, 2011, Java işletim dizileri, http://download.oracle.com/javase/1.4.2/docs/api/java/lang/Thread.html

[37] Perumalla,K.S., 2006, Parallel and Distributed Simulation: traditional techniques and recent advances, Winter Simulation Conference.

[38] Peters, J.G., Syska, M., 1996, Circuit-switched broadcasting in torus networks, Parallel and Distributed Systems, IEEE Transactions on , vol.7, no.3, pp.246-255.

[39] Pinkston, T.M., 1999, Flexible and Efficient Routing Based on Progressive Deadlock Recovery, IEEE Transctions on Computers Vol.48 No.7.

[40] Rawany, S., 1995, Routing in Wormhole Networks, Doktora Tezi, Department of Computer Science, University of Saskatchewan.

[41] Schildt, H., 2002, Java 2: The Complete Reference, McGraw-Hill Osborne Media, 1184s.

[42] Schwetman, H., 1986, CSIM: A C-Based Process-Oriented Simulation Language, Winter Simulation Conference.

[43] Seitz, C.L., 1985, The cosmic cube, CACM, vol. 28, pp. 22-33.

[44] Stunkel, C.B., Hochschil, P.H., 1994, SP2 high-performance switch architecture, In Proc. of the Symposium on Hot Interconnects, pages 115–121, Stanford.

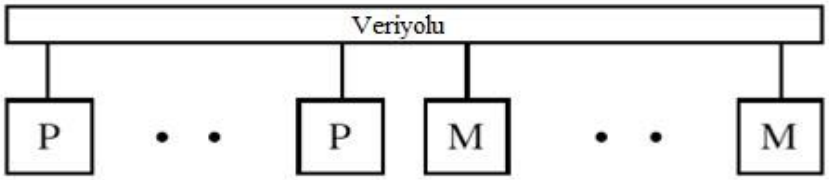
[45] Yih-Jia Tsai, McKinley, P.K., 1996, A broadcast algorithm for all-port wormhole-routed torus networks , Parallel and Distributed Systems, IEEE Transactions on , vol.7, no.8, pp.876-885..

EKLER

1. Bileşen Bağlantılı Ağ Sınıfları

1.1. Paylaşımlı Ortam Ağları

Paylaşımlı ortam ağlarında, sistem üzerindeki bütün işlemciler aynı iletişim ortamını paylaşırlar. Bu ağlarda, ağ içindeki aygıtların hepsi, iletişim ortamına adres ve veri bilgilerini geçebilmek için gerekli devre elemanları barındırırlar.



Şekil E-1. Bus network modeli

(P=işlemci, M= bellek)

Bu ağlardaki en önemli problem, iletişim ortamındaki ağ çakışmalarını engelleyecek ve ortamın, kaynağı paylaşan elemanlar arasında etkin kullanımını sağlayacak bir hakemlik stratejisi (*arbitration strategy*) belirlemektir. Bununla birlikte, ağ bileşenlerinin hepsi ağ ortamını dinledikleri için, yayım (*broadcast*) gibi toplu iletişim protokollerinin gerçekleştirimi kolaylaşır. Paylaşımlı ortam ağları, bileşen sayısı arttığında iletişim ortamında oluşacak darboğazlardan dolayı genişlemeye ve ölçeklendirilmeye uygun değildir (Duato, 2002).

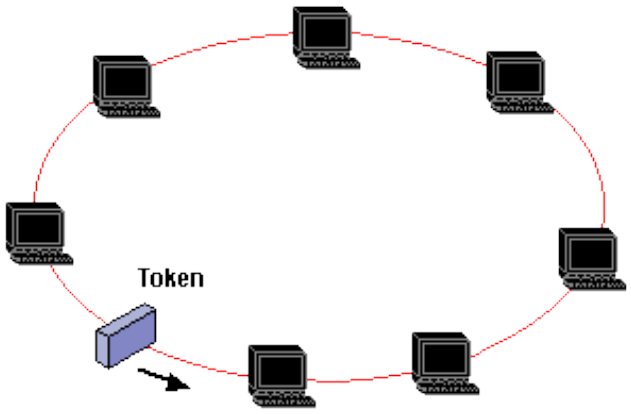
Paylaşımlı ortam ağlarının en bilinen uygulamaları, Çizelge 2-1'de belirtildiği üzere, *Paylaşımlı Ortam Yerel Alan Ağları* ve *Backplane bus* ağlarıdır.

Paylaşımlı Ortam Yerel Alan Ağlarının, *Contention Bus, Token Bus* ve *Token Ring* uygulamaları bulunur. *Contention Bus* modelinde, aygıtlar arasında iletişim ortamını elde etmek için yarış vardır. Ağ üzerindeki aygıtların hepsi iletişim ortamını dinlerler ve ilk olarak ortamı ele geçiren, iletisi bitene kadar ortamı elinde tutar. Bu ağlarda iletişim ortamının üç durumu vardır: *Uygun, Meşgul* ve *Çakışma*. Çakışma durumunda birden fazla aygıtın ortamı kullandığı anlaşılır ve ortamı elinde tutan aygıtlar ortam hakkını serbest bırakıp tekrar denerler. *Contention Bus* modelinde,

ortamı ele geçiren aygıtın, ortamı ne zaman bırakacağına dair durağan bir bilgi yoktur. Dolayısıyla bu ağlar gerçek zamanlı uygulamalara uygun değildir (Duato, 2002).

Token Bus modeli, *contention bus* modelindeki sorunlara alternatif olarak önerilmiştir. Bu modelde, ağ aygıtları arasında bir *bilet (token)* geçmektir. Jetonu elde eden aygıt, ortamı kullanma hakkını kazanır. İleti geçişi tamamlandıktan sonra, jeton belirlenen zamanlama disiplinlerine göre bir sonraki aygıta geçer.

Token Ring modeli *Token Bus* modelinin doğal bir genişlemesidir. Bu modelde, ortam hakları için yine bilet yaklaşımı kullanılır fakat aygıtlar *ring* topolojisinde bağlanmışlardır. Şekil E-2'de *Token Ring* modeli gösterilmiştir.



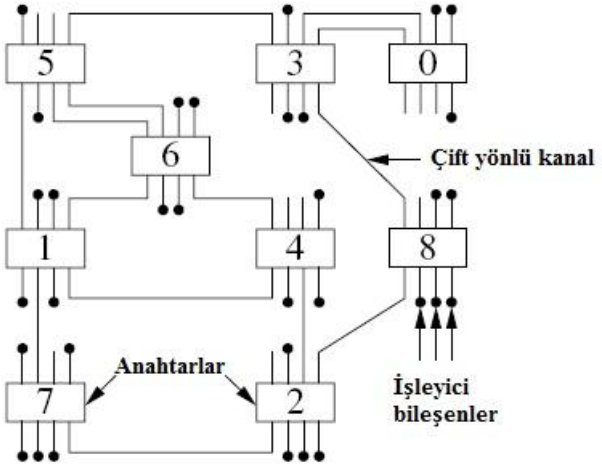
Şekil E-2. Token Ring modeli

1.2. Dolaylı Ağlar

Dolaylı veya *Anahtar tabanlı ağlar* bileşen bağlantılı ağların diğer bir temel sınıfıdır. Bu ağlarda, dolaysız ağlardan farklı bir biçimde, düğümler arasında dolaysız bağlantılar kurulmaz. Dolaylı ağlarda, herhangi iki düğüm arasındaki iletişim, ağ ortamında bulunan bir veya daha fazla sayıda anahtar üzerinde geçerek yapılır. Dolaylı ağların genel yapısı Şekil E-3'te betimlenmiştir.

Dolaylı ağ mimarisindeki koşut sistemlerde de, dolaysız ağlara benzer biçimde, değişik sayıdaki düğümün belirlenen bir dolaylı ağ topolojisine göre bağlanmasından meydana gelir. Dolaylı ağ mimarisindeki düğümler, dolaysız ağlarda kullanılan düğümlere benzer özellikler gösterirler, fakat dolaylı ağ düğümlerinde yönlendiriciye ihtiyaç olmaz. Dolaylı ağ düğümlerinin her biri bir ağ bağdaştırıcısına sahiptir ve bu

bağdaştırıcıyla sistem alan ağında bulunan bir anahtara bağlanır. Bu anahtarlar değişken sayıda kapıya sahiptir. Her kapı bir giriş ve çıkış kanalı çiftini barındırır. Her anahtarda bulunan kapılar her biri ya sistemde bulunan bir düğüme (başka bir deyişle işleyici) , ya da sistem içinde bulunan başka bir anahtara bağlantı içerir. Bu anahtarların bağlantı biçemi, çeşitli ağ topolojileri tanımlar (Duato, 2002).



Şekil E-3. Dolaylı ağ mimarisi (Duato, 2002)

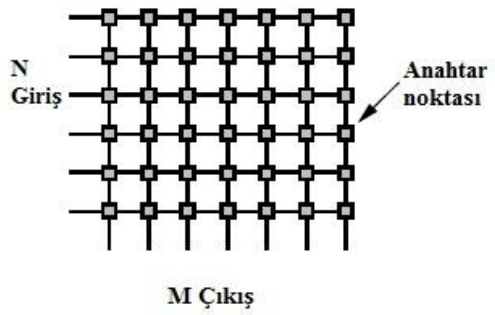
Dolaylı ağ topolojileri ilerleyen zamanla dikkate değer bir biçimde gelişmiştir ve bu ağ sınıfları için birçok ağ topolojisi önerilmiştir. Önerilen bu ağ topolojileri, dağıtılmış bellekli bilgisayarlar, iş istasyonları ve dizi işleyicileri içeren geniş bir sistem alanını kapsar.

Dolaylı ağlar *G(N,C)* modelinde çizge ile tanımlanabilirler. Bu çizgedeki *N* sistem alan ağındaki anahtarlar kümesini ve C anahtarlar arasındaki tek veya çift yönlü kanallar kümesini betimler. Ağ üzerindeki birçok özelliğin analizi için, bu çizgeye işleyicilerin dahil edilmesine ihtiyaç duyulmaz (Duato, 2002).

Dolaylı ağlarda bir iletinin bir düğümden diğer bir düğüme iletiminde; iletinin sırasıyla kaynak düğümle bağlı olduğu anahtarla arasındaki kanaldan, iki düğüm arasındaki anahtarlardan ve en son olarak hedef düğümle bağlı olduğu anahtar arasındaki kanaldan geçmesi gerekir. Dolayısıyla, iki düğüm arasındaki mesafe, bu düğümlerin bağlı oldukları anahtarlar arasındaki mesafe artı iki kanal toplamı kadar olacaktır. Benzer şekilde iki düğüm arasındaki en uzun mesafe, düğüm anahtarları arasındaki en uzun mesafe artı iki kanal geçiş maliyeti olarak belirtilecektir. Bu bağlamda, dolaylı

bir ağda aynı anahtara bağlı iki işleyici arasındaki iletim mesafesi sıfır değil ikidir (Duato, 2002).

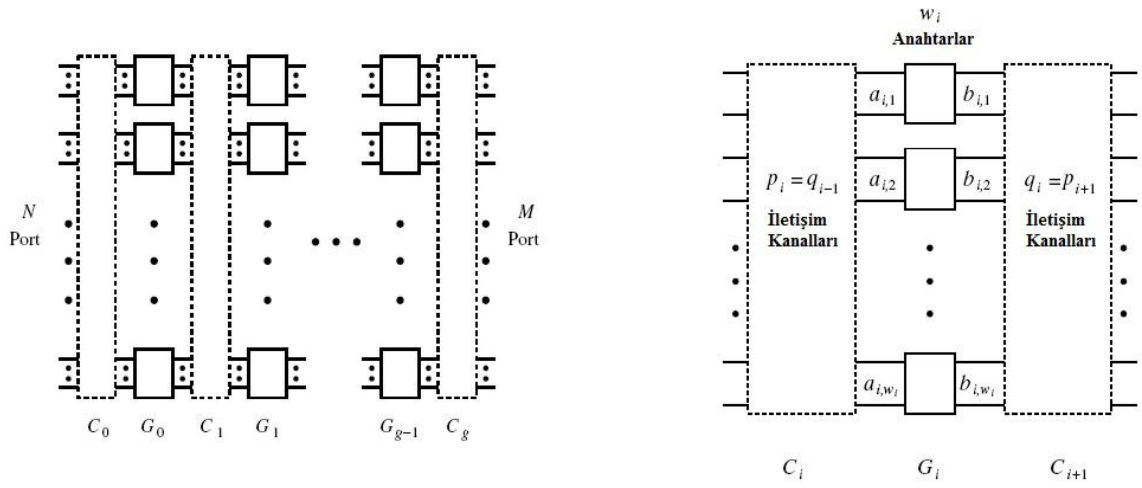
Dolaysız ağlara benzer biçimde, dolaylı ağlar da temel olarak üç temel etkenle tanımlanırlar: *topoloji, yönlendirme* ve *anahtarlama*. Topoloji, dolaylı ağ içindeki anahtarların birbirleriyle nasıl bir modelde bağlandıklarını tanımlar. Dolaylı ağlar için ideal ağ topolojisi, *çapraz bağlantılı ağlar* olarak tanımlanır. *N* tane düğüm içeren bir ağ için, ağdaki tüm düğümleri bir *NxN* anahtar (çapraz anahtar) ile bağlamaktır. Bu topolojide düğümler arası mesafede anahtar gezme maliyeti olmayacak ve herhangi iki düğüm için iletim mesafesi iki olacaktır. Önerilen bu ağ topolojisi, benzer yapıda bir dolaysız ağ topolojisine göre (N tane yönlendirici ve her yönlendirici için bir *NxN* çapraz anahtar) çok daha düşük maliyetlidir. Fakat donanımsal kısıtlardan ötürü bir anahtarın destekleyeceği bağlantı sayısı sınırlıdır. Dolayısıyla bu topolojinin yüksek sayıda işleyici içeren koşut sistemler için gerçekleştirimi olanaksızdır. Şekil E-4'te bir çapraz bağlantılı ağ modeli gösterilmiştir. Gösterilen çapraz bağlantılı anahtar N girişi ve M çıkışı desteklemektedir.



Şekil E-4. NxM çapraz bağlantılı anahtar (Duato, 2002)

Çapraz bağlantılı ağların fiziksel sınırları sonucunda, dolaylı ağlar için birçok alternatif ağ topolojisi modeli önerilmiştir. Bu topolojilerde, ileti veya iletiyi içeren paketler, hedef düğüme ulaşmadan önce bir veya birkaç anahtardan geçmek zorundadır. Önerilen topolojilerden bilinen bir sınıf Düzenli Ağlardır. Bu ağlarda, anahtarlar belli düzeylerde düzenlenmiştir. Bu düzeylerden her biri için giriş kanalları bir önceki düzeye ait çıkış kanallarını ve her düzeye ait çıkış kanalları bir sonraki düzey için giriş kanallarını oluşturmaktadır. Bu ağlar ayrıca Çok Düzeyli Bileşen Bağlantılı Ağlar (*Multistage Interconnection Networks, MIN*) olarak da tanımlanırlar (Duato, 2002).

IBM SP2 süper bilgisayarı çok düzeyli bileşen bağlantılı ağ topolojisinde gerçekleştirilmiştir (Dally, 2004, Stunkel, 1994).



Şekil E-5. Genelleştirilmiş MIN modeli (Duato, 2002)

Şekil E-5'de genelleştirilmiş bir MIN modeli gösterilmiştir. Anlaşılacağı üzere, model N giriş ve M çıkış bağlantısı desteklemektedir. Şekilde belirtildiği üzere, genel olarak bir MIN modelinin $G_0 \text{dan} G_{g-1}$ 'e kadar gtane düzeyi bulunur. Her düzeyde, düzeyleri G_i olarak varsayarsak, w_i tane anahtar bulunur. Her anahtar a tane giriş kanalını ve b tane çıkış kanalını destekler. Daha önce belirtildiği üzere, her düzey için giriş bağlantı sayısı (Şekil E-5'teki düzeyde p_i), bir önceki kanala ait çıkış bağlantı sayısına (Şekil E-5'teki düzeyde q_{i-1}) eşittir (Duato, 2002).

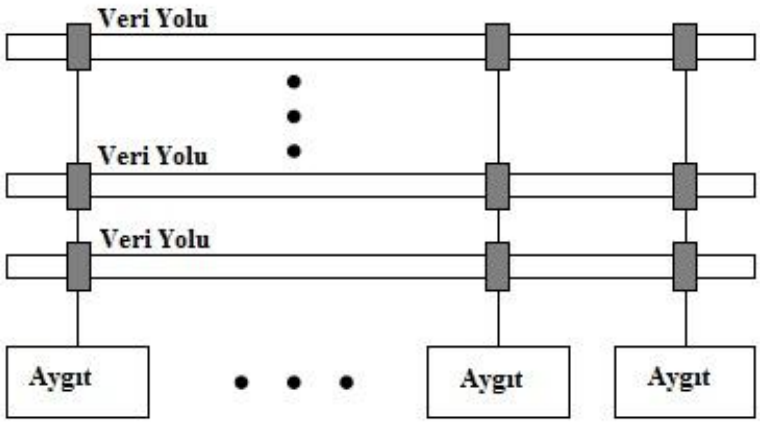
Dolaylı ağların tanımlanmasında önemli diğer iki etken yönlendirme ve anahtarlamadır. Yönlendirme, ağ içindeki bir iletim sırasında takip edeceği yolun belirlenmesi işlemleridir. Anahtarlama ise, iletim sırasında ağ kaynaklarının (kanallar ve yastıklar vb.) nasıl atanacağını belirler.

1.3. Melez Ağlar

Melez ağlar, Paylaşımlı Ortam Ağları, Dolaylı ve Dolaysız ağların özelliklerini ve mekanizmalarını birleştirirler. Böylece, ağın aktarım sığasını paylaşımlı ortam ağlarına göre yükseltirler ve düğümler arasındaki mesafeleri dolaylı ve dolaysız ağlara göre azaltırlar. Bununla birlikte, yüksek performans gerektiren sistemlerde dolaylı ve dolaysız ağlar melez ağlara göre çok daha başarılı ölçeklenebilirlik gösterirler. Çünkü noktadan noktaya kanallar, paylaşımlı alan veri yollarına göre daha

basit ve daha hızlıdırlar. Melez ağların, köprülü yerel alan ağları gibi iyi kurulmuş uygulamaları vardır.

Yüksek başarımlı koşut bilgisayarların çoğu dolaylı veya dolaysız ağları kullanırlar. Bununla birlikte, fiber optik teknolojilerin gelişmesi, yüksek performanslı veri yollarının gerçekleştirimine olanak vermiştir ve melez ağların kullanımı için yeni alanları mümkün kılmıştır (Duato, 2002 .

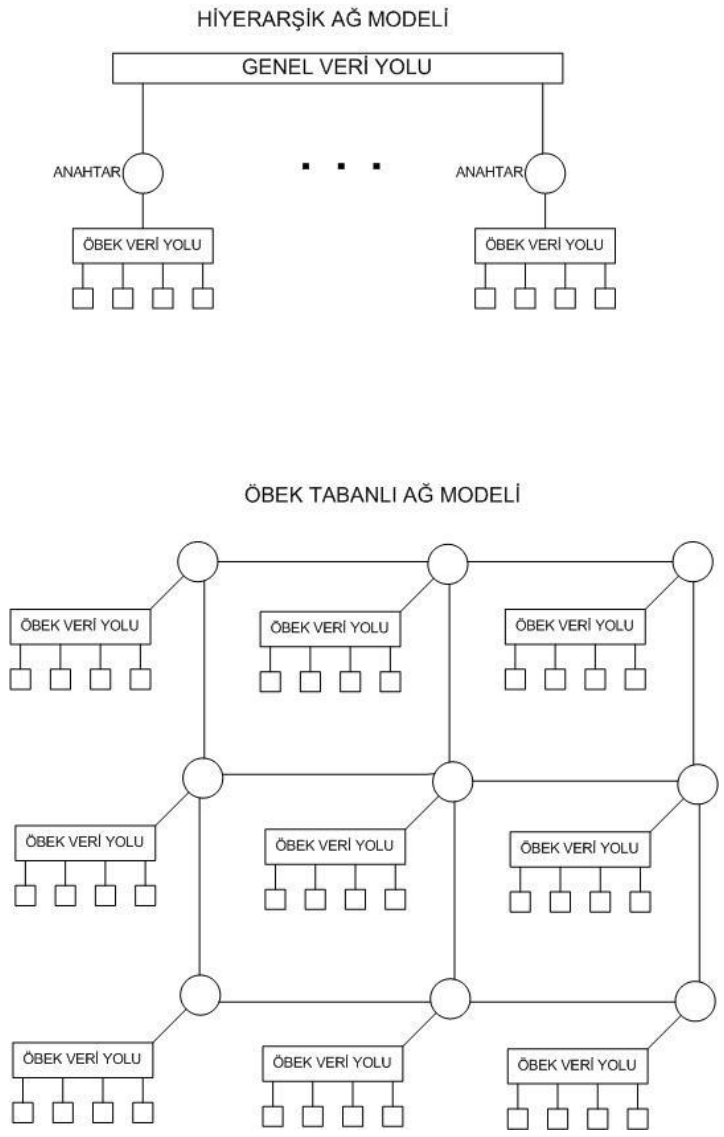


Şekil E-6. Çoklu veri yolu modeli (Duato, 2002)

Farklı amaçlarla bugüne kadar birçok melez ağ modeli önerilmiştir. Genel olarak, bir melez ağ bir hiper çizge ile tanımlanabilir. Tanımlanan hiper çizgede, köşeler işleyiciler kümesini ve kenarlar iletişim kanallarını ve/veya veri yollarını betimler. Hiper çizgelerde, bir köşenin birden fazla düğüme bağlanabileceği unutulmamalıdır. Bir köşe sadece iki düğüme bağlıysa, o köşe bir kanalı aksi takdirde bir veri yolunu betimler. Bazı ağ modellerinde, her düğüm için bir veri yolu sürülür. Bir düğüme atanmış veri yoluna diğer düğümlerin veri sürmesine izin verilmez. Bu ağlarda veri yolu kaynak yönetimi için herhangi bir hakemlik algoritmasına ihtiyaç duyulmaz (Ni, 1993). Bu ağ modeli tasarımında, veri yolu aktarım sığası artırılarak olası darboğazların engellenmesi ve ağın destekleyebileceği düğüm sayısının artırılması amaçlanmıştır. Çoklu veri yolu ağ modeli Şekil E-6'da gösterilmiştir.

Paylaşımlı ortam ağları ve dolaylı/dolaysız ağların birleştirildiği diğer yaklaşımlar *hiyerarşik ağlar* ve *öbek tabanlı ağlar* modelleridir. Bu modellerde ağ bileşenleri hiyerarşik bir yapıda inşa edilir. Ağ modelinin alt bileşenleri paylaşımlı ortam ağı özellikleri taşır ve bu alt bileşenler kendi içlerinde sahip oldukları veri yolu üzerinden

iletişim sağlarlar. Ağ içinde değişken sayıda bulunan bu alt bileşenlerin, aralarında haberleşmeleri ise dolaylı veya dolaysız ağlara benzer biçimde gerçekleştirilir. Başka bir deyişle bu ağlar üst seviyede dolaylı/dolaysız ağ özellikleri gösterirler. Öbek tabanlı ve hiyerarşik ağ modelleri Şekil E-7'de örneklenmiştir.



Şekil E-7. Hiyerarşik ve Öbek tabanlı ağ modelleri

2. Topoloji Türleri

Bir ağ topolojisinde, düğümler yalnız dikey n boyutlu bir uzayda düzenlenebiliyorsa ve her bağlantı sadece tek bir boyutta yer alıyorsa, bu topoloji *dikey topoloji (orthogonal topology)* olarak tanımlanır. Ticari koşut sistemlerde kullanılan topolojilerin birçoğu dikey topolojiler içerir. Bu topolojilerde düğümler arası uzaklıkların her boyuttaki koordinatlarının mutlak değerlerinin toplamı olması, uzaklıkların hesaplanmasını kolaylaştırır.

Dikey topolojiler daha ileri düzeyde *sıkı dikey topolojiler* ve *zayıf dikey topolojiler* olarak sınıflandırılırlar (Duato, 2002). Sıkı düzey topolojilerde, her düğümün her boyutta en az bir bağlantısı (kanalı) bulunur. Zayıf dikey topolojilerde bazı düğümlerin bazı boyutlar boyunca hiç bağlantısı bulunmaz. Dolayısıyla zayıf dikey topolojide bağlanan ağlarda, düğümler her boyuttan geçişi desteklemezler. Eksik bağlantılı düğümlere geçiş başka boyutlardan dolaşmayı gerektirir. Sıkı dikey topolojiler, topolojinin her yerinde aynı yapıya sahiptir. Bu nedenle bu topolojiler *düzenli topolojiler* olarak da tanımlanırlar.

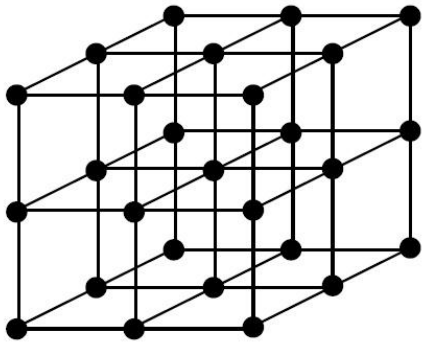
2.1. Hasır Doku (Mesh)

Genel olarak, n boyutlu bir hasır doku topolojisi, $k_i \geq 2$ olmak üzere, her *i.* boyutunda k_i tane düğüm, toplam $k_0\ x\ k_1\ x \ldots x\ k_{n-2}\ x\ k_{n-1}$tane düğüm içerir. Hasır doku içindeki herhangi bir *x* düğümü, $\sigma_{n-1}(x), \sigma_{n-2}(x) \ldots \sigma_1(x), \sigma_0(x)$olmak üzere, n tane koordinatla tanımlanır. x düğümüne ait herhangi bir $\sigma_i(x)$ koordinatı için $0 \leq \sigma_i(x) \leq k_i - 1$ olmak zorundadır. Hasır dokuda, herhangi *x* ve *y* düğümleri için, denklem (2.1) ile ifade edilen koşul sağlanıyorsa, x ve y düğümleri *komşu düğümlerdir* denir.

$$\forall i\ için\ \sigma_i(x) = \sigma_i(y)\ ve\ \ bir\ değer\ hariç\ \forall j\ değeri\ için$$

$$\sigma_j(y) = \sigma_j(x) \pm 1 \qquad (2.1)$$

Hasır dokuda düğümler topoloji içindeki konumlarına bağlı olarak n ile 2n arasında komşuya sahip olurlar. Şekil E-8'de 3x3x3 3 boyutlu bir hasır doku gösterilmiştir.



Şekil E-8. 3x3x3 3 boyutlu hasır doku topolojisi

MIT (*Massachusetts Institute of Technology*) tarafından gerçekleştirilmiş, J-Machine süper bilgisayarında üç boyutlu hasır doku topolojisi kullanılmıştır (Dally, 2004, Nuth, 1992).

3. Dolaysız Ağlarda Kullanılan Anahtarlama Türleri

Bu bölümde anlatılacak anahtarlama modellerinin karşılaştırılması amacıyla, her anahtarlama biriminde L ikili boyunda iletinin aktarıldığı varsayılacaktır. *phit* ve *flit* boyları eşdeğer ve kanal genişliğine eşit alınacaktır. Yönlendirme başlığı 1 *flit* kabul edilecek, dolayısıyla ileti boyu *L+W* ikili olarak kabul edilecektir. İki yönlendirici arasındaki fiziksel kanalın sıklık değeri *B* Hz, kanal aktarım sığası saniyede *BW* ikili olarak alınacaktır. Anahtarlama modellerinde bir *flit* aktarımının bir saat vuruşunda yapıldığı varsayılacaktır. Bu bağlamda söz konusu kanal için yayılım süresi $t_w = \frac{1}{B}$ varsayılacaktır. Bahsedilecek anahtarlama modellerinin tümünde yönlendirme beklemesi t_r , yerel akış kontrol beklemesi veya anahtarlama gecikmesi, t_s olarak varsayılacaktır. İletilerin *D* tane kanaldan geçtiği varsayılacaktır.

3.1. Devre Anahtarlama

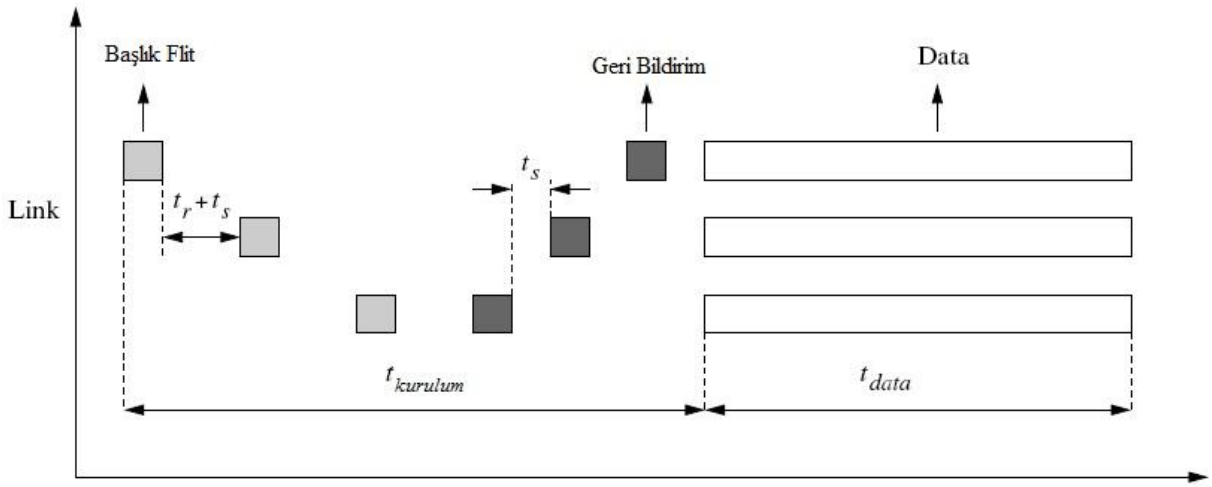
Devre anahtarlamada göndericiyle alıcı arasında fiziksel bir hat oluşturulur ve ileti oluşturulan hat üzerinden, tek hatta gönderiliyormuş gibi gönderilir. Bu hat üzerinden iletim sonlanana kadar başka bir iletinin geçmesine izin verilmez.

Devre anahtarlamada bir iletim oturumu başlangıcında, kaynak düğüm yönlendiricisi hedef düğüm adresi ve kontrol bilgilerini içeren başlık *flit* iletisini ağ ortamına aktarır. Başlık *flit* ileti, geçtiği ara düğümlerde kendi iletimi için gerekli yol üzerindeki kanalları kurarak, kendisine tahsis ederek ilerler. Başlık fliti hedef düğüme eriştiğinde, bütün yol kurulmuş olur ve kaynak düğüme iletiyi hatta vermesi için geri bildirim yollanır. Geri bildirim kaynak düğüme ulaştığında kurulan fiziksel hat, devre, üzerinden ileti tek bir hat üzerinden iletiliyormuş gibi hedef düğüme iletilir. Gönderilen ileti alıcı düğüme ulaştığında, kaynak düğüme gönderilen geri bildirime benzer biçimde, tahsis edilen hat içindeki kanalların serbest bırakılması için bir geri bildirim yollanır ve ara düğümlerdeki kullanılan kanallar serbest bırakılır. Kanalların serbest bırakılmasıyla iletim oturumu sonlanmış olur.

Bu bağlamda, devre anahtarlama üç evreye ayrılabilir: devre kurulum aşaması, alıcı-gönderici arasında kurulan devre hattından iletinin gönderimi ve devre hattının geri bildirimle serbest bırakılması. Eğer devre hattının kurulumunda, istenilen düğümlerden biri kilitli durumda ise, devre bu kanalı ele geçirene kadar bekler. İletim

aşamasında, ileti kurulan hat üzerinden gönderilir. Son olarak devre sonlandırma aşamasında, kurulan devre kaldırılır. Devre anahtarlama modelinin zaman uzayı üzerinde davranışı Şekil E-9'da gösterilmiştir.

Devre anahtarlama modelinin kullanımı, ileti trafiğinin seyrek ve ileti boylarının uzun olduğu uygulamalarda anlamlıdır. Söz konusu uygulamalarda başlık *flit* boyunun, ileti boyuna göre çok uzun olduğu uygulamalar için devre anahtarlama uygun bir seçim olacaktır. Böylece hat kurulum ve serbest bırakma zamanları, iletim zamanına bağlı olarak çok daha kısa olacağı için zamandan kazanç sağlanabilir. Bununla birlikte, ileti trafiğinin sık ve ileti boyunun göreli olarak daha kısa olduğu uygulamalarda devre anahtarlama modeli düşük başarım gösterir. Çünkü bu anahtarlama modelinde her ileti gönderiminde tekrar devre kurulumu yapılır ve hat içindeki bütün kanallar iletim süresi boyunca kilitli durumda kalır. Bu durumda, kilitli kanallara ihtiyaç duyan diğer iletiler iletim süresi bitene kadar beklemede kalacak ve ağ trafiğinde tıkanmalar oluşacaktır.



Şekil E-9. Devre anahtarlamanın zaman ekseninde gösterimi (Duato, 2002)

Devre anahtarlamadaki yönlendirici modelinin, Bölüm 3.2'de bahsedilen yönlendirici modelinden farklı olacağı açıktır. Devre anahtarlamada, genel yönlendirici modelinden farklı olarak, ara düğümlerde sadece başlık *flit* iletileri yastıkta saklanır. Fiziksel hattın kurulumundan sonra ileti yastıklarda saklanmadan doğrudan sonraki düğüme iletilir. Dolayısıyla devre anahtarlamada ileti boyunda yastıklara ihtiyaç kalmaz. Devre anahtarlamada iletim zaman uyumlu veya zaman uyumsuz biçimde

yapılabilir. İletimin başarım ölçütü; zaman uyumlu iletimde saat vuruşu sıklığı, zaman uyumsuz iletimde el sıkışma imlerinin hızı olacaktır. Yüksek im hızlarını destekleyen hatlar üzerinden birden fazla ikili eş zamanlı olarak, veri dalgaları şeklinde yollanabilir. Bu biçimdeki veri transferi *dalga ardışık düzeni (wave pipelining)* olarak tanımlanır (Duato, 2002).

Dalga ardıl düzeni olmadan, devre anahtarlama üzerinde ileti beklemesi Denklem (3.1) ile tanımlanabilir (Duato, 2002).

$$t_{devre} = t_{kurulum} + t_{data}\ s$$

$$t_{kurulum} = D[t_r + 2(t_s + t_w) \quad (3.1)$$

$$t_{data} = \frac{1}{B}\left\lceil\frac{L}{W}\right\rceil$$

Yukarıdaki ifade genelleştirilmiştir bir değerdir ve gerçek beklemeler sayısız gerçekleştirim detayına bağlıdır.

3.2. Paket Anahtarlama

Devre anahtarlama yönteminde, bütün ileti devre kurulduktan sonra iletilir. Alternatif bir yaklaşım olarak iletiler, sabit boydaki (örneğin, 128 sekizli) *paketlere* bölünür. Her paketin ilk bir kaç sekizli değeri paketin ağ içinde yönlendirilmesi için gerekli bilgileri içerir. Paketin yönlendirme ve kontrol bilgilerini içeren bu bölümü *paket başlığı* olarak tanımlanır (Duato, 2002). Bu yaklaşımda, iletinin her paketi ayrı olarak yönlendirilir ve kaynaktan hedefe iletilir. Bu yaklaşıma *paket anahtarlama* denir (Duato, 2002). Paket anahtarlama yönteminde her paket ulaştığı ara düğümde, iletilmeden önce yastıklarda saklanır. Bu sebepten ötürü bu anahtarlama tekniği *sakla ve ilet anahtarlama (store and forward switching)* olarak da tanımlanır.

Paket anahtarlamada, paket başlığının eriştiği her ara düğümün yönlendiricisi tarafından açılır ve paketin hangi çıkış kanalına yönlendirileceğinin kararı verilir.

Paket anahtarlama iletilerin kısa ve ileti trafiğinin sık olduğu uygulamalar için uygundur. Paket anahtarlamada, devre anahtarlamadan farklı biçimde, bir kanal iletim süresince tek iletiye tahsis edilmez. Her paket iletimi ayrı bir iletim oturumu olduğu için, arada farklı iletimler de gerçekleştirilebilir. Bununla birlikte, aynı iletiye ait paketlerin birçoğu, aynı anda ağ alanında bulunabilirler. Böylece ağ alanın aktarım sığası daha etkin kullanılmış olur.

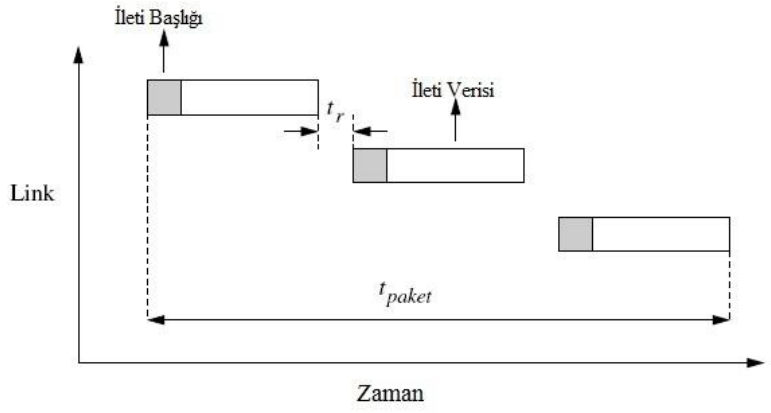
Bir iletiyi paketlere bölerek anahtarlamanın birtakım ek yükleri olur. Öncelikle, her paketin kendisine ait bir iletim oturumu olduğu için, her paketin ayrı olarak yönlendirilmesi gerekir. Bu durum her paket için bir yönlendirme maliyetini beraberinde getirir. Ayrıca, Şekil E-10'da görüldüğü üzere, paket anahtarlamada paketin iletim süresi kaynak ve hedef düğümler arasındaki uzaklığa bağıldır. Devre anahtarlamada ise devre kurulduktan sonra ileti tek bir hat üzerinden gönderildiği için iletim süresi mesafeden bağımsızdır.

Çok boyutlu dolaysız ağlarda, paket anahtarlama kullanıldığında ve paket boyları büyük olduğunda yönlendirici içinde ihtiyaç duyulacak yastık alanının artacağı açıktır. Cosmic Cube (Seitz, 1985) ve Intel iPSC/1 (Close, 1986, Intel, 1985) gerçekleştirimlerinde, paketler düğüm içindeki yerel bellekte saklanmaktadır. Günümüz koşut sistemlerinde, paketlerin yerel belleklerde saklanması mantıksızdır. Modern koşut sistemlerde ağ ortamındaki iletilerin ve ileti paketlerinin bütününün yönlendiricilerde saklanması gerekir. Yastık gereksinimlerini düşürmek için, yönlendirici içinde ortak bir yastık alanı paylaşımlı biçimde kullanılabilir (Duato, 2002). Paket anahtarlama için iletim beklemesi Denklem (3.2) ile belirtilebilir (Duato, 2002).

$$t_{paket} = D\left\{t_r + (t_s + t_w)\left\lceil\frac{L+W}{W}\right\rceil\right\} \qquad (1.1)$$

Bu bağlamda, ifadede L+W ikili uzunluğunda bir paketin iletim süresi hesaplanmaktadır. Söz konusu paket kaynak ve hedef düğümler arasında D tane bağlantıdan geçmektedir.

Paket anahtarlama ile iletimin zaman eksenindeki davranışı Şekil E-10'da gösterilmiştir. Denklem (1.1) Şekil E-10'daki iletim modelini kullanır.



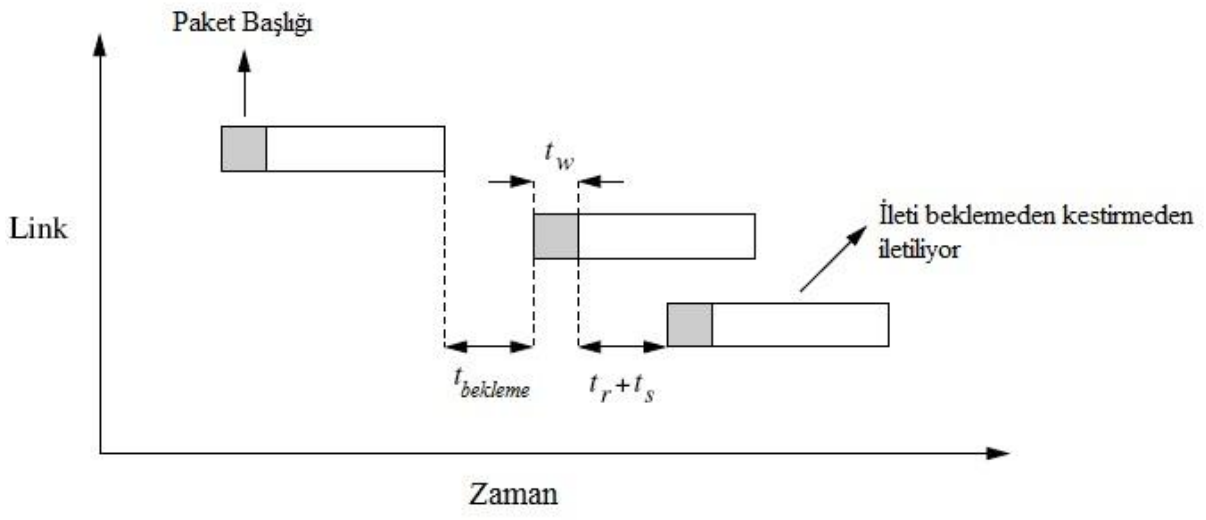
Şekil E-10. Paket anahtarlamanın zaman ekseninde gösterimi (Duato, 2002)

3.3. Sanal Kısa yol Anahtarlama (Virtual Cut-through Switching)

Paket anahtarlamada, paketin yönlendirilmesi ve gerekli dış kanala aktarılması için paketin bütünün alınması gerektiği varsayılır. Fakat bu doğru değildir. Çünkü paketin aktarılması için, paketin başlığının alınması yeterlidir. Paket başlığı ise genelde paketin ilk bir kaç sekizlik bölümünü kapsar. Dolayısıyla, paketinin tamamıyla iletimini ve yastıkta saklanmasını beklemek yerine, paket başlığı alındıktan sonra yorumlanabilir ve gerekli yönlendirme kararına göre gerekli çıkış kanalına iletilebilir. Paket iletimi bitmeden yönlendirme yapıldığı bu anahtarlama tekniği *sanal kısa yol anahtarlama (virtual cut-through switching)* olarak tanımlanır (Dally, 2004, Kermani, 1979).

Engellemelerin olmadığı durumda, sanal kısa yol anahtarlamada sadece paketlerin yönlendirme beklemeleri ve paketin kanallarda aktarımı gecikmelere sebep olur. Sanal kısa yol yönteminde iletilen paket başlığı engellendiğinde (iletinin ihtiyaç duyduğu kanalın kullanımda olma durumu), paket bulunduğu düğümde tamamen yastıkta saklanır ve bekletilir. Bu bağlamda, ağ trafiğinin yüksek olduğu durumlarda, sanal kısa yol yöntemi paket anahtarlama gibi davranır. Şekil E-11'de sanal kısa yol anahtarlama yönteminin zaman ekseninde davranışı gösterilmiştir. Şekilde, paket bir

sonraki kanal kullanımda olduğu için ilk yönlendiriciye tamamen iletilmiş ve kanal uygun olana kadar bu yönlendiricide beklemiştir. Daha sonraki iki kanal uygun olduğu için beklemeden ardıl biçimde iletilmiştir.



Şekil E-11. Sanal kısa yol anahtarlama modelinin zaman ekseninde gösterimi

Bir paketin sanal kısa yol yöntemiyle herhangi bir ara düğüme iletimi için gereken iletim gecikmesi Denklem (3.3) ile hesaplanabilir (Duato, 2002).

$$t_{vct} = D(t_r + t_s + t_w) + \max(t_s, t_w)\left\lceil \frac{L}{W} \right\rceil \quad (1.2)$$

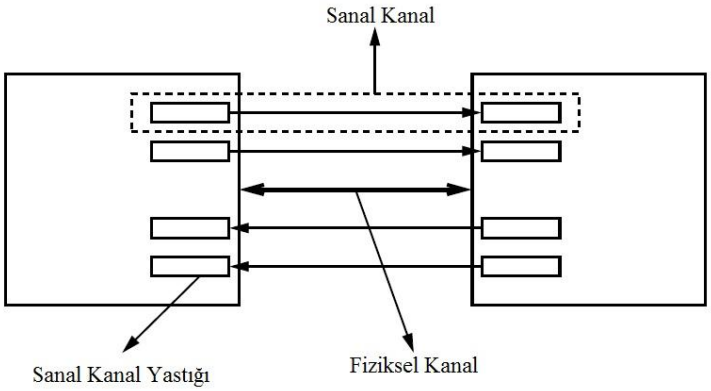
Yukarıdaki ifadede, paket başlığı 1 *flit* boyunda kabul edilmiş ve paketin *D* adet kanaldan geçtiği varsayılmıştır. İfadede genelleştirilmiş yönlendirme modeline eşdeğer biçimde bir *flit* iletisinin aktarım süresi t_w, yönlendirme ve iç akış kontrol beklemeleri sırasıyla t_r ve t_s olarak alınmıştır. İletilen paket, iletim sırasında bir engellemeyle karşılaştığında, ara düğümün gelen kanal yastığında bekletilecek ve istenilen kanal uygun olunca gereken giden kanal yastığına aktarılacaktır. Bu durumda anahtarlama beklemesi t_s paketin aktarım süresi ve bekleme süresinin toplamına eşit olacaktır. İfadede bekleme zamanında t_s ve t_w sürelerinden büyük olan seçilerek beklemelerin ileti gecikmesine dahil edilmesi hedeflenmektedir.

İfadeden anlaşılacağı üzere, ağ tıkanıklıklarının olmadığı durumda VCT anahtarlama ve *wormhole* anahtarlama aynı iletim gecikmesine sahiptir.

Sanal kısa yol anahtarlamada engellenen ileti ağdan çekilerek yastıkta bekletildiği için yüksek ağ etkinliği sağlar. Fakat sanal kısa yol anahtarlamada yastık atamaları paket düzeyinde yapıldığı için büyük yastık boylarına ihtiyaç vardır (Duato, 1996).

3.4. Sanal Kanallar

Açıklanan anahtarlama modellerinin hepsinde, iletiler yönlendiriciler içindeki gelen veya giden kanallarla eşleşmiş yastıklarda saklanıyordu. Bu yastıklar genel olarak FIFO kuyruklar gibi çalışıyordu. Bununla birlikte, bir ileti veya paket bir fiziksel kanalı kullandığı süre boyunca, ileti engellenmiş bile olsa, o kanalı herhangi başka bir ileti kullanamıyordu. Bu durum trafikler arası beklemeleri ve olası kilitlenme risklerini artırmaktaydı. Bununla birlikte, bazı uyarlanır yönlendirme algoritmaları, komşular arasında birden fazla kanalın bulunmasını gerektirir. Komşular arasında birden fazla (tek yönlü veya çift yönlü) fiziksel kanalın kullanılmasının, yönlendirici maliyetini yükselteceği açıktır.



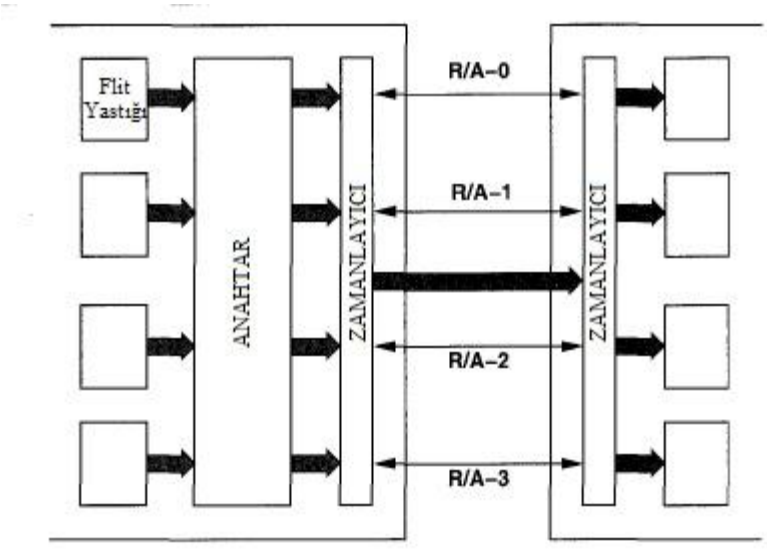
Şekil E-12. Sanal kanallar

Alternatif bir yaklaşımda, bir fiziksel kanal, kendisi üzerinde birden fazla *mantıksal veya sanal kanalı* destekleyebilir. Bu yaklaşımda, kanal yastıkları birden fazla sanal kanala dağıtılır ve birden fazla sanal kanal tek bir fiziksel hattı paylaşır (Dally, 1990b).Sanal kanalların her biri için ayrı *flit* yastığı ve gönderme / alım ikilisi bulunur. Sanal kanal modeli Şekil E-12'de gösterilmiştir. Şekil E-12'de gösterilen modelde, her

yön için iki tek yönlü çalışan kanal, bir çift yönlü fiziksel kanal üzerinde çoklanmıştır.

Sanal kanallar karşılıklı kilitlenmeleri engellemek için önerilmiştir (Dally, 1987). Sanal kanal kullanımının bazı faydaları vardır. İlk olarak sanal kanallar ağ katmanının iletişimsel olarak uyarlanabilmesini artırır ve farklı topolojilere sahip uygulamaların koşut sistemle eşleştirilmesini kolaylaştırır (Ni, 1993). İkinci olarak uygulama ve işlemci grupları aynı topolojilere sahip olsalar bile, fazladan iletişim kapasitesi sağlayan sanal kanallar kullanılarak kapalı iletim trafiklerinde alternatif iletim yolları kullanılabilir. Üçüncü olarak sanal kanalların tek bir fiziksel kanalı kendi aralarında paylaşıyor olması, adil zamanlama algoritmasıyla, aktarım sığasının etkin kullanılmasını sağlar.

Sanal kanallar ile ilgili en büyük sorun, fiziksel kanalın sanal kanallar arasında adil bir şekilde nasıl paylaştırılacağıdır. Aktarım sığasının *W*, *m* tane sanal kanala çoklanan ve aktif *k* kanalın bulunduğu düşünülen bir fiziksel kanal için, *k* tane sanal kanalın her birinin *W/k* etkin aktarım sığasına sahip olması gerekir.



Şekil E-13. Sanal kanallı yönlendirici modeli (Duato, 2002)

Şekil E-13'te sanal kanal altyapısını kullanan bir yönlendirici çizimi verilmiştir. Dört sanal kanalın her biri için *R/A* ikilisi ve *flit* yastığı bulunur. Hangi sanal kanalın fiziksel kanalı kullanacağı zamanlayıcı (*scheduler*) tarafından belirlenir. Aktarım sığasının etkin kullanılması için; alıcı kanallardan sadece *flit* yastığı boş olanlar ve gönderen kanallar için sadece *flit* yastığı dolu olanlar fiziksel kanala zamanlanır (yönlendirilir).

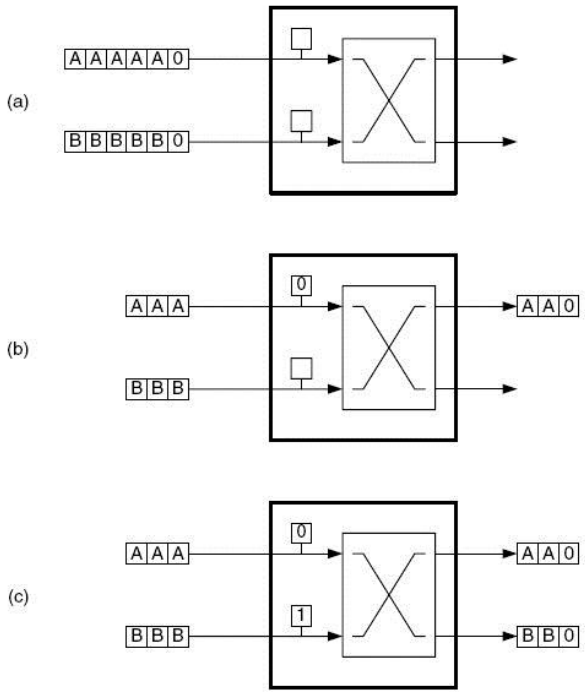
Sanal kanal kullanımın bazı dezavantajları da vardır. Sanal kanal sayısı arttıkça, zamanlama algoritması karmaşıklaşır ve fiziksel kanalın paylaşımının yönetimi için ek donanımlara ihtiyaç duyulur. Ayrıca çok fazla sanal kanal kullanımı ağ beklemelerini artırır ve kanal başarımını düşürür. Örneğin bir fiziksel kanal yönlendirme yolu boyunca üç ayrı paket tarafından paylaşılırsa, bu kanal üzerinde darboğaz olur ve kanala ait aktarım sığası *W/3*'e düşer.

4. Dolaysız Ağlarda Akış Kontrolü Yöntemleri

4.1. Yastıksız Akış Kontrolü

Yastıksız akış kontrolleri yaklaşımında iletişim aygıtları, bekleyen paketlerin depolandığı yastıkları içermezler. Dolayısıyla bu aygıtlarda, paketler arasında paylaşılan ağ kaynakları sadece kanallardır.

Yastıksız akış kontrolü yaklaşımlarında çakışmalar oluştuğunda, kaynakları ele geçiremeyen paket, yönlendirici tarafından yanlış yönlendirilir veya düşürülür. Paketin düşürüldüğü senaryoda, gönderenin paketleri yastıkladığı varsayılır ve gönderen ilgili paket için alındı paketini (*Acknowledgement Packet*) almadıkça, paketi yastıkta saklar ve göndermeye devam eder. Yanlış yönlendirme yaklaşımında ise, kaynağı ele geçiremeyen ileti, boşta olan başka bir düğüme yönlendirilir. Yastıksız akış kontrolü yaklaşımları Şekil E-14'te gösterilmiştir.



Şekil E-14. Yastıksız akış kontrolü yaklaşımları (Dally, 2004)

(a) İletim örüntüsü, (b) İleti düşürülmesi yaklaşımı (c) Yanlış yönlendirme yaklaşımı

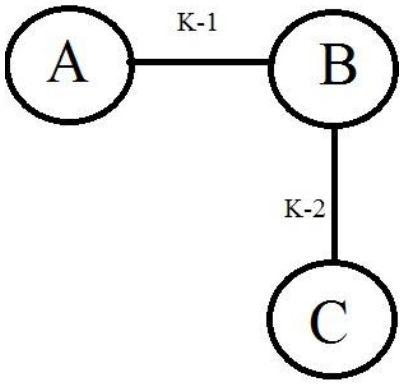
Devre anahtarlama yöntemi, iletimde ara düğümlerde yastıklara ihtiyaç duymadığı için yastıksız akış kontrolünü kullanır. Yastıksız akış kontrolü yaklaşımı, kanal

kaynaklarını kötü kullandığı için, etkinliğin önemli olduğu ağlarda tercih edilmemektedir.

4.2. Yastıklı Akış Kontrolü

Ağ iletişiminde yönlendiricilere yastık eklemek iletişim etkinliğini belirgin düzeyde yükseltir. Çünkü yönlendiricilerde yastık kullanılması, *komşu kanalların* her ikisinin de serbest kalmasını sağlar. Şekil E-15'teki iletim trafiğinde A düğümünden B düğümüne bir paket iletimi olduğunu varsayalım. B düğümünde iletim için bir yastık olmadığı durumda, iletim boyunca K-1 ve K-2 kanalları (komşu kanallar) kullanılacaktır. B düğümünde yastık olduğu durumda ise, K-1 iletimde iken K-2 kanalı, K-2 iletimde iken ise K-1 kanalı kullanılmayacaktır.

Yastıklı akış kontrol yöntemlerinde, akış kontrol biriminin kanallarla birlikte yastıkları da yönetmesi gerekir. Akış kontrol yöntemlerinin çoğu, trafiklere kanal ve yastık atamalarını birlikte yaparlar (Dally, 2004).



Şekil E-15. Komşu kanallar

Yastıklı akış kontrol yöntemleri, kaynakların ileti trafiklerine atanma yaklaşımına göre çeşitlilik gösterirler. Yastıkların ve kanalların paketlere atandığı yöntemler *paket akış kontrol yöntemleri, flit* iletilere atandığı yöntemler ise *flit akış kontrol yöntemleri* olarak tanımlanırlar (Dally, 2004).

İletişim sırasında eğer bir paket engellenirse, bu paketin bir yastık alanı içinde bekletilmesi gerekir. Dolayısıyla alıcı tarafında yeterli yastık alanı olmadığı durumda, akış kontrolü iletimi durdurur ve gereksinim duyulan yastık alanı uygun olduğunda iletimi devam ettirir. Akış kontrolünün olmadığı durumda, yeterli yastık alanı

bulunamayınca iletilen paket düşürülecektir veya gönderen düğüme tekrar yönlendirilecektir.

Yastıklı akış kontrol birimlerinin bütünü, ileti gönderdikleri düğümün yastık durumu hakkında bilgiye ihtiyaç duyar. Alıcı tarafında iletim olmadığı durumda, alıcı düğümün yastıkları bir süre sonra dolacaktır. Bu durumda gönderen düğümün yastık boşalana kadar ileti göndermemesi gerekir.

Alıcı düğümün yastığı dolduğunda gönderen düğüme bildirerek iletimi durdurmasını sağlaması *geri bildirim (backpressure)* olarak tanımlanır (Dally, 2004). Yaygın olarak kullanılan geri bildirimli akış kontrol yöntemleri *kredi tabanlı* (Bölüm 3.4.1)*, açık/kapalı* ve *ACK/NACK* akış kontrolleridir.

Açıklanacak düşük seviye akış kontrolü yöntemlerinde ele alınacak senaryolarda, ağ içindeki *flit* boyunun L_f ikili olduğu ve aktarım sığası *b* olduğu varsayılacaktır.

4.2.1. Açık/Kapalı Akış Kontrolü

Açık/Kapalı akış kontrolünde, düğümler arasında bir kontrol imi bulunur. Bu kontrol imi bir iletim hattı veya alındı paketi (*ACK* paketi) olabilir. Açık/Kapalı akış kontrol yönteminde, gönderen düğüme sadece iletim durumu değiştiğinde durum bildirim yapılır. Açık/Kapalı akış kontrolünde her yastık için doluluk eşik değeri ($F_{kapalı}$) ve boşluk eşik değeri ($F_{açık}$) bulunur. Alıcı düğüm yastıktaki dolu alan sayısı $F_{kapalı}$ değerine ulaştığında "*Kapalı"* imi yollar, yastıktaki boş alan sayısı $F_{açık}$ değerine erişince de "*Açık"* imi yollar (Dally, 2004).

Açık/Kapalı akış kontrolü yöntemi kullanan iki düğüm arasındaki trafik Şekil E-16'da örneklenmiştir. Şekildeki örnekte $Düğüm_2$ iletimi durdurduğu doluluk eşik değerine ($F_{kapalı}$) t_1 anında erişmiştir ve $Düğüm_1$'e "Kapalı" imi göndermiştir. $Düğüm_2$ iletim trafiğini ele geçirdikten sonra karşıya *flit* göndermeye başlamıştır. t_6 anında tekrar ileti alabileceği boşluk eşik değerine ($F_{açık}$) erişmiştir ve "Açık" imini $Düğüm_1$'e göndermiştir.

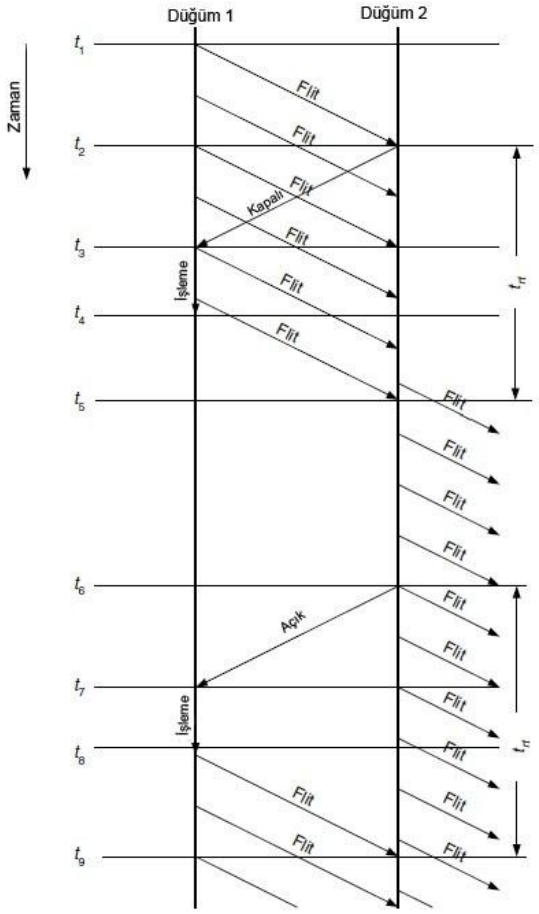
Şekil E-16'da $Düğüm_2$, t_2 anında "*Açık*" imi göndermiş, $Düğüm_1$ bu imi işledikten sonra t_5 anında *flit* gönderimini durdurmuştur. Burada $F_{kapalı}$ için iletim süresi şekilde t_{krt} olarak gösterilmiştir ve ($t_5 - t_2$) olarak hesaplanmıştır. Gönderen düğüm "*Kapalı*" imi alana kadar *flit* gönderimine devam edeceği için, *flit* boyunun L_f ve kanal aktarım

sığası b olduğu senaryoda beklenen $F_{kapalı}$ değeri Denklem (4.1) ile hesaplanmıştır (Dally, 2004).

$$F_{kapalı} \geq \frac{t_{krt} b}{L_f} \tag{2.1}$$

Düğüm$_2$ karşı alıcı için trafiği elde ettikten sonra Düğüm$_1$'e "*Açık*" imini yolladığı t_6 anından, bu im işlendikten sonra Düğüm$_1$ tarafından yollanan *flit* iletisinin Düğüm$_2$'ye eriştiği t_9 anına kadar geçen süre t_{krt} olarak gösterilmiştir ve (t_9-t_6) olarak hesaplanmıştır. Düğüm$_2$'nin "*Açık*" imi yolladıktan sonra karşı tarafa ilettiği *flit* iletilerinin tükenmesi ve karşı kanalı kullanamaması durumunu engellemesi için gereken $F_{açık}$ değeri Denklem (4.2) ile hesaplanmıştır (Dally, 2004).

$$F - F_{açık} \geq \frac{t_{krt} b}{L_f} \tag{2.2}$$



Şekil E-16. Açık/Kapalı akış kontrolün zaman ekseninde gösterimi (Dally, 2004)

4.2.2. ACK/NACK Akış Kontrolü

ACK/NACK akış kontrolü yönteminde, gönderen düğüm alıcı düğümde kendisine atanmış yastık hakkında bir durum bilgisi tutmaz. Alıcı düğüm kabul ettiği her *flit* için gönderene “Alındı” paketi (*ACK* paketi), yastığında yer olmadığı için kabul etmediği her *flit* için “Olumsuz Alındı Paketi” (*NACK* paketi) yollar. Gönderen düğüm, alındı paketi aldığında ilgili *flit* iletisini, yastığından siler ve olumsuz alındı paketi aldığında ise ilgili *flit* iletisini tekrar yollar.

ACK/NACK akış kontrolü, yastık ve kanal kaynaklarını etkin kullanmadığı için yüksek başarım gerektiren çok işlemcili ağlarda kullanılmazlar (Dally, 2004).

5. Dolaysız Ağlarda Yönlendirme Algoritmaları

Yönlendirme algoritmaları, ilk olarak hedef sayısına göre sınıflandırılabilir: tekil *gönderimli yönlendirme (unicast routing)* ve *çoğul gönderili yönlendirme (multicast routing).* Tekil gönderili yönlendirmede, ileti tek bir hedefe gönderilir ve ileti bu bağlamda yol boyu yönlendirilir. Çoğul gönderimli yönlendirmede ileti hedefi tek bir düğüm değil, bir grup düğümü kapsar.

Yönlendirme algoritmaları, bir diğer ölçüt olarak, yönlendirme kararın alındığı yere göre dört sınıfa ayrılır: merkezi *yönlendirme (centralized routing), kaynak kararlı yönlendirme (source routing), dağıtılmış yönlendirme* (*distributed routing*) *ve melez yönlendirme (hybrid routing).*

Yönlendirme kararlarının merkezi bir denetleyici tarafından alındığı yönlendirme yaklaşımı *merkezi yönlendirme (centralized routing)* olarak tanımlanır (Duato, 2002). Bu yönlendirme yönteminde merkezi bir işleyici, bütün iletiler için yönlendirme kararlarını alır. Merkezi yönlendirme SIMD makinelerde tercih edilir.

Bununla birlikte yönlendirme kararı, ileti veya paket ağ ortamına verilmeden önce kaynak düğümde verilebilir. Bu yaklaşım *kaynak kararlı yönlendirme (source routing)* olarak tanımlanır. Kaynak kararlı yönlendirmede, hedef düğüme erişim boyunca takip edilecek iletim yolu ileti başlığında taşınır. Ara düğümler sadece gerekli kanalların ayrılmasından sorumludur. Bu yöntem düzensiz topolojilerde, her düğümün eşit sayıda komşusu olmayan topolojilerde, tercih edilir. Kaynak kararlı yönlendirmede ağ koşulları çok hızlı değiştiği için, bu yönlendirme algoritmalarında uyarlamalı yönlendirme yöntemleri tercih edilmezler.

Kaynak kararlı yönlendirmede, tüm ileti yolu ileti başlığında iletilir ve ağ aktarım sığası tüketilir. Dolayısıyla paket başlığının küçültülmesi gerekir. *Sokak işareti yönlendirme (street sign routing)* bu gereksinimlere cevap verir (Duato, 2002). Sokak işareti yönlendirmede, iletim yolu boyunca gezilecek tüm düğüm adresleri yerine, ilerlenecek yönler ve her yönde gezilecek düğüm sayısı metrikler halinde tutulur.

Bir diğer yaklaşım *dağıtılmış yönlendirme* (*distributed routing*) yaklaşımıdır. Bu yaklaşımda yönlendirme kararları, iletim yolu boyunca gezilen ara düğümlerde dağıtılmış bir yaklaşımla verilir. Her ara düğüm, ileti kendisine iletildikten sonra,

anahtarlama modeline bağlı biçimde, bir sonraki iletilecek ara düğüme karar verir. Bu sürecin her ara düğümde tekrarlanması sonucunda paket hedef düğüme ulaşır. Dağıtılmış yönlendirmede, ileti başlığı içinde hedef düğüm adresi ve birkaç denetim bilgisinin bulunması yeterli olacaktır. Dolayısıyla dağıtılmış yönlendirmede ileti/paket başlık boyları, kaynak kararlı yönlendirmeye göre oldukça düşüktür. Dağıtılmış yönlendirmede yönlendirme işlemleri yerel ağ durum bilgisiyle yapılır. Bu durumda yazılımcının ağ topolojisini bilmesi gerekebilir. Dağıtılmış yönlendirme düzenli topolojilerde, her düğümün eşit sayıda komşusu olan topolojilerde, tercih edilir. Böylece aynı yönlendirme algoritması bütün düğümlerde çalıştırılabilir.

Bu yöntemlerin birden fazlasının bir arada kullanıldığı melez yöntemler de bulunur. Bu yöntemler *çok fazlı yönlendirme (multiphase routing)* olarak tanımlanır. Çok fazlı yönlendirmede, yönlendirme işlemleri birden fazla adımda gerçekleşir. Örneğin kaynak düğüm bir ara düğüm hesaplar ve ileti bu düğüme kadar, dağıtılmış veya kaynak kararlı olarak iletilir. Ara düğüme erişince iletinin bir sonraki hedefi hesaplanır ve aynı şekilde tekrar iletilir. Bilinen birçok fazlı yönlendirme algoritması, *rastgele yönlendirme (random routing)* yöntemidir. Rastgele yönlendirme yönteminde, kaynak rastgele bir hedef düğüm seçer ve ileti bu rastgele düğüme eriştikten sonra asıl hedef düğüme iletilir. Rastgele yönlendirmede, iletim yolları rastgele belirlenerek çakışmaların azaltılması hedeflenir. Bununla birlikte bu yöntem iletişim yerelliğini (Ni, 1993) yok eder.

Yönlendirme algoritmaları birçok yöntemle gerçekleştirilebilir. En çok önerilen gerçekleştirme yöntemleri, *tablo yöntemi (table lookup)* ve *sonlu durum makineleridir (finite state machine)* (Duato, 2002).

Tablo yönteminde, en basit yaklaşımla, her yönlendiricide bir yönlendirme tablosu bulunur ve bu tabloda ağ içindeki her düğüm için bir satır bulunur. Bu yöntem dağıtılmış yönlendirme ve kaynak kararlı yönlendirme yaklaşımlarının her ikisi ile de gerçekleştirilebilir. İlk durumda, tablo içinde seçilen hedef düğüme eşleşmiş tablo satırında, bu düğüme erişmek için seçilecek giden kanal bulunur. İkinci durumda ise hedef düğüm satırı, hedef düğüme erişmek için takip edilecek bütün iletim yolunu içerir. Bununla birlikte, bu gerçekleştirim sadece küçük sistemler için elverişlidir. Çünkü düğüm sayısının artması, yönlendirici içinde yönlendirme tablosunun

saklanmasını zorlaştırır. Dağıtılmış yönlendirme için tablo boyunu düşürmeye yönelik bir yaklaşım, tablo içinde her kanal ile eşleşmiş düğüm aralığını tutmaktır.

Sonlu durum makineli (SDM) yönlendirme yaklaşımında, yönlendirme algoritmaları basit sonlu durum makineleri şeklinde tasarlanır ve her yönlendirme koşulu bir sonlu durum olarak ele alınır. Yönlendirme algoritması, iletinin halihazırda bulunduğu duruma göre bir sonraki yönlendirme durumunu seçer. SDM yönlendirme *torus*, *hypercube* gibi düzenli topolojilerde tercih edilir.

Yönlendirme algoritmaları, yönlendirme algoritmasının karar verme davranışına göre iki sınıfa ayrılır: durağan *yönlendirme (deterministic routing)* ve *uyarlamalı yönlendirme (adaptive routing).*

Durağan yönlendirmede herhangi bir kaynak/hedef çifti için her zaman aynı iletim yolu kurulur. Kaynak kararlı yönlendirme kullanıldığında, iletim yolu hesaplaması kaynak düğüm tarafından yapılır. Dağıtılmış yönlendirme yönteminde, her ara düğüm benzer hedef düğümler için aynı iletim yolunu seçer. Durağan yönlendirmede yönlendirme kararı sadece kararın verildiği düğüm ve hedef düğüm bilgileri kullanılarak verilir. Yönlendirme kararı verilirken, ağ koşulları veya seçilen kanalın anlık durumu dikkate alınmaz. Durağan yönlendirme algoritmasının basit olarak sonlu durum makinesi şeklinde kurulması, bu bağlamda çalışan basit ve hızlı yönlendirici donanımlarının tasarlanmasını sağlar. Bu yüzden ticari koşut sistemlerin birçoğu durağan ve dağıtılmış yönlendirme kullanırlar. Durağan yönlendirme, kararlı ağ koşulları olan sistemlerde yüksek başarım sağlar. Bununla birlikte, ağ trafiğinin değişken olduğu ve komşu olmayan düğümler arası veri trafiğinin yüksek olduğu uygulamalarda durağan yönlendirme başarımı düşüktür.

Alternatif yaklaşımda, yönlendirme kararı alınırken ağ koşulları ve seçilecek giden kanalının kullanım durumunun dikkate alındığı yönlendirme yaklaşımına *uyarlamalı yönlendirme (adaptive routing)* denir. Uyarlamalı yönlendirme algoritması iki işlevin birleşiminden oluşur: *yönlendirme* ve *seçme*. Yönlendirme işlevi, kararı veren düğüm adresi ve hedef düğüm adresi bilgilerini kullanarak bir giden kanal kümesi seçer. Seçme işlevi ise yönlendirme işlevi tarafından sağlanan giden kanal kümesinden, kanalların anlık kullanım durumlarına bağıl olarak bir giden kanalı belirler. Sonuç

olarak uyarlamalı yönlendirmede, ileti seçilen kanalın uygun olmasını beklemek yerine diğer iletim yollarını kullanır.

Uyarlamalı yönlendirme kendi içinde iki sınıfa ayrılır: *ilerleyen (progressive)* ve *geri izlemeli (backtracking) uyarlamalı yönlendirme.* İlerleyen uyarlamalı yönlendirme algoritmaları her yönlendirme işlemi başlangıcında kanalı ayırır ve ileti başlığını kanala sürer. Geri izlemeli yönlendirmede ise yönlendirme algoritması, ilerlemeden önce geri izleyerek bir önceki ayırdığı kanalları serbest bırakır. Çoğu anahtarlama modeli, ileti başlığından hemen sonra ileti verisini hatta verirler dolayısıyla, bu anahtarlama modellerinde geri izlemeli yönlendirme gerçekleştirilemez.

Daha düşük seviyede, yönlendirme algoritmaları, karar verilen iletim yolu uzunluğuna göre *doğrudan (minimal) ve dolaylı (nonminimal) yönlendirme* olarak iki sınıfta toplanır. Doğrudan yönlendirmede her zaman kaynak ve hedef düğümler arasındaki en kısa yol seçilir. Dolaylı yönlendirmede ise seçilen iletim yolunun, iki düğüm arasındaki en kısa yol olduğu garanti edilmez. Dolaylı yönlendirmede, iyimser bir yaklaşımla, kullanılmayan kanalların kullanılarak iletimin sürmesi ve hedef düğüme daha çok yaklaşılması hedeflenir. Dolaylı yönlendirme esnek olmasıyla beraber ağ kaynaklarını daha çok tüketir.

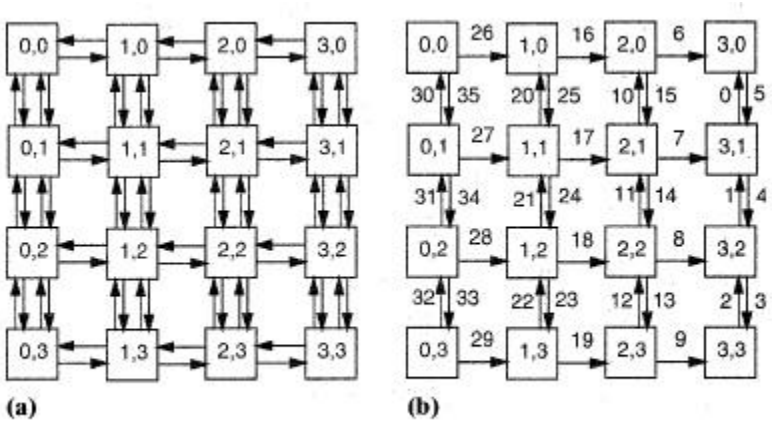
5.1. Genel k-ary n-küp topolojiler için yönlendirme

k-ary n-küp topolojilerde k>4 durumu için, kilitlenmelerden bağımsız bir yönlendirme algoritması tanımlamak imkansızdır. n=1 durumu için bile, anlaşılacağı üzere, topolojileri ring topolojisine benzeyecektir ve döngüsel çevrim oluşacaktır. Kilitlenmeleri engellemek için, kanal bağımlılık çizgesinden yararlanılarak, döngü oluşturan köşelerden birinin silinmesi gerekir. Bu durumda da, bazı yönlendirmeler için en kısa yolun seçileceği garanti edilemez. Dolayısıyla, k-ary n-küp topolojileri için, kilitlenmelerden bağımsız dolaylı durağan yönlendirme algoritmaları geliştirilebilir.

5.2. Doğrudan Uyarlanır Yönlendirme (*Minimal Adaptive Routing*)

Genel bir uyarlanır doğrudan yönlendirme yaklaşımı, ağ içindeki kanalları ayrık alt kümelere bölmektir. Her alt küme farklı bir ağ içerir ve paket hedef düğümüne göre farklı alt ağlar içinde ilerler. Şekil E-17'de iki boyutlu hasır doku (a) ve bu hasır dokudan bölümlenmiş +X alt ağı (b) gösterilmiştir.

Şekil E-17'de görüldüğü üzere, y ekseninde düğümler arasında ikişer tane çift yönlü (*duplex*) sanal kanal bulunmaktadır. Ağ, artan x ekseni yönündeki kanalları içeren +X ve azalan x ekseni yönündeki kanalları içeren -X alt ağlarına bölünebilir. Y eksenindeki hat çiftlerinden biri +X diğeri -X alt ağına tahsis edilecektir. Eğer hedef düğüm(d_x), kaynak düğümün($s_{x)}$ sağ tarafında ($d_x > s_x$) ise paket +X alt ağı, ağın sol tarafında ($d_x < s_x$) ise paket -X alt ağı üzerinden iletilir. Eğer hedef düğüm ile kaynak düğüm koordinatlarının x değerleri aynıysa, paket her iki ağ üzerinden de iletilebilir.



Şekil E-17. Minimal uyarlanır yönlendirme (Duato, 2002)

Bahsedilen çift Y-kanal yönlendirme algoritması her zaman doğrudan ve uyarlanırdır. İleti, alt ağ üzerindeki en kısa yollardan herhangi biri üzerinden iletilebilir. Bununla birlikte, kanallar (Şekil E-17-b) şeklinde olduğu gibi numaralandırılır ve her zaman azalan sıradaki kanallar üzerinden iletilirse her zaman kilitlenmelerden bağımsızdır.

Torus, *hypercube* gibi k-ary n-küp topolojileri için tam uyarlanır minimal yönlendirme algoritmaları tanımlamak için ek kanalların ağa eklenmesi gerekir. Linder ve Harden, bir k-ary n-küp topolojisinin 2^{n-1}tane alt ağa bölünebileceğini, her ağda *n+1* düzey ve k^n tane kanal olacağını göstermiştir (Linder, 1991). Bununla birlikte, bu yaklaşımın uygulanması büyük n değerleri için çok düşüktür.

5.3. Dolaylı Uyarlanır Yönlendirme (*Nonminimal Adaptive Routing*)

Eğer yönlendirme algoritması, seçilen yolun her zaman minimal olmasını istemiyorsa, daha az ek kanal kullanılarak dolaylı uyarlanır yönlendirme algoritmaları tanımlanabilir. Eğer her düğümü arasında r çift kanal olduğu düşünülürse; Dally ve Aoki tarafından tanımlanan algoritma k-ary n-küp ve mesh topolojilerine uyarlanabilir (Dally, 1993). Bu algoritmalarda, ağ durumuna bağıl en kısa yol için gerekli düğümler kullanılır olduğunda, yönlendirme için daha uzun yollar seçilebilir.

5.4. Durağan Ters Boyut Sıralı Yönlendirme

Bu yönlendirme algoritmasında herhangi iki düğüm arasında *r* çift kanal bulunur. Bu yolla ağ *r* adet alt ağa bölünebilir. *i.* alt ağı her düğüm arasındaki *i.* kanal çiftlerini içerir ve sınıf-i alt ağı olarak tanımlanır (Dally, 1993). Paket başlığı içinde c isimli bir sınıf alanı bulunur. c alanı (r-1) değerinden daha küçük olan paketler, sınıf-c alt ağı içinde, her yöne yönlendirilebilirler. Bununla birlikte, paket düşük boyuttan yüksek boyuta her yönlendirildiğinde ki bu yönlendirme boyut sıralı yönlendirmeyle terstir, paket başlığındaki c alanı bir artırılır. c değeri (r-1) değerine ulaştıktan sonra, paket geri kalan yol boyunca durağan boyut sıralı yönlendirme algoritmasına göre yönlendirilmelidir. Bu algoritmada, sistemdeki ek kanallar paketlerin ters boyut sıralı, y boyutundan x boyutuna gibi, yönlendirilebilmesini sağlar. Bir paketin kaç kere yüksek boyuttan düşük boyuta yönlendirilebileceği, r sayısıyla sınırlıdır.

5.5. Devingen Ters Boyut Sıralı Yönlendirme

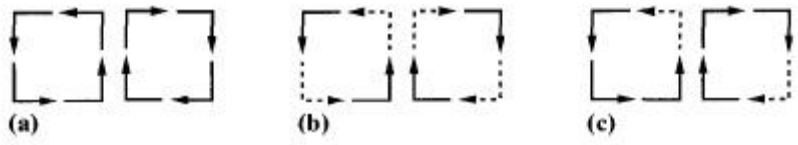
Bu algoritmada, durağan ters boyutlu yönlendirmeye benzer şekilde, paket başlıkları içinde c alanı bulunur ve paketin her ters boyutlu yönlendirilmesinde c alanı bir artırılır. Bunun yanında devingen ters boyut sıralı yönlendirmede paketlerin ters sıralı yönlendirilmesinde uygulanacak yaklaşım farklıdır.

Devingen ters boyut sıralı yönlendirmede, ağ içindeki kanallar durağan (*deterministic*) ve uyarlanır (*adaptive*) olmak üzere iki sınıfa ayrılır (Dally, 1993). Bütün paketler yönlendirilmeye uyarlanır alt ağda başlarlar. Ağ içinde paketlerin kaç kere ters sıralı yönlendirileceğine dair bir sınır bulunmadan, paketler herhangi bir yönde boyut sırasından bağımsız yönlendirilirler. Fakat c değeri c_1 olan bir paket; c_1 değerinden daha küçük bir c=c_2 değerine sahip bir pakete ayrılmış bir kanal üzerinde

bekleyemez. Örneğin c_1=3 olan bir paket, c_2=2 paketi tarafından kullanımda olan kanal üzerinde bekleyemez. Büyük c değerli paket başka bir kanalı kullanır veya o kanal üzerinde bekler. Tüm ihtiyaç duyduğu kanallar, daha küçük c değerlerine sahip paketlere tahsis edilmiş olan bir paket, uyarlanır ağ üzerinde başka bir kanal üzerinde bekleyemez ve durağan ağa geçer. Durağan ağa geçen paket, durağan boyut sıralı yönlendirme kurallarına göre yönlendirilir ve tekrar uyarlanır ağa geçemez.

Bu algoritmada, c alanının kullanımına bakılarak, algoritmanın kilitlenmelerden bağımsız olduğu açıktır. Bu ağların tasarımında, hangi kanalların uyarlanır hangi kanalların durağan ağda bulunacağı bir tasarım sorunudur.

5.6. Dönüş Modeli (*Turn Model*)



Şekil E-18. Dönüş modeli

Uyarlanır yönlendirme algoritmaları genelde, verilmiş bir ağ topolojisi ve bu ağa atanmış kanal kümesine özel getirilmiş yaklaşımlardır. Fakat Glass ve Ni tarafından bulunan dönüş modeli uyarlanır yönlendirme için sistematik bir yaklaşım getirmiştir (Glass, 1992). Bu algoritma ek kanal eklemeden gerçekleştirilebilir. Bilindiği üzere, karşılıklı kilitlenmelerin paketin yönlendirme boyunca takip edeceği yolda döngüler olduğu zaman meydana gelir. Dönüş modeli döngüleri engelleme yaklaşımına dayanır ve bu yaklaşım doğrudan ve dolaylı yönlendirmelerin her ikisi için de gerçekleştirilebilir. n-boyutlu hasır ve k-ary n-küp topolojileri için uyarlanır bir algoritma gerçekleştirmek için aşağıdaki altı adım kullanılır:

1. Kanalları yönlendirme yaptıkları yöne bağıl olarak sınıflandır.
2. 0 ve 180 derece dönüşleri göz ardı edilerek, yönler (boyutlar) arasında yapılabilecek dönüşler belirlenir.
3. Bu dönüşlerle ortaya çıkabilecek döngüler belirlenir.
4. Her döngü için bir dönüş engellenir.
5. k-ary n-küp topolojileri için sarmal kanallar, topolojide boyut içinde dönüş sağlayan kanallar, içeren döngülerden, mümkün olanların tümü birleştirilir.

6. Dolaylı yönlendirmeler için gerekli olan 0 ve 180 derece dönüşler ve varsa aynı yöne ait çoklu kanallar da sisteme eklenir.

İki boyutlu hasır doku üzerinde dönüş modelinin nasıl çalıştığı Şekil E-18'de gösterilmiştir. İki boyutlu bir hasır dokuda sekiz çeşit dönüş bulunur ve iki tane olası döngü bulunur (Şekil E-18-a). Boyut sıralı yönlendirme algoritması bu dönüşlerin dört tanesine izin vermez. Geri kalan dönüşlerle döngüsel bir yönlendirme yolu üretilemez ve karşılıklı kilitlenmeler engellenmiş olur. Bununla birlikte, bu dönüşlerle uyarlanır bir yönlendirme algoritması da tanımlanamaz (Şekil E-18-b).

Dönüş modelinden küçük sayıda dönüşü çıkararak döngüleri engelleme mantığına dayanır. İki boyutlu hasır doku için, sadece iki dönüşün çıkarılması döngülerin engellenmesi için yeterlidir (Şekil E-18-c).

Dönüş modeli yaklaşımı, birçok uyarlanır yönlendirme algoritmalarının üretilmesine olanak verir. Dönüş modeli yaklaşımından faydalanılarak çıkarılan algoritmaların en bilinenleri Batı-Öncelikli (*West-First*) , Negatif-Öncelikli (*Negative-First*), Kuzey-Sonda (*North-Last*) ve *p-cube* algoritmalarıdır.

5.7. Batı-Öncelikli Yönlendirme (*West-First Routing*)

Bu algoritma dönüş modelini temel alır. Batı öncelikli yönlendirmede, eğer paket için gerekliyse, paket ilk adımda batıya yönlendirilir. Daha sonraki yönlendirme adımlarında, paket uyarlanır yaklaşımla güneye, doğuya ve kuzeye yönlendirilir (Ni, 1993).

Batı-öncelikli yönlendirme yaklaşımında batıya dönüşlere izin verilmediği için döngüler engellenir. Dolayısıyla bu algoritma karşılıklı kilitlenmelerden bağımsızdır. Bu algoritma doğrudan yönlendirme için; hedef sağ el tarafında(doğu) ise uyarlanır yönlendirme, hedef sol el tarafında (batı) ise durağan yönlendirme uygular. Dolaylı yönlendirme için algoritma hedef konumundan bağımsız olarak uyarlanır yönlendirme yapar.

5.8. *p-küp* Yönlendirme

p-küp algoritması dönüş modeli yaklaşımıyla k-ary n-küp topolojisinde ağlar için üretilmiş bir algoritmadır. Bu algoritmanın önemli özelliği sarmal kanalların kullanımına olanak verir. Algoritma doğrudan veya dolaylı yönlendirme yapacak biçimde gerçekleştirilebilir. Algoritma negatif-öncelikli (*negative-first*) yaklaşımını kullanır (Duato, 2002).

p-küp algoritmasında, düğümleri adreslerken, her ikilisi bir boyutu göstermek üzere, düğümler n ikiliden oluşan adreslere sahip olurlar. Algoritmada gönderilecek iletinin gönderilmesi iki fazda gerçekleşir. İlk fazda; kaynak düğümden ilk ara düğüm olmak üzere, hedef düğüm adresiyle ara düğüm adresleri karşılaştırılır. İleti; ara düğüm adres ikili değeri, hedef düğüm adres ikili değerinden büyük olduğu boyut(lar)dan herhangi birinde yönlendirilir. Başka bir deyişle, kaynak / ara düğüm adres değeri S, hedef adres değeri D olmak üzere; S_x = 1 D_x=0 ise ileti x boyutunda iletilebilir ve x boyutuna denk gelen kanal üzerinden yönlendirilir. Eğer ara düğüm adresi, hedef adresinin aynı boyuta gelen ikililerinin hiçbirinde, hedef adres değerinden büyük değilse yönlendirme ikinci faza geçer. İkinci fazda ise, birinci faza ters yaklaşım uygulanır. İleti; hedef adres ikili değerinin, ara düğüm adres ikili değerinden büyük olduğu boyutlarda yönlendirilir. Bu fazın sonunda, ara düğüm ve hedef düğüm adresleri aynı olduğunda ileti ağ ortamını terk eder ve yerel işlemciye yönlendirilir.

Algoritmanın ilk fazında, ara düğüm adres değerlerinin daha büyük olduğu boyutlar seçilerek, iletinin negatif yönde yönlendirilmesi sağlanır. İletinin negatif yönde gideceği düğüm kalmayınca düğüm negatif boyutlardan pozitif boyutlara geçer ve pozitif yönlerde yönlendirilir.

5.9. Hata Hoşgörülü Ağlar (*Fault-Tolerance Networks*)

Ağ içindeki yönlendirici veya fiziksel kanal gibi birimlerde oluşacak hatalar, ağın belli bölgelerinde iletimlerin yanlış olmasına veya iletimin durmasına sebep olabilir. Yönlendirme veya anahtarlama yaklaşımlarını ağdaki aykırı durumları da göz ününe alarak, tanımlayan ağ yaklaşımları *hata toleranslı ağlar* olarak tanımlanırlar (Duato, 2002). Bu tür ağlar için anahtarlama ve yönlendirme yaklaşımları bulunmaktadır (Dao, 1995, Duato, 1997, Linder, 1991).

6. Benzetim için kayıt kütüğü örnekleri

6.1. Benzetim Altyapısı Olay işleme kayıt kütüğü Örneği

OLAY KİMLİĞİ	OLAY- TÜRÜ	KAYNAK	ZAMAN
0	N	3	1
1	N	0	1
2	N	24	1
3	N	9	1
4	N	3	2
5	N	0	2
6	M	24	2
7	N	9	2
8	N	3	3
9	N	0	3

Benzetim alt yapısı, her bir mantıksal görev için yukarıda gösterilen biçimde kayıt kütükleri oluşturur. Kayıt kütüklerinde, mantıksal görevlerin olayları işlenme sırasıyla işlenen olaylar tutulur. Her bir satır ayrı bir olay işlemesine karşılık gelir. Kütük içindeki sütunlar sırasıyla, olayın işlenme sırası, olayın türü (N=Boş, M= ileti), olayı gönderen mantıksal görev kimliği ve olay zamanı bilgilerini içerir. Kayıt kütükleri virgülle ayrılmış değer (*Comma separated value, csv*) biçiminde tutulurlar.

6.2. Ağ Altyapısı ileti işleme kayıt kütüğü örneği

GÖREV	BİLEŞEN	İLETİ	OLAY	KAYNAK/HEDEF
35	Router	Header Flit0:C:4	GÖNDER	0
35	Router	Flit1:C:3	GÖNDER	0
34	Router	Header Flit0	AL	35
35	Router	Flit2:C:2	GÖNDER	0
34	Router	Header Flit0:C:4	GÖNDER	35
34	Router	Flit1	AL	35
34	Router	Flit1:C:3	GÖNDER	35
35	Router	ACK Flit0	AL	34

Ağ altyapısı, benzetim altyapısına benzer biçimde, benzetim uygulaması olarak iletilerin kayıtlarını tutar. Ağ altyapısında, her bileşen kayıt oluşturabilir. Kayıt kütüğündeki her bir satır, işlenen bir olayı içerir. Sütunlar sırasıyla, satırı yazan görev kimliği, iletiyi işleyen benzetim uygulaması bileşeni, ileti içeriği, uygulamada iletinin işlenme biçimi ve iletiyi yollayan / alan görev kimliği bilgilerini içerir. Ağ altyapısı katmanında bütün görevler için tek bir kayıt kütüğü oluşturulmaktadır.

7. Peters-Syska Yayım Algoritması İşlemci Gerçekleştirimi

Bu bölümde Peters-Syska algoritmasının gerçekleştirimindeki İşlemci benzetim sınıfı verilmiştir. Okunurluğu artırmak için kütüphane tanımları verilmemiştir. Okunurluğu artırmak için sadece arayüz davranışında ve algoritmada tanımlanan metotlar verilmiştir.

Algoritma Bölüm 5.2.6'da bahsedilen İşlemci soyut sınıfını genişletilerek gerçekleştirilmiştir. Gerçekleştirimde; *initSystem()* metodunda,sistem başlangıcında yayım yapan düğüm baskın düğümlerin konumlarını hesaplar ve iletiyi baskın düğümlere gönderir. Her baskın düğüm, ileti ulaşınca, iletiyi komşu düğümlerine iletir. Bu bölüm *processMessage()* metodu içinde gerçekleştirilir. Yayım sahibi düğüm, *sendInternalAck()* metodu içinde, gelen alındı imi sayısı 4'e ulaşınca tüm baskın düğümlere iletimin sona erdiğine karar verir ve kendi komşu düğümlerine yayım iletisini göndermeye başlar.

```
public class PetersBroadcast extends Processor{

	private boolean bCastOwner;

	private int phase = 1;

	private intdominatingNodeACKs = 0;

	private int bCastSize;

	private Object bCastData;


	public PetersBroadcast(Router router) {

		super(router);

		bCastOwner = false;

	}

	protectedvoid multicastDNodes(int size, Object data) {

		List<Header> headers = new ArrayList<Header>();

		//Calculate X-Y- zone dominating node route

		List<SubRoute> routes = new ArrayList<SubRoute>();

		SubRoute x = new SubRoute(1, Direction.X_negative);

		routes.add(x);
```

```
        SubRoute y = new SubRoute(2, Direction.Y_negative);

        routes.add(y);

        //Set Header

        MetricRoutedMHeader header = new MetricRoutedMHeader(id, size,
              MessagingType.MULTICAST, routes);

        header.setNodeZone(1);  headers.add(header);

        //Calculate X+ Y- zone dominating node route

        routes = new ArrayList<SubRoute>();

        y = new SubRoute(1, Direction.Y_negative);

        routes.add(y);

        x = new SubRoute(2, Direction.X_positive);

        routes.add(x);

        //Set Header

       header = new MetricRoutedMHeader(id, size,
MessagingType.MULTICAST, routes);

        header.setNodeZone(2);

        headers.add(header);

        //Calculate X+ Y+ zone dominating node route

        routes = new ArrayList<SubRoute>();

        x = new SubRoute(1, Direction.X_positive);

        routes.add(x);

        y = new SubRoute(2, Direction.Y_positive);

        routes.add(y);

        //Set Header

        header = new MetricRoutedMHeader(id, size,

                    MessagingType.MULTICAST, routes);

        header.setNodeZone(3);

        headers.add(header);
```

```
        //Calculate X- Y+ zone dominating node route

        routes = new ArrayList<SubRoute>();

        y = new SubRoute(1, Direction.Y_positive);

        routes.add(y);

        x = new SubRoute(2, Direction.X_negative);

        routes.add(x);

        //Set Header

        header = new MetricRoutedMHeader(id, size,
MessagingType.MULTICAST, routes);

        header.setNodeZone(4);

        headers.add(header);

        //Set Multicast message and send

        MulticastMessage m = new MulticastMessage(size,

              MessagingType.MULTICAST, id, headers);

        m.setData(data);

        router.send(m);

}

protected void multicastNeighbours(int size, Object data) {

        List<Header> headers = new ArrayList<Header>();


        //Calculate X- neighbour message

        SubRoute s = new SubRoute(1, Direction.X_negative);

        headers.add(new MetricRoutedHeader(id, size,

                    MessagingType.BROADCAST, s));

        //Calculate X+ neighbour message

        s = new SubRoute(1, Direction.X_positive);

        headers.add(new MetricRoutedHeader(id, size,

                    MessagingType.BROADCAST, s));

        //Calculate Y- neighbour message

        s = new SubRoute(1, Direction.Y_negative);

        headers.add(new MetricRoutedHeader(id, size,

                    MessagingType.BROADCAST, s));
```

```
        //Calculate Y+ neighbour message
        s = new SubRoute(1, Direction.Y_positive);
        headers.add(new MetricRoutedHeader(id, size,
                    MessagingType.BROADCAST, s));


        //Create multicast message and send
        MulticastMessage m = new MulticastMessage(size,
              MessagingType.MULTICAST,id, headers);
        m.setData(data);
        router.send(m);
    }
    @Override
    public void initSystem() {
        if(id == 12) {
              bCastOwner = true;
              String s = "Message-"+id;
              bCastData = s;
              bCastSize = 40;
        multicastDNodes(bCastSize, s);
        }
    }
    @Override
    public void processMessage(Packet p) {
        if(p.getMessagingType() == MessagingType.MULTICAST){
              Object data = p.getData();
              int size = p.getSize();
              multicastNeighbours(size, data);
        }
    }
```

```
@Override
public void sendInternalACK(int dest) {
	if(bCastOwner && phase == 1) {
		dominatingNodeACKs++;
		if(dominatingNodeACKs == 4) {
			multicastNeighbours(bCastSize, bCastData);
			phase = 2;
		}
	}
} }
```

8. İki Boyutlu Torus Topolojisi Gerçekleştirimi

Bu bölümde iki boyutlu torus topolojisi için topoloji kurucu nesne gerçekleştirimi verilmiştir. Gerçekleştirim, Topoloji Kurucu arayüzün *getNeighbour()* ve setNeighbours*()* metotlarını gerçekleştirir. *getNeighbour()* metodu, torus topolojisine göre, ağ kimliği verilen düğümün, parametre yöndeki komşusunun ağ kimliğini hesaplar. *setNeighbours()* metodu parametre olarak gönderilen mantıksal görev için, komşu mantıksal görevleri belirler.

```
public class TorusBuilder2D implements TopologyBuilder {

	private Coordinate dimension;

	public TorusBuilder2D(Coordinate dimension) {

		this.dimension = dimension;

	}

	@Override

	public int getNeighbour(int id, Direction d) {

		Coordinate coordinate = Coordinate.getCoordinates(id, dimension);

		int dimX = dimension.getX();

		int dimY = dimension.getY();

		int neighbourID = 0;

		switch (d) {

		case X_negative:

			neighbourID = (coordinate.getX() == 0 ? (id+(dimX-1)) : id-1);

			break;

		case X_positive:

			neighbourID = (coordinate.getX() == (dimX-1) ? (id-(dimX-1)) :
id+1);

			break;

		case Y_negative:

			neighbourID = (coordinate.getY() == 0 ? (id+(dimY-1)*dimX) :
id-dimX);

			break;

		case Y_positive:

			neighbourID = (coordinate.getY() == (dimY-1) ?
```

```
                        (id-(dimY-1)*dimX) : id+dimX);
                break;
        }
        return neighbourID;
    }
    @Override
    public void setNeighbours(LogicalProcess process) {
        SimNetwork simulation = process.getNetwork();
        int id = process.getId();

        //    Sets X negative neighbour
        int nID = getNeighbour(id, Direction.X_negative);
        LogicalProcess neighbour = simulation.getNode(nID);
        process.setNeighbour(neighbour, Direction.X_negative);

        //    Sets X positive neighbour
        nID = getNeighbour(id, Direction.X_positive);
        neighbour = simulation.getNode(nID);
        process.setNeighbour(neighbour, Direction.X_positive);

        //    Sets Y negative neighbour
        nID = getNeighbour(id, Direction.Y_negative);
        neighbour = simulation.getNode(nID);
        process.setNeighbour(neighbour, Direction.Y_negative);

        //    Sets Y positve neighbour
        nID = getNeighbour(id, Direction.Y_positive);
        neighbour = simulation.getNode(nID);
        process.setNeighbour(neighbour, Direction.Y_positive);
    }
```

ÖZGEÇMİŞ

Adı Soyadı : Hüseyin Temuçin

Doğum Yeri : Malatya

Doğum Yılı : 1985

Medeni Hali : Bekar

Eğitim ve Akademik Durumu:

Lise	:	2000-2004	Malatya Y.D.A. Lisesi
Lisans	:	2004-2006	Kocaeli Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü
		2006-2008	Hacettepe Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü
Yabancı Dil	:	İngilizce	

İş Tecrübesi:

2008-...	Araştırma Görevlisi, Hacettepe Üniversitesi Bilgisayar Mühendisliği Bölümü