KOCAELİ ÜNİVERSİTESİ * FEN BİLİMLERİ ENSTİTÜSÜ

KABLOSUZ ALGILAYICI AĞLARININ OMNeT++ İLE BENZETİMİ

YÜKSEK LİSANS

Elektronik ve Haberleşme Müh. Muharrem SIRMA

Anabilim Dalı: Elektronik ve Haberleşme Mühendisliği

Danışman: Y.Doç.Dr. Mehmet YAKUT

KOCAELİ, 2006

KOCAELİ ÜNİVERSİTESİ * FEN BİLİMLERİ ENSTİTÜSÜ

KABLOSUZ ALGILAYICI AĞLARININ OMNeT++ İLE BENZETİMİ

YÜKSEK LİSANS TEZİ

Elektronik ve Haberleşme Müh. Muharrem SIRMA

Tezin Enstitüye Verildiği Tarih: 26 Mayıs 2006

Tezin Savunulduğu Tarih: 02 Ağustos 2006

Tez Danışmanı	Üye	Üye
Y.Doç.Dr.Mehmet YAKUT	Prof.Dr.Hasan DİNÇER	Y.Doç.Dr.İbrahim ÖZÇELİK
(.....)	(.....)	(.....)

KOCAELİ, 2006

ÖNSÖZ ve TEŞEKKÜR

Bu çalışma ile kablosuz algılayıcılar üzerine dikkat çekilmek istenmiş ve bu teknolojinin en önemli unsuru olan haberleşme üzerine incelemeler yapılmıştır. Kablosuz algılayıcılar özellikleri gereği Tıp'tan Savunma Sanayii'ne kadar çok geniş bir alana hitap edebilmektedir. Bu konularda yapılacak yeni projelerin ülkemize büyük katma değer sağlayacağı açıktır. Halen kablosuz algılayıcıların ve kablosuz haberleşmenin gelişime ihtiyaç duyduğu hususlar vardır: haberleşme mesafesi, verimli ve güvenli haberleşme, enerji ve algılama bunların en önemlileridir. Kablosuz haberleşmenin geliştirilmesi için yeni bir iletişim kuralı geliştirmek hedeflenmektedir. OMNeT++ benzetim programı kullanılarak yapılan inceleme ile Flooding iletişim kuralı ile IEEE tarafından geliştirilen 802.11x standardının dar boğazları olduğu görülmüştür, yapılacak yeni çalışmalar ile bunlara çözüm getirilmeye çalışılacaktır.

Yardımlarından, katkı ve desteklerinden dolayı Sayın Y.Doç.Dr.Mehmet YAKUT'a, Sayın Y.Doç.Dr.Ali TANGEL ve Sayın Doç.Dr. Adnan KAVAK'a çok teşekkür ederim.

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

TABLOLAR DİZİNİ

SEMBOLLER

C	: Bataryanın Amper-Saat olarak kalan kapasitesi
Cin	: Bataryanın başlangıç kapasitesi
d	: Gecikme süresi
Gr	: Alıcı anten kazancı
Gt	: Verici anten kazancı
hr	: Alıcı anten yüksekliği
ht	: Verici anten yüksekliği
I	: Algılayıcı düğüm tarafından Amper olarak çekilen toplam akım
I(t)	: Δt süresinde algılayıcı düğüm tarafından çekilen akım
L	: Sistem kaybı ve dalga boyu
Pt	: İletilmiş işaretin gücü
Pr	: Alınmış işaretin gücü
T	: Bataryadan dayanması beklenilen (saat cinsinden) süre

Kısaltmalar

ACK	: Acknowledgement
AID	: Association Identity
AP	: Access Point
API	: Application Program Interface
ARP	: Address Resulation Protocol
BSS	: Basic Service Set
BSSID	: BSS Identifier
CPU	: Central Processor Unit
CRC	: Cyclic Redundancy Check
CSMA/CA	: Carrier Sense Multiple Access/ Collision Avoidance
CSMA_CD	: Carrier SenseMultiple Access with Collision Dedect
CSMA/CD	: Carrier Sense Multiple Access/ Collision Dedection
CTS	: Clear To Send
DA	: Destination Address
DIFS	: Distributed Inter Frame Space
DS	: Distribution System
DSS	: Distribution System Servise
DSSS	: Direct Sequence Spread Spectrum
EAPOL	: Extensible Authentication Protocol (EAN) over LAN
EIFS	: Extended IFS
ESS	: Extended Service Set
FHSS	: Frequency-Hopping Spread Spectrum

FSC	: Frame Check Sequence
GHz	: Giga Hertz
GUI	: Graphical User Interface
Hz	: Hertz
IBSS	: Independent Basic Service Set
ID	: Identification
IEEE	: Institue Of Electrical and Electronics Engineers
IFS	: Inter Frame Space
ISO	: International Standart Organization
ISM	: Industrial, Scientific and Medical
IP	: Internet Protocol
IR	: Infrared
LAN	: Local Area Network
LLC	: Logical Link Control
MAC	: Media Access Control
MBps	: Mega Bits Per Second
MF	: Mobility Framework
MHz	: Mega Hertz
MPI	: Message Passing Interface
MPDU	: MAC Protocol Data Unit
MSDU	: MAC Service Data Unit
NAV	: Network Allocation Vector
NIC	: Network Interface Card
OFDM	: Orthogonal Frequency Division Multiplexing
OMNeT++	: Objective Modular Network Test-bed in C++
OSI	: Open Systems Interconnection
PHY	: Physical
PIFS	: Point Coordination IFS
PPP	: Point-to-point Protocol
PS	: Power Save
RA	: Receiver Address
RTS	: Request to Send
SA	: Source Address
SIFS	: Short Inter Frame Space
SNR	: Signal-to-Noise Ratio
SSID	: Service Set Identifier
STA	: Station
TA	: Transmitter Address
WAP	: Wi-Fi Protected Access
WEP	: Wired Equivalent Privacy
WLAN	: Wireless Local Area Network

1. GİRİŞ

Bu çalışma ile kablosuz algılayıcılar ve kablosuz algılayıcı ağları incelenmiş ve bu alanda projeler üretmek ve geliştirmek amaçlanmıştır (Kablosuz Algılayıcıların Tıbbi Bakımlarda Kullanılması ve Kablosuz Yangın Alarm ve Kontrol Paneli isimli projelerin tasarım ve planlama aşamaları tamamlanmıştır). Bu çalışmada ayrıca yeni nesil teknoloji olan kablosuz algılayıcıların ve ağları'nın öneminden bahsedilmiş olup, yapısal özellikleri, kullanım alanları, kullanım amaçları üzerine yapılan araştırma sonuçları anlatılmış, çalışma ve haberleşme şekillleri ile birbirleri ile olan ilişkileri incelenmiş, dar-boğaz olan ve olabilecek noktalara işaret edilmiştir. Ayrıca kablosuz algılayıcı uygulamalarına yer verilmiş olup, kullanılması ile elde edilecek faydalar, uygulama alan ve şekilllerine göre karşılaşılabilecek problemler anlatılmıştır. Kablosuz algılayıcıların haberleşmelerinde ana unsur olan Kablosuz Algılayıcı İşletim Sistemi üzerine araştırmalarımız devam etmektedir, neticesinde progralanabilir özellikte olan bu donanımlara özel yeni bir işletim sistemi yazmak hedeflenmiştir. Çok küçük boyutlarda olması gereken bu yazılımın ölçüm yapma, ölçülen veriyi sayısala dönüştürme, veri saklama, veri iletimi, veri/paket alma ve veri/paket yorumlama gibi temel ve sıralı işlemleri kapsayan küçük bir çekirdekten oluşması yeterli olacaktır. Kablosuz haberleşme iletişim protokolü ile ilgili elde edilecek yenilikler ve geliştirilecek işletim sistemi sayesinde bu teknolojiye büyük girdi sağlanacağı düşünülmektedir.

Bu çalışmada ayrıca kablosuz haberleşme ve plansız ağ'ların benzetiminin yapılması ile ilgili OMNeT++ benzetim programı ve onun hareketlilik modülü ile Linux (Fedora Core 4) tabanlı yapılan çalışmalar anlatılmıştır. İleride bu alanda çalışmak isteyenlere yardımcı olmak maksadıyla OMNeT++ kurulum ve kullanımı, simülasyon işlem basamkaları, OMNeT++ üzerinde ağ ve modül oluşturma, istatistik elde etme vb. ile yapılan örnek bir çalışma anlatılmıştır. Plansız bir ağa yönelik haberleşmenin benzetimi OMNeT++ Mobility modülü ile yapılmış olup, bu kapsamda Taşma ve IEEE 802.11 iletişim kuralları analiz edilmiştir. Elde edilen

sonuçlar daha iyi bir haberleşme yapabilmek amacı ile değerlendirilmiş ve önerilerde bulunulmuştur. Kablosuz haberleşme iletişim kurallarının analizine taşma protokolü ile başlanmış bu amaçla oluşturulan örnek bir ağ üzerinde inceleme yapılmıştır. Simülasyon neticelerine dayanılarak taşma protokolünün kablosuz haberleşmede kullanılması ile karşılaşılan durumlar analiz edilmiş, problemler için çözüm önerilerinde bulunulmuştur. IEEE 802.11 iletişim kuralının yine aynı şekilde plansız bir ağ üzerinde benzetimi yapılmış ve elde edilen sonuçlar ile iyileştirilmesi gereken noktalar tespit edilmiştir; bu iletişim kuralı ile kablosuz haberleşmenin verimliliği ölçülmüş, iyileştirilmesi için denetimli ve enerji kontrollü bir iletişim kuralı geliştirmek hedeflenmektedir.

Mevcut iletişim kurallarından yola çıkılarak, plansız ağlarda kablosuz haberleşme yöntem ve ilkelerine yönelik geliştirilmesi hedeflenen yeni iletişim kuralının kablosuz algılayıcılar ve ağında kullanılması ile, bu çalışmada elde edilen istatistiklere nazaran daha verimli sonuçlar vereceği sanılmaktadır. Bu konunun devam edecek akademik çalışmalarımızda temel olması ve hayata geçirilmesi amaç edinilmiştir.

2. KABLOSUZ ALGILAYICILAR

Kablosuz algılayıcılar ve kablosuz algılayıcı ağları yeni bir teknolojidir. Kablosuz yerel alan ağ'larda (WLAN) görülen gelişmeler ve düşük güçlü kablosuz haberleşme olanaklarının artması sonucu bu teknolojiye olan ilgi artmıştır. "Tanecik" adı verilen ve fiziksel boyutları oldukça küçük olan yeni nesil algılayıcıların kablosuz haberleşebilme ve programlanabilme özelliği ile kullanım alanı daha da artmıştır. Tanecikler bilgisayar ağları ile bütünleştirilerek kullanıcılarına ek özellikler kazandırabilmektedir.

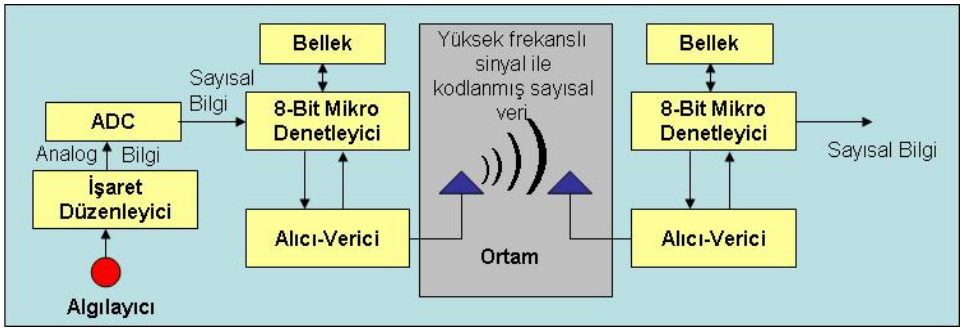
Şekil 2.1: Tanecik (Mote)

Bilgi teknolojileri endüstrisinde en son dürtü kablosuz haberleşme aygıt ve sistemleridir. Ayrıca teknolojide yaşanan darboğazların çoğunluğu, kablosuz algılayıcıların kullanımı ile çözüme kavuşturulmuştur. Araştırmalarda kablosuz algılayıcı ağları [1]-[2]-[3] artarak ilgi duyulan bir konu olmuştur. Bilindiği gibi internet geniş bir kullanıcı kitlesine farklı bilgi formlarını taşımayı sağlamış ve bu yüzden iş, savunma, eğitim, endüstri, araştırma ve bilimde devrim yapmıştır. Algılayıcı ağları [4]-[5] etrafımızdaki fiziksel olayların ölçümünü kolaylaştırır, ayrıca bir uygulama aracılığı ile veri elde etme, biriktirme ve işleme görevi de yapar. Güncel uygulama alanları savunma endüstrisi, tarım, çevre ve doğal ortamı (doğa olaylarını) izleme [4]-[6], sağlık hizmetleri [7], ulaşım [8], üretim [9] ve arama ve kurtarma [10] faaliyetlerinden oluşmaktadır.

Kablosuz haberleşme teknolojisinde meydana gelen en son gelişmeler ile plansız (ad-hoc) ağların yönlendirme ve protokollerinde görülen gelişmelerin birleşmesi

sonucunda, “kablosuz algılayıcı ağları”na yönelik yeni çalışmalar ortaya çıkmıştır. Bir kablosuz algılayıcı ağındaki donanım bileşenleri, genel olarak kablosuz algılayıcılar, aktarıcılar, uygulama ve depolama istasyonları ve gözlem terminallerinden oluşur.

Tipik bir kablosuz algılayıcı ağı, bir ana istasyon ve ortam içerisine dağıtılmış düğümlerden (nodes) oluşur. Her bir düğümden algılama yapması ve elde ettiği bilgileri iletmesi beklenir. Bir düğümdeki bilgi, ya doğrudan ya da çok atlamalı (multi-hop) biçimde, yönlendirmeli olarak, ağdaki diğer birkaç düğüm üzerinden ya da doğrudan ana istasyona iletilebilir.



Şekil 2.2: Kablosuz Algılayıcıyı oluşturan bileşenler ve iletişim

Burada kullanılan düğüm’e literatürde “tanecik” (mote) [1]-[3] adı verilir (Şekil 2.1). Herbir tanecik; batarya birimi, algılayıcı, işaret düzenleyici, ADC, mikro denetleyici, bellek, alıcı-verici, anten ve işletim sisteminden meydana gelmektedir (Şekil 2.2). Crossbow Tech. Inc. [1] tarafından geliştirilen taneciklerde nesC (Nested C) tabanlı TinyOS (Tiny Operating System) [11]-[12] adı verilen işletim sistemi kullanılmaktadır. Taneciklerin kullanıma hazır olabilmeleri için, kullanıcı isteklerini yerine getirecek görevlerin, TinyOS kullanılarak, tanecik üzerinde kullanıcı tarafından programlanmaları gerekmektedir. Ölçülmek istenilen veri türüne, topolojiye, yönlendirmeye ve protokole uygun olarak programlanabilirler. Bu sayede kullanım yer ve amaçlarına göre ek özellikler kazanmaları mümkündür. Tek bir tanecik programlamasının yanında taneciklerin oluşturduğu bir ağın programlanması da kullanıcı tarafından yapılabilir, böylece bireysel faaliyetler ile birlikte takım faaliyetlerini de gerçekleştirebilirler. Elinizde kullanıma ve geliştirmeye hazır bir

donanım var ve siz bunu kodlayarak ve/veya donanımını geliştirerek mevcut bir sisteme entegre edebilir, onun imkan ve kabiliyetlerini arttırabilirsiniz ya da yeni bir sistemi oluşturabilirsiniz. Mevcut teknolojide eksikliği duyulan hususların, özellikle kablolamaya dayalı problemlerin, kablosuz algılayıcıların kullanılması ile çözüleceği düşünülmektedir. Muhakkak ki kablosuz algılayıcıların, özellikle kablosuz haberleşme yeteneklerinin, geliştirilmeye ihtiyaçları vardır. Dikkat edilmesi gereken temel unsurlardan birisi de kaynak yönetimi olup, işlem kapasitesi, bellek, güç gibi kaynaklar uzun süreli çalışma ve verimlilik göz önüne alınarak kullanılmalı ve algılayıcılar buna uygun algoritmalar kullanılarak programlanmalıdırlar.

Tanecik tarafından algılama ile elde edilen veri ana istasyona gönderilir, daha sonra bu bilgi işlenir ve gerekli olaylar tetiklenir. Veriyi göndermek için iki mümkün durum vardır. Düğüm ana istasyon haberleşme mesafesi içinde ise, veri doğrudan ana istasyona iletilir, değilse plansız (ad-hoc) ağ içerisine bırakılır ve platform üzerinden çok-atlamalı (multi-hop) yöntem kullanılarak iletilmesi sağlanır. Çok-atlamalı yöntemde her tanecik bir aktarım terminali olarak görev yapabilir ve aktarımlar ile verinin ana istasyona ulaşması sağlanır. Burada taneciklerin aktarım yaparken kullanacakları yönlendirme algoritması denetimli olmalıdır; tanecik bu veriyi daha önce aktarmadığını sorgulamalıdır. Veri aynı tanecik üzerinden sadece bir defa geçmelidir; aksi durumda tanecik, aynı veriyi yönlendirmek amacı ile, birden fazla aktarım için fazladan enerji harcar. Enerji ve zaman yönüyle verimli olması istenilen, çok-atlamalı bir sistemi gerçekleştirmek için, en kısa yoldan ulaşımı sağlayacak uygun yönlendirme [5] ve daha az güç tüketimi sağlayacak algoritma tercih edilmelidir.

Taneciklerden gelen veriler bir arabirim üzerinden ya da bir istasyon aracılığı ile sunumcu bilgisayar statüsünde olan ana istasyona ulaştırılır. Bu haberleşmede alma ve gönderme farklı frekanslarda yapılabilir. Ana istasyon ya da alıcı terminal konumundaki bilgisayar üzerine alınan kablosuz algılayıcı verisi, buradan klinik ve/veya hastane bilgi sistemleri ortamına aktarılabilir (Şekil 2.3) ve ihtiyaç duyulan her türlü bilgi sistemi faaliyeti için kullanılabilir.

Şekil 2.3: Kablosuz Algılayıcı ve Bilgi Sistemleri Ağı

Kablosuz algılayıcı ağlarında güç (batarya) yönetimi [13], kablosuz algılayıcıların harcadığı gücü kontrol edebilme yönüyle önem arzetmektedir ve bu alanda yapılan araştırmalar ile güç yönetimi önemli bir boyut kazanmıştır. Düşük güçlü algılayıcılar ile RF alıcı ve vericiler bu alandaki anahtar uygulamalar için çok önemlidir. En gelişmiş güç yönetimi "uyku modu" ve "uyanık kalma modu" çevrimlerini içerir. Bir tanecik veri toplarken mA seviyesinde ya da uyku modunda µA seviyesinde akım çeker [12]. Bu kapsamda yapılan çalışmalar [14] ve gelecek kablosuz algılayıcı ağ teknolojilerine yönelik yapılan fizibilite araştırmaları [15] halen devam etmektedir.

Üzerinde durulması gereken önemli noktalardan biri de güvenliktir. Kablosuz algılayıcıların ve ağının kullanıldığı alana göre, ölçüm yapılarak elde edilen veri, yüksek önem derecesine sahip olabilir. Böyle bir verinin, başlangıç kademesi olan ölçme işleminden, ara kademe olan aktarım işlemlerine ve uygulama kademesi olan bilgi sistemleri faaliyetlerine kadar, her türlü bozucu ve istenmeyen unsurlara karşı güvenliği sağlanmalıdır. Örneğin, böyle bir sistem tıbbi bakım için kullanılıyorsa, ölçüm değeri, bakım gören hastanın hayati işaret verilerine yönelik olacaktır ve hastanın tedavisinde önemli rol oynayacaktır.

3. OMNeT++

Bölüm 3 için [16], [17] ve [18] numaralı kaynaklardan yararlanılmıştır.

OMNeT++ nesneye-yönelik (object-oriented) modüler bir ayrık olay ağ benzeticisidir [16]. Aşağıdaki amaçlar için kullanılabilir:

- Haberleşme trafiğinin modellenmesi
- İletişim kuralı modelleme
- Ağ modellemesi
- Çok işlemcili ve diğer dağıtık donanım sistemlerini modelleme
- Donanım yapılarını inceleme
- Karmaşık sistemlerin performans durumlarının değerlendirilmesi
- Ayrık olay yaklaşımının elverişli olduğu diğer sistemlerin modellemesi.

Bir OMNeT++ modeli hiyerarşik olarak iç içe yuvalanmış modüllerden oluşur. Modüllerin iç içe yuvalanmaları sınırlı değildir, kullanıcıya model yapı içinde gerçek sistemin lojik yapısını yansıtma imkanı verir. Modüller mesaj geçişi yolu ile haberleşirler. Mesajlar keyfi olarak karmaşık veri yapılarını içerirler. Modüller, ya doğrudan kendilerinin hedefine ya da kapılar ve bağlantılar arasından önceden tanımlanmış bir yol boyunca mesajları gönderirler. Modüllerin kendi parametreleri vardır ve bunlar modül davranışını özelleştirmek (customize) ve modelin topolojisini programlamak (parameterize etmek) için kullanılırlar.

Modül hiyerarşisinin en alt düzeyinde bulunan modüller, davranışları belirler. Bunlar basit modüller olarak isimlendirilirler ve C++ kütüphaneleri kullanılarak programlanabilirler. OMNeT++ benzetimleri farklı amaçlar için çeşitli kullanıcı arabirimlerini ön plana çıkarabilirler: hata ayıklama, gösterim ve toplu yürütme. Gelişmiş kullanıcı arabirimleri, simülasyon çalışmasına yönelik kontrol etme ve model içindeki değişkenlere/nesnelere yönelik olarak, kullanıcıya değişiklik yapma

imkanı tanır ve bu bir modelin nasıl çalıştığını göstermede oldukça kullanışlıdır. Kullanıcı arabirimleri gibi benzeticiler ve araçlar da portatiftirler: çeşitli C++ derleyicileri kullanılarak Windows ve bazı Unix sürümleri üzerinde çalıştırılabilirler.

OMNeT++ aynı zamanda paralel dağıtık benzetimi de destekler. OMNeT++, paralel dağıtık bir benzetimin bölümleri arasındaki haberleşme için çeşitli mekanizmalar kullanabilir, örneğin MPI (Message Passing Interface) ya da pipe (kanallama). Paralel benzetim algoritması kolay bir şekilde genişletilebilir ya da yeni bir tane dahil edilebilir. Modeller, paralelde çalıştırılmak için herhangi bir özel alet düzeneğine gerek duymazlar, bu yanlızca bir konfigurasyon konusudur. OMNeT++ paralel benzetim algoritmalarının sınıf gösterimi için de kullanılabilir, çünki simülasyonlar olup biten hakkında detaylı geri besleme sağlayan GUI altında bile pararlelde çalıştırılabilirler.

OMNEST, OMNeT++'in ticari olarak desteklenen bir sürümüdür. OMNeT++ yanlızca akademik ve kar amacı gütmeyen kullanımlar için ücretsizdir. Ticari amaçlı çalışmalar için OMNEST yazılımı Omnest Global Inc firmasından (http://www.omnest.com/) temin edilebilir.

3.1 OMNeT++ Simülatörü

OMNeT++ bir benzetim programıdır ve sabit, kablolu, dağıtık sistemler (örneğin: bilgisayar ağları, çok-işlemcili sistemler vb.) için tasarlanmıştır. Temel özellikleri :

- Zamanda ayrık özellikte çalışan bir simülatördür. Simüle edilmiş nesneler, zamanın ayrık anlarında mesajları değiş tokuş ederek, birbirleri ile haberleşirler;
- C++ ve Tcl/Tk ile yazılmıştır. Temel avantajlarından biri taşınabilir kodlu olmasıdır: DOS, UNIX ve Windows üzerinde, kullanıcıdan herhangi bir değişiklik yapmasını gerektirmeden, çalışır;
- bazı grafik arabirimler ile kolay hata-ayıklama (debugging) ve değişkenlerin denetimine imkan verir. Aynı zamanda çıktı dosyalarında verinin vektör ve sayısal değerlerini kayıt etmeyi desteklemektedir;
- paralel yürütmeyi desteklemektedir;

- simüle edilen nesneler, modüller tarafından temsil edilir. Modüller yalın ya da birleşik olabilir (modüllerin iç içe koyulma derinliği sınırlı değildir). Modüller mesajlar aracılığı ile haberleşirler (mesajlar doğrudan ya da kapılar vasıtası ile gönderilir). Herbir modül tanımı: bir arabirim tanımı ve bir davranış tanımından oluşur;
- farklı çekirdekler ile başlayan bazı rastgele sayı üreteçleri (ayrıca bazı dağıtımlardan başlayan üreteçleri) vardır;
- Benzetimler .ini dosyası kullanarak kolayca konfigüre edilebilirler. Aynı simülasyonun birden fazla (toplu) uygulamasını (batch executing) farklı parametreler ile çalıştırabilecek özelliklere sahiptir.

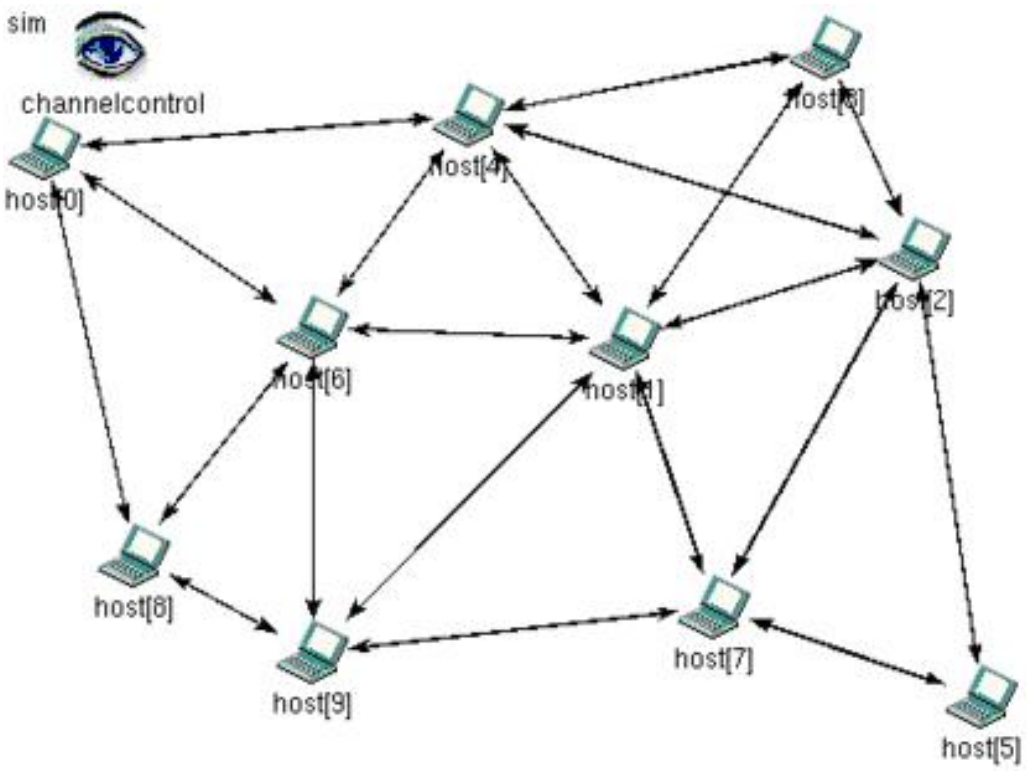
Simüle edilmiş tüm nesneler (modüller, kapılar ve bağlantılar) ya statik olarak (simülasyonun başlangıcında, konfigürasyon dosyasından) ya da dinamik olarak (simülasyon boyunca) oluşturulabilir.

3.2 OMNeT++ Hareketlilik İskeleti

Hareketlilik iskeleti ile, OMNeT++ içinde, kablosuz ve gezgin olma durumlarına yönelik simülasyonlar gerçekleştirilebilir. Çekirdek iskelet içerisindeki düğümün hareketliliğini, bağlantının dinamik olarak yönetimini ve kablosuz kanal için model desteğini içerir [17]. İlaveten, kullanıcının kendi modüllerini gerçekleştirebilmesi için, türetilebilen “basic module”ler sağlar. Bu kavram sayesinde bir programcı “Mobility Framework” (MF) (hareketlilik iskeleti) için kendi protokolünü (iletişim kurallarını), gerekli arabirim ve malzemelerin birlikte çalışabilirliği ile uğraşmaksızın, kolaylıkla geliştirebilir. Bu iskelet aşağıdaki benzetimler için kullanılabilir:

- değişmez (sabit) kablosuz ağlar
- gezgin kablosuz ağlar
- dağınık (ad-hoc) ve merkezileştirilmiş ağlar
- algılayıcı ağlar
- çok kanallı kablosuz ağlar

- hareketlilik desteğine ve/veya kablosuz bir arabirime gereksinim duyan diğer simülasyonlar



Şekil 3.1: 10-Düğümlü simülasyon düzeneği

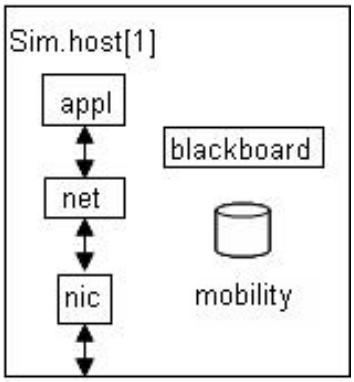
MF (Mobility Framework: Hareketlilik İskeleti)'nin iki çekirdek bileşeni, gezginlik, dinamik bağlantı yönetimi ve OMNET++ içinde gezgin bir host'un (örneğin bir bilgisayar) modeli için, mimari yapıyı oluşturur.

"ChannelControl" modülü host'lar arasındaki tüm muhtemel bağlantıları kontrol ve idame eder [17]. MF içinde yer alan bir OMNeT++ bağlantı halkası otomatik olarak gösterilmez, fakat ilgili host'lar veri değiş tokuşu yapabilir ve birbirleri ile haberleşebilirler. "ChannelControl" modülü yanlızca, birbirlerine mani olma ihtimali olan, tüm host'ları birbirlerine bağlar. Bir haberleşme halkası, kendi tümleyicisi tarafından, kolaylıkla tanımlanır: Bibirlerine bağlı olan host'lar kesinlikle birbirlerine mani olmazlar. Bu kavram sayesinde bir host paketinin tümünü alabilir ve kendi alıcı-vericisi imkan dahilinde algılama yapabilir.

Fiziksel katman, alınmış sinyalin sağlamlığına bağlı olarak, veri paketinin işlenip işlenmeyeceğine ya da onun gürültü olarak davranıp davranmayacağına dair o vakit bir karar vermek zorundadır. Bağlantı yönetimi üzerine diğer detaylar bölüm 3.7.2'de anlatılmıştır.

Gezgin bir host'un iç yapısı şekil 3.2'de gösterilmiştir. ISO/OSI katmanları standartından ayrı olarak ayrıca bir "Mobility" modülü ve "Blackboard" olarak isimlendirilen bir modül bulunmaktadır. "Mobility" modül host'un coğrafi pozisyonunu bulur ve onun hareketlerini işler. İlerleyen bölümlerde (Bkz. Bölüm 3.7.1) hareketlilik yapısı ile ilgili detaylı tanımlama verilecektir.

"Blackboard" modülü çapraz katman haberleşmeleri için kullanılır. Host'un güncel enerji durumu gibi birden fazla katman ile ilgili bilgi sağlar, dış görünüşü ya da radyo ünitesinin durumunu görüntüler. Diğer tüm modüller ISO/OSI katman ile ilgili olan fonksiyonları yerine getirirler. Bu modüllerin uygulamasına dair detaylar ileride (Bkz. Bölüm 3.4) anlatılacaktır.



Şekil 3.2: Gezgin hostun yapısı

MF ile çeşitli kablosuz gezgin ağların simülasyonu canlandırabilir.

3.2.1 Hareketlillik iskeletine giriş

Bu bölümde MF'nin temel kullanımı açıklanacaktır. OMNeT++ ile programlamaya yatkınlık kazanmak için gerekli klavuz dökümanlar http://www.omnetpp.org/

adresinden temin edilebilir. MF'nin kurulumunu açıklandıktan sonra kendi simülasyon ağının oluşturulması ve modül türetme ile ilgili bilgi verilecektir. Modüller hakkında detaylı bilgi sonraki bölümlerde bulunabilir, ayrıca konu ile ilgili uygulama (API) dökümanlarından da faydalınabilir.

3.2.2 Hareketlilik iskeletinin kullanımı

Hareketlilik iskeletini kullanmanın bir çok yöntemi vardır. Hangisinin en uygun yöntem olduğu MF'yi ne amaçla kullanmak istediğimize bağlıdır. MF başarılı bir şekilde derlendikten sonra elde edilen kütüphaneyi kendi simülasyonumuz için kullanabiliriz ya da eğer Hareketlilik İskeleti'ne dayanarak yeni bir simülasyon oluşturmak istiyorsak "networks/" ve "core/basicNetworks" dizinleri altındaki örneklere bakabiliriz. Unutulmamalı ki, "lib" dizini kullanıcı profilinde tanımlı olan "LD_LIBRARY_PATH"e ilave edilmelidir, "LD_LIBRARY_PATH" profilde tanımlı değilse oluşturulmalıdır ya da bağlayıcı ile çalıştırmak için MF onları nerede arıyorsa kütüphane dosyaları oraya kopyalanmalıdır.

Birçok farklı seneryolar için ağ örneklerini kurulum dizini altında bulmak mümkün. Ağlar ve kullanılan iletişim kuralı uygulamaları hakkındaki bilgiler API dökümanstayonu ("doc/api/index.html") içinde dökümante edilmiş.

"template" dizini kaynak ya da modüller için şablon dosyalar içermektedir. İhtiyaca uygun dosyalar buradan kopyalanabilir ve ihtiyaç olan yerlerde kullanılarak uyarlama yapılabilir ve kullanıcı kendi kodlarını içlerine ilave edebilir. "Makefile.gen" dosyası olduğu gibi kopyalanabilir ve içindeki "MOBFW" değişkeni "mobility-fw" dizinine işaret edecek şekilde değiştirilebilir ve daha sonra "opp_makemake –f Makefile.gen" komutu çalıştırılarak "Makefile" dosyası oluşturulabilir.

"mobilityfw" kütüphanesi "development/" dizininden itibaren herhangi bir modül içermemektedir. Bu uygulamalara deney gözüyle bakılması gerektiği için kütüphane içine dahil edilmemişler. Netice olarak eğer iletişim kuralı uygulaması geliştirmek istiyorsak ya da olanları kendi uygulamamız için taban olarak kullanacak isek ilgili

dosyaları olduğu gibi kopyalamamız gerekir. “development/” dizin yapısı içindekiler kendi çalışmamızı yapmak için alternatif olacaklardır. Yeni oluşturulan sınıflarını “Makefiles” ile birlikte ilgili dizinler içerisine kayıt etmek için, “make makefiles” komutu çalıştırılabilir.

3.2.3 Dizin mimarisi

Doğru dosyaları bulmamızı sağlayacak olan ve özet tanımlamalar içeren tüm dizinlere ait bir liste aşağıda verilmiştir:

Tablo 3.1: Dizin listesi

Dizin Adı	Muhteviyatı
Mobility-fw/	(root) çekirdek dizini
Bitmaps/	Ağ grafikleri için semboller
core/	iskeletin özü (core:merkezi)
basicMessages/	Temel mesaj sınıfları
basicModules/	Temel modül sınıfların tamamı
basicNetworks/	2 temel ağ örneği
blackboard/	Blackboard maddesi
channelcontrol/	ChannelControl modülleri
utils/	yardımcı uygulamalar
development/	iletişim kuralı deneysel uygulamaları
applLayer/	uygulama katmanı modülleri
Messages/	.msg dosyalarının tamamı
Mobility/	Hareketlilik modülleri
netwLayer/	Ağ katmanı modülleri
Nic/	Ağ arabirimleri modülleri
macLayer/	MAC katmanı modülleri
phyLayer/	fiziksel katman modülleri
Decider/	karar verici modüller
snrEval/	snr değerlendirme modülleri
utils/	yardımcı uygulamalar
doc/	manuel, API, neddoc, ...
İnclude/	Başlık ve .ned dosyaları
Lib/	“libmobilityfw.so” dizini
Networks/	Ağ örnekleri
Protocols/	Test edilmiş iletişim kuralları uygulamaları
template/	Kendi modüllerimizi oluşturmak için şablonlar
testSuite/	Test takımı arasındaki ilişkiler

3.3 Kurulum ve İlk Çalıştırma

Çalışmalarımı OMNet++'nın 3.2.1 sürümü kullanarak yaptım. MF'nin son sürümü http://mobility-fw.sourceforge.net sitesinden indirilebilir, MF dosyaları kurulum yapılmak istenen dizine kopya edildikten sonra aşağıdaki komutlar kullanılarak kurulum gerçekleştirilebilir: "`tar xzf mobility-fw-<surum_no>.tgz`", bu komut ile (tar) sıkıştırılmış kurulum dosyaları açılarak sıkıştırılmış yapıdan çıkartılırlar. Komut ile ilgili detaylı bilgi için "man tar" ile kılavuz dökümana bakılabilir.

"LD_LIBRARY_PATH=/<calisma_dizinimiz>/mobility-fw-<surum_no>/lib" değişken tanımlaması, kullanıcı profil ($HOME/.bash_profile) dosyasına eklenir. Bu sayede, program ihtiyaç duyduğu kütüphane dosyalarına erişim ile alakalı olarak, dosyaların yolu konusunda, hata vermeden çalışır.

```
cd mobility-fw-<surum_no>/ ; make install; make
```

Burada "surum_no" (version) internet sitesinden indirdiğimiz programa ait sürüm numarasıdır. Herşeyin doğru bir şekilde inşa edildiğinden emin olmak için "core/basicNetworks" dizini altındaki örnek simülasyonlar çalıştırılabilir. "doc/hp/index.html" dosyası MF dökümantasyon (API ve Neddoc referans ve klavuz dökümanları) için bir bağlantı sağlamaktadır, bu bağlantı ile dökümantasyona erişim yapılabilir. Aynı zamanda, kendi uygulamamız ve klavuz dökümantasyon ve kodlama için, bağlar (link) da bulmak mümkündür.

3.3.1 Ağ oluşturma

Yeni bir simülasyonu kolaylıkla oluşturabilmek için ihtiyaç duyulacak şablon dosyalar sağlanmış. Bu dosyaları "template/" dizini altında bulmak mümkün. Temelde yapılması gereken, benzetim için gerekli olan, şablon dosyalarını kendi ağ'ımızın dizini içerisine kopyalamak ve onları uyarlamaktır. En önemli tavsiye ve kurallar ile ilgili açıklayıcı bilgiler bu dosyaların içerisine yorum olarak katılmış. Yeni bir ağ oluşturmak için "yourNetwork" (oluşturduğumuz ağ'ın ismi) ile ayni

isimde bir dizin olusturabiliriz. “yourNetwork.ned”, “YourHost.ned” (Host:kullandığımız bilgisayarın adını ya da kendisini temsil etmektedir) ve Makefile.gen (ve eğer gerekli iseYourNic.ned) dosyalarını bu ağ dizini içerisine kopyalamalıyız. Bu dosyalara seçtiğimiz herhangi bir isim verilebilir ve aynı zamanda bu dosyalar içindeki modül isimlerine de uygun şekilde uyarlanabilirler. Kendi modülümüzü oluşturmak istersek eğer, uygun şablon dosyalarını “yourNetwork” dizini altına kopyalanabilir ve bunlara uygun isimler verebiliriz. Bunların içerisine yazılacak kodlar bir sonraki bölümde (Bkz. Bölüm 3.3.2) anlatılmıştır.

“Makefile.gen” dosyası içerisine, kendi “mobility-fw" dizinimizin yolunu “MOBFW” değişkeni olarak set etmeliyiz, bu sayede Makefile, MF kütüphanesine ait başlık ve ned dosyalarını bulabilir. “YourHost.ned” dosyası içinde, kullanmak istediğimiz NIC ve iletişim kuralı katmanı olarak kullanmak istediğimiz modülleri bildirmek zorundayız. Eğer kendi “Nic Module”ümüzü oluşturmak istersek “YourNic.ned” dosyasını da kopyalamak ve değiştirmemiz gerekir. Herşey hazır olduğunda “make –f Makefile.gen” (-f:force) komutunu çalıştırabiliriz. “Makefile” dosyası oluşturmak için “make” komutu çalıştırılmalıdır.

3.3.2 Modül oluşturma

Tüm basit modüller için üç dosya oluşturulmalıdır: .ned, .h ve .cc dosyaları. Eniyi tercih, modüllerimizi “yourNetwork” dizini içine koymaktır. “YourMacLayer” şablonundan yeni bir MAC modülü oluşturmayı örnek olarak yapmaya çalışalım, diğer tüm modüllerin usülleri aşağı yukarı aynı olacaktır. Şablon dosyaları zaten ana yapıyı içermektedir, yanlızca değişiklik yapılması ve fonksiyonlar ile donatılmaları gerekir. En önemli ipuçları, kurallar, tavsiyeler ve öneriler dosyalar içerisinde yorum olarak (# ile başlayan satırlar) verilmiştir.

“YourMacLayer.ned” dosyasına “BasicMacLayer” modülü tarafından gerek duyulur; bu dosya zorunlu olan kapıları ve parametreleri içerir. Ayrıca bu dosyaya ile kullanıcı kendi modülü için ilave parametreler eklemesi yapabilir ve bu dosya ile modül isimleri olmasını istediğimiz şekilde değiştirilebilir. Eğer varolan bir iletişim

kuralı uygulamasına (örneğin Mac802.11.ned) rağmen bir “BasicMacLayer”dan kullanıcı kendi modülünü türetemiyorsa, gerekli parametreleri ilave etmemiş demektir. “YourMacLayer.h” dosyası, “BasicMacLayer” sınıfı ve “Module_Class_Members() makrosunun türetilmiş sınıf tanımlamalarını zaten içermektedir.

Önceden varolan bir iletişim kuralı uygulamasını kullanarak kullanıcı kendi modülünü oluşturmak isterse, “BasicMacLayer”’ı (örneğin Mac802.11 ile) değiştirmelidir. Sonra kendi modülü için uygun gördüğü bir isim ile “YourMacLayer”ı “bul ve değiştir” ile değiştirebilir. O zaman bu dosya basit bir şekilde tüm harici fonksiyonlar ve ihtiyaç duyulan değişkenler ile genişletilmiş olur.

“Blackboard” kullanmak için “blackboard*” fonksiyonu aktif yapılmalıdır, bunun için ilgili satır önündeki “#” sembolü silmek yeterli olacaktır. “BasicMacLayer.cc” dosyası, modüle fonksiyonellik katmak için, temel mimariyi içermektedir. İlk olarak, başlık dosyası içinden seçilen sınıf adı ile “YourMacLayer” “bul ve değiştir” yaparak değiştirilmelidir. “Define_Module()” makrosuda aynı zamanda dahil edilebilir. Kullanıcı eğer kendi .ned dosyasını kullanmak istemiyorsa, bunu “Define_Module_Like()” makrosunu kullanarak değiştirebilir.

“handleUpperMsg()” ve “handleLowerMsg()” fonksiyonları zorunlu yapı dahilindeki örnek parametrelerin yanında “sendUp()” ve “sendDown()” fonksiyonlarını da içerir. “Blackboard”un kullanılması için “blackboard*” fonksiyonunun aktif yapılması (# sembolünü kaldırarak) ve kodlanması gerekir. Olması gereken yapı zaten mevcuttur, bunda değişiklik yapılmaması yeterli olacaktır (Bkz. Bölüm 3.6).

3.4 Simülasyonumuzu İnşa Etme

Bu bölümde Hareketlilik İsketi’nin arkasındaki temel kavramlar açıklanmıştır. Sınıf hiyerarşisi açıklanmış ve “Basic*” modüllerinin ilgili fonksiyonları tanıtılmıştır. Modüllerin tek tek bütün fonksiyonları, detaylı tanımlamaları ile, ilerleyen bölümlerde (Bkz. Bölüm 3.5-3.7) verilmiştir.

3.4.1 Temel modül kavramı

Tüm “Host” alt modül sınıfları ortak taban olarak bir “BasicModule” sınıfına sahiptir. “BasicModule”ün kendisi, “cSimpleModule” ve fonksiyonelliğinin sürdürülmesini ve “Blackboard” modül üzerindeki bilginin yayınlanmasını sağlamak için, “BlackboardAccess”den türetilmiştir. “Blackboard”un kullanımı ilerleyen bölümde (Bkz. Bölüm 3.6) açıklanmıştır.

3.4.1.1 BasicModule

“BasicModule” OMNeT++’in çoklu başlangıç durumuna getirme aşamalarını kullanır, bu sebeple MF içindeki tüm modüller iki-aşamalı başlangıç durumuna getirme ile gerçekleştirilir. “Blackboard”u kullanabilmek için, tüm “publish” çağrıları “stage 0” da ve “subscribe” çağrıları da “stage 1”de gerçekleştirilmelidir. Böylece, bir parametrenin yayınlanmadan önce oluşmasını sağlayacak imza (onay) önlenmiş olur. Şablon dosyalarında yeralan “initialize(int)” fonksiyonları, “Basic Module”den elde edilen modülün “initialize(int)” fonksiyon çağrısını içermektedir. Başlangıç durumuna getirilmesi gereken “Basic Modules”in zorunlu parametrelerinden olduğu için, bu satır silinmemelidir. İlaveten “BasicModule” “logName()” fonksiyonu sağlar, “logName” fonksiyonu, Host modülüne ait olan OMNeT++ “index()” modülü ve ned modülü adını geri döndürür. “logName()” fonksiyonu, host hata ayıklama kod çıktısının tanımlanması için, hata ayıklama makrosu kullanır. Hata ayıklama mesajlarını yazdırmak için “EV <<”hata ayıklama çıktı fonksiyonu”>>” kullanılabilir.

3.4.1.2 Adres kavramı

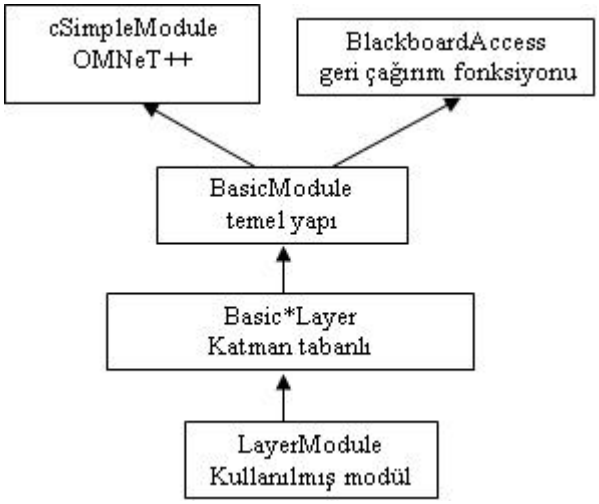
MF içerisinde adresleme yapmak için “id()” fonksiyonu kullanılabilir. “nis” modül “id()”si “mac” adresi olarak kullanılabilir ve “netwLayer” modül “id()”si, ağ katmanı adresi olarak kullanılabilir. “netwLayer id()” aynı zamanda uygulama katmanı adresi olarak da kullanılabilir. Modül adresini elde etmek için sırasıyla “BasicApplLayer, BasicNetwLayer ve BasicMacLayer” modülleri içerisinde bir “myApplAddr(), myNetwAddr() ve myMacAddr()” fonksiyonu sağlanmıştır.

“ChannelControl” modülünün “netw2mac()” fonksiyonu, ağ adresinin MAC adresine çevirimi gibi, basit bir ARP sağlar. Bununla beraber, “BasicNetwLayer”ın “getMacAddr()” fonksiyonunu yeniden tanımlamak sureti ile, bize ait bir ARP fonksiyonelliği gerçekleştirilebilir.

3.4.1.3 İsimlendirme kuralları

“ned” dosyaları içinde modüller için takip edilmesi gereken bazı isimlendirme kuralları vardır. Takip edilmemeleri kodun çalıştırılması sırasında bölümleme hatasına (segmentation fault) sebep olur. Tüm host modülleri, adının karşılığı olan yerde “host ya da Host” karakterlerini içermelidir. Örneğin: baseStationHost, mobileHost vb. Diğer ned modüllerinin, hemen hemen tamamının, isimleri değiştirilmemelidir. Bunlar: “channelcontrol, blackboard, net, phy, snrEval”dir.

3.4.1.4 Basic* modül



Şekil 3.3: Genel türetme yapısı

“BasicModule”den itibaren tüm katmanlar için bir “Basic*” modül verilmiştir. “Basic*” modül kavramı, yapılması zorunlu fakat fonksiyonellik açısından önemi belirsiz görevleri organize eden, taban sınıfına sahip olmak demektir. “Mobility Framework” modülünün türetilmiş genel yapısı Şekil 3.3’de gösterilmiştir.

“Basic*” modüller, “BasicNetwLayer” mesajlarını, üst ve alt katmanlar içine ve içinden gönderme kabiliyetine sahiptirler, yönlendirme fonksiyonelliğine henüz sahip olmamaları nedeni ile oldukça basit bir fonksiyonellik sağlarlar. Farklı katmanların modülleri simülasyonun belirli ihtiyaçlarına uyarlanabilmelidir. Bu amaca hizmet etmasi amacı ile, “handle*Msg() ve convenience” gibi iki tür fonksiyon tanımlanmış.

3.4.1.5 Handle*Msg()

“handle*Msg()” fonksiyonları, iletişim kuralı fonksiyonelliğini içermektedir. Bir mesaj gelir ve bilgiyi gerek duyulan yerlere, gerekli tüm işlemleri içererek, yönlendirir. Üç farklı mesaj olayı türünü işlemek için, üç farklı fonksiyon sağlanmıştır:

handleSelfMsg: Zamanlayıcıları OMNeT++ mesajları içinde uygulamak için, en kolay yöntemdir. “handleSelfMsg()” zamanlayıcı bağlantısı olan herşeyi işleme ve zaman aşımı olduğunda eylemleri başlatma yeridir.

handleUpperMsg: Bu fonksiyon, üst katmana bir mesaj ulaştığı zaman çağırılır. Mesaj zaten ilgili “n” katmanının biçimine sahiptir. Mesaj işlendikten sonra, eğer gerekli ise alt katmanlara “sendDown()” fonksiyonu ile gönderilir.

handleLowerMsg: Alt katmanlardan gönderilen mesajlar için diğer bir yöntemdir. İşlem yapıldıktan sonra, gerekli ise üst katmana iletilmeleri gerekir. Bu işlem, aynı zamanda kapsülden çıkarma işini de organize eden, “sendUp()” fonksiyonu kullanılarak yapılabilir.

3.4.1.6 Convenience

“convenience” fonksiyonları ortak arabirimlere yardım etmek ve kullanıcıyı arabirim yönetiminden korumak için tanımlanmıştır. Bu amaçla üç fonksiyon sağlanmıştır:

- **encapsMsg**: Bu fonksiyon, üst katmandan bir mesaj geldiği zaman çağırılır. “n+1” katmanına ait bir mesajın “n” katmanı mesajı içerisine giydirilmesi (encapsulation) ile sorumludur. Bu, gerekli tüm başlık bilgisini sağlamayı ve uygulanabilir ise, aynı zamanda “n+1” katmanı başlık bilgisinin “n” katmanı bilgisine dönüşümünü, ifade eder (ağ katmanını uygulamaya dönüştürme ya da aktarım katmanı adresini, bir ağ adresine, dönüştürme işlemi olduğu taktirde). Sonra bu mesaj “handleUpperMsg()” fonksiyonuna geçer.
- **sendUp**: “sendUp()” bir mesaj üst katmanlara gönderilirse çağırılan bir fonksiyondur ve genellikle “handleLowerMsg()” içinden çağırılır. Mesajı “n+1” modül katmanına göndermeden önce, onu kapsülünden çıkartır (decapsulate).
- **sendDown**: Mesajları “n-1” katmanına gönderme işlemi “sendDown()” fonksiyonu ile yapılır. Bazen, ilave meta bilgisini burada işlemek veya sağlamak için, gerekli olabilir. Örneğin bir sonraki sıçramanın gerekli olduğu durumlarda. Ağ katmanı hedef adresi, sonraki-sıçrama ile ilgili olarak, bir mesajın hedef olarak gönderileceği MAC adresi hakkında bilgi içermez, bu nedenle çevirilmesi gerekir (ARP bunu IP için yapmaktadır).
- **sendDelayedUp**: Üst katmana göndermek istediğimiz mesajın zamanını geciktirmek için bu fonksiyon kullanılabilir. Geciktirilebilen mesajın zamanı bir parametre olarak verilebilir.
- **sendDelayedDown**: “sendDelayedUp()” ile aynıdır, sadece geciktirilen mesajlar alt katmana gönderilir.

Bu altı fonksiyon, MF’nin herbir katmanı için, “Basic*” modül içinde (saydam farklılıklar ile) verilir. Genellikle “handle*Msg() ve initialize(int) ve finish()” fonksiyonlarının üçü, bir programcının kendi modülünü ve onun fonksiyonelliğini oluşturmak için, yeniden uygulamak zorunda olduğu fonksiyonlardır. “convenience” fonksiyonları yukarda tanımlanmış görevler için kullanılabilirler ve türetilmiş modüller ile değiştirilemezler.

3.4.2 Mesaj kavramı

“Basic*” modüllerde mesajların sarmalanması (encapsulating) ve kapsülden çıkartılmaları (decapsulating) gibi temel fonksiyonları tedarik etmek için, her

katmana yönelik mesaj formatlarını kararlaştırmak zorundayız. Tedarik edilen mesaj türleri uygun katman için gerekli olan çok önemli alanlara sahiptir. Bu alanlardan dolayı bu mesaj türleri zorunludur ve yanlızca geliştirilebilirler, fakat değiştirilemezler. Tüm mesaj sınıfları ve parametrelerinin listesi aşağıda verilmiştir:

Tablo 3.2: Fiziksel katman mesajı ve parametreleri

AirFrame	Fiziksel Katman Mesajı	
	Psend	güç gönderimi
	channelId	kanal mesajı onun üzerine gönderilir
	İd	Pozisyon elde etmeyi başlatanın id'si
	Duration	mesajı kanal üzerinden göndermek için gerekli olan süre
	SnrControlInfo	Control Information sınıfı; Sn(i)r bilgisini karar vericiye geçirmek için kullanılır

Tablo 3.3: MAC mesajı ve parametreleri

MacPkt	MAC (Medium Access Control) Mesajı	
	DestAddr	Hedef MAC adresi
	SrcAddr	kaynak MAC adresi
	channelId	kanal mesajı üzerine gönderilir

Tablo 3.4: Ağ Katmanı mesajı ve parametreleri

NetwPkt	Ağ Katmanı Mesajı	
	destAddr	hedef ağ adresi
	SrcAddr	kaynak ağ adresi
	SeqNum	sıra numarası
	Ttl	(time to life) ömür süresi
	MacControlInfo	Control Information sınıfı; sonraki sıçramanın adresini MAC iletişim kuralına söylemek için kullanılır

Tablo 3.5: Uygulama Katmanı mesajı ve parametreleri

ApplPkt	Uygulama Katmanı Mesajı	
	destAddr	hedef uygulama adresi
	SrcAddr	kaynak uygulama adresi

3.4.2.1 Mesaj oluşturma

İlave parametre ihtiyacı olduğunda, kullanıcı OMNeT++ içindeki temel mesaj sınıflarından birini kullanarak, kendi mesaj sınıfını aşağıdaki gibi türetebilir:

```
cplusplus {
{
#include "NetwPkt_m.h"
}
};

class NetwPkt;
message YourPkt extends NetwPkt
{
   fields:
        int     extraField1;
        double  extraField2;
};
```

".h" içeren dosya adının, "_m" ile, bu dosyada olduğu gibi değişmesi gerekir, OMNeT++ bir ".msg" dosyası oluşturur.

3.4.2.2 Mesaj kullanma

Yeniden oluşturulmuş bir mesajı kullanabilmek için, herşeyden önce "regPrototype()" fonksiyonu, ilgili katmanın modül sınıfı başlık dosyasında, yazılmış olmalıdır. Kendi ağ katmanımıza ait mesajı kullanabilmek için, aşağıdaki kod, "YourNetwModule.h" dosyasına ilave edilmelidir.

```
virtual void regPrototype() {
   if(netwPrototype) delete netwPrototype;
     netwPrototype = new YourPkt;
};
```

Kullanıcı kendi simülasyonu için yeni bir "YourPkt" mesajı oluşturmak isterse, bunu aşağıdaki satırı çalıştırarak yapabilir:

```
YourPkt* pkt = static_cast<YourPkt *>(dupPrototype());
```

“dupPrototype()” fonksiyonu her bir katman için temel modül içinde gerçekleştirilmelidir. Bu adımlar, aşağıdaki çevirme işlemini yapabilmek için gereklidir. Bir mesaj, “handleUpperMsg()” ve “handleLowerMsg()” fonksiyonları içinde, temel mesaj formatında bir parametre olarak verilir. Bu nedenle ilave alanlara erişmek istersek mesaj kendi formatımıza çevirilmelidir. “YourNetwModule” için aşağıdaki kod kullanılabilir:

```
void YourNetwModule::handleUpperMsg(NetwPkt *packet)
{
  YourPkt *pkt = check_and_cast<YourPkt *>(packet);

  // do something with the message...
}
```

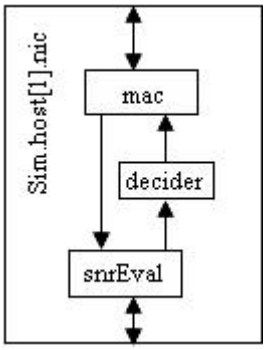
3.4.2.3 Kontrol bilgisi sınıfları

OMNeT++ “Control Information classes” tanımlamalarına imkan verir. Sonraki işlem katmanı ile ilgili olan bir mesaja, meta bilgisini eklemek için, kullanılır. Bir katman her nezaman bu mesajı alırsa, çıkarma ve bilgiyi işleme görevini yapabilir. Tabloda görüldüğü gibi, “AirFrame ve NewPkt” böyle bir kontrol bilgisi sınıfına sahiptir.

“SnrControlInfo()” “s(i)nr” bilgisini “decider”a doğru iletmek için kullanılır, bu sayede mesajın “s(i)nr” bilgisine dayalı bit hatalarını hesaplayabilir. “MacControlInfo”, mesajın gönderileceği, bir sonraki sıçranacak MAC adresi bilgisini içerir. MF mesaja eklenen kontrol bilgisini genişletmek için bir yöntem sağlamaz. Eğer ilave meta bilgisini değiştirmek gerekirse, türetilmiş mesaja normal bir parametre olarak ilave edilebilir, her bir mesajda yanlızca tek bir kontrol bilgisine müsade edilmiştir. İhtiyaç duyulduğu taktirde ilave kontrol bilgisi için, “SnrControlInfo” veya “MacControlInfo” kullanılabilir. Mevcut sürümde sadece mutlak gerekli olan bilgi sağlanmıştır. Fakat özellikle “MacControlInfo”, MAC ve ağ katmanları, ki mümkün olduğunca güçlü olması istenilir, arasında bir arabirim olarak çalışır.

3.4.3 Ağ arabirim kartı

Ağ arabirim kartı Nic (Network Interface Card) verici, alıcı, orta kademe erişim mekanizmaları modülasyonu gibi fiziksel katman fonksiyonlarını içerir. MF içinde yer alan "nic" modülü bu yüzden parçalı gibi (snrEval ve decider) bir fiziksel katman ve bir MAC katmanına (macLayer) bölünmüştür. "snrEval" modülü alınmış bir mesaj için bir miktar sn(i)r bilgisini hesaplamada kullanılır, oysa "decider" modülü bu bilgiyi, bir mesajın kaybolup kaybolmadığı, bit hatalarının olup olmadığı ya da doğru bir şekilde alınıp alınmadığının kararını vermek için kullanır. Bu sebeple "decider" yanlızca alınan mesajlara işlem yapar ve gönderilene yapmaz. Konuyla ilgili bileşik modül, kendi yalın modülleri ile birlikte, Şekil 3.4'de gösterilmiştir.



Şekil 3.4: Nic modülün yapısı

Fiziksel ve MAC bileşenlerinin bir bileşik modül içerisine koyulmasının sebebi kolaylıkla açıklanabilir. En düşük katman iletişim kuralları için, MAC ve fiziksel katman koordine edilmelidir, bu nedenle bir iletişim kuralı için (örneğin IEEE 802.11), uygun bir "decider" modül, uygun bir "mac" modül ve uygun bir "snrEval" modül omalıdır. Bu sayede PHY/MAC iletişim kurallı üzerinde bir yönlendirme iletişim kuralı çalıştırmak istenirse, "host"u inşa etme aşamasında (Bkz. Bölüm 3.3.2) uygun "nic" modülü seçilebilir.

3.4.3.1 SnrEval

"snrEval" modüllerin yapısı diğer modüllere göre biraz farklıdır. "handleLowerMsg()" fonksiyonu, iletim gecikmesini simüle edebilmek için iki

fonksiyona ayrılmıştır. Bu fonksiyonların kullanımı ile ilgili detaylı örnekler "YourSnrEval" şablon dosyaları içinde verilmiştir.

- **handleLowerMsgStart**: Bu fonksiyon alış başladığı anda çağırılır, örneğin ilk bit ulaştığı anda. Alış işlemi başladığında yapılması gerekli olan herşey burada yapılabilir, örneğin SNR değerlerini saklamak için bir SNR-listesi oluşturmak ve onu başlangıç durumuna getirmek, iskeleti alma-tampon belleği içine koymak vb.
- **handleLowerMsgEnd:** Bu fonksiyon bir mesajın gönderimi bittiğinde çağırılır. Burada, mesaj "decider"a verilmeden önce gerekli olan herşey yapılabilir, örneğin mesajın alma-tampon belleğinden dışarı alınması, "sendUp" fonksiyonunun çağırılması vb.

3.5 Fiziksel Katman Modülleri

Tüm "PhyLayer" modüllerinin temel sınıfı "BasicModule"den türetilmiş olan "ChannelAccess"dir. "ChannelAccess"in kanala (örneğin diğer düğümlere) bağlanabilirlik fonksiyonu sağlar. "sendToChannel()" fonksiyonu mesajları kanala bırakmak için kullanılır. Mesajı, Host modülün bağlı olan tüm kapılarına, gönderecektir. "PhyLayer"ın iki ayrı sürümü mevcuttur. İlk sürüm Bölüm 3.5.3'de anlatılmıştır, Host'lar arasında "noktadan noktaya" bağlantı olarak varsayılır ve "P2PphyLayer" olarak çağırılır ve Bölüm 3.5.3'de tanımlanmıştır. Düşünebileceğimiz en yalın "PhyLayer"dır ve özellikle ayrıntılı yayılma ve arabirim modellerine ihtiyaç yoksa kullanışlıdır.

İkinci sürüm ile ilgili olarak fiziksel katman fonksiyonelliği iki alt modüle ayırılmıştır. "PhyLayer" modülü "SnrEval" ve "Decider" alt modüllerine bölünmüştür (Şekil 3.4). SNR hesaplama ve değerlendirme bit hataları ile ilgili karardan ayrılması istenmiştir. Bu kavram, "SnrEval" modüllerini kullanan farklı "Decider" modüllerinin oluşturulmasını kolaylaştırır. Örneğin, doğru eşik değeri ile hesaplanmış SNR ile hata düzeltme fonksiyonu ve aynı "SnrEval" modülünü kullanarak iki modülün karşılaştırmasını yapabilen bir "Decider" modül tanımlanabilir.

“snrEval” modülün tipik parametreleri “transmitterPower, carrierFrequency ve pathLossAlpha”dır. Bunlar “snir” değerleri gibi bir sinyalin zayıflamasını hesaplamada kullanılabilir. “ChannelControl” modülü aynı zamanda bu parametrelerin (pMax, carrierFrequency, alpha) uyarlamalarına sahiptir, fakat onlar “snrEval” parametreleri tarafından birbirini etkilemeden kullanılırlar.

“ChannelControl” modülü yanlızca potansiyel olarak birbirlerine müdahale eden düğümlerin mesafesini hesaplar, örneğin Omnet bağlantılarının kurulmasındaki gibi. Kullanıcı sinyal dayanıklılığının ne kadar olduğunu tanımlayabilir, alınmış güç seviyeleri sinyal zayıflama eşik değeri (sat:signal attenuation threshold) nedeni ile ihmal edilmiş olabilir, örneğin “sat”dan daha zayıf olan her sinyal ihmal edilir. Sonuç olarak aynı güç –ve frekans- değerleri “snrEval ve ChannelControl” parametreleri için kullanılabilmelidir. Eğer farklı verici güç seviyeleri kullanılır ise, maksimum güç seviyesi “pMax” için kullanılması gerekir. “ChannelControl” modül ve MF içindeki bağlantılar hakkında daha fazla bilgi Bölüm 3.7.2’de bulunabilir.

3.5.1 SnrEval

“SnrEval” modülü alınan tüm mesajlar için bir aktarım gecikmesini simüle eder ve aynı zamanda SNR bilgisini hesaplar. “BasicSnrEval” yayılım gecikmesini açıklamaz. SNR bilgisi “SnrList” içinde depolanır. Herbir “SnrList” girişi tarih bilgisi ve bu tarih için bir SNR değeri içerir. “SnrEval” modülleri için temel fonksiyonlar Bölüm 3.4.1’de tanımlananlardan biraz farklıdır. “handleLowerMsg()” fonksiyonu “handleLowerMsgStart() ve handleLowerMsgEnd()” bölümlerine ayrılmıştır. İlaveten bir “bufferMsg()” ve bir “unbufferMsg()” fonksiyonu tanımlanmıştır.

Bir mesaj alınır alınmaz “handleLowerMsgStart()” fonksiyonu çağırılır. Bu fonksiyonda SNR bilgisini tutmak için bu çerçeveye uygun bir “SnrList” oluşturulmuş olmalı ve başlangıç girişi ilave edilmiş olmalı. İlaveten alıcı tampon belleğindeki diğer tüm mesajların SNR bilgileri güncellenmiş olmalı. Daha sonra mesaj, aktarım gecikmesini simüle etmek için, tampon belleğe alınır (“bufferMsg()” fonksiyonu). Bu zaman zarfınca diğer mesajlar ulaşabilir ve tampon belleğe alınmış

mesaj ile çatışabilirler ve bu durum bu mesaj için SNR'daki bir değişimi göstermeye yönelik olarak ilave SnrList girişlerine yol açabilir. Aktarım tamamlandıktan sonra (örneğin mesaj tamamen alındıktan sonra) "unbufferMsg()" fonksiyonu mesajı tampon belleğe almaz. "handleLowerMsgEnd()" fonksiyonu, mesaj "Decider" modüle doğru yukarı geçirilince, doğruca çağırılır. Bu noktada mesaj alınmış ve tampon belleğinden silinmiş olmalıdır ve hesaplanmış SNR değerlerini içeren SnrList, sendUp() fonksiyonuna doğru bir argüman (bağımsız değişken) olarak geçirilmelidir.

Alma işleminin başlangıcında SNR'ı yeniden hesaplayabilmek için, mesaj başına yanlızca bir (ortalama) SNR hesaplanmasından dolayı, "SnrEval" modüllerini gerçekleştirmenin ayrı yolları vardır, ilave bir mesaj her zaman SNR değerlerinin tam bir listesine sonuç olarak ulaşır. Bu kavram tüm bu farklı modelleri çok karmaşık hale getirmeden, fakat aynı zamanda karmaşık modelleri desteklemek için, komplike olan modelleri etkin kılabilir.

3.5.2 Decider

"Decider" modülü yanlızca kanaldan (örneğin alt katmandan) gelen mesajları işler. Üst katmandan gelen mesajlar "Decider" modülden atlatılır ve doğrudan "SnrEval" modüle doğru uzatılırlar. Bit hataları ya da kaybolan mesajlar hakkındaki kararlar yanlızca kanaldan gelen mesajlar hakkında yapılmış olmalıdır. Sonuç olarak üst katmandan gelen mesajların "Decider" modülde işlenmesi gerekli değildir. "Decider" modül "SnrEval" modül tarafından oluşturulan "SnrList"i alır ve SNR değerlerini bit hatalarına göre dönüştürür.

Mümkün olan en basit uygulamada SNR değerleri bir SNR eşik değeri ile karşılaştırılacaktır. "SnrList"in kapsadığı SNR değerlerinden en az bir tanesi SNR eşik değerini geçerse, mesaj bit hatalarından dolayı düşürülür. Elbette daha karmaşık uygulamaların olması da mümkündür. Daha evvel ifade edildiği gibi Decider aynı zamanda hata algılama ve/veya kod düzeltme için uygulama yeri de olacaktır.

3.5.3 P2PPhyLayer

“P2PphyLayer”ın büyük avantajı alt bölümlere ayrılmış olan “SnrEval” ve “Decider” modüllerinden çok daha hızlı olmasıdır. Hızın bedeli karmaşık girişim modellerinin ve orta kademe erişim tekniklerinin simüle edilememesidir. P2PhyLayer yanlızca basit bit hata olasılığı “pBit”i (genellikle omnetpp.ini’den) alır. Bit hata olasılığı, bit hatalarını ve mesaj kayıplarının mümkün olan tüm çeşitlerini kapsar. Aynı zamanda çarpışmadan kaynaklanan mesaj kayıplarına karşı da sorumludur. Netice olarak, karmaşık ara kademe erişim teknikleri ve arabirim modelleri artık gerekli değildir. Avantajı, mesajların doğrudan istenilen sonraki sıçrama noktasına doğru gönderilebilmeleri ve etrafta bağlı olan tüm komşulara yayımlanmalarına gerek olmamasıdır. Bu sayede mesajların gereksiz yere çoğaltılmalarını, gönderilmelerini ve işlenmelerini önlemiş olur.

3.6 Blackboard Kullanımı

“Blackboard” düşüncesi ara katman haberleşmesi için bir örnek sağlamaya yöneliktir. Bazı nesneler yanlızca oluşturuldukları ya da oluşturulacakları katman ile ilgili olmayabilir. Örneğin fiziksel katman (MF içindeki snrEval), bir kanal meşgul olsun ya da olmasın, algılama yapabilir. Eğer MAC iletişim kuralı taşıyıcı duyusuna dayanıyor ise, fiziksel katmanın sahip olduğu bilgiye ihtiyaç duyar. Blackboard bir modüldür ve uygun bilgiyi yayınlar ve sonra ilgili olan herhangi bir modül ona erişim yapabilir. BasicModule, “BlackboardAccess” sınıfından türetilmiştir ve bu sınıf Blackboard’a ve Blackboard üzerinden yayınlanan bilgiyi okuyup imza ile onayladıktan sonra geri çağırım fonksiyonlarına işaretçi döndüren blackboard() fonksiyonunu sağlar. BasicModule her modülün temeli iken, tüm modüller imkan dahilinde Blackboard kullanabilirler.

Bölüm 3.6.1’de Blackboard üzerinden yayınlanan bir parametrenin nasıl imzalanarak onaylandığı anlatılmıştır, bölüm 3.6.2’de konu ile ilgili örnekler verilmiştir. Bölüm 3.6.3’de, parametrelerin değişiklikleri hakkında güncel bilgiler elde etmek için, BlackboardAccess tarafından sağlanan geri çağırım fonksiyonlarının kullanımı açıklanmıştır. Bölüm 3.6.4’de kendi parametremizin nasıl yayınlanacağı anlatılmıştır.

3.6.1 İmzalama/Onaylama

3.6.1.1 Parametrenin imza ile onaylanması

Bir parametreyi imzalayarak onaylamak için, örneğin parametrenin içerik/değer değişikliği ile ilgili bilgi sahibi olmak istenebilir, Blackboard'un "subscribe()" fonksiyonu çağırılmalıdır. Modülümüze bir işaretçi ve bir dizgi kadar değişkeni parametre olarak dahil edilmelidir. Eğer isim ile yayınlanmış parametre yok ise, program buna karşılık gelen bir hata mesajı ile sonlanacaktır. Fonksiyon "BBItemRef" türünün bir ilgisini geri döndürür. İmzanın çıkartılma ya da doğrudan bir parametreyi arama işlemi için bu gereklidir. Örnek:

```
BBItemRef yourRef = blackboard()->subscribe(this,
                                ``nameOfParameter'');
```

Eğer bir parametreyi, modülümüzün başlangıç durumuna getirilme aşamasında, imzalamak istiyorsak "initialize()" fonksiyonundaki durum 1'i yapmalıyız.

3.6.1.2 Parametreden imzanın çıkartılması

Bir parametreden imzayı çıkartmak istersek sadece Blackboard "unsubscribe()" fonksiyonunu çağırmamız gerekir ve modülümüze bir işaretçi ve parametreye değişken olarak "BBItemRef" vermemiz gerekir.

```
blackboard()->unsubscribe(this, yourRef);
```

3.6.1.3 Yeniden yayınlanmış tüm parametrelerin imzalanması

Blackboard üzerinde yayınlanmış, değer atanmış yeni bir parametreyi her seferinde elde etmenin bir olasılığı vardır. Bu fonksiyonelliği kullanmak için Blackboard "registerClient()" fonksiyonu çağırılmalıdır. Değişken modülümüz için bir işaretçidir.

```
blackboard()->registerClient( this );
```

Bu imzayı iptal etmek için sadece “removeClient()” fonksiyonu çağırılmalıdır:

```
blackboard()->removeClient( this );
```

3.6.2 Örnekler

Bu fonksiyonlar ile birlikte, imzalayıcı, üç çeşit imza arasında seçim yapma şansına sahiptir. Aşağıdaki örnekler Blackboard modülünün API ile ilgili kaynağından alınmıştır.

3.6.2.1 İsime göre imzalama

Önceden tanımlandığı gibi, eğer bir imzalayıcı bir parametrenin adını biliyorsa, onu aşağıdaki gibi imzalayabilir:

```
class Foo : public Basic*Module {
  protected:
    BBItemRef ref;
    ...
};
void Foo::initialize(int stage) {
    if(stage==0)
    {
        ...
    }
    else if(stage==1)
    {
        ref = blackboard()->subscribe(this,"routingTable");
        ...
    }
}
void Foo::blackboardItemChanged(BBItemRef item) {
    if (item==ref)
    {
        RoutingTable *rt =
          dynamic_cast<RoutingTable *>(ref->data());
        ...
    }
    else ...
}
```

3.6.2.2 Yayınlanır yayınlanmaz imzalama

Parametreler “initialize(int)” fonksiyonu içinde, çalışma zamanı dışında, yayınlanmazlar. Bir imzalayıcı kendini “registerClient()” fonksiyonu ile kayıt edebilir ve sonra parametreyi yayınlanır yayınlanmaz imzalayabilir.

```
class Bar : public Basic*Module{
   ...
};
void Bar::initialize(int stage) {
    if(stage==0)
    {
        blackboard()->registerClient(this);
        ...
    }
}
void Bar::blackboardItemPublished(BBItemRef item) {
    // if label begins with "nic." and it's a
    // NetworkInterfaceData => subscribe
    if (!strncmp(item->label(),"nic.",4) &&
        dynamic_cast<NetworkInterfaceData *>(item->data()))
    {
        // new network interface appeared, subscribe to
        // it and do whatever other actions are necessary
        blackboard()->subscribe(this, item);
        ...
    }
}
```

3.6.2.3 Gözatma ile imzalama

Blackboard içeriğine göz atılabilir ve belirli bir cinsin tüm parametreleri imzalanabilir. Bu işlem aşağıdaki gibi yapılır:

```
class Zak : public Basic*Module{
   ...
};
void Zak::letsCheckWhatIsOnTheBlackboard(){
    // subscribe to all NetworkInterfaceData
    Blackboard *bb = blackboard();
    for(Blackboard::iterator i=bb->begin(); i!=bb->end(); ++i)
        if(dynamic_cast<NetworkInterfaceData *>((*i)->data()))
```

```
            bb->subscribe(this, *i);
}
```

3.6.3 Bilgilendirilme

Bir imzalayıcı Blackboard geri çağırım fonksiyonları aracılığı ile Blackboard tarafından bilgilendirilir. Onlar, her Basic* modülün dolaylı yönden elde edildiği, BlackboardAccess sınıfı içinde tanımlanırlar, gerektiğinde yeniden tanımlanmalı ve fonksiyonellik ile donatılmalıdırlar. ".cc" şablon dosyaları içine dahil edilirler ve önlerindeki # sembolü kalıdırılmalıdır (yorumsuz yapılmalıdırlar).

"initialize(int) fonksiyonuna benzer geri çağırım fonksiyonları ana modülün yerini tutan fonksiyonu çağırmalıdır. Ana modülün bazı parametreleri de imzalanmış olmalıdır ve ana modül bu parametrelerdeki değişiklik hakkında bilgilendirilmelidir. Sadece geri çağırım fonksiyonlarına yazarsak, ana modülün kodu hiç yürütülmüş olmaz ve o bu değişiklik hakkında bilgilendirilmemiş olur.

Modül hiyerarşisi yürüyorken simülasyonu hızlandırmak için geri çağırım fonksiyonları eğer bir parametreyi işlemişse "true" geri döndürür. Normalde yanlızca imzaladığımız parametreler ile ilgilenir ve ana modülün imzaladığı parametreleri dikkate almayız. Bu sebeple önceden işlenmemiş olan tüm parametreleri işlememiz gerekir. "blackboard*()" geri çağırım fonksiyonlarının genel yapısı aşağıda gösterilmiştir. Normal olarak bunu değiştirmemiz gerekmez. "Enter_Method()" makrosu aşağıdaki kodun "YourModule" ile, sahibi olarak yürütüldüğünden emin olmak için kullanılır.

```
bool YourModule::blackboard*(BBItemRef item) {
  Enter_Method("blackboard*(\"%s\")", item->label());

  if( Basic*Module::blackboard*(item) ){
    print("item already handled by parent class");
    return true;
  }
  // eger islemek istedigimiz parca ise
  if( item == yourItem ){
    // BIZE AIT KODU BURAYA YAZABILIRIZ
    return true;
  }
```

```
    // parca (item) islenmedi ise false geri dondurur
    return false;
}
```

Üç geri çağırım fonksiyonu aşağıda tanımlanmıştır:

```
bool blackboardItemChanged(BBItemRef item);
```

Her ne zaman bir parametreyi imzalasak çağırılan bu fonksiyon değiştirilmiş olur, bu bir parametre değişikliğini etkilemek için kod koyma yeridir.

```
bool blackboardItemPublished(BBItemRef item);
```

Eğer "registerClient()" fonksiyonundan dolayı yayınlama olaylarını imzalarsak, bu fonksiyon Blackboard üzerinden yeni bir parametre yayınlanacağı zaman, her seferinde blackboard tarafından çağırılacaktır.

```
bool blackboardItemWithdrawn(BBItemRef item);
```

Blackboard tarafından geri çağırılan parametre bu fonksiyonu her zaman çağıracaktır.

3.6.4 Parametre yayınlama yöntemleri

3.6.4.1 Parametre

Yayınlanacak bir parametre cPolymorphic'den türetilmiş bir sınıf olarak tanımlanmalıdır (örneğin YourParameter şablon dosyası). Parametre ile ilgili bir referans bildirilmesi gerektiğinden farklı bir "BBItemRef türü de yayınlanmalıdır. Bu BlackboardAccess içinde tanımlanır ve bu nedenle bütün modüller tarafından bilinir. Referans parametresi tanımlayıcı olarak kullanılır ve daha sonraki parametreye erişim yapmak için gerekli olur. Bundan başka, yayınlanacak değeri tutan YourParameter değişkeni tanımlanmalıdır. Bundan dolayı, YourModule sınıfının başlık dosyasında aşağıdaki gibi bir ifade yazılmalıdır:

```
BBItemRef yourRef;
YourParameter *yourParameter;
```

3.6.4.2 Yayınlama

Bir parametre, publish() fonksiyonu çağırılarak yayınlanır. Bu fonksiyon BBItemRef türü için bir referans geri döndürür. Hesaba katılması gereken değerler parametre için bir isim (dizi olarak verilir) ve onun parametreleridir. Her parametrenin benzersiz bir isim olarak yayınladığından emin olmak gerekir, aksi taktirde Blackboard bir hata döndürecektir. Bir parametre yayınladığı zaman aşağıdaki gibi görünür:

```
yourRef = blackboard()->publish(``nameOfParameter'',
                                        yourParameter);
```

Yayılım, initialize() fonksiyonunun "stage 0" evresinde yapılmalıdır. Parametre yayınlamadan önce başlangıç durumuna getirilmeli ve önceden değer içermediğinden emin olunmalıdır.

3.6.4.3 Parametre değişikliği

Bir parametre yeni bir değer alabilir. Parametrenin bu yeni içeriği genellikle Blackboard üzerinden yayınlanabilir. Bunu yapabilmek için öncelikle parametre nesnesine yönelik yeni bir bilgi yazılması gerekir. Daha sonra Blackboard "changed()" fonksiyonu çağırılmalıdır. Hesaba katılması gereken değerler referans ve parametre nesnesidir ve aşağıdaki gibidir:

```
blackboard()->changed(yourRef, yourParameter);
```

3.6.4.4 Parametreyi kaldırmak

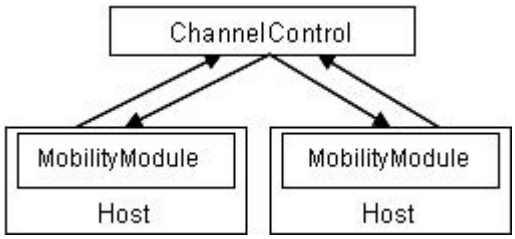
Bir parametreyi Blackboard'dan geri çekmek istersek, Blackboard "withdraw()" fonksiyonu çağırılmalı ve referans olarak bir parametre verilmelidir:

```
blackboard()->withdraw(yourRef);
```

3.7 Hareketlilik Modülleri

3.7.1 Hareketlilik mimarisi

Bu bölümde MF'de kullanılan hareketlilik mimarisi tanımlanmıştır. İlk olarak genel kavram Bölüm 3.7.2'de ve kendi hareketlilik modüllerimizi nasıl oluşturacağımız ise Bölüm 3.7.3'de anlatılmıştır.



Şekil 3.5: Hareketlilik Mimarisi

Hareketlilik mimarisini simülasyona dahil etmekle cevap bulunacak iki önemli soru vardır: İlki, hareketlilik bilgisinin nerede işlem görecek olması ve "Host" hareketlerinin nasıl işlenecek olması, diğeri ise, etkili bir mesafe içindeki bağlantıların nerede ve nasıl dinamik olarak işleneceğidir. Host modül içinde dağıtık yapıdaki hareketliliği işletelim; emirlerin hareketleri ne diğer Host'ları etkiler ne de onları bu bilgiye muhtaç bırakır. Bağlantı yönetimi, merkezi olarak bir merkezi yönetici tarafından işletilir. Bağlantıları kurmak ve ana parçalara ayırmak için, Host'lar arasındaki mesafe, ihtiyaç duyduğumuz tüm hostlara ait genel pozisyon bilgisi için hesaplanmalıdır.

Hareketlilik mimarimizin çekirdek bileşeni her bir "Host" modüldeki genel "ChannelControl modül" ile birlikte bağımsız "MobilityModule"dür (Şekil 3.5). "ChannelControl" ilgili durumların tüm bağlantılarını işletir; "MobilityModules" iki ana göreve sahiptir: Birinci görev Host'un hareketlerini işletmektir. Bu, çeşitli hareketlilik modelleri (Manhattan Mobility ya da Brownian Motion gibi) kullanılarak yapılabilir. İkincisi ise, "MobilityModule" Host'un lokasyon değişimlerini "ChannelControl"a aktarır. Daha sonra "ChannelControl" bu host için olan tüm bağlantıları günceller. Ayrıntılı bilgi Bölüm 3.7.3'de verilmiştir.

3.7.2 ChannelControl

"ChannelControl" modülü, haberleşme mesafesi içinde yer alan Host modüller arasındaki haberleşme kanallarını etkin kılmak ve bu bağlantıları ana parçalara ayırarak bağlanabilirliği hemen serbes bırakmak ile sorumludur. Bağlanabilirlik kaybı harekelilikten ileri gelebilir (örneğin Host'lar bir tarafa doğru çok hızlı hareket edebilirler) ya da aktarım gücünün değişiminden ileri gelebilir ya da Host çökmüş olabilir vb. Doğrudan mesajı önlem olarak, Host modülleri arasındaki bağ kavramı muhafaza edilmelidir, çünki belli haberleşme yolları bilgi için (hata ayıklamada) önemli bir kaynaktır.

OMNeT++ içinde yeralan modüller arasındaki belirgin bağlar, her bir modül için (ve herbir alt modül için de), biri giriş (in) ve değieri çıkış (out) olan, en az iki kapı nesnesi gerektirir. MF için her birime ait bağ kapısının asgari miktarı altı'dır çünki, Nic modül Host modül içerisine gömülmüştür ve kendisi bir "SnrEval", bir "Decider" modül ve bir "Mac" modül gibi alt bölümlere ayrılmıştır (Bkz. Bölüm 3.4.3). En kötü durum senaryolarında (tüm Host'lar doğrudan bağlanmış) bile yeterli kapılara sahip olduğumuzdan emin olmalı ve ağdaki her tekil Host modülü için, herbir Host modül en az iki çift kapı gerektirdiğini hatırlamalıyız. Ağda "n" tane Host olduğunu varsayarak, bir Host için:

$6(n-1)$ kadar kapılar veya $3(n-1)$ kadar kapı çifti tüm ağdaki $6n(n-1) \approx 6n^2$ kadar kapıya öncülük eder.

Daha fazla bellek-verimliliği kapıların dinamik olarak oluşturması ile mümkün olabilir. Ağ başlangıç durumuna getirlirken kapılar açık olarak oluşturulmazlar fakat, talep üzerine dinamik olarak açık oluşturulabilirler. Herbir Host modülü iki listenin güncellemesini yapar: biri giriş-kapıları serbestliği, diğeri ise çıkış-kapıları serbestliği içindir. ChannelControl, iki Host arasındaki bir bağı etkin kılmak istediği zaman ilk olarak heriki Host'daki kapı listesini, serbest kapıların elverişli olup olmadığını bilmek amacı ile, kontrol eder ve ancak serbest bir kapı bulamadığında yeni bir tane oluşturur. Bağ kesmesi (link break) ile ChannelControl bağlantıyı

parçalara ayırır ve yeniden serbest olmuş kapıları uygun kapı listesine ekler. Bu yaklaşım ile, kapıları çok fazla oluşturmak ve silmek için gerekecek ek işlem yükünü azaltarak, bellek gereksinimi düşürülebilir.

Kablosuz ağ simülasyonlarında, sadece gerçek iki Host'un bağlanıp bağlanmadığı (örneğin birbirleri ile konuşabilirler) değil, aynı zamanda gerçek iki Host'un bir birlerine müdahale edip etmedikleri de önemlidir. Başlangıç durumuna getirmede, ChannelControl modülü kanalın taşıyıcı frekansına, mümkün olan azami gönderme gücüne ve diğer yayılıma özgü parametreler gibi genel ağ parametrelerine dayanarak maksimum girişim mesafesini tanımlar. Azami girişim mesafesi, Host için bir komşusunun haberleşmesini karıştırma (müdahale etme) olasılığı olan azami mesafesi ile ilgili ılımlı bir sınırdır, çok daha uzaktaki tüm Host'lar gönderilen mesajı tamamen tanımayacaklardır. Azami girişim mesafesi ne mesajların (doğru olarak) alınabildiği azami mesafeyi belirtir ne de Host'ların kesinlikle alabildiği bazı sinyallerin (eğer yanlızca gürültü ise) aralığını belirtir. Ağdaki basit Host'lar, belirlenmiş maksimum güçden daha az olan, gönderme gücüne sahip olabilirler ve bu sebeple (teorik olarak) bağlandıkları bir Host ile iletişim kuramazlar. Maksimum girişim mesafesi MF'nin ek yük indirgemesi için yanlızca teorik bir yöntemidir.

Azami girişim mesafesine dayanarak, ağın başlangıç durumuna getirilmesi sırasında , ChannelControl tüm Host'lar arasındaki bağlantıları hesaplar ve bir Host'un her hareketinde bağlantıları yeniden günceller. Host'lar arasındaki bağlantıları güncellemek sayısal olarak pahalı bir operasyondur. Bir ağdaki "n" kadar Host'un her çifti arasındaki mesafeyi hesaplamak $O(n^2)$ karmaşıklığına sahiptir.

3.7.3 Hareketlilik modellerinin gerçekleştirilmesi

Yeni hareketlilik modellerinin geliştirilmesi mümkündür, "BasicMobility" sınıfından türetilebilir. BasicMobility "getRandomPosition()" fonksiyonu sağlar. "getRandomPosition()" fonksiyonu rastgele bir başlangıç konumu seçer, konum yok ise omnetpp.ini dosyasının içinden belirlenir. Eğer başlangıç konumunu elde etmek için başka bir yöntem kullanmak istenirse bu fonksiyon yeniden tanımlanabilir.

Ayrıca kullanıcı şunları yapmalıdır: Kendi Host'unu hareket ettirmek istediği zamanlarda kendi kendine bir mesaj göndermeli ve tanımlanmış olması gereken yeni bir konum içinde "handleSelfMsg()" fonksiyonunu yeniden tanımlamalıdır. Konum "Coord"un bir çeşidi olmalıdır (bunun için API referansına bakılabilir). Bu fonksiyonun sonucu olarak, "updatePosition()" fonksiyonu çağırılmalıdır. O hareketli OMNeT++ GUI'sini otomatik olarak günceller ve bundan başka yeni konumu "Blackboard"a yazar. Daha fazla detay için "BasicMobility"nin API kaynağına bakılabilir. Hareketlilik uygulamasına bir örnek olarak ayrıca "protocols/mobility" dizini altındaki "ConstSpeedMobility" modüle bakılabilir.

4. OMNeT++ İLE BENZETİM ÇALIŞMALARI

Bölüm 4 için [19] numaralı kaynakatan yararlanılmıştır.

4.1 İki Düğümlü Ağ Benzetimi

Başlangıç için, iki düğümden oluşan bir ağ oluşturulabilir Bu düğümler basit bir iş yapacaklar: düğümlerden biri bir paket oluşturmalı ve iki düğüm aynı paketi ileri geri aralarında pas etmeliler. Bu düğümlere "mhr" ve "srm" adını verelim. Simülasyon uygulaması için, ilk olarak simülasyonun adını (mhrsrm) taşıyan bir dizin oluşturalım. Daha sonra bir topoloji dosyası oluşturarak ağ'ımızı tanımlayalım. Topoloji dosyası bir metin dosyasıdır ve ağ'daki düğümleri ve onlar arasındaki bağlantıları tanımlar. Herhangi bir metin editörü kullanılarak (örneğin vi, gvim, notepad vb.) bu dosyalar oluşturulabilir. Örnek için topoloji dosyasına "mhrsrm1.ned" adını verelim, dosya aşağıdaki gibi olabilir:

```
// mhrsrm1.ned
// Bu dosya simulasyon icin topoloji yi tanimlar.
simple Txc1
    gates:
        in: in;
        out: out;
endsimple
// Txc1 in iki durumu (mhr ve srm) iki tarafli birbirine baglanmis oldu
// Mhr ve srm mesajlari birbirlerine vericekler
module Mhrsrm1
    submodules:
        mhr: Txc1;
        srm: Txc1;
    connections:
        mhr.out --> delay 100ms --> srm.in;
        mhr.in <-- delay 100ms <-- srm.out;
endmodule
network mhrsrm1 : Mhrsrm1
endnetwork
```

- mhrsrm1 olarak isimlendirdiğimiz bir ağ tanımlamış olduk, bu Mhrsrm1 modül türü için bir örnektir (network ..... endnetwork);
- Mhrsrm1 bir bileşik modüldür, alt modül olan mhr ve srm'nin birleştirilmesi ile meydana gelir. "mhr" ve "srm" , Txc1 olarak isimlendirilen aynı modül;

- türünün örnekleridirler. “mhr”nin çıkış kapısı (out) “srm”nin giriş kapısına (in) bağlanır (module.....endmodule). Heriki taraf (in,out) için 100 ms yayılım gecikme süresi olacaktır;
- Txc1 yalın modül türüdür. Txc1 “in” olarak isimlendirilen bir giriş kapısına ve “out” olarak isimlendirilen bir çıkış kapısına sahiptir (simple....endsimple)

İki düğümlü ağ benzetiminin kaynak kodları Ek-A’da verilmiştir.

Txc1 yalın modül türü, **cSimleModule**’den itibaren alt sınıflara ayrılması gereken, C++ **Txc1** sınıfı tarafından temsil edilir ve **Define_Module()** makrosu ile OMNeT++ içerisine kayıt edilir. Burada **cSimpleModule**’den itibaren iki metod tanımladık: **initialize()** ve **handleMessage()**. Bunlardan birincisi yanlızca bir defa ikincisi ise her nezaman modul’e bir mesaj ulaşırsa başlatılır. “**initialize()**” içinde bir mesaj nesnesi (**cMessage**) oluşturulur ve “**out**” kapısı üzerinden dışarı gönderilir. Bu kapı diğer modülün giriş kapısına bağlı olduğu için, simülasyon programı bu mesajı **handleMessage()** daki değişken vasıtası ile NED dosyası içinden, aradaki bağlantıya atanmış olan 100 ms yayılım gecikmesinden sonra, diğer modüle ulaştırır.

Mesajlar (paketler, çerçeveler, işler vb.) ve olaylar (zamanlayıcılar, zaman aşımları) OMNeT++ içindeki **cMessage** nesnesi (ya da kendi alt sınıfı) tarafından temsil edilirler. Onları gönderdikten ya da zamanladıktan sonra, “zamanlanmış görevler” ya da “gelecek olaylar” listesi içerisinde, kendi zamanları gelene kadar, simülasyon programı tarafından tutulurlar ve handleMessage() yolu ile modüllere ulaştırılırlar.

Simülasyonun inşa edilirken bu işlemi durdurma imkanı yoktur. GUI üzerinden durdurulabilir (ayrıca, konfigürasyon dosyası içerisinde, bir simülasyon süresi limiti ya da CPU süre limiti belirlenebilir). Şimdi derlemede yardımcı olacak bir Makefile oluşturulmalı ve yürütülebilir bir mhrsrm elde etmek için programla bağlantısı kurulmalıdır. Bu komut, içinde çalıştığımız “mhrsrm” dizininde, Makefile dosyası oluşturur.

```
$ opp_makemake
```

Şimdi “make” komutunu kullanılarak simülasyon dosyaları derlenebilir.

```
$ make
```

Eğer yürütmeyi şimdi başlatırsak “omnetpp.ini” dosyasını bulamadığına dair bir uyarı verecektir, bu nedenle yürütmeden önce omnetpp.ini dosyası oluşturulmalıdır. Bu dosya simüle etmek istediğimiz ağ hakkında simülasyon programına bilgi verir (bazı ağlar aynı simülasyon programı içerisinde bulunabilir), parametreler modele göre arttırılmalı, rastlantısal sayı üretecinin başlangıç değeri açıkça belirlenmelidir. Kullanıcı profilinde derleyici ile ilgili kütüphane dosyalarının bulunmasına, tanımlı olmasına, dikkat edilmelidir. Basit bir omnetpp.ini dosyası aşağıdaki gibi olabilir:

```
[General]
network = mhrsrm1
```

mhrsrm2 ve diğer adımlardaki omnetpp.ini dosyası aşağıdakinin aynısı olacaktır:

```
# Bu dosya tum mhrsrm simulasyonu tarafindan paylasilir.
# “#” ile baslayan satirlar açıklama satirlaridir
[Parameters]
mhrsrm4.srm.limit = 5
# exponential()’a yonelik degiskenler ortalamadir;
# truncnormal(),tepesi kesik Gauss dagilimindan
# negatif olmayan degerelere kadar cevap verir
mhrsrm6.mhr.delayTime = exponential(3)
mhrsrm6.srm.delayTime = truncnormal(3,1)
mhrsrm9.n = 5
mhrsrm10.n = 5
mhrsrm11.n = 5
```

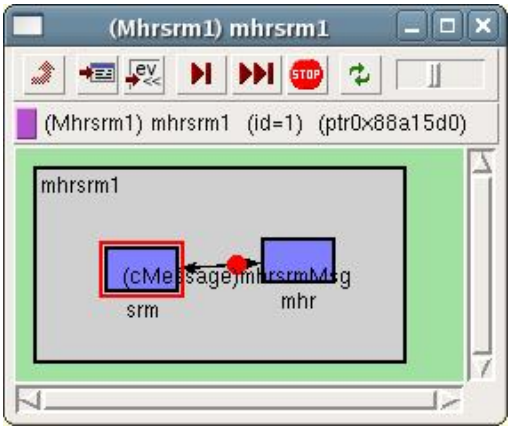
4.1.1 Simülasyonun çalıştırılması

Yukarıdaki adımlar tamamladıktan sonra aşağıdaki komut çalıştırılarak simülasyon başlatılabilir :

```
$ ./mhrsrm
```

Simülasyonu başlatmak için araç çubuğu üzerindeki “run” sembolüne tıklamalı ve “mhr” ve “srm” un bir birlerine mesaj alıp verdikleri görülmelidir. Ana pencere araç çubuğu simülasyon zamanını görüntüler. Bu varsayılan zamandır, güncel zaman değildir. Aslında, gerçek dünya zamanındaki 1 saniye içerisinde, simülasyonun ne kadar OMNeT++ saniyesi olarak çalışacağı donanımımızın özelliğine, simülasyon

modelinin türüne ve karmaşıklığına bağlıdır. Bir düğüm için mesaj işlemek sıfır simülasyon süresi alır. Bu modelde simülasyon süresini aşan tek şey bağlantılar (aradaki hatlar) üzerindeki yayılım gecikmesidir.



Şekil 4.1: Modüller arası haberleşme

Animasyon grafik pencerenin üzerindeki kaydırıcı ile oynayarak yavaşlatılabilir ya da hızlandırılabilir. F8 tuşuna basarak simülasyon durdurulabilir, F4 ile tek adımlı olarak çalıştırılabilir, F5 ile "run" yapılabilir ve F6 ile animasyonsuz olarak çalıştırılabilir. F7 (hızlı mod) maksimum hız için izleme özelliklerini tamamen kapatır. "event/sec" ve "simsec/sec" ölçekleri ana pencerenin durum çubuğu üzerindedir.

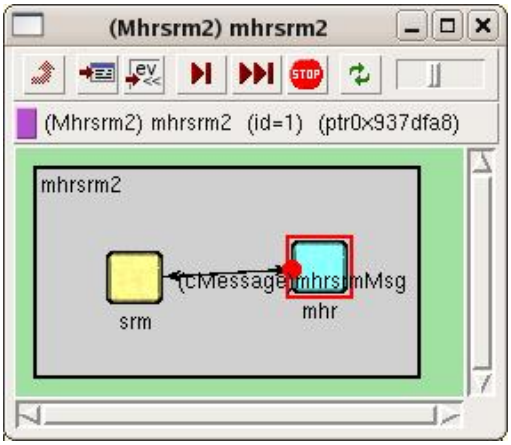
Simülasyon programından File|Exit seçerek ya da "close" ikonunu tıklanmak sureti ile çıkılabilir.

4.2 2-Düğümlü Mhrsrm'yi Geliştirme

4.2.1 2nci adım: Grafik özellik ve hata ayıklama

Burada yeni bir model yapma örneği anlatılmıştır. "proc1" isimli simgeyi (bitmaps/proc1.gif) "mhr "için cam göbeği rengine ve "srm" için de sarı renge boyma işlemi, NED dosyasına "display" komut dizileri eklenmek sureti ile, yapılabilir. "display" satırındaki "i" etiketi, ikonu belirler.

```
submodules:
        mhr: Txc2;
            display: "i=block/process,cyan"; // "i=<icon>,<color>"
        srm: Txc2;
            display: "i=block/process,gold"; // "i=<icon>,<color>"
    connections:
```



Şekil 4.2: Simülasyon için yeni bir model

Aynı zamanda, bazı hata ayıklama mesajlarını "Txc1"e eklemek için, OMNeT++ 'ın "ev" nesnesini aşağıdaki gibi yazmak sureti ile, C++ dosyası değiştirilmelidir:

```
ev << "Baslangic mesaji gonderiliyor\n";
ev << "Alinan mesaj `" << msg->name() << "', yeniden gonderiliyor\n";
```

Ayrıca mhr ve srm için, kendi sembollerinin üzerinde sağ+klik yaparak ve menüden "Module output" seçerek, ayrı pencereler açılabilir. Bu özellik, büyük bir modelimiz var ve belirli bir modülün log mesajları ile ilgileniyorsak faydalı olur.

Grafik özellik ve hata ayıklama mesajı eklemenin kaynak kodları Ek-B'de verilmiştir.

4.2.2 3ncü adım: Durum değişkenleri ekleme

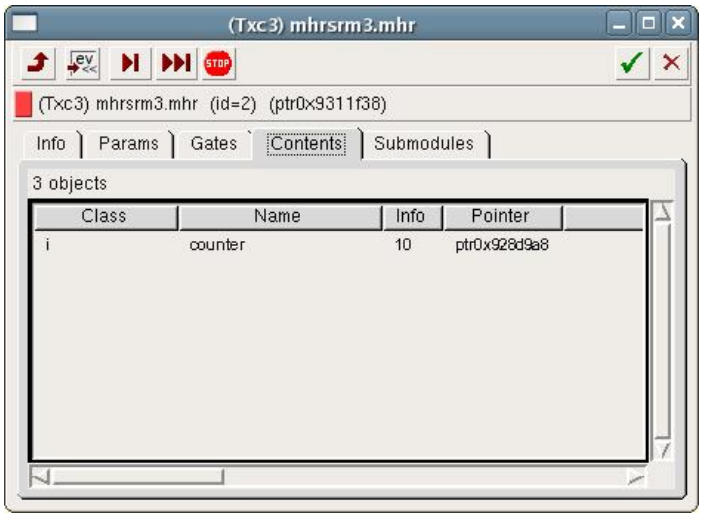
Bu adımda modüle bir sayaç ekleme ve 10 defa değiş-tokuş yapıldıktan sonra mesajı silme işlemi anlatılmıştır. Sayaç, bir sınıf (class) üyesi olarak eklenir:

```
class Txc3 : public cSimpleModule
{
  protected:
    int counter;  // sayac
  public:
```

“initialize()” içindeki değişkene 10 değeri set edilmeli ve o “handleMessage()” içinden azaltılmalıdır, yani herbir mesaj ulaştığında bir azaltılmalıdır. Sıfıra ulaştıktan sonra simülasyon olay dışında çalışacağından işlem sonlanır.

```
WATCH(counter);
```

Yukarıdaki satır sayaç değerinin Tkenv içinden görülebilmesine imkan verir. “mhr” ikonu üzerinde çift-klik yapılır ve sonra denetleyici pencereden Contents sayfası seçilir. Simülasyonun çalıştırılmasına devam edebilir ve sıfır olana kadar sayacın azalması izlenebilir.



Şekil 4.3: Mesaj sayaç değerini görme

Durum değişkenleri eklemenin kaynak kodları Ek-C’de verilmiştir.

4.2.3 4ncü adım: Parametre ekleme

Bu adımda simülasyona parametre ekleme anlatılmıştır: “magic number” 10’lu bir parametreye dönüştürülmüştür. Modül parametreleri NED dosyası içinde tanımlanmalıdır. Veri sayısal, dizgi, boolean ya da XML’den biri biçiminde olabilir.

```
simple Txc4
    parameters:
        limit: numeric const;  // yeni parametre bildiriyor
    gates:
```

Aynı zamanda initialize() içindeki parametreyi okumak ve okunan değeri sayaca atamak için kodlarda değişiklik yapılmalıdır.

```
    counter = par("limit");
```

Şimdi, parametreler NED dosyası içinden ya da omnetpp.ini dosyasından atanabilir. Bir çok parametrenin atamasını omnetpp.ini dosyası kulanılarak yapmak daha uygundur, çünkü bu yöntem bir modeli daha fazla esnek yapmaktadır. Burada NED dosyası içinden bir parametre ataması aşağıdaki şekilde yapılabilir:

```
// Module parametreler ekleme
module Mhrsrm4
    submodules:
        mhr: Txc4;
            parameters:
                limit = 8;  // mhr'nin limit'i 8
            display: "i=block/process,cyan";
        srm: Txc4;
            // dikkat edin; srm'nin limit'ini sinirlandirmiyoruz
            display: "i=block/process,gold";
    connections:
```

ve diğeri omnetpp.ini içinde:

```
mhrsrm4.srm.limit = 5
```

"omnetpp.ini" özel sembolleri desteklemektedir ve NED dosyasından atanmış olan parametreler, omnetpp.ini içindekilerden birine göre öncelik alırlar, bu nedenle aşağıdaki gibi kullanılır:

```
mhrsrm4.t*c.limit=5
```

ya da:

```
mhrsrm4.*.limit=5
```

hatta:

```
**.limit=5
```

aynı etkiyi yapacaklardır (* ve ** arasındaki fark, * bir nokta işaretini yakalayamaz ve ** ise yakalayabilir). Tkenv içinde, ana pencerenin sol köşesi üzerindeki nesne ağacından ya da modül denetleyicinin Parameters sayfasından (modül sembolü üzerinde çift-klik yaparak) herbir modül parametresini yoklamak mümkün. Daha küçük limit'i olan modül mesajı siler ve o sebeple simülasyonu sona erdirir.

Parametre eklemenin kaynak kodları Ek-D'de verilmiştir.

4.2.4 5nci adım: İşlem gecikmesini modelleme

Önceki modellerde, mhr ve srm aldıkları mesajı hemen geri gönderiyorlardı. Burada zaman ayarı eklemesi anlatılmıştır: mhr ve srm mesajı geri göndermeden önce onu 1 simülasyon saniyesi kadar süre tutarlar. OMNeT++ deki bunun gibi zaman ayarlarını modülün mesajı kendi kendisine göndermesi ile gerçekleştirilebilir. Bu gibi mesajlar öz-mesajlar olarak bilinir (fakat yanlızca kendilerinin kullandığı yöntemden dolayı, yoksa onlarda alışılmış mesaj nesneleridirler). "event" ve "mhrsrmMsg" sınıflarına, zamanlama için kullanılan mesajı ve simüle etmek istenilen gecikme işleminin mesajını tanıması için, iki "cMessage *" değişkeni eklenebilir.

```
class Txc5 : public cSimpleModule
{
  protected:
    cMessage *event; //zamanlamada kullanilan event nesnesi icin isaretci
    cMessage *mhrsrmMsg; //mesajı biz onu geri gonderene kadar hatirlamayi saglayan
degisken
  public:
```

"scheduleAt() fonksiyonu ile öz-mesajlar (SelfMessage) gönderilebilir; modül'e ne zaman geri teslim edileceği burada belirtilmelidir.

```
        scheduleAt(simTime()+1.0, event);
```

Yeni bir mesajın giriş kapısı üzerinden ulaşıp ulaşmadığı ya da öz-mesajın geri gelip gelmediği (zamanlamanın sona erdiğini), "handleMessage()" fonksiyonu içinden, kontrol edilmelidir. Bu noktada aşağıdaki ifade kullanılabilir:

```
if (msg==event)
```

ya da:

```
if (msg->isSelfMessage())
```

Kodu küçültmek için sayaç bırakılabilir. İşlem gecikmesini modellemenin kaynak kodları Ek-E'de verilmiştir.

4.2.5 6ncı adım: Rastlantısal sayılar ve parametreler

Gecikme 1 saniyeden rastgele bir değere değiştirilebilir, bu NED dosyası ya da omnetpp.ini ile set edilerek yapılabilir. Modül parametreleri rastgele değişkenler geri döndürebilirler; bununla beraber bu özellikten faydalanabilmek için, handleMessage() fonksiyonuna, kullanılan her sefer için, parametre okutulmalıdır:

```
double delay = par("delayTime");
ev << "Mesaj ulasti, bekleme suresi basliyor " << delay << " secs...\n";
mhrsrmMsg = msg;
scheduleAt(simTime()+delay, event);
```

İlaveten küçük bir olasılık ile paketi silenir:

```
if (uniform(0,1) < 0.1)
{
    ev << "\"Mesaj\" Siliniyor\n";
    delete msg;
}
```

"omnetpp.ini" içinden parametreler atanır:

```
mhrsrm6.mhr.delayTime = exponential(3)
mhrsrm6.srm.delayTime = truncnormal(3,1)
```

Simülasyonun kaç kere tekrar çalıştırılabileceği (yada onu Simulate|Rebuild network menü öğesi ile yeniden başlatılabileceği) test edilebilir, bu yapıldığında netice olarak aynı sonuç elde edilir. Bunun sebebi OMNeT++ rastalantısal sayılar üretmek için belirleyici bir algoritma kullanır ve onu aynı çekirdeğe göre başlangıç durumuna getirir. Bu tekrarlanabilir simülasyonlar için önemlidir.

Aşağıdaki satır omnetpp.ini dosyasına eklenerek farklı başlangıç değerleri için testler yapılabilir:

```
[General]
seed-0-mt=532569  # ya da herhangi 32-bitlik deger
```

Rastlantısal sayılar ve parametreler için kaynak kodlar Ek-F'de verilmiştir.

4.2.6 7nci adım: Zaman aşımı ve zamanlayıcıları iptal etme

Ağ protokollerini modellemede, gerçeğe daha yakın bir adım elde etmek amacı ile, simülasyon durdur-ve-beklet şeklinde çalıştırılabilir. Bunun için mhr ve srm farklı sınıflara ayırılmalıdır. Temel senaryo öncekine benzerdir: mhr ve srm birbirlerine bir mesaj verirler. Bununla beraber, srm sıfırdan farklı bir olasılık ile mesajı kaçırabilir ve o taktirde mhr mesajı yeniden göndermek zorunda kalır. "srm"nin kodu aşağıdaki gibi olmalıdır:

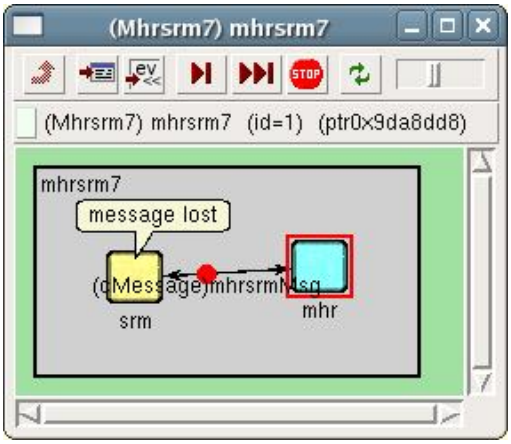
```
void Srm7::handleMessage(cMessage *msg)
{
    if (uniform(0,1) < 0.1)
    {
        ev << "\"Mesaj \" Siliniyor.\n";
        bubble("mesaj kaybedildi");
        delete msg;
    }
    else
```

"srm" her nezaman mesaj düşürürse buble() fonksiyonu sayesinde yeni bir mesaj yayınlar. "mhr" her nezaman mesaj gönderse bir zamanlayıcı çalıştırır. Zamanlayıcı'nın süresi bittiği zaman, mesajın kaybolduğu ve başka bir tane gönderildiği varsayılır. Eğer srm ulaşanları cevaplar ise, zamanlayıcı iptal edilir. Zamanlayıcı öz-mesaj'dır.

```
        scheduleAt(simTime()+timeout, timeoutEvent);
```

Zamanlayıcıyı iptal etme cancelEvent() fonksiyonu çağırılarak yapılır. Bu durum aynı zaman aşımı mesajının sık sık kullanılmasına engel değildir.

```
        cancelEvent(timeoutEvent);
```



Şekil 4.4: buble() ile mesaj görüntüleme

Zaman aşımı ve zamanlayıcıları iptal etmenin kaynak kodları Ek-G'de verilmiştir.

4.2.7 8nci adım: Aynı mesajı yeniden iletmek

Bu adımda önceki model küçültülecektir. Eğer yeniden iletme ihtiyacı duyulursa başka bir paket oluşturulabilir. Orjinal paketin bir kopyasını saklamak çok fazla elverişli değildir, tekrar inşa etmeye gereksinim duymadan yeniden gönderilebilir. Orjinal paketi saklamak ve yanlızca onun kopyasını göndermek için ne yapmalıyız? "srm"nin alındı bilgisi ulaştığı zaman orjinal olanı silinebilir. Model doğrulamasını kolaylaştırmak için, mesaj isimlerine, mesaj sıra numarası eklenebilir. Aşırı büyüyen "handleMessage()"ı feshetmek için, uygun kodu yeni iki fonksiyon içerisine koymalıyız: generateNewMessage() ve sendCopyOf() ve bunlar handleMessage() tarafından çağırılırlar.

```
cMessage *Mhr8::generateNewMessage(){
    // bir mesaji her zaman farklı bir isim ile uretir.
    char msgname[20];
    sprintf(msgname, "mhr-%d", ++seq);
    cMessage *msg = new cMessage(msgname);
    return msg;
}
void Mhr8::sendCopyOf(cMessage *msg) {
    // mesajin ikinci kopyasini olustur ve kopyasini gonder
    cMessage *copy = (cMessage *) msg->dup();
    send(copy, "out");
}
```

Aynı mesajı yeniden iletmenin kaynak kodları Ek-H'de verilmiştir.

4.3 Gerçek Ağ Yapısına Dönüşüm

4.3.1 9ncu adım: İkiden fazla düğüm

Bu bölümde, birden fazla mhr modülünün oluşturulması ve bir ağ içinde birbirine bağlanması, anlatılmıştır. Düğümlerden biri bir mesaj üretir ve diğerleri onu, önceden belirlenmiş düğüme ulaşıncaya kadar, rastgele yönlere doğru çevreye fırlatır. NED dosyasında bir kaç değişiklik yapmamız gerekebilir. Herşeyden önce, Txc modülünün birçok giriş ve çıkış kapısına sahip olması gerekir:

```
simple Txc9
    gates:
        in: in[];  // in[] ve out[] vektor kapilari olmalari ilan ediliyor
        out: out[];
endsimple
```

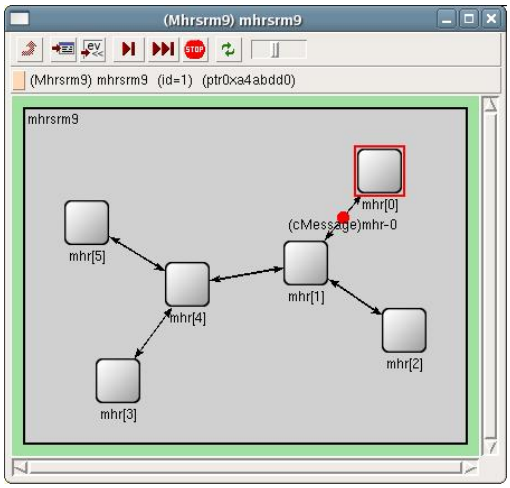
"[ ]" kapıları kapı vektörlerine dönüştürür. Vektörün büyüklüğü (kapıların sayısı), ağ inşa etmek amacı ile, Txc nerede kullanılıyorsa orada tanımlanır:

```
module Mhrsrm9
    submodules:
        mhr: Txc9[6];   // 6 Txc modulumuz olacak
            display: "i=block/process";
    connections:
        mhr[0].out++ --> delay 100ms --> mhr[1].in++;
        mhr[0].in++ <-- delay 100ms <-- mhr[1].out++;
        mhr[1].out++ --> delay 100ms --> mhr[2].in++;
        mhr[1].in++ <-- delay 100ms <-- mhr[2].out++;
        mhr[1].out++ --> delay 100ms --> mhr[4].in++;
        mhr[1].in++ <-- delay 100ms <-- mhr[4].out++;
        mhr[3].out++ --> delay 100ms --> mhr[4].in++;
        mhr[3].in++ <-- delay 100ms <-- mhr[4].out++;
        mhr[4].out++ --> delay 100ms --> mhr[5].in++;
        mhr[4].in++ <-- delay 100ms <-- mhr[5].out++;
endmodule
```

Burada 6 vektör modül oluşturulmuş ve birbilerine bağlanmışlardır. mhr[0] çevreye gondermek için mesaj üretir. Bu işlem initialize() içinde index() fonksiyonunun yardımı ile yapılır; index() fonksiyonu vektör içindeki modülün index bilgisini geri döndürür. Kodun özü, bir mesajın düğüme ulaştığı herhangi bir zamanda handleMessage() aracılığı ile çağırdığımız, forwardMessage() fonksiyonudur. Rastlantısal bir kapı numarası (size() kapı vektörünün büyüklüğüdür) çeker ve mesajı bu kapı üzerinden gönderir:

```
void Txc9::forwardMessage(cMessage *msg) {
    // Bu ornekte, mesaji uzerinden gondermek icin yanlizca
    // rastgele bir kapi seciyoruz
    // 0 ve out[] kapisinin buyuklugu arasinda rastgele bir sayi seciyoruz.
    int n = gate("out")->size();
    int k = intuniform(0,n-1);
    ev << "Mesaj sevk ediliyor " << msg << " cikis portu uzerinde[" << k << "]\n";
    send(msg, "out", k);
}
```

Mesaj mhr[3]'e ulaştığı zaman, onun handleMessage() fonksiyonu mesajı siler. Bu basit yönlendirme çok verimli değildir: paket farklı bir yöne gönderilmeden önce, iki düğüm arasında kısa bir süre sıçrama yapar. Paket kendisini ilk gönderene geri iletilmeyecek ise, bu durumu geliştirmek daha kolay olur. Bunu gerçekleştirmek için: cMessage::arrivalGate(), cGate::index(). Eğer mesaj, bir self-message (öz-mesaj) olmasına ragmen, bir kapı üzerinden gelmiyorsa o zaman arrivalGate() NULL geri döndürür.



Şekil 4.5: İkiden fazla düğüm için simülasyon topolojisi

İkiden fazla düğüm ile benzetim yapmanın kaynak kodları Ek-I'de verilmiştir.

4.3.2 10ncu adım: Kendi mesaj sınıfımızı tanımlama

Bu adımda hedef adresi mhr[3] içine doğrudan yazılmayacak, rastgele bir hedef seçilecek ve hedefin adresi mesajın içine eklenecektir. Eniyi yöntem cMessage altsınıfına yöneliktir ve hedefi bir veri üyesi olarak eklemektir. Mesaj el ile yazılmak istenmez ise OMNeT++ tarafından oluşturulması sağlanabilir. Mesaj sınıf özelliği mhrsrm10.msg dosyası içinde yer almalıdır:

```
message MhrsrmMsg10 {
    fields:
        int source;
        int destination;
        int hopCount = 0;
}
```

Makefile, mesaj derleyici için bir kurulum dosyasıdır, opp_msgc, mhrsrm10_m.h ve mhrsrm10_m.cc dosyalarını üretmesi için, mesajın ilanından itibaren çağrılır. Bunlar cMessage'den itibaren alt sınıflara ayrılmış, üretilmiş bir MhrsrmMsg10 sınıfı içerirler; bu sınıf her alan için çeşitli yöntemlere sahip olur. C++ kodu içine mhrsrm10_m.h eklenebilir ve diğer sınıfların yerine MhrsrmMsg10 kullanılabilir.

```
#include "mhrsrm10_m.h"
```

Örneğin, mesaj oluşturmak ve onun sahalarını doldurmak için, generateMessage() fonksiyonu içinde, aşağıdaki satır kullanılabilir:

```
    MhrsrmMsg10 *msg = new MhrsrmMsg10(msgname);
    msg->setSource(src);
    msg->setDestination(dest);
    return msg;
```

Daha sonra, handleMessage() aşağıdaki gibi başlayabilir:

```
void Txc10::handleMessage(cMessage *msg) {
    MhrsrmMsg10 *ttmsg = check_and_cast<MhrsrmMsg10 *>(msg);

    if (ttmsg->getDestination()==index())
```

handleMessage()'e yönelik değişkende, mesajı bir cMessage * işaretçisi olarak elde ediyoruz. Bununla beraber, eğer msg'yi MhrsrmMsg10 *'a doğru fırlatırsak, yanlızca MhrsrmMsg10'da tanımlı alanlara erişilebilir. Sade C-stil fırlatış ((MhrsrmMsg10 *)msg) emniyetli değildir çünkü, eğer mesaj bir MhrsrmMsg10 değilse programın çökmesine ve araştırması zor olan bir hataya sebep olacaktır. C++ dynamic_cast olarak bilinen bir çözüm sunmaktadır.

Burada OMNeT++ tarafından sağlanan check_and_cast<>() fonksiyonu kullanılmış: dynamic_cast yardımı ile işaretçi fırlatmayı deniyor ve eğer hata verirse simülasyon bir hata mesaji ile duruyor: Modelimizi daha uzun çalışır yapmak için, mesaj

hedefine ulaştıktan sonra, hedef düğüm rastlantısal adresi olan başka bir mesaj üretmelidir. Mesaj üzerinde çift-klik yaparak denetleyici penceresi açılabilir. (Bunun için program geçici olarak durdurulmalıdır). Denetleyici pencere bir çok faydalı bilgiyi görüntüleyecektir; mesaj alanları Contents sayfası üzerinde görülebilir. Bu modelde, yanlızca, herhangi bir zamana ait bir mesaj vardır: düğümler, kendilerine mesaj ulaştığında yanlızca bir mesaj üretiler. Modül sınıfı değiştirilerek periyodik mesaj üretmesi sağlanabilir. Mesajlar arasındaki aralık, dağıtık rastlantısal sayılar geri dönen, bir modül parametresi olmalıdır.

Kendi mesaj sınıfımızı tanımlamanın kaynak kodları Ek-İ'de verilmiştir.

4.4 İstatistik Elde Etme

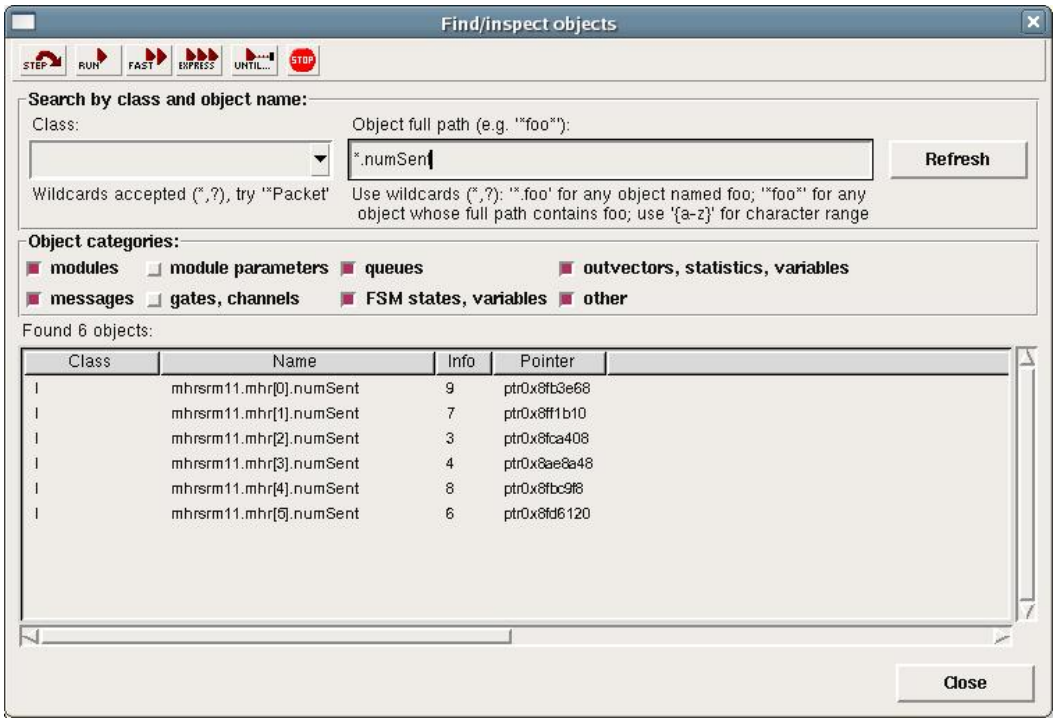
4.4.1 11nci adım: Gönderilen/Alınan paketlerin sayısını görüntüleme

Çalışma süresince herbir düğüm kaç mesaj gönderdi ve aldı konusunda bir görüş elde edebilmek için, modul sınıfına iki sayac eklenmiştir: numSent ve numReceived.

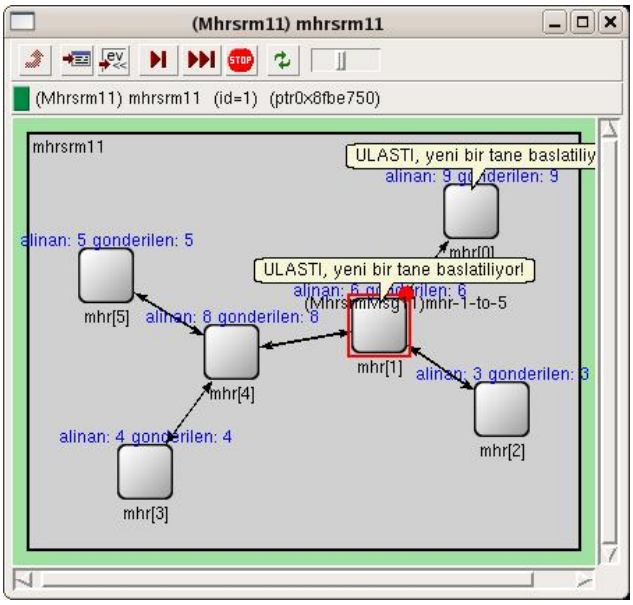
```
class Txc11 : public cSimpleModule {
  protected:
    long numSent;
    long numReceived;

  public:
```

Sıfır değerine set edilirler ve initialize() yöntemi içinde gözlenirler (WATCH). Şimdi "Find/inspect objects" diyalog penceresini kullanarak muhtelif düğümlerden kaç paket gönderilmiş veya alınmış öğrenilebilir. Bu simülasyon modelindeki sayılar aşağı yukarı aynı olacaktır, netice intuniform() aracılığı ile öğrenilebilir. Yukarıdaki modül sembolleri üzerinde bazı ayarlanabilir bilgiler görülmektedir: t=metin (text) belirten dizgi biçiminin gösterimidir; sadece çalışma süresince dizgi gösterimlerinin değişimine ihtiyaç duyulur. Aşağıdaki kod bu işi yapmaktadır:

```
        if (ev.isGUI())
            updateDisplay();
void Txc11::updateDisplay(){
    char buf[40];
    sprintf(buf, "alinan: %ld gonderilen: %ld", numReceived, numSent);
    displayString().setTagArg("t",0,buf);
}
```



Şekil 4.6: Find/inspect objects diyalog penceresi



Şekil 4.7: 11nci adım simülasyon çalışması

Paketlerin sayısını görüntülemenin kaynak kodları Ek-J'de verilmiştir.

4.4.2 12nci adım: İstatistik derlemelerini dahil etme

Bu bölümde istatistik toplama anlatılmıştır. Örneğin, bir mesaj hedefine ulaşırken gezerek gitmektedir ve gezme esnasında yaptığı sekme miktarının ortalamasını bilmek önemli olabilir. Her mesajın bir çıkış vektörü (çiftlerin (zaman, değer) dizisi, zaman dizisinin sırası) içerisine varışı hususunda sekme sayısı kayıt edilebilir. Aynı zamanda ortalama, standart sapma, her bir düğümün minimum ve maksimum değerleri hesaplanarak simülasyon sonunda bir dosya içerisine yazılabilir. Daha sonra çıktıları analiz etmek için "off-line" araçlar kullanılabilir. Bu maksatla, Txc12 sınıfı içine "çıktı vektörü nesnesi" (cOutVector) (omnetpp.vec'e kayıt edecek) ve bir histogram nesnesi (cLongHistogram) (aynı zamanda ortalama vb. hesaplar) eklenmelidir:

```
class Txc12 : public cSimpleModule
{
  protected:
    long numSent;
    long numReceived;
    cLongHistogram hopCountStats;
    cOutVector hopCountVector;

  public:
```

Bir mesaj hedef düğüme ulaştığı zaman, bu istatistiki bilgi kayıt edilmelidir. Bu maksatla aşağıdaki kod handleMessage()'a ilave edilir:

```
hopCountVector.record(hopcount);
hopCountStats.collect(hopcount);
```

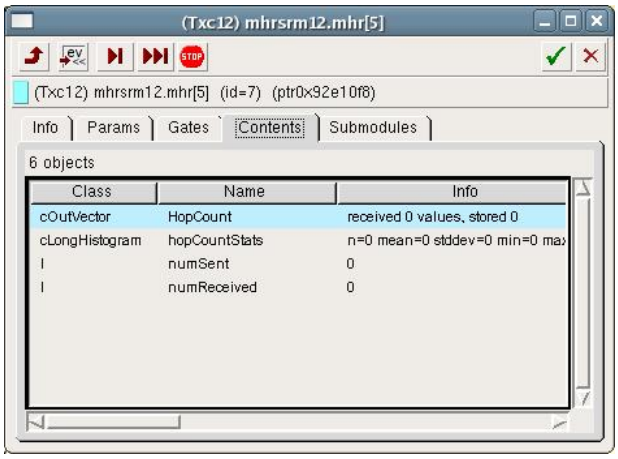
hopCountVector.record() veriyi omnetpp.vec dosyasına yazdırır. Simülasyon modeli ya da uzun yürütme süresi nedeni ile, omnetpp.vec dosyasının büyüklüğü artabilir. Bu tür simülasyonları idare etmek için, omnetpp.ini dosyası içinde belirli bir biçimde "vector enable/disable" yapılabilir ve ayrıca bir simülasyon süresi belirlenebilir.

Yeni bir simülasyona başlandığı zaman varolan omnetpp.vec dosyası silinir. Simülasyon boyunca histogram tarafından toplanmış sayısal veri, finish() fonksiyonu içinde manuel olarak kayıt edilmek zorundadır, finish() fonlksiyonu simülasyonun başarılı bir şekilde tamamlanmasını başlatır, bir hata ile sonlanmış ise başlatmaz.

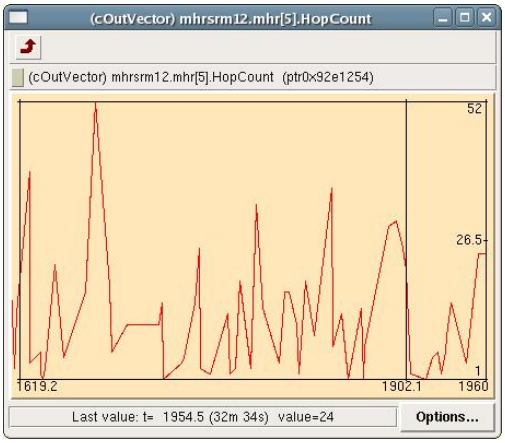
```
void Txc12::finish()
{
    // bu fonksiyon simulasyon sonunda OMNeT++ atarafindan cagirilir.
    ev << "Gonderilen:     " << numSent << endl;
    ev << "Alinan: " << numReceived << endl;
    ev << "Minimum sekme sayisi:    " << hopCountStats.min() << endl;
    ev << "Maksimum sekme sayisi:    " << hopCountStats.max() << endl;
    ev << "Ortalama sekme sayisi:   " << hopCountStats.mean() << endl;
    ev << "Sekme sayisi standart sapmasi: " << hopCountStats.stddev() << endl;
    recordScalar("#sent", numSent);
    recordScalar("#received", numReceived);
    hopCountStats.recordScalar("hop count");

}
```

Omnetpp.vec den farklı olarak, omnetpp.sca dosyası simülasyonların çalıştırılmaları aralığında silinmez. Bunun yerine, yeni veriler dosya içine ilave edilir. Burada amaç, farklı simülasyonların çalışmasından (örneğin farklı giriş parametreleri ile) elde ettiğimiz çıktıları biriktirmek ve onları bir arada analiz etmektir. Farklı dosya isimleri (omnetpp.ini seçeneği) kullanmak mümkündür fakat bu birçok simülasyon sonuçları için her birinin farklı dosyalara yazılması demek olur.

Simülasyon boyunca veriyi görüntülemek mümkündür. Modül denetleyicisinin (inspector) "Contents" sayfasında hopCountStats ve hopCountVector nesneleri bulunmaktadır, bunların denetleyicileri, üzerlerinde çift-klik yapılarak, açılabilir. Başlangıçta boş olmaları gerekir, yeterli veriyi elde etmek için simülasyon "fast" (ya da Express) modda çalıştırılabilir. Akabinde aşağıdaki gibi sonuçlar elde edilebilir:



Şekil 4.8: Modül denetleyicisi içeriği



Şekil 4.9: Çizgisel sıçrama grafiği

Yeterli verinin toplandığı düşünüldüğünde simülasyon durdurulabilir ve sonra "off-line" durumda sonuç dosyaları (omnetpp.vec ve omnetpp.sca) analiz edilebilir. Çıkmadan önce menü üzerinden "Simulate|Call finish()" seçilir, bu finih() fonksiyonunun çalışmasını ve verinin omnetpp.sca dosyasına yazılmasını sağlar.

İstatistik derlemelerini dahil etmenin kaynak kodları Ek-K'da verilmiştir.

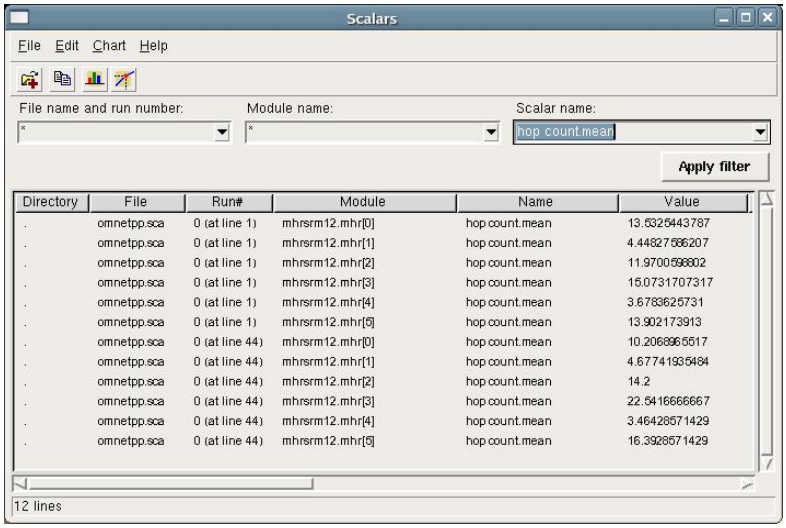
4.5 Plove ve Scalars ile Sonuç Analizi

4.5.1 Scalar istatistikleri

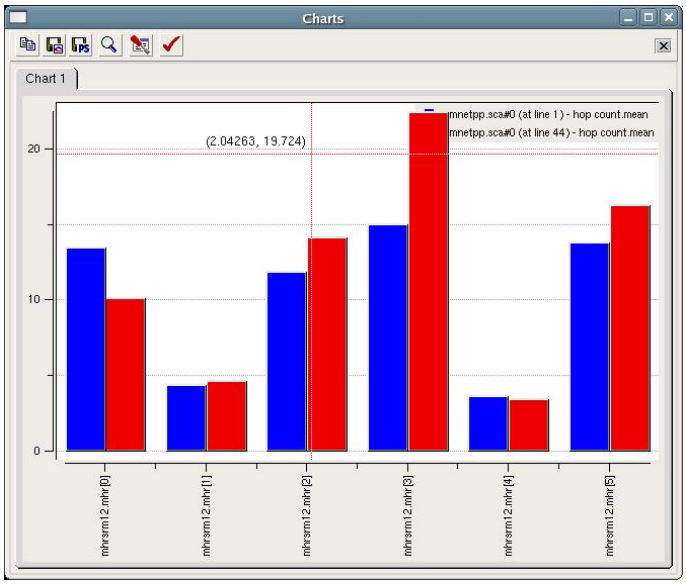
Scalars aracı omnet.sca dosyasının içeriğini şekilsel olarak canlandırmak amacı ile kullanılabilir. Çubuk grafik, x-y noktalaması (örneğin üretilen iş vs.nin verdiği yük) çizebilir ya da veriyi geçici veri alanı (clipboard) aracılığı ile, daha detaylı analizler için, tablolar ya da diğer programlar içerisine aktarabilir.

```
$ scalars omnetpp.sca
```

Program veriyi bir tablo içerisinde görüntüler ve dosya adı, çalıştırma numarası, kayıt edilmiş modül adı ve değeri kolonlara ayrılmış olarak gösterilir. Genellikle gözden geçirilecek bir çok satır bulunur, bundan dolayı en üstte yer alan üç "combo box" vasıtası ile bir seçim yaparak (ya da düzenleme yaparak) filtreleme yapılmalıdır. Filtreleme işlemi * veya ** özel sembolleri kullanılarak da yapılabilir.

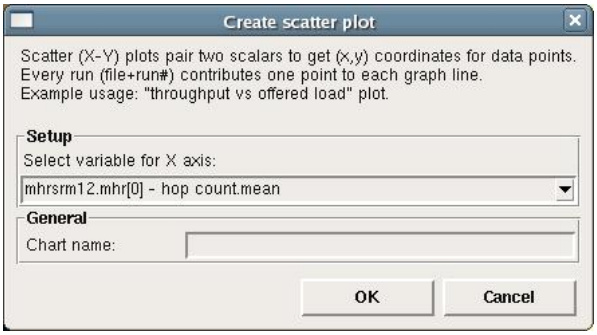


Şekil 4.10: Scalars ana penceresi

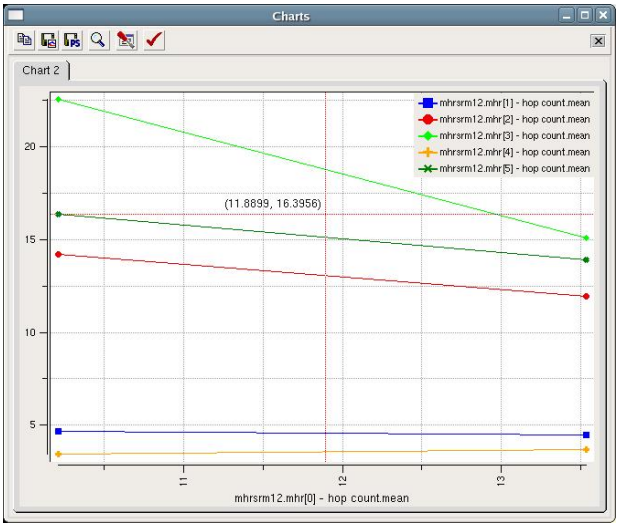


Şekil 4.11: Scalars grafiği

Yeni scalar dosyaları da bu pencere içerisine yüklenebilir ve onların analizi de yapılabilir. Seçili satırları “Edit|Copy” ya da uygun araç çubuğu simgesi aracılığı ile geçici veri alanına (clipboard) kopyalayabilir ve örneğin OpenOffice Hesap Makinesi, MS Excel ya da Gnumeric içine yapıştırılabilir. Çubuk grafiği düğmesi yeni bir pencerede bir çubuk grafik oluşturur. Grafik üzerinde çift klik yaparak ve içerik menüsünden tercih yapılarak özelleştirilebilir. Aynı zamanda EPS, GIF ya da bir metafile formatına çevirilebilir.



Şekil 4.12: Scatter plot oluşturma penceresi



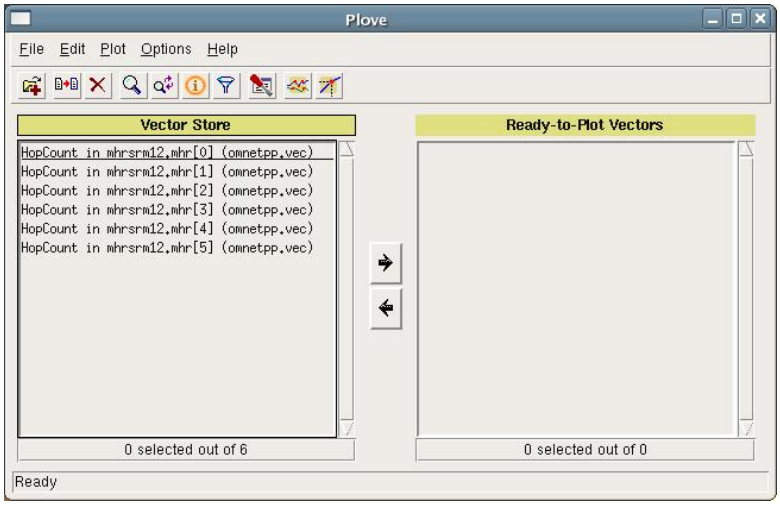
Şekil 4.13: Plot grafiği

4.5.2 Çıkış vektörlerini plotlama

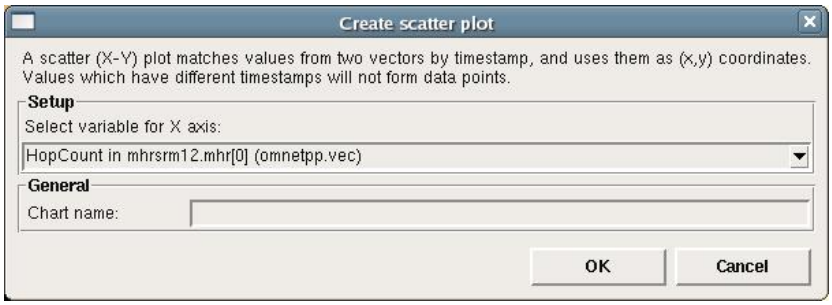
Çıkış vektör dosyaları Plove kullanılarak şekile dönüştürülebilir:

```
$ plove omnetpp.vec
```

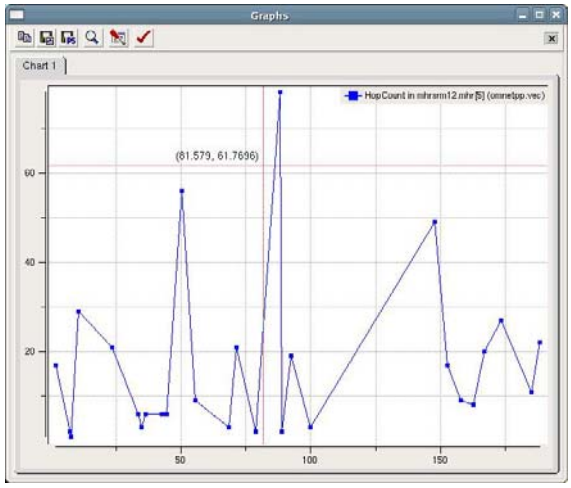
Plove penceresindeki sol bölüm omnetpp.vec dosyası içinde mevcut vektörleri gösterir. Plotlama yapmak için, bazı vektörler bölüme kopyalanmalıdır, birinin ya da birden fazlasının seçilmesi ve araç çubuğu üzerindeki “Plot” sembolünün klik’lenmesi yeterli olacaktır.



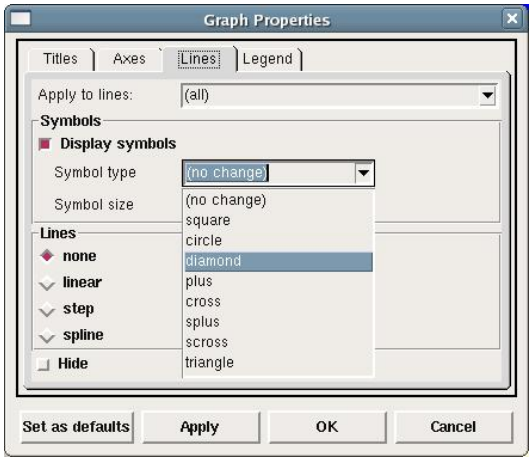
Şekil 4.14: Plove (Vector) ana penceresi



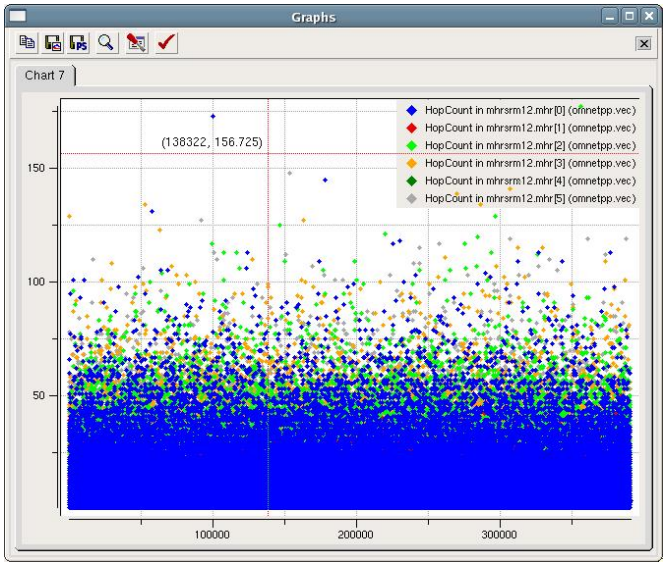
Şekil 4.15: Plove plot oluşturma penceresi



Şekil 4.16: Plove çizgisel grafiği

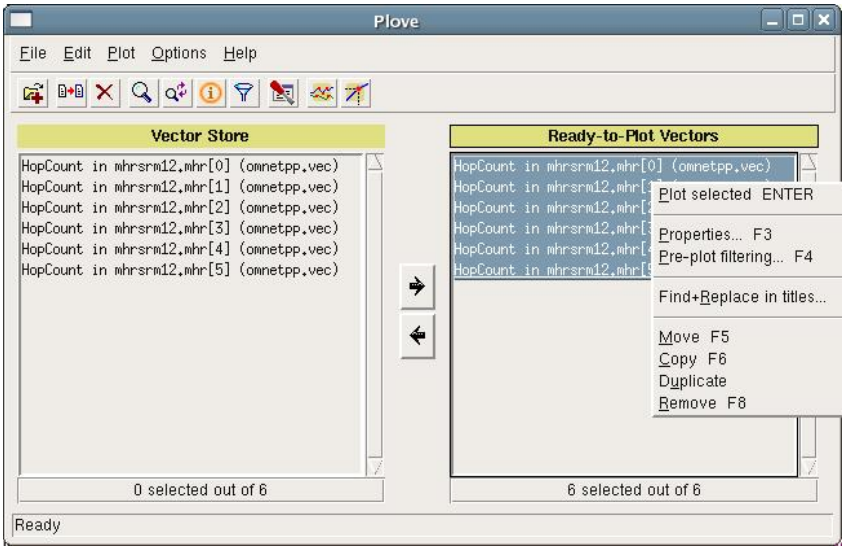


Şekil 4.17: Plove grafik çıktısını değiştirme

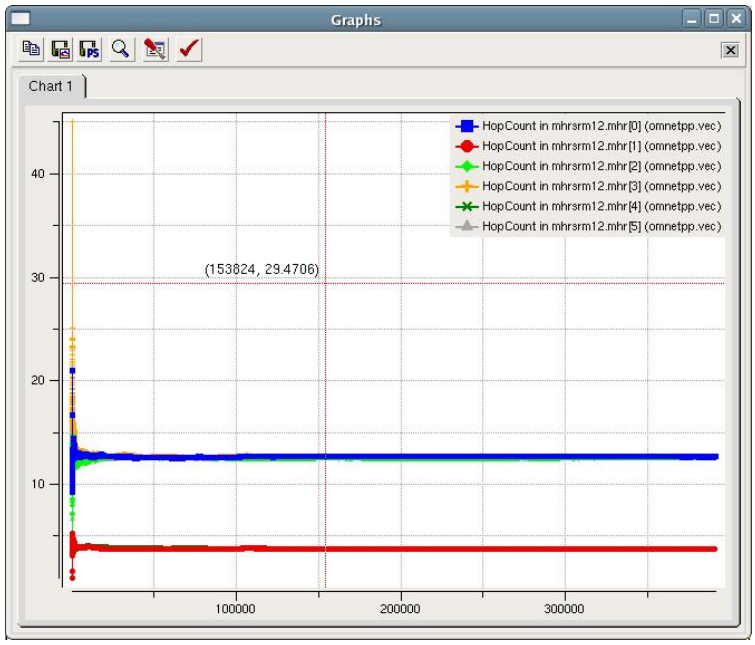


Şekil 4.18: Plove noktasal grafik

[0,t) ye doğru ortalama değeri çizdiren bir filtre uygulanabilir. Ana pencerede, vektörler üzerinde sağ-klik yaparak içerik menüsünden “Pre-plot filtering” seçerek bunu yapmak mümkün. İlkin diyalog penceresinde, “filter” açılır-listesinden “mean” seçilmeli ve OK tuşlanmalı. Sonraki işlem olarak araç çubuğu üzerindeki Plot sembolü tuşlanabilir.



Şekil 4.19: Plove filtreleme penceresi



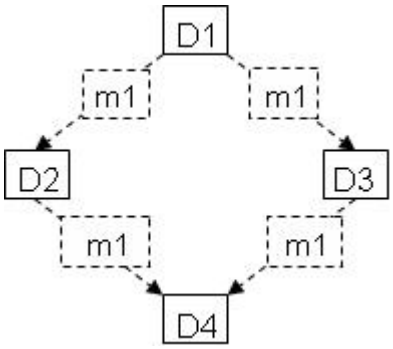
Şekil 4.20: Plove filtre edilmiş grafik

4.6 Algılayıcı Ağlar için İletişim Kuralları ve OMNeT++ Benzetimleri

4.6.1 Taşma iletişim kuralı

Bir ağ içerisinde toplanmış algılayıcılar, bireysel olarak hata yapmaları durumunda bile fonksiyonlarını doğru olarak sürdürebilirler; örneğin, eğer ağdaki bir algılayıcı

çok önmeli bir bilginin bir parçasını kaybederse, diğer algılayıcılar hatalı veriyi sağlayarak kurtarabilirler. Algılayıcı düğümleri birleştiren kablosuz bağlantıların bant genişliği sınırlıdır, bu algılayıcı haberleşmenin kısıtlanmasına neden olur.



Şekil 4.21: İçe doğru göçme problemi

Şekil 4.21'de D1 düğümü verisini tüm komşularına taşma yaparak gönderir. İşlem sonunda verinin iki kopyası D4'e ulaşır. Sistem gereksiz alma ve göndermeler ile enerji ve bant genişliği israfına neden olur. Bu basit yaklaşım algılayıcı ağları için bir iletişim kuralıdır fakat üç eksiği onun yetersiz kalmasına neden olur.

4.6.1.1 İçe doğru göçme

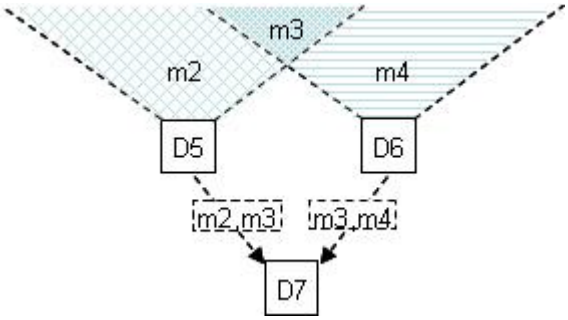
Klasik taşma yönteminde, bir düğüm yapıp yapamayacağına bakmaksızın ya da veriyi başka kaynaktan alıp almadığına bakmaksızın daima veriyi kendi komşularına gönderir. Bu iç patlama problemine yol açar, şekil 4.21'de gösterilmiştir. Burada, D1 düğümü veriyi kendi iki komşusuna, D2 ve D3, veriyi taşma şeklinde gönderir. Bu düğümler D1 düğümünden gelen veriyi depolar ve bir kopyasını kendi komşuları olan D4'e aynı şekilde gönderirler. Protokol aynı verinin iki kopyasını D4'e göndererek kaynak israfına neden olur.

4.6.1.2 Örtüşme

Algılayıcı düğümleri genellikle coğrafik olarak örtüşen alanları kapsarlar ve düğümler genellikle algılayıcı verisinin örtüşen parçalarını da toplarlar. Şekil 4.22'de, iki düğümün (D5, D6) örtüşen (m3 bölgesindeki) veriyi nasıl topladıkları ve veriyi

ortak komşularına (D7) nasıl taşma şeklinde gönderdikleri gösterilmiştir. İlaveten, uygulanan algoritma, verinin bir parçsının iki kopyasını aynı düğüme göndererek, enerji ve bant genişliği israfına neden olur. Örtüşme, iç patlama problemine göre çözümü daha zor bir problemdir. İçe doğru patlama, yanlızca ağ topolojisinin bir fonksiyonu iken, örtüşme hem topolojinin hem de örtüşen verinin algılayıcı düğümler ile eşleşmesinin bir fonksiyonudur.

4.6.1.3 Kaynak körlüğü



Şekil 4.22: Örtüşme problemi

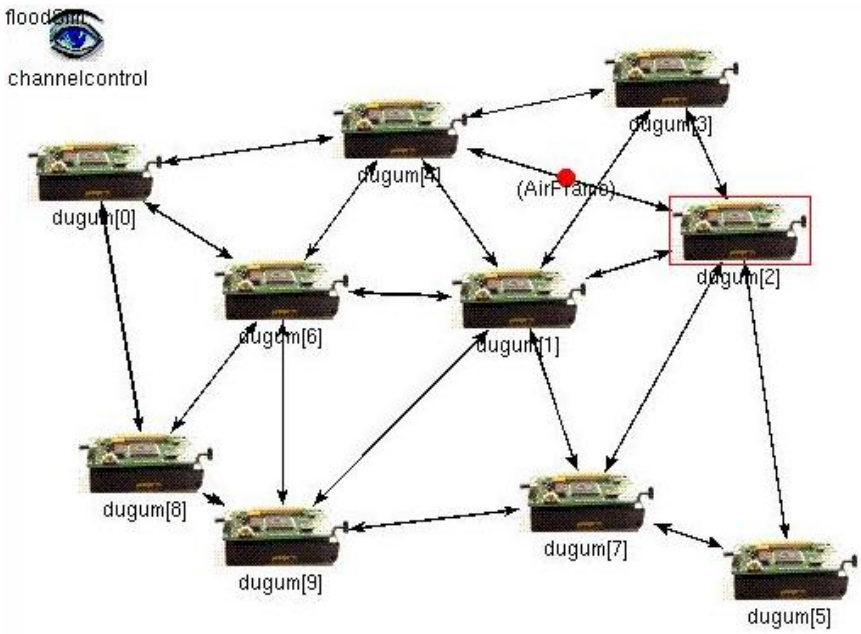
Klasik taşmada düğümler, kendilerine herhangi bir zamanda verilen kullanılabilir enerji miktarına dayanarak, kendi aktivitelerini değiştirmezler. Algılayıcılar kendi enerji kaynaklarının durumuna bağlı olarak hesaplama ile haberleşmeyi ayarlayabilirler. Şekil 4.22'deki iki algılayıcının coğrafi örtüşme bölgesi bulunmaktadır. Bu algılayıcılar kendi verilerini D7 düğümüne taşma ile (broadcast yaparak) gönderdikleri zaman, D7 düğümü m3 olarak işaretlenen bölgeye ait verinin iki kopyasını almış olur.

Enerji kısıtlaması olan bir kablosuz algılayıcı ağında, daha verimli bir haberleşme için, algılayıcılar arasında verimli bir şekilde bilgi yayılımını sağlayan ve SPIN (Sensor Protocols for Information via Negotiation) olarak isimlendirilen uyarlanabilir protokoller kullanılmaktadır. SPIN haberleşme protokolü çalıştıran düğümler, meta-data olarak bilinen yüksek seviyeli veri tanımlayıcıları ile, kendi verilerine isim verirler. Meta-data tanımlayıcısı, ağ üzerindeki fazlalık verinin aktarımını engellemek amacı ile kullanılır. İlaveten, SPIN düğümleri, hem uygulamanın veri bilgisine hem de elverişli kaynak bilgisine dayanarak karar alırlar. Bu sayede,

algılayıcılar sınırlı enerji kaynağına rağmen verimli bir şekilde veri dağıtımı yapabilirler.

İç patlama ve örtüşme problemlerini önlemek amacı ile, SPIN düğümleri veriyi göndermeden önce birbirleri ile müzakere ederler. Müzakere, yanlızca kullanışlı verinin gönderileceği konusunda emin olma konusunda onlara yardım eder. Başarılı bir müzakere için, düğümler tanımlanabilmeli ya da gölzemledikleri veriyi isimlendirebilmeliler. SPIN'de, düğümler veriyi aktarmadan önce kendi kaynaklarını yoklarlar. Herbir algılayıcı düğüm kaynak tüketimini aklında tutan "kaynak yöneticisi"ne sahiptir; uygulamalar veriyi işlemeden ya da göndermeden önce yöneticiyi incelerler. Buna dayanarak algılayıcılar, enerji seviyesi düşük olduğu zaman, belirli aktiviteler üzerinde kesinti yaparlar. Bu özellikler sayesinde klasik taşma protokolünün üç eksik noktası giderilmektedir. Verinin gönderilmesinden önce yapılan müzakere işlemi neticesinde iç patlama problemi giderilir çünki, fazlalık veri mesajlarının yayılımı engellenmiş olur. Meta-data tanımlayıcısının kullanımı ise yaşanabilecek örtüşmeyi ortadan kaldırır çünki, verinin ilgilendikleri kısmını elde etmek için, düğümlere isimlendirme yapma imkanı sağlar.

4.6.1.4 OMNeT++ ile taşma iletişim kuralının benzetimi



Şekil 4.23: Simülasyonu yapılan kablosuz ağ devresi

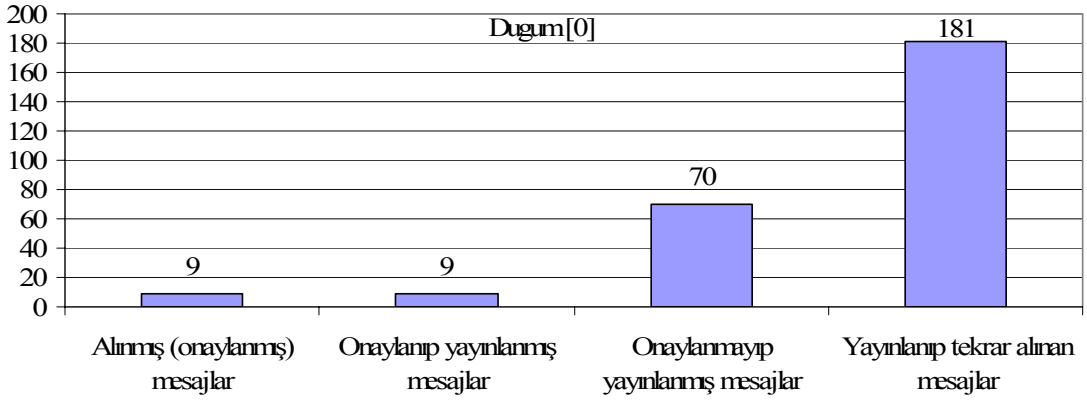
OMNeT++ hareketlilik modülü ile taşma protokolü benzetiminin kaynak kodları Ek-L'de verilmiştir.

4.6.1.4.1 Benzetim sonuçları

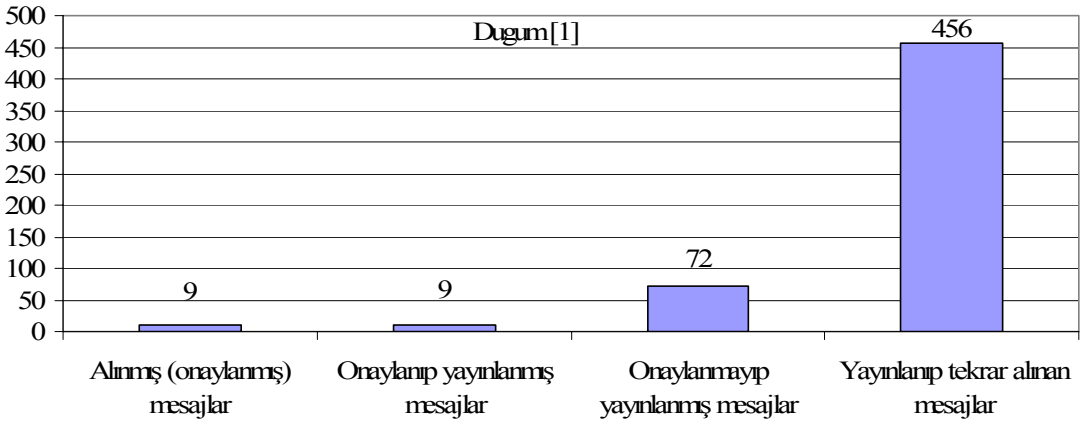
Benzetimi yapılacak ağ oluşturulurken düğüm (Host) olarak kablosuz algılayıcı modeli kullanılmıştır. Kablosuz algılayıcı ağ düzeneği şekil 4.23'de gösterilmiş olup, benzetim sonuçları ise tablo 4.1'de verilmiştir. Elde edilen sonuçlarda etki alanı içerisinde daha fazla komşusu olan düğümlerin yoğun bir haberleşme trafiğine maruz kaldığı görülmüştür. Bunlar taşma iletişim kuralının ilkeleri gereğince yayınlanmış bir bilginin, her bir komşusundan ayrı ayrı olmak sureti ile, bir çok kopyasını almış ve bu bilgiyi aynı ilkeler gereğince yeniden yayınlayarak birbirleri arasında gereksiz bir döngü kurulmasına neden olmuşlardır. Düğümlere göre bir mesajın tekrar yayınlaması grafiği şekil 4.35'de götserilmiştir. 1 numaralı düğümün davranışlarını incelediğimizde, almış olduğu toplam 9 adet mesajı sadece 9 kez yayınlamış olması gerekirken, taşma iletişim kuralının neticesi olarak, bu olayın, tekrar yayınlamalardan dolayı, toplamda 456 kez gerçekleştiğini görmekteyiz. Bu düğüm 50,6 kat oranda fazla aktivasyon içerisinde olmuş ve bu oranda enerji harcamıştır. Diğer düğümlerin oranlarını da hesaba kattığımızda haberleşme ortamının daha yüksek katlarda meşgul edildiğini görmüş oluruz.

Tablo 4.1: Benzetim sonuçları

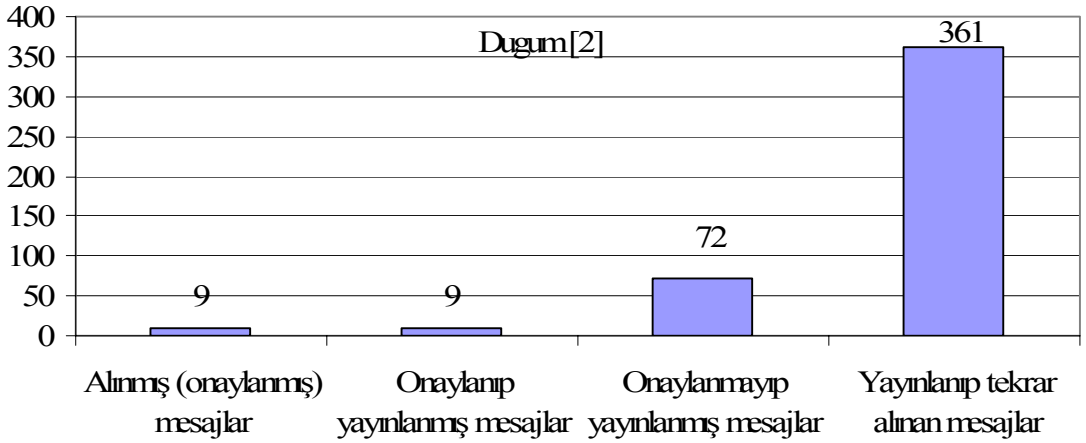
	D.0	**D.1**	**D.2**	**D.3**	**D.4**	**D.5**	**D.6**	**D.7**	**D.8**	**D.9**
Alinmis kullanışlı mesaj sayisi	9	9	9	9	9	9	9	9	9	9
Alinmis ve yayınlanmış kullanışsız mesajlar	9	9	9	9	9	9	9	9	9	9
Kullanışsız mesajların yayinlanma sayısı	70	72	72	72	72	68	72	72	70	72
Mesajların tekrar yayınlanma sayisi	181	456	361	183	363	92	361	270	181	272



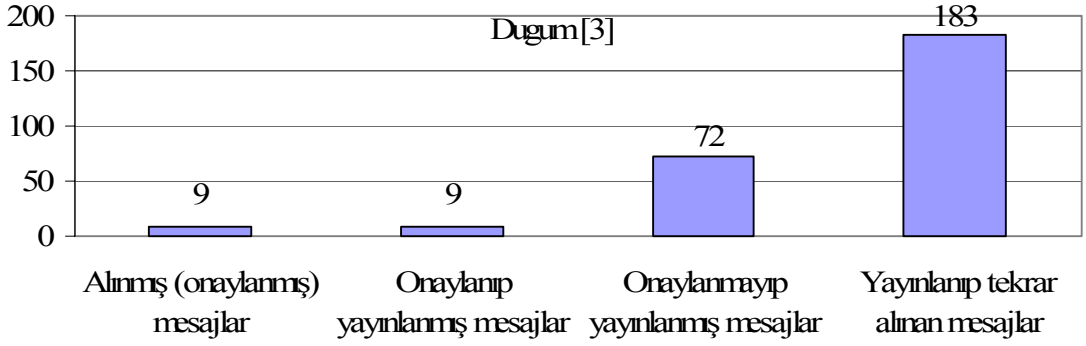
Şekil 4.24: Düğüm[0] için simülasyon sonuç grafiği



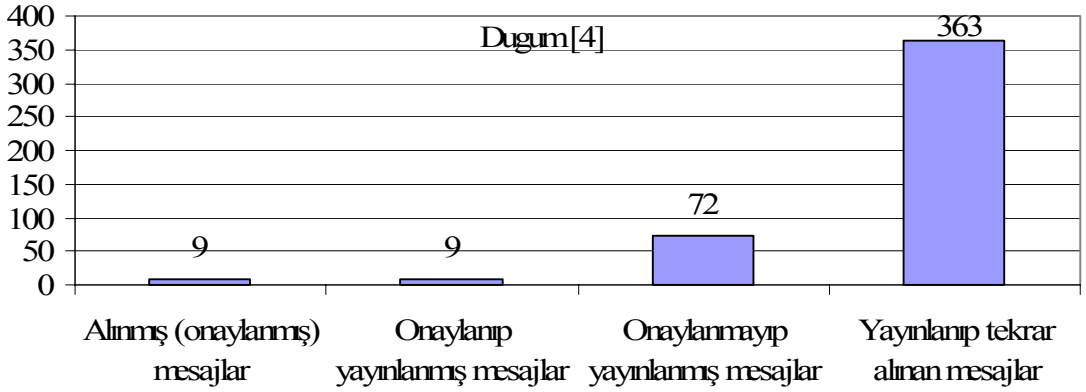
Şekil 4.25: Düğüm[1] için simülasyon sonuç grafiği



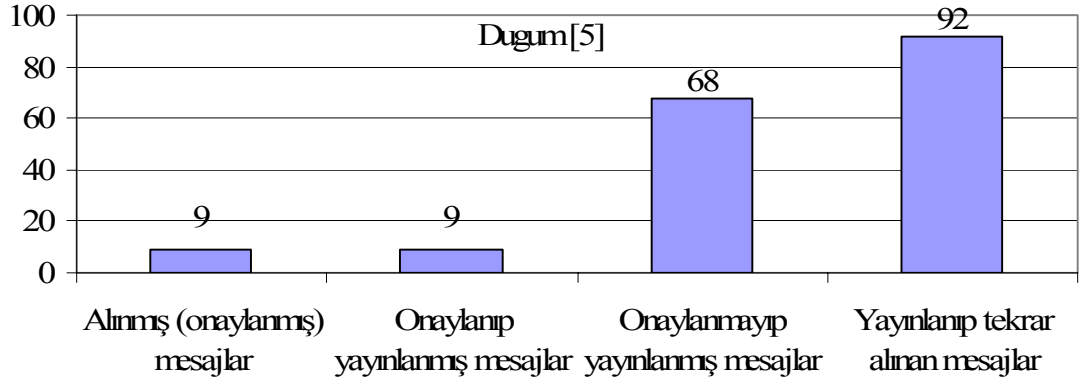
Şekil 4.26: Düğüm[2] için simülasyon sonuç grafiği



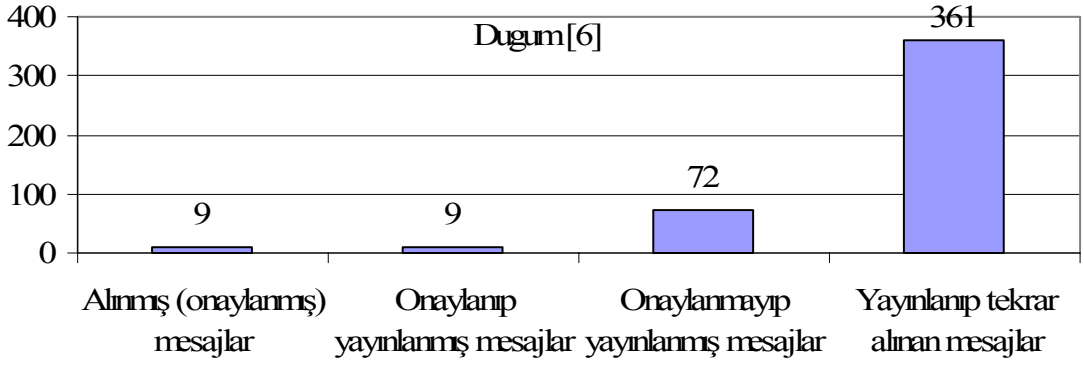
Şekil 4.27: Düğüm[3] için simülasyon sonuç grafiği



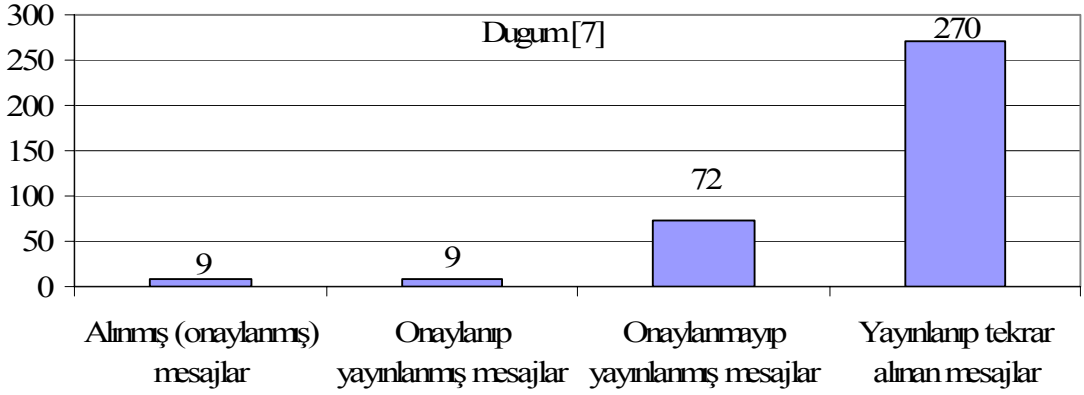
Şekil 4.28: Düğüm[4] için simülasyon sonuç grafiği



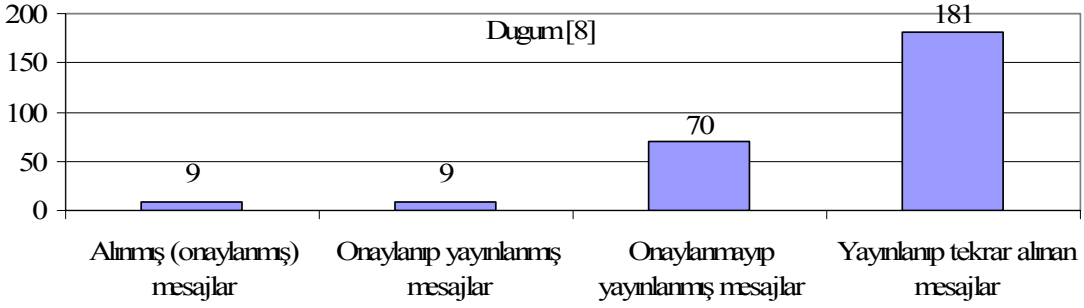
Şekil 4.29: Düğüm[5] için simülasyon sonuç grafiği



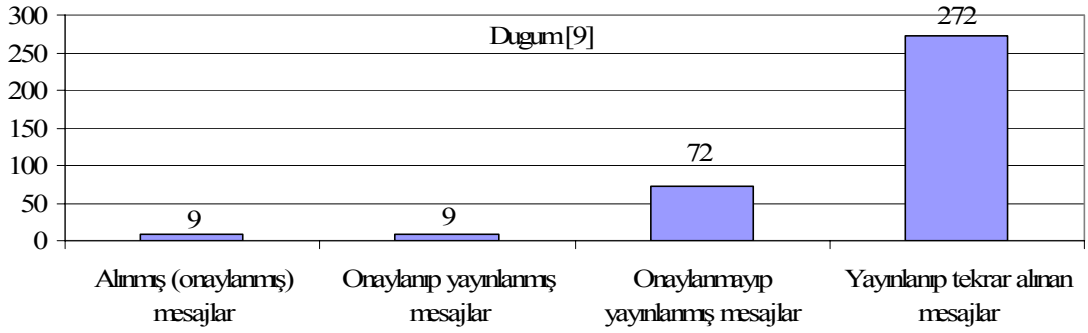
Şekil 4.30: Düğüm[6] için simülasyon sonuç grafiği



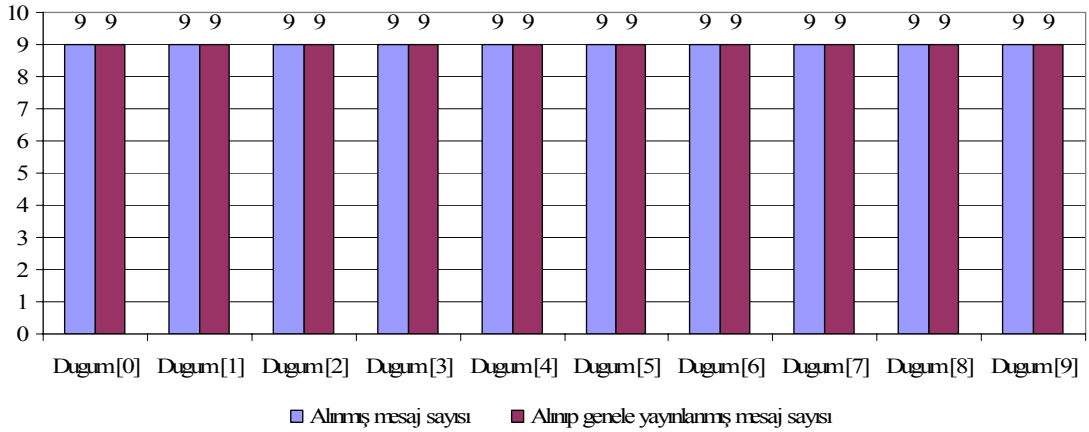
Şekil 4.31: Düğüm[7] için simülasyon sonuç grafiği



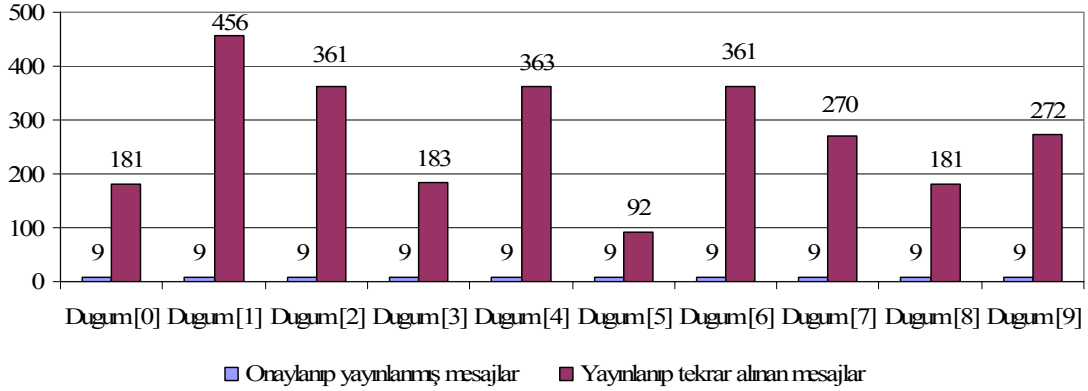
Şekil 4.32: Düğüm[8] için simülasyon sonuç grafiği



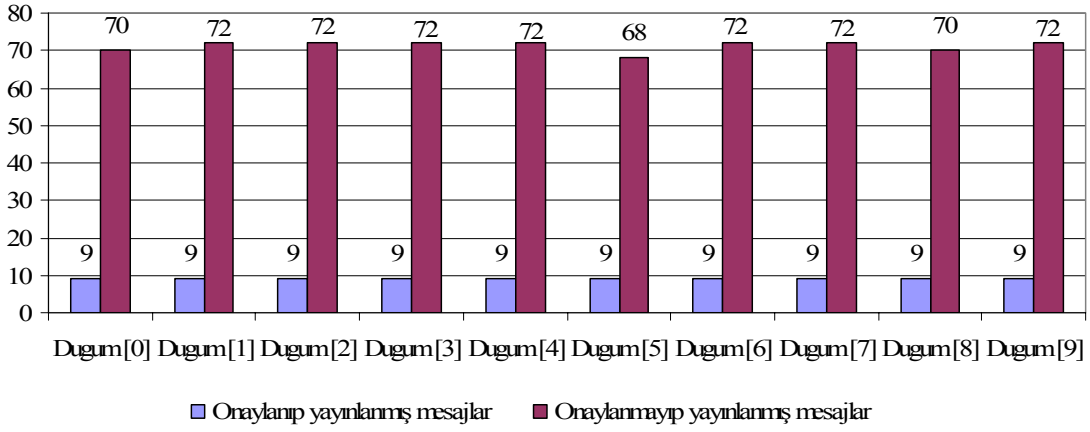
Şekil 4.33: Düğüm[9] için simülasyon sonuç grafiği



Şekil 4.34: Alınmış (kabul edilmiş, onaylanmış) ve yayınlanmış mesajlar



Şekil 4.35: Alınıp yayınlanmış ve tekrar alınmış mesajlar grafiği



Şekil 4.36: Onaylı (kabul edilmiş) ve onaysız (kabul edilmemiş) mesajlar

4.6.2 IEEE 802.11 iletişim kuralı

Bölüm 4.6.2 için [20] ve [21] numaralı kaynaklardan yararlanılmıştır.

Tablo 4.2: IEEE 802.11x standartları

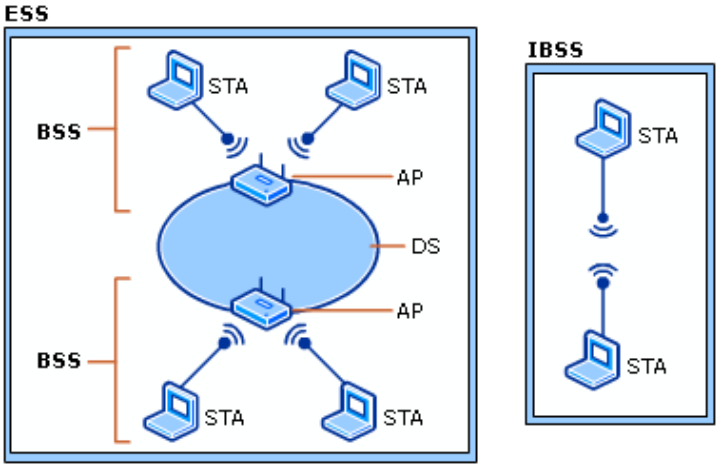
802.11a	5 GHz, OFDM, 54 Mbps (most radios do 4.9GHz through 6.1GHz)
802.11e	Polling, QoS, Hidden node support
802.11g	2.4 GHz OFDM, 54 Mbps (radios do 2.3GHz thru 2.6GHz)
802.11h	5 GHz Extension for Europe auto frequency and power control
802.11j	4.9 GHz Extension for Japan (may apply to USA public safety)
802.11k	Radio Resource Measurement Extensions
802.11m	Maintenance and Management Extensions
802.11n	MIMO Extensions (40MHz bant genişliğinde 12,24,48,96 ve 216 Mbps)
802.11p	WAVE – Moving vehicle Extensions
802.11r	Advanced Roaming and Context Transfer
802.11s	Mesh Extensions
802.11t	Wireless Performance Predictions
802.1x	Authentication and Encryption
WPA	Wi-Fi Protected Access
WME	Wi-Fi Multimedia Extensions
WPA2	Wi-Fi Hot Spot Extensions

IEEE 802.11 protokolü, kablosuz istasyonlar ile kablolu alt yapılar arasında bağlantı sağlayan bir ağ erişim teknolojisidir. IEEE 802.11 protokolü ve ilgili teknolojilerin yayınlanmasi ile birlikte, çeşitli yerelere doğru hareket edebilen bir gezgin

kullanıcının, hareket halinde iken, ağdaki veriye erişimini sağlamak mümkün olmuştur. 802.11 ailesi, Çarpışma Sinyali ile Taşıyıcı Dinleyen Çoklu Erişim (CSMA_CD: Carrier SenseMultiple Access with Collision Dedect) kullanarak bir WLAN'ı tanımlar. 802.11 kablosuz LAN standartları her bir frekans bandı içerisinde birkaç kanal ve veri hızı sağlarlar. Enyüksek veri hızları tablo 4.2'de listelenmiştir:

802.11a [**Wi-Fi**] Dikey Frekans Bölmeli Çoğullama (OFDM: Orthogonal Frequency Division Multiplexing) yöntemi kullanarak 54 Mbps veri hızlı 5 GHz frekansında iletim yapar. **802.11b** [**Wi-Fi**] Doğrudan Dizili Yayılım Spektrumu (DSSS : Direct Sequence Spread Spectrum) yöntemini kullanarak 11 Mbps veri hızlı 2.4 GHz frekansında iletim yapar. **802.11g** OFDM ve DSSS yöntemlerini kullanarak 54 MBps hızlı 2.4 GHz frekansında iletim yapar. IEEE 802.11b ve 802.11g uyumludurlar, bunları kullanan aygıtlar aynı ağda bir arada olabilirler. **802.11h** 100 MBps veri hızlı 5 GHz frekansında iletim yapar. IEEE 802.11b ve 802.11g ile uyumludur, böylece aynı ağda bir arada olabilirler. 2.4 GHz bandı ISM'in (Industrial, Scientific and Medical) lisans gerektirmeyen telsiz bandlarının bir parçasıdır.

4.6.2.1 IEEE 802.11 mimarisi ve bileşenleri



Şekil 4.37: 802.11 yapısı

802.11 mantıksal yapısı bazı ana bileşenlerden oluşur: Station (STA), wireless Access Point (AP), Independent Basic Service Set (IBSS), Basic Service Set (BSS), Distribution System (DS) ve Extended Service Set (ESS). 802.11 mimari

bileşenlerinden bazıları doğrudan donanım aygıtlarına yöneliktir, örneğin STA ve AP'ler gibi. Kablosuz STA bir bağdaştırıcı (adaptör) kart, PC kart veya kablosuz bağlantıyı sağlamak için bir gömülü aygıt içermektedir. Kablosuz AP fonksiyonları, kablosuz STA'lar ve mevcut ağ omurgası arasında, ağ erişimi için bir köprüdür.

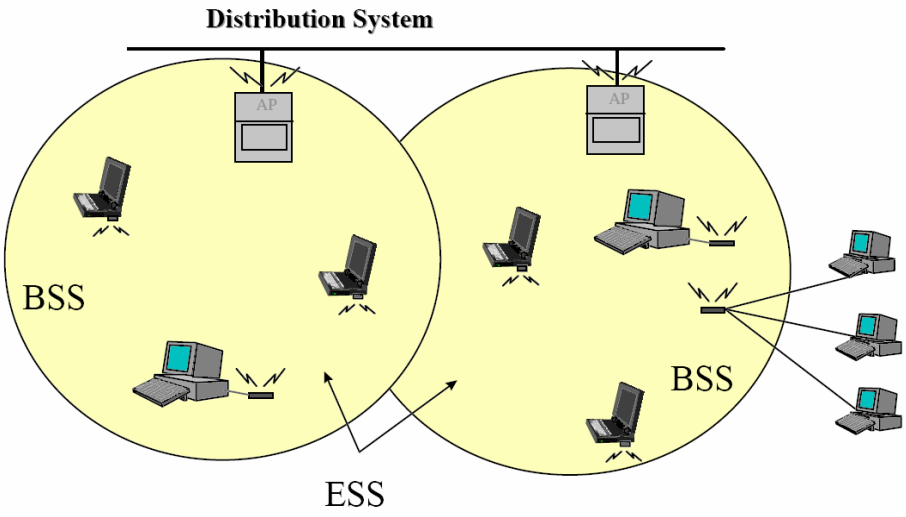
IEEE 802.11 terminelojisine göre BSS olarak isimlendirilen her bir hücrenin, AP olarak isimlendirilen Base Station tarafından kontrol edilmesi durumunda, 802.11 LAN'ı hücresel bir yapıya (sistemin hücreler şeklinde parçalara bölündüğü yapıya) dayandırılmış olur. Kurulumlar, tek bir hücre tarafından tek bir AP ile kablosuz LAN oluşturulması halinde (AP olmadan da çalışabilir), AP DS olarak isimlendirilen omurgalara bağlandığında, bu hücrelerin aracılığı ile yapılacaktır. Birbirine bağlı farklı hücreler içeren, AP'leri ve DS'i ayrı ayrı olan tüm WLAN'ler, OSI modelinin üst katmanında tek bir 802 ağı gibi görünürler ve ESS standardı olarak bilinirler.

IEEE 802.11 standardı aynı zamanda bir Portal kavramını tanımlamaktadır, Portal bir aygıttır ve 802.11 ve diğer bir 802 LAN'ı birbirine bağlar. Bu kavram, "nakil köprüsünün" (translation bridge) fonksiyonelliğine yönelik bir parçanın özet tanımıdır. IEEE 802.11 standardı gerektirmese de, AP ve Portal tipik kurulumlarda tek bir fiziksel gövde üzerinde yer alır. IBSS kablosuz bir ağdır, en az iki STA'dan meydana gelir, kullanıma hazır bir DS'e erişim yapılamadığı yerlerde kullanılır.

Bir IBSS bazen plansız bir (ad hoc) ağa kaynak olarak gösterilir. BSS kablosuz bir ağdır, bir ya da daha fazla kablosuz istemciyi destekleyen tek bir kablosuz AP'den meydana gelir. BSS bazen kablosuz ağ altyapısı olarak kaynak gösterilir. Bir BSS içindeki tüm STA'lar, AP vasıtası ile haberleşirler. AP, kablolu ağa bağlanabilirlik sağlar ve STA ile ya da DS üzerindeki bir düğüm ile haberleşmeyi başlattığı zaman köprüleme yapmış olur.

ESS, aynı kablolu ağ'da bağlanmış iki ya da daha fazla kablosuz AP ile oluşan bir takım ağ'ı ve ayrıca yönlendirici (subnet) tarafından sınırlandırılmış tek bir mantıksal ağ'ı tanımlar. BSS ile ilgili AP'ler DS aracılığı ile birbirlerine bağlanırlar. Bu sayede hareketlilik kabiliyeti kazanırlar, STA'lar bir BSS'den diğerine hareket edebilme imkanı kazanmış olurlar. AP'ler kablolu ya da kablosuz olarak birbirlerine

bağlanabilirler; bununla beraber, çoğu kez kablolar ile bağlanırlar. DS, BSS'leri birbirine bağlamak için kullanılan mantıksal bir bileşendir. DS, BSS'ler arasında STA'ların dolaşımını sağlamak için dağılım servisleri çalıştırır.



Şekil 4.38: 802.11 LAN örneği

4.6.2.2 IEEE 802.11 katmanları ve erişim metodları

Her 802.x protokolü gibi, 802.11 protokolü de MAC ve fiziksel katmanlarına sahiptir, şu anki standart üç PHY (PHYsical) (tümü 1 ve 2Mbit/s'de çalışıyor) katman ile birbirini etkileyen, tek bir MAC'ı tanımlamaktadır:

- 2.4 GHz Band içinde Frekans Atlamalı Yayılım Spektrumu (FH-Frequency Hopping Spread Spectrum)
- 2.4 GHz Band içinde Doğrudan Sıralı Yayılım Spektrumu (DS-Direct Sequence Spread Spectrum)
- InfraRed

<table>
<tr><td colspan="3">802.2</td><td rowspan="2">Data Link Katmanı</td></tr>
<tr><td colspan="3">802.11 MAC</td></tr>
<tr><td>FH</td><td>DS</td><td>IR</td><td>PHY Katmanı</td></tr>
</table>

Şekil 4.39: 802.x katmanları

IEEE 802.11 standardı fonksiyonellik dışında, genellikle MAC katmanları tarafından gerçekleştirilir, 802.11 MAC genellikle üst katman protokolleri ile ilişkili olan diğer fonksiyonları da yerine gertirir, örneğin Fragmentation, Packet Retransmittions ve Acknowledges. MAC katmanı iki farklı erişim yöntemi tanımlar, Distributed Coordination Function ve Point Coordination Function:

4.6.2.2.1 Temel erişim metodu: CSMA/CA

Distributed Coordination Function olarak isimlendirilen ana erişim mekanizması (genellikle CSMA/CA olarak bilinir), Collision Avoidance (Engel Sakınma) mekanizmalı bir Carrier Sense Multiple Access (Taşıyıcıyı Dinleyen Çoklu Erişim)'dir. CSMA iletişim kuralları, CSMA/CD (Collision Dedection-Çarpışma Algılama) protokolü olan Ethernet'in popüler olduğu yerlerde daha iyi bilinir.

Bir CSMA iletişim kuralı şu şekilde çalışır: Gönderme (TX) yapmak isteyen bir istasyon ortamı algılar (duyumsama yapar), eğer ortam meşgul ise (mesela diğer bazı istasyonlar iletim yapıyor olabilirler), o zaman iletim yapmak isteyen istasyon bu isteğini sonraki bir zamana erteler, eğer ortam serbes olarak algılanmış ise o zaman istasyon iletimi gerçekleştirir.

Bu tür iletişim kuralları ortam ağır bir şekilde yüklenmediği zamanlarda oldukça etkilidir, çünkü bu istasyonlara en az gecikme ile iletim yapma imkanı verir, fakat istasyonların aynı anda iletim yapma (çarpışma) ihtimali her zaman vardır, buna sebep olan istasyonların ortamı serbes olarak algılaması ve derhal iletim yapması gerektiğine karar vermesidir.

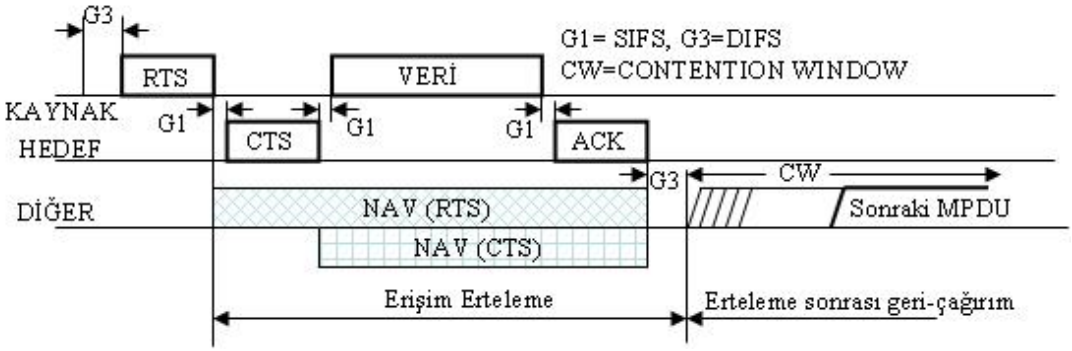
Çarpışma durumları önceden tanımlanmış olmalıdır, bu sayede MAC katmanı paketin tekrar iletimini kendi kendine yapabilir ve gecikmeye sebep olacak üst katman aracılığına gerek olmaz. Ethernet'te ise, bu çarpışma, "exponantial random backoff" ile yeniden iletim yapmak isteyen/yapan, istasyonlar tarafından bilinir. Bu Collision Dedection mekanizmaları kablolu bir ağa yönelik olarak iyi bir fikirdir, fakat Wireless LAN ortam üzerinde kullanılamazlar [20], bunun iki nedeni vardır:

- Çarpışmayı Algılama Mekanizması (Collision Dedection Mechanism) uygulaması, Full Duplex alıcı/verici ile, hemen alma ve hemen iletme kabiliyetlerinin gerçeklenmesini gerektirir, bu yaklaşım uygulamanın bedelini önemli derecede attıracaktır.
- Kablosuz bir ortam üzerinde tüm istasyonların birbirlerini işittiğini varsayamayız (ki bu Collision Dedection için temel varsayımdır) ve bir istasyonun iletim yapma isteği ve serbes ortamı algılama gerçeği, alıcı bölgesi çevresinin mutlaka serbes olduğu anlamına gelmez.

802.11 bu problemleri gidermek için, aşağıdaki gibi bir Mutlak Kabul (Positive Acknowledge) planı ile birlikte, Gürültü Sakınma (Collision Avoidance) mekanizmasını kullanır: Bir istasyon duyularını ortama iletmek ister ve eğer ortam meşgul ise, o zaman bu isteğini erteler. Ortam belirli bir süre için serbes olursa (standart içerisinde DIFS (Distributed Inter Frame Space: Dağıtılmış Ara İleti Boşluğu) olarak isimlendirilir), o zaman istasyon iletim yapabilir, alış yapan istasyon alınan paketin CRC (Cyclic Redundancy Check: Çevrimsel Fazlalık Denetimi)'ni kontrol eder ve bir ACK (Acknowledgement: Alindi) paketi gönderir.

Sanal Taşıyıcı Duyusu: İki istasyonun birbirlerini işitmemelerinden dolayı meydana gelecek çarpışmanın olasılığını düşürmek için, IEEE 802.11 standardı Sanal Taşıyıcı Duyusu (Virtual Carrier Sense) mekanizmasını tanımlanmıştır: Bir istasyon iletmek istediği bir paket için, ilk olarak RTS (Gönderme İsteği) olarak isimlendirilen ve kaynak, hedef ve takip eden işlemin (mesela paket ve kendisine ait ACK) süresini içerecek olan, kısa bir kontrol paketi gönderecektir, hedef istasyon (eğer ortam serbes ise) yanıt olarak aynı süre bilgisini içerecek CTS (Veri Gönderme Müsadesi) kontrol paketi ile cevap verecektir. Tüm istasyonlar ya RTS ya da CTS alarak, kendi Virtual Carrier Sense göstergelerini (NAV (Ağ Atama Vektörü) olarak isimlendirirlen) verilen süre içerisinde set edecekler ve bu bilgiyi ortamı algıladıkları zaman Physical Carrier Sense ile birlikte kullanacaklardır. Bu mekanizma, vericiden "gizli" olan bir istasyon aracılığı ile, alış sahası üzerindeki çarpışmanın olasılığını RTS gönderiminin kısa bir süresi için azaltır, çünkü istasyon CTS'yi işitecek ve ortamı hareketin (işlemin) sonuna kadar "meşgul" olarak rezerv edecektir.

RTS üzerindeki bilginin süresi, ACK süresince verici bölgesini çarpışmalardan korur (bilinen istasyondan itibaren alanın dışındaki istasyonlar aracılığı ile). Ayrıca RTS ve CTS'nin kısa iletiler oldukları gerçeğine de dikkat edilmelidir, bu aynı zamanda çarpışmaların getirdiği ek yükü azaltır, çünkü bunlar tüm paketin iletilmiş olduğu durumdan daha hızlı tanınırlar, (paket RTS'den daha büyük ise, bu stadart kısa paketlerin RTS/CTS işlemleri olmaksızın iletilmesine imkan verir ve RTSThreshold adı verilen bir parametre aracılığı ile her istasyon tarafından kontrol edilir). Aşağıdaki çizim A ve B gibi iki istasyon arasındaki işlemi ve kendi komşularının NAV ayarını gösterir:



Şekil 4.40: NAV (Network Allocation Vector:Ağ Atama Vektörü)

- **MAC Düzeyi Bildirimleri:** Daha önce bahsedildiği gibi, MAC katmanı gönderilmiş herhangi bir parçaya yönelik onay bilgisini aldığını varsayarak, Collision Dedection'ı gerçekleştirir.
- **Parçalama ve Yeniden Kurma:** Tipik LAN iletişim kuralları birkaç yüz byte'lık paketler kullanır (mesela en uzun Ethernet paketi 1518 byte uzunluğuna kadar çıkabilir), kablosuz bir LAN ortamı üzerinde, daha kısa paketler kullanmanın neden tercih edilebilir olduğuna dair bazı sebepler vardır [20]:

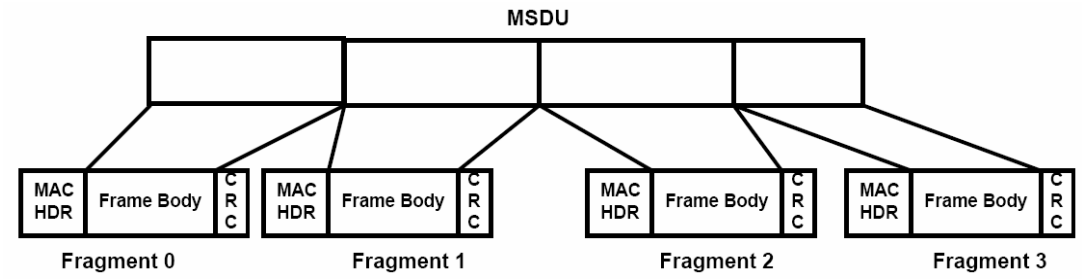
- Bir radyo link ile ilgili yüksek Bit Hata Oranı nedeniyle, paketin bozulma olasılığı paket boyutuna dayalı olarak artar.
- Pakette bozulma olduğu taktirde (ya çarpışmadan ya da gürültüden dolayı), ek yüksüz en küçük paket onun yeniden iletimine neden olur.

- Frekans Atlamalı bir sistemde, sıçrama için ortamda belirli aralıklarla kesinti yapılır (mesela her 20 ms'de bir), daha küçük paketten dolayı iletimin yerleşme zamanından sonrasına ertelenecek olması daha küçük ihtimaldir.

Ethernet üzerinde kullanılan 1518 byte'lık paketler ile ilgilenmeyen yeni LAN iletişim kuralını tanıtmanın da anlamı olmaz, büyük boyutlu iletilerin iletim problemini çözmek için, MAC katmanına parçalama/yeniden kurma mekanizması eklenmiştir. Mekanizma basit bir Gönder-ve-Bekle algoritmasıdır, aşağıdaki işlemler olana kadar iletim istasyonunun yeni bir parça göndermesine izin verilmez:

1. Sözü edilen parça için bir ACK alınır.
2. Parça bir çok kez yeniden nakledilmiş mi diye, karar verir ve tüm iletiyi bırakır.

Verilen bir parçanın yeniden iletimleri arasında, istasyonun farklı bir adrese iletim yapmasına standart izin vermektedir, AP farklı hedefler için önemli paketlere sahip olduğu zaman ve onlardan biri yanıt vermediği zaman bu uygulama yararlı olur.



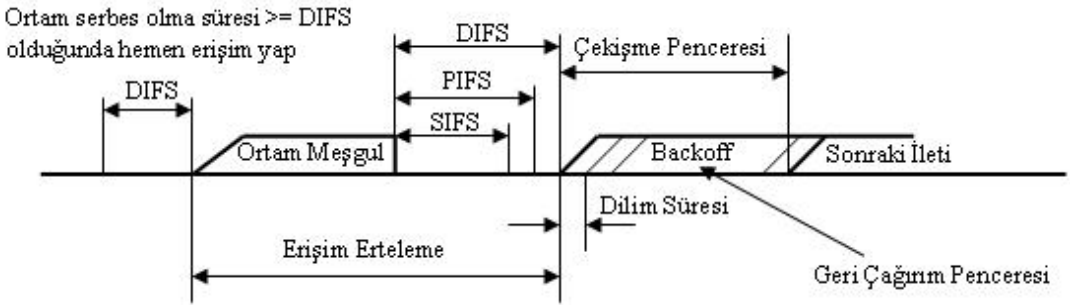
Şekil 4.41: Bir iletinin (MSDU: MAC Service Data Unit) çeşitli parçalara (MPDU: MAC protocol data unit) bölünmesi

Ara İleti Boşlukları: IEEE 802.11 standardı, farklı özellikler sağlayan, 4 tür Ara İleti Boşlukları (Inter Frame Spaces) tanımlamaktadır:

- SIFS (Short Inter Frame Space) tek bir diyalog'a ait iletimleri ayırmak için kullanılır ve en küçük Inter Frame Space'dir ve verilen zaman içinde iletim yapacak bir istasyon daima vardır, bu sebeple diğer istasyonların tamamı üzerinde önceliğe sahip olmaktadır. Bu değer her bir PHY için sabittir ve hesaplanabilir, verici istasyon

alma moduna doğru geri anahtarlama yapabilir ve gelen paketin kodunu çözecebilir, 802.11 FH PHY üzerindeki bu değer 28 mikro saniye'ye set edilir.

- PIFS (Point Coordination IFS), diğer her hangi bir istasyondan önce ortama erişmek için, Access Point (ya da bu durumda Point Coordinator olarak isimlendirilir) tarafından kullanılır. Bu değer SIFS artı bir Slot Time'dır, mesela 78 mikro saniye'dir.
- DIFS (Distributed Inter Frame Space) yeni bir iletim başlatmak isteyen istasyon için, kullanılan Inter Frame Space (Ara İleti Boşluğu)'dur, PIFS artı bir slot süresi olarak hesaplanır, mesela 128 mikro saniyedir.
- EIFS (Extended IFS), anlaşılamayan bir paketi alan istasyon tarafından kullanılır, IFS'nin daha uzun olanıdır, bir istasyonun (Virtual Carrier Sense için süre bilgisini anlayamamıştır) devam eden iletişime yönelik sonraki bir paket ile çarpışmasını önlemek için gereklidir.



Şekil 4.42: Erişim mekanizması şeması (IFS: Inter Frame Space, PIFS: Point Coordination IFS, DIFS: Distributed IFS, SIFS: Short IFS)

Üstel Backoff (Geriçekme) Algoritması: Backoff, ortama erişmek isteyen farklı istasyonlar arasındaki çekişmeyi çözecek bir yöntemdir ve her istasyona 0 ve verilen bir sayı arasında Rastlantısal bir Numara (n) seçimi yapmayı gerektirir. Ortama erişim yapmadan önce Dilim'lere ait bu numarayı bekler, daima farkllı bir istasyonun ortama daha önce erişip erişmediğini kontrol eder. Dilimin başlangıcında ortama erişim yapmış bir istasyonu belirleyecek şekilde Slot Time (*Dilim Süresi*) tanımlanır.

Exponantial Backoff, çarpışma olduğu ve istasyonun bir dilim seçtiği anlamına gelir, üssel rastgele seçim için en yüksek numarayı arttırır. 802.11 standardı Exponantial Backoff Algorithm tanımlar, aşağıdaki durumlarda yürütülmelidir:

- Paketin ilk iletiminden önce, istasyon ortamı algıladığı ve ortamın meşgul olduğu zaman,
- Her yeniden iletimin ardından,
- Başarılı bir iletimdenden sonra.

Tek sorun, istasyon yeni bir paketi iletmeye karar verdiği zaman bu mekanizma kullanılmaz ve ortam çoğu DIFS (*Distributed Inter Frame Space*) için serbes olur.

İstasyon var olan bir Hücreye (BSS) Nasıl Katılabilir?: İstasyon bir BSS'ye erişim yapmak istediği zaman (ya enerjilendirildikten sonra, uyku modundan sonra ya da sadece BSS alanına giriyorken), senkronizasyon bilgisini Access Point'den (ya da ad-hoc (plansız) modda olduğunda diğer istasyonlardan) almalıdır. İstasyon bu bilgiyi aşağıdaki 2 durumdan biri aracılığı ile elde eder, iki yöntemde geçerlidir ve ikisinden biri "güç tüketimi/performastan ödün verme" durumlarına göre seçilebilir:

1. Pasif tarama: Bu durumda istasyon yanlızca AP'den bir Beacon Frame (Hat kesinti İletisi) almayı ister, (hat kesinti iletisi, senkronizasyon bilgisi ile birlikte AP tarafından gönderilen, sürekli bir iletidir).
2. Aktif Tarama: Bu durumda, istasyon Probe Request Frames (Araştırma İsteği İletileri) göndererek bir Access Point bulmayı dener ve AP'den Probe Response (Araştırma Cevabı) bekler.

Kimlik Denetim Süreci (Authentication Process): İstasyon bir Access Point bulmuştur ve onun BSS'ine katılmaya karar vermiştir, bilginin AP ve istasyon arasında değiş tokuşu demek olan Authentication Process'i gözden geçirecektir, burada her bir taraf verilen şifre ile bilginin doğruluğunu tespit eder.

İlişki Süreci (Association Process): İstasyon kimlik denetimine tabi tutulduğunda İlişki Süreci'ni başlatacaktır, Association Process istasyonlar hakkındaki bilginin ve BSS kabiliyetlerinin değişmesidir ve buna DSS (Distribution System Servise) (istasyonun o anki konumunu bilmeye yarayacak Access Point'ler seti) imkan verir. İişki süreci tamamlandıktan sonra, istasyon verinin alma ve göndermesini yapabilir.

Amaçsız Gezinme (Roaming): Avare dolaşma, bir hücreden (ya da BSS'den) diğerine bağlantıyı kaybetmeden, hareket etme sürecidir. Bu fonksiyon iki ana farklılığı dışında cep telefonlarının el değiştirme işlemine benzerdir: Paket tabanlı bir LAN sistemi üzerinde hücreden hücreye geçiş, paket iletimleri arasından yapılabilir, telefon'dan farklı olarak (ki orada geçiş telefon konuşması süresince olur) bu olay LAN dolaşımını bir parça daha kolay yapar, fakat bir ses sistemi üzerinde hattın geçici olarak kesilmesi konuşmayı etkilemeyebilir, paket tabanlı bir ortamda iken performansı önemli miktarda düşürecektir çünkü yeniden iletim üst katman protokolleri tarafından yerine gitirilecektir. 802.11 standardı, dolaşımın nasıl yapılması gerektiğini tanımlamaz, fakat bununla ilgili temel araçları tanımlar, bu tanım aktif/pasif taramayı ve yeniden ilişkilendirme sürecini içermektedir, bu süreçte bir Access Point'den diğerine dolaşan istasyon yeni bir tanesiyle ilişkilendirilmiş olacaktır.

Senkronizasyonu Koruma: İstasyonlar senkronizasyonu muhafaza etmelidirler, bu eş zamanlı sıçramayı ve güç korunumu gibi diğer fonksiyonları korumak için gereklidir. Saat bilgisinin AP saat bilgisi ile güncellenmesi için, BSS içinde istasyonlar tarafından aşağıdaki mekanizma kullanılır: AP, Beacon Frames (Hat Kesintisi İletileri) iletilerini düzenli olarak gönderir, bunlar iletim anındaki AP'nin saat bilgisini içerirler (iletim anı, transmisyonun meydana geldiği zamandır ve Beacon Frame'lerin CSMA kuralları kullanılarak iletilmesi nedeni ile, iletim için sıraya koyulmaları, transmisyonda gecikmeye neden olur). İstasyonlar alış anında kendi saat (clock) değerlerini kontrol ederler ve AP'nin saati ile senkronizasyonu korumak için düzeltme yaparlar, bu sayede saatin sapmasını önlemiş olurlar, çünkü bir kaç saatlik bir işlemden sonra senkronizasyonu kaybetme riskleri vardır.

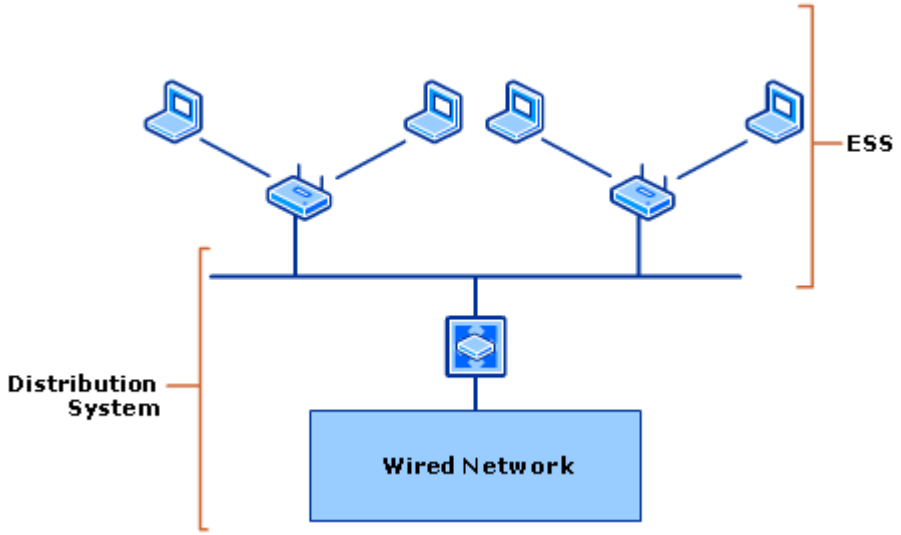
4.6.2.3 IEEE 802.11 işletim modları

IEEE 802.11 altyapı ve plandız (ad-hoc) işletim modlarını tanımlar. Her iki işletim modunda, bir SSID (Service Set IDentifier), aynı zamanda kablosuz ağ adı olarak bilinir, kablosuz ağı tanımlar. SSID, kablosuz AP (altyapı modu için) üzerinde ya da baştaki kablosuz bir istemci (plansız mod için) üzerinde tanımlı bir isimdir ve

kablosuz ağı tanımlar. SSID periyodik olarak, kablosuz AP ya da baştaki kablosuz istemci tarafından, “beacon frame” (hat kesintisi iletisi) olarak bilinen özel bir 802.11 MAC yönetim iletisi kullanılarak, ilan edilir.

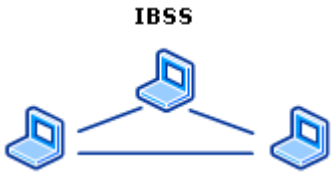
4.6.2.3.1 IEEE 802.11 altyapı modu

Altyapı modunda, en az bir tane kablosuz AP ve bir tane kablosuz istemci vardır. Kablosuz istemci, alışılmış bir kablolu ağın kaynaklarına erişmek için kablosuz AP kullanır. Kablolu ağ, kablosuz AP’nin yerleştirilmesine bağlı olarak, bir intranet ya da internet organizasyonu olabilir. ESS (Extended Service Set: Geliştirilmiş Hizmet Takımı) aşağıdaki şekilde gösterilmiştir.



Şekil 4.43: 802.11 Altyapı Modu

4.6.2.3.2 IEEE 802.11 plansız mod



Şekil 4.44: Plansız Mod içindeki Kablosuz 802.11 İstemcileri

Plansız mod’da, kablosuz istemciler kablosuz bir AP kullanmadan, şekil 4.44’de gösterildiği gibi, birbirleri ile doğrudan haberleşirler ve aynı zamanda eşler-arası

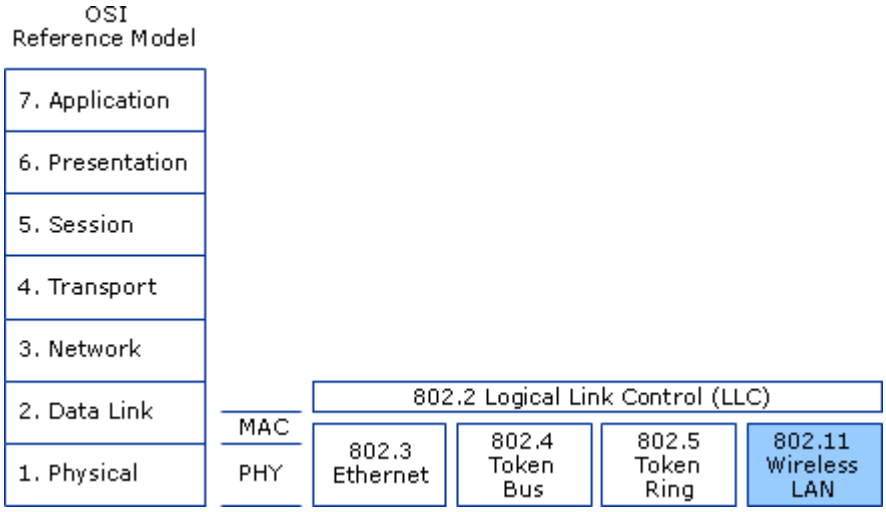
(peer-to-peer) mod olarak da isimlendirilir. Plansız mod içindeki kablosuz istemciler IBSS'yi biçimlendirirler. IBSS içindeki kablosuz istemcilerden birtanesi kablosuz AP'nin bazı sorumluluklarını üzerine alır. Bu sorumluluklar periyodik işret verme işini ve yeni üyelerin kimlik denetimi (doğrulanma) işini kapsamaktadır. Bu kablosuz istemci, bilgiyi kablosuz istemciler arasında anahtarlamak için, bir köprü gibi davranmaz. Plansız mod, kablosuz bir AP sunumcu olmadığı zaman, kablosuz istemcilerini bir arada bağlamak için kullanılır.

4.6.2.4 IEEE 802.11 iletişim kuralları ve teknolojileri

- **802.11:** IEEE 802.11 kablosuz standardı fiziksel katman ve MAC (*Media Access Control*) katmanı özelliklerini tanımlar.
- **802.1X:** Ethernet ağlarına uygun kimlik denetimi yapılmış ağ erişimi sağlamak amacı ile kullanılan, port-tabanlı ağ erişim kontrolünü tanımlar.
- **EAPOL (Extensible Authentication Protocol (EAN) over LAN):** EAP, PPP (point-to-point protocol: uçtan-uca iletişim kuralı) tabanlı bir kimlik denetim mekanizmasıdır ve uçtan-uca LAN bölümleri üzerinde kullanım için uyarlanmıştır.
- **WEP (Wired Equivalent Privacy):** WEP, kablosuz düğümler arasında gönderilen veriyi şifreleyerek, kaliteli gizlilik sağlar.
- **WAP (Wi-Fi Protected Access):** IEEE 802.11i tasdik edilinceye kadar geçici kabul edilmiş bir standarttır. Bu standartlar, WEP standardı için vekil olma ile ilgilidirler, veri şifreleme ve ağ kimlik denetimi ile ilgili sağlam yöntemler sunarlar.

4.6.2.4.1 IEEE 802.11 iletişim kuralı

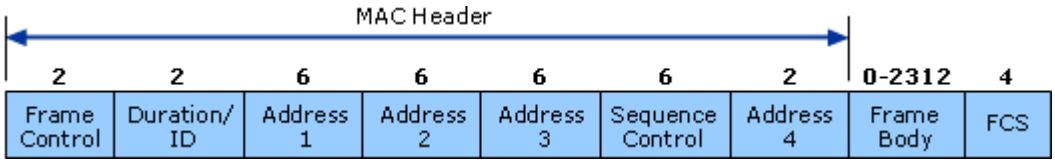
IEEE standartları topluluğu OSI modelinin Data-Link katmanının yerine iki ayrı katman tanımlıyor: LLC (Logical Link Control) ve MAC (Media Access Control). IEEE 802.11 kablosuz standardı, fiziksel katman ve MAC katmanı için özellikleri tanımlar ve aşağıdaki şekilde gösterildiği gibi LLC katmanına kadar haberleşir. 802.11 mimarisindeki bileşenlerin tümü ya Data-Link katmanı içindeki MAC alt katmanı ya da fiziksel katman içerisine denk gelirler.



Şekil 4.45: 802.11 ve OSI Modeli

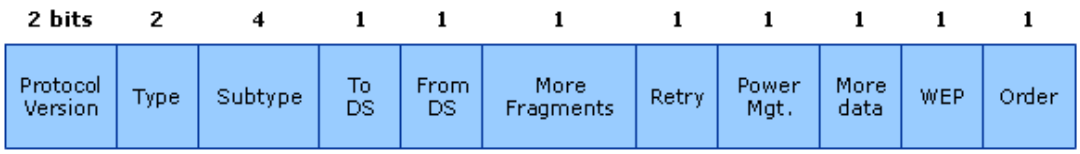
4.6.2.4.2 IEEE 802.11 MAC iletisi

802.11 MAC iletisi, aşağıdaki şekilde gösterildiği gibi MAC başlığı (header) ileti gövdesi ve bir FSC (Frame Check Sequence)'den meydana gelir. Aşağıdaki şekilde yer alan sayılar her bir alanın byte'larını temsil eder.



Şekil 4.46: 802.11 MAC İleti Biçimi

İleti Kontrol Alanı (Frame Control Field): İleti Kontrol (Frame Control) alanı, aşağıdaki şekilde gösterilmiştir, 802.11 MAC iletisinin türünü tanımlamak için kullanılan ve MAC iletisinin nasıl işleneceğini anlamaya yönelik olarak takip eden alanlar için gerekli bilgiyi sağlayacak kontrol bilgisini içerir.



Şekil 4.47: İleti Kontrol Alanı

Her bir İleti Kontrol (Frame Control) alanı alt bölümlerinin tanımı aşağıdaki gibidir:

- Protocol Version, kullanılan 802.11 iletişim kuralının mevcut sürümünü sağlar. STA'ları alarak, alınmış iletiye ait iletişim kuralı sürümünün desteklenip desteklenmediğini tanımlamak için, bu değeri kullanır.
- Type ve Subtype, iletinin fonksiyonunu tanımlar. Üç farklı ileti alan türü vardır: kontrol, veri ve yönetim. Her bir ileti türünün bir çok alt türü vardır. Her alt tür, kendisi ile ilişkilendirilmiş ileti türünü gerçekleştirecek bir fonksiyonu tanımlar.
- To DS ve From DS (Distributed System), iletinin DS'e doğru gidip gitmediğini ya da DS'den çıkıp çıkmadığını belirtir ve yanlızca bir AP ile ilişkilendirilmiş STA iletilerinin veri türü dahilinde kullanılır.
- More Fragments, iletinin daha fazla parçasının olup olmadığını belirtir, her bir veri veya yönetim türünü takip içindir.
- Retry, her bir veri veya yönetim iletisi için, yeniden iletilip iletilmediğini belirtir.
- Power Management, STA gönderiminin aktif mod ya da güç-korunumu modunda olup olmadığını belirtir.
- More Data, bir STA'nın güç-korunumu modunda ve AP'nin gönderecek çok iletisi olduğunu belirtir. Aynı zamanda, takip eden ilave broadcast/multicast iletilerini belirtme amacı ile AP'ler için kullanılır.
- WEP, ileti içinde şifreleme ve kimlik denetiminin olup olmadığını belirtir. Kimlik denetimi için veri ve yönetim iletilerine, set edilebilir.
- Order, tüm alınmış veri iletilerinin sıra ile işlenme zorunluluğunu belirtir.

Duration/ID Alanı: Bu alan, Power Save (PS) Poll dışındaki tüm kontrol tip iletilere yönelik olarak, sonraki iletiyi almada gerekli süreyi göstermek için kullanılır. Alt tür PS Poll olduğunda STA iletiminin AID (Association Identity) bilgisini içerir.

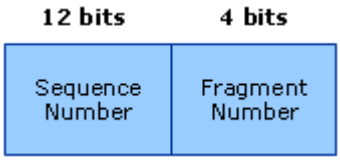
Adres Alanları: İleti türüne bağlı olarak aşağıdaki adres türlerinin kombinasyonunu kapsarlar:

- BSSID (BSS Identifier) her bir BSS'yi benzeri olmayacak şekilde tanımlar. İleti BSS altyapısındaki bir STA'dan geldiği zaman, BSSID AP'nin MAC adresidir. İleti

IBSS'deki bir STA'dan geldiği zaman ise, BSSID rastlantısal olarak üretilir, IBSS ile ilişkilendirilmiş STA'nın MAC adresi lokal olarak yönetilir.

- DA (Destination Address) iletiyi alacak en son hedefin MAC adresini belirtir.
- SA (Source Address) başlangıçta iletiyi oluşturan ve ileten orjinal kaynağın MAC adresini belirtir.
- RA (Receiver Address) kablosuz ortam üzerinde ileti almak için, elde hazır olan sonraki STA'nın MAC adresini belirtir.
- TA (Transmitter Address) kablosuz ortam üzerinden ileti gönderen STA'nın MAC adresini belirtir.

Sequence Control: Sequence Control alanı iki alt cisim içerir, Fragment Number ve Sequence Number alanı, aşağıdaki şekilde gösterilmiştir.



Şekil 4.48: Sequence Control Alanı

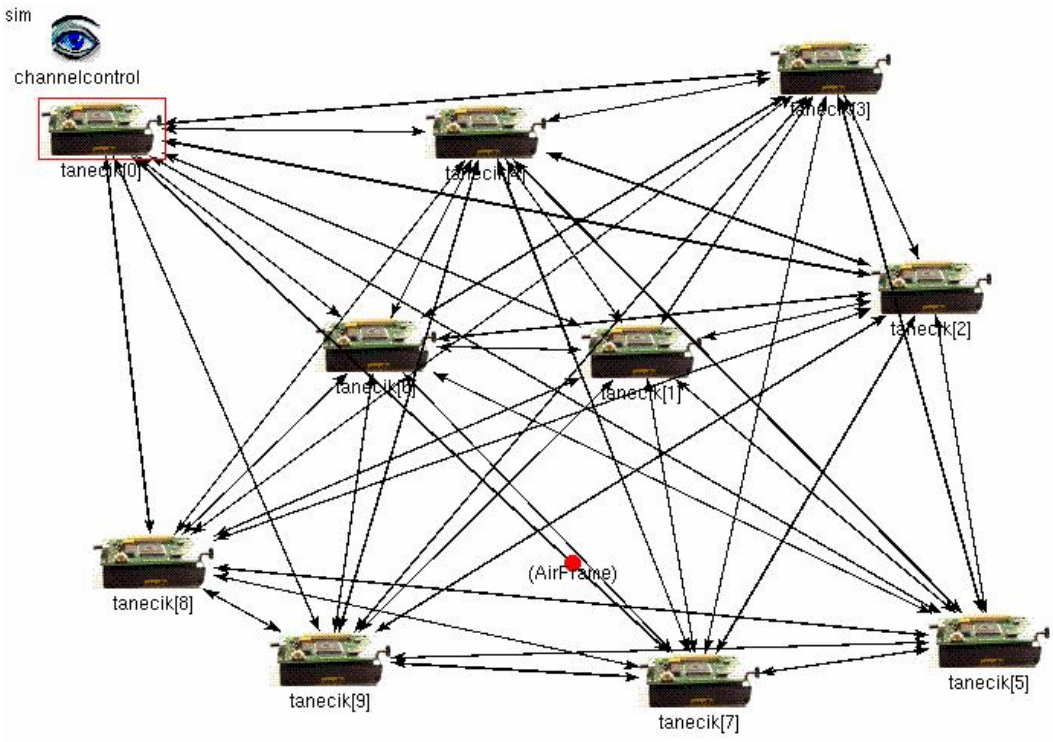
Sequence Control alanı alt cisimlerinin her biri aşağıda tanımlanmıştır: Sequence Number her bir iletinin dizi numarasını belirtir. Dizi numarası, parçalara ayrılmış bir iletide gönderilen her bir ileti için aynıdır; aksi halde, numara 4095'e ulaşana kadar birer birer arttırılır, 4095'e ulaştığında ise yeniden sıfırdan başlar. Fragment Number parçalara ayrılmış bir iletinin gönderilen her bir iletisine ait sıra numarasını belirtir. Başlangış değeri 0'a set edilir ve parçalara ayrılmış iletinin giden sonraki her bir iletisi için birer birer arttırılır.

İleti Gövdesi: Yönetim ya da veri iletileri ihtiva eden veri ya da bilgi içerirler.

İleti Kontrol Numarası: STA iletimi MAC başlığının tüm alanları üzerinde bir CRC (Cyclic Redundancy Check) ve FCS değerini üretmek için ileti gövde alanı kullanır. STA alımı ise, iletim süresince iletide bir hata olup olmadığını doğrulamak için, FCS alanı ile ilgili kendisine ait değeri tanımlamada, aynı CRC hesaplamasını kullanır.

4.6.2.5 OMNeT++ ile IEEE 802.11 iletişim kuralının benzetimi

OMNeT++ hareketlilik modülü ile IEEE 802.11 iletişim kuralı benzetiminin kaynak kodları Ek-M'de verilmiştir.

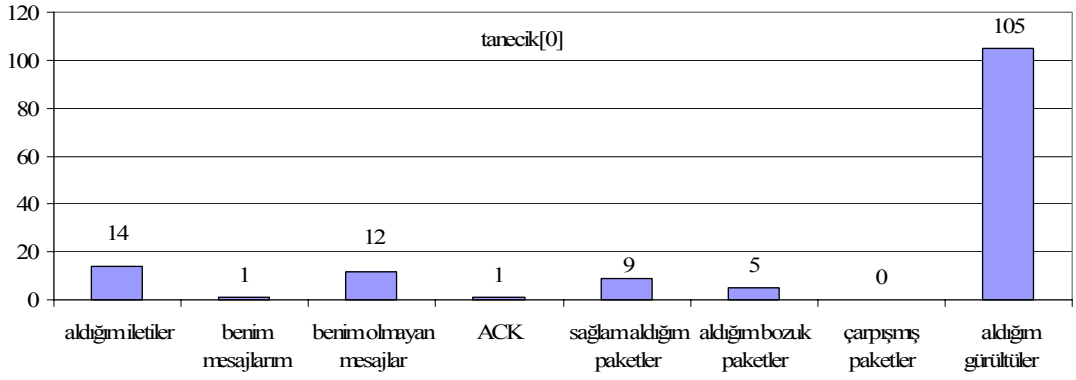


Şekil 4.49: Simülasyon devresi

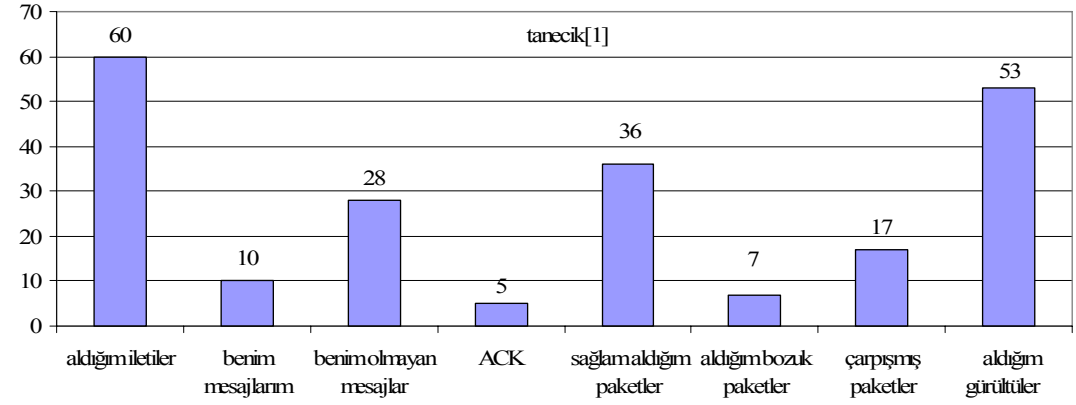
4.6.2.5.1 Benzetim sonuçları

Tablo 4.3: Benzetim sonuçları

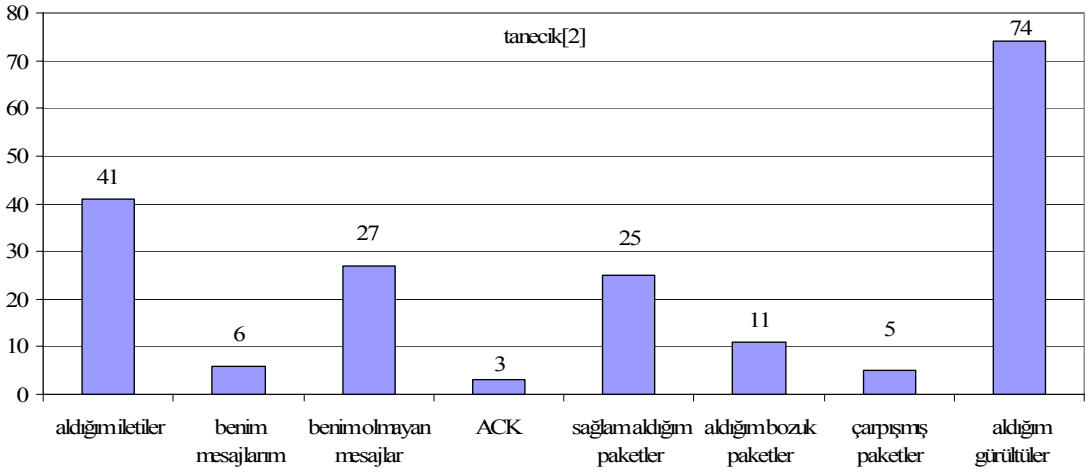
	T.0	T.1	T.2	T.3	T.4	T.5	T.6	T.7	T.8	T.9
Alinan ileti sayisi	14	60	41	41	42	22	51	35	26	30
Düğümlerin kendi mesajları	1	10	6	6	5	4	8	6	4	6
Düğümlere ait olmayan mesajlar	12	28	27	27	27	15	27	21	20	21
Gonderilen ACK iletileri	1	5	3	3	3	2	3	3	2	3
Doğru alınan paket sayısı	9	36	25	25	28	14	33	23	24	22
Hatalı alınan paket sayısı	5	7	11	11	7	7	7	7	2	8
Paketlerin çarpışma sayısı	0	17	5	5	7	1	11	5	0	0
Alınan gürültü sayısı	105	53	74	69	68	93	63	83	96	86



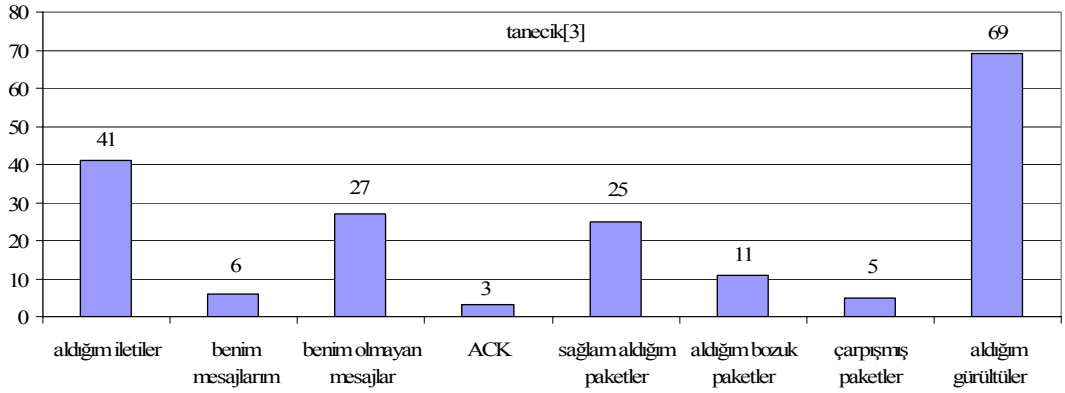
Şekil 4.50: Tanecik[0] için simülasyon sonuç grafiği



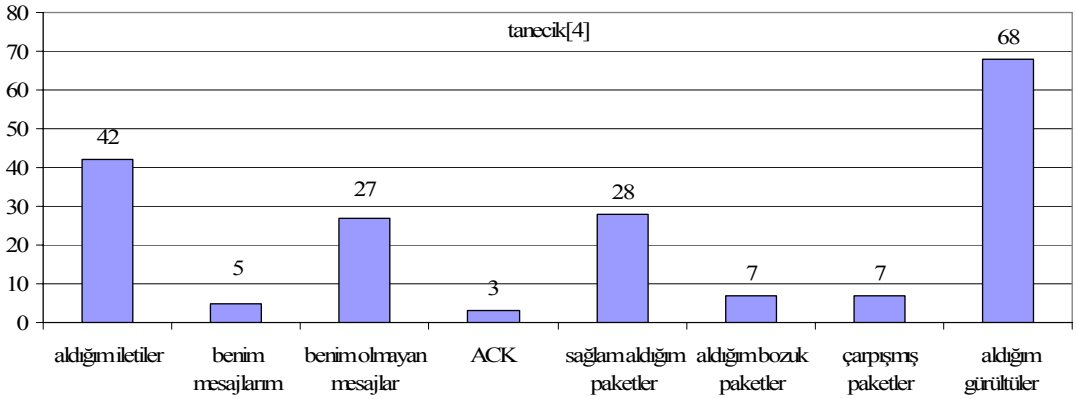
Şekil 4.51: Tanecik[1] için simülasyon sonuç grafiği



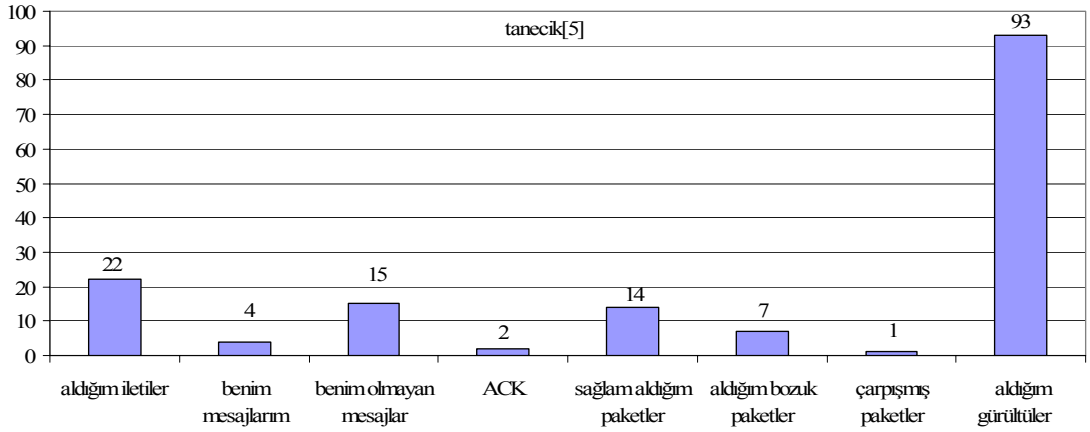
Şekil 4.52: Tanecik[2] için simülasyon sonuç grafiği



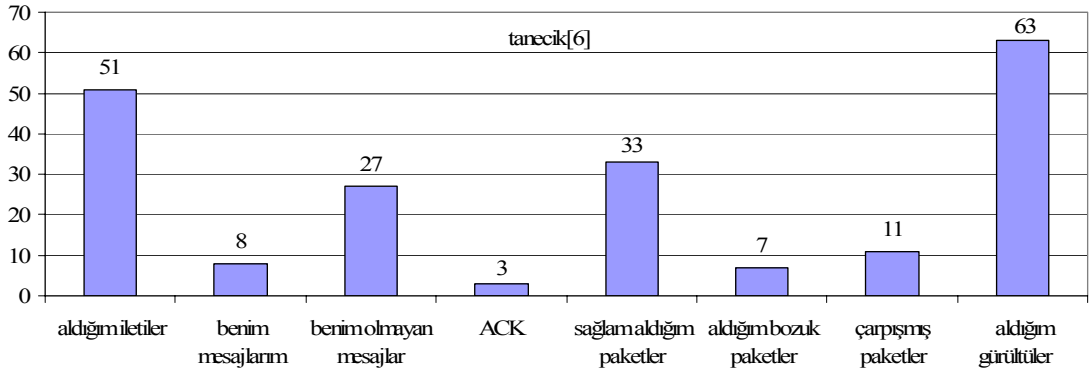
Şekil 4.53: Tanecik[3] için simülasyon sonuç grafiği



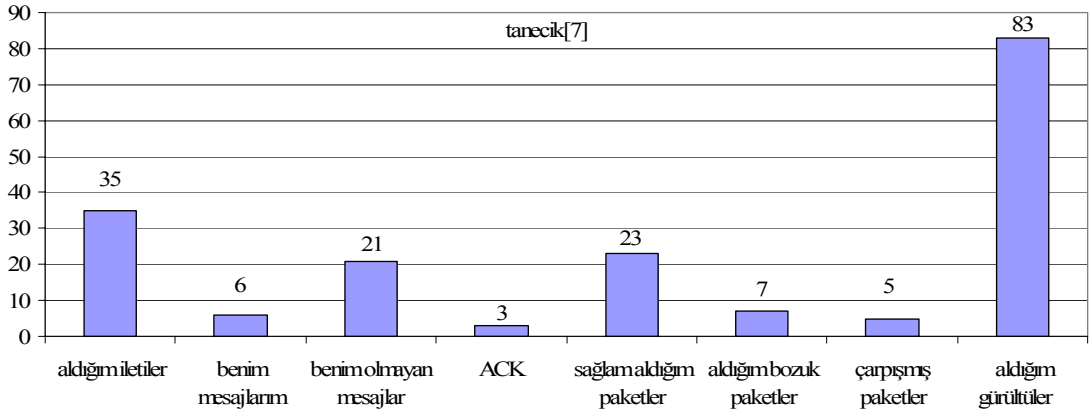
Şekil 4.54: Tanecik[4] için simülasyon sonuç grafiği



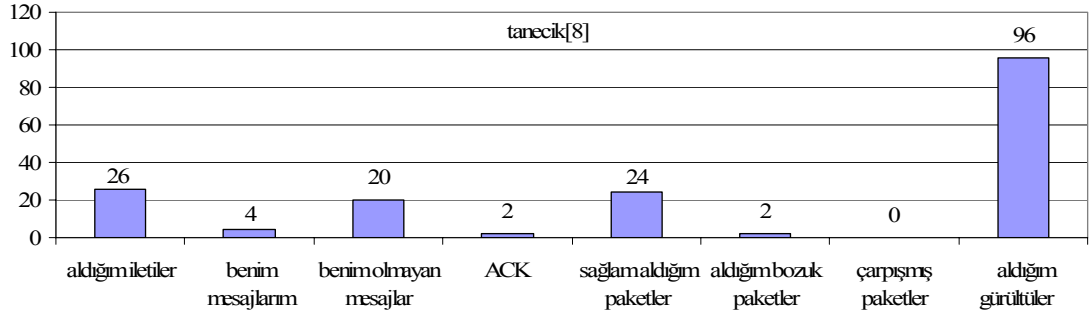
Şekil 4.55: Tanecik[5] için simülasyon sonuç grafiği



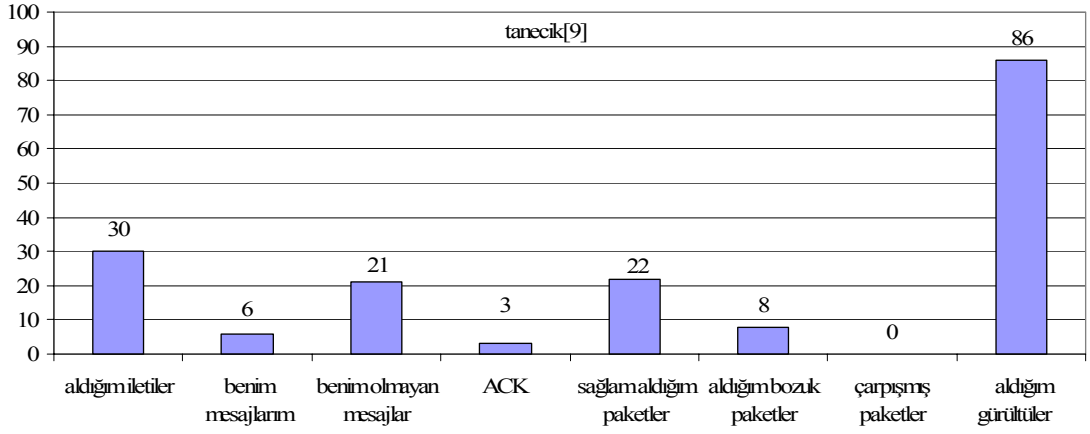
Şekil 4.56: Tanecik[6] için simülasyon sonuç grafiği



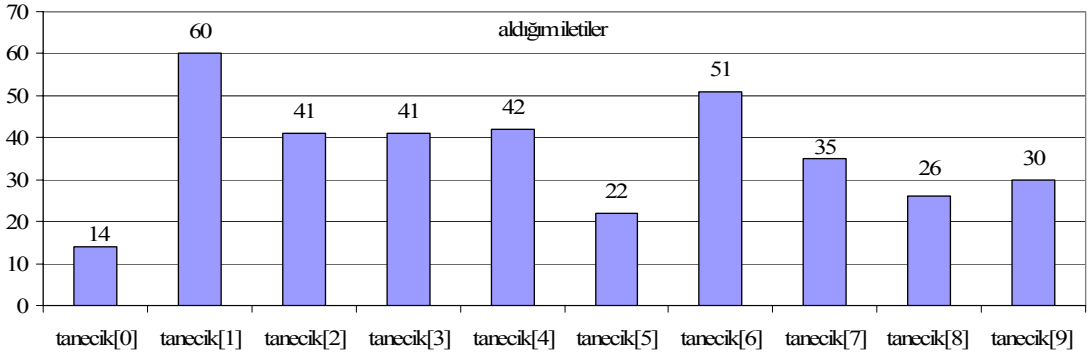
Şekil 4.57: Tanecik[7] için simülasyon sonuç grafiği



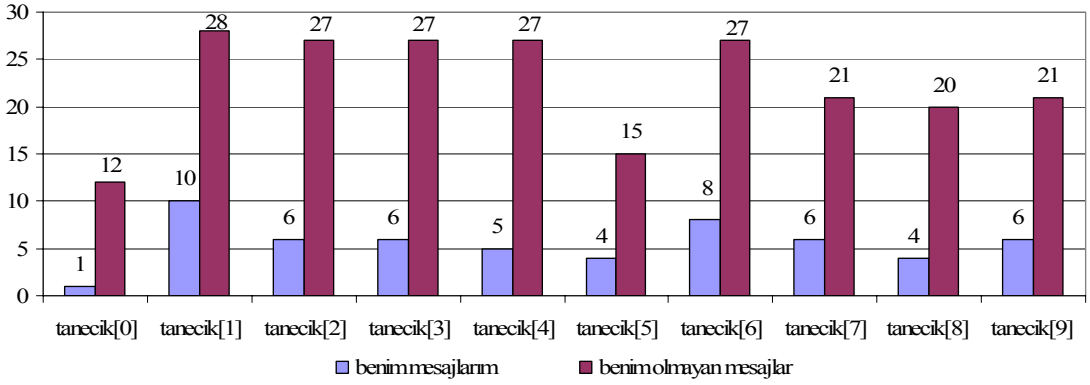
Şekil 4.58: Tanecik[8] için simülasyon sonuç grafiği



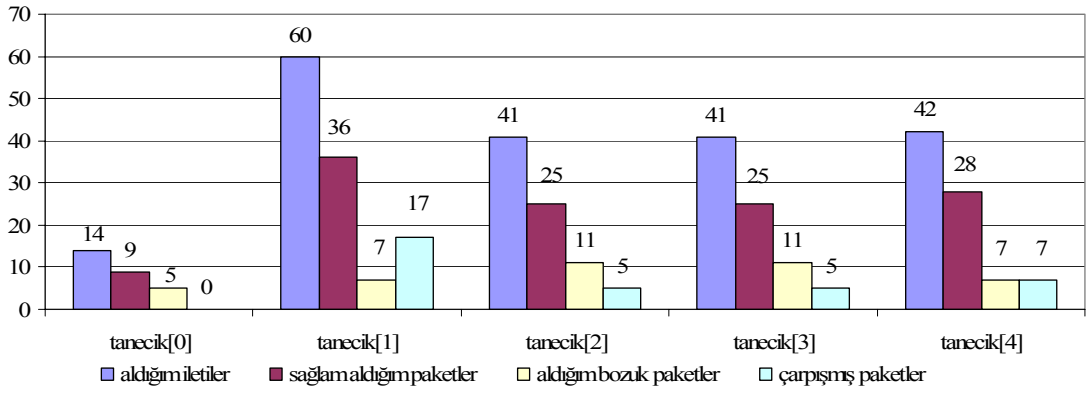
Şekil 4.59: Tanecik[9] için simülasyon sonuç grafiği



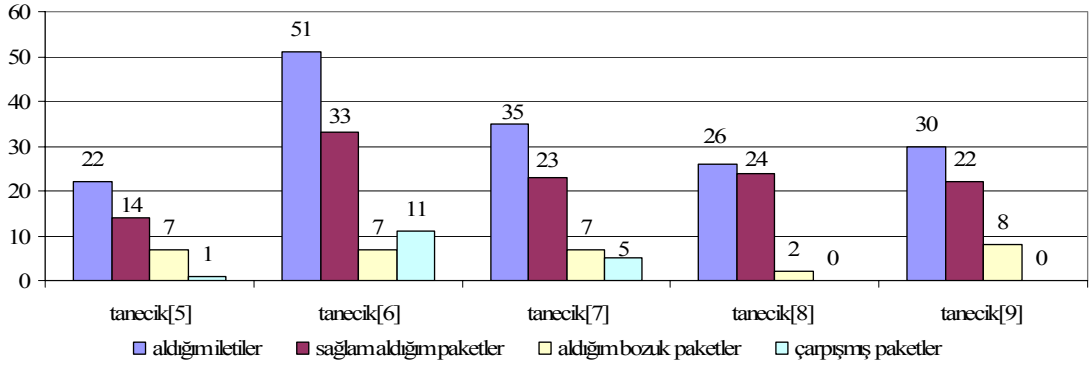
Şekil 4.60: Tanecik(düğüm)lerin ileti alma durumları



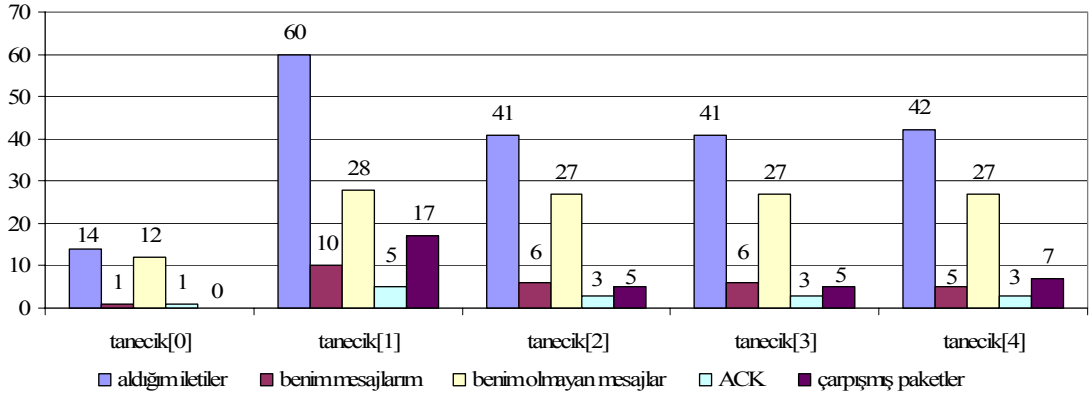
Şekil 4.61: Gelen iletilerden taneciklere ait olan ve olmayan mesajlar



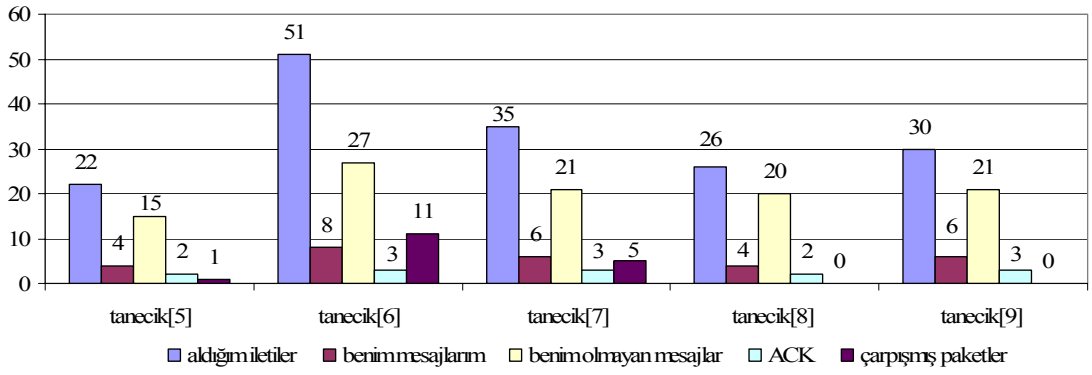
Şekil 4.62: [Alınan iletiler]=[sağlam]+[bozuk]+[çarpışmış]



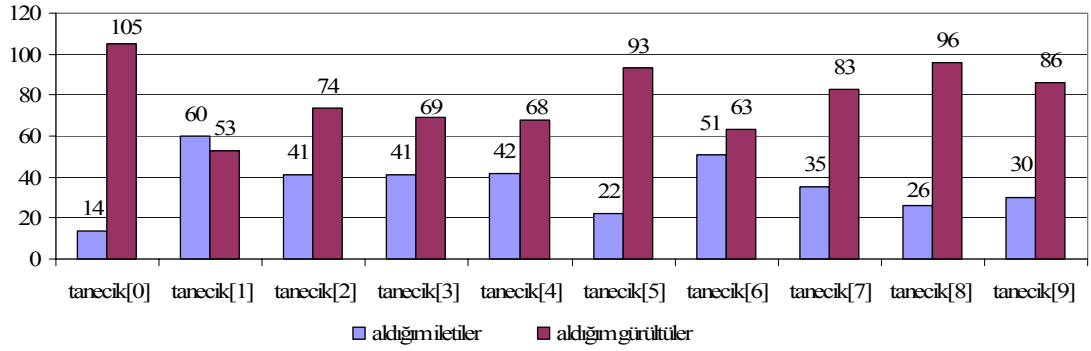
Şekil 4.63: [Alınan iletiler]=[sağlam]+[bozuk]+[çarpışmış]



Şekil 4.64: [Alınan iletiler]=[benim]+[benim değil]+[ACK]+[çarpışmış]



Şekil 4.65: [Alınan iletiler]=[benim]+[benim değil]+[ACK]+[çarpışmış]



Şekil 4.66: Alınan ileti ve gürültüler için karşılaştırma grafiği

5. KABLOSUZ ALGILAYICI ĞLARININ OMNeT++ İLE BENZETİMİ

Bölüm 5 için [22] ve [23] numaralı kaynaklardan yararlanılmıştır.

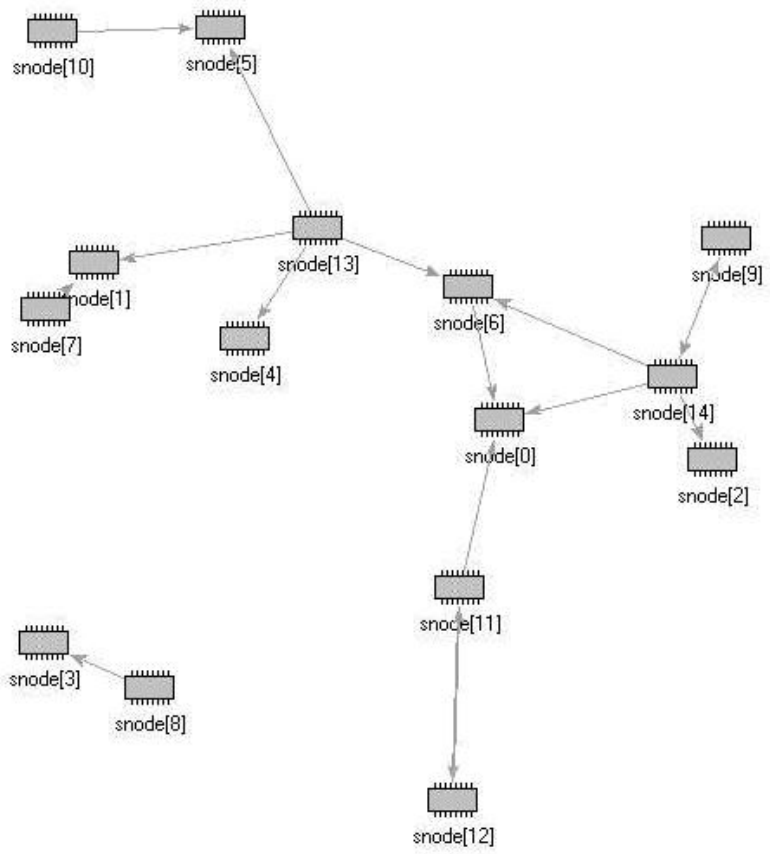
5.1 Kablosuz Algılayıcı Ağlar İçin Bir Benzetim Şablonu

Kullanıcı kendi protokolü için kod yazmak ve onu şablon içerisine entegre etmek zorundadır (kodun yerleştirileceği modülleri oluşturan araçlar vardır). Daha sonra kullanıcı genel ağ parametrelerini tanımlamalı ve simülasyonu başlatmalıdır. Ağ parametrelerinin tamamı herhangi bir kodu yeniden yazmayı gerektirmeden değiştirilebilir. Kod'un ana fonksiyonelitesine erişim için kullanımı kolay makrolar bulunmaktadır.

- Hareketlilik, varsayılan olarak Random Way Point algoritması ile gerçekleştirilir. Her düğüm kendi yörüngesini tanımlamak ve onu simülatöre bildirmek ile yükümlüdür;
- düğümler kablosuz haberleşmeyi kullanarak mesajları değiş tokuş ederler. Bir mesaj, iletim kapsamı içindeki tüm komşuları tarafından işitilir (iletim alanı içindeki modüller birbirlerine otomatik olarak bağlanırlar);
- kullanıcı, kullanılması zorunlu olan tek yönlü ya da çift yönlü bağlantıları belirleyebilir. Her bir düğüm kendi iletim kapsamını, bağımsız bir şekilde, belirleyebilir ve güncelleyebilir;
- bazı temel enerji yönetimi uygulamaları ayrıca dahil edilmiştir;
- düğümler farklı hata olasılıklarına sahiptirler (bireysel hatalar, harita içindeki bölgeye etki eden hatalar, vb.). Alan hataları için haritalar tayin edilebilir ve kullanılabilir. Diğer haritalar engel, sönümlenme, vb. için kullanılabilir.

Tüm bu sıralanan faydaları [16] yerine getirmek için, merkezi bir yönetici seçilmelidir, yönetici bağlanabilirlik durumunu muhafaza eder ve istek üzerine bunu günceller. Aynı zamanda ilave konfigürasyon dosyalarını okumaya ve bilgileri her

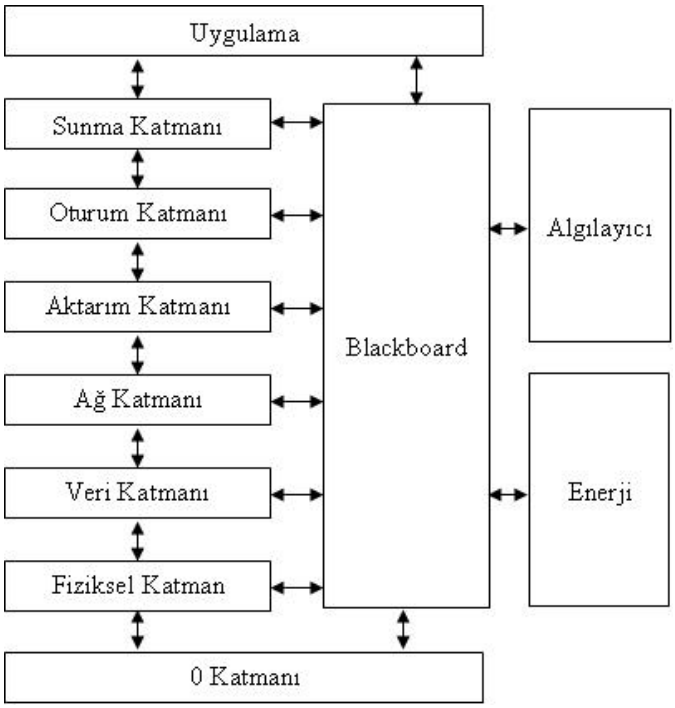
bir düğüme göndermeye çalışır. Yönetici, görüntüden ve düğümler arasındaki tüm haberleşmeden ve kullanıcıdan sakınır. Bu sayede, grafik görüntü ile yanlızca kullanıcının ilgilendiği ağ temsil edilmiş olur (Şekil 5.1).



Şekil 5.1: Benzetim Örneği

Herbir algılayıcı düğüm Şekil 5.2'de gösterilmiş örneğe benzer bir yapıya sahiptir. OSI ağ katmanları yanlızca bir örnek olarak gösterilmektedir. Kullanıcı onlardan herhangi birini ilave ederek ya da çıkartarak onları modifiye edebilir. Herbir algılayıcı içerisindeki dahili modüller, *blackboard* olarak isimlendirilen modülü kullanarak bilgiyi değiştirebilir.

Burada "0 Katmanı" olarak isimlendirilen başka bir modül vardır. İki algılayıcı düğüm arasında ve aynı zamanda herbir algılayıcı düğüm ila merkezi yönetici arasında bir arabirim görevi görür, özel ağ katmanlarının tüm foknsiyonelliğini sağlar. Daha düşük (protokol katmanında) bir özel katman iletim mesafesi içindeki komşularına mesaj göndermek istediği zaman, mesajı yanlızca "0 Katmanı"na iletir.



Şekil 5.2: Algılayıcı Düğüm Betimlemesi

5.2 Kablosuz Algılayıcı Ağlarını OMNeT++ ile Simüle Etmek

5.2.1 OMNeT++ kafesi

OMNeT++ (Objective Modular Network Test-bed in C++) açık-kaynaklı, bileşen-tabanlı, modüler bir benzetim iskeleti(kafesi)dir. Haberleşme ağlarını ve diğer dağıtık sistemlerin benzetimini yapmak için kullanılır. Şekil 5.2'de gösterilmiş olan OMNeT++ modeli hiyerarşik bir düzende yuvalanmış modüllerin bir derlemesidir. Enüst modül aynı zamanda System Module ya da Network olarak da isimlendirilir. Bu modül bir veya daha fazla alt modül içerir, bu alt modüller de kendi alt modüllerini içerirler.

Modüller birkaç derinlikte iç içe olabilirler ve bu sebeple OMNeT++ içindeki karmaşık sistem modellerini yakalamak mümkündür. Modüller ya yalın ya da bileşik varlık olarak ayırt edilebilir. Yalın bir modül, algoritmalar ile tanımlanmış istenilen davranışları sağlayan bir C++ dosyası ile ilişkilendirilir. Bileşik olmayan modüller, modül hiyerarşisinin en düşük düzeyini teşkil ederler. Kullanıcılar, OMNeT++

benzetim sınıf kütüphanelerini kullanarak yalın modüller gerçekleştirebilirler. Bileşik modüller, yalın modüllerin bir araya gelmesinden oluşur ve hareketleri sağlayan C++ dosyası ile doğrudan ilişkilendirilmemişlerdir.

Modüller mesajları değiştirerek haberleşirler. Herbir mesaj karmaşık veri yapısında olabilir. Mesajlar, doğrudan ya da bir dizi kapı (gate) ve bağlantılar aracılığı ile, yalın modüller arasında (kendilerine ait ve benzeri olmayan ID'ye dayanarak) değiş tokuş edilebilir. Mesajlar, bir bilgisayar ağındaki iletileri (frames) ve paketleri temsil ederler. Lokal simülasyon zamanı, bir modül başka bir modülden ya da kendisinden mesaj aldığı zaman, ilerler. Sonraki olayları zamanlamak için, modül kendi mesajlarını kullanır. Modüllerin arabirimi ve yapısı, ağ tanımlama dili kullanılarak belirlenir. Onlar yalın modüllerin temel davranışlarını gerçekleştirirler. Simülasyonun yürütmesi başlangıç dosyaları ile kolayca konfigüre edilebilir. Mesajlar doğru modüllere vaktinde ulaştırılsın diye olayları izlerler.

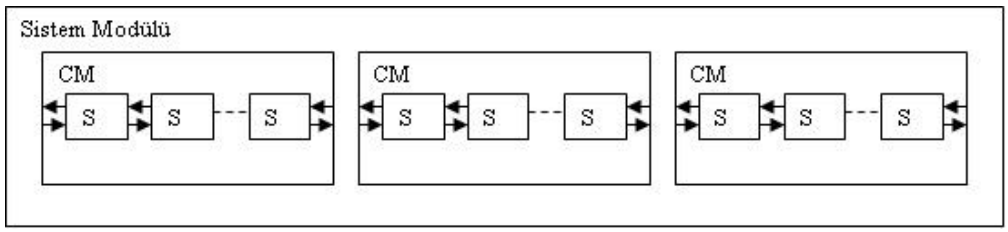
Algılayıcı Ağ Simülasyonu için, yukarıdaki avantajları sağlayacak OMNeT++'in iskelet (kafes) yapısı tercih edilebilir. Faydaları aşağıdaki gibidir:

- OMNeT++ modüler (birimsel) benzetim modellerinin tasarımına imkan verir, bu modeller birleştirilebilir ve esnek bir şekilde yeniden kullanılabilirler.
- Modelleri sıradüzen ile birleştirmek mümkündür
- OMNeT++'in nesne-tabanlı yaklaşımı, simülasyon çekirdeği içinde sağlanan, temel sınıf kütüphanelerinin esnek bir şekilde genişletilmesine imkan verir.
- Model bileşenleri derlenebilir. Simülasyon kütüphanesi ile kullanıcı arabirim kütüphaneleri yürütülebilir bir programı biçimlendirmek için birbirlerine bağlanabilir. Biri kullanıcı arabirim kütüphanesini bir diğeri simülasyonu izleme ve hata ayıklama için kullanılabilen grafik kullanıcı arabirimini çalıştırıyorken, komut satırı ve topluca yürütme aynı zamanda kullanılabilir.
- OMNeT++: giriş/çıkış, istatistik, veri toplama, simülasyon verisinin grafiksel sunuşu, rastlantısal sayı üreteçleri ve veri yapıları için destek içeren yaygın bir benzetim kütüphanesi sunmaktadır.
- OMNeT++, benzetim çekirdeği daha geniş uygulamalar içerisine gömülebilme özelliğine sahip, C++ kullanır.

- OMNeT++ modelleri NED ve omnetpp.ini ile inşa edilir ve konfigüre edilmiş olan çeşitli simülasyonlar için script kullanmaz.

Takip eden bölümde OMNeT++ üzerindeki simülasyon senaryosunun detaylı uygulaması verilmiştir.

5.2.2 Simülasyonun tasarımı

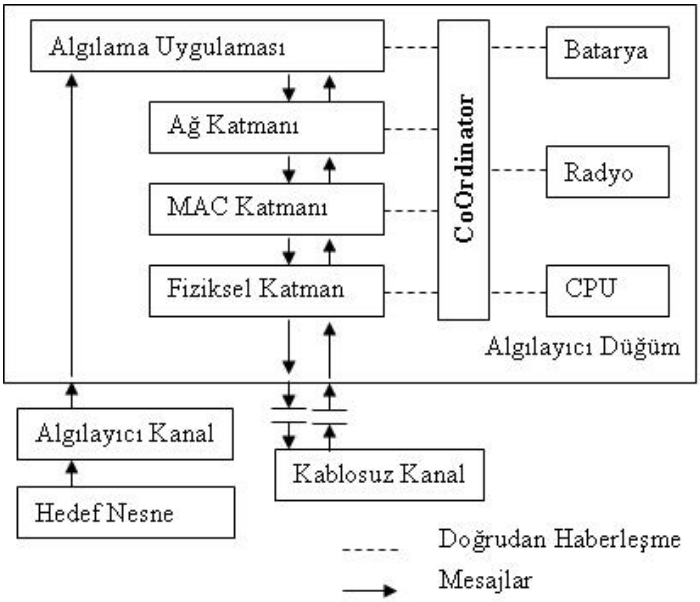


- Simple Modüller arasındaki Compund (bileşik) Modül Mesajları

Şekil 5.3: OMNeT++ içindeki yalın ve bileşik modüller

Bu bölüm bir algılayıcı düğümün yapısını ve simülatörün tüm tasarımını sunmaktadır. Simülasyonumuzdaki Algılayıcı Ağ sahasının topolojisi, OMNeT++ iskeleti(kafesi)nin yalın ve bileşik modül kavramından elde edilmiştir. Şekil 5.3'de görüldüğü gibi, bir düğümün katmanları yalın modüller gibi ve bir algılayıcı düğüm ise bileşik bir modül gibi davranır ve tüm bu algılayıcı düğümler, sistem modülü olarak tanımlanan, algılayıcı ağı'nı oluştururlar.

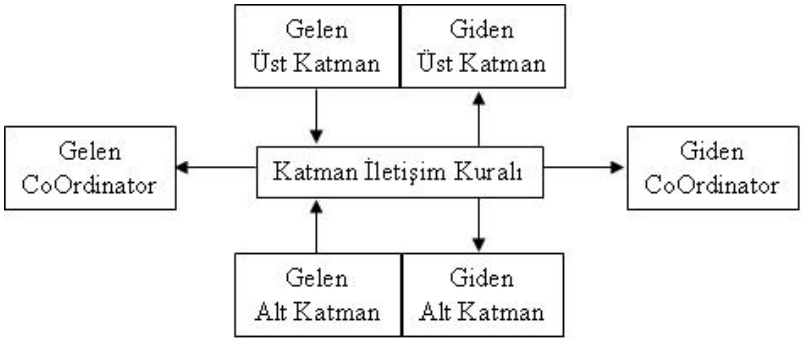
Algılayıcı düğümün her bir katmanı OMNeT++'ın yalın bir modülü olarak temsil edilir. Katmanlar kapılar üzerinden birbirleri ile haberleşirler ve katmanların herbiri CoOrdinator'e ilişkin bir referansa sahiptir. Bir katmanın yapısı Şekil 5.4'deki gibi temsil edilir. Yalın modüller bir algılayıcı düğümün katmanlı mimarisine göre birleştirilirler. Algılayıcı düğümün farklı katmanları, algılayıcı düğüm yığınını düzenlemek için, algılayıcı düğümün diğer katmanlarına yönelik kapılara sahiptir. Kablosuz kanal fonksiyonelitesine sahip bir yalın modül, bileşik modüller (algılayıcı düğümler) ile, kapılar içerisinden haberleşmek için, kullanılır. Herbir modül tarafından sağlanan fonksiyonellik, algılayıcı düğümün donanım modülünü oluşturan Radyo, CPU ve Batarya Modülü olarak aşağıda tanımlanmıştır.



Şekil 5.4: Algılayıcı Düğümün Basit Yapısı

5.2.2.1 CoOrdinator modülü

CoOrdinator modül, algılayıcı düğümün donanım ve yazılım modüllerinin aktivitelerini koordine edebilecek fonksiyonlara sahiptir. Temelde katmanların birbirleriyle haberleşmesi için kullanılır. CoOrdinator genişletilmiş olmaya ihtiyaç duyar ve ilave tüketicilerin ya da yeni donanımların özelliklerine erişim için fonksiyonellik eklenmesi zorunludur. Şekil 5.5'de gösterildiği gibi, CoOrdinator sınıfı algılayıcı düğümdeki tüm katmanlara yönelik referansa sahiptir ve algılayıcı düğümdeki tüm katmanlar CoOrdinator sınıf uygulamasına erişebilirler. CoOrdinator içerisindeki herhangi bir katman diğer katmanın özelliklerine erişim ve güncelleme yapabilir. Örneğin Battery Module, gönderilen ve alınan paketler hakkında, Physical Module tarafından bilgilendirilmiş olmayı gerektirir, böylece enerji tüketimi düğümde ona göre güncellenir. Simülasyon boyunca CoOrdinator sınıfı, algılayıcı düğümün algılayıcı ağ'a kayıt olmasından sorumludur. Algılayıcı düğümün kayıt olması, algılayıcı düğümün faal ve çalışır durumda olduğuna dair bir belirtidir. Kullanılabilir enerji tamamen tüketildiği zaman, düğümün algılayıcı ağ'dan kaydı silinir.



Şekil 5.5: Algılayıcı Düğüm içindeki bir katmanın gösterimi

İki katman arasındaki bağlantı kapılar, haberleşme ise mesajlar aracılığı ile yapılır. Bunlar her katmanın diğer katmanlara ve CoOrdinator ile olan bağlantı çeşitleridir.

5.2.2.2 Donanım modeli

Batarya Modeli : Düğümün önemli bir bileşenidir; CPU Modül, Radyo Modül ve ortamı algılamak için kullanılan algılayıcılara gerekli enerjiyi sağlar. Bu nedenle Batarya düğüm içindeki tüm bileşenlere bağlanır ve enerji kaynağı tüm bileşenler tarafından çekilen güç miktarına bağlı olarak azalır. Düzenli aralıklarda, kullanılan batarya model türüne bağlı olarak, modül kendisinin kalan enerji miktarını günceller. Simülasyonda doğrusal batarya modeli, deşarj hızı bağımlı model gibi çeşitli modeller uygulanabiliyor. Tüm donanım aygıtları kendi güç tüketimlerini rapor ettikleri zaman, bataryanın o anki güç tüketimi ve dolayısı ile tahmini süre, bataryadan dayanması beklenilen T (saat cinsinden) süresi, aşağıdaki gibi tanımlanır:

$$\mathbf{T = C / I} \tag{5.1}$$

- C, bataryanın Amper-Saat olarak kalan kapasitesi
- I, algılayıcı düğüm tarafından Amper olarak çekilen toplam akım

Bataryada kalan kapasite, ya doğrusal bir model ya da deşarj hızı bağımlı bir model türüne bağlı olarak, tahmini olabilir. Doğrusal deşarj modelinde, kalan kapasite aşağıdaki formül ile hesaplanabilir:

$$C = C_{in} - \int_{\Delta t} I(t)\Delta t \tag{5.2}$$

- Cin, bataryanın başlangıç kapasitesi
- I(t), Δt süresinde algılayıcı düğüm tarafından çekilen akım.

Bu model öz-boşaltımlı değildir ve batarya kullanım yaşı itibari ile bozulmuş kabul edilir. Boşaltım hızı model türüne bağlı olup, daha yüksek deşarj hızı bataryanın kalan kapasitesini etkin bir şekilde küçültür. Uygulama türüne bağlı olarak çeşitli modelleri gerçekleştirebilmek için basit bir Batarya Modülü kullanılır; BatteryBase, farklı batarya modelleri için genel sınıfı teşkil eder. BatteryLinear, BatteryBase'in alt sınıfıdır ve tüketicilerin miktarına ve tüketici aktivitelerinin durumuna bağlı olarak enerjiyi günceller. BatteryDischargeRate, BatteryBase'in bir alt sınıfıdır ve enerj, tüketimi akımın doğrusal fonksiyonudur.

CPU Modeli : Bir algılayıcı ağ içerisindeki düğümler genellikle çok düşük uç işlemci ya da mikro denetleyiciler ile donatılırlar. Çeşitli işlemlere yönelik güç tüketimi düşük olmalıdır ve işlemci'nin enerji tüketimi için standart parametreler kümesi kullanılabilir. İşlemci Idle, Sleep ve Active çalışma durumlarında farklı seviyelerde enerji tüketimi yapar. İşlemcinin güç tüketim modeli çok önemlidir ve ihmal edilmesi ağ içindeki güç tüketiminde hatalı yönelmelere sebep olacaktır. Gelişmiş özelliklere ve geliştirilmiş enerji tüketim seviyelerine sahip yeni işlemci modelleri, uygulamaların çeşitli türlerini test etmek için bu modül içinde birleştirilebilir. CPUBase farklı CPU modelleri için temeli teşkil eder ve bu modülün CoOrdinator ve Batarya arabirimlerini tanımlar. CPUSimple, CPU'nun farklı durumlardaki (Idle, Sleep ve Active) güç tüketimi uygulamasına sahiptir.

Radyo Modeli : Bu model bir düğümün anten özelliğini karakterize etmek (tanımlamak) için kullanılır. RadioBase farklı Radio modelleri için genel bir sınıftır. RadioSimple, RadioBase'in bir alt sınıfı olup, Radio'nun Idle, Sleep, Transmit ve Receive durumlarına bağlı olarak bataryanın enerjisini günceller. Donanım ve tüketicilerin farklı özellikleri için gerekli olan değerler, konfigürasyon dosyası vasıtası ile sağlanabilir.

5.2.2.3 Kablosuz kanal modeli

Algılayıcı düğümler arasındaki tüm muhtemel bağlantıları kontrol eder ve bakımını yapar. Bu statik (durgun) bağlantılar, tüm düğümlerden Wireless Channel Module'e doğru ve modülden tüm düğümlere doğru olacak şekilde, NED dosyası içerisinde sağlanır. Bu bağlantılar algılayıcı düğümlerin veriyi değiş tokuş etmesine ve birbirleri ile haberleşmesine imkan verir. Her bir mesaj, bir düğümden aktarım bölgesindeki tüm komşularına "d" gecikmesi ile gönderilir; burada

d = (Haberleşen düğümler arasındaki mesafe) / (Işık hızı)'dır. (5.3)

Çeşitli Radio Propagation modelleri herbir paketin alınmış işaret gücünü öngörmek için kullanılır. Bu modeller 2 düğüm arasındaki haberleşme bölgesini etkiler ve Wireless Channel tarafından türetilir.

Boşlukta Yayılım Modeli: Boşlukta yayılım modeli, ideal yayılım durumlarının var olduğunu kabul eder ki, verici ve alıcı arasında yanlızca bir açık görüş hattı yolu vardır. Vericiden uzakta boşlukta alınmış işaretin gücü tahmini değer olarak aşağıdaki formül ile hesaplanabilir:

$$P_r = \left(P_t * G_t * G_r * \lambda^2\right) \Big/ (4\pi)^2 * d^2 * L^2 \qquad (5.4)$$

- Pt, iletilmiş işaretin gücü
- Pr, alınmış işaretin gücü
- Gt, Gr sırası ile vericinin ve alıcının anten kazançları
- L, sistem kaybı ve dalga boyudur.

İki-Işınlı Toprak Yansıma Modeli: İki gezgin düğüm arasındaki tek bir görüş hattı yolu nadiren yanlızca yayılım anlamına gelir. İki ışınlı toprak yansıma modeli hem doğrudan yolu hem de bir toprak yansıma yolunu dikkate alır. Bu model, uzun mesafede boşluk modelinden daha doğru sonuç verir. Mesafeye bağlı alınmış güç, tahmini değer olarak, aşağıdaki formül ile hesaplanabilir:

$$P_r = \left(P_t * G_t * G_r * h_t^2 * h_r^2\right) \Big/ \left(d^4 * L\right) \tag{5.5}$$

- ht ve hr, sırası ile verici ve alıcı antenlerin yükseklikleri

Yukarıdaki denklem, mesafe artışı olarak Free Space Model'den daha hızlı bir güç kaybını göstermektedir.

5.2.2.4 Algılayıcı düğüm demeti

Algılayıcı düğüm hiyerarşisinin enüst düzeyindeki, AppLayerSimple olarak isimlendirilen, yalın modül Application Layer'ın davranışını simüle eder. Bu modül, NetLayerBase Module ile her bir mesajı zamanlamak için, kapılar üzerinden haberleşir. Yeni uygulamalar bu modüle dahil edilebilir.

Simple Network Module ağdaki düğümler tarafından gönderilmiş ve alınmış paketleri simüle eder. Network Module başlangıçta Application Layer mesajlarını AppLayer Module'den alır ve onlara Network Header ekler. Bu katmanın belirli özellikleri protokol uygulamasına bağlıdır. MAC katmanına çevirilmiş Network Layer'ın paket yapısı yönelendirme bakımından sonraki sıçrama noktasına sahiptir. MAC katmanı, Physical Layer ve Routing Layer arasında bir arabirim sağlar. Media Access'in temel fonksiyonelliğine sahiptir ve bu modülün işlevselliği 802.11b uygulaması için daha büyük ayrıntı tanımlar.

5.3 OMNeT++ Benzetim Modellemesi İşlem Basamakları

- OMNET uygulamalarının modelleme felsefesine göre, bir sistem, mesaj değiş-tokuşu yaparak haberleşen modül denilen bir nesneden meydana gelir; modüller iç içe yuvalanabilirler, yani bazı modüller daha büyük bir birim (bir birleşik modül) teşkil etmek için, birlikte gruplanabilirler. Haberleşebilen modüller içerisine kendi sistemimiz uyarlanabilir. Kendi modelimizin içindeki bileşenler ve onların haberleşme yolları tanımlanmalıdır. Bileşenler içerisindeki alt bileşenler de

tanımlanmalıdır. Bileşik Modülleri (CM) tasarlamak için OMNeT++ in grafik düzenleyicisi (editörü) olan GNED kullanılabilir. /<path_to_gned/gned

- Model yapısını tanımlamak için, OMNeT++'in Topoloji Tanımlama Dili kullanılabilir.
- C++ kullanarak modelin aktif bileşenleri eşzamanlı işlemler olarak tanımlanmalıdır. OMNeT++ sınıf kütüphaneleri kullanılabilir. OMNeT++ aynı zamanda yeniden kullanılabilir model kütüphanelerini de destekler.
- Elverişli bir konfigürasyon dosyası sağlanmalıdır. Konfigürasyon dosyası OMNeT++ ve kendi modelimize yönelik bazı seçenekler içerebilir. Konfigürasyon dosyası, farklı parametreleri olan bazı simülasyon yürütmelerini tanımlayabilir. Rastlantısal sayı üreteçleri için kaynakları dikkatlice seçmeliyiz. OMNeT++ in kaynak aracı yapılan bu işe de yardım edebilir.
- Benzetim programını inşa etmeli ve çalıştırmalıyız. Yazdığımız kodu OMNeT++ benzetim çekirdeği ve OMNeT++'in sağladığı kullanıcı arabirimlerinden biri ile birbirlerine bağlayacağız. Komut satırı (batch) biçiminde ve etkileşimli, grafiksel kullanıcı arabirimleri vardır. Konfigürasyon dosyasında önceden tanımlanmış tüm akış otomatik olarak yürütülecektir.
- Simülasyon sonuçları "vector" ve "scalar" çıktı dosyalarına yazılır. Çıktıları göz önüne getirmek için Plove ve Scalars kullanılabilir. Daha kusursuz istatistiksel analiz için, R, Octave ya da Matlab gibi bağımsız paketler kullanılabilir, hatta OpenOffice hesap makinesi ya da Gnumeric gibi hesap çizelgeleri bile kullanılabilir.

/<path_to_plove/plove , /<path_to_scalars/scalars

6. BENZETİM SONUÇLARININ DEĞERLENDİRİLMESİ VE GELECEK İÇİN ÇÖZÜM ÖNERİLERİ

Bu bölümde yapılan öneriler uygulanmamış ve test edilmemiş olup, benzetim sonuçlarına dayanılarak yapılan önerilerin iletişim kurallarının iyileştirilmesi yönünde atılacak sonraki adımlara yardımcı olacağı düşünülmektedir. Önerilerin aynı benzetim ortamında simüle edilmesi ve doğruluklarının ispatı yapılacak çalışma olarak hedeflenmiştir. Buradan elde edilecek sonuçlar, gerçek ortam şartlarında kablosuz algılayıcı ağları uygulanması için, esas alınacaktır.

6.1 Taşma İletişim Kuralı ile İlgili Sonuç ve Öneriler

Taşma protokolü, gerek taşma gerekse örtüşme nedeni ile, bir mesajı aynı düğüme birden fazla kez gönderiyorken SPIN protokolü bunu önlemek amacı ile düğümleri müzakere yaptırarak fazladan iletişim yapmalarına neden oluyor. Bu onların kanalı fazladan meşgul etmeleri, diğer düğümlerin gereksiz uyarılmaları ve fazladan enerji harcamaları anlamına gelmeketdir. Yine Taşma protokolü enerji kontrolü yapmazken SPIN protokolü enerji kontrolü yapıyor fakat biten enerjiye yönelik çözüm getirmemektedir.

Kablosuz algılayıcıların luzumsuz hareket ya da aktivasyon içerisinde olmamaları gerekir. Bir algılayıcı ya da tanecik iki önemli görev üstlenmiştir. Birincisi ölçme yapmak, ikincisi ise yönlendirme yapmak. Her iki durumda da bir bilginin iletimi söz konusudur. Ölçülen bilgi kadar iletilen bilgi de önemlidir ve hedefe ulaştırılmalıdır, bu amaçla yayınlanmalıdır. Ne varki iletimde gereksiz tekrarlar ya da müzakereler de yaşanmaktadır. Bu bilgiyi sana göndereyim mi? Müzakeresinden daha kolay çözümler geliştirilebilir. Enerjim bitiyor mu? Sorusunun cevabı düğüm için ne anlama gelecektir? Enerjisinin bitiyor olmasını diğer düğümler ve ana istasyonda bilmek isteyebilir, hatta bilmesi iletişimin sağlıklı sürdürülmesine yardımcı olabilir. Enerjinin belirli seviyelerde kontrol edilip davranışların buna göre şekilllendirilmesi

ve tanecik üzerinde kalan enerjiye göre yaşam süresinin uzatılması mümkün olabilir. Bu durum bir tanecik'e akıllı (enerji denetimli) bir haberleşme stratejisi kazandıracak ve enerjisinin bitmekte olduğunu bilmekten daha fazlasını ona kazandırmış olacaktır.

Tanecik (kablosuz algılayıcı) haberleşme alogritmasında veri mutlaka iletilmeli ilkesi olmalıdır, ama aynı algoritma, iletim kararını vermeden önce kendi içinde yapacağı bir kaç denetleme ile hem kendisi, hem diğer düğümler ve hem de ortam için, daha effektif bir haberleşme yapabilir. Herşeyden önce paket başlık dosyasında bir kimlik bilgisi olmalıdır, bu hane "MoteID+MsgNO" birleşiminden oluşmalıdır, yani gönderim yapacak düğüm kimlik bilgisi hanesine ilk olarak kendi kimliğini ve ilaveten göndereceği mesajın sıra numarasını yazmalıdır. Bu sayede mesajı alan düğüm bu mesajın kimden geldiğini "Mote ID"si ile bilecek ve mesaj numarası ile de gönderenden aldığı son mesajı karşılaştırmak sureti ile bu mesajın kendisine daha evvel gönderilip gönderilmediğini öğrenmiş olacaktır, yani gönderen düğüm ile müzakere yapmadan kontolünü yapmış olacaktır, müzakere hem gönderen hem de alan için fazladan iletim demektir.

Değişmez bir durum var ki, o da düğümlerin yayın (broadcast) yapmak durumunda olmalarıdır, çünki kablosuz dağıtık (ad-hoc) bir ağ'da herhangi bir düğümün mevkisi sabit değildir, bu nedenle sabit bir yönlendirme algoritması uygulanamaz. Buna karşılık gönderme her zaman ve her durumda yapılmamalı fakat alma her durumda yapılabilmelidir. Gönderme (yayın) yaparken tanecikler daha titiz hatta tutucu davranmalı alınmış her mesaj için, her durumda ve her zaman, gönderme yapılmamalıdır.

Kablosuz algılayıcı ne kadar az enerji harcar ise o kadar uzun süre kullanılabilir, kullanım amaç ve yerine göre bu durum önem arzedebilir. Kablosuz algılayıcı ne kadar az gönderme yapar ise enerji tüketimi okadar az olacaktır. İletim algoritmaları gönderme işlemine karşı daha denetimli olmalı, gelen paket incelenmeli, denetlenmeli, muhteviyatına ve hareket tarzına bakılmalı, ortam ve komşu düğümler gözlendikten sonra gönderme yapılmalıdır. Tanecik'lerin üzerinde çalışıtırılacak ve bu durumu sağlayacak olan kodlar uygun iletişim kuralını çalıştırırlarsa haberleşmede verim artışı sağlanabilir.

Enerji tüketimi zaten farklı çalışma durumları için azaltılmak istenmiştir, örneğin bir tanecik uyku modunda mikro amper seviyesinde güç tüketmektedir, burada bilinmesi zor ama önemli bir etken vardır: bir tanecik ne kadar sıklıkla uyku moduna geçmeli ve ne kadar süre için uyku modunda kalmalıdır? Belki enerji tüketiminin kontrolü ve düzenlenmesi için, çalışma modlarına ilaveten çalışma seviyelerini de belirlemek ve bunu kodlamak gereklidir: birinci seviyede batarya %100-%40 arası seviyede olup, ölçme, ölçtüğü değeri yayma ve yönlendirme yapabilmesi; ikinci seviye de ise batarya seviyesi %40-%10 arası olup, ölçme ve ölçtüğü değeri yayma yapabiliyorken, yönlendirme yapamıyor olması; üçüncü seviye bataryanın kritik limite (%10-%1) inmesi demek olup, ölçme ve yönlendirme yapılamazken, açık aralıklarla kimlik bilgisini de içeren "batarya düşük uyarısı" mesajını yayması şeklinde seviye uygulamaları belirlenebilir. Bu sayede enerjinin kontrollü harcanması ile birlikte azalması durumunda tüketimin yavaşlatılması ve tükenmeye yakın enerji ihtiyacını haber vermesi gibi özellikler kazanılmış olacaktır. Bu özellikler sistemin performansını, verimini ve güvenilirliğini arttıracak etkenlerdir.

6.2 IEEE 802.11 İletişim Kuralı ile İlgili Sonuç ve Öneriler

Çarpışmaları önlemek amacı ile sıra tahsisli veri aktarımı; SenderNO: Bir istasyon iletmek istediği bir paket için, ilk olarak RTS olarak isimlendirilen ve kaynak, hedef ve takip eden işlemin süresini içerecek olan, kısa bir kontrol paketi gönderir, hedef istasyon (eğer ortam serbes ise) yanıt olarak aynı süre bilgisini içeren CTS kontrol paketi ile cevap verir. Ortam serbes değil ise, meşgul olarak set edilmiş ise, hedef CTS gönderemeyeceği için transfer başlayamaz, kaynak istasyon bu isteğini ikinci veya üçüncü bir RTS paketi göndererek tekrar etmek durumunda kalır. Bu tıkanıklığı çözmek için host'lardan birinin kontrolör görevi yapması düşünülebilir ya da kaynak istasyon RTS göndermesi ile (RTS paketi içerisinde) SenderNO bilgisini de diğer istasyonlara yayar.

SenderNO kaynak istasyonlar için gönderici sıra numarası demek olup, kaynak istasyon bir kere RTS göndermesi yaptıktan sonra veri transferi için sıranın kendisine gelmesini beklemelidir. İstasyonlar SenderNO sayaç bilgilerini güncel tutmalı,

gönderme yapmak istedikleri zaman enson SenderNO değerini (sayacını) arttırmak sureti ile kendilerine yeni bir sıra numarası tahsis etmeli ve bunu diğer istasyonlara duyurmalılar (ilan etmeliler).

Kaynak istasyon RTS paketi gönderdiği zaman diğer istasyonlar bunu birbilerine aktarırlar ve bu sayede SenderNO bilgilerini güncel tutabilirler, yani gönderme yapmak isteyen ve bunu ilan eden kaynak düğüme ait SenderNO bilgisi ağdaki tüm düğümler tarafından aynı değer olarak set edilir, kaynak düğüm hariç diğer düğümler için bu bilgi başka bir düğümün gönderme yapacağı, kanalın meşgul olacağı anlamını taşırken, kaynak düğüm için gönderici sıra numarası ve takip edilmesi gereken işlem sırası anlamını taşımaktadır.

Tüm istasyonlar ortam meşgul süresini takip ettikleri NAV değerinin yanında ortamı kullanma yani veri gönderme sırasını da takip etmeliler; eğer ortam meşgul ise ve/veya aktif olan SenderNo kendisine ait değil ise ortama yayma yapamaz. Gönderme yapmak için bekleyen bir istasyon aynı anda ölçme yapabilir, ölçüm işlemine bu amaç için ayrılmış bellek kapasitesini doldurana kadar devam edebilir ve ölçüm değerlerini buraya yazabilir, gönderme ile birlikte belleğini temizler.

Kaynak istasyon gönderici sıra numarası (SenderNO) kendisine geldiğinde veri transferine başlar, veri ağ üzerinde düğümler üzerinden aktarma yapılarak hedefe iletilir. Verinin iletimi tamamlandığında kaynak düğüm hedef düğüme bu durumu bildirmek için ACK paketi gönderir ve hedef alma işleminin yani transferin tamamlandığını duyurmak için ağdaki tüm düğümlere CTS paketi gönderir. Ağdaki düğümler CTS ile birlikte sayaç değerlerini bir arttırırlar, böylece tüm düğümlerde yeni SenderNo değeri aktif olur ve sıradaki kaynak düğüm veri iletişimine başlayabilir, yani CTS ile birlikte yeni SenderNO değerine geçiş olur ve veri transfer işlemi bir sonraki kaynak istasyona geçer. Yeni bir RTS gönderimi olmamışsa, CTS alan düğümler ortam bilgisini meşgul durumdan normal (elverişli) duruma set ederler, bu düğümler için RTS gönderimi yapılabilir anlamı taşımaktadır.

Normal 802.11 uygulamasında çapışma olmamış ve işaret gürültüye maruz kalmamış ise, alma ve gönderme STA'lar arasında sağlıklı bir şekilde gerçekleşmiş ve veri

hedef STA tarafından doğru olarak alınmış demektir. SenderNo ile kazanılacak sıra tahsisli aktarım sayesinde çarpışmanın büyük oranlarda önleneceği düşünülmektedir. Geliştirilmek istenilen iletişim kuralında bu husus uygulanmaya çalışılacaktır.

Bozulmaları önlemek için veri kodlama algoritmaları kullanılabilir: Bir radyo link ile ilgili yüksek Bit Hata Oranı nedeniyle, paketin bozulma olasılığı paket boyutuna dayalı olarak artar. Paketlerde çarpışmadan ya da gürültüden dolayı da bozulmalar meydana gelebilir. Aynı anda yapılan iletimler işaretlerin çarpışmasına neden olmaktadır, bu en sık rastlanan ve iletileri bozan durumdur. İletim esnasında bozulan işaretler hedef istasyon tarafından bozuk/tanımsız olarak değerlendirilmekte ve işlem yapılmamaktadır. Verinin gürültülere karşı korunması amacı ile kodlanması üzerine mevcut algoritmalar kullanılabilir. Kodlanma ile veri hem küçülmüş hem de bir miktar kaybı önleyecek şekilde korunmuş olacaktır. Bu amaçla kablosuz algılayıcı ağında yer alan tanecikler (host:düğüm) üzerinde sadece kodlayıcı olması yeterlidir, böylece sınırlı bellek alanı kullanımında daha az yer kullanılmış olacaktır.

Kod çözücü algoritmalar veri işleme istasyonlarında olması yeterlidir. Kod çözücü algoritmalar verideki küçük kayıpları çözebilecek yeteneklere sahip olmalıdır. Sonuç olarak veri transferinde kazanılacak verim ile Backoff (yeniden çağırım, yeniden alma-verme isteği) uygulamasının çalıştırılması azalacaktır, bu taneciklerin daha az enerji harcaması, ortamın daha az meşgul edilmesi demektir.

Haberleşme trafiğinin düzenlenmesi ve kontrolü için, gönderme yapmak isteyen bir düğüm RTS ile bu isteğini ilan eder, RTS küçük boyutlu bir duyuru paketi olup her bir düğüm tarafından alınmalı ve düğümler SenderNO sayaçları ile bu talebi kayıt etmeliler, aynı şekilde talep yapan düğüm de sayacını set eder ve gönderme yapmak için sırasının gelmesini bekler. Ağ içerisinde kaynak ve hedef düğümler arasındaki iletişimin tamamlanması için kaynak düğüm verinin tamamlandığını ACK paketi ile hedefe bildirir, hedef alımı tamamladığında CTS paketi ile tüm düğümlere iletimin tamamlandığını duyurur. CTS ile birlikte SenderNO sayacı ilerletilir ve sırası gelen düğüm veri transferine başlar. Bu işlem sıralı olarak diğer gönderme yapmak isteyen düğümler için de tekrar eder. Netice de sıra esasına dayalı veri iletimi ile ağ trafiği azaltılmış, çarpışmalar önlenmiş ve kaynak tasarrufu sağlanmış olur.

RTS ve CTS paketlerinin tüm düğümlere iletilmesi gerekirken, verinin hedefe ulaştırılması için yapılacak aktarımlara yönelik olarak düğümler gerekli/gereksiz kontrolü yapabilmelidir. Bu amaçla düğümler komşularına ait kimlik bilgilerini tutacakları bir NeighborMap'e sahip olmalılar. Ağ üzerindeki bir düğüm gelen sinyalin (paketin) enerji seviyesine bağlı olarak, kaynak düğümün kendisine olan mesafesi hakkında fikir sahibi olabilir ve kaynak düğümü NeighborMap tablosuna kayıt edebilir. Düğüm gönderilen veri için hedef konumunda değil ise, gelen verinin transferi yapacak ise, bunu yapıp yapmaması gerekliliği konusunda şu karşılaştırmaları yapmalıdır: Veriyi gönderen düğüm NeighborMap'de kayıtlı bir yakın komşu ise veri transferi yapılmayabilir, çümkü paket NeighborMap'de kayıtlı başka bir komşu tarafından alınabilir ve aktarılabilir. Gönderen uzak komşu ve alıcı yakın komşu ise veri transferi yine yapılmayabilir, çünkü gelen paketin yine alıcı olan yakın komşu tarafından alınabilme ihtimali vardır. Gönderen uzak komşu ve alıcı da uzak komşu ise, yani her ikisi de NeighborMap'de kayıtlı değil ise, gelen verinin transferi yapılmalıdır. Bu sayede MultiHop aktarım atlamalı olarak gerçekleştirilmiş, Hidden Terminal problemi çözülmüş, fazladan yapılan yayınlar önlenmiş ve enerji korunumu sağlanmış olacaktır.

RTS, CTS ve ACT paketlerinin aktarımında NeighborMap kontrolü yapılmamalıdır, bu bilgiler her durumda tüm düğümlere ulaştırılmalı, trafiğin sağlıklı olarak akışına yardımcı olunmalıdır.

Bir düğüm için ilan edilmesi gereken önemli durumlardan bir tanesi de RunLevel (çalışma seviyesidir), enerji kapasitesi %40'ın altına düşen bir düğüm 2nci seviyede çalıştığını ilan etmeli ve bu bilgi tüm düğümlere ulaştırılmalıdır, yine enerji kapasitesi %10'un altına düşen bir düğüm birinci seviyeye düştüğünü aynı şekilde tüm düğümlere duyurmalıdır. Bu hem düğümlerin enerji seviyelerinin kontrol edilebilmesini sağlayacak hem de veri aktarımında pozitif etki yapacaktır. Bir düğüm kendisine gelen verinin göndereni yakın komşusu ise ve bu komşunun run level'ı 2 ya da 3 ncü seviye ise mutlaka verinin aktarımını yapacaktır, kendisinden sonrakiler için bu gerekli değildir.

Şu durumda bir düğüm NeighborMap içerisinde hem komşu düğüm id'lerini hem de bunların enerji seviyelerini bilgi olarak tutmalı, bu bilgiyi gelen paketlerin içerik ve enerji seviyeleri ile güncellemelidir.

Yapılan bu çalışma ile elde edilen sonuçların katkısı olarak, kablosuz algılayıcıların haberleşmesine yönelik bir takım çözümler önerilmiştir. Bu öneriler bir sonraki çalışma programı olarak OMNeT++ üzerinde yazacağım yeni modüller ile simüle edilecek olup, kablosuz algılayıcıların temini ile gerçek ortam uygulaması da yapılabilecektir.

KAYNAKLAR

[1] Crossbow Technology Inc., *Motes, Smart Dust Sensors* [online], http://www.xbow.com/Products/,(**Ziyaret Tarihi: Mart 2005**).

[2] Department of EECS at the University of California at Berkeley, *Berkeley Wireless Research Center (BWRC)* [online], http://bwrc.eecs.berkeley.edu/ , (**Ziyaret Tarihi: Temmuz 2005**).

[3] Intel Corporation, *Intel® Mote research project* [online], http://www.intel.com/research/exploratory/motes.htm, (**Ziyaret Tarihi: Kasım 2005**).

[4] S.N. Simić and S. Sastry, "Distributed environmental monitoring using random sensor networks", ***Proc. 2nd Int. Workshop on Information Processing in Sensor Networks at the Palo Alto Research Center,CA***, 582-592, (April 2003).

[5] D. Ganesan, R. Govindan, S. Shenker and D. Estrin, "Highly resilient, energy efficient multipath routing in wireless sensor networks", vol. 1, no. 2, ***Mobile Computing and Communications Review***, 10-24, (2002).

[6] A. Mainwaring, J. Polastre, D. Culler, R. Szewczyk and J. Anderson, "Wireless sensor networks for habitat monitoring", ***Proc. First ACM International Workshop on Sensor Networks and Applications, Atlanta, GA***, 88-97, (September 2002).

[7] Intel Corporation, *Improving Life and Industry with Wireless Sensors* [online], http://www.intel.com/research/exploratory/wireless_promise.htm, (**Ziyaret Tarihi: Kasım 2005**).

[8] A.N. Knaian, "A wireless sensor network for smart roadbeds and intelligent transportation systems", M.Eng. Thesis, ***Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology***, (June 2000).

[9] W.W. Manges, G.O. Allgood, S.F. Smith, T.J. McIntyre, and M.R. Moore, *Intelligent wireless sensors for industrial manufacturing* [online], Oak Ridge National Laboratory, http://www.sensorsmag.com/articles/0400/44/main.shtml , (**Ziyaret Tarihi: Eylül 2005**).

[10] G. Kantor, S. Singh, R. Peterson, D. Rus, A. Das, V. Kumar, G. Pereira and J. Spletzer, "Distributed search and rescue with robot and sensor teams", ***Proc. 4th Int. Conf. on Field and Service Robotics, Lake Yamanaka, Japan***, (July 2003).

[11] UC Berkeley, *TinyOS is an open-source operating system designed for wireless embedded sensor networks* [online], http://today.cs.berkeley.edu/tos/, (**Ziyaret Tarihi: Temmuz 2005**).

[12] Jason Lester Hill, "System Architecture for Wireless Sensor Networks", ISBN: 0-496-52806-8, Source: DAI-B 64/09, ***Computer Science, University of California at Berkeley***, 4458, (Mar 2004)

[13] Sinha, A.; Chandrakasan, A., "Dynamic power management in wireless sensor networks, Design & Test of Computers", ***IEEE Volume 18, Issue 2***, 62-74, (March-April 2001*).*

[14] S.D. Servetto, "On the feasibility of large-scale wireless sensor networks," **Proc. 40th *Annual Allerton Conf. on Communication, Control and Computing, Urbana, IL*,** (October 2002).

[15] Leo Szumel and John D. Owens, "On the Feasibility of the UC Davis Metanet". ***Technical Report ECE-CE-2003-2, Computer Engineering Research Laboratory, University of California***, (2003).

[16] OMNeT++ Community Site, *OMNeT++ Discrete Event Simulation System* [online], http://www.omnetpp.org, (**Ziyaret Tarihi: Kasım 2005**).

[17] Witold Drytkiewicz, Steffen Sroka, Vlado Handziski, Andreas Köpke, Holger Karl, "A Mobility Framework for OMNeT++", ***Telecommunication Networks Group, Technische Universität Berlin***, (January 22, 2003).

[18] Marc Löbbers, Daniel Willkomm, *A Mobility Framework for OMNeT++ User Manual* [online], http://mobility-fw.sourceforge.net/, (**Ziyaret Tarihi: Aralık 2005**).

[19] Tutorial for OMNeT++ 3.2, *TicToc Tutorial for OMNeT++ 3.2* [online], http://www.omnetpp.org/doc/tictoc-tutorial/ , (**Ziyaret Tarihi: Aralık 2005**).

[20] Pablo Brenner, *A Technical Tutorial on the IEEE 802.11 Protocol* [online], BREEZECOM Wireless Comm., www.breeze.com, (**Ziyaret Tarihi: Ocak 2006**).

[21] Microsoft TechNet, *How 802.11 Wireless Works* [online], http://technet2.microsoft.com/WindowsServer/en/library/370b019f-711f-4d5a-8b1e-4289db0bcafd1033.mspx, (**Ziyaret Tarihi: Şubat 2006**).

[22] C. Mallanda, A. Suri, V. Kunchakarra, S.S. Iyengar, R. Kannan and A. Durresi, "Simulating Wireless Sensor Networks with OMNeT++", ***Submitted for publication to IEEE Computer***, (2005).

[23] Stefan Dulman, Paul Havinga, "A Simulation Template for Wireless Sensor Networks", ***Supplement of the The Sixth International Symposium on Autonomous Decentralized Systems, Pisa, Italy***, (April 2003).

İki düğümlü ağ benzetiminin kaynak kodları

```
// txc1.cc
// bu dosya Txc1 yalin modulunun fonksiyonlarini tanimlar
#include <string.h>
#include <omnetpp.h>
class Txc1 : public cSimpleModule
{
    // Bu bir makrodur; insa edenin tanimlamasini genisletir.
    Module_Class_Members(Txc1, cSimpleModule, 0);
    // Asagida yeniden tanimlanmis sanal fonksiyon, algoritmayi tutar
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};
// Module sinifi OMNET++ e kayitli olmayi gerektirir
Define_Module(Txc1);
void Txc1::initialize()
{
    // Initialize simulasyonun baslangicidir.
    // mhr-srm-mhr-srm islemini onden yuklemek icin, modullerden birinin
    // ilk mesaji gondermesine ihtiyac vardir.
    // Bu `mhr' in olusmasina izin verir
    // Mhr miyim? Yoksa Srm miyim?
    if (strcmp("mhr", name()) == 0)
    {
        // ilk mesaji olusturun ve "out" kapisi uzerinden gonderin.
        // "mhrsrmMsg" gelisiguzel bir string’dir ve
        // message nesnesinin adini olusturacaktir.
        cMessage *msg = new cMessage("mhrsrmMsg");
        send(msg, "out");
    }
}
void Txc1::handleMessage(cMessage *msg)
{
    // Nezaman bir mesaj module ulasirsa handleMessage() metodu cagirilir.
    // Burada biz, “out” kapisi vasitasi ile,
    // onu yanlizca diger module gonderiyoruz.
    // Cunku “mhr” ve “srm” nin herikiside ayni seyi yapiyor,
    // mesaj ikisi arasinda cevirilecektir.
    send(msg, "out");
}
```

Grafik özellik ve hata ayıklama mesajı eklemenin kaynak kodları

```
// txc2.cc
// OMNeT++/OMNEST ile “model olusturma” simulasyonunun bir parcasidir
#include <string.h>
#include <omnetpp.h>
class Txc2 : public cSimpleModule
{
  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};
Define_Module(Txc2);
void Txc2::initialize()
{
    if (strcmp("mhr", name()) == 0)
    {
        // `ev' nesnesi C44 deki `cout' gibi calisir.
        ev << "Baslangic mesaji gonderiliyor\n";
        cMessage *msg = new cMessage("mhrsrmMsg");
        send(msg, "out");
    }
}
void Txc2::handleMessage(cMessage *msg)
{
    // msg->name() msg nesnesinin adididr, burada "mhrsrmMsg" olacaktir.
    ev << "Alinan mesaj `" << msg->name() << "', yeniden gonderiliyor\n";
    send(msg, "out");
}
```

Durum değişkenleri eklemenin kaynak kodları

```
// txc3.cc
// OMNeT++/OMNEST ile “sayac” simulasyonunun bir parcasidir.
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
class Txc3 : public cSimpleModule
{
  private:
    int counter;  // sayac burada ekleniyor
  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};
Define_Module(Txc3);
void Txc3::initialize()
{
    // Sayac baslangic degeri olarak 10’a set ediliyor.
    // sayac degeri surekli azalacak ve mesaj sifira ulastiginda silinecek
    counter = 10;

    // The WATCH() bize Tkenv altinda iken degisken durumunu gozden
    // gecirmemizi saglar
    // Simulasyon birkac adim ilerledikten sonra, ya mhr ya da srm uzerinde
    // cift-klik yaparak acilan diyalog penceresinden Contents tab’ini secin
    // orada liste icinde “counter” i bulabilirsiniz
    WATCH(counter);
    if (strcmp("mhr", name()) == 0)
    {
        ev << "Baslangic mesaji gonderiliyor\n";
        cMessage *msg = new cMessage("mhrsrmMsg");
        send(msg, "out");
    }
}
void Txc3::handleMessage(cMessage *msg)
{
    // sayaci arttir ve degerini kontrol et
    counter--;
    if (counter==0)
    {
        // Eger sayac sifir ise mesaji sil
        // Eger bu modeli calistirirsak, simulasyon “no more events” mesajı
       // ile bu noktada durdugunu goruruz
        ev << name() << "'nin sayaci sifira dustu, mesaj siliniyor\n";
        delete msg;
    }
    else
    {
        ev << name() << "'nin sayaci " << counter << ", mesaj geri
gonderiliyor\n";
        send(msg, "out");
    }
}
```

Parametre eklemenin kaynak kodları

```
// txc4.cc
// OMNeT++/OMNEST ile “parametre ekleme” simulasyonunun bir parcasidir
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
class Txc4 : public cSimpleModule
{
  private:
    int counter;
  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};
Define_Module(Txc4);
void Txc4::initialize()
{
    // sayaci “limit” modul parametresi ile baslangic durumuna getiriyoruz
    // bunu NED dosyasi (mhrsrm4.ned) icinde ifade ediyoruz.
    counter = par("limit");
    if (strcmp("mhr", name()) == 0)
    {
        ev << "Baslangic mesaji gonderiliyor\n";
        cMessage *msg = new cMessage("mhrsrmMsg");
        send(msg, "out");
    }
}
void Txc4::handleMessage(cMessage *msg)
{
    counter--;
    if (counter==0)
    {
        ev << name() << "'nin sayaci sifira dustu, mesaj siliniyor\n";
        delete msg;
    }
    else
    {
        ev << name() << "'nin sayaci " << counter << ", mesaj geri
gonderiliyor\n";
        send(msg, "out");
    }
}
```

İşlem gecikmesini modellemenin kaynak kodları

```
// txc5.cc
// OMNeT++/OMNEST ile “geciktirme” simulasyonunun bir parcasidir.
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
class Txc5 : public cSimpleModule
{
  private:
    cMessage *event; // zamanlamada kullanilan event nesnesi icin isaretci
    cMessage *mhrsrmMsg; // mesajı biz onu geri gonderene kadar hatirlamayi
                         // saglayan degisken
  public:
    Txc5();
    virtual ~Txc5();
  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};
Define_Module(Txc5);
Txc5::Txc5()
{
    // isaretciye deger olarak NULL ata, bu sayede patlatici cokmeyecektir
    // baslangic islemi esnasinda kullanici iptal istemi yapsa ya da
    // islem zamani hatasi nedeni ile initialize() elde edilmese bile cokmez
    event = mhrsrmMsg = NULL;
}
Txc5::~Txc5()
{
    // dinamik olarak tahsis edilmis nesnelerin bir duzen icerisinde
    // yerlestirilmesi
    cancelAndDelete(event);
    delete mhrsrmMsg;
}
void Txc5::initialize()
{
    // zamanlama icin olusturcagimiz olay nesnesinin olusturulmasi
    // yanlizca siradan bir mesaj.
    event = new cMessage("event");

    // henuz mhrsrm mesaji yok.
    mhrsrmMsg = NULL;

    if (strcmp("mhr", name()) == 0)
    {
        // hemen baslatmiyoruz, kendi kendimize bir mesaj
        // gonderebiliriz
        // bir "oz-mesaj" – bize geri ulastiginda ilk olarak t=5.0s
        // simulasyon suresinde gonderme yapacagiz.
        ev << "ilk gonderme zamanlamasi t=5.0s\n";
        mhrsrmMsg = new cMessage("mhrsrmMsg");
        scheduleAt(5.0, event);
    }
}
void Txc5::handleMessage(cMessage *msg)
{
    // mesajlari ayirt etmek icin bazi yontemler vardir
    // ornegin mesaj turu (cMessage’in bir int ozelligi) ya da dynamic_cast
```

```
    // kullanarak sinif (cMessage tarafindan saglanan alt sinif) yardimi vb.
    // Eger en kolay ve en hizli yontem olan isaretciyi onayliyorsak.
    // Burada yanlizca kontrol yapiyoruz.
    if (msg==event)
    {
        // oz-mesaj ulastiginda, isaretcisini mhrsrmMsg ve NULL'a
        // gonderebiliriz
        // boylece daha sonra kafasi karismaz
        ev << "Periyod tamamlanana kadar bekle, mesaj geri gonderiliyor\n";
        send(mhrsrmMsg, "out");
        mhrsrmMsg = NULL;
    }
    else
    {
        // Aldigimiz mesaj eger oz-mesaj degilse, ortagimizdan gelen mhr-srm
        // mesaji olmak zorundadir.
        // mhrsrmMsg degiskenindeki isaretciyi hatirlariz ve sonra kendi
       // oz-mesajimizi 1s simulasyon suresinde bize geri gelmesi icin
       // zamanlariz
        ev << "Mesaj ulasti, 1 saniye gecikme baslatiliyor...\n";
        mhrsrmMsg = msg;
        scheduleAt(simTime()+1.0, event);
    }
}
```

Rastlantısal sayılar ve parametreler için kaynak kodlar

```
// txc6.cc
// OMNeT++/OMNEST ile “rastlantisal sayilar” simulasyaonunun bir parcasidir
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
class Txc6 : public cSimpleModule
{
  private:
    cMessage *event;
    cMessage *mhrsrmMsg;
  public:
    Txc6();
    virtual ~Txc6();
  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};
Define_Module(Txc6);
Txc6::Txc6()
{
    event = mhrsrmMsg = NULL;
}
Txc6::~Txc6()
{
    cancelAndDelete(event);
    delete mhrsrmMsg;
}
void Txc6::initialize()
{
    event = new cMessage("event");
    mhrsrmMsg = NULL;

    if (strcmp("mhr", name()) == 0)
    {
        ev << "Birinci gonderme t=5.0s olarak duzenleniyor\n";
        scheduleAt(5.0, event);
        mhrsrmMsg = new cMessage("mhrsrmMsg");
    }
}
void Txc6::handleMessage(cMessage *msg)
{
    if (msg==event)
    {
        ev << " Periyod tamamlanana kadar bekle, mesaj geri gonderiliyor\n";
        send(mhrsrmMsg, "out");
        mhrsrmMsg = NULL;
    }
    else
    {
        // 0.1 olasilik ile mesaj kaybediliyor:
        if (uniform(0,1) < 0.1)
        {
            ev << "\"Mesaj\" Siliniyor\n";
            delete msg;
        }
        else
        {
            // "delayTime" modul parametresi "exponential(5)"
```

```
          // (mhrsrm6.ned, omnetpp.ini) gibi degerlere set edilebilir
            // ve sonra burada her sefer icin farkli bir gecikme alacagiz
            double delay = par("delayTime");

            ev << "Messaj ulasti, bekleme suresi baslatiliyor " << delay <<
" secs...\n";
            mhrsrmMsg = msg;
            scheduleAt(simTime()+delay, event);
        }
    }
}
```

Zaman aşımı ve zamanlayıcıları iptal etmenin kaynak kodları

```
// txc7.cc
// OMNeT++/OMNEST ile “zaman asimi” simulasyonunun bir parcasidir
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>

class Mhr7 : public cSimpleModule
{
  private:
    double timeout;  // zaman asimi
    cMessage *timeoutEvent;  // zaman asimi oz-mesajina yonelik isaretci
                             // tutar
  public:
    Mhr7();
    virtual ~Mhr7();
  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};

Define_Module(Mhr7);
Mhr7::Mhr7()
{
    timeoutEvent = NULL;
}

Mhr7::~Mhr7()
{
    cancelAndDelete(timeoutEvent);
}

void Mhr7::initialize()
{
    // degiskenleri baslangic durumuna getirir
    timeout = 1.0;
    timeoutEvent = new cMessage("timeoutEvent");
    // baslangic mesaji uretir ve gonderir
    ev << "Baslangic mesaji gonderiliyor\n";
    cMessage *msg = new cMessage("mhrsrmMsg");
    send(msg, "out");
    scheduleAt(simTime()+timeout, timeoutEvent);
}

void Mhr7::handleMessage(cMessage *msg)
{
    if (msg==timeoutEvent)
    {
        // eger zaman asimi olayi alirsak, bu demek oluyor ki
        // paket zamaninda ulasmadi ve onu yeniden gondermek zorundayiz
        ev << "zaman asimi suresi doldu, mesaj yeniden gonderiliyor ve
zamanlayici yeniden baslatiliyor \n";
        cMessage *msg = new cMessage("mhrsrmMsg");
        send(msg, "out");
        scheduleAt(simTime()+timeout, timeoutEvent);
    }
    else // mesaj ulasti
    {
```

```
        // onay alindi – saklanan mesaj silinebilir ve
        // zaman asimi olayi iptal edilebilir
        ev << "Zamanlayici iptal edildi.\n";
        cancelEvent(timeoutEvent);

        // Baska bir tane gondermek icin hazir
        cMessage *msg = new cMessage("mhrsrmMsg");
        send(msg, "out");
        scheduleAt(simTime()+timeout, timeoutEvent);
    }
}

class Srm7 : public cSimpleModule
{
  protected:
    virtual void handleMessage(cMessage *msg);
};

Define_Module(Srm7);
void Srm7::handleMessage(cMessage *msg)
{
    if (uniform(0,1) < 0.1)
    {
        ev << "\"Mesaj \" kaybediliyor.\n";
        bubble("mesaj kaybedildi");  // animasyon yapmak daha aydinlatici...
        delete msg;
    }
    else
    {
        ev << "Ayni mesaj onay olarak geri gonderiliyor\n";
        send(msg, "out");
    }
}
```

Aynı mesajı yeniden iletmenin kaynak kodları

```
// txc8.cc
// OMNeT++/OMNEST ile “ayni mesaji yeniden gonderme” simulasyonunun
// bir parcasidir
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>

class Mhr8 : public cSimpleModule
{
  private:
    double timeout;  // zaman asimi
    cMessage *timeoutEvent;  // hoz-mesaj zaman asimina yonelik olarak
                             // isaretci tutar
    int seq;  // mesaj sira numarasi
    cMessage *message;  // zaman asiminda yeniden gonderilmesi gereken mesaj
  public:
    Mhr8();
    virtual ~Mhr8();
  protected:
    virtual cMessage *generateNewMessage();
    virtual void sendCopyOf(cMessage *msg);
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};

Define_Module(Mhr8);
Mhr8::Mhr8()
{
    timeoutEvent = message = NULL;
}
Mhr8::~Mhr8()
{
    cancelAndDelete(timeoutEvent);
    delete message;
}

void Mhr8::initialize()
{
    // degiskenler baslangic durumuna getirilir
    seq = 0;
    timeout = 1.0;
    timeoutEvent = new cMessage("timeoutEvent");
    // baslangic mesajini uret ve gonder
    ev << "Baslangic mesaji gonderiliyor \n";
    message = generateNewMessage();
    sendCopyOf(message);
    scheduleAt(simTime()+timeout, timeoutEvent);
}

void Mhr8::handleMessage(cMessage *msg)
{
    if (msg==timeoutEvent)
    {
        // eger zaman asimi olayi alirsak, bu demek oluyor ki
        // paket zamaninda ulasmadi ve onu yeniden gondermek zorundayiz
        ev << "Zaman asimi suresi doldu, mesaj yeniden gonderiliyor ve
zamanlayici yeniden baslatiliyor\n";
```

```
        sendCopyOf(message);
        scheduleAt(simTime()+timeout, timeoutEvent);
    }
    else // mesaj ulasti
    {
        // onay alindi!
        ev << "Alindi: " << msg->name() << "\n";
        delete msg;
        // ayni zamanda saklanmis mesaji sil ve zaman asimi olayini iptal et
        ev << "Zamanlayici iptal edildi.\n";
        cancelEvent(timeoutEvent);
        delete message;
        // baska bir tane gondermek icin hazir
        message = generateNewMessage();
        sendCopyOf(message);
        scheduleAt(simTime()+timeout, timeoutEvent);
    }
}

cMessage *Mhr8::generateNewMessage()
{
    // ir mesaji her zaman farkli bir isim ile uret
    char msgname[20];
    sprintf(msgname, "mhr-%d", ++seq);
    cMessage *msg = new cMessage(msgname);
    return msg;
}

void Mhr8::sendCopyOf(cMessage *msg)
{
    // mesaji cogalt ve kopyasini gonder
    cMessage *copy = (cMessage *) msg->dup();
    send(copy, "out");
}

class Srm8 : public cSimpleModule
{
  protected:
    virtual void handleMessage(cMessage *msg);
};

Define_Module(Srm8);
void Srm8::handleMessage(cMessage *msg)
{
    if (uniform(0,1) < 0.1)
    {
        ev << "\"Mesaj \" Kaybediliyor " << msg << endl;
        bubble("Mesaj kaybedildi");
        delete msg;
    }
    else
    {
        ev << msg << " alindi, geriye onay gonderiliyor.\n";
        delete msg;
        send(new cMessage("ack"), "out");
    }
}
```

İkiden fazla düğüm ile benzetim yapmanın kaynak kodları

```
//txc9.cc
// OMNeT++/OMNEST ile “ikiden fazla dugum” simulasyonunun bir parcasidir
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>

class Txc9 : public cSimpleModule
{
  protected:
    virtual void forwardMessage(cMessage *msg);
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};

Define_Module(Txc9);

void Txc9::initialize()
{
    if (index()==0)
    {
        // baslangic mesajini oz-mesaj olarak duzenlemek icin islemi onyukle
        char msgname[20];
        sprintf(msgname, "mhr-%d", index());
        cMessage *msg = new cMessage(msgname);
        scheduleAt(0.0, msg);
    }
}

void Txc9::handleMessage(cMessage *msg)
{
    if (index()==3)
    {
        // Mesaj ulasti
        ev << "" << msg << " mesaji ulasti.\n";
        delete msg;
    }
    else
    {
        // mesaji iletmeliyiz
        forwardMessage(msg);
    }
}

void Txc9::forwardMessage(cMessage *msg)
{
    // bu ornekte, mesai uzerinden gondermek icin rastagele bir kapi
    // seciyoruz
    // 0 ve out[] kapisi buyuklugu arasinda rastgele bir sayi seciyoruz
    int n = gate("out")->size();
    int k = intuniform(0,n-1);

    ev << "" << msg << " mesaji out[" << k << "] portu uzerinden
iletiliyor\n";
    send(msg, "out", k);
}
```

Kendi mesaj sınıfımızı tanımlamanın kaynak kodları

```
// txc10.cc
// OMNeT++/OMNEST ile “mesaj class tanimlama” simulasyonunun bir parcasidir.
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
// onceden uretilmis bir dosya icerir: baslik dosyasi mhrsrm10.msg
// tarafindan olusturulur
// cMessage tarafindan elde edilmis, MhrsrmMsg10 sinifinin tanimini icerir
#include "mhrsrm10_m.h"

class Txc10 : public cSimpleModule
{
  protected:
    virtual MhrsrmMsg10 *generateMessage();
    virtual void forwardMessage(MhrsrmMsg10 *msg);
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};

Define_Module(Txc10);

void Txc10::initialize()
{
    // Modul 0 ilk mesaji gonderir
    if (index()==0)
    {
        // onyukleme islemi baslangic mesajini bir oz-mesaj olarak
duzenliyor
        MhrsrmMsg10 *msg = generateMessage();
        scheduleAt(0.0, msg);
    }
}

void Txc10::handleMessage(cMessage *msg)
{
    MhrsrmMsg10 *ttmsg = check_and_cast<MhrsrmMsg10 *>(msg);

    if (ttmsg->getDestination()==index())
    {
        // Mesaj ulasti
        ev << "" << ttmsg << " mesaji " << ttmsg->getHopCount() << "
sicramadan sonra ulasti.\n";
        bubble("ULASTI, yeni bir tane gonderiliyor!");
        delete ttmsg;
        // baska bir tane uretiliyor
        ev << "Baska bir mesaj uretiliyor: ";
        MhrsrmMsg10 *newmsg = generateMessage();
        ev << newmsg << endl;
        forwardMessage(newmsg);
    }
    else
    {
        // mesaji iletmemeiz gerekiyor
        forwardMessage(ttmsg);
    }
}

MhrsrmMsg10 *Txc10::generateMessage()
```

```
{
    // kaynak ve hedef adresi uretme
    int src = index();   // modulumuzun dizini
    int n = size();      // modul vektor buyuklugu
    int dest = intuniform(0,n-2);
    if (dest>=src) dest++;
    char msgname[20];
    sprintf(msgname, "mhr-%d-to-%d", src, dest);
    // mesaj nesnesi olusturma ve kaynak ve hedef alani set etme
    MhrsrmMsg10 *msg = new MhrsrmMsg10(msgname);
    msg->setSource(src);
    msg->setDestination(dest);
    return msg;
}

void Txc10::forwardMessage(MhrsrmMsg10 *msg)
{
    // Increment hop count.
    msg->setHopCount(msg->getHopCount()+1);
    // Same routing as before: random gate.
    int n = gate("out")->size();
    int k = intuniform(0,n-1);
    ev << "" << msg << " mesaji out[" << k << "] portu uzerinden
iletiliyor\n";
    send(msg, "out", k);
}
```

Gönderilen/alınan paketlerin sayısını görüntülemenin kaynak kodları

```
// txc11.cc
// OMNeT++/OMNEST ile “paket sayisi goruntuleme” simulasyonunun bir
// parcasidir
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
#include "mhrsrm11_m.h"
class Txc11 : public cSimpleModule
{
  private:
    long numSent;
    long numReceived;
  protected:
    virtual MhrsrmMsg11 *generateMessage();
    virtual void forwardMessage(MhrsrmMsg11 *msg);
    virtual void updateDisplay();

    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};

Define_Module(Txc11);

void Txc11::initialize()
{
    // degiskenlerin baslangic degerlerini set edelim
    numSent = 0;
    numReceived = 0;
    WATCH(numSent);
    WATCH(numReceived);
    // Module 0 ilk mesaji gonderir
    if (index()==0)
    {
        // onyukleme islemi baslangic mesajini oz-mesaj olarak duzenliyor
        MhrsrmMsg11 *msg = generateMessage();
        scheduleAt(0.0, msg);
    }
}

void Txc11::handleMessage(cMessage *msg)
{
    MhrsrmMsg11 *ttmsg = check_and_cast<MhrsrmMsg11 *>(msg);
    if (ttmsg->getDestination()==index())
    {
        // Mesaj ulasti
        int hopcount = ttmsg->getHopCount();
        ev << "" << ttmsg << " mesaji " << hopcount << " sicramadan sonra
ulasti.\n";
        numReceived++;
        delete ttmsg;
        bubble("ULASTI, yeni bir tane baslatiliyor!");
        // baska bir tane uretiliyor.
        ev << "baska bir mesaj uretiliyor: ";
        MhrsrmMsg11 *newmsg = generateMessage();
        ev << newmsg << endl;
        forwardMessage(newmsg);
        numSent++;
        if (ev.isGUI())
```

```
            updateDisplay();
    }
    else
    {
        // mesaji iletmeliyiz
        forwardMessage(ttmsg);
    }
}

MhrsrmMsg11 *Txc11::generateMessage()
{
    // kaynak ve hedef adreslerini uretmeliyiz.
    int src = index();   // modulumuzun dizini
    int n = size();      // modul vektor buyuklugu
    int dest = intuniform(0,n-2);
    if (dest>=src) dest++;
    char msgname[20];
    sprintf(msgname, "mhr-%d-to-%d", src, dest);
    // mesaj nesnesi olusturalim ve kaynak ve hedef alanlarini set edelim
    MhrsrmMsg11 *msg = new MhrsrmMsg11(msgname);
    msg->setSource(src);
    msg->setDestination(dest);
    return msg;
}

void Txc11::forwardMessage(MhrsrmMsg11 *msg)
{
    // sicrama sayacaini arttiralim
    msg->setHopCount(msg->getHopCount()+1);
    // oncekinde oldugu gibi yonlendirme ayni: rastlantisal kapi.
    int n = gate("out")->size();
    int k = intuniform(0,n-1);
    ev << "" << msg << " mesaji out[" << k << "] portu uzerinden
iletiliyor\n";
    send(msg, "out", k);
}

void Txc11::updateDisplay()
{
    char buf[40];
    sprintf(buf, "alinan: %ld gonderilen: %ld", numReceived, numSent);
    displayString().setTagArg("t",0,buf);
}
```

İstatistik derlemelerini dahil etmenin kaynak kodları

```
// txc12.cc
// OMNeT++/OMNEST ile “istatistik cikarma” simulasyonunun bir parcasidir.
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
#include "mhrsrm12_m.h"

class Txc12 : public cSimpleModule
{
  private:
    long numSent; long numReceived;
    cLongHistogram hopCountStats;
    cOutVector hopCountVector;
  protected:
    virtual MhrsrmMsg12 *generateMessage();
    virtual void forwardMessage(MhrsrmMsg12 *msg);
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);

    // finish() fonksiyonu simulasyon sonunda OMNeT++ tarafindan cagirilir:
    virtual void finish();
};

Define_Module(Txc12);

void Txc12::initialize()
{
    // degiskenler baslangic degegerlerine set ediliyor
    numSent = 0;
    numReceived = 0;
    WATCH(numSent);
    WATCH(numReceived);
    hopCountStats.setName("hopCountStats");
    hopCountStats.setRangeAutoUpper(0, 10, 1.5);
    hopCountVector.setName("HopCount");
    // Module 0 ilk mesaji gonderir
    if (index()==0)
    {
        // on yukleme islemi baslangic mesajini oz-mesaj olarak duzenler.
        MhrsrmMsg12 *msg = generateMessage();
        scheduleAt(0.0, msg);
    }
}

void Txc12::handleMessage(cMessage *msg)
{
    MhrsrmMsg12 *ttmsg = check_and_cast<MhrsrmMsg12 *>(msg);

    if (ttmsg->getDestination()==index())
    {
        // Mesaj ulasti
        int hopcount = ttmsg->getHopCount();
        ev << "" << ttmsg << " mesaji " << hopcount << " sicramadan sonra
ulasti.\n";
        bubble("ULASTI, yeni bir tane baslatiliyor!");
        // istatistikler guncelleniyor.
        numReceived++;
```

```
        hopCountVector.record(hopcount);
        hopCountStats.collect(hopcount);
        delete ttmsg;
        // baska bir tane uretiliyor.
        ev << "baska bir mesaj uretiliyor: ";
        MhrsrmMsg12 *newmsg = generateMessage();
        ev << newmsg << endl;
        forwardMessage(newmsg);
        numSent++;
    }
    else
    {
        // mesaji letmeliyiz.
        forwardMessage(ttmsg);
    }
}

MhrsrmMsg12 *Txc12::generateMessage()
{
    // kaynak ve hedef adresleri olusturuluyor.
    int src = index();
    int n = size();
    int dest = intuniform(0,n-2);
    if (dest>=src) dest++;
    char msgname[20];
    sprintf(msgname, "mhr-%d-to-%d", src, dest);
    // mesaj nesnesini olusturalim ve kaynak ve hedef alaninin set edelim.
    MhrsrmMsg12 *msg = new MhrsrmMsg12(msgname);
    msg->setSource(src);
    msg->setDestination(dest);
    return msg;
}

void Txc12::forwardMessage(MhrsrmMsg12 *msg)
{
    // sicrama sayisini arttiralim.
    msg->setHopCount(msg->getHopCount()+1);
    // onceki ile ayni yonlendirme: rastlantisal kapi.
    int n = gate("out")->size();
    int k = intuniform(0,n-1);
    ev << "" << msg << " mesaji out[" << k << "] portu uzerinden
iletiliyor\n";
    send(msg, "out", k);
}

void Txc12::finish()
{
    // bu fonksiyon simulasyon sonunda OMNeT++ tarafindan cagirilir.
    ev << "Gonderilen:     " << numSent << endl;
    ev << "Alinan: " << numReceived << endl;
    ev << "En dusuk sekme sayisi:    " << hopCountStats.min() << endl;
    ev << "En yuksek sekme sayisi:    " << hopCountStats.max() << endl;
    ev << "Ortalama sekme sayisi:   " << hopCountStats.mean() << endl;
    ev << "Sekme sayisi standart sapmasi: " << hopCountStats.stddev() <<
endl;
    recordScalar("#sent", numSent);
    recordScalar("#received", numReceived);
    hopCountStats.recordScalar("hop count");
}
```

OMNeT++ hareketlilik modülü ile taşma protokolü benzetiminin kaynak kodları

```
//
// FloodHost_n.cc
//
#include <math.h>
#include "omnetpp.h"
// NEDC surum kontrolu
#define NEDC_VERSION 0x0302
#if (NEDC_VERSION!=OMNETPP_VERSION)
#    error Version mismatch! Probably this file was generated by an earlier
version of nedc: 'make clean' should help.
#endif
// Kullanilmayan degiskenler ile ilgili uyarilari disable edebiliriz
(yanlizca MSVC ve BC icin):
// GCC icin boyle bir yontem yok
#ifdef _MSC_VER
#  pragma warning(disable:4101)
#endif
#ifdef __BORLANDC__
#  pragma warn -waus
#  pragma warn -wuse
#endif

static cModuleType *_getModuleType(const char *modname)
{
    cModuleType *modtype = findModuleType(modname);
    if (!modtype)
        throw new cRuntimeError("%s modul tipi icin tanimlama bulunamadi
(C++ kodundan dolayi Define_Module() hatali olabilir mi?)", modname);
    return modtype;
}

static void _checkModuleVectorSize(int vectorsize, const char *mod)
{
    if (vectorsize<0)
        throw new cRuntimeError("Negatif modul vektor buyuklugu %s[%d]",
mod, vectorsize);
}

static void _readModuleParameters(cModule *mod)
{
    int n = mod->params();
    for (int k=0; k<n; k++)
        if (mod->par(k).isInput())
            mod->par(k).read();
}

static int _checkModuleIndex(int index, int vectorsize, const char *modname)
{
    if (index<0 || index>=vectorsize)
        throw new cRuntimeError("S%s[%d] Altmodul indeksi erimin (araligin)
disinda, %s'nin buyuklugu %d'dir", modname, index, modname, vectorsize);
    return index;
}

static cGate *_checkGate(cModule *mod, const char *gatename)
{
```

```
    cGate *g = mod->gate(gatename);
    if (!g)
        throw new cRuntimeError("%s'nin %s isimli kapisi yok",mod-
>fullPath().c_str(), gatename);
    return g;
}

static cGate *_checkGate(cModule *mod, const char *gatename, int gateindex)
{
    cGate *g = mod->gate(gatename, gateindex);
    if (!g)
        throw new cRuntimeError("%s'nin %s[%d] kapisi yok",mod-
>fullPath().c_str(), gatename, gateindex);
    return g;
}

static cGate *_getFirstUnusedParentModGate(cModule *mod, const char
*gatename)
{
    int baseId = mod->findGate(gatename);
    if (baseId<0)
        throw new cRuntimeError("%s'nin %s[] kapisi yoktur",mod-
>fullPath().c_str(), gatename);
    int n = mod->gate(baseId)->size();
    for (int i=0; i<n; i++)
        if (!mod->gate(baseId+i)->isConnectedInside())
            return mod->gate(baseId+i);
    throw new cRuntimeError("%s[] kapilarinin tamami baglidir, `++'
isleticisi icin kapi kalmamistir",mod->fullPath().c_str(), gatename);
}

static cGate *_getFirstUnusedSubmodGate(cModule *mod, const char *gatename)
{
    int baseId = mod->findGate(gatename);
    if (baseId<0)
        throw new cRuntimeError("%s'nin %s[] kapisi yoktur",mod-
>fullPath().c_str(), gatename);
    int n = mod->gate(baseId)->size();
    for (int i=0; i<n; i++)
        if (!mod->gate(baseId+i)->isConnectedOutside())
            return mod->gate(baseId+i);
    int newBaseId = mod->setGateSize(gatename,n+1);
    return mod->gate(newBaseId+n);
}

static cFunctionType *_getFunction(const char *funcname, int argcount)
{
    cFunctionType *functype = findFunction(funcname,argcount);
    if (!functype)
        throw new cRuntimeError("%d degerli %s fonksiyonu bulunamadi",
argcount, funcname);
    return functype;
}

static cChannel *_createChannel(const char *channeltypename)
{
    cChannelType *channeltype = findChannelType(channeltypename);
    if (!channeltype)
        throw new cRuntimeError("%s kanal tipi bulunamadi",
channeltypename);
    cChannel *channel = channeltype->create("channel");
    return channel;
}

static cChannel *_createNonTypedBasicChannel(double delay, double error,
double datarate)
{
```

```
    cBasicChannel *channel = new cBasicChannel("channel");
    if (delay!=0) channel->setDelay(delay);
    if (error!=0) channel->setError(error);
    if (datarate!=0) channel->setDatarate(datarate);
    return channel;
}

static cXMLElement *_getXMLDocument(const char *fname, const char
*pathexpr=NULL)
{
    cXMLElement *node = ev.getXMLDocument(fname, pathexpr);
    if (!node)
        throw new cRuntimeError(!pathexpr ? "xmldoc(\"%s\"): elemani
bulunamadi" : "xmldoc(\"%s\", \"%s\"): elemani bulunamadi",fname,pathexpr);
    return node;
}

ModuleInterface(FloodHost)
    // parametreler:
    Parameter(applLayer, ParType_String)
EndInterface
Register_ModuleInterface(FloodHost);
class FloodHost : public cCompoundModule
{
  public:
    FloodHost() : cCompoundModule() {}
  protected:
    virtual void doBuildInside();
};

Define_Module(FloodHost);
void FloodHost::doBuildInside()
{
    cModule *mod = this;
    // gecici degiskenler:
    cPar tmpval;
    const char *modtypename;
    mod->setBackgroundDisplayString("p=10,10;b=180,200,rect;o=white");
    // altmoduller:
    cModuleType *modtype = NULL;
    int submodindex;
    //
    // 'blackboard' altmodulu:
    //
    int blackboard_size = 1;
    modtype = _getModuleType("Blackboard");
    cModule *blackboard_p = modtype->create("blackboard", mod);
    {
        cContextSwitcher __ctx(blackboard_p); // kalanlari bu modul
icerisinde yap
        _readModuleParameters(blackboard_p);
        blackboard_p->setDisplayString("p=130,70,rect;b=25,25;o=black");
    }
    //
    // 'mobility' altmodulu:
    //
    int mobility_size = 1;
    modtype = _getModuleType("BasicMobility");
    cModule *mobility_p = modtype->create("mobility", mod);
    {
        cContextSwitcher __ctx(mobility_p); // kalanlari bu modul icerisinde
yap
        _readModuleParameters(mobility_p);
        mobility_p->setDisplayString("p=130,140;i=cogwheel2");
    }
    //
    // 'appl' altmodulu:
```

```
    //
    int appl_size = 1;
    modtypename = mod->par("applLayer");
    modtype = _getModuleType(modtypename);
    cModule *appl_p = modtype->create("appl", mod);
    {
        cContextSwitcher __ctx(appl_p); // kalanlari bu modul icerisinde yap
        _readModuleParameters(appl_p);
        appl_p->setDisplayString("b=35,26;p=60,50;i=app");
    }
    //
    // 'net' altmodulu:
    //
    int net_size = 1;
    modtype = _getModuleType("Flood");
    cModule *net_p = modtype->create("net", mod);
    {
        cContextSwitcher __ctx(net_p); // kalanlari bu modul icerisinde yap
        _readModuleParameters(net_p);
        net_p->setDisplayString("b=32,30;p=60,108;i=prot1");
    }
    //
    // 'nic' altmodulu:
    //
    int nic_size = 1;
    modtype = _getModuleType("BasicNic");
    cModule *nic_p = modtype->create("nic", mod);
    {
        cContextSwitcher __ctx(nic_p); // kalanlari bu modul icerisinde yap
        _readModuleParameters(nic_p);
        nic_p->setDisplayString("b=32,30;p=60,166;i=iface");
    }
    //
    // baglantilar:
    //
    cGate *srcgate, *destgate;
    cChannel *channel;
    cPar *par;
    // baglanti
    srcgate = _checkGate(nic_p, "uppergateOut");
    destgate = _checkGate(net_p, "lowergateIn");
    srcgate->connectTo(destgate);
    // baglanti
    srcgate = _checkGate(net_p, "lowergateOut");
    destgate = _checkGate(nic_p, "uppergateIn");
    srcgate->connectTo(destgate);
    // baglanti
    srcgate = _checkGate(net_p, "uppergateOut");
    destgate = _checkGate(appl_p, "lowergateIn");
    srcgate->connectTo(destgate);
    // baglanti
    srcgate = _checkGate(appl_p, "lowergateOut");
    destgate = _checkGate(net_p, "uppergateIn");
    srcgate->connectTo(destgate);
    // baglanmis tum kapilari kontrol et:
    mod->checkInternalConnections();
    //
    // this level is done -- recursively build submodules too
    //
    blackboard_p->buildInside();
    mobility_p->buildInside();
    appl_p->buildInside();
    net_p->buildInside();
    nic_p->buildInside();
}
```

```
//
// Flood.cc
//

#include "Flood.h"
#include <NetwPkt_m.h>

#define EV (ev.disabled()||!debug) ? (std::ostream&)ev : ev << logName() <<
"::Flood: "

Define_Module(Flood);

/**
 * ini dosyasindan tum parametreleri oku. Eger ini dosyasinda bir parametre
 * belirlenmemis ise varsayilan deger set edilecek.
 **/

void Flood::initialize(int stage)
{

  BasicNetwLayer::initialize(stage);
    // raporlama degiskenlerinin baslangic degerlerini set edelim
    banaUygunAldim = 0;
    aldimGeneleYayinladim = 0;
    banaAitDegilYenidenYayinla = 0;
    oncedenSacilmisti = 0;
    WATCH(banaUygunAldim);
    WATCH(aldimGeneleYayinladim);
    WATCH(banaAitDegilYenidenYayinla);
    WATCH(oncedenSacilmisti);
       //aldimStats.setName("aldimStats");
       //banaUygunAldimStats.setRangeAutoUpper(0, 10, 1.5);
       //aldimVector.setName("aldim");

       // baslangic durumuna getirme icin yasanan gecikmenin hesaplanmasi
       //endToEndDelayVec.setName("Uctan-Uca Gecikme");
       /*banaUygunAldimVector.setName("Alinmasi uygun mesaj");
       aldimGeneleYayinladimVector.setName("Alinmis genele yayinlanacak
mesaj");
       banaAitDegilYenidenYayinlaVector.setName("linamamis genele
yayinlanacak mesaj");
       oncedenSacilmistiVector.setName("Onceden genel yayinlanmis mesaj");
       */
  if(stage==0){
    //sira numarasini 0 baslangic durumuna getir
    seqNum = 0;
    if(hasPar("defaultTtl"))
      defaultTtl = par("defaultTtl");
    else{
      defaultTtl = 5;
      ev <<logName()<<"::BasicDecider: Flood.ned icerisinde varsayilan ttl
verilmemistir. Varsayilan Ttl set ediliyor "<<defaultTtl<<endl;
    }
    EV <<"defaultTtl = "<<defaultTtl<<endl;
    if(hasPar("plainFlooding"))
      plainFlooding = par("plainFlooding");
    else{
      plainFlooding = true;
      ev <<logName()<<"::BasicDecider: Flood.ned icinde plainFlooding
bayragi yoktur. plainFlooding etkinlestiriliyor\n";
    }
    if(plainFlooding){
      //bu parametreler yanlizca basit tasma icin gereklidir
      if(hasPar("bcMaxEntries")){
       if((int)par("bcMaxEntries") < 1)
         error("bcMaxEntries 0'dan buyuk bir pozitif tamsayiya sahip
olmalidir ");
```

```
      else
        bcMaxEntries = par("bcMaxEntries");
      }
      else{
      bcMaxEntries = 30;
      ev <<logName()<<"::BasicDecider: bcMaxEntries, Flood.ned icerisinde
tanimlanmamistir. bcMaxEntries set ediliyor "
         <<bcMaxEntries<<endl;
      }
      EV <<"bcMaxEntries = "<<bcMaxEntries<<endl;

      if(hasPar("bcDelTime"))
      if((double)par("bcDelTime") < 1)
        error("bcDelTime 0'dan buyuk pozitif tam sayiya sahip olmalidir");
      else
        bcDelTime = par("bcDelTime");
      else{
      bcDelTime = 3.0;
      ev <<logName()<<"::BasicDecider: bcDelTime, Flood.ned icerisinde
tanimlanmamistir. bcDelTime set ediliyor "
         <<bcDelTime<<endl;
      }
      EV <<"bcDelTime = "<<bcDelTime<<endl;
      bcMsgs = new cBroadcastList();
    }
  }
}

Flood::~Flood()
{
  if(plainFlooding)
    delete bcMsgs;
}

/**
 * Tum mesajlar bir sira numarasi almalidir ve ttl belirlenmelidir.
 * Daha sonra mesajlar mac katmanina dogru uzatilabilir.
 * Mac adresi -1'e (broadcast adresi) set edilir, cunki
 * mesaj tasirilir (ornegin tum komsularina gorilmesi gerekebilir)
 *
 * duz (sade) tasma ile ilgili olarak, mesaj sira numarasi ve kaynak adresi
 * bcMsgs listesi icinde tutulmalidir, boylece o mesaj yeniden
 * yayimlanmayacaktir (rebroadcast), eger bir kopyasi komsu dugumlerden
geriye
 * dogru tasirilmis olacaksa.
 *
 * Eger en yuksek girdi sayisina ulasilirsa, birinci giris (en eski olan)
silinir.
 **/

void Flood::handleUpperMsg(NetwPkt* msg)
{
  //msg seqNum degerini set et
  msg->setSeqNum( seqNum );
  //sira numarasi sayacini arttir
  seqNum++;
  //Eger msg defaultTtl'ye yonelik ise, ttl alanini set et
  msg->setTtl( defaultTtl );
  if(plainFlooding){
    if( bcMsgs->size() >= bcMaxEntries ){
      cBroadcastList::iterator it;
      //tarihi gecmis girislerin yayim listesini ara ve onlari sil
      for( it=bcMsgs->begin(); it!=bcMsgs->end(); it++ ){
      if( it->delTime < simTime() ){
        bcMsgs->erase(it);
        it--;
        break;
```

```
      }
      }
      //en yuksek boyuta ulasilmis ise, en eski girisi sil
      if( bcMsgs->size() >= bcMaxEntries ){
       EV <<"bcMsgs dolmustur, eski girisleri sil"<<endl;
       bcMsgs->pop_front();
      }
    }
    //listeye yeni girisi ekle
    bcast entry;
    entry.seqNum = msg->getSeqNum();
    entry.srcAddr = msg->getSrcAddr();
    entry.delTime = simTime() + bcDelTime;
    bcMsgs->push_back(entry);
  }
  //yonlendirme yoktur, bu nedenle tum mesajlar mac katmani icin sacilirlar
  sendDown( msg, -1, lowergateOut );
}

/**
 * Mac katmanindan gelen mesajlar,
 * yanlizca yayinlanmis ya da bu dugume dogru yol almislarsa,
 * uygulamaya dogru ilerletilecekler.
 *
 * Eger ulasan mesaj bir nesredilmis (broadcast), yayinlanmis, mesaj ise,
 * eger tll alani bir'den daha buyuk ise, o ayrica yeniden tasirilir.
 * Mesaj geriye mac katmanina devredilmeden once, bu sicramanin sebebini
belirtmek icin,
 * ttl alani bir tarafindan azaltilir.
 *
 * Normal tasma oldugu taktirde, mesaj yanlizca isleme tabi tutulur,
 * eger bcMsgs listesine (@ref notBroadcasted) uygun giris yapilmamissa.
 * Diger durumda mesaj silinecektir.
 **/

void Flood::handleLowerMsg(NetwPkt* msg)
{
  //msg henuz nesredilmesi (yayinlanmadi)
  if( notBroadcasted( msg ) ){
    //bana uygun msg
    if( msg->getDestAddr() == myNetwAddr() ){
      EV <<" yayilimi "<<getSenderName(msg)<<"'den baslayan msg verisi bana
uygun! ust kisima gonder"<<endl;
      sendUp( msg );
             // performans olcumu icin "banaUygunAldim" degerinin kontrolu
             banaUygunAldim++;
             banaUygunAldimVector.record(banaUygunAldim);
             banaUygunAldimStats.collect(banaUygunAldim);

      if (ev.isGUI())
      updateDisplay();
    }
    //mesaji nesret (yayinla)
    else if( msg->getDestAddr() == -1 ){
      //ttl ve yeniden nesretmeyi kontrol et
      if( msg->getTtl() > 1 ){
       EV <<" yayilimi "<<getSenderName(msg)<<"'den baslayan msg verisi
SACILIYOR! ttl = "<<msg->getTtl()
          <<" > 1 -> msg'yi yeniden yayimla & ust kisima gonder\n";
       msg->setTtl( msg->getTtl()-1 );
       sendDown( (NetwPkt*)msg->dup(), -1, lowergateOut );
             // performans olcumu icin "aldimGeneleYayinladim" degerinin
kontrolu
             aldimGeneleYayinladim++;
             aldimGeneleYayinladimVector.record(aldimGeneleYayinladim);
             aldimGeneleYayinladimStats.collect(aldimGeneleYayinladim);
```

```
      if (ev.isGUI())
      updateDisplay();
      }
      else
      EV <<" max sicramaya ulasildi (ttl = "<<msg->getTtl()<<") -> yanliz
ust kisima gonder\n";
      sendUp( msg );
    }
    //bana uygun degil -> yeniden yayinla (nesret)
    else{
      //ttl ve yeniden yayinlamayi kontrol et
      if( msg->getTtl() > 1 ){
      EV <<getSenderName(msg)<<"'den alinan msg verisi bana ait degil! ttl
= "<<msg->getTtl()
         <<" > 1 -> "<<getLogName(msg->getDestAddr())<<" numarali host'a
iletiliyor"<<endl;
      msg->setTtl( msg->getTtl()-1 );
      sendDown( msg, -1, lowergateOut );
            // performans olcumu icin "banaAitDegilYenidenYayinla"
degerinin kontrolu
            banaAitDegilYenidenYayinla++;

      banaAitDegilYenidenYayinlaVector.record(banaAitDegilYenidenYayinla);

      banaAitDegilYenidenYayinlaStats.collect(banaAitDegilYenidenYayinla);
            //EV <<"banaAitDegilYenidenYayinla =
"<<banaAitDegilYenidenYayinla<<endl;

      if (ev.isGUI())
      updateDisplay();
      }
      else{
      //max sicramaya ulasildi -> sil
      EV <<" max sicramaya ulasildi (ttl = "<<msg->getTtl()<<") -> msg'yi
sil\n";
      delete msg;
      }
    }
  }
  else{
    EV <<" msg verisi onceden YAYINLANMISTI! msg'yi sil\n";
      // performans olcumu icin "oncedenSacilmisti" degerinin kontrolu
      oncedenSacilmisti++;
      oncedenSacilmistiVector.record(oncedenSacilmisti);
      oncedenSacilmistiStats.collect(oncedenSacilmisti);

      if (ev.isGUI())
      updateDisplay();
    delete msg;
  }
}

/**
 * bcMsgs listesi ulasan mesaj icin taranir.
 * Eger mesaj listede ise, o onceden yayinlanmistir ve bu nedenle fonksiyon
 * hata geri dondurur.
 *
 * Ayni zamanda tarihi gecmis (bcDelTime'dan daha eski) olanlar silinir.
 * Eger liste tam ise ve yeni bir mesaj girilmek zorunda ise,
 * en eski giris silinir.
 **/

bool Flood::notBroadcasted( NetwPkt* msg )
{
  if(!plainFlooding)
    return true;
  cBroadcastList::iterator it;
```

```
  //suresi gecmis girislerin yayin listesini ara ve onlari sil
  for( it=bcMsgs->begin(); it!=bcMsgs->end(); it++ ){
    if( it->delTime < simTime() ){
      bcMsgs->erase(it);
      it--;
    }
    //mesaj onceden yayinlandi
    if( (it->srcAddr==msg->getSrcAddr()) && (it->seqNum==msg->getSeqNum())
){
      // girisi guncelle
      it->delTime = simTime() + bcDelTime;
      return false;
    }
  }
  //maksimum boyuta ulasildi ise en eski girisi sil
  if( bcMsgs->size() >= bcMaxEntries ){
    EV <<"bcMsgs dolmustur, eski girisleri sil\n";
    bcMsgs->pop_front();
  }
  //yeni bir giris ekle
  bcast entry;
  entry.seqNum = msg->getSeqNum();
  entry.srcAddr = msg->getSrcAddr();
  entry.delTime = simTime() + bcDelTime;
  bcMsgs->push_back(entry);
  return true;
}

void Flood::updateDisplay()
{
    char bufstr[90]="";
    ev <<logName()<<" alinan:"<<banaUygunAldim<<"
yayinlanan:"<<aldimGeneleYayinladim<<" ait
olmayan:"<<banaAitDegilYenidenYayinla<<"
yayinlanmayan:"<<oncedenSacilmisti<<endl;
    sprintf(bufstr, "alinan:%d", banaUygunAldim);
    //sprintf(bufstr, "alinan:%d yayinlanan:%d ait olmayan:%d
yayinlanmayan:%d", banaUygunAldim, aldimGeneleYayinladim,
banaAitDegilYenidenYayinla, oncedenSacilmisti);
    displayString().setTagArg("t",0,bufstr);
    displayString().setTagArg("i",1,"Muharrem");
}

void Flood::finish()
{
    // bu fonksiyon simulasyon sonunda OMNeT++ tarafindan cagirilir.
    ev << logName() << " tarafindan alinmis mesaj sayisi
: " << banaUygunAldim << endl;
    ev << logName() << " tarafindan alinmis ve genele yayinlanmis mesaj
sayisi      : " << aldimGeneleYayinladim << endl;
    ev << logName() << " tarafindan alinamamis fakat genele yayinlanmis
mesaj sayisi: " << banaAitDegilYenidenYayinla << endl;
    ev << logName() << " tarafindan onceden genel yayinlanmis mesaj sayisi
: " << oncedenSacilmisti << endl;
    recordScalar("Aldim", banaUygunAldim);
    recordScalar("Yayinladim", aldimGeneleYayinladim);
    recordScalar("UygunDegilYenidenYayinla", banaAitDegilYenidenYayinla);
    recordScalar("Sacilmisti", oncedenSacilmisti);
}
```

OMNeT++ hareketlilik modülü ile IEEE 802.11 iletişim kuralı benzetiminin kaynak kodları

```
//
// Mac80211.cc
//
#include "Mac80211.h"
#include "RadioState.h"
#define EV (ev.disabled()||!debug) ? (std::ostream&)ev : ev << logName() <<
"::Mac80211: "

Define_Module( Mac80211 );

void Mac80211::initialize(int stage)
{
  BasicMacLayer::initialize(stage);
    // raporlama degiskenlerinin baslangic degerlerini set edelim
    iletiAlindi = 0;
    msgBanaAit = 0;
    msgBanaAitDegil = 0;
    ackIletisiGonderildi = 0;
    WATCH(iletiAlindi);
    WATCH(msgBanaAit);
    WATCH(msgBanaAitDegil);
    WATCH(ackIletisiGonderildi);
  if(stage==0){
    EV <<" 0'nci seviye baslangic durumuna getiriliyor.\n";
    headerLength=272;//has to be 272, including final CRC-field; this
    //o oldugunu dogrula!
    maxQueueSize=par("maxQueueSize");
    //zamanlayicilar
    timeout = new cMessage( "timeout", TIMEOUT);
    nav = new cMessage("NAV", NAV);
    contension = new cMessage("contension", CONTENSION);
    endTransmission = new cMessage ("transmission", END_TRANSMISSION);
    endSifs = new cMessage("end SIFS", END_SIFS);
    BW = 0;
    state = IDLE;
    retryCounter = 1;
    broadcastBackoff=par("broadcastBackoff");
    rtsCts=par("rtsCts");
    bitrate=par("bitrate");
    delta = 1E-9;
    EIFS = SIFS +  DIFS + packetDuration(LENGTH_ACK);
    EV <<"SIFS: "<<SIFS<<" DIFS: "<<DIFS<<" EIFS: "<<EIFS<<endl;
  }
  else{
      EV <<" 1'nci seviye baslangic durumuna getiriliyor.\n";
    //tasiyici duyusunun bilgisine yonelik surdurum
    bbRs=blackboard()->subscribe( this, "RadioState");
  }
}

void Mac80211::handleUpperMsg(MacPkt *ma)
{
  Mac80211Pkt *mac = static_cast<Mac80211Pkt *>(ma);
  if(mac->encapsulatedMsg()->length()>18496)
    error(" Ag paketi 802.11b icin cok uzun! Parcalara bolme henuz
desteklenmiyor!!");
```

```
  EV <<" ust msg alindi! DURUM: "<<state<<endl;
  if ((int)fromUpperLayer.size() < maxQueueSize){
    fromUpperLayer.push_back(mac);
    //Eger MAC, IDLE durumda ise, o zaman yeni bir carpisma sureci baslat

    if ( state == IDLE && !endSifs->isScheduled() ) {
      tryWithoutBackoff=true;
      beginNewCycle();
    }
  }
  else{
    delete mac;
    EV <<" Mac kuyrugu doludur; paket silindi!\n";
  }
}

void Mac80211::handleLowerMsg(MacPkt *mac)
{
  Mac80211Pkt *af = static_cast<Mac80211Pkt *>(mac);
  //alis sonu
  phy_receiving=false;//<- probably not needed anymore...
  EV <<" iletim bilgisi alindi.\n";
             // performans olcumu icin "iletiAlindi" degerinin kontrolu
             iletiAlindi++;
             iletiAlindiVector.record(iletiAlindi);
             iletiAlindiStats.collect(iletiAlindi);
  switch ( af->kind() ){
    //paket kaybi ya da bit hatasi
  case COLLISION:
    delete af;
    if (state == CONTEND)
      beginNewCycle();
    break;
  case BITERROR:
    handleMsgNotForMe(af);
    break;
    //broadcast paketi!
  case BROADCAST:
    handleBroadcastMsg(af);
    break;
    //diger paketler
  default:
    if (af->getDestAddr() != myMacAddr() )
      handleMsgNotForMe(af);
    else
      handleMsgForMe(af);
  }
}

/** zamanlayiciyi islet*/

void Mac80211::handleSelfMsg(cMessage *msg)
{
  switch (msg->kind()) {
  case END_SIFS:
    handleEndSifsTimer();
    break;
  case END_TRANSMISSION:
    handleEndTransmissionTimer();
    break;
  case CONTENSION:
    handleEndContensionTimer();
    break;
    //MAC CTS,DATA, veya bir ACK paketi bekliyordu, fakat zamanlayici suresi
bitti
  case TIMEOUT:
    handleTimeOutTimer();
```

```
    break;
    //MAC bekliyordu cunki diger bir haberlesme kanali kullaniyordu. Bu
haberlesme simdi sona erdi
  case NAV:
    handleNavTimer();
    break;
  default:
    error(" bilinmeyen zamanlayici cesidi. ");
  }
}

void Mac80211::handleMsgNotForMe(Mac80211Pkt *af)
{
  EV <<" tanitici msg benim degil.\n";
             // performans olcumu icin "msgBanaAitDegil" degerinin kontrolu
             msgBanaAitDegil++;
             msgBanaAitDegilVector.record(msgBanaAitDegil);
             msgBanaAitDegilStats.collect(msgBanaAitDegil);
  //eger bu paket dogru bir sekilde okunamaz ise
  if (af->kind() == BITERROR)
    af->setDuration( EIFS );
  //paket devami anlamsiz ise bir sey yapma (kendi mesajimizin elverissiz
zamanlamasindan sakinmak icin)
  if (af->getDuration() != 0) {
    //dugum zaten erteleme yapiyor
    if (state == QUIET){
      //NAV'in o anki degeri yeterli degildir
      if (nav->arrivalTime() < simTime() + af->getDuration() ){
       cancelEvent( nav);
       scheduleAt(simTime() + af->getDuration(), nav);
       EV <<af->getDuration()<<" NAV zamanlayicisi HAREKETSIZ (QUIET) Durum
icin baslatildi.\n";
      }
    }
    //diger durumlar
    else{
      //MAC diger bir ileti icin beklerse, onun zaman asimini iptal
edilebilir
      //(exchange is aborted)
      if (timeout->isScheduled() )
       cancelEvent(timeout);
      //durum == WFCTS ya da WFACK'dir, veri aktarimi hata vermistir...
      //dugum iletim zamaninin ertelemek zorundadir
      scheduleAt(simTime() + af->getDuration(), nav);
      EV <<" NAV zamanlayicisi baslatildi, "<<af->getDuration()<<"
HAREKETSIZ degil."<<endl;
      state = QUIET;
    }
  }
  if(!rtsCts){
    if(state==CONTEND){
      if(af->kind()==BITERROR){
       if(contension->isScheduled())
         cancelEvent(contension);
       scheduleAt(simTime()+backoff()+EIFS,contension);
      }else
       beginNewCycle();
    }
  }
  delete af;
}

void Mac80211::handleMsgForMe(Mac80211Pkt *af)
{
  EV <<" Tanitici mesaj benim \n";
             // performans olcumu icin "msgBanaAit" degerinin kontrolu
             msgBanaAit++;
```

```
            msgBanaAitVector.record(msgBanaAit);
            msgBanaAitStats.collect(msgBanaAit);
  switch (state){
    //MAC kaymani RTS ya da carpisma surecinin sonu icin CONTEND ya da IDLE
state:waiting'dadir
  case IDLE:
  case CONTEND:
    //mesaj bir RTS ya da DATA ise
    if (af->kind() == RTS)
      handleRTSframe(af);
    else if(af->kind() == DATA)
      handleDATAframe(af);
    else
      EV <<" handleMsgForMe() IDLE/CONTED icerisinde tuhaf mesaj, darf
das?\n";
    break;
    //mac katmani DATA icin bekliyor
  case WFDATA:
    //ileti data iletisi ise
    if (af->kind() == DATA)
      handleDATAframe(af);
    else
      EV <<" handleMsgForMe() WFDATA icerisinde tuhaf mesaj, darf das?\n";
    break;
    //MAC katmani ACK icin bekliyor
  case WFACK:
    //ileti ACK ise
    if (af->kind() == ACK)
      handleACKframe(af);
    else
      EV <<" handleMsgForMe() WFACK icerisinde tuhaf mesaj, darf das?\n";
    delete af;
    break;
    //MAC CTS icin bekliyor
  case WFCTS:
    //ileti CTS ise
    if (af->kind()==CTS)
      handleCTSframe(af);
    else
      EV <<" handleMsgForMe() WFCTS icerisinde tuhaf mesaj, darf das?\n";
    break;
    //dugum su an erteleme yapiyor, herhangi bir paketi MAC adresi ile
isletemez
  case QUIET:
    delete af;
    break;
    //dugum suan bir ACK ya da BROADCAST paketin iletimini yapiyor
  case BUSY:
    EV <<"HATA, myId:"<<parentModule()->id()<<" penceresinde kullanici
hatasi goruldu."<<endl;
    error(" dugum halen gonderim yapiyor ... alim yapamaz ... fiziksel
katman isini dogru bir sekilde yapiyor mu? ... handleMsgForMe icindeki BUSY
durumu!");
    break;
  default:
    error(" MAC katmaninda bilinmeyen durum.");
  }
}

void Mac80211::handleRTSframe(Mac80211Pkt* af)
{
  //iletiler arasi kisa bir bosluk bekle
  endSifs->setContextPointer(af);
  scheduleAt(simTime() + SIFS, endSifs);
}

void Mac80211::handleDATAframe(Mac80211Pkt* af)
```

```
{
  if(rtsCts)
    //cancel time-out event
    cancelEvent( timeout );
  //bir kopya al
  Mac80211Pkt* copy = (Mac80211Pkt*) af->dup();
  //pakedi ust katmana gecir
  sendUp(af);
  // iletiler arasi kisa bir bosluk bekle
  endSifs->setContextPointer(copy);
  scheduleAt(simTime() + SIFS, endSifs);
}

void Mac80211::handleACKframe(Mac80211Pkt* af)
{
  //zaman asimi olayini iptal et
  cancelEvent( timeout );
  //iletim kabul edilir: long_retry_counter baslangic degerin getir
  retryCounter = 1;
  //onaylandi (kabul edildi) paketini kuyruktan cikart
  Mac80211Pkt* temp = fromUpperLayer.front();
  fromUpperLayer.pop_front();
  delete temp;
  //gonderilecek bir paket varsa ve kanal serbes ise o zaman yeni bir
cekisme sureci baslar
  beginNewCycle();
}

void Mac80211::handleCTSframe(Mac80211Pkt* af)
{
  //zaman asimi olayini iptal et
  cancelEvent( timeout );
  //iletiler arasi kisa bir bosluk bekle
  endSifs->setContextPointer(af);
  scheduleAt(simTime() + SIFS, endSifs);
}

void Mac80211::handleBroadcastMsg(Mac80211Pkt *af)
{
  EV <<" Yayinla \n";
  if (state == BUSY)
    error(" dugum halen gonderim yapiyor ... alim yapamaz ... fiziksel
katman isini dogru olarak yapiyor mu? ... handleBroadcastMsg() icinde");
  else {
    sendUp( af );
    if (state == CONTEND)
      beginNewCycle();
  }
}

void Mac80211::handleEndContensionTimer()
{
  if (state == CONTEND){
    //dugum kanali elde etti, backoff penceresi silinir ve sonraki cekisme
surecinde yeniden hesaplanir
    BW = 0;
    //dugume ozel paket
    if (!nextIsBroadcast){
      if(rtsCts){
       //RTS gonderimi
       sendRTSframe();
       state=WFCTS;
       //updateDisplay(WFCTS);
      }else{
       sendDATAframe();
       state=WFACK;
      }
```

```
      //broadcast paketi
    }else{
      sendBROADCASTframe();
      //kabul edilmesini (onay) beklemeden paketi kuyruktan cikart
      Mac80211Pkt* temp = fromUpperLayer.front();
      fromUpperLayer.pop_front();
      delete(temp);
    }
  }
  else
    error(" CONTEND durum disindaki cekisme zamanlayicisinin sonu ...
olmamali");
}

void Mac80211::handleNavTimer()
{
  if (state == QUIET)
    //gonderilecek bir paket varsa ve kanal serbes ise yeni bir cekisme
sureci baslat
    beginNewCycle();
  else
    error(" HAREKETSIZ (QUIET) durumu disindaki NAV zamanlayicisinin sonu
... olmamali");
}

void Mac80211::handleTimeOutTimer()
{
  //if (state == WFCTS || state == WFACK)testMaxAttempts();
    //gonderilecek bir paket varsa ve kanal serbes ise yeni bir cekisme
sureci baslat
  if (state != QUIET) beginNewCycle();
}

void Mac80211::handleEndSifsTimer()
{
  Mac80211Pkt* frame = (Mac80211Pkt*) endSifs->contextPointer();
  switch (frame->kind()){
  case RTS:
    sendCTSframe(frame);
    break;
  case CTS:
    sendDATAframe();
    break;
  case DATA:
    sendACKframe(frame);
    break;
  default :
    error (" onceden alinmis RTS, CTS, ya da DATA paketinden dolayi sifs
zamanlayicisi bitisi");
  }
  //onceki iletilere artik ihtiyac yok
  delete frame;
}

void Mac80211::handleEndTransmissionTimer()
{
  EV <<" ACK/BROADCAST'in iletimi bitiyor\n";
  if (state == BUSY)
    //gonderilecek bir paket varsa ve kanal serbes ise yeni bir cekisme
sureci baslat
    beginNewCycle();
  else
    error(" BUSY durumu disindaki end_transmission zamanlayicisinin sonu ...
olmamali");
}

void Mac80211::sendDATAframe()
```

```
{
  //zaman asmi programi
  scheduleAt(simTime() + timeOut(DATA, 0), timeout);
  if(!rtsCts)
    //retryCounter arttirimi
    retryCounter++;
  //DATA iletisi gonderimi
  sendDown(buildDATAframe());
  //udurumu ve goruntuyu guncelle
  state = WFACK;
  //updateDisplay(WFACK);
}

void Mac80211::sendACKframe( Mac80211Pkt* af )
{
  //MAC, baska bir cekisme surecine baslamadan once, iletimin sonunu
beklemelidir
  scheduleAt(simTime() +packetDuration(LENGTH_ACK) + delta, endTransmission
);
  //ACK iletisi gonder
  sendDown(buildACKframe(af));
  EV <<" ACK iletisi gonderildi!\n";
             // performans olcumu icin "ackIletisiGonderildi" degerinin
kontrolu
             ackIletisiGonderildi++;
             ackIletisiGonderildiVector.record(msgBanaAitDegil);
             ackIletisiGonderildiStats.collect(msgBanaAitDegil);
  //durumu ve goruntuyu guncelle
  state = BUSY;
  //  updateDisplay(BUSY);
}

void Mac80211::sendRTSframe()
{
  //zaman asimi programi
  scheduleAt(simTime() + timeOut(RTS, 0), timeout);
  //long_retry_counter attriimi
  retryCounter++;
  //RTS iletisi gonder
  sendDown(buildRTSframe());
  //durumu ve goruntuyu guncelle
  state = WFCTS;
  //updateDisplay(WFCTS);??
}

void Mac80211::sendCTSframe( Mac80211Pkt* af )
{
  //zaman asimi programi
  scheduleAt(simTime() + timeOut(CTS, af->getDuration() ), timeout);
  //CTS iletisi gonder
  sendDown(buildCTSframe(af));
  //durumu ve goruntuyu guncelle
  state = WFDATA;
  //  updateDisplay(WFDATA);
}

void Mac80211::sendBROADCASTframe( )
{
  //MAC, baska bir cekisme surecine baslamadan once, iletimin sonunu
beklemelidir
  scheduleAt(simTime() + packetDuration( fromUpperLayer.front()->length() ),
endTransmission );
  //ACK iletisi gonder
  sendDown(buildBROADCASTframe());
  //durumu ve goruntuyu guncelle
  state = BUSY;
  //  updateDisplay(BUSY);
```

```
}

Mac80211Pkt* Mac80211::buildDATAframe()
{
  // kuyrugun onundeki iletinin bir kopyasini gonder
  Mac80211Pkt *frame = (Mac80211Pkt*)(fromUpperLayer.front() )->dup();
  frame->setSrcAddr(myMacAddr());
  frame->setKind(DATA);
  frame->setName("DATA");
  if(rtsCts)
    frame->setDuration ( SIFS + packetDuration(LENGTH_ACK));
  else
    frame->setDuration(0);
  return(frame);
}

Mac80211Pkt* Mac80211::buildACKframe( Mac80211Pkt* af )
{
  Mac80211Pkt* frame = static_cast<Mac80211Pkt *>(createCapsulePkt());
  frame->setName("ACK");
  frame->setKind(ACK);
  frame->setLength(LENGTH_ACK);
  //Alinmis paketin RTS veya DATA'nin dest adresi src adresi olmalidirress
of the RTS or the DATA. Src dugumun adresidir
  frame->setSrcAddr(myMacAddr());
  frame->setDestAddr(af->getSrcAddr());
  frame->setDuration(0);
  return(frame);
}

Mac80211Pkt* Mac80211:: buildRTSframe()
{
  Mac80211Pkt *frame  = new Mac80211Pkt;
  frame->setName("RTS");
  frame->setKind(RTS);
  frame->setLength(LENGTH_RTS);
  //src ve destination adresi kuyruk icindeki iletide kopya edilir
(gonderilen ileti)
  frame->setSrcAddr( ( ( Mac80211Pkt*) fromUpperLayer.front() )-
>getSrcAddr());
  frame->setDestAddr( ( (Mac80211Pkt*) fromUpperLayer.front() )-
>getDestAddr());
  frame->setDuration ( 3*SIFS + packetDuration(LENGTH_CTS) + packetDuration(
fromUpperLayer.front()->length() ) +packetDuration(LENGTH_ACK));
  return(frame);
}

Mac80211Pkt* Mac80211::buildCTSframe( Mac80211Pkt* af )
{
  Mac80211Pkt* frame = new Mac80211Pkt;
  frame->setName("CTS");
  frame->setKind(CTS);
  frame->setLength(LENGTH_CTS);
  //Alinan RTS'nin dest adresi src adresi olmalidir. Src dugumun adresidir
  frame->setSrcAddr(myMacAddr());
  frame->setDestAddr(af->getSrcAddr());
  frame->setDuration (af->getDuration() - SIFS -
packetDuration(LENGTH_CTS));
  return(frame);
}

Mac80211Pkt* Mac80211:: buildBROADCASTframe( )
{
  //kuyrugun onundeki iletinin bir kopyasini gonder
  Mac80211Pkt *frame = (Mac80211Pkt*)(fromUpperLayer.front())->dup();
  frame->setKind(BROADCAST);
  frame->setName("BROADCAST");
```

```
  return(frame);
}

void Mac80211::beginNewCycle()
{
  //bir paketi bir ya da daha fazla kez gondermeye calismadan once,
 // eger max deneme limitine ulasildi mi test et. Vaziye bu ise, paketi sil
 // ve sonrakini gonder.
  testMaxAttempts();
  if (!fromUpperLayer.empty()) {
    //sonraki paket unicast (hosta ozel) ya da broadcast (genel) mi bak
    nextIsBroadcast = ( ( (Mac80211Pkt*) fromUpperLayer.front() )-
>getDestAddr() == BROADCAST_ADDRESS);
    //    print("next is broadcast = "<<nextIsBroadcast);
    //eger kanal serbes ise rastgele bir zamani bekle ve gonderme yap
    if((static_cast<const RadioState *>(bbRs->data()))-
>getState()==RadioState::IDLE){
      //eger kanal IDLE ise son trasmit eden ben degildim
      //data: no backoff
      if(tryWithoutBackoff){
       EV <<" geri cekmeden gondermeye calisiliyor...\n";
       scheduleAt(simTime() + DIFS, contension);
      }
      else{   // backoff!
       scheduleAt(simTime() + backoff() + DIFS, contension);
      }
    }
    tryWithoutBackoff=false;
    //kanal serbes olana kadar bekle
    //simdiki durum: contend (cekisme)
    state = CONTEND;
    EV <<" Su anki seviye: "<<state<<endl;
    //updateDisplay(CONTEND);
  }
  else {
    tryWithoutBackoff=false;
    state = IDLE;
    EV <<" Su anki seviye: "<<state<<endl;
    //  updateDisplay(IDLE); ??
  }
}

double Mac80211::backoff()
{
  //MAC onceki cekismeyi kazandi, yeni bir backoff penceresi hesaplamamiz
gerekli
  if (BW == 0)
    BW = ((double)intrand(CW()+1)) * ST;
  //CW (Contention Window) ceksme penceresidir. ST (Slot Time) dilim
suresidir.
  //Bir onceki cekisme periyodunu kaybeden dugume daha buyuk oncelik vermek
icin
  // BW'nin eski degerini aliyoruz.
  EV <<BW+DIFS<<" icin geri cekiliyor."<<endl;
  return(BW);
}
int Mac80211::CW()
{
  //sonraki paket bir hosta ozel paket tir
  if (!nextIsBroadcast){
    int cw;
    cw = (CW_MIN+1) * (unsigned int) pow(2.0, (int) retryCounter -1) - 1;
    //en buyuk degere ulasilir ise Cwmax ya da hesaplanmis deger geri doner
    if (cw <= CW_MAX) return (cw);
    else return (CW_MAX);
  }
  //sonraki paket broadcats yaplabilir: the next packet is broadcast :
```

```
cekisme penceresi en buyuk olmak zorunda
  else return (broadcastBackoff);
}

void Mac80211::testMaxAttempts()
{
  if (retryCounter > RETRY_LIMIT) {
    //initialize counter
    retryCounter = 1;
    //reportLost(fromUpperLayer.front());
    //to do publish on Blackboard
    //gonderilecek iletiyi sil
    Mac80211Pkt* temp = fromUpperLayer.front();
    fromUpperLayer.pop_front();
    delete(temp);
  }
}

bool Mac80211::blackboardItemChanged( BBItemRef bbItem)
{
  Enter_Method("blackboardItemChanged(\"%s\")", bbItem->label());
  // parcanin onceden ana modul tarafindan ilsletilip isletilmedigini
kontrol et
  if( BasicMacLayer::blackboardItemChanged( bbItem ) ){
    EV <<"bbItem ust sinif tarafindan isletildi -> geri don\n";
    return true;
  }
  if( bbRs == bbItem ){
    EV <<" Parcayi blackboardItemChanged() icinde isletmek zorundayim!\n";
    //alis baslangici
    if ((static_cast<const RadioState *>(bbRs->data()))-
>getState()==RadioState::RECV){
      EV <<" Tasiyici Duyusu: aliniyor!\n";
      phy_receiving = true; //is it needed?
      //eger carpisma periyodu var ise
      if (contension->isScheduled()){
       //sonraki mucadelede daha yuksek oncelik vermek icin backoff
penceresini guncelle
       if(simTime()-contension->sendingTime()>=DIFS){
         BW = contension->arrivalTime() - simTime();
         EV <<" Geri cekme penceresi daha kucuk yapildi, yeni BW (Back off
Window): "<<BW<<endl;
       }
       cancelEvent( contension );
      }
      //eger bir SIFS periyodu var ise
      if(endSifs->isScheduled() ){
       //onceden alinmis iletiyi sil
       delete ( (Mac80211Pkt*) endSifs->contextPointer());
       //sonraki iletimi iptal et
       cancelEvent(endSifs);
       //su an ki durum IDLE ya da CONTEND
       if (fromUpperLayer.empty())
         state = IDLE;
       else
         state = CONTEND;
      }
    }
    EV <<"Su anki seviye: "<<state<<endl;
    // parca isletildi-> true geri donusu oldu
    return true;
  } // end if( bbRs == bbItem )
  // parca benim ile ilgili degil
  EV <<bbItem->label()<<" parcasi benim degil, turetilmis modulun onu
isletmesine izin verelim\n";
  return false;
}
```

```
double Mac80211::timeOut(_802_11frameType type, double last_frame_duration)
{
  double time_out = 0;
  switch (type){
  case RTS :
    time_out = SIFS + packetDuration(LENGTH_RTS) +
packetDuration(LENGTH_CTS) + delta;
    break;
  case CTS :
    time_out = last_frame_duration - packetDuration(LENGTH_ACK) - 2*SIFS +
delta;
    break;
  case DATA :
    time_out = SIFS + packetDuration( fromUpperLayer.front()->length() ) +
packetDuration(LENGTH_ACK) + delta;
    break;
  default :
    EV <<" kullanilmamis ileti turu, timeout() cagirildigi vakit,
verilmisti, bu sekilde olmamali!\n";
  }
  return(time_out);
}

double Mac80211::packetDuration(int bits) {
  return(bits/bitrate+PHY_HEADER_LENGTH/BITRATE_HEADER);
}

void Mac80211::updateDisplay()
{
    char bufstr[90]="";
    ev <<logName()<<" alinmis ileti:"<<iletiAlindi<<" msg
benim:"<<msgBanaAit<<" msg benim degil:"<<msgBanaAitDegil<<" gonderilen
ack:"<<ackIletisiGonderildi<<endl;
    //sprintf(bufstr, "alinan:%d yayinlanan:%d ait olmayan:%d
yayinlanmayan:%d", banaUygunAldim, aldimGeneleYayinladim,
banaAitDegilYenidenYayinla, oncedenSacilmisti);
    //displayString().setTagArg("t",0,bufstr);
    //displayString().setTagArg("i",1,"Muharrem");
}

void Mac80211::finish()
{
    // bu fonksiyon simulasyon sonunda OMNeT++ tarafindan cagirilir.
    ev << logName() << " tarafindan alinan ileti sayisi            : " <<
iletiAlindi << endl;
    ev << logName() << " dugumune ait mesaj sayisi                 : " <<
msgBanaAit << endl;
    ev << logName() << " dugumune ait olmayan mesaj                : " <<
msgBanaAitDegil << endl;
    ev << logName() << " tarafindan gonderilen ACK Iletisi sayisi  : " <<
ackIletisiGonderildi << endl;
    recordScalar("Alinmis Ileti", iletiAlindi);
    recordScalar("Benim msg",msgBanaAit);
    recordScalar("Benim olmayan msg",msgBanaAitDegil);
    recordScalar("Gonderilen ACK",ackIletisiGonderildi);
}
```

ÖZGEÇMİŞ

1968 yılında Kocaeli/Gölcük'te doğdu. İlk, orta ve lise öğrenimini Gölcük'te tamamladı. 1997 yılında girdiği Kocaeli Üniversitesi Kocaeli Meslek Yüksek Okulu Endüstriyel Elektronik Bölümü'nden birincilikle, 1999 yılında, mezun oldu. 1999 yılında girdiği Kocaeli Üniversitesi Mühendislik Fakültesi Elektronik ve Haberleşme Mühendisliği Bölümü'nden, 2004 yılında Elektronik ve Haberleşme Mühendisi olarak, mezun oldu. 1999-2000 yılları arasında ingizlice hazırlık eğitimi aldı. 1986 yılından buyana Deniz Kuvvetleri Komutanlığı'nın çeşitli birimlerinde Elektronik Astsubayı olarak görev yapmakta olup, evli ve iki çocuk babasıdır.