

T.C.

GEBZE YÜKSEK TEKNOLOJİ ENSTİTÜSÜ

MÜHENDİSLİK VE FEN BİLİMLERİ ENSTİTÜSÜ

DAĞITIK SİSTEMLERDE NEDENSEL SIRALI MESAJ
DAĞITIM ALGORİTMALARI VE YENİ BİR YAKLAŞIM

Tolga BAŞTÜRK

YÜKSEK LİSANS TEZİ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

GEBZE

2008

T.C.

GEBZE YÜKSEK TEKNOLOJİ ENSTİTÜSÜ

MÜHENDİSLİK VE FEN BİLİMLERİ ENSTİTÜSÜ

DAĞITIK SİSTEMLERDE NEDENSEL SIRALI MESAJ
DAĞITIM ALGORİTMALARI VE YENİ BİR YAKLAŞIM

Tolga BAŞTÜRK

YÜKSEK LİSANS TEZİ

DANIŞMANI: Yrd. Doç. Dr. Fatih Erdoğan SEVİLGİN

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

GEBZE

2008



**GEBZE YÜKSEK
TEKNOLOJİ ENSTİTÜSÜ**

YÜKSEK LİSANS/DOKTORA JÜRİ ONAY FORMU

G.Y.T.E. Mühendislik ve Fen Bilimleri/Sosyal Bilimler Enstitüsü Yönetim Kurulu'nun 29/01/2008 tarih ve 2008/04 sayılı kararıyla oluşturulan jüri tarafından 22/02/2008 tarihinde tez savunma sınavı yapılan Tolga BAŞTÜRK'ün tez çalışması Bilgisayar Mühendisliği Anabilim Dalında YÜKSEK LİSANS tezi olarak kabul edilmiştir.

JÜRİ

ÜYE (TEZ DANIŞMANI) : Yrd. Doç. Dr. Fatih Erdoğan SEVİLGİN

ÜYE : Yrd. Doç. Dr. H. Ali Mantar

ÜYE : Yrd. Doç. Dr. İlyas Kandemir

ONAY

G.Y.T.E. MÜHENDİSLİK VE FEN BİLİMLERİ ENSTİTÜSÜ Yönetim Kurulu'nun __/__/__ tarih ve __/__/__ sayılı kararı.

İMZA/MÜHÜR

ÖZET

TEZİN BAŞLIĞI :DAĞITIK SİSTEMLERDE NEDENSEL SIRALI MESAJ DAĞITIM ALGORİTMALARI VE YENİ BİR YAKLAŞIM

YAZAR ADI :Tolga BAŞTÜRK

Dağıtık sistemler fiziksel olarak birbirinden ayrı işlem birimlerinin belli bir iletişim ağı kullanarak bir arada çalışmasına olanak sağlayan yapılardır. Bu yapılar genel olarak ortak bir hafıza alanı kullanmazlar ve birbirleri ile mesaj aktarımı (“message passing”) yöntemi ile haberleşirler [1]. Bu sistemlerde mesaj iletiminde oluşan gecikmeler sonlu olsa da tahmin edilebilir değildir ve iletilen mesajların sırasının bozulmasına neden olabilir. Bu durumda olaylar arasındaki neden-sonuç sıraları da bozulabilir ve sistemde mantıksal hatalar oluşmaya başlayabilir.

Dağıtık sistemlerde bu problemi çözmek için, ortak bir sistem saati ile tüm sistem birimlerinin eş zamanlı kılınması ve mesajların neden-sonuç ilişkileri gözetilerek sıralanmaya çalışılması gibi çeşitli yaklaşımlar önerilmiştir. Nedensel sıralı mesajlaşma (NSM) algoritmaları, ortak bir sistem saatine bağlı kalmadan mesajları neden-sonuç ilişkisine göre düzenlemeyi esas alır. Yapılan deneyler bu tür algoritmaların ortak sistem saatlerinin kullanıldığı çözümlere göre daha iyi performans sağladığını göstermiştir [2]. Ancak NSM algoritmalarının mesajlar üzerinde taşıdıkları kontrol bilgileri bazı durumlarda oldukça büyüebilmekte ve ağ üzerinde gereksiz veri yüküne neden olabilmektedir. Bu tez çalışması bünyesinde geliştirilen Merkezi Kontrol Dağıtık Mesajlaşma (MKDM) yaklaşımı ile mesajların birimler arasında doğrudan akmasına izin verilmekle birlikte, mesaj yönetimini sağlamakla görevli yönetici bir birime de mesajı tanımlayan basit bir kontrol bilgisi gönderilerek mesajların sıralanmasını sağlanmaktadır.

Tez kapsamında kullandıkları kontrol bilgisi boyutu bakımından en uygun (“optimal”) NSM algoritmaları olan Modified-Schiper-Eggli-Sandoz (MSES) ve Kshemkalyani-Singhal (KS) algoritmaları ile MKDM algoritmasının teorik ve deneysel analizleri yapılmış, MKDM algoritmasının farklı simülasyon modelleri için diğer algoritmalara göre daha ölçeklenebilir ve daha yüksek performans sağlayan algoritma olduğu gözlenmiştir.

SUMMARY

**THESIS TITLE : CAUSAL ORDER MESSAGE DELIVERY ALGORITHMS
FOR DISTRIBUTED SYSTEMS AND A NEW APPROACH**

AUTHOR : Tolga BASTURK

Distributed systems are composed of distinct processes that do not have a global time or a shared memory. A typical system realizes interoperability of processes by transferring messages (messages passing) via a communication network [1]. In a large communication network, latency in message transmission is unpredictable and the order of messages may change. So cause and effect relation between messages may not be preserved and logical errors may happen in the system. Therefore special algorithms are required to preserve the harmony of “time synchronization” and “data synchronization” in distributed systems to ensure that the right message is received at the right time.

Message ordering techniques based on global time allows processes to run on the same time clock and puts time info called “time-stamp” in each message to order messages. On the other hand, causal order message delivery algorithms do not use global time but use control information both in messages and in processes. This control information is used by the processes to evaluate the causal dependencies of messages. In experiments, optimal casual order algorithms give better performance results than time-stamp techniques [2]. On the other hand, under some situations, causality based methods may require large control information to be appended to the message which can cause latency in network. In this study, a new approach called Central Control Distributed Messaging (CCDM) is proposed. In CCDM, there is a central process to manage message ordering. Simulators send their messages directly to the destinations and also send control information about messages to the central process.

In this study two optimal causal order algorithms called Modified-Schiper-Eggli-Sandoz(MSES), Kshemkalyani-Singhal(KS) and our method CCDM are theoretically and experimentally analyzed in a simulation environment which can simulates a distributed system.

TEŞEKKÜR

Bu çalışmanın tüm aşamalarında engin bilgi birikimini, deneyimini ve ilgisini hiçbir zaman esirgemeyen ve bana destek olan danışman hocam Yrd. Doç. Dr. Fatih Erdoğan Sevilgen'e, nedensel sıralı mesajlaşma algoritmaları üzerindeki çalışmalarını ve edindiği tecrübeyi benimle paylaşan ve pek çok noktada beraber çalışarak her türlü desteği veren İnanç Tahralı'ya, paylaştığı fikirler ve yönlendirmeler ile çalışmamın şekillenmesinde katkısı bulunan TÜBİTAK-MAM Bilişim Teknolojileri Enstitüsü personelinden Ramazan Cengiz'e ve son olarak beni bu günlere getiren aileme ve tezimin hazırlanma sürecinde bana hep destek veren ve anlayışını hiç esirgemeyen Gülen Okyar'a sonsuz teşekkürlerimi sunarım.

İÇİNDEKİLER DİZİNİ

	<u>Sayfa</u>
1. Giriş	1
1.1 Problemin Tanımı ve Genel Bakış	1
1.1.1. Alış Sıralı Mekanizma	1
1.1.2. Öncelik Sıralı Mekanizma	2
1.1.3. Zaman Damgalı Mekanizma	2
1.1.4. Nedensel Sıralı Mekanizma	2
1.1.5. Nedensel ve Tamamen Sıralı Mekanizma	3
1.1.6. Nedensel Sıralı Mesajlaşma Yaklaşımı	3
1.2 Tez Çalışmasının Amaçları	8
1.3 Tez Çalışmasının Katkıları	9
1.4 Genel Bakış	9
2. Nedensel Sıralı Mesaj Dağıtım Algoritmaları	11
2.1. Modified Schiper-Eggli-Sandoz Algoritması	12
2.1.1. Veri Yapıları	12
2.1.2. Mesaj Gönderme Mekanizması	14
2.1.3. Mesaj Alma Mekanizması	15
2.2. Kshemkalyani-Singhal Algoritması	19
2.2.1. Veri Yapıları	20
2.2.2. Mesaj Gönderme Mekanizması	21

2.2.3.	Mesaj Alma Mekanizması	24
2.3.	Merkezi Kontrol Dağıtık Mesajlaşma Algoritması	26
2.3.1.	Veri Yapıları	28
2.3.2.	Mesaj Gönderme Mekanizması	29
2.3.3.	Mesaj Alma Mekanizması	31
3.	Algoritmaların Teorik Analizi	34
3.1.	Mesaj Gönderme Mekanizması	34
3.2.	Mesaj Alma Mekanizması	37
3.3.	Veri Yapıları	37
3.4.	Mesaj Formatı ve Kontrol Bilgisi	39
4.	Algoritmaların Deneysel Analizi	43
4.1.	Mesaj Üretici	43
4.2.	Dağıtık Ortam Simülatörü (DOS)	44
4.3.	Algoritmalarının Gerçeklenmesi	46
4.4.	Deneysel Sonuçlar	47
4.4.1.	Gevşek Bağlı Simülasyonlarda Algoritmaların Kontrol Bilgisi, İşlem Yüğü ve Ağ Gecikmesi Analizleri	48
4.4.2.	Sıkı Bağlı Simülasyonlarda Algoritmaların Kontrol Bilgisi, İşlem Yüğü ve Ağ Gecikmesi Analizleri	55
4.4.3.	Mesaj Alıcılarındaki Değişimin Kullanılan Kontrol Bilgisi ve İşlem Hızına Etkisinin Analizi	60
4.4.4.	Sistemdeki Simülatörlerin Artışına Paralel Olarak Algoritmaların Ölçeklenebilirliğinin Analizi	65

5. Sonular	71
6. neriler	74
7. Kaynaklar Dizini	75
8. zgemiř	76

SİMGELER VE KISALTMALAR DİZİNİ

NSM	:	Nedensel Sıralı Mesajlaşma
MSES	:	Modified Schiper-Eggli-Sandoz (Değiştirilmiş SES)
KS	:	Kshemkalyani-Singhal
SIS	:	Integrated Scientific Information System (Entegre Bilgi Sistemi)
CBCAST	:	Causal Broadcast (Nedensel Tüme Gönderim)
SES	:	Schiper-Eggli-Sandoz
MKDM	:	Merkezi Kontrol Dağıtık Mesajlaşma
CCDM	:	Central Control Distributed Messaging
TSO	:	Time Stamped Order (Zaman Damgalı Sıralama)
VT	:	Time Vector (Zaman Vektörü)
CV	:	Causality Vector (Nedensellik Vektörü)
RTI	:	Real Time Infrastructure (Gerçek Zaman Altyapısı)
DOS	:	Dağıtık Ortam Simülatörü
FIFO	:	First in First out (İlk Giren İlk Çıkar)

XML : Extensible Markup Language (Geniřletilmiř İřaretleme Dili)

SR : Latest Send Received

ŞEKİLLER DİZİNİ

<u>Sekil</u>	<u>Sayfa</u>
1.1: Nedensel Sıralı Mesajlaşma Yaklaşımı.	4
1.2: Doğrudan Bağlantı İzleme Metodu.	7
2.1: MSES algoritması mesaj gönderme komut dizisi.	14
2.2: MSES algoritması mesaj alma komut dizisi.	16
2.3: MSES algoritması mesaj işleme kontrol komut dizisi.	16
2.4: MSES algoritması mesaj alma komut dizisi.	17
2.5: KS algoritmasına ait mesaj gönderme komut dizisi.	22
2.6: KS algoritmasının mesaj alma komut dizisi.	25
2.7: Nedensel sıralı mesajlaşmada FIFO ağ kuralı kullanımı.	26
2.8: İstemci-Sunucu mimarisi.	27
2.9: Simülatörler için mesaj gönderme komut dizisi.	29
2.10: Yönetici simülatöre ait mesaj işleme komut dizisi.	30
2.11: MKDM algoritması mesaj işleme kontrol komut dizisi.	31
2.12: MKDM yönteminin mesaj alma komut dizisi.	33
3.1: Teke gönderim ortamında mesaj akışı.	34
3.2: Çoğa gönderim ortamında mesaj akışı.	35
4.1: Mesaj Üreticine ait yapılandırma dosyası.	44
4.2: DOS yazılımı ağ modeli tanımlama dosyası.	45

4.3: DOS yazılımı için tasarlanan ağ modeli.	46
4.4: Teke gönderimde gevşek bağlı simülasyonda kontrol bilgisi kullanımı.	48
4.5: Teke gönderimde gevşek bağlı simülasyonda işlem yükü.	49
4.6: Teke gönderimde gevşek bağlı simülasyonda her bir simülatör üzerindeki işlem yükü.	50
4.7: Teke gönderim ortamında gevşek bağlı simülasyonda ağ gecikmesi.	51
4.8: Çoğa gönderim ortamında gevşek bağlı simülasyonda kontrol bilgisi kullanımı.	52
4.9: Çoğa gönderim ortamında gevşek bağlı simülasyonda işlem yükü.	52
4.10: Gevşek bağlı simülasyonda her bir simülatör üzerindeki işlem yükü.	53
4.11: Çoğa gönderim ortamında gevşek bağlı simülasyonda ağ gecikmesi.	54
4.12: Teke gönderim ortamında sıkı bağlı simülasyonda algoritmaların analizi.	55
4.13: Teke gönderim ortamında sıkı bağlı simülasyonda işlem yükü.	56
4.14: Sıkı bağlı simülasyonda her bir simülatör üzerindeki işlem yükü.	57
4.15: Teke gönderim ortamında sıkı bağlı simülasyonda ağ gecikmesi.	58
4.16: Çoğa gönderim ortamında sıkı bağlı simülasyonda kontrol bilgisi kullanımı.	58
4.17: Çoğa gönderim ortamında sıkı bağlı simülasyonda işlem yükü.	59
4.18: Çoğa gönderim ortamında sıkı bağlı simülasyonda ağ gecikmesi.	60
4.19: Teke gönderim ortamında mesaj alıcılarındaki değişimin kontrol bilgisine etkisi.	61
4.20: Teke gönderim ortamında mesaj alıcılarındaki değişimin işlemi yüküne etkisi.	62
4.21: Mesaj alıcılarındaki değişimin simülatör başına işlem yükü etkisi.	63

4.22: Mesaj alıcılarındaki değişimin ağ gecikmesine etkisi.	63
4.23: Çoğa gönderim ortamında mesaj alıcılarındaki değişimin kontrol bilgisine etkisi.	64
4.24: Teke gönderim ortamında, karışık simülasyonda kontrol bilgisi kullanımı.	65
4.25: Çoğa gönderim ortamında, karışık simülasyonda kontrol bilgisi kullanımı.	67
4.26: Çoğa gönderim ortamında, karışık simülasyonda işlem yükü.	67
4.27: Çoğa gönderimde, karışık simülasyonlarda, her bir simülatör üzerindeki işlem yükü.	68
4.28: Teke gönderim ortamında, karışık simülasyonda ağ gecikmesi.	69
4.29: Çoğa gönderim ortamında, karışık simülasyonda ağ gecikmesi.	70

ÇİZELGELER DİZİNİ

Cizelge

Sayfa

2.1: KS algoritmasında kullanılan simgeler.

19

1. GİRİŞ

1.1 Problemin Tanımı ve Genel Bakış

Dağıtık bir sistem fiziksel ya da coğrafi olarak birbirinden ayrı pek çok işlem biriminin belli bir iletişim ağı kullanarak bir arada çalışması ile oluşan yapıdır. Bu yapılar genel olarak ortak bir hafıza alanı kullanmazlar ve birbirleri ile sadece mesajlaşarak haberleşme ilkesine dayanan mesaj geçme (“message passing”) yöntemi ile haberleşirler [1].

Dağıtık sistemlerde mesaj trafiğinin düzgün bir şekilde akması hayati derecede önemlidir. Mesajların işlenmesi gereken sıranın dışında işlenmesi yapılan işi tamamen geçersiz kılabilir. Örneğin dağıtık olarak koşan bir savaş simülasyonunda bir savaş uçağının bir düşman tankını vurması senaryosunda (ileride daha detaylı anlatılmaktadır), patlama olayını içeren mesajın vurulma olayını içeren mesajın önüne geçmesi tüm simülasyon akışında bir mantıksal hata oluşturmakta ve simülasyonun devamında oluşabilecek olayların sırasının da bozulmasına neden olabilmektedir. Özellikle analitik simülasyonlar gibi aynı girdiler ile aynı sonuçların çıkmasının beklendiği durumlarda, nedenselliği koruyamamak simülasyonun verdiği sonuçların güvenilirliğini zedeler.

Dağıtık sistemlerde mesajları sıralamak için 5 farklı yaklaşım mevcuttur [3]. Bu yöntemler sırası ile alış sıralı (“received ordered”), öncelik sıralı (“priority ordered”), zaman damgalı sıralı (“time stamped ordered”), nedensel sıralı (“causally ordered”) ve nedensel ve tamamen sıralı (“causally and totally ordered”) mekanizmalarıdır.

1.1.1. Alış Sıralı Mekanizma

Alış sıralı mekanizmada sistemde oluşan mesajlar zaman ile ilişkilendirilmezler. Mesajlar birimlere ulaştığı anda işlenirler. Gerçeklenmesi en kolay mekanizmadır. Birimler kendilerine gelen mesajları bekletmedikleri için hem birimler üzerinde hem de sistem genelinde en az gecikmeye neden olan mekanizmadır. Ancak bu yöntem mesajları nedensel olarak sıralamamaktadır.

1.1.2. Öncelik Sıralı Mekanizma

Öncelik sıralı mekanizmada sistemde akan mesajlar önceliklerine göre sıralanırlar. Mesajların öncelikleri genellikle olaylar ile beraber gönderilen zaman damgaları ile belirlenebilir. Zaman damgası ilgili olayın ne zaman işlenmesi gerektiğini gösteren değerdir. Gerçekleme kararına bağlı olarak fiziksel ya da mantıksal zaman kullanılabilir. Olaylar zaman damgası küçük olandan büyük olana doğru sıralanırlar. Bu yöntemde birimlere kendi yerel zamanlarından öncesine ait olaylar da gelebilir. Bu yöntem de mesajlar arasındaki neden-sonuç ilişkisini gözetmemektedir.

1.1.3. Zaman Damgalı Mekanizma

Zaman damgalı mekanizma, öncelik sıralı mekanizmaya benzerlik göstermektedir. Sistemde akan her olay beraberinde bir zaman damgası taşımaktadır. Bu zaman damgası olayın ne zaman işlenmesi gerektiğini söylemektedir. Her birimin kendi yerel zamanı vardır. Yerel zamanlar sistemdeki ortak saat ile uyumludur. Çoğu gerçeklemede mesajları sıralamak ve birimlerin kendi zamanlarını iletletmelerini kontrol etmek için sunucu görevi gören bir birim vardır. Sistem saati bu birimin kontrolü altındadır. Tüm birimler sunucudan izin alarak zamanlarını iletletirler. Sunucu birimlere kendi yerel zamanlarından önceye ait olayların ulaşmamasından da sorumludur. Bu mekanizma tüm sistemde eşzamanlamayı sağlar. Sistemde gerçekleşen olaylar tüm birimlere aynı sırada iletilir. Bu mekanizmanın dezavantajı ise birimlerin sunucudan izin alarak kendi zamanlarını iletletmek zorunda olmalarıdır. Sistemde zaman ortak olduğu için işini erken bitiren birim sistemin geri kalanı henüz işini tamamlayamadığı durumlarda zamanını iletletemeyebilir ve kendini beklemeye alabilir. Çok sayıda birimden oluşan bir sistemde bu tür gecikmeler çok artabilir ve sistemin performansını düşürebilir.

1.1.4. Nedensel Sıralı Mekanizma

Nedensel sıralı mekanizma zaman damgalı mekanizmanın bu dezavantajını kapatmak için geliştirilmiştir. Sistemde gerçekleşen olaylar neden-sonuç ilişkisine göre sıralanırlar. Sebebi işlenmemiş olayların sonucu da işletilmez ve bekletilir.

Aralarında nedensellik ilişkisi bulunmayan olaylara bağımsız olaylar denir ve bu olaylar birimlere geldikleri anda alış sıralı mekanizmada olduğu gibi hemen işlenebilirler. Nedensel sıralı mesajlaşmada ortak bir sistem saati yoktur. Onun yerine birimlerde yerel saatler vardır. Kullanılan zaman genelde mantıksal zamandır ve birimlerin zamanı bir sayaç ile ilerletilir. Zaman damgalı sıralı mekanizmadan en büyük farkı mesaj sıralama kontrolünün ortak bir sunucudan tüm birimlere kaydırılmış olmasıdır. İleride nedensel sıralama yönteminden daha ayrıntılı bahsedilecektir.

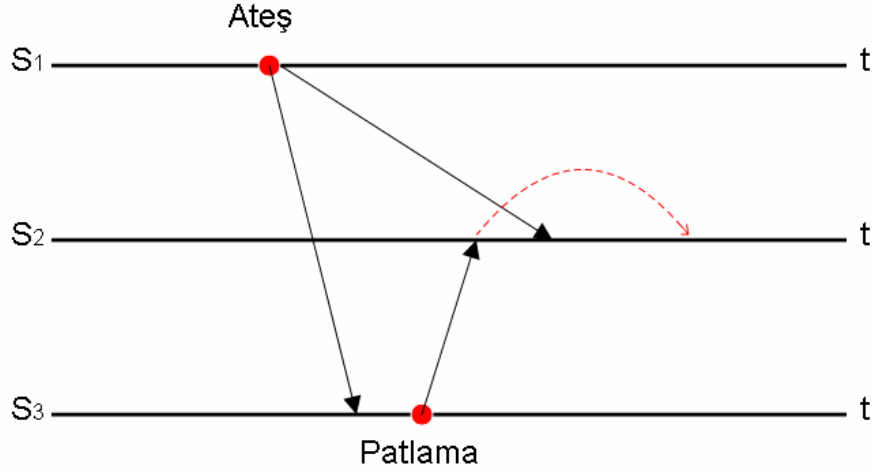
1.1.5. Nedensel ve Tamamen Sıralı Mekanizma

Nedensel ve tamamen sıralı mekanizma, nedensel sıralı mekanizmadan farklı olarak bağımsız mesajların da birimlere gönderildikleri sırada iletilmesini sağlamaktadır. Mesajların nedensellik bağlantısının yanında gönderim sıraları da göz önünde bulundurulur. Tüm birimler gönderilen mesajları aynı sırada alırlar.

Dış dünyada olan olaylar birimler tarafından birbirlerine mesajlar vasıtasıyla aktarıldığı için bu tez çalışmasında olaylar ve mesajlar eş anlamlı olarak kullanılacaktır. Ayrıca bu tez çalışması dağıtık sistemlerde ortak bir sorun olan mesaj sıralama problemini özelde güzel bir örnek teşkil eden dağıtık simülasyon sistemleri üzerinden anlatacak ve yukarıda bahsedilen sistemdeki her bir alıcı birim ise bundan böyle simülatör olarak anılacaktır. Ortaya atılan algoritmalar, çözümler ve yeni yaklaşımlar ise özelde simülasyon sistemlerine genelde de tüm dağıtık sistemlere uygulanabilir.

1.1.6. Nedensel Sıralı Mesajlaşma Yaklaşımı

Nedensel sıralı mesajlaşma (NSM) yaklaşımının temelinde alınan mesajların ne sırada işleneceğine “neden-sonuç” ilişkisini gözeterek karar vermek ve böylece sistemde mantıksal hataların oluşması engellemek yatmaktadır. Şekil 1.1’de nedensel sıralı mesajlaşma yaklaşımı anlatılmaktadır.



Şekil 1.1: Nedensel Sıralı Mesajlaşma Yaklaşımı.

Dağıtık bir sistemde gerçekleşen olaylar bağımsız ve bağımlı olaylar olarak ikiye ayrılır. Bağımsız olaylar, kendi başlarına ortaya çıkabilirler ve işlenmeleri için başka olayların gerçekleşmesi gerekmez. Şekil 1.1'de üç simülatörden oluşan bir simülasyon ortamına ait olay diyagramı görülmektedir. S_1 simülatörü bir savaş uçağını temsil etmekte ve simülasyona ateş olayı yayınlamaktadır. Ateş olayı bağımsız bir olaydır ve S_2 ve S_3 simülatörleri tarafından hemen işlenebilir. Bağımlı olaylar ise önceki olaylardan etkilenen ve o olayların sonucunda ortaya çıkan olaylardır. S_3 simülatörü düşman tankını temsil etmekte ve ateş olayı sonucunda vurularak patlama olayı yayınlamaktadır. S_2 simülatörü ise sistemdeki olayları görselleştirerek yayınlayan bir simülatördür. Şekil 1.1'de görüldüğü gibi ağ üzerinde oluşabilecek bir gecikmeden dolayı S_2 simülatörü patlama olayını ateşleme olayından önce alabilir. Bu durumda S_2 simülatörü tankın önce patladığını gösterecek ardından vurulmasını görselleştirecektir. Böyle bir durumda olayların simülatöre geliş sıralarına göre işlenmeleri simülasyonda mantıksal hatalara neden olacaktır.

Nedensel sıralama yaklaşımına göre henüz nedeni işlenmemiş bir mesaj alıcı simülatöre geldiğinde mesaj bekletilir. Bekletilen mesajlar genellikle düzenli aralıklarla kontrol edilir ve işlenebilir duruma geçen mesajlar işletilir. Şekil 1.1'de, S_2 simülatörü patlama olayını ateşleme olayı kendisine gelinceye kadar bekletir ve ateşleme olayını gerçekleştikten sonra patlama olayını işleterek nedenselliği koruyabilir.

Nedensel sıralama yaklaşımı ilk olarak Lamport tarafından 1978 yılında ortaya atılmıştır [4]. Lamport'un önce olma ("happened before") yaklaşımı ya da nedensellik ilişkisi ("causality relation") ($\rightarrow$) ile belirtilir [4]. İki olay (o_1, o_2) için " $o_1 \rightarrow o_2$ " ilişkisi şu durumlar için doğrudur [5].

1. o_1 ve o_2 aynı işlem biriminde oluşmuştur ve o_1, o_2 'den önce oluşmuştur.
2. o_1 mesajın gönderilmesi ($\text{Send}(M)$), o_2 ise mesajın işlenmesi ($\text{Deliver}(M)$) olaylarıdır.
3. o_3 adında bir olay vardır ki bu olay için $o_1 \rightarrow o_3$ ve $o_3 \rightarrow o_2$ ilişkileri doğrudur.

Nedensellik ilişkisi sadece iki mesaj (M_1, M_2) aynı hedefe gidiyor ise gözetilir. Formül 1.1'de mesajların gönderim sırasına göre işlenmelerinin gerekliliğini tanımlayan nedensellik ilişkisi anlatılmaktadır.

$$(\text{Send}(M_1) \rightarrow \text{Send}(M_2)) \Rightarrow (\text{Deliver}(M_1) \rightarrow \text{Deliver}(M_2)) \quad (1.1)$$

NSM algoritmaları mesajların ağ üzerinde sıralı akmalarını sağlamaya çalışmaz. Onun yerine uç birimler(alıcı birimler) üzerinde çalışırlar ve gelen mesajların ne zaman işleneceğine karar verirler. Ortak bir sunucu sisteme ihtiyaç duymazlar, mesajlar doğrudan alıcılara gönderilir. Bu yönleri ile merkezi bir sunucunun kullanıldığı ve sistemdeki birimlerin zamanlarını iletmemelerini yalnızca sunucu birimin onayını alarak gerçekleştirdikleri mesajlaşma yaklaşımlarına göre çok daha az gecikmeye sebebiyet verirler. Öte yandan birimler artık "Gelen mesajdan önce işlemem gereken bir mesaj var mı?" sorusunu sorabilecekleri bir sunucu olmadığı için nedensellik bilgilerini hem kendi üzerlerinde hem de her mesajın içinde bulundurmak zorunda kalmaktadırlar [6]. Bu nedensellik bilgileri zaman içinde oldukça büyüyeabilmekte ve ağ üzerinde akan mesajların boyutlarını ve ağ trafiğini ciddi oranda attırabilmektedir.

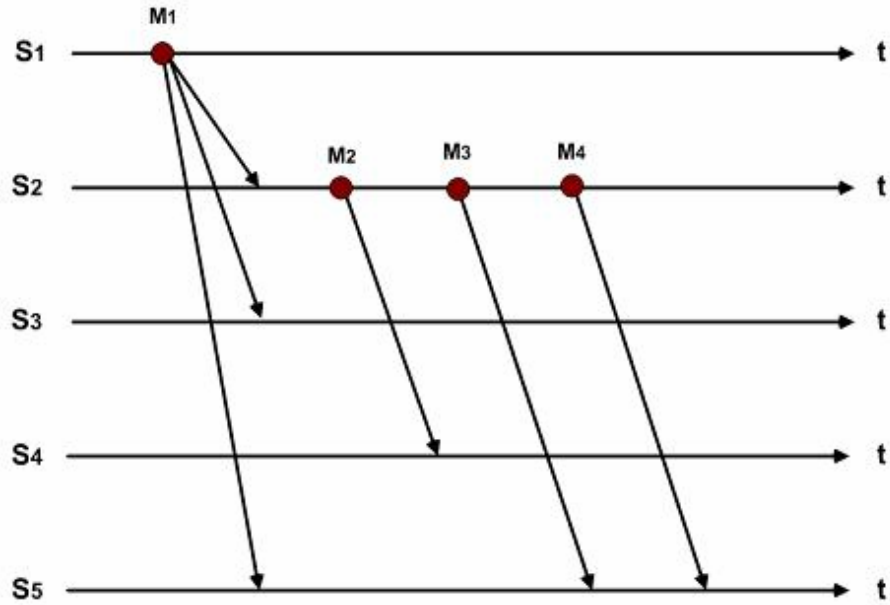
NSM algoritmaları, "Nasıl en az miktarda kontrol bilgisi kullanarak mesajları nedensel olarak sıralayabilirim?" sorusuna cevap aramaktadır. Burada bahsedilen kontrol bilgisi nedensellik bilgisi ve mesajları sıralamak için gerekli olan diğer

bilgilerdir. Literatürde bu konu hakkında çeşitli çalışmalar yapılmış, çeşitli yöntemlerle çözümler aranmıştır [1][2][6][7][8][9].

Kenneth p. Birman öncülüğünde Cornell Üniversitesi’de geliştirilen ISIS sisteminde ilk nedensel sıralı mesajlaşma uygulaması sayılabilecek CBCAST ağ kuralı geliştirilmiştir [10]. İletişim katmanı seviyesinde çalışan bu ağ kuralı, Lamport’un önce olma ilkesine uygun şekilde mesajları nedensellik ilişkilerine göre sıralayabilmektedir. CBCAST ağ kuralı bir birimden diğer birime mesaj gönderirken mesajın nedensel olarak bağlı olduğu henüz işlenmemiş bütün mesajları da bir paket halinde göndermektedir. Örneğin S_1 simülatöründen S_2 simülatörüne M mesajı gönderilecek ise, $M' \rightarrow M$ önermesini doğrulayan bütün M' mesajları da M mesajı ile beraber gönderilir. M' mesajının alıcı listesinde S_2 olmasa bile mesajlar S_2 simülatörüne iletilir. Böylece işlenmemiş mesajlar gerçek alıcılarını bulana kadar birimler arasında kopyalanır ve nedensellik korunmuş olur. Bu yöntem kısa sürede ağ üzerinde akan mesajların boyutlarının artmasına ve mesajlarda gecikmelere neden olmaktadır. Daha sonra geliştirilen SES (Schiper-Eggle-Sandoz) [6] algoritmasında ise, mesajların içine eski mesajları koymak yerine mesajları temsil eden kontrol bilgileri kullanılmıştır ve mesaj boyutları küçültülmüştür. Örneğin N birimden oluşan bir sistemde her mesaj içine N^2 boyutunda kontrol bilgilerini tutan matrisler konulmaya başlanmıştır. Bu matriste her simülatörden her simülatöre gönderilen en son mesajın bilgisi mevcuttur. Bu yapı mesajlar arası nedensellik bilgisini korumak için yeterlidir. Ancak, SES algoritması belli bir mesaj eleme yöntemine sahip olmadığı için, mesaj içinde gönderilen kontrol bilgileri sürekli olarak N^2 olarak kalmaktadır.

Daha sonra geliştirilen ve mesaj içinde ve simülatör üzerinde tutulan kontrol bilgileri bakımından en uygun (“optimum”) olduğu ispatlanmış MSES ve KS algoritmaları ise daha küçük kontrol bilgisi ile çalışabilmek için doğrudan bağlantı izleme (“direct dependency tracking”) metodunu kullanmaktadır. Bu metod en son ata (“immediate predecessor”) yaklaşımı üzerine kurulmuştur. En son ata, bir mesajın kendinden hemen önceki atasını temsil eder. Örneğin bir sistem içinde M_1, M_2 ve M_3 mesajlarının olduğu, $M_1 \rightarrow M_2$ ilişkisi bulunduğu ve $M_1 \rightarrow M_3 \rightarrow M_2$ önermesini sağlayacak bir M_3 mesajının olmadığı koşullar altında, M_1, M_2 ’nin en son atası olarak kabul edilir. Doğrudan bağlantı izleme metodu mesajlar sadece aynı

simülatöre gidiyor ise uygulanabilir. Metodun dayandığı temeller Şekil 1.2’de anlatılmaktadır.



Şekil 1.2: Doğrudan Bağlantı İzleme Metodu.

Sistemde S_1 , S_2 , S_3 , S_4 , S_5 isimli beş simülatör ve M_1 , M_2 , M_3 ve M_4 isimli dört mesaj bulunmaktadır. M_1 , M_3 ve M_4 mesajları S_1 ve S_2 simülatöründen S_5 simülatörüne gönderilmektedir. M_3 ve M_4 mesajlarının M_1 mesajının S_2 simülatörünü etkilemesi sonucunda olduğu düşünülürse mesajların aralarında $M_1 \rightarrow M_3 \rightarrow M_4$ ilişkisi olduğu gözlemlenir. Bu ilişkide nedenselliği koruyabilmek için M_3 mesajı içinde M_1 mesajının bilgisi bulunmalıdır. M_4 mesajı ise M_3 mesajından sonra gerçekleşmekte olduğu için M_1 mesajı M_4 mesajının en son atası değildir. Bu nedenle M_4 mesajı içinde M_1 mesajının bilgisinin bulunması gerekmez. S_5 simülatörü için M_4 mesajının işlenme koşulu, daha önceden M_3 mesajının işlenmiş olmasıdır. Benzer şekilde M_3 mesajının işlenme koşulu ise M_1 mesajının işlenmiş olmasıdır. Bu bağımlılık dolaylı yoldan M_4 mesajının işlenmesini M_1 mesajının da önceden işlenmiş olmasına bağlamaktadır. Doğrudan bağlantı izleme yaklaşımı mesajlar içinde gönderilen kontrol bilgilerinin küçültülmesine yardımcı olur.

Bütün bu iyileştirmelere rağmen hem MSES hem de KS algoritmaları N simülatörden oluşan bir simülasyon sisteminde en kötü şartlarda (“worst case”) $O(N^2)$ büyüklüğünde kontrol bilgisi kullanmaktadır. Bu kontrol bilgisi ağ üzerinde akan her mesaj içine konulmakta ve simülatörler üzerinde de tutulmaktadır.

İstemci-Sunucu mimarili sistemlerde ise mesajları sıralamak için kullanılan servisler sistem üzerinde fazladan yük oluşturmakta, ayrıca işini tamamlayan simülatörlerin de ancak sunucudan izin alarak ilerleyebilmesine imkan tanımaktadır.

Bu tez çalışmasında geliştirilen Merkezi Kontrol Dağıtık Mesajlaşma (MKDM) algoritması NSM algoritmaları gibi dağıtık mesajlaşmaya izin veren ama mesajları sıralamak için istemci-sunucu mimarisinde olduğu gibi bir sunucu kullanan yaklaşımdır. NSM algoritmalarının mesaj içinde gönderdiği kontrol bilgisinden çok daha az kontrol bilgisi kullanan algoritma, sunucu üzerinden de mesajları akıtmadığı için performans düşmesine neden olmaz. Bu çalışmasında iki en uygun nedensel sıralı mesajlaşma algoritması olan MSES ve KS algoritmaları ile tez kapsamında geliştirilen MKDM algoritması dağıtık bir sistemi benzeten bir ortamda gerçekleşmiş, farklı koşullar altında karşılaştırılmış, teorik ve deneysel olarak analiz edilmiştir. Deneyler sonucunda algoritmaların farklı simülasyon modellerinde simülatörler üzerinde oluşturdukları işlem yükleri, mesajlar içinde kullandıkları kontrol bilgisi, ağ üzerinde yarattıkları gecikme miktarı ölçülmüştür.

1.2 Tez Çalışmasının Amaçları

Bu tezin genel amacı nedensel sıralı mesajlaşma algoritmaları üzerinde literatür araştırması yapmak ve bu algoritmalara kullandığı kontrol bilgisi ve ağ üzerinde oluşturduğu gecikme bakımından alternatif olabilecek yeni bir algoritma geliştirmeye çalışmaktır. Bu tez kapsamında temel olarak ele alınan problem, dağıtık bir sistemde kontrol bilgisi ihtiyacı, birimler üzerindeki işlem yükü ve ağ gecikmesi bakımından mevcut algoritmalarından daha etkin mesaj sıralama algoritması geliştirmeye çalışmaktır. Bu amaç doğrultusunda yapılması hedeflenenler şunlardır:

- Literatürde bulunan çözümleri ortaya koymak ve bu problemlerin çözüm yöntemleri hakkında genel bir fikir edinmek,
- Bahsedilen yöntemlerin hangilerinin bu problem için daha iyi sonuçlar verdiğini deneysel olarak araştırmak,

- Elde edilen bilgiler çerçevesinde daha az kontrol bilgisi kullanan, birimlere daha az işlem yükü getiren ve ağ üzerinde daha az gecikmeye sebebiyet veren bir yöntem geliştirmeye çalışmak ve analiz etmek.

1.3 Tez Çalışmasının Katkıları

Tezin temel katkıları aşağıdaki gibi sıralanabilir:

- Bu konuda çalışmalar yapacak araştırmacılara kaynak sağlamak,
- Literatürde teorik olarak en uygun olduğu ispatlanmış algoritmaların, gerçekleştirilebilirliğini ve deneysel sonuçlara göre gerçek performanslarını ölçmek
- Geliştirilen yeni bir algoritma ile çözüme yönelik ilerleme sağlamak

1.4 Genel Bakış

Bu çalışmada öncelikle nedensel sıralı mesajlaşma algoritmalarının teorik alt yapısı ile ilgili bilgiler verilmekte ve ardından yapılan uygulamalar anlatılmaktadır. Son bölümlerde yapılan deneyler ve sonuçları değerlendirilmektedir.

Bölüm 1’de bu tez kapsamında çözüm aranan dağıtık sistemlerde mesajlaşma problemi tanımlanmış ve bu konu ile alakalı yapılan literatür araştırması hakkında bilgi verilmiştir. Bölüm 2’te kullanılan çözüm yöntemleri hakkında bilgilendirme yapılmaktadır. Bu yöntemler MSES ve KS algoritmaları ile tez çerçevesinde sunulan yeni bir algoritma olup, bu algoritmalar hakkında gerekli açıklamalar bu kısımda verilmektedir.

Bölüm 3’te MSES, KS ve yeni MKDM algoritmalarının teorik analizleri yapılmaktadır. Algoritmaların mesaj gönderme, mesaj alma mekanizmaları, mesaj formatları, kullandıkları kontrol bilgileri ve veri yapıları irdelenmektedir.

Bölüm 4’te algoritmalarının deneysel analizi yapılmaktadır. Algoritmalar dağıtık bir sistemi benzeten sanal bir ortamda gerçekleştirilmiş ve ağ üzerinde oluşturdukları mesaj yüküne, işlem hızlarına, ölçeklenebilirliklerine ve teke veya

çoğa gönderim ortamlarında ve gevşek veya sıkı bağlı sistemlerdeki performanslarına göre incelenmiş ve analiz edilmiştir.

Bölüm 5 ve Bölüm 6’da yapılan çalışmanın sonuçları değerlendirilmekte ve ileride yapılabilecek çalışmalar hakkında önerilerde bulunmaktadır.

2. NEDENSEL SIRALI MESAJ DAĞITIM ALGORİTMALARI

Dağıtık sistemlerde simülatörlerden gönderilen mesajlar iki farklı yöntem ile nedensel olarak sıralanabilir. Bu yöntemler; dağıtık simülatörler veya ortak bir kontrol mekanizması üzerinden mesajları sıralamaktır. MSES ve KS algoritmalarının temelini oluşturan neden-sonuç ilişkisine göre sıralama yönteminde, sistemde akan mesajların içine diğer mesajlar ile nedensel olarak ilişkilerini tanımlayan kontrol bilgileri konur. Bu kontrol bilgileri alıcı simülatörler tarafından yorumlanır ve mesaj uygun şartlar sağlandığında işletilir. Nedensel sıralı mesajlaşma yöntemi simülatörlerin ortak bir kontrol mekanizmasına bağlı kalmadan mesajlaşabilmesine imkan tanır. Bu yöntemin dezavantajı ise mesajlar içine konulan kontrol bilgilerinin bazı durumlarda çok büyüyebilmesi ve ağda gecikmeye sebebiyet verebilmesidir. Bu tez kapsamında incelenecek olan MSES ve KS algoritmaları bu yöntemi kullanan nedensel sıralı mesajlaşma algoritmalarıdır.

Diğer bir mesaj sıralama yöntemi ise mesajları ortak bir kontrol mekanizması kullanarak sıralamaktır. İstemci-sunucu mimarisini de anımsatan bu yaklaşımda kontrol mekanizması çoğu zaman mesajları sıralamak ile görevli ayrı bir simülatörde çalıştırılır. Sistemdeki simülatörler mesajlarını sunucu simülatör üzerinden alıcılara iletirler. Mesajlar, sunucu simülatör tarafından sıralanır ve alıcıların doğru mesajları doğru zamanda alması sağlanır. Mesajları sıralamak için kullanılan yöntem mesajların içine zaman pulu koymaktır. Zaman pulu mesajın hangi zamanda atıldığını gösterir. Sistemde ortak bir sistem saati vardır ve sunucu bu sistem saatine dayanarak mesajları zamanı küçük olandan büyük olana doğru sıralar. Özellikle simülasyon dünyasında kullanılan RTI [11] gibi dağıtık sistem altyapı çerçeve yazılımlarının günümüz gerçeklemelerin pek çoğu bu gruba girmektedir. Bu yöntemin dezavantajı ağ üzerinde akan yüksek sayıdaki mesajın sunucu simülatör üzerinde yük oluşturabilmesi ve sunucunun cevap verme zamanında (“response-time”) oluşabilecek gecikmeden dolayı sistemin yavaşlayabilmesidir.

Bu tez kapsamında önerilen yeni yaklaşım yukarıda bahsedilen iki yaklaşımı da kullanan nedensel sıralı mesajlaşma yöntemidir. Merkezi Kontrol Dağıtık

Mesajlaşma yönteminde, mesajlar NSM algoritmalarında olduğu gibi doğrudan alıcı simülatörlere gönderilir. Aynı zamanda mesajın numarası ve alıcı listesinden oluşan kontrol bilgisi de sunucu simülatöre iletilir. KS ve MSES gibi NSM algoritmalarına göre çok küçük kontrol bilgisi kullanan bu algoritma, basit bir sıralama işlemi yapan sunucu üzerinde de en alt seviyede işlem yükü oluşturmaktadır. Bu tez kapsamında MSES, KS ve önerilen Merkezi Kontrol Dağıtık Mesajlaşma algoritması tanımlanacak, teorik ve deneysel olarak analiz edilecektir.

2.1. Modified Schiper-Eggli-Sandoz Algoritması

MSES algoritması SES algoritması üzerine geliştirilmiştir [2]. Yukarıda bahsedilen doğrudan bağlantı izleme metodunu kullanarak mesaj içindeki ve simülatörler üzerindeki gereksiz kontrol bilgilerini azaltmaya çalışır. Gereksiz kontrol bilgileri mesajlar arası nedenselliği korumak için zorunlu olmayan bilgilerdir. Bu eleme yaklaşımına rağmen algoritma en kötü şartlarda (worst case), N simülatörün bulunduğu bir ortamda $O(N^2)$ 'lik kontrol bilgisine ihtiyaç duyar. Ortalama durumlarda (average case) ise bu oran çok daha düşüktür.

2.1.1. Veri Yapıları

MSES algoritması mesaj içinde ve simülatörler üzerinde kontrol bilgisi bulundurur. Mesaj içindeki kontrol bilgisi o mesajın hangi mesajlara nedensel olarak bağlı olduğunu ve gönderen simülatörün simülasyon hakkındaki bilgisini barındırır. Simülatörler üzerindeki kontrol bilgisi ise ilgili simülatörün diğer simülatörler ile ilgili bildiği güncel bilgileri barındırır.

MSES algoritması zaman vektörü (VT) ve nedensellik vektörü (CV) olmak üzere iki veri yapısı kullanır. VT ve CV vektörleri simülatörler üzerinde tutulur ve ayrıca her bir mesaja da eklenerek gönderilir.

MSES algoritmasında ortak sistem saati kullanılmamaktadır. Onun yerine her simülatörün mesaj gönderiminde ve mesaj alımında arttırdığı bir sayaç kullanılır. Bu sayaçlar TSO mesajlaşma yaklaşımında kullanılan zaman değişkenlerinden farklıdır. TSO yaklaşımında tüm simülasyonda geçerli olan ortak bir saat vardır ve simülatörler bu saate uyarak ve genelde aynı zaman aralığında ilerlerler. MSES

algoritmasında ise her simülatörün zaman sayacı mesaj gönderdiğinde ve aldığıda artacak şekilde tasarlandığı için diğer simülatörlerden farklılık gösterebilir. Zaman vektörü ilgili simülatörün simülasyondaki diğer simülatörlerin sayaçlarını tuttuğu yerdir. Yeni gelen mesajların işlenip işlenemeyeceğinin kontrolünde kullanılır. Bu kontrol sonucunda alıcı simülatörler gelen mesajın nedensel olarak bağımlı olduğu ata mesajları işleyip işlemediklerine ve mesajı bekletip bekletmeyeceklerine karar verirler. VT vektörünün yapısı aşağıda tanımlanmıştır.

$$VT_{s_1} = \left[(T_{s_1}), (T_{s_2}), (T_{s_3}) \right] \quad (2.1)$$

N simülatörden oluşan bir simülasyonda VT vektörü N 'lik bir dizi olarak tanımlanır. Dizinin her elamanı ilgili simülatörün bilinen en son sayaç değerini göstermektedir.

MSES algoritmasında nedensellik vektörü CV , mesajlar arası bağımlılıkları tanımlamada kullanılan veri yapısıdır. MSES algoritmasının atası sayılan SES [6] algoritması ile beraber nedensellik vektörlerinin içine mesajlar yerine mesaj imzaları konulmaya başlanmıştır. Bir mesajı tanımlamak için kullanılan veri kümesi olarak ifade edilebilecek mesaj imzası, nedensellik vektörlerinin küçülmesinde büyük rol oynamıştır. İmzaların yapıları farklı algoritmalarda farklı tanımlanmıştır. SES algoritmasında her simülatörün kendini diğer simülatörlere tanıtmak için kullandığı benzersiz bir numarası vardır. İmzalar mesajı gönderen simülatörün numarası ve gönderim yaptığı sayaç değerini barındıran (k,x) ikililerinden oluşturulur. Daha önce geliştirilen SIS sistemine ait CBCAST [10] algoritmasında her mesaj beraberinde kendinden önce gönderilmiş mesajları da taşıma yaklaşımının yarattığı sınırsız büyüyeabilen mesajların yerini boyutları belli olan mesajlar almıştır. SES algoritması, nedensellik vektörünün içine her simülatörden her simülatöre gönderilen en son mesajların imzalarını koymaktadır. Dolayısıyla N simülatörden oluşan bir simülasyon için mesajlar ile beraber N^2 'lik mesaj imzası ya da kontrol bilgisi kullanmaktadır.

MSES algoritmasının kullandığı CV vektörünün yapısı aşağıda tanımlanmıştır.

$$CV_{S_2} = \left\{ \underbrace{\{(k, x), (), \dots, ()\}}_{S_1}, \underbrace{\{\}}_{S_2}, \underbrace{\{\}}_{S_3}, \dots, \underbrace{\{\}}_{S_N} \right\} \quad (2.2)$$

CV vektörü, N simülatörden oluşan bir simülasyon ortamında her simülatör için en kötü şartlarda N 'e kadar büyüeyebilen vektörlerden oluşan vektördür. Dolayısıyla en kötü şartlarda (worst case), nedensellik vektörü $O(N^2)$ 'lik mesaj imzası barındırır. Ancak MSES algoritması, mesaj alma mekanizmasında kullandığı doğrudan bağlantı izleme eleme yöntemi ile nedensellik vektörünün boyutunu normal şartlarda çok daha küçük tutabilmektedir.

2.1.2. Mesaj Gönderme Mekanizması

MSES algoritmasının mesaj gönderme mekanizmasının komut dizini Şekil 2.1'de verilmiştir.

```
// Si simülatörü M mesajını gönderiyor
//(1) Si simülatörünün zaman vektörünü güncelle
VTSi[i] = VTSi[i] + 1
//(2) Mesajın kontrol bilgisini hazırla
VTM = VTSi
CVM = CVSi
//(3)(M, VTM, i, Dest(M), CVM) mesajını Dest(M) içindeki bütün alıcılara gönder
//(4) Si simülatörünün nedensellik vektörünü güncelle
for all Sk ∈ Dest(M)
    CVSi[k] = {(i, VTM[i])}
```

Şekil 2.1: MSES algoritması mesaj gönderme komut dizisi.

MSES algoritması, mesaj gönderim mekanizması dört adımda incelenebilir.

1. S_i simülatörü kendi zamanını bir arttırır. Algoritma zaman artırımını sadece mesaj atarken ve mesaj alırken gerçekleştirir.

2. Mesajın içine konulacak kontrol bilgisi hazırlanır. MSES algoritması mesaj içine kontrol bilgisi olarak hem VT hem de CV vektörlerini koymaktadır. Bu vektörler ise S_i simülatörünün kontrol bilgilerinin üzerinde değişiklik yapılmadan kullanılmasıyla oluşturulur.
3. Mesaj alıcılara gönderilir. Gönderilen mesajın formatı $(M, VT_M, i, Dest(M), CV_M)$ şeklindedir. Burada M , mesajın içeriği, VT_M mesajın zaman vektörünü, i mesajı gönderen simülatörün benzersiz numarasını, $Dest(M)$ alıcı simülatörlerin listesini ve CV_M 'de mesajın nedensellik vektörünü temsil eder. Bu mesaj tüm alıcı simülatörlere gönderilir. MSES algoritması tüm alıcı simülatörlere aynı kontrol bilgisini gönderdiği için hem teke gönderim, hem de çoğa gönderim ortamlarında çalışabilir.
4. S_i simülatörü kendi nedensellik vektörünü günceller. Yeni gönderilen mesajı temsil eden imza tanımı oluşturulur, nedensellik vektörü içindeki alıcı simülatörleri temsil eden alanlara eklenir. Böylece alıcı simülatörlerin S_i simülatöründen aldığı en son mesajın bilgisi nedensellik vektörüne kaydedilmiş olur. Bu bilgi ileride gönderilecek mesajlar içine konur ve yeni mesajların bu mesaj işlenmeden işlenmemesi gerektiği bilgisi alıcılara iletilir.

MSES algoritmasının mesaj gönderim mekanizması oldukça basittir. Algoritmanın en uygun olmasını sağlayan kontrol bilgileri üzerindeki gereksiz bilgilerin silinmesi işlemleri mesaj alma mekanizmasına kaydırılmıştır. Bu yaklaşım mesaj gönderim mekanizmasını basitleştirmesine rağmen gereksiz kontrol bilgilerinin de ağ üzerinde akmasına neden olabilmektedir.

2.1.3. Mesaj Alma Mekanizması

MSES algoritmasının mesaj alma mekanizmasının komut dizimi Şekil 2.2'de anlatılmaktadır. Mesaj alma mekanizmasında öncelikle mesajın işlenebilir bir mesaj olup olmadığını kontrol edilir. Mesaj sadece bağımlı olduğu ata mesajlar işlenmiş ise işlenebilirdir. Şekil 2.3'te mesaj işleme koşulu anlatılmaktadır. P_j simülatörü kendisine gelen yeni mesajı ancak nedensel olarak işlemesi gereken hiç mesaj yok ise ya da gelen mesajların hepsini işlemiş ise alıp işleyebilir. Aksi halde mesaj için

bellekte yer ayrılır ve mesaj beklemeye alınır. Şekil 2.2’de adı geçen *Buff* bekletilecek mesajları saklamak için ayrılmış bellek alanıdır.

```
//  $P_j$  simülatörü  $(M, VT_M, i, Dest(M), CV_M)$  mesajını alıyor
ifCanBeProcessed( $CV_M$ )
then
    //(1) mesajı işle
    Receive( $M$ )
    //(2) bekletilen mesajlardan artık işlenebilir olanlarını işle
    for all  $(M', VT_M, i', Dest(M'), CV_{M'}) \in Buff_{S_i}$ 
        if CanBeProcessed( $M'$ )
            Receive( $M', VT_M, i', Dest(M'), CV_{M'}$ )
else
    //(3)  $(M, VT_M, i, Dest(M), CV_M)$  mesajını  $Buff_{S_i}$  içine ekle
```

Şekil 2.2: MSES algoritması mesaj alma komut dizisi.

```
// $P_j$  simülatörü  $M$  mesajını alıyor.
Procedure CanBeProcessed( $CV_M$ )
    //(1)Eğer  $P_j$  simülatörün nedensel olarak işlemesi
    //gereken mesaj yok ise veya hepsini işlemiş ise
    // $M$  mesajını işle
    if  $(CV_M[j] = NULL) \vee ((\forall (k, x) \in CV_M[j])(x \leq VT_{P_j}[k]))$ 
        then return true;
    else
        return false;
```

Şekil 2.3: MSES algoritması mesaj işleme kontrol komut dizisi.

Mesajın işlenmesi daha önceden gelmiş ve beklemeye alınmış mesajlardan bazılarının da işlenebilir olmasını sağlayabilir. Bu tür bir bağımlılık olup olmadığının kontrolü için bekletilen mesajlara teker teker bakılır ve işlenebilir duruma gelmiş mesajlar için mesaj işleme komut dizisi çalıştırılır. Mesaj işleme komut dizisi Şekil 2.4’te anlatılmıştır. MSES algoritması, mesaj işleme mekanizması dört adımda incelenebilir.

```

//  $P_j$  simülatörü,  $M$  mesajını ( $M, VT_M, i, Dest(M), CV_M$ ) işliyor
Procedure Receive( $M, VT_M, i, Dest(M), CV_M$ )
  //(1) Mesajın iletileceği bütün hedef simülatörler için
  //nedensellik vektörünü güncelle
  for all  $P_k \in Dest(M)$ 
     $CV_{P_j}[k] = CV_{P_j}[k] \cup \{(i, VT_M[i])\}$ 
     $CV_{P_j}[k] = CV_{P_j}[k] -_{\max} \{(l, VT_M[l]) \mid l = 1 \dots N, l \neq i\}$ 
  //(2)  $P_i$  simülatörü için nedensellik vektörünü güncelle
   $CV_{P_j}[i] = CV_{P_j}[i] -_{\max} \{(l, VT_M[l]) \mid l = 1 \dots N\}$ 
  //(3) Sistemdeki diğer simülatörler için nedensellik vektörünü güncelle
  for all  $P_k \notin Dest(M) \wedge (P_k \neq P_i) \{$ 
    //  $P_j$  simülatörüne ait nedensellik vektöründeki eski bilgileri sil
    for all  $(l, x) \in CV_{P_j}[k]$ 
      if  $(x \leq VT_M[l]) \wedge ((l, x) \notin CV_M[k])$ 
        then  $CV_{P_j}[k] = CV_{P_j}[k] - \{(l, x)\}$ 
    //Mesaja ait nedensellik vektöründeki eski bilgileri sil
    for all  $(l, x) \in CV_M[k]$ 
      if  $(x \leq VT_{P_j}[l]) \wedge ((l, x) \notin CV_{P_j}[k])$ 
        then  $CV_M[k] = CV_M[k] - \{(l, x)\}$ 
    //  $P_j$  simülatöre ve mesaja ait nedensellik vektörlerini birleştir
     $CV_{P_j}[k] = CV_{P_j}[k] \cup_{\max} CV_M[k]$ 
  }
  //(4)  $P_j$  simülatörüne ait zaman vektörünü güncelle
   $VT_{P_j}[j] = VT_{P_j}[j] + 1$ 
  for all  $k = 1 \dots N, VT_{P_j}[k] = \max(VT_{P_j}[k], VT_M[k])$ 
}

```

Şekil 2.4: MSES algoritması mesaj alma komut dizisi.

1. P_j simülatörü mesajın gönderildiği diğer simülatörlerin kendi nedensellik vektöründeki ilgili dizinlerini günceller. Güncelleme işlemini yeni mesajın imzasını girerek ve artık gereksiz oldukları anlaşılan eski imzaları silerek gerçekleştirir. Gereksiz mesajları silme işleminde kullandığı $-_{\max}$ operatörü aşağıdaki şekilde tanımlanır.[2]

$$\begin{aligned}
& T_1 -_{\max} T_2 \\
& \text{if } \exists (i, x) \in T_1, (i, y) \in T_2 \wedge x \leq y \text{ then } T_1 = T_1 - \{(i, x)\}
\end{aligned} \tag{2.3}$$

Burada T_1 ve T_2 , simülatörün ve mesajın nedensellik vektörleridir. İki vektör içinde aynı simülatörden gönderilmiş mesaj imzaları bulunur ise doğrudan bağlantı izleme yaklaşımına göre zamanı küçük olan mesaj imzası ilgili vektörden çıkarılır.

2. P_j simülatörü bu adımda mesajı gönderen simülatörün nedensellik vektörü içindeki ilgili dizinini birinci adımdaki yaklaşım ile günceller.
3. Mesajı almayacak simülatörlerin P_j simülatörüne ait nedensellik vektörü içindeki dizinlerinin güncellemesi bu adımda gerçekleştirilir. Güncelleme işleminde mesaj içinde gelen CV ile simülatör üzerindeki CV kullanılır. İki nedensel vektör karşılaştırılırken bir vektörde olup diğerinde olmayan ve zamanlarının zaman vektörü ile karşılaştırıldığında eski olduğu anlaşılan mesaj imzalarının geçerliliklerini kaybettiklerine kanaat getirilir ve ilgili dizinlerinden silinirler. Hem simülatör içindeki hem de mesaj içindeki nedensellik vektörleri işlendiği bilinen imzalarının çıkartılmasından sonra $U_{\max}$ operatörü ile birleştirilir. $U_{\max}$ operatörü aşağıdaki şekilde tanımlanır.

$$\begin{aligned}
& T_1 \cup_{\max} T_2 \\
& T_1 = T_1 \cup T_2 \\
& \text{if } \exists (i, x) \in T_1, (i, y) \in T_2 \wedge x \leq y \text{ then } T_1 = T_1 - \{(i, x)\}
\end{aligned} \tag{2.4}$$

Bu operatör önce T_1 ve T_2 vektörlerini T_1 vektörü üzerinde birleştirir ve ardından iki vektör üzerinde $-_{\max}$ operatörünü uygular. $-_{\max}$ operatörü doğrudan bağlantı izleme yaklaşımına göre aynı simülatörden aynı alıcıya gönderilen mesajlardan sadece en sonuncusunun mesaj imzasının saklanması sağlar.

4. Bu adımda P_j simülatörüne ait zaman vektörü güncellenir. Simülatörünün kendisine ayrılmış dizininin değeri bir arttırılır. MSES algoritmasında zaman sadece mesaj alındığında ya da gönderildiğinde arttırılır. Zaman vektöründeki

diğer simülatörlerin zamanlarını gösteren alanlarda mesaj içinde gelen zaman vektörü ile karşılaştırılarak en güncel olan seçilmek kaydı ile güncellenir.

2.2. Kshemkalyani-Singhal Algoritması

KS algoritması Ajay D. Kshemkalyani ve Mukesh Singhal tarafından 1996 yılında önerilmiş nedensel sıralı mesajlaşma algoritmasıdır [1]. MSES algoritması gibi doğrudan bağlantı izleme yöntemini kullanmaktadır. MSES algoritmasından farklı olarak kontrol bilgisinin küçültülme işlemleri hem mesaj gönderim hem de mesaj alım mekanizmalarına dağıtılmıştır. Böylece gönderilecek mesaj içindeki kontrol bilgisi de en az tutulmaya çalışılmıştır. Kontrol bilgisini en az tutabilmek için alıcılara özel mesaj hazırlama yaklaşımını benimseyen KS algoritması bu yönü ile teke gönderim ortamlarına göre tasarlanmıştır. Çoğa gönderim ortamlarında ise aslında alıcı simülatör sayısı kadar mesaj oluşturarak yine teke gönderim ortamındaymış gibi çalışmaktadır. MSES algoritması ise teke ve çoğa gönderim ortamlarında çalışabilir.

Tablo 2.1: KS algoritmasında kullanılan simgeler.

Simgeler	Tanımı
$clock_i$	i simülatörüne ait zaman sayacı
$M_{i,a}$	i simülatörü tarafından a zamanında gönderilen M mesajı
(i, a)	i simülatörü tarafından a zamanında gerçekleşen olay
$M_{i,a}.Dests$	i simülatörü tarafından a zamanında gönderilen M mesajının gönderileceği hedef simülatörler
$o_{i,a} = (i, a, Dests)$	Mesajı temsil eden mesaj imzası. i ; kaynak simülatör, a ; mesajın gönderim zamanı, $Dests$; hedef simülatörler
O_M	M mesajı ile birlikte gönderilen bütün mesaj imzaları

KS algoritması da MSES algoritması gibi N simülatörden oluşan bir simülasyonda en kötü durumda $O(N^2)$ 'lik kontrol bilgisi kullanır. Ortalama

durumlarda ise bu miktar çok daha azdır. Tablo 2.1’de KS algoritmasında kullanılan simgelerin kısa tanımları bulunmaktadır.

2.2.1. Veri Yapıları

KS algoritması, MSES algoritması gibi hem mesaj içinde hem de simülatörler üzerinde kontrol bilgisi bulundurur. Mesaj içindeki kontrol bilgisi o mesajın hangi mesajlara nedensel olarak bağlı olduğu bilgisini barındırır. Simülatörler üzerindeki kontrol bilgileri ise simülatörlerin tüm simülasyon hakkında en son bildiklerini barındırır.

KS algoritması, MSES algoritmasına benzer şekilde zaman vektörü (SR) ve nedensellik vektörü (LOG) kullanır. KS algoritmasında nedensellik vektörü simülatör üzerindeki günlük olarak anılır.

KS algoritmasında kullanılan zaman vektörü, simülasyon ortamındaki her bir simülatör için ilgili dizinde bir alan barındıran vektördür. Bu hali ile MSES algoritmasında kullanılan zaman vektörü ile aynıdır. Ancak hazırlanışı ve içinde barındırdığı bilgi bakımından MSES algoritmasında kullanılan zaman vektöründen farklıdır. MSES algoritmasındaki VT vektörü bir simülatörün gözünden simülasyondaki tüm simülatörlerin bilinen en son zamanlarını tutarken, KS algoritmasındaki SR vektörü, bir simülatörün kendisine diğer simülatörlerden gelen en son mesajların zamanlarını tutar. Başka bir deyişle $SR_j[k]$, j simülatörüne k simülatöründen gelen en son mesajın zamanını göstermektedir. SR vektörü simülatöre gelen mesajların bekletilip bekletilmeyeceğinin kontrolünde kullanılır. N simülatörden oluşan bir simülasyonda SR vektörünün büyüklüğü $O(N)$ ’dir.

KS algoritmasında kullanılan nedensellik vektörü, MSES algoritmasında olduğu gibi alınan mesajın işlenebilirliğini test etmede kullanılan kontrol bilgisini barındırır. KS algoritmasında nedensellik vektörüne günlük (LOG) denir. Günlük içinde MSES algoritmasında olduğu gibi ata mesajlara ait imzalar bulunur. KS algoritmasında kullanılan mesaj imzaları, MSES’te kullanılanlardan farklıdır. P_j simülatörüne ait günlüğün yapısı aşağıdaki gibidir.

$$LOG_j = \{o_{i,a}, o_{k,b}, \dots, o_{s,t}\} \quad (2.5)$$

Her bir $o_{i,a}$ bir mesaj imzasıdır ve $o_{i,a} = (i,a,Dests)$ şeklinde ifade edilir. Bu imzada, i kaynak simülatör, a mesajın kaynak simülatöre göre gönderim zamanı, $Dests$ de, mesajı alacak olan hedef simülatörlerin listesidir. KS algoritmasının mesaj imzası uzun hedef simülatör listesi bulunan mesajlar için MSES algoritmasının kullandığı ikili mesaj imzalarına göre daha az yer kaplamaktadır. Örneğin k alıcılı bir mesaj, MSES algoritmasında k tane ikili ile yani $2k$ tam sayı ile ifade edilirken KS algoritmasında $k+2$ tam sayı ile ifade edilebilir.

2.2.2. Mesaj Gönderme Mekanizması

KS algoritmasının mesaj gönderme mekanizması, MSES algoritmasının mesaj gönderme mekanizmasından gönderilen mesaj sayısı ve hazırlanan kontrol bilgisi bakımından farklıdır. KS algoritmasında, her alıcı simülatör için ayrı mesaj hazırlanır. Buradaki amaç mesaj içinde gönderilen kontrol bilgisini en alt seviyede tutabilmektir. Her simülatörün yeni gönderilecek mesajı işlemek için farklı bağımlılıkları vardır. Alıcı simülatörlerin farklı bağımlılıklarına göre farklı nedensellik vektörü oluşturulur ve mesaj ile beraber gönderilir. Teke gönderim ortamları için ağ üzerindeki kontrol bilgisini azaltan bu yaklaşım, çoğa gönderim ortamları için tam tersine fazladan kontrol bilgisi akışına neden olabilmektedir.

KS algoritmasının mesaj gönderme mekanizmasının komut dizimi 5 adım altında Şekil 2.5'te anlatılmaktadır.

1. Kaynak simülatör zamanını ilerletir. KS algoritmasında simülatörler zamanlarını sadece mesaj gönderimi sırasında ilerletirler. Simülatörlerin kullandığı SR vektörleri, diğer simülatörlerden kendilerine gelen en son mesajların zamanlarını tutmaktadır. Dolayısıyla SR vektörü dış dünyadaki simülatörlerin hangi zaman diliminde olduklarının kaydını yapmaktan ziyade en son hangi mesajların alındığını bilmek için kullanılır.

2. MSES algoritması yukarıda bahsedildiği gibi mesaj gönderim mekanizmasında zaman (VT) ve nedensellik (CV) vektörlerini mesajın içine koymakta ve tüm hedef simülatörlere aynı mesajı göndermektedir.

```

//Pi simülatörü M mesajını gönderiyor
//(1)Pi simülatörü kendi zamanını günceller
clocki ← clocki + 1;
//(2)Her hedef simülatöre ayrı mesaj gönder
for all d ∈ M.Dests do
    OM ← LOGi
    for all o ∈ OM do
        //Hedef listesini küçült
        if d ∉ o.Dests then
            o.Dests ← (o.Dests \ Dests);
        if d ∈ o.Dests then
            o.Dests ← (o.Dests \ Dests) ∪ {d}
        //Hedef listesi boşalmış olanları sil
    for all os,t ∈ OM do
        if os,t.Dests = 0 ∧ (∃ os,t' ∈ OM | t < t')
            then OM ← OM \ {os,t}
        send(i, clocki, M, Dests, OM) to d;
//(3)M mesajının bağımlılıkları işlendiğinde, işlenmiş olacağı garanti olan
dolaylı bağımlılıkları sil
for all l ∈ LOGi do l.Dests ← l.Dests \ Dests;
//(4)Günlüğü yeni gönderilen mesaj için güncelle
LOGi ← LOGi ∪ {(i, clocki, Dests)};
//(5)Hedefindeki bütün simülatörler tarafından işlenmiş veya
//aynı simülatörlerden gelen yeni mesajlar ile eskimiş imzaları sil
PURGE_NULL_ENTRIES(LOGi)

```

Şekil 2.5: KS algoritmasına ait mesaj gönderme komut dizisi.

KS algoritması ise, kaynak simülatörün günlük bilgisi üzerinde her bir hedef simülatör için gerekli eleme yöntemi ile boyut küçültmek ayrı mesajlar göndermektedir. Günlüğü küçültmede kullandığı yöntem, mesajın gönderileceği hedef simülatörlerin numaralarını eski mesaj imzalarından silmektir. Hedef simülatörler yeni gönderilen mesajı işleyebilmek için eski mesaj imzalarının

temsil ettiği bağımlılıkları da yerine getirmiş olmalıdırlar. Dolayısıyla yeni mesaj işlendiği zaman, önceki mesaj imzaları ilgili hedef simülatörler tarafından işlenmiş olacaktır. Bu varsayım üzerinden giden KS algoritması örneğin q , p ve r simülatörlerine mesaj gönderecek ise q simülatörüne gönderdiği mesaj içindeki ata mesaj imzalarından p ve r 'yi çıkarmaktadır. Çünkü bu simülatörler de bu mesajı ileride işleyecek ve dolayısıyla ata mesajları da işlemiş olacaklardır. O halde bu bilgiyi şimdiden kullanarak q simülatörüne gidecek nedensellik bilgisini küçültmek mümkündür. Benzer uygulama p ve r simülatörlerine gönderilen nedensellik vektörlerine de uygulanır. Bu yaklaşım avantaj ve dezavantajlara sahiptir. En önemli avantajı mesaj başına gönderilen kontrol bilgisini küçültmek ve ağ üzerinde akan bilgi miktarını azaltmaktır. Dezavantajı ise çoğa gönderim ortamlarında hala her simülatöre ayrı mesaj göndermek gerektiğinden toplamda çok daha fazla mesaj atılmasına neden olabilmekte ve ağ trafiğine ve kaynak tüketimine yol açabilmektir. KS algoritmasında mesajın yapısı aşağıdaki gibidir.

$$(i, clock_i, M, Dests, O_M) \quad (2.5)$$

i kaynak simülatörün numarasını, $clock_i$ kaynak simülatörün mesaj gönderim zamanını, M mesajın kendisini, $Dests$ hedef simülatör listesini ve O_M alıcıya özelleştirilmiş günlük bilgisini temsil etmektedir. Son olarak hedef listesi boşalmış veya aynı simülatörden gelen daha yeni mesaj imzası ile eski duruma düşmüş mesaj imzaları da günlük içinden silinir. İlgili hedef simülatöre özel hazırlanmış günlük mesaj içine konur ve ilgili alıcıya gönderilir.

3. Kaynak simülatör kendisinde tuttuğu orijinal günlük üzerinde de yeni gönderilen mesajın etkisini gerçekler. Yeni mesajı işleyebilen hedef simülatörlerin, kesinlikle bu mesajın atası olan ve önceden işlemeleri gereken mesajları da işlemiş olmaları gerekmektedir. Bu yaklaşım ile kaynak simülatör günlüğündeki mesaj imzalarından hedef simülatörleri çıkarır.
4. Yeni mesajın imzası günlüğe eklenir.
5. Hedef listesi sıfır olan veya eski bilgi taşıdığı anlaşılan mesaj imzaları günlükten silinir.

KS algoritması mesaj göndermeden önce gerçekleştirdiği eleme yöntemi ile mesaj içinde gönderdiği günlük bilgisini çok önemli oranda küçültmektedir.

2.2.3. Mesaj Alma Mekanizması

KS algoritması kontrol bilgisi eleme yöntemlerini hem mesaj gönderme hem de mesaj alma mekanizmalarında kullanır. Algoritmanın mesaj alma mekanizması beş adımda incelenebilir. Mesaj alma mekanizmasını tanımlayan komut alma dizisi Şekil 2.6'da gösterilmiştir.

1. Bu adımda MSES algoritmasında olduğu gibi mesajın işlenebileceği ya da bekletilmesi gerektiği kararı verilir. Mesaj içinde gelen günlük bilgisi içindeki bütün imzalar incelenir. Bu imzaların içinde, j simülatörünün işlemediği mesaj olup olmadığı SR zaman vektörü üzerinden kontrol edilir. SR_j vektörü ilgili simülatörlerden j simülatörüne gönderilen en son mesajların zamanlarını tuttuğu için bu vektör ile mesaj imzalarının gönderim zamanları karşılaştırılarak mesajların daha önceden işlenilip işlenilmediği kontrol edilebilir. MSES algoritmasından farklı olarak KS algoritmasında bekletilmesi gereken mesajların depolanabileceği bir hafıza alanı tanımlanmamıştır. Dolayısıyla algoritmanın kullandığı tıkayarak bekleme ("blocking") yöntemi, yeni gelen mesajların nasıl işleneceğini anlatamamaktadır. Algoritmaların deneysel analizi safhasında KS algoritması gerçeklenirken bu yaklaşım MSES algoritmasında olduğu gibi bekletilmesi gereken alanları hafızaya atıp, mesaj gönderme-alma işlevlerine devam edecek şekilde değiştirilerek gerçekleştirilmiştir.

```

//(1)Eğer mesajın nedensel olarak bağlı olduğu mesajlar işlenmemiş
//ise işleninceye kadar bekle
for all  $o_{m,t_m} \in O_M$  do
    if  $j \in o_{m,t_m}.Dests$  wait until  $t_m < SR_j[m]$ ;
//(2)Mesajı al, kaynak simülatörün mesaj gönderim zamanını kaydet
Deliver  $M$ ;  $SR_j[k] \leftarrow t_k$ ;
//(3)Yeni mesaj imzasını kaydet, simülatöre önceden gönderilmiş mesajların
//imzalarından kendisini çıkar
 $O_M \leftarrow \{(k, t_k, Dests)\} \cup O_M$ 
    for all  $o_{m,t_m} \in O_M$  do
         $o_{m,t_m}.Dests \leftarrow o_{m,t_m}.Dests \setminus \{j\}$ 
//(4)Mesaj içinde gelen günlük bilgisi ile simülatöre ait günlük bilgilerini
//eski bilgileri atarak ve aynı kaynaklı bilgilerden güncel olanları alarak
//güncelle ve birleştir
for all  $o_{m,t} \in O_M$  and  $l_{s,t'} \in LOG_j$  such that  $s = m$  do
    if  $t < t' \wedge l_{s,t} \notin LOG_j$  then mark  $o_{m,t}$  for deletion
    if  $t' < t \wedge o_{m,t'} \notin O_M$  then mark  $l_{s,t'}$  for deletion
for all  $l_{s,t'} \in LOG_j$  and  $o_{m,t} \in O_M$ , such that  $s = m \wedge t' = t$  do
     $l_{s,t}.Dests \leftarrow l_{s,t}.Dests \cap o_{m,t}.Dests$ 
    Delete  $o_{m,t}$  from  $O_M$ ;
     $LOG_j \leftarrow LOG_j \cup O_M$ ;
//(5)Hedefindeki bütün simülatörler tarafından işlenmiş veya
//aynı simülatörden gelen yeni mesajlar ile artık eskimiş imzaları sil
PURGE_NULL_ENTRIES( $LOG_j$ )

```

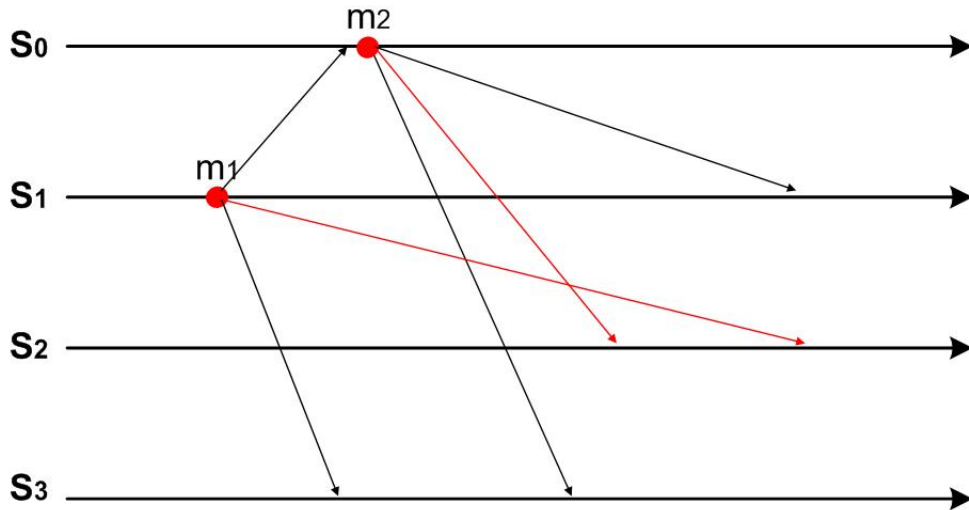
Şekil 2.6: KS algoritmasının mesaj alma komut dizisi.

2. Bu adımda mesaj işlenir ve SR_j zaman vektöründe, mesajın gönderim zamanı kaynak simülatörün ilgili dizindeki alanına girilerek güncellenir.
3. Mesaj içinde gelen günlük bilgisi güncellenir. Öncelikle mesaj günlük bilgisi yeni mesajın imzası da eklenerek güncellenir. Birinci adıma dayanarak, j simülatörü yeni mesajı işleyebildiği için aslında mesaj içinde gelen ve j simülatörünün işlemesi gereken ata mesajları da işlemiş olması gerekmektedir. Bu sonuç çerçevesinde mesaj içinde gelen mesaj imzalarının hedef simülatörler listelerinden j simülatörü çıkarılır.

4. Bu adımda mesaj günlük bilgisi ile simülatör günlük bilgisi karşılaştırılır, eski ve gereksiz olduğu anlaşılan mesaj imzaları karşılıklı olarak silinir ve en sonunda iki günlük birleştirilerek simülatör günlüğü güncellenir.
5. Son adım elde edilen günlük içinde olabilecek gereksiz ve eski mesaj imzaların silinmesi safhasıdır. Mesaj imzalarından, hedef listesindeki tüm simülatörler tarafından işlenmiş olanlar bu adımda günlük içinden silinir.

2.3. Merkezi Kontrol Dağıtık Mesajlaşma Algoritması

Dağıtık bir sistemde bir simülatörden başka bir simülatöre gönderilen mesajların sırası uygun ağ iletişim kuralı kullanarak korunabilir. FIFO ağ iletişim kuralı, ağ üzerinde aynı kaynak simülatörden aynı hedef simülatörüne gönderilen mesajların gönderildikleri sırada karşı tarafa ulaşmasını garanti eder. FIFO, iki simülatör arasındaki mesaj akışını koruyabilmesine karşın nedensel sıralı mesajlaşmayı garanti etmez. Şekil 2.7’de FIFO ağ kuralına göre çalışan bir ağ ortamında mesajların nedensel olarak sıralarının bozulabileceği gösterilmiştir.



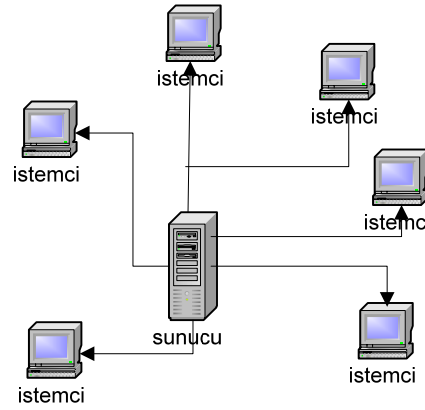
Şekil 2.7: Nedensel sıralı mesajlaşmada FIFO ağ kuralı kullanımı.

S_1 simülatöründen atılan m_1 mesajı S_0 , S_2 ve S_3 simülatörleri tarafından alınmakta ve cevap olarak S_0 simülatöründen gönderilen m_2 mesajı S_1 , S_2 , S_3 simülatörlerine iletilmektedir. Yukarıdaki durumda m_1 mesajı m_2 mesajının atasıdır ve aralarında $m_1 \rightarrow m_2$ ilişkisi vardır. FIFO ağ kuralı aynı kaynak simülatörden aynı

alıcıya giden mesajları sıralayabilirken, farklı kaynak simülatörlerden aynı alıcıya giden mesajları sıralayamaz. Şekil 2.7’de S_0 ve S_1 simülatörlerinin gönderdiği m_1 ve m_2 mesajları S_2 simülatörüne herhangi bir sırada ulaşabilir.

FIFO ağ kuralı tek başına dağıtık bir sistemde mesajları nedensel olarak sıralamak için yeterli değildir ancak mesajların üzerinden aktığı ortak bir mekanizma ile birlikte mesaj sıralamakta kullanılabilir.

İstemci-sunucu mimarilerinde; ağ üzerinde akan mesajları sıralamak, neden-sonuç ilişkisine dayanarak mesaj sıralamaktan daha kolaydır. Bu yöntemde istemci simülatörler kendi içlerinde kontrol bilgisi barındırmazlar ve mesaj içine de kontrol bilgisi eklemesler. Mesajların sıralanma sorumluluğu tamamen bu iş ile özelleşmiş bir sunucu vasıtasıyla gerçekleştirilebilir. İstemci sunucu yapısında, FIFO ağ kuralının kullanımı ile mesajlar sıralı olarak gönderilebilir. Şekil 2.8’de örnek bir istemci-sunucu mimarisi gösterilmektedir.



Şekil 2.8: İstemci-Sunucu mimarisi.

İstemci-sunucu mimarisi ve FIFO ağ kuralı ile birlikte Şekil 2.7’de bahsedilen mesaj sıralama problemi çözülebilir. S_1 simülatörü tarafından gönderilen m_1 mesajı, sunucu üzerinden S_0 , S_2 , S_3 alıcılarına iletilir. S_0 simülatörü üzerinden gönderilen m_2 mesajı da önce sunucuya ardından diğer alıcılara gittiği için, m_2 mesajı m_1 mesajından önce S_2 simülatörüne ulaşamaz. Çünkü her iki mesajı gönderen de artık sunucudur ve sunucudan S_2 simülatörüne giden mesajların sırası FIFO ağ kuralı ile sağlanmaktadır. İstemci sunucu yaklaşımı ile mesaj sıralama problemini çözmek mümkün olsa da, pratikte kullanım alanı kısıtlıdır. Sistemde çok sayıda simülatör

olduğunda ya da mesaj akışının çok yoğun olduğu durumlarda tüm mesajlar sunucu üzerinden aktığı için, sunucu tüm sistemde bir darboğaza neden olacak ve mesaj iletiminde gecikmeler yaşanacaktır.

Bu tez kapsamında ortaya konan Merkezi Kontrol Dağıtık Mesajlaşma (MKDM) yöntemi istemci-sunucu mimarisinde olduğu gibi bir sunucu barındıran ama mesajların nedensel sıralı mesajlaşma algoritmalarında olduğu gibi doğrudan alıcı simülatlörlere iletildiği karma bir modeldir. Bu yaklaşımda simülatlörler birbirleri ile doğrudan haberleşirler ve birbirlerine gönderdikleri mesaj içinde mesaj numarası dışında bir kontrol bilgisi barındırmazlar. Kaynak simülatlörden başka simülatlörlere gönderilen her mesaj için sadece mesaj yönetimi ile sorumlu yönetici simülatlöre bir kontrol mesajı gönderilir. Kontrol mesajı içinde mesajın ve alıcı simülatlörlerin tanımlayıcı numaraları bulunmaktadır. Bu bilgiler yönetici simülatlörün ilgili alıcı simülatlörlere, kendilerine gönderilmiş mesajı ne zaman işlemeleri gerektiğini söylemesi açısından yeterlidir. Mesajlar simülatlörler arasında doğrudan gönderildiği için, yönetici simülatlör üzerinde en az seviyede yük oluşur. MSES ve KS algoritmasında mesajlar içine en kötü şartlarda (worst case), N simülatlörün bulunduğu bir ortamda $O(N^2)$ 'lik kontrol bilgisi konulmaktadır. MKDM yönteminde ise yönetici simülatlöre gönderilen kontrol bilgisi $O(N)$ 'den yani simülatlör sayısından büyük olamaz.

2.3.1. Veri Yapıları

MKDM algoritmasında, simülatlörler arasında gönderilen mesajlarda kontrol bilgisi olarak sadece mesajın benzersiz numarası gönderilir. Simülatlörler ile yönetici arasında akan kontrol mesajlarında ise mesajın ve alıcı simülatlörlerin tanımlayıcı numaraları gönderilmektedir. Yönetici simülatlör ise her bir alıcı simülatlöre mesaj numarası ve mesaj işleme zamanından oluşan yeni bir kontrol mesajı göndermektedir. Simülatlörler kendi içlerinde kendi zamanlarını gösteren bir zaman değişkenine sahiptir. Her simülatlörün güncel zamanları da sadece yönetici simülatlörde bir zaman vektörü içinde tutulmaktadır. Bu zaman vektörü MSES algoritmasında kullanılan VT ve KS algoritmasında kullanılan SR vektörü ile aynı yapıdadır ancak MSES algoritmasında olduğu gibi her simülatlörlerde değil, sadece yönetici simülatlörde bulunmaktadır ve mesajlar içinde gönderilmemektedir.

2.3.2. Mesaj Gönderme Mekanizması

KS ve MSES algoritmaları, sistemdeki en güncel bilgilerin tüm simülatörler tarafından bilinmesi fikri üzerine kurulmuştur. Böylece simülatörler dış dünya hakkında fikir sahibi olabilecek ve kendilerine gelen mesajları ne zaman işlemeleri gerektiğine karar verebileceklerdir. Karar verebilmelerini mümkün kılmak için, her gönderilen mesaj içine sistem hakkındaki en güncel bilgiler konulmaya çalışılır. Bu güncel bilgiler, çoğu zaman simülatörlerin bilinen en son sanal zamanları ve henüz işlenildiği bilinmeyen ya da yeni işlenmiş olan mesajları tanımlayan mesaj imzalarıdır. Her mesaj içine bu kontrol bilgilerinin konulması ağ üzerinde kaynak tüketimine neden olmaktadır. Simülatörlerin gelen kontrol bilgilerini yorumlaması ve kendi veri yapılarını güncellemesi ayrıca iş yüküne de neden olabilmektedir.

MKDM algoritmasında sistem hakkındaki bilgiler tüm simülatörler tarafından değil tek bir simülatör tarafından bilinmektedir. Simülatörler birbirleri ile doğrudan mesajlaşabilmekte, mesaj hakkındaki asgari bilgiyi ise mesaj yönetimini sağlamak ile görevli simülatöre iletmektedirler. Bu yaklaşım ile boyutu sistemdeki simülatör sayısına ve mesaj yoğunluğuna bağlı olarak artan nedensellik bilgisinin yerine mesaj ve alıcı simülatörlerin numaralarından oluşan daha basit ve küçük sıralama bilgileri ağ üzerinde akmaktadır. Mesajlar doğrudan yönetici simülatör üzerinden akmadığı için ağ üzerinde bir darboğaz olma ihtimali azalmaktadır. Şekil 2.9’da bir simülatörün mesaj gönderme komut dizisi gösterilmektedir.

```
//Pi simülatörü mesaj ( $k, M$ ) gönderiyor.
//(1) ( $k, M$ ) mesajını bütün alıcılara gönder
//(2) Yönetici simülatöre sıralama mesajını ( $k, Dests$ ) gönder
```

Şekil 2.9: Simülatörler için mesaj gönderme komut dizisi.

Simülatöre ait mesaj gönderme komut dizisi oldukça basittir. MSES ve KS algoritmalarında olduğu gibi nedensellik vektörü ve zaman vektörünün hazırlanması işlemleri mevcut değildir. MKDM algoritmasında iki farklı mesaj vardır. İlki simülatörlerin birbirleri arasında doğrudan gönderdikleri ve orijinal mesajı barındıran haberleşme mesajı diğeri ise yönetici simülatör ile diğer simülatörler arasında

kullanılan sıralama mesajıdır. Mesaj gönderme mekanizması iki adımda incelenebilir.

1. P_i simülatörü alıcı simülatörlere haberleşme mesajını, tanımlayıcı mesaj numarası ile birlikte gönderir. Mesajı içindeki k , mesajın numarasını, M de asıl mesajı temsil etmektedir.
2. Yönetici simülatöre yeni gönderilen mesaj hakkında sıralama mesajı gönderilir. Sıralama mesajının içeriğini temsil eden $(k, Dests)$ 'te k mesajın numarasını, $Dests$ ise alıcı simülatörlerin listesini temsil etmektedir. Yönetici simülatör kendi içinde tuttuğu ve tüm simülatörlerin zamanlarını gösteren zaman vektörüne dayanarak ilgili alıcılara numarası belirtilen mesajı ne zaman işleyebileceklerini söyleyecektir.

```
//Yönetici simülatör alıcılara kontrol mesajı  $(k, t)$  gönderiyor.
//Kaynak simülatörden gelen sıralama mesajından alıcı simülatörleri bul
//(1) Herbir alıcı simülatörün zaman bilgisini bul ve simülatöre ilgili
//mesajı işleyeceği zamanı yeni bir sıralama mesajı olarak gönder
for all  $d \in Dests$ 
     $t = ZV[d]$ 
    send message  $(k, t)$  to  $d$ 
//(2) simülatöre ait zamanı güncelle
 $ZV[d] = t + 1$ 
```

Şekil 2.10: Yönetici simülatöre ait mesaj işleme komut dizisi.

Şekil 2.10'da yönetici simülatörün mesaj işleme komut dizisi anlatılmaktadır. Yönetici simülatör kendi içinde tüm simülatörlerin zamanlarını gösteren zaman vektörüne (ZV) sahiptir. Simülatörler kendi aralarında mesajlaştıklarında, bu mesajı tanımlayan kontrol bilgisini de yönetici simülatöre gönderirler. 1 Yönetici simülatör kendisine gelen sıralama mesajı içinde gerçek mesajı alacak olan simülatörleri bulur ve onların en son zamanlarını kullanarak, yeni mesajı bu zamanda işlemeleri yönünde her birine yeni sıralama mesajı gönderir. Yeni gönderdiği sıralama mesajı (k, t) formatındadır. k orijinal mesajın numarası, t ise alıcı simülatörün bu mesajı işlemesi gerektiği zamandır. Her simülatör kendi zaman diliminde ilerleyebilir. Zamanların ilerlemesi tamamen akan mesaj sayısına bağlıdır. Yönetici simülatör son

olarak ilgili alıcı simülatörlerin zamanlarını ilerletir. Böylece yeni gelecek mesaj için yeni mesaj alım zamanlarını tanımlamış olur.

MKDM algoritmasında, simülatörler üzerindeki hesap yükü yönetici simülatöre taşınmış olmasına karşın, bu simülatör üzerinde yapılan iş oldukça basittir. Mesajların bir kontrol mekanizması üzerinden akması sıralı gitmesini sağlamak için yeterlidir. NSM algoritmalarının aksine bu yaklaşımda yönetici simülatör üzerinden sadece zaman vektörünü kullanarak mesajları sıralamak mümkündür.

2.3.3. Mesaj Alma Mekanizması

MKDM algoritmasında simülatörler iki tip mesaj alırlar. Bunlardan ilki simülasyon hakkında kendi içlerinde haberleşmek ve veri transferi için kullandıkları haberleşme mesajları, ikincisi ise yönetici simülatörün gönderdiği tamamen ilk tip mesajları sıralamak için kullanılan sıralama mesajlarıdır. Bu iki tip mesaj simülatörlere herhangi bir sırada gelebilir. Sıralama mesajının önce gelmesi durumunda, mesaj içinde gelen mesaj numarası ve işlenme zamanı haberleşme mesajı gelip işleninceye kadar kalmak üzere bir vektörde kaydedilir. Bu vektöre sıralama vektörü(SV) denir. Haberleşme mesajının önce gelmesi durumunda da mesaj, kendisine ait kontrol mesajı gelinceye ve uygun işlenme koşulu oluşuncaya kadar saklanır. Şekil 2.11’de simülatörlere ait mesaj işleme kontrol komut dizisi gösterilmiştir.

```
//Pj simülatörü  $M(k, Data)$  mesajını alıyor
Procedure CanBeDelivered( $M$ )
    //Eğer yönetici simülatör tarafından gönderilen mesaj
    //işlenme zamanı simülatörün zamanına eşit ise mesajı işle
    if( $SV[k] == simTime$ )
        return true;
    else
        return false;
```

Şekil 2.11: MKDM algoritması mesaj işleme kontrol komut dizisi.

Simulatorlerde haberleşme mesajının işlenmesi koşulu, mesaja ait kontrol bilgisinin gelmiş olması ve işlenmesi için belirtilen zamanın simülörün o anki zamanına denk düşmesidir. Henüz zamanı gelmemiş mesajlar ileride tekrar kontrol edilmek üzere saklanır.

Mesaj alma komut dizisi 5 adımda incelenebilir. Şekil 2.12’de bu adımlar gösterilmektedir.

1. Simülöre iki tip mesaj gelebilir. Bunlar haberleşme mesajları ve sıralama mesajlarıdır. İlk adımda simülöre gelen mesajın tipi kontrol edilir. Sıralama mesajları ve haberleşme mesajları simülörde farklı değerlendirilirler.
2. Sıralama mesajı geldiği zaman, mesaj içindeki orijinal mesajın benzersiz numarasını gösteren sayı ve işleme zamanı sıralama vektörüne kaydedilir.
3. Yeni gelen sıralama bilgisinin beklemeye alınmış haberleşme mesajlarından birine ait olup olmadığı kontrol edilir. Eğer ait olduğu haberleşme mesajı bulunursa mesajın işlenip işlenilemeyeceği kontrol edilir. Bir simülöre hem haberleşme hem de sıralama mesajları gelse bile bu mesajın işlenmesi için yeterli değildir. Mesajdan önce işlenmesi gereken başka mesajlar olabilir. Bunun kontrolü mesaj işleme kontrol komut dizisinde yapılır.
4. Gelen mesajın haberleşme mesajı olması durumunda ise öncelikle mesaja ait kontrol mesajının daha önceden gelip gelmediği kontrol edilir. Bu kontrol işlemi aynı zamanda mesajın işlenebilirliğinin de kontrol edildiği mesaj işleme kontrol komut dizisidir. Mesajın işlenebilir olması durumunda mesaj işletilir ve bekletilen haberleşme mesajlarından işlenebilir duruma gelen mesaj olup olmadığı kontrol edilir. Simülör zamanını sadece mesaj işlediğinde artırır.

```

// $P_j$  simülatörü  $M(k, Data)$  veya  $M'(k, t)$  mesajlarını alıyor
//(1)Eğer gelen mesaj sıralama mesajı ise
if(message =  $M'$ )
  then
    //(2)mesaj numarasını ve işleme zamanını sakla
     $SV[k] = t$ 
    //(3) Mesaj bekletiliyor ise işle
    if(( $M'' = findMessage(k)$ )  $\neq$  NULL) then
      if(CanBeProcessed( $M''$ )) then
        receive( $M''$ )
    else //(4)mesaj heberleşme mesajı ise işlenebilirliğini test et
      if(CanBeDelivered( $M$ )) then
        //(4)mesajı işle ve bekletilen mesajlardan artık
        //işlenebilir olanlarını işle
        receive( $M$ )
        for all  $M'' \in Buff_{P_j}$ 
          if(CanBeDelivered( $M''$ ))
            receive( $M''$ )
      else
        //(5)mesajı beklet

```

Şekil 2.12: MKDM yönteminin mesaj alma komut dizisi.

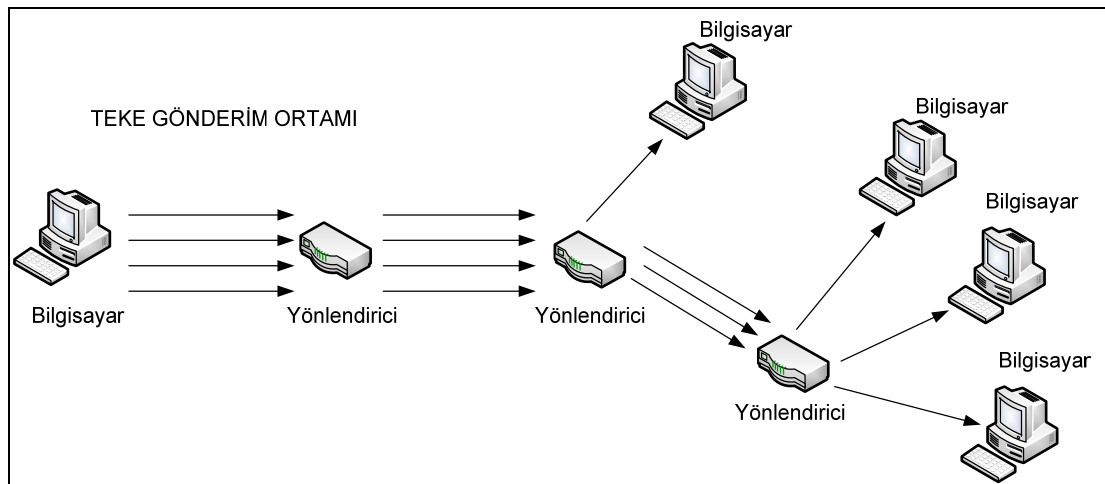
5. Haberleşme mesajı iki durumda işlenemez ve bekletilir. Mesaj hakkında kontrol simülatöründen işleneceği zaman bilgisinin gelmemiş olması yada simülatörün şu anki zamanından daha ileriki bir zamanı gösteren zaman bilgisinin gelmiş olması mesajın bekletilmesine neden olur. Bu durumda hafızada mesaj için yer açılır ve mesaj saklanır.

3. ALGORİTMALARIN TEORİK ANALİZİ

Bu bölümde MSES, KS ve MKDM algoritmalarının teorik analizi yapılacaktır. Algoritmalar altı kriter altında incelenecektir. Bu kriterler mesaj gönderme, mesaj alma işlemleri, mesaj formatları, nedenselliği sağlamak için kullanılan kontrol bilgileri, veri yapıları ve hafıza tüketimidir.

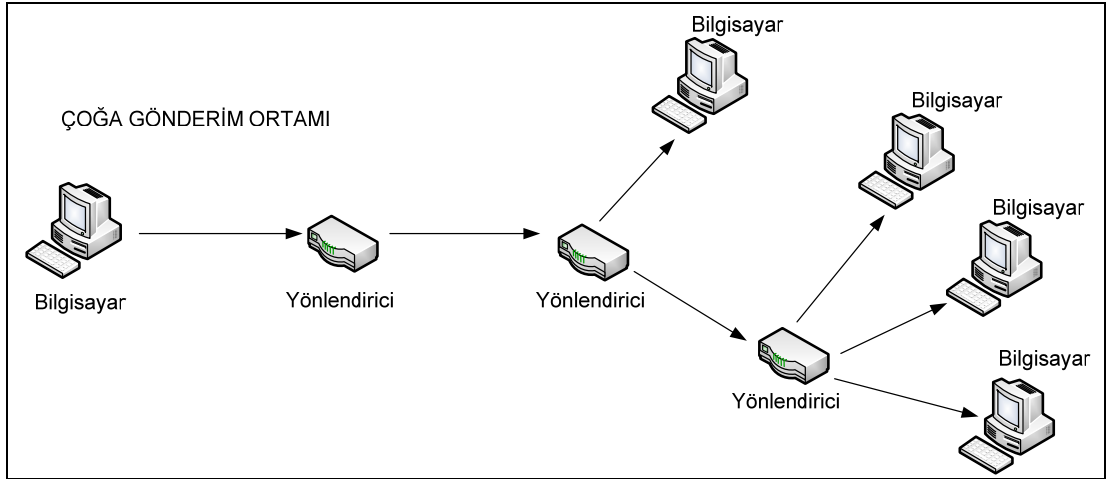
3.1. Mesaj Gönderme Mekanizması

Ağ üzerinde mesaj göndermek için üç ortamdan biri seçilebilir. Bu ortamlar teke gönderim, çoğa gönderim ve herkese gönderimdir. Bu tez çalışmasında algoritmalar teke ve çoğa gönderim ortamlarında incelenmiştir. Teke gönderim ortamı Şekil 3.1’de gösterilmektedir.



Şekil 3.1: Teke gönderim ortamında mesaj akışı.

Şekil 3.1’de gösterildiği gibi bir mesajı k alıcıya göndermek teke gönderim ortamlarında k mesajın ağ üzerinde akmasına neden olmaktadır. Çoğa gönderim ortamlarında ise herhangi iki nokta arasında (bilgisayar-bilgisayar yada yönlendirici-bilgisayar) hep tek mesaj olarak akmaktadır. Şekil 3.2’de çoğa gönderim ortamı verilmiştir.



Şekil 3.2: Çoğa gönderim ortamında mesaj akışı.

MSES algoritması hem teke hem de çoğa gönderim ortamlarında çalışabilmektedir. Algoritma, mesaj gönderme mekanizmasında mesajı ileteceği her alıcı simülâtör için aynı nedensellik ve zaman vektörünü kullanmaktadır. Bu özelliği ona çoğa gönderim ortamlarında tek bir mesajı bütün alıcılara gönderebilme imkanı tanımaktadır. Ancak her alıcıya aynı nedensellik vektörünü göndermek, çok genel bir veri hazırlamayı ve alıcı simülâtörlerin hepsinin ihtiyaç duymayacağı bilgileri de fazladan almasına neden olmaktadır. MSES algoritması bu sebeple gereksiz nedensellik bilgilerini nedensellik vektöründen eleme işini mesaj alma mekanizmasına bırakmıştır. Bu yaklaşım mesaj içinde gönderilen nedensellik bilgisinin gereğinden büyük olmasına neden olmaktadır. KS algoritması ise teke gönderim ortamlarında çalışacak şekilde tasarlanmıştır. Algoritma çoğa gönderim ortamlarında k alıcılı bir mesajı k tane tek alıcılı mesaj haline getirip göndermektedir. Mesaj gönderme mekanizmasında her alıcı için ayrı nedensellik vektörü oluşturmakta ve her alıcı simülâtöre ayrı mesaj göndermektedir. Bu yönü ile çoğa gönderim ortamlarının tekil mesaj gönderme avantajlarını kullanmayan algoritma, her alıcıya özel ürettiği nedensellik bilgisi ile MSES algoritmasına göre mesaj başına daha az nedensellik bilgisi gönderebilmektedir.

MKDM algoritması ise hem teke, hem de çoğa gönderim ortamlarında çalışabilmektedir. Algoritma haberleşme mesajının içine nedensellik bilgisi koymadığı için, tüm alıcılar için tek mesaj oluşturabilmektedir. Simülâtörlerden yönetici simülâtöre giden sıralama mesajı ise MSES ve KS algoritmalarının mesaj

içine koyduğu nedensellik ve zaman vektöründen çok daha küçük kontrol bilgisi taşımaktadır. Yönetici simülatörden alıcı simülatörlere gönderilen sıralama mesajları ise her alıcı simülatör için ayrı hazırlanır.

MSES algoritmasının nedensellik bilgisinin bir parçası olarak mesajlar içine koyduğu VT zaman vektörü algoritmaya hem avantaj hem de dezavantaj getirmektedir. Bir yandan mesaj boyutunun artmasına neden olan VT, öte yandan algoritmaya sağladığı fazladan bilgi ile nedensellik vektörü üzerinde daha iyi eleme yapmasına yardımcı olmaktadır. Gevşek bağlı simülasyonlar, simülatörlerin birbirleri ile daha az haberleştiği ve mesajlaşma oranının düşük olduğu sistemlerdir. Bu nedenle simülatörler simülasyon ortamı hakkında daha az bilgiye sahip olurlar. Böyle bir ortamda zaman vektörü MSES algoritmasının daha verimli çalışmasını sağlayabilir. KS algoritması ise SR zaman vektörünü mesaj içinde göndermemektedir. Bu nedenle gevşek bağlı simülasyonlarda KS algoritması kullanan simülatörler birbirleri ve dış dünya hakkında daha az bilgiye sahip olurlar. Birbirlerinin son durumlarını yeterince iyi bilmeyen simülatörler aslında gerekli olmayan nedensellik bilgilerini de mesaj içine koyarak gönderebilirler. Sıkı bağlı ortamlarda ise simülatörler birbirleri ile oldukça sık mesajlaşmaktadır. Dolayısıyla simülatörler birbirleri ve dış dünya hakkında daha çok bilgiye sahip olabilmektedirler. Sıkı bağlı ortamlarda VT vektörünün önemi azalmakta ve kullanımı mesaj boyutunu büyütmesi nedeniyle dezavantaja neden olabilmektedir.

MKDM algoritması ise haberleşme mesajları üzerine nedensellik bilgisi koymadığı, sunucu simülatöre gönderdiği kontrol bilgisinde de sadece haberleşme mesajının ve mesaj alıcılarının numaralarını taşıdığı için, ağ üzerinde çok az gecikmeye neden olmaktadır. Algoritmanın darboğaz yaratabileceği durumlar sıkı bağlı simülasyonlarda yönetici simülatörün sıralama mesajlarını yeterince hızlı cevaplayamaması durumlarında oluşabilir. Yönetici simülatöre akan mesajların artması simülatörün bağlı olduğu hat üzerinde de gecikmelerin oluşmasına sebebiyet verebilir. Öte yandan sadece zaman vektörünü kullanarak sıralama yapabiliyor olması yönetici simülatörün sistem isteklerine hızlı bir şekilde cevap vermesine yardımcı olmaktadır. Algoritmaların performanslarına yönelik deneysel analizler ve sonuçlar bölüm 4'te verilmiştir.

3.2. Mesaj Alma Mekanizması

MSES ve KS algoritmalarının mesaj alma mekanizmaları benzerlikler sergilemektedir. Her iki algoritmada mesaj içinde gelen nedensellik vektörü ile simülatörler içindeki nedensellik vektörlerini karşılaştırarak iki vektörün güncel bilgileri eşliğinde gereksiz ve eski bilgilerini silmektedir. Aralarındaki en önemli fark zaman vektörlerindedir. MSES'in zaman vektörü VT mesajlar içinde de gönderilir ve tüm simülasyondaki federelerin en son zamanlarını tutar. KS algoritmasındaki SR vektörü ise diğer federelerden kendisine gelen en son mesajların zamanlarını tutar. İki vektör de gelen mesajların işlenebilirliğinde kullanılır. MKDM algoritmasının mesaj alma mekanizmasının işlem yükü çok azdır. Diğer algoritmalarındaki nedensellik vektörlerinin karşılaştırılması, üzerlerinde eleme ve birleşme işlemlerinin yapılması işleri bu algoritmada yoktur. Bu nedenle simülatörler üzerinde en az seviyede işlem yükü oluşturur. Bir mesajın işlenmesi için yönetici simülatör tarafından o mesaja ait sıralama mesajının simülatöre ulaşmış olması gerekmektedir. Dolayısıyla mesajlaşmanın çok yoğun olduğu durumlarda, kontrol simülatörünün cevap vermesindeki gecikme, simülatörlerin mesaj işleme hızlarını doğrudan etkileyebilmektedir.

3.3. Veri Yapıları

MSES algoritmasının kullandığı CV nedensellik vektöründen bölüm 2.1.1.2'de bahsedilmiştir. CV vektörünün formatı formül 3.1'de tanımlanmıştır.

$$CV_{S_2} = \left\{ \underbrace{\{(k, x), (), \dots, ()\}}_{S_1}, \underbrace{\{\}}_{S_2}, \underbrace{\{\}}_{S_3}, \dots, \underbrace{\{\}}_{S_N} \right\} \quad (3.1)$$

N simülatörden oluşan bir simülasyon ortamında nedensellik vektörü her simülatör için en kötü şartlarda N 'e kadar büyüeyen vektörlerden oluşan vektördür. Dolayısıyla en kötü şartlarda, nedensellik vektörü $O(N^2)$ lik kontrol bilgisi barındırır. MSES algoritmasında ata mesajları temsil için (k, x) formatında ikili veriler kullanılır. Burada k , mesajı gönderen simülatörün tanımlayıcı numarası, x ise

k simülatörüne göre mesaj gönderim zamanıdır. Bu ikili yapı, CV içinde mesajı alacak olan alıcı simülatörleri temsil eden ilgili alanlara yazılır. Mesaj içinde gönderirken bu ikililer birer tam sayı ile ifade edilebilir. N simülatörden oluşan bir simülasyonda, toplam k simülatörün mesajı alacağı bir senaryo için MSES algoritması k tane ikili oluşturacak ve alıcı simülatörleri temsil eden alanlara kaydedecektir. Dolayısıyla MSES algoritmasında bir mesajın nedensellik vektöründe temsil edilmesi $2k$ tam sayı ile olacaktır.

KS algoritmasının kullandığı LOG nedensellik vektöründen bölüm 2.2.1.2’de bahsedilmiştir. LOG vektörünün formatı formül 3.2’de tanımlanmıştır.

$$LOG_j = \{o_{i,a}, o_{k,b}, \dots, o_{s,t}\}; o_{i,a} = (i, a, Dests) \quad (3.2)$$

LOG vektörü içindeki $o_{i,a}$ yapısı bir ata mesajın imzasını temsil etmektedir. MSES algoritmasındaki farklı olarak mesaj temsil eden (k,x) ikilileri oluşturup ilgili alıcı simülatörün dizinine kaydedilmez. Onun yerine tek bir imza oluşturulur. $(i,a, Dests)$ formatındaki mesaj imzasında, i kaynak simülatörün tanımlayıcı numarasını, a , i simülatörüne göre mesajın gönderim zamanını, $Dests$ ise alıcı simülatörlerin tanımlayıcı numaralarını barındırır. Yukarıdaki bahsedilen N simülatörden oluşan bir simülasyonda, toplam k simülatörün mesajı alacağı bir senaryo için, i ve a değişkenleri için birer tam sayı ve $Dests$ alıcı listesi için k tam sayı ile mesaj imzası toplamda $2+k$ tam sayı ile ifade edilebilir. Bu oran MSES için $2k$ ’dır. Dolayısıyla aynı bilgiyi ifade etmek için KS algoritması daha verimli bir mesaj imza formatı kullanmaktadır. Ağ üzerinde iki algoritmanın da mesajının aktığı bir ortamda mesajlar aynı nedensellik bilgisini barındırsa bile LOG vektörü CV vektöründen küçük olacaktır. Aynı bilgilerin simülatörler üzerinde de tutulduğu düşünüldüğünde KS algoritmasının aynı nedensellik bilgisini tutmak için daha az hafıza alanı kullanacağı söylenebilir.

MKDM algoritmasının ise yönetici simülatör üzerinde tuttuğu zaman vektörü vardır. Bu zaman vektörü simülasyonda bulunan simülatörlerin zamanlarını tutmaktadır. Dolayısı ile N simülatörden oluşan bir simülasyonda bu vektör hafızada

N tam sayılık yer kaplayacaktır. Simülatörler üzerinde de zaman değerini tutan 1 tam sayılık sayaç değişkeni bulunmaktadır.

3.4. Mesaj Formatı ve Kontrol Bilgisi

MSES algoritmasının mesaj formatı aşağıda tanımlandığı gibidir.

$$(M, i, Dests(M), CV_M, VT_M) \quad (3.3)$$

M mesajın kendisini, i kaynak simülatörün numarasını, $Dests(M)$ mesajın gönderileceği alıcı simülatörleri, CV_M mesajın nedensellik vektörünü ve VT_M mesajın zaman vektörünü göstermektedir. Boyut olarak bakıldığında i kaynak simülatör numarası 4 baytlık bir tam sayı ile ifade edilebilir. N simülatörden oluşan bir simülasyonda, toplam k simülatörün mesajı alacağı bir senaryo düşünülürse $Dest(M)$, k tam sayı ile ifade edilir. CV_M vektörünün büyüklüğü simülasyonun yapısına, simülatör sayısına ve simülasyonda geçen zamana bağlı olarak değişmektedir. VT_M zaman vektörü ise yukarıdaki senaryoya göre N tam sayıdan oluşan bir dizidir. Sonuç olarak MSES mesajları CV_M vektörü ile beraber $1+k+N$ tam sayılık (4 baytlık) nedensellik bilgisi taşımaktadır.

$$\{i, clock_i, M, Dests, O_M\} \quad (3.4)$$

KS algoritmasının mesaj formatı 3.4'te tanımlanmıştır. KS algoritmasının mesaj formatı, MSES algoritmasının mesaj formatına benzer yapıdadır. i kaynak simülatörün numarasını, $clock_i$ mesajın zamanını, M mesajın kendisini, $Dests$ alıcı simülatörlerin numaralarını ve O_M ise nedensellik vektörünü göstermektedir. Yukarıda bahsedilen senaryoya göre KS mesajı, i kaynak simülatör numarası için 1 tam sayı, $clock_i$ mesaj gönderim zamanı için 1 tam sayı, $Dests$, alıcı simülatörler için k tam sayı ve O_M nedensellik vektörünü taşıyacaktır. KS mesajları sonuçta O_M vektörü ile beraber $2+k$ tam sayılık (4 baytlık) nedensellik bilgisi taşımaktadır.

MSES algoritması KS algoritmasından farklı olarak mesaj içinde VT zaman vektörünü göndermektedir. Kontrol bilgisi ve veri yapıları bölümünde daha detaylı incelenecek olan VT vektörü, MSES'in simülatörler içindeki eski ve gereksiz nedensellik bilgilerini daha etkin silmesine yardımcı olmakla birlikte mesajların boyutlarının büyümesine de neden olabilmektedir.

CV_M ve O_M vektörlerinin boyutlarının birbirine yakın olduğu ortamlarda MSES mesajlarının boyutları KS mesajlarından büyük olmaktadır. Özellikle her alıcı simülatör için ayrı mesaj gönderilmesi gereken teke gönderim ortamlarında MSES mesajları ağ üzerinde daha fazla kaynak tüketimine neden olabilir. Çoğa gönderim ortamlarında ise tüm alıcılara tek bir mesaj gönderebilen MSES algoritması, alıcı sayısı arttıkça, KS algoritmasından daha az ağ kaynağı tüketecektir.

Simülatörler üzerindeki işlem yüküne bakıldığında MSES algoritmasının mesaj gönderme mekanizmasında sadece nedensellik vektörlerini mesaj içine kopyalama işlemleri vardır. KS algoritmasının mesaj gönderme mekanizmasında ise her simülatör için ayrı nedensellik vektörleri oluşturma ve mesajlara kopyalama işleri bulunmaktadır. Bu nedenle benzer veri büyüklükleri için MSES'in mesaj gönderme mekanizması KS'nin mesaj gönderme mekanizmasından daha hızlı çalışacaktır. Mesaj alma mekanizmalarında ise MSES ve KS algoritmaları simülatörlerin ve mesajların içindeki nedensellik vektörleri üzerinde karşılaştırma, eleme ve birleşme işlemleri yapmaktadır. Şekil 2.4 ve Şekil 2.6'da anlatılan mesaj alma komut dizinlerinden de anlaşılacağı üzere her iki algoritma da N^2 büyüklüğündeki nedensellik vektörleri üzerinde çalıştığı için $O(N^2)$ 'de çalışmaktadır. MSES algoritması fazladan VT zaman vektörünü de kullanarak daha fazla iş yapmakta ve veri üzerinde çalışmaktadır. Ancak VT zaman vektöründe bir veriyi okuyup yazmak $O(N)$ zaman aldığı için algoritma üzerine asimptotik olarak fazladan işlem yükü getirmemektedir.

MKDM algoritmasında sıralama bilgileri iki farklı sıralama mesajında akmaktadır. Simülatörlerden yönetici simülatöre gönderilen sıralama mesajının formatı formül 3.5'te verilmiştir.

$$\{i_m, Dests\} \quad (3.5)$$

i_m haberleşme mesajının tanımlayıcı numarası ve $Dests$ ise yine haberleşme mesajını alacak simülatörleri göstermektedir. Mesajın kendisi simülatörler arasında doğrudan gönderildiği için bu mesaj içinde yer almaz. Boyut olarak MSES ve KS mesajından çok daha küçük olan sıralama mesajı, diğer algoritmaların aksine tüm alıcı simülatörlere gönderilmek yerine sadece yönetici simülatöre gönderilmektedir. N simülatörden oluşan bir simülasyonda, toplam k simülatörün mesajı alacağı bir senaryo düşünülürse i_m 4 baytlık 1 tam sayı ile, $Dests$ ise k tam sayı ile ifade edilebilir. Dolayısıyla her haberleşme mesajı için yönetici simülatöre $1+k$ tam sayı boyutunda sıralama mesajı gelmektedir. MSES ve KS mesajlarının boyutları düşünüldüğünde çok küçük sayılabilecek sıralama mesajları simülatörler arasında çok yoğun mesaj trafiği olması durumunda yönetici simülatör üzerinde darboğaz oluşmasına neden olabilir. Özellikle yönetici simülatöre gelen ve ondan çıkan ağ bağlantıları üzerinde gecikmeler oluşabilir.

Yönetici simülatörden haberleşme mesajının alınacağı simülatörlere gönderilen diğer sıralama mesajının formatı ise formül 3.6'da verilmiştir.

$$\{i_m, t\} \quad (3.6)$$

i_m haberleşme mesajının tanımlayıcı numarasını, t ise bu mesajın ilgili alıcı simülatör tarafından hangi zamanda işlenmesi gerektiğini göstermektedir. Her simülatörün mesajı işleyeceği zaman farklı olabileceği için alıcı simülatörler için her birine özel sıralama mesajları gönderilmektedir. Bu sıralama mesajlarının boyutu simülatör sayısına ve yayın ortamına bağlı değildir. Mesajın büyüklüğü düşünüldüğünde i_m ve t birer tam sayı ile ifade edilebilir. Yukarıda bahsedilen N simülatörden oluşan bir simülasyonda, toplam k simülatörün mesajı alacağı bir senaryo için k sıralama mesajı oluşturulacak ve bir haberleşme mesajı için toplamda $2k$ tam sayı kullanılacaktır. Yönetici simülatöre gelen büyüklüğü $1+k$ tam sayılık

sıralama mesajı da düşünülürse bir haberleşme mesajı için harcanan toplam mesaj sıralama bilgisi boyutu $(1+3k)$ tam sayıdır.

Bölüm 4'te bu bölümde teorik olarak incelenen MSES, KS ve MKDM algoritmaları gerçekleştirilmiş ve çeşitli durumlarda deneysel olarak analiz edilmiştir.

4. ALGORİTMALARIN DENEYSEL ANALİZİ

Bu bölümde MSES, KS ve MKDM algoritmalarının deneysel analizleri yapılmıştır. Algoritmaları karşılaştırmak için iki yol izlenebilir. İlki gerçek dağıtık bir sistem oluşturmak ve algoritmaları bu sistem üzerinde test etmektir. İkinci yöntem ise dağıtık bir ortamı simüle edebilen bir yazılım geliştirmek ve bu sanal ortam üzerinde algoritmaları gerçekleyip analiz etmektir. Bu çalışmada ikinci yöntem benimsenmiştir. Simülatör sayısının belirlenebildiği, farklı mesaj akışları sonucu ağ üzerinde akan veri büyüklüklerini, simülatörlerin işlem yüklerini, ağda oluşabilecek gecikmeleri modelleyebilen dağıtık ortam simülatörü (DOS) yazılımı gerçekleştirilmiş ve algoritmalar bu yazılım üzerinde koşturularak sonuçlar toplanmıştır.

Simülatörlerin simülasyon boyunca sergiledikleri davranışlar, ağ üzerinde akan mesajlar ile doğrudan ilişkilidir. Mesaj sayısındaki değişim, mesaj alıcılarının sayısının artıp azalması simülatörlerin performanslarını ölçmek açısından önemlidir. Farklı senaryolar oluşturabilmek açısından mesaj üretici yazılımı gerçekleştirilmiştir. Bu yazılım ile tüm simülasyon boyunca ağ üzerinde akacak mesajlar modellenilebilir. Üretilen mesajlar DOS yazılımında girdi olarak alınır ve gönderim zamanlarına göre birer olaya dönüştürülerek kullanılır.

4.1. Mesaj Üretici

Bir simülasyonda simülatörleri değiştirmeden sadece mesaj sayısı, mesaj alıcı sayısı ve mesaj gönderim zamanlarını değiştirerek farklı senaryolar geliştirmek mümkündür. Mesaj Üretici yazılımı DOS yazılımı ile tümleşik çalışan ve farklı senaryolar için ihtiyaç duyulan mesajları üreten bir yazılımdır. Visual Studio 2005 geliştirme ortamında, C++ programlama dili kullanılarak geliştirilmiştir. Kullanıcıya mesaj sayısını, mesaj alıcı sayısını, mesaj gönderim ve alım zamanlarını değiştirerek simülasyonda akacak mesajları üretme imkanı tanır. Mesajı gönderecek simülatörün, mesaj alıcılarının, gönderim ve alım zamanlarının seçimi tamamen düzenli dağılıma göre üretilmektedir. Mesaj üretmek için gerekli parametreleri XML tabanlı bir dosyadan okumaktadır. Şekil 4.1’de mesaj üretici yazılımının kullanabileceği örnek XML dosyası görülmektedir.

```

<Process_Number value="50"/>
<Message_Number value="7000"/>
<Pattem>
  <Send_Interval>
    <min value="2"/>
    <max value="5"/>
  </Send_Interval>
  <Receive_Interval>
    <min value="2"/>
    <max value="5"/>
  </Receive_Interval>
  <Receiver_Numbers>
    <min value="20"/>
    <max value="25"/>
  </Receiver_Numbers>
</Pattem>

```

Şekil 4.1: Mesaj Üreticine ait yapılandırma dosyası.

“Process_Number” etiketi simülasyonda bulunacak simülatör sayısını, “Message_Number” etiketi üretilecek toplam mesaj sayısını tanımlar. “Pattern” etiketi ise üretilecek mesajların gönderilme, alınma ve alıcı sayılarının tanımlandığı yerdir. “Send_Interval” etiketi mesajların gönderim zaman aralıklarının girildiği yerdir. “min” ve “max” etiketlerinde belirtilen aralıklarda mesaj gönderim zamanları üretilir. “Receive_Interval” etiketi ise mesajların gönderimden itibaren ne kadar gecikme ile alınabileceklerinin belirtildiği yerdir. Burada girilen gönderim ve alım zamanları mantıksal zamanlardır ve DOS yazılımının mantıksal zamanına göre tanımlanmıştır. Mesaj alım zamanları ayrıca DOS yazılımı tarafından ağ gecikmeleri ve simülatörler üzerinde harcanan zamanlar göz önüne alınarak koşum sırasında üretilebilmektedir. “Receiver_Numbers” etiketinde ise mesajları kaç alıcının alacağı bilgisi girilir. Üretilen mesajlar metin tabanlı bir dosyaya kaydedilir.

4.2. Dağıtık Ortam Simülatörü (DOS)

DOS, dağıtık bir sistemin benzetimini yapan bir yazılımdır. Mesaj üretici tarafından üretilen her mesaj için bir gönderim ve mesaj alıcı sayısı kadar da alım olayları oluşturur. DOS yazılımı sanal zaman kullanmaktadır. Olaylar listesinden zamanı en küçük olayı seçer. Eğer olay bir gönderim olayı ise kaynak simülatörü uyarır ve simülatörün mesaj gönderme komut dizisini çalıştırır. Simülatörler MSES, KS veya MKDM algoritmaları ile modellenmiş olabilirler. Mesaj alım olaylarında ise

alıcı simülatörleri uyarır ve mesaj alım komut dizilerini çalıştırmalarını sağlar. Bu arada belirtilen ağ topolojisine uygun olarak mesajın ağ üzerinde akmasını ve alıcı simülatörlere gecikmeli gitmesini modelleyebilir. DOS yazılımı tamamen şeffaf bir yazılımdır ve simülatörler açısından gerçek bir sistemde çalışmalarından farkı yoktur. DOS yazılımı simülasyonda kullanacağı ağ topolojisini XML tabanlı bir dosyadan okumaktadır.

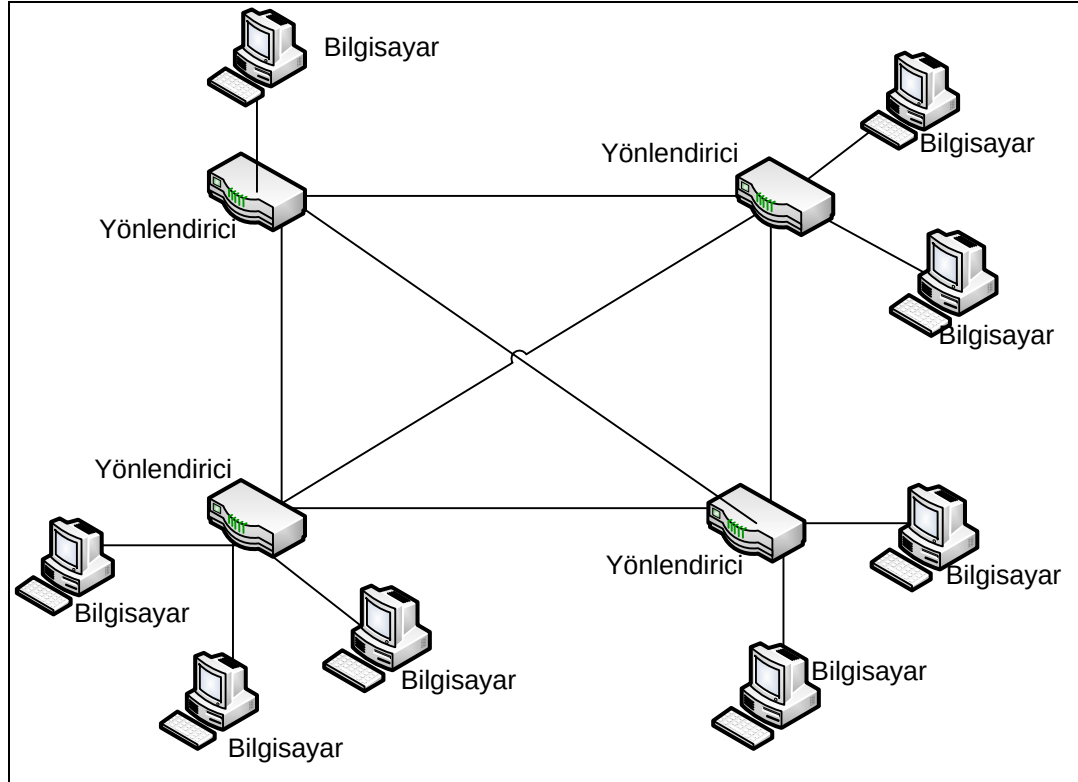
Şekil 4.2’de 6 simülatör ve 4 yönlendiriciden oluşan ağ modeli tanımlama dosyası gösterilmektedir.

```
<Network_Model>
  <type value="1" />
  <!-- 0=>unicast, 1=>multicast -->
  - <Links>
    <Link id="0" connected1="0" connected2="6" capacity="10000" />
    <Link id="1" connected1="1" connected2="7" capacity="10000" />
    <Link id="2" connected1="2" connected2="7" capacity="10000" />
    <Link id="3" connected1="3" connected2="8" capacity="10000" />
    <Link id="4" connected1="4" connected2="9" capacity="10000" />
    <Link id="5" connected1="5" connected2="9" capacity="10000" />
    <Link id="6" connected1="6" connected2="7" capacity="10000" />
    <Link id="7" connected1="6" connected2="8" capacity="10000" />
    <Link id="8" connected1="8" connected2="9" capacity="10000" />
    <Link id="9" connected1="7" connected2="9" capacity="10000" />
    <Link id="10" connected1="7" connected2="8" capacity="10000" />
    <Link id="11" connected1="6" connected2="9" capacity="10000" />
  </Links>
  - <Routers>
    - <Router id="6">
      <connected_device id="0" />
    </Router>
    - <Router id="7">
      <connected_device id="1" />
      <connected_device id="2" />
    </Router>
    - <Router id="8">
      <connected_device id="3" />
    </Router>
    - <Router id="9">
      <connected_device id="4" />
      <connected_device id="5" />
    </Router>
  </Routers>
</Network_Model>
```

Şekil 4.2: DOS yazılımı ağ modeli tanımlama dosyası.

Kullanıcı “Links” etiketi içinde ağ modelini oluşturacak bağlantıları tanımlar. Bağlantılar bilgisayar ile bilgisayar, yönlendirici ile bilgisayar ya da yönlendirici ile yönlendirici arasında olabilir. Bağlantılara bayt cinsinden kapasite değeri de verilebilir.

“Router” etiketi içinde ise yönlendirici tanımlanır. Her yönlendiriciye bağlı olması düşünülen bilgisayarlar tek tek belirtilir. DOS yazılımı Şekil 4.3’te belirtilen şekilde tanımlanan bütün ağ topolojilerini modelleyebilir.



Şekil 4.3: DOS yazılımı için tasarlanan ağ modeli.

DOS yazılımı Visual Studio 2005 geliştirme ortamında, C++ programlama dilinde gerçekleştirilmiştir.

4.3. Algoritmalarının Gerçeklenmesi

Algoritmalar DOS yazılımında bulunan mesaj ve işlemci sınıflarından türetilerek gerçekleştirilmiştir. Böylece tüm algoritmalar çok küçük değişiklikler yaparak DOS üzerinde çalıştırılabilir hale getirilmiştir. Tüm algoritmalar farklı mesaj yapıları kullandıkları için DOS içindeki temel mesaj sınıfından kendi mesaj yapılarına uygun sınıflar türetilerek mesaj modelleri oluşturulmuştur. KS

algoritmasına ait mesaj işleme kontrol komut dizinindeki tıkayarak bekleme yaklaşımı, MSES algoritmasındaki işlenemeyecek mesajları hafızaya atarak ileride kontrol etme yaklaşımı ile değiştirilmiştir.

Tüm algoritmalar Visual Studio 2005 geliştirme ortamında, C++ programlama dili ve STL standart kütüphanesi kullanılarak gerçekleştirilmiştir.

4.4. Deneysel Sonuçlar

Bu tez kapsamında 4 yönlüticiden oluşan ve bütün yönlüticilerin birbirine bağı olduğu, simülatör sayısının 5 simülatörden 50 simülatöre kadar değıştiğı ağı modelleri kullanılmıştır. Yönlüticiler üzerindeki gecikme miktarı ihmal edilmiş, simülatörlerde oluşan işlem yükü, ağıda akan nedensellik bilgisi ve ağıda oluşan gecikme miktarları hesaplanmıştır. Algoritmalar hem teke hem de çoğa gönderim ortamlarında karşılaştırılmıştır. Karşılaştırmalar 4 farklı senaryoda gerçekleştirilmiştir. Bu senaryolar, gevşek bağı, sıkı bağı, düzgün dağılımlı simülasyon ortamları ile aynı mesaj sayısı için mesaj alıcılarının arttırıldığı senaryolardır.

Simülatör-simülatör ya da simülatör-yönlendirici arasındaki bağılantılarda oluşan ağı gecikmesi hesaplanırken kullanılan formül 4.1’de tanımlanmıştır. Kapasite ve mesaj boyutu bayt cinsinden verilmektedir.

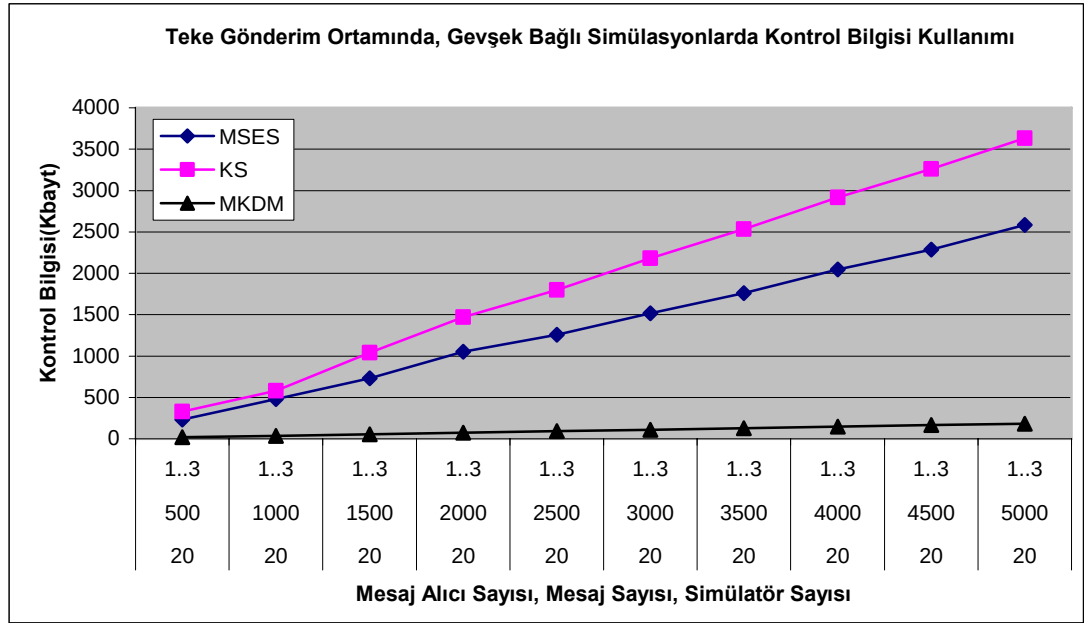
$$\text{Ağı gecikmesi} = (1/\text{Kapasite}) * \text{Mesaj Boyutu} \quad (4.1)$$

Elde edilen ağı gecikmesi birimsizdir. DOS yazılımı tarafından simülasyon koşumu sırasında tüm bağılantılardaki ağı gecikmeleri hesaplanmaktadır.

Tüm deneysel çalışmalar Intel Dual Core 1.8 GHz işlemci ve 2.0 GB RAM’e sahip bir bilgisayarda yapılmıştır. Her test üçer kez yapılmış ve ortalamaları alınmıştır.

4.4.1. Gevşek Bağlı Simülasyonlarda Algoritmaların Kontrol Bilgisi, İşlem Yüğü ve Ağ Gecikmesi Analizleri

Gevşek bağlı simülasyonlarda simülatörler arasındaki bağımlılık oldukça düşüktür. Bu nedenle simülatörler arası mesajlaşma da azdır. Böyle bir ortamı modellemek için alıcı sayısının 1 ile 3 arasında eşit dağıldığı, simülatör sayısının 20 simülatör ile sabit tutulduğu ancak mesaj sayısının değiştirildiği modeller hazırlanmıştır. Bu senaryoya göre algoritmaların nedensellik bilgisi kullanımı Şekil 4.4'te verilmiştir.

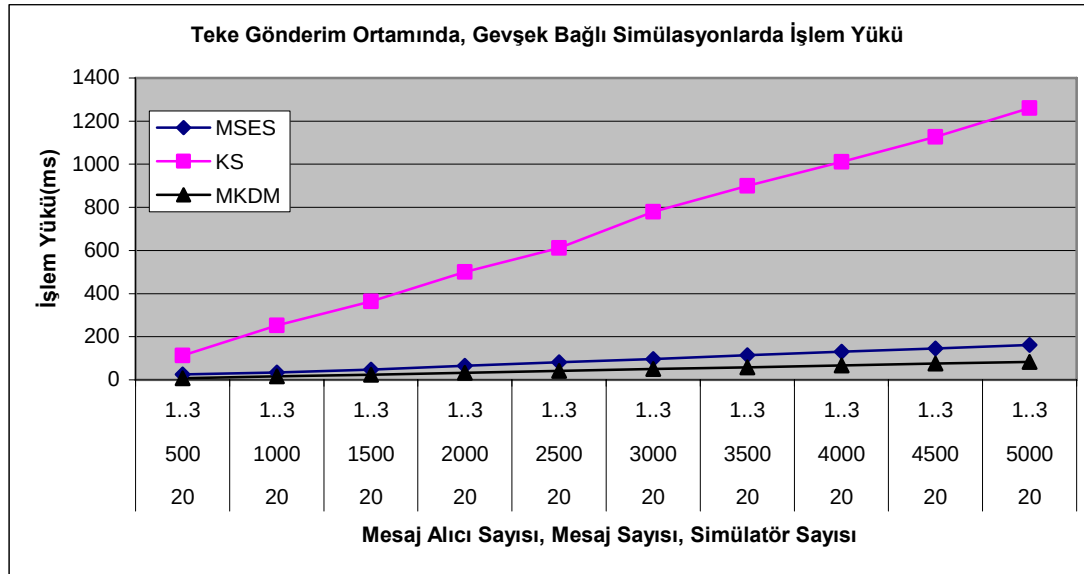


Şekil 4.4: Teke gönderimde gevşek bağlı simülasyonda kontrol bilgisi kullanımı.

Mesaj sayısındaki artış en çok KS algoritmasının kontrol bilgisinde artışa neden olmaktadır. Mesaj alıcı sayısının az olduğu durumlarda KS algoritması, VT vektörünü de kullanarak daha iyi bilgi elemesi yapabilen MSES algoritmasından daha kötü sonuçlar vermektedir. MSES algoritması iletişimin zayıf olduğu gevşek bağlı simülasyonlarda VT vektörünü kullanarak çok daha iyi nedensellik bilgisi eledebilmekte ve daha az kontrol bilgisini mesajlar içinde gönderebilmektedir. Özellikle az alıcının olduğu durumlarda KS algoritması dış dünya hakkında yeterince bilgi sahibi olamamaktadır. Bu nedenle mesaj eleme kabiliyeti zayıflamaktadır. MKDM algoritması ise diğer algoritmalarından çok daha az kontrol bilgisi

kullanılmaktadır. Bunun en önemli nedeni haberleşme mesajları içine kontrol bilgisi koymayı ve her mesaj için oldukça küçük kontrol bilgisinin sadece yönetici simülatöre gönderilmesidir. Şekil 4.5'te yukarıda kullanılan kontrol bilgisinin algoritmalara ne kadar iş yükü getirdiği gösterilmektedir.

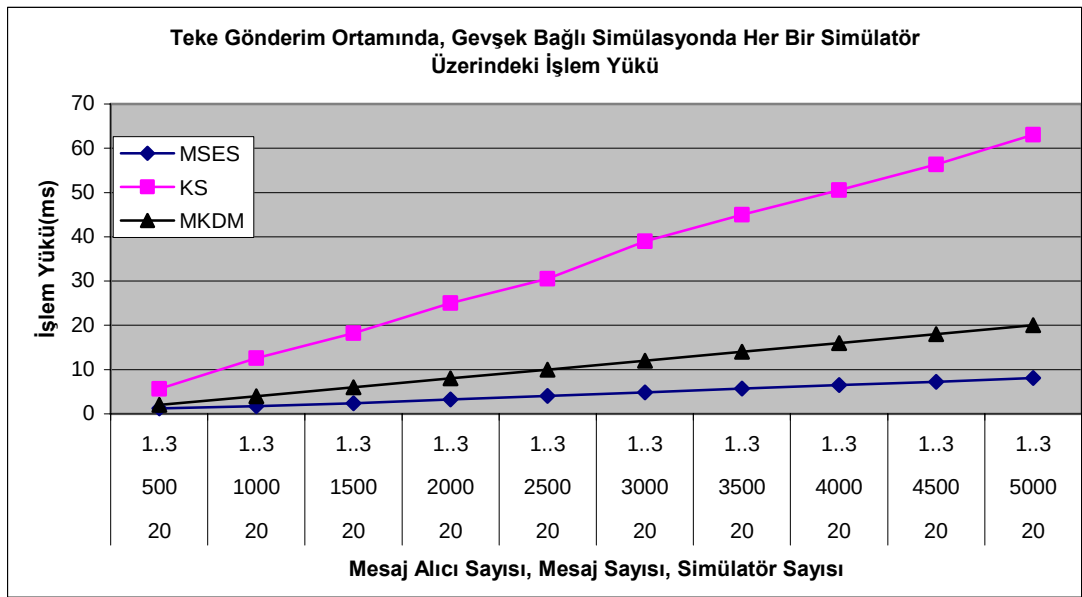
KS algoritması mesaj gönderim mekanizmasında her bir alıcı için mevcut nedensellik vektörünü düzenlemekte, vektör üzerinde gerekli eleme işlemleri yapmakta ve mesaj gönderimini gerçekleştirmektedir. Sonuç olarak k alıcı bir mesaj için $k \cdot \text{LOG}$ büyüklüğünde veri üzerinde çalışmaktadır. Bu yaklaşım algoritmanın simülatörler üzerinde diğer algoritmalarından çok daha fazla işlem yükü oluşturmaya neden olmaktadır. MSES algoritması ise mesaj gönderim mekanizmasında sadece kendi nedensellik vektörünü mesaj içine koymakta ve mesajı göndermektedir. Bu noktada simülatörler üzerinde oluşturduğu işlem yükü çok daha azdır. MKDM algoritması ise diğer algoritmalarından daha küçük kontrol bilgisi kullandığı için simülatörler üzerinde de toplamda en az işlem yüküne neden olmaktadır. Sonuç olarak MSES ve MKDM algoritmaları gevşek bağlı simülasyonlarda mesaj artışına karşı daha dayanıklı ve ölçeklenebilir algoritmalarlardır.



Şekil 4.5: Teke gönderimde gevşek bağlı simülasyonda işlem yükü.

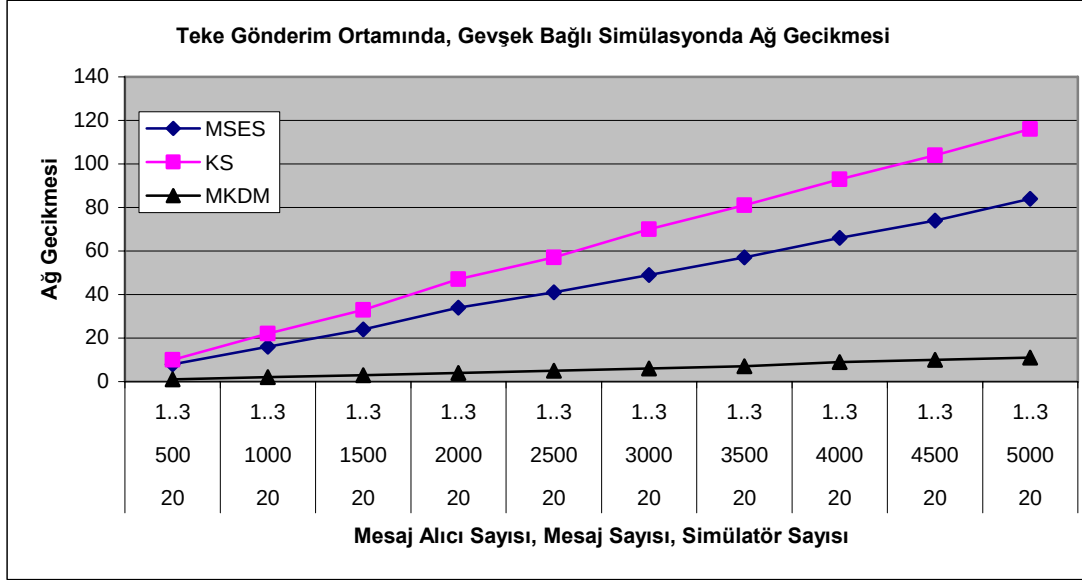
Her bir simülatörün üzerinde oluşan işlem yüküne bakıldığında ise yukarıdaki sonuçlardan farklı sonuçlar elde edilmektedir.

Şekil 4.6’da her bir simülatör üzerinde oluşan işlem yükü gösterilmiştir. MKDM algoritmasında ise yönetici simülatörün işlem yükü kullanılmıştır. MKDM algoritmasının toplam işlem yükü daha önce MSES algoritmasından küçük çıkmasına rağmen yönetici simülatörün neden olduğu işlem yükü MSES algoritması kullanan herhangi bir simülatörden daha fazladır. Yönetici simülatör üzerinde oluşan işlem yükü tüm işlem yükünün %17 ile %20 arasında değişmektedir. Bu oran diğer algoritmalar için çok daha düşüktür. Sonuç olarak MKDM algoritmasında yönetici simülatör üzerinde oluşan yük, diğer simülatör üzerinde oluşan işlem yükünden daha fazladır ve bazı durumlarda sistemin yavaşlamasına neden olabilir.



Şekil 4.6: Teke gönderimde gevşek bağlı simülasyonda her bir simülatör üzerindeki işlem yükü.

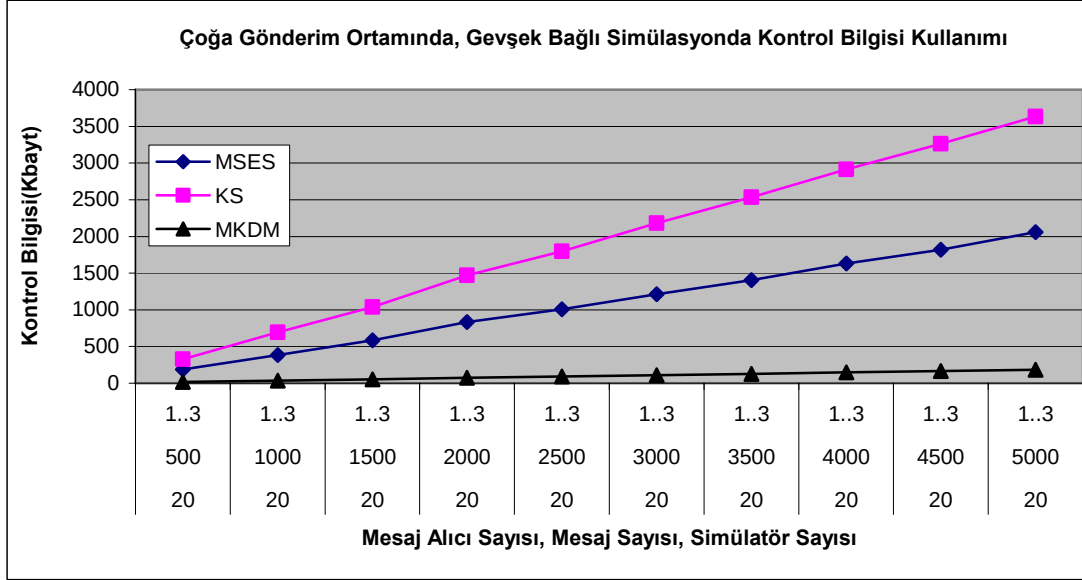
Şekil 4.7’ye dayanarak aynı ortamda algoritmaların ağ üzerinde oluşturduğu gecikme ise ağ üzerinde akan kontrol bilgisi ile uyumlu bir görüntü sergilemektedir.



Şekil 4.7: Teke gönderim ortamında gevşek bağlı simülasyonda ağ gecikmesi.

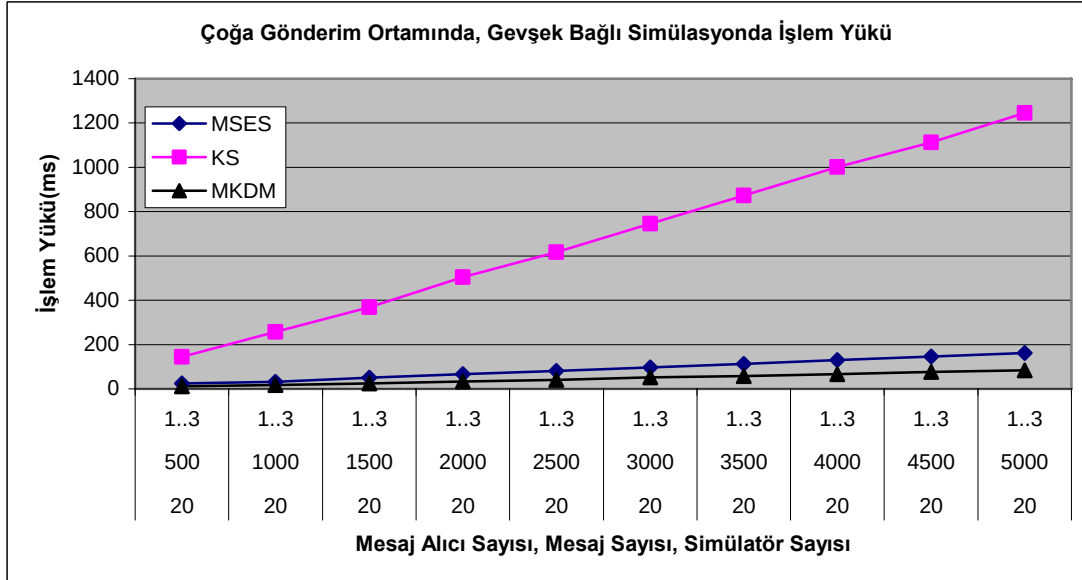
Bölüm 3'te ayrıntılı anlatılan çoğa gönderim ortamı MSES algoritmasına avantaj sağlamaktadır. MSES algoritması daha önce belirtildiği gibi mesaj alıcılarının tümüne genel bir nedensellik bilgisi ile aynı mesajı göndermektedir. Dolayısıyla çoğa gönderim ortamında ağ üzerinde akan mesaj sayısını ağ modeline bağlı olarak önemli oranda düşürebilmektedir. Bu tez kapsamında kullanılan ağ modelinde de MSES algoritmasının çoğa gönderim ortamlarında, teke gönderim ortamlarına göre %20 ile %60 daha az mesaj gönderdiği tespit edilmiştir. MSES algoritması nedensellik bilgilerini de mesajlar içinde gönderdiği için ağ üzerinde akan kontrol bilgisi miktarı da bu oranlarda düşmektedir.

Şekil 4.8'de çoğa gönderim ortamında algoritmaların kontrol bilgisi kullanımı verilmektedir. KS algoritması daha önce bahsedildiği gibi, mesajı alacak her simülör için ayrı nedensellik bilgisi oluşturduğundan hem teke hem de çoğa gönderim ortamlarında aynı sayıda mesaj ağ üzerinde akmaktadır. Dolayısıyla KS algoritmasının kontrol bilgisinde düşüş yoktur. MKDM algoritması ise çoğa gönderim ortamlarında alıcılara gönderdiği haberleşme mesajlarını azaltmaktadır. Ancak kontrol bilgilerinin konduğu sıralama mesajlarının sayısında bir azalma yoktur. Her simülör bir haberleşme mesajı için yönetici simülöre bir sıralama mesajı göndermektedir. Yönetici simülör de her haberleşme mesajı için alıcı sayısı kadar sıralama mesajı oluşturur.



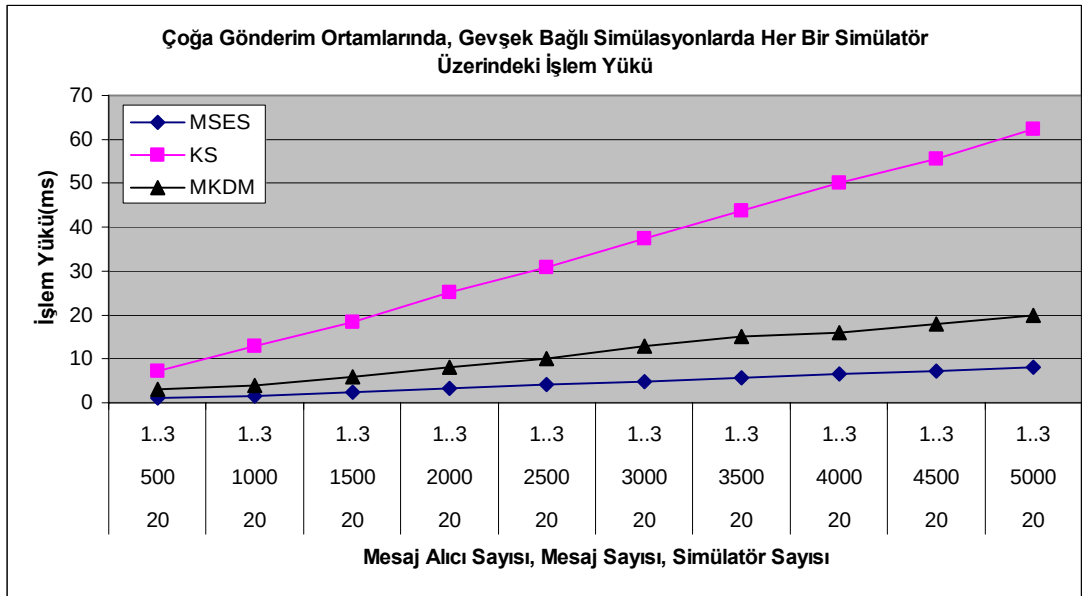
Şekil 4.8: Çoğa gönderim ortamında gevşek bağlı simülasyonda kontrol bilgisi kullanımı.

Dolayısıyla çoğa gönderim ortamında MKDM algoritması toplam mesaj sayısını ve ağ üzerinde akan toplam veriyi azaltabilmekte ama kontrol bilgisi aynı kalmaktadır. Öte yandan kullandığı kontrol bilgisi bakımından diğer algoritmalarından çok daha iyi sonuçlar vermektedir.



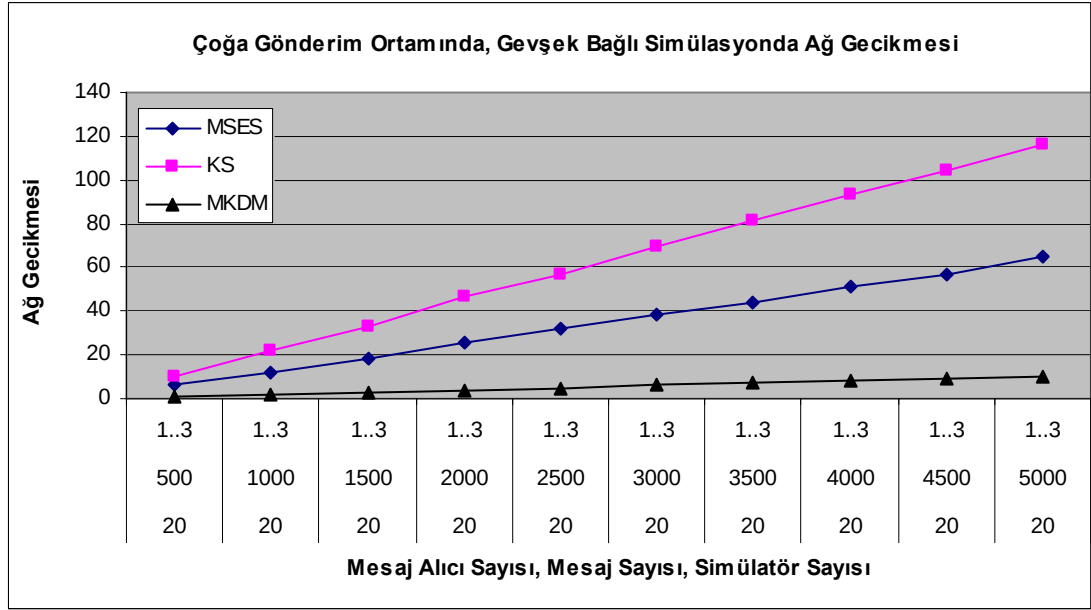
Şekil 4.9: Çoğa gönderim ortamında gevşek bağlı simülasyonda işlem yükü.

Şekil 4.9’da çoğa gönderim ortamında algoritmaların işlem yükleri gösterilmektedir. Çoğa veya teke gönderim ortamlarında ağ üzerinde akan kontrol bilgileri değişse bile algoritmaların aynı örnek modeller için nedenselliği korumak adına simülörler üzerinde çalıştıkları kontrol bilgilerinin boyutu aynıdır. Dolayısıyla MSES algoritmasında çoğa gönderim ortamında ağ üzerinde akan kontrol bilgisi teke gönderime göre azalmış olsa da, algoritmanın üzerinde çalıştığı kontrol bilgisinde azalma yoktur. Bu nedenle Şekil 4.5 ve Şekil 4.9’da sergilenen sonuçlar benzerdir. Şekil 4.10’da MKDM algoritması için yönetici simülör üzerinde, diğer algoritmalar için ise her bir simülör üzerinde oluşan işlem yükü milisaniye cinsinden görülebilmektedir.



Şekil 4.10: Gevşek bağlı simülasyonda her bir simülör üzerindeki işlem yükü.

Yukarıdaki şekilden de görüldüğü gibi MKDM algoritmasının yönetici simülörü üzerinde oluşan işlem yükü, MSES algoritmasındaki herhangi bir simülör üzerinde oluşan işlem yükünden daha fazladır. Şekil 4.11’de aynı ortam için ağ gecikmesi verilmiştir.



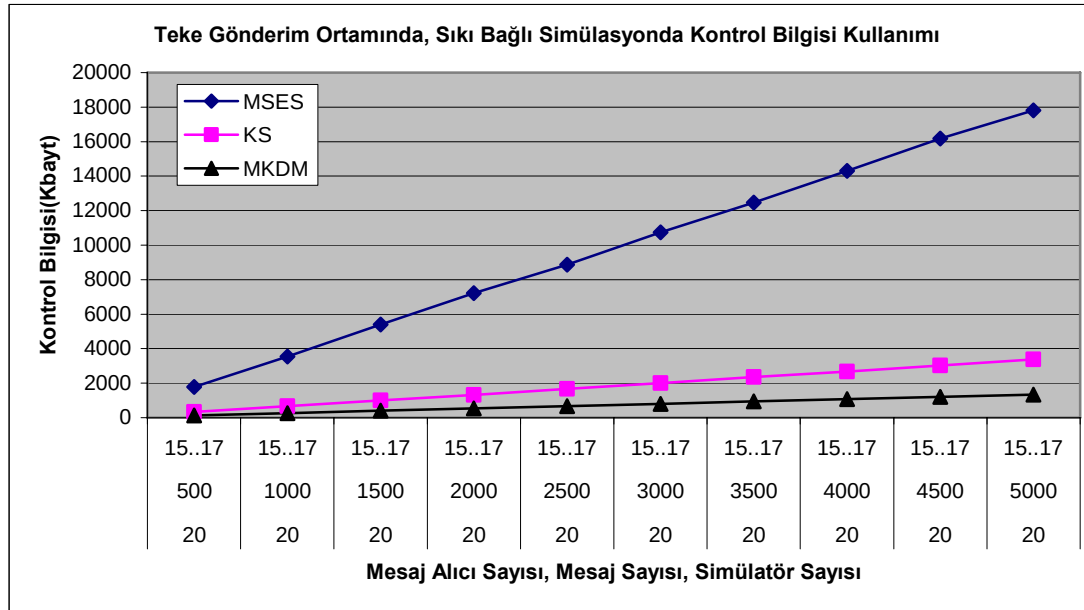
Şekil 4.11: Çoğa gönderim ortamında gevşek bağlı simülasyonda ağ gecikmesi.

Bilgisayarlar arasındaki ağ bağlantılarının bant genişlikleri DOS yazılımına bir XML dosyası içinde verilmektedir. Bu bağlantı kapasiteleri simülasyonun başlangıcındaki değerlerdir ve ağ üzerinden her mesaj geçişinde kapasite azalmakta ve gecikmeler daha da artmaktadır. Bu gözle bakıldığında ağ üzerindeki kontrol bilgilerinin arttığı durumlarda ağ gecikmeleri de artmaktadır. Çoğa gönderim ortamında, gevşek bağlı simülasyonlarda KS algoritmasının oluşturduğu ağ gecikmesi diğer algoritmalarından daha fazladır. MSES algoritması da yine kontrol bilgisindeki artışa bağlı olarak ağ gecikmesi yaratmaktadır. En az kontrol bilgisi kullanan MKDM algoritması ise ağ üzerinde en az gecikmeye sebebiyet verir. Burada dikkat edilmesi gereken durum, ağ üzerindeki farklı bağlantı noktalarında oluşan gecikmelerin toplam gecikmeye eklemesi ile tüm simülasyonda oluşan ağ gecikmesinin hesaplanmış olmasıdır. MKDM algoritmasında ise kontrol bilgisi haberleşme mesajlarında değil de sıralama mesajlarında olduğu için, ağ gecikmesinin büyük bir kısmı yönetici simülâtöre ulaşan ve yönetici simülâtörden çıkan bağlantılarda oluşmaktadır. Bu nedenle tüm ağ düşünüldüğünde az gözüken ağ gecikmesi tüm sistemde ve tüm bağlantılarda eşit dağılmamış olabilir.

4.4.2. Sıkı Bağlı Simülasyonlarda Algoritmaların Kontrol Bilgisi, İşlem Yüğü ve Ağ Gecikmesi Analizleri

Sıkı bağlı simülasyonlar, simülatörlerin birbirleri ile yoğun olarak haberleştiği ve mesaj trafiğinin fazla olduğu simülasyon tipleridir. Mesajlaşma yoğunluğunu arttırmak için deneysel çalışma çerçevesinde hem mesaj sayılarının arttığı hem de yoğun mesaj alıcılarının olduğu simülasyon modelleri kullanılmıştır. Kullanılan simülasyonlar 20 simülatörden oluşan ve mesajların 500 mesajdan 5000 mesaja doğru arttığı ve her mesajın alıcılarının da 15 ile 17 simülatör arasında eşit dağılımlı değiştiği örneklerdir.

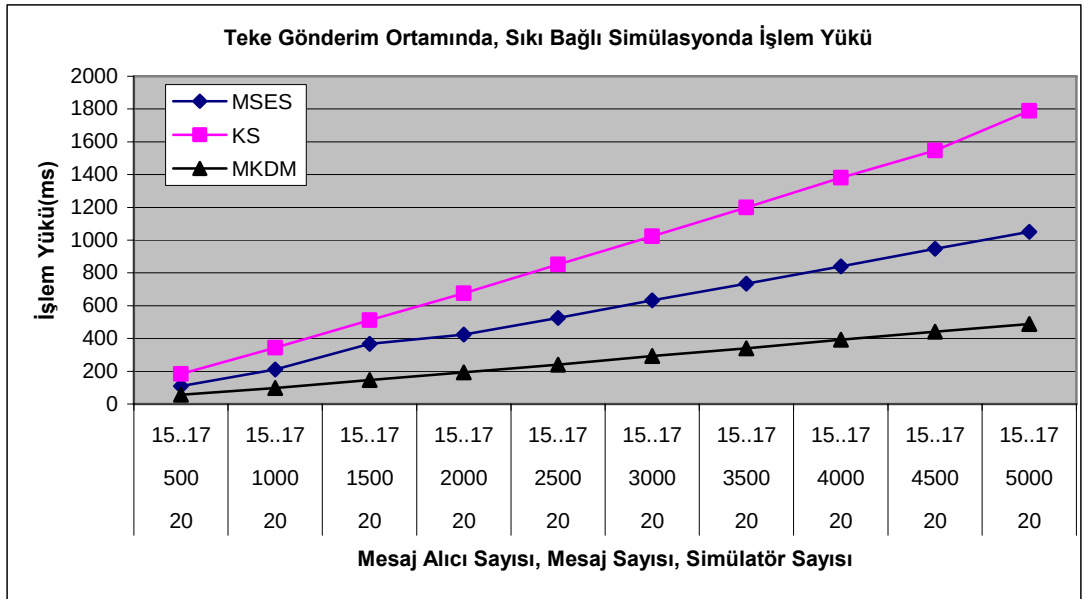
Şekil 4.12’de teke gönderim ortamında, sıkı bağlı simülasyonda algoritmaların kontrol bilgisi kullanımı anlatılmaktadır.



Şekil 4.12: Teke gönderim ortamında sıkı bağlı simülasyonda algoritmaların analizi.

Sıkı bağlı simülasyonlarda ve teke gönderim ortamında MSES algoritması ağ üzerinde akan kontrol bilgisi bakımından diğer algoritmalarından daha kötü sonuç vermektedir. Bunun en önemli nedeni mesaj içinde gönderilen ve mesaj alıcılarına göre özelleştirilmeyen genel nedensellik bilgisidir. Mesaj alıcı sayısı arttıkça kontrol bilgisi artmakta, öte yandan simülatörlere kendileri ile ilgili olmayan kontrol bilgileri ise gitmeye devam etmektedir. Sonuç olarak kontrol bilgisi artışı diğer

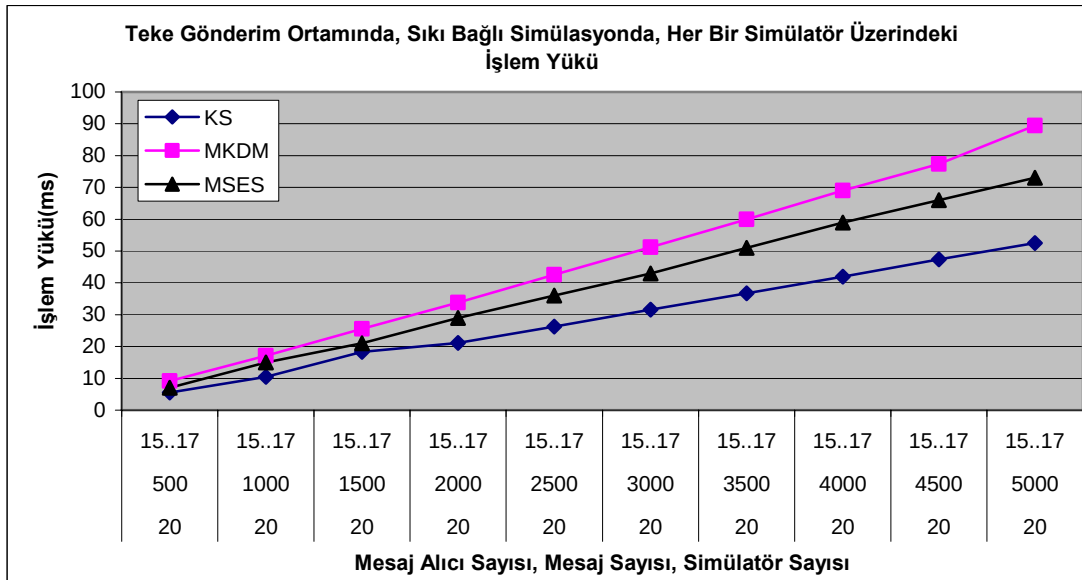
algoritmalarından daha fazla olmaktadır. KS algoritması ise mesaj alıcı sayısının arttığı ortamlarda daha az kontrol bilgisi kullanmaktadır. Simülasyondaki simülatörlerin birbirleri hakkında daha iyi bilgi sahibi oldukları bu ortamlarda KS algoritması mesajlar içinde çok daha küçük kontrol bilgisi taşıyabilmektedir. KS algoritması kontrol bilgisi küçültme işini mesaj gönderme mekanizmasına taşıdığı için mesajlar ağ üzerine çıkmadan kontrol bilgileri azaltılabilmektedir. MSES algoritması ise kontrol bilgisi eleme işini mesaj alma mekanizmasında yapmaktadır. Sistemdeki simülatörlerin birbirleri hakkında daha çok bilgiye sahip olduğu çoğa gönderim ortamında MSES algoritmasının mesaj içine koyduğu VT vektörü avantaj olmaktan çıkıp dezavantaja dönüşmektedir. MKDM algoritması ise diğer algoritmalarla göre daha az kontrol bilgisi kullanmaktadır. Nedensellik bilgisi kullanmıyor oluşu algoritmaya ciddi oranda avantaj sağlamaktadır. Şekil 4.13 aynı şartlarda algoritmaların oluşturduğu işlem yükleri görülmektedir.



Şekil 4.13: Teke gönderim ortamında sıkı bağlı simülasyonda işlem yükü.

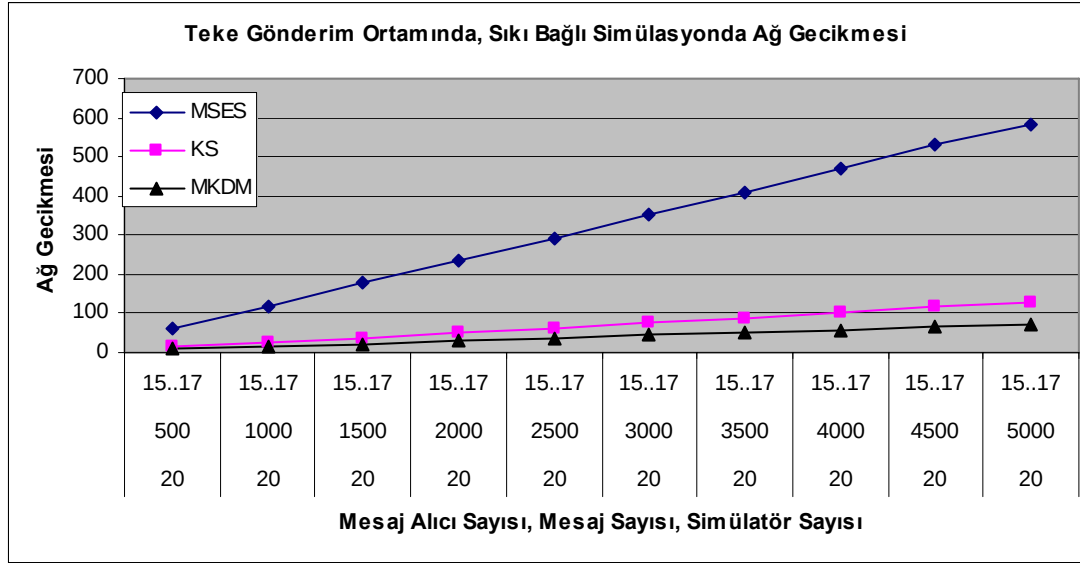
MSES algoritması KS ve MKDM algoritmalarından daha fazla kontrol bilgisini ağ üzerinde gönderiyor olmasına rağmen KS algoritmasından daha hızlı çalışmaktadır. Bunun en önemli nedeni mesaj gönderim mekanizmalarındaki farklılıklardır. KS algoritması, N simülatörden oluşan bir simülasyonda k alıcıya bir mesaj gönderecek ise, mesaj gönderim mekanizmasında en kötü şartlarda $k \cdot O(N^2)$ 'lik veri üzerinde işlem yapmaktadır. MSES algoritması ise nedensellik

vektörünü üzerinde herhangi bir işlem yapmadan doğrudan mesaj içine koymaktadır. Dolayısıyla KS algoritması simülatörler üzerinde işlem yükü oluşturarak, ağ üzerinde kontrol bilgisini azaltmakta, MSES ise simülatörler üzerinde az iş yapmasına bağlı olarak, ağ üzerinde daha fazla kontrol bilgisi akmasına neden olmaktadır. MKDM algoritması ise hem ağ üzerinde en az seviyede kontrol bilgisi kullanmakta hemde simülatörler üzerinde basit bir karşılaştırma işlemi dışında işlem yükü oluşturmamaktadır. MKDM algoritmasının neden olduğu işlem yükünün büyük bir kısmı sadece yönetici simülatöre aittir. Şekil 4.14'te MKDM algoritması için yönetici simülatör, diğer algoritmalar için her bir simülatör üzerindeki işlem yükleri gösterilmiştir.



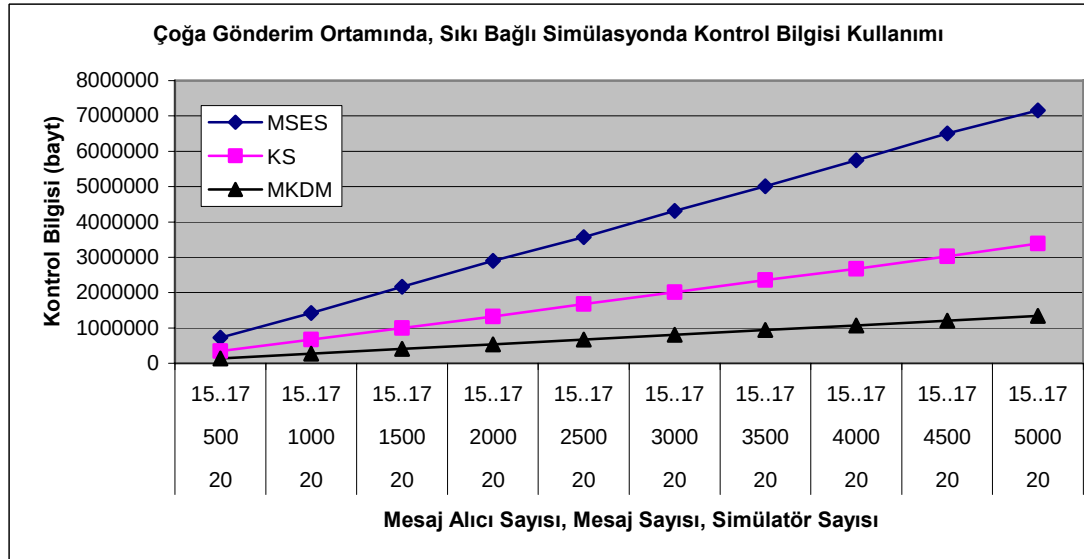
Şekil 4.14: Sıkı bağlı simülasyonda her bir simülatör üzerindeki işlem yükü.

KS algoritması daha önceden bahsedilen nedenden dolayı simülatörler üzerinde en fazla iş yüküne neden olan algoritmadır. MKDM algoritmasının ise yönetici simülatörü bu örnekte tüm işlem yükünün %12.5 ile %15'ine tek başına neden olmaktadır. Diğer simülatörlerin her biri ise tüm işlem yükünün %3.5'ine neden olmaktadır. Sonuç olarak yönetici simülatör MSES algoritmasındaki herhangi bir simülatörden daha fazla iş yapıyor olmasına rağmen, MKDM algoritmasındaki diğer simülatörler çok az iş yüküne sahiptirler. Şekil 4.15'te aynı şartlar altında algoritmaların ağ üzerinde yarattıkları gecikmeler görülebilmektedir.



Şekil 4.15: Teke gönderim ortamında sıkı bağlı simülasyonda ağ gecikmesi.

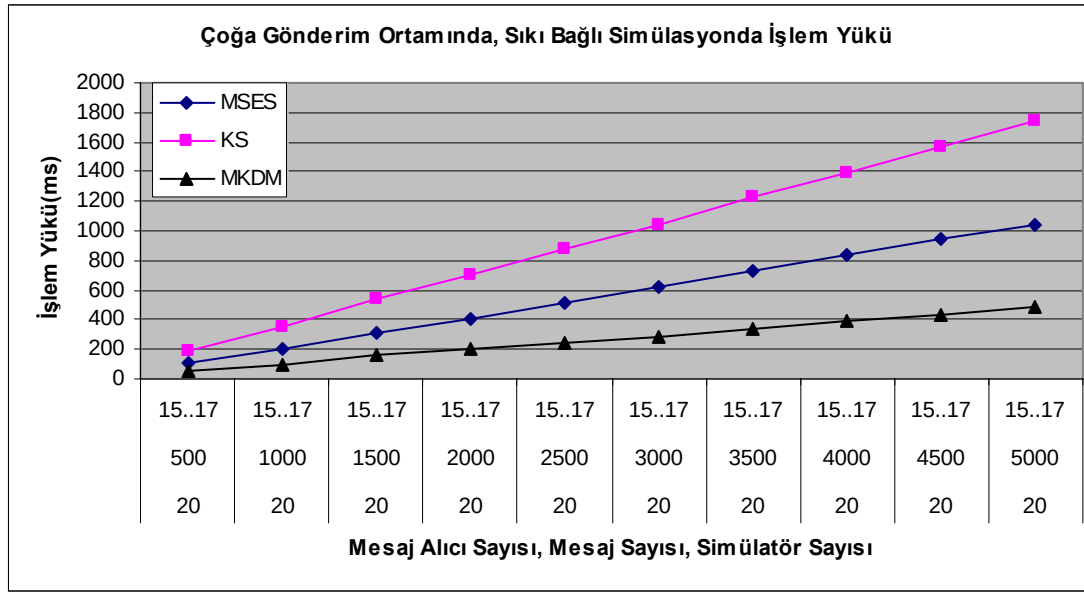
Bütün algoritmalar kullandıkları kontrol bilgisine paralel bir ağ gecikmesi sergilemektedirler. Senaryolar çoğa gönderim ortamlarında denendiğinde ise MSES algoritması ile KS algoritması arasındaki fark kapanmakta ve MSES algoritması teke gönderime göre daha az mesaj kullanarak haberleşebildiği için kullanılan kontrol bilgisi miktarı da düşmektedir.



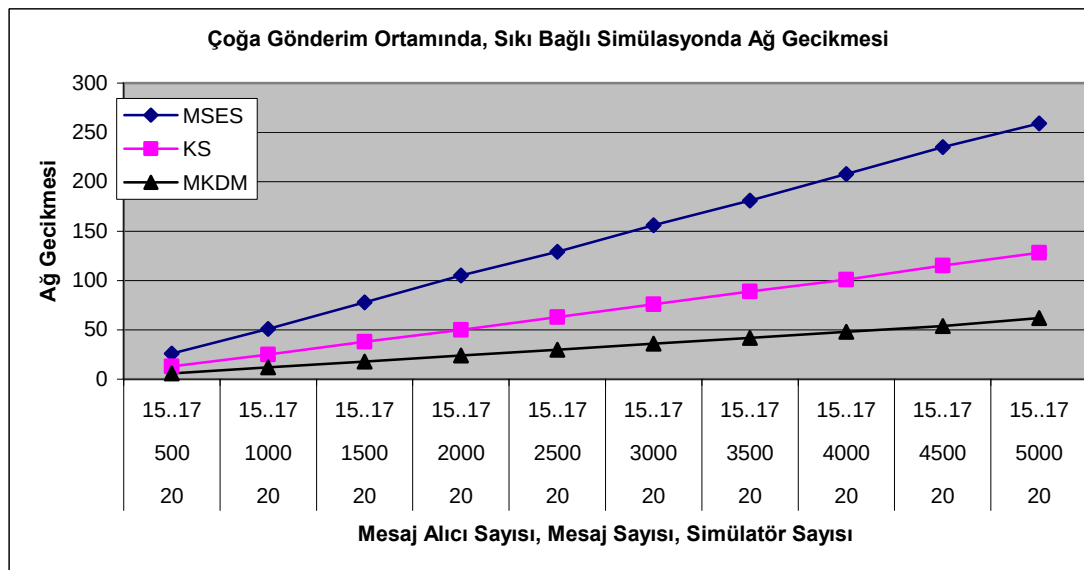
Şekil 4.16: Çoğa gönderim ortamında sıkı bağlı simülasyonda kontrol bilgisi kullanımı.

KS ve MKDM algoritmalarının kontrol bilgisi kullanımı ise teke gönderim ortamındaki ile aynı kalmaktadır. Sonuçlar

Şekil 4.16’da incelenebilir. Simülörler üzerinde oluşan işlem yüküne bakıldığında ise, KS algoritması en çok işlem yüküne neden olan algoritmadır. MKDM algoritması da en az işlem yüküne neden olmaktadır. Sonuçlar Şekil 4.17’de görülebilir.



Şekil 4.17: Çoğa gönderim ortamında sıkı bağlı simülasyonda işlem yükü.



Şekil 4.18: Çoğa gönderim ortamında sıkı bağlı simülasyonda ağ gecikmesi.

Ağ üzerinde oluşan gecikmeler incelendiğinde de ise yine kontrol bilgileri ile doğru orantılı sonuçlar elde edilmektedir.

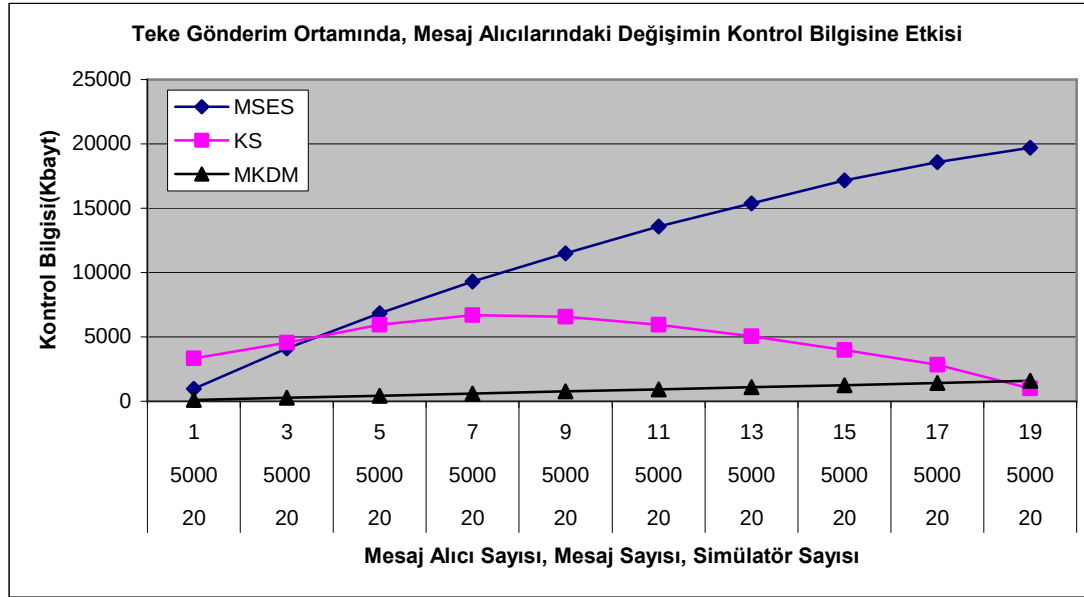
Şekil 4.18’de çoğa gönderim ortamında algoritmaların oluşturduğu ağ gecikmeleri sergilenmiştir.

Sonuç olarak sıkı bağlı simülatörlerde KS algoritması MSES algoritmasına göre gereksiz ve eski nedensellik bilgilerini elemeye daha başarılıdır. Sıkı bağlı simülasyonun, simülatörlerin birbirlerinden daha iyi haberdar olma özelliğini MSES’ten daha iyi kullandığı için mesajlar içinde gönderdiği kontrol bilgisini ciddi oranda küçülebilmektedir. Ancak kontrol bilgisini küçültme işlemi sırasında harcadığı zaman onu diğer algoritmalarından daha yavaş bir algoritma yapmaktadır. MSES algoritması ise çoğa gönderim ortamlarında ağ üzerinde akan mesaj sayısını azaltabildiği için kullandığı kontrol bilgisini de ciddi oranda düşürebilmektedir. MKDM algoritması ise ağda akan kontrol bilgisi, işlem yükü ve ağ gecikmesi kıstaslarında diğer algoritmalarından daha iyi çıkmaktadır. Ancak MKDM algoritmasında işin büyük bir kısmı yönetici simülatöre yönlendirildiği için,

Şekil 4.18’te görülen ağ gecikmesinin önemli bir oranının (%12.5-%15) sadece yönetici simülatörden kaynaklandığı düşünülebilir.

4.4.3. Mesaj Alıcılarındaki Değişimin Kullanılan Kontrol Bilgisi ve İşlem Hızına Etkisinin Analizi

Bu bölümde simülatör ve mesaj sayısı sabit tutularak mesaj alıcılarının arttırılmasının algoritmalar üzerindeki etkisi analiz edilmiştir. 20 simülatörden oluşan bir simülasyon 5000 mesaj için çalıştırılmış ve her seferinde mesaj alıcıları arttırılmıştır. Böylece simülasyonun yapısı gevşek bağlı simülasyondan sıkı bağlı simülasyona doğru kaydırılmış ve bu değişime karşı algoritmaların davranışları incelenmiştir. Aynı zamanda simülasyon hakkında az bilgiye sahip olmaktan çok bilgiye sahip olmaya doğru geçişte algoritmaların değişime nasıl cevap verdiği ölçülmeye çalışılmıştır. Şekil 4.19’da teke gönderim ortamında, mesaj alıcılarındaki değişimin kontrol bilgileri kullanımına etkisi gösterilmektedir.

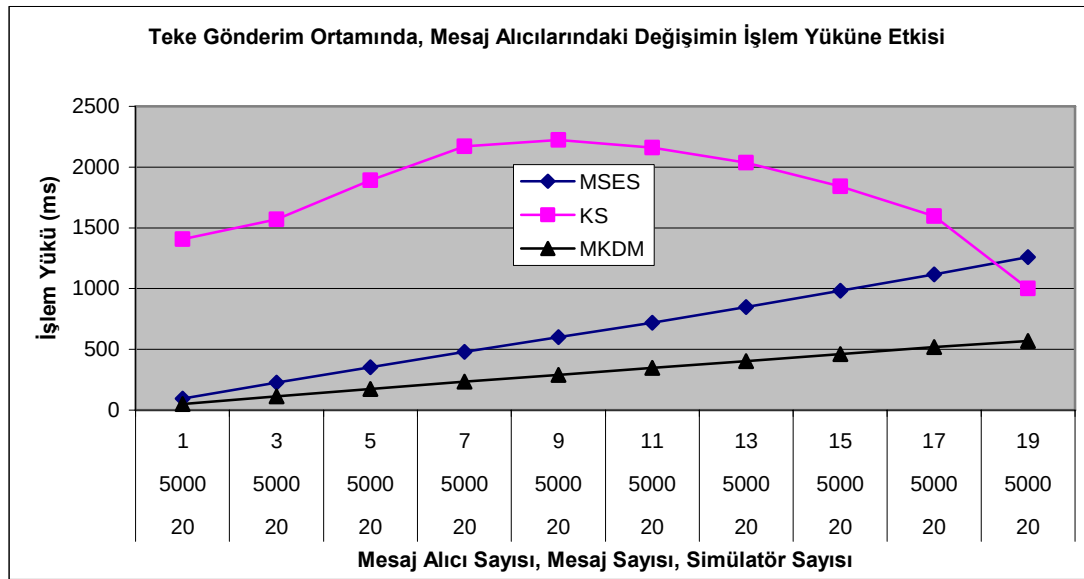


Şekil 4.19: Teke gönderim ortamında mesaj alıcılarındaki değişimin kontrol bilgisine etkisi.

KS algoritması mesaj alıcı sayısının artması ile birlikte daha fazla mesaj göndermeye başladığı için kullandığı kontrol bilgisi de artmaktadır. Bu nedenle tek alıcılı mesajlardan 7 alıcılı mesajlara doğru ilerlendiğinde kullanılan kontrol bilgisinin arttığı görülür. Mesaj alıcı sayısının daha fazla artması ise KS algoritmasında kontrol bilgisinin tekrar düşmesine neden olmaktadır. Bunun en önemli nedeni daha önce bahsedilen mesaj gönderme mekanizmasında alıcı simülöre özel kontrol bilgisinin hazırlanıyor olmasıdır. Ayrıca mesajların simülörlerden büyük bir çoğunluğuna gitmesi, simülasyon içinde daha iyi haberleşmeye ve daha az kontrol bilgisi kullanarak mesajları sıralamaya imkan tanımaktadır. Kontrol bilgisini mesaj koymadan önce küçültmek KS'ye ağ üzerinde daha az bilgi gönderme imkanı tanır. MSES algoritması ise simülasyon ortamının değişimine tüm örneklerde benzer tepki vermiştir. Mesaj alıcı sayısının artması doğrudan kontrol bilgisinin artmasına neden olmuştur. Algoritmanın nedensellik bilgisini küçültmek için uyguladığı yöntem mesaj alma mekanizması içinde yer aldığından ve tüm alıcılara ayırım yapmadan aynı nedensellik bilgisini gönderdiğinden kontrol bilgisi alıcı sayısının artışı ile doğru orantılı olarak artmaktadır. Artış miktarında ise azalma görülmektedir. Bunun nedeni ise algoritmanın herkesin birbirinden haberdar olmaya başladığı durumlarda öncesine

göre biraz daha az kontrol bilgisi kullanma gereği duymasıdır. KS algoritmasının MSES algoritmasından daha akıllı olduğu ve simülasyon ortamının değişen durumuna karşı daha kararlı davrandığı söylenebilir.

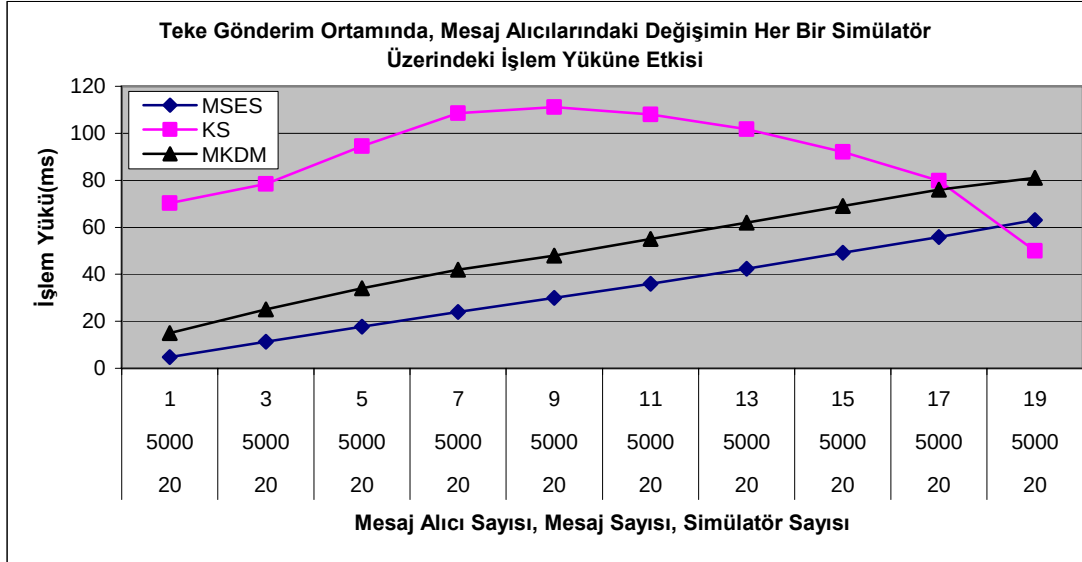
MKDM algoritmasında MSES algoritmasında olduğu gibi alıcı sayısına bağlı olarak sıralama mesajı artmakta ve dolayısıyla daha fazla kontrol bilgisi ağ üzerinde akmaktadır. MSES algoritmasından farklı olarak artış miktarı her noktada aynıdır çünkü MKDM algoritması simülatörlerin birbirleri hakkında ne kadar bilgi sahibi oldukları bilgisini kullanmamaktadır. Ancak diğer algoritmaların kullandığı nedensellik bilgisinden çok daha küçük olan kontrol bilgisi onu yine en az kontrol bilgisi kullanan algoritma yapmaktadır. Ancak Şekil 4.19’da da görüldüğü gibi 20 simülatör, 5000 mesaj ve 19 alıcı senaryosunda KS algoritmasının kontrol bilgisi kullanımı MKDM algoritmasından daha azdır. Bu durumda algoritmanın 19, 20 alıcılı senaryolarda en iyi sonucu vereceği söylenebilir. Algoritmaların teke gönderim ortamında, simülatörler üzerinde neden olduğu işlem yükleri Şekil 4.20’de gösterilmektedir.



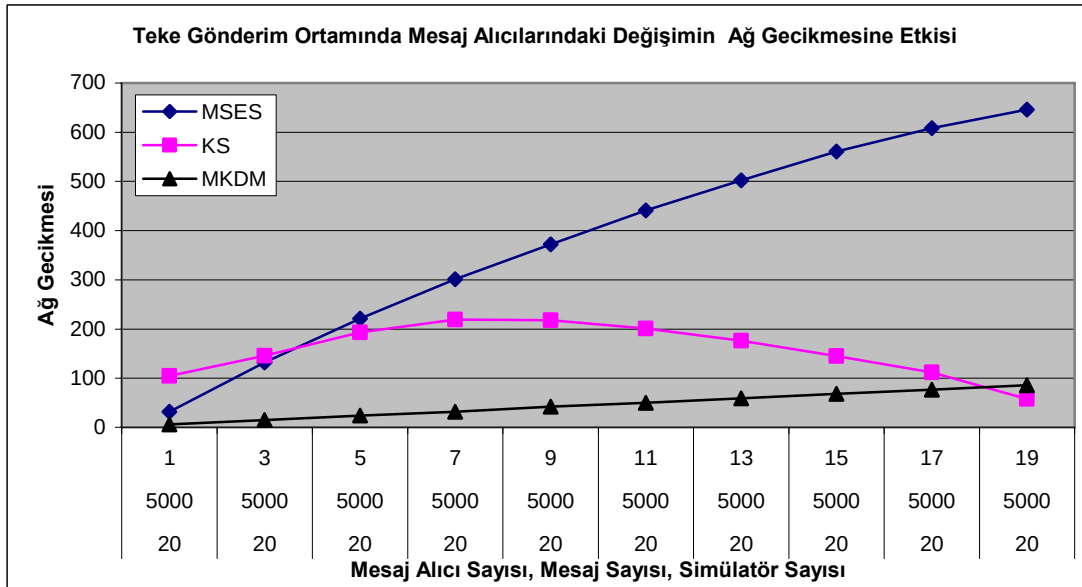
Şekil 4.20: Teke gönderim ortamında mesaj alıcılarındaki değişimin işlemi yüküne etkisi.

Simülatörler üzerinde oluşan işlem yükleri kullandıkları kontrol bilgileri ile de doğru orantılıdır. MSES algoritması KS algoritmasından daha hızlı bir algoritmadır. İşlem yükü bakımından simülatörler üzerinde KS algoritmasından daha az yük

oluşturmaktadır. En az işlem yükü oluşturan algoritma ise MKDM algoritmasıdır. İşlem yükünün büyük bir kısmı yönetici simülatöre aittir ancak o simülatörün tek yaptığı ise gelen mesajlar için sıradaki zaman değerini vermektir. MKDM algoritmasındaki yönetici simülatörün ve diğer algoritmalarındaki herhangi bir simülatörün üzerinde oluşan işlem yükü ise Şekil 4.21’de görülebilir.



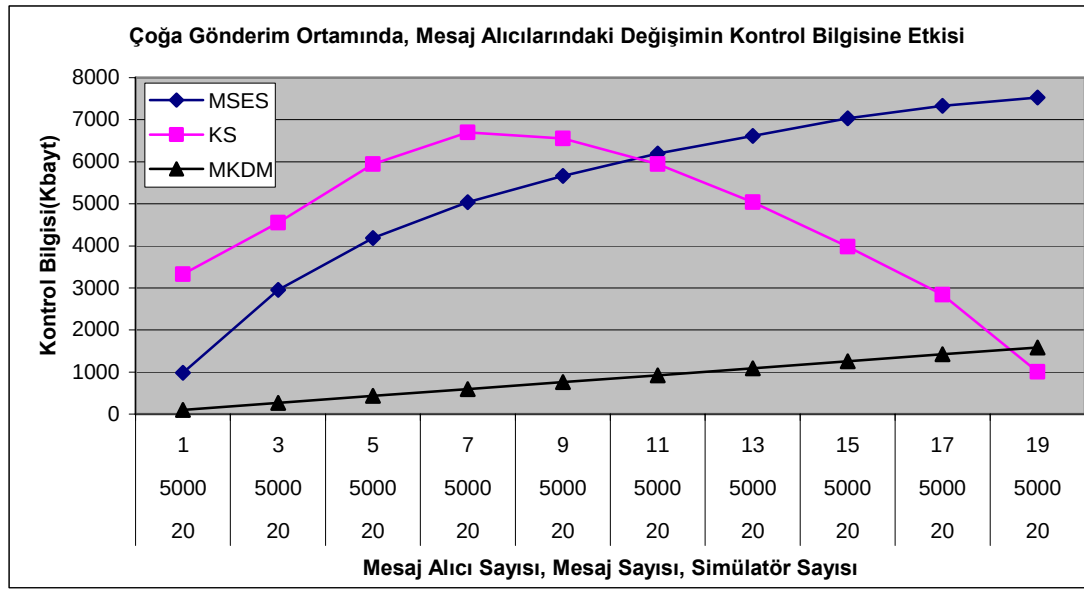
Şekil 4.21: Mesaj alıcılarındaki değişimin simülatör başına işlem yükü etkisi.



Şekil 4.22: Mesaj alıcılarındaki değişimin ağ gecikmesine etkisi.

Teke gönderim ortamında, algoritmaların ağ üzerinde oluşturdukları gecikme Şekil 4.20’de sergilenmektedir. Algoritmaların ağ üzerinde oluşturduğu gecikme, mesajları sıralamak için ihtiyaç duydukları kontrol bilgileri ile doğru orantılıdır. MSES algoritması özellikle mesaj alıcılarının çok olduğu durumlarda ciddi ağ gecikmesine neden olabilmektedir.

Çoğa gönderim ortamlarında da algoritmalar benzer davranışlar sergilemektedir. Şekil 4.23’te çoğa gönderim ortamında, algoritmaların kontrol bilgisi kullanımı anlatılmaktadır.



Şekil 4.23: Çoğa gönderim ortamında mesaj alıcılarındaki değişimin kontrol bilgisine etkisi.

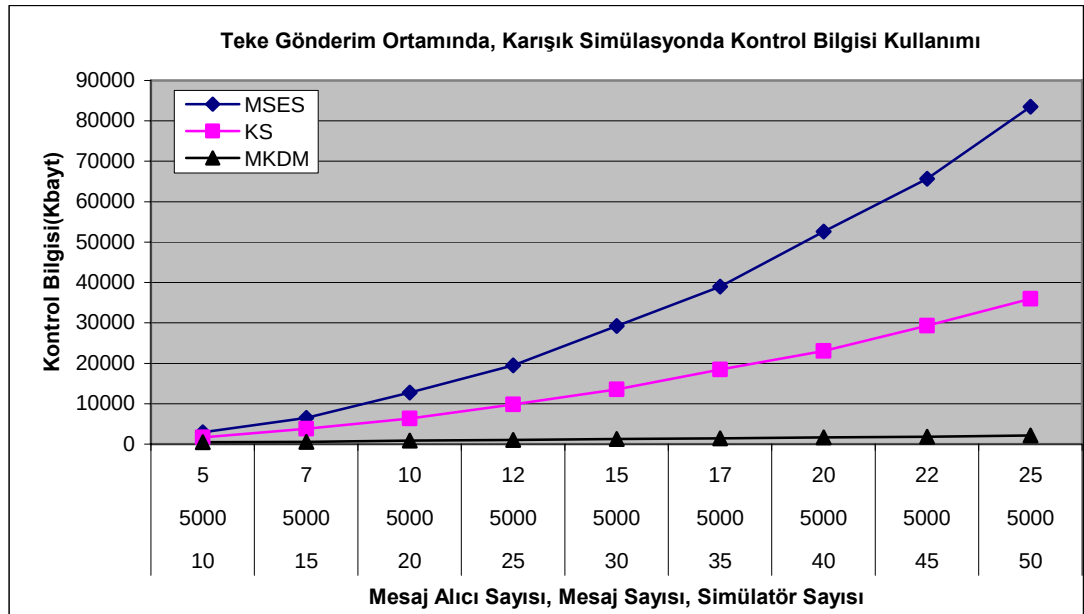
KS ve MKDM algoritmaları kullanılan kontrol bilgileri, ağ üzerinde oluşan gecikmeler ve işlem yükleri bakımından teke gönderim ortamı ile benzer sonuçlar üretmektedirler. Bunun en önemli nedeni ise her iki ortamda da aynı miktarda kontrol bilgileri kullanıyor olmalarıdır. KS algoritması daha önce bahsedildiği gibi alıcı sayısının artışına bağlı olarak kontrol bilgisi üzerinde daha iyi eleme yapabilmekte ve kullanımını gittikçe düşürebilmektedir. MSES algoritması ise çoğa gönderim ortamında, ağ üzerinde akan mesaj miktarını azaltabildiği için, hem kullandığı kontrol bilgisi miktarını hem de ağ üzerindeki gecikme miktarını düşürebilmektedir. Ancak yine de mesaj alıcılarının arttığı durumlarda en kötü

sonucu vermektedir. Sonuç olarak mesaj alıcı sayısının simülasyondaki simülatör sayısının yarısından az olduğu durumlarda MSES, çok olduğu durumlarda ise KS kullanılan kontrol bilgisi bakımından daha iyi sonuç vermektedir. Algoritmaların yukarıdaki mesaj gönderme işlemleri sırasında harcadıkları zamanlar ise teke gönderim ortamındaki ile benzerlik göstermektedir.

4.4.4. Sistemdeki Simülatörlerin Artışına Paralel Olarak Algoritmaların Ölçeklenebilirliğinin Analizi

Bu bölümde karışık simülasyonlarda algoritmaların ölçeklenebilirlikleri test edilecektir. Karışık simülasyonlar mesajların simülasyondaki simülatörlerin yarısına gönderildiği, başka bir deyişle her mesajın simülasyonun ortalama olarak yarısını dolaştığı simülasyonlardır. Bu çalışmada 10 simülatörden oluşan bir simülasyon ile başlanılacak ve kademeli olarak büyütülecektir. Çalışmada amaçlanan simülasyon büyüklüğünün artmasına müteakip algoritmaların kararlılıklarını ve ölçeklenebilirliklerini analiz etmektir. Senaryolar teke ve çoğa gönderim ortamları için koşturulmuştur.

Şekil 4.24'te teke gönderim ortamında, karışık simülasyonda kontrol bilgisi kullanımı gösterilmektedir.

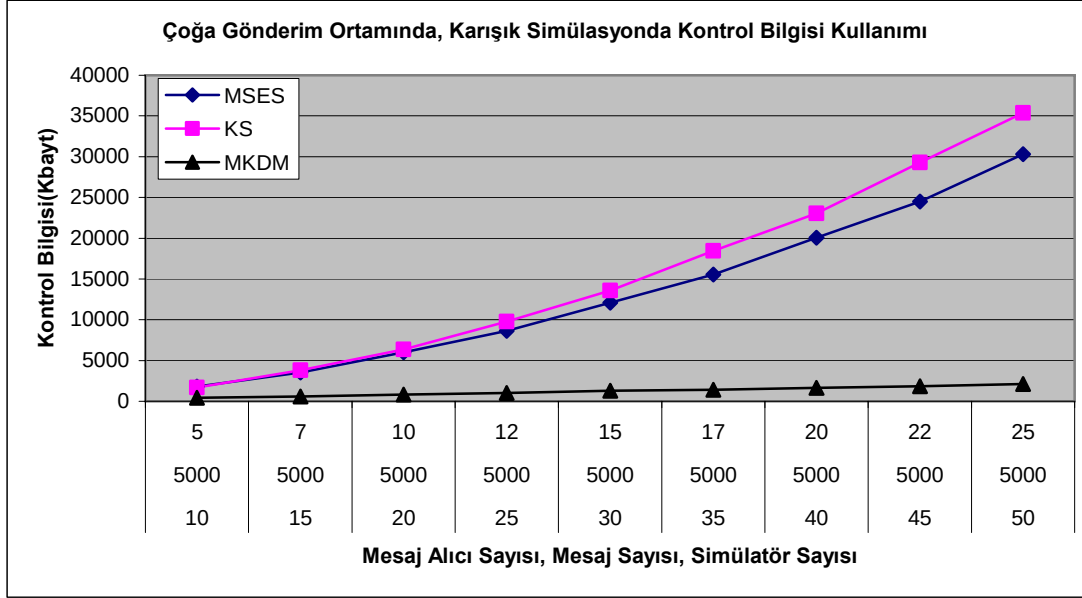


Şekil 4.24: Teke gönderim ortamında, karışık simülasyonda kontrol bilgisi kullanımı.

Örneklemelemlerde hem simülatör sayısının hem de alıcı sayısının artması KS ve MSES algoritmalarında nedensellik vektörlerinin ve mesaj sayılarının aynı anda artmasına neden olmaktadır. Dolayısıyla MSES ve KS algoritmaların kullandıkları kontrol bilgileri parabolik olarak büyüme göstermektedir. Sadece mesaj alıcı sayısının artmasının algoritmalar üzerindeki etkileri önceki bölümlerde incelenmiştir. Algoritmalar mesaj alıcı sayının artışına doğrusal artış ile tepki vermişlerdir. Mesaj alıcı sayısı ile beraber simülasyonun büyümesine ise parabolik artış ile tepki vermektedirler. Bu artışın en önemli nedeni ağ üzerinde artan mesaj sayısı ile birlikte nedensellik vektörlerinin de büyüyor olmasıdır. Nedensellik vektörlerinin boyutunun ortamdaki simülatör sayısı ile belirlendiği ve en kötü şartlarda karesi ile orantılı olduğu düşünülürse algoritmaların kontrol bilgisi kullanımının parabolik olarak arttığı söylenebilir. Öte yandan MSES algoritmasının kontrol bilgisi kullanımı KS algoritmasından daha hızlı artmaktadır.

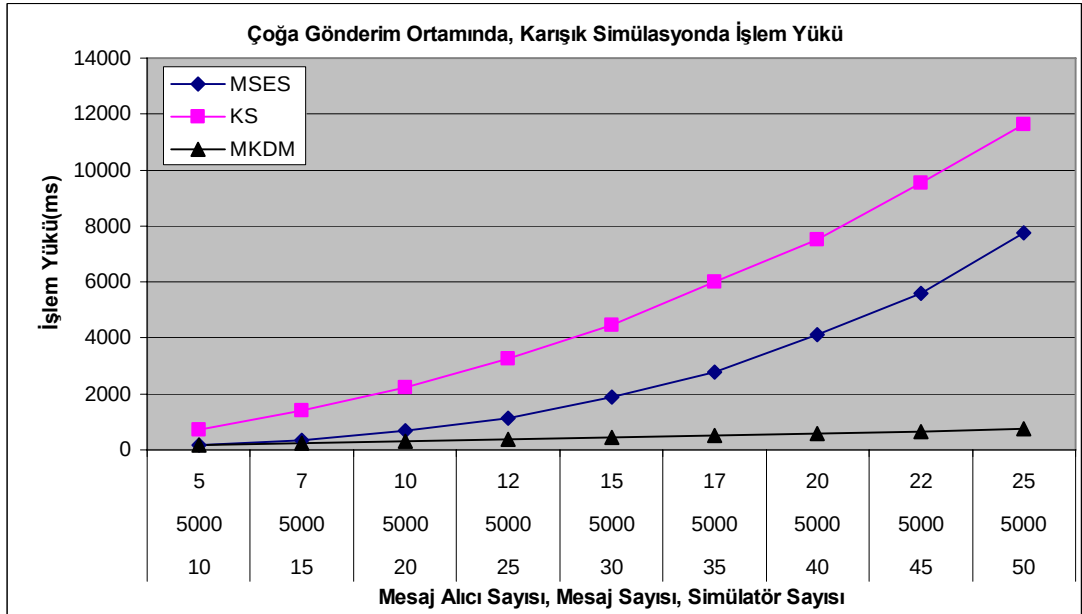
MKDM algoritması ise simülasyonun büyüklüğünden etkilenmez çünkü haberleşme mesajları içinde kontrol bilgisi taşımamaktadır. Sadece sıralama mesajları içinde taşınan kontrol bilgisi ise mesaj alıcı sayısı ile artmaktadır. Dolayısıyla diğer algoritmaları tetikleyen mesaj sayısı ve nedensellik vektörünün beraber artması MKDM algoritmasında sadece alıcı sayısının artması olarak kendini gösterir. Bu nedenle algoritmanın kontrol bilgisi kullanımı simülasyon büyüdükçe doğrusal olarak artmaktadır. Çoğa gönderim ortamlarında da benzer sonuçlar çıkmaktadır.

Şekil 4.25'te sonuçlar sunulmuştur. MSES algoritması bu ortamda mesaj kullanım sayısını teke gönderim ortamına göre azaltabildiği için KS algoritmasına göre daha az kontrol bilgisi kullanmaktadır. Ancak iki algoritmanın da kontrol bilgisi artış oranı hala paraboliktir. MKDM algoritması ise teke ve çoğa gönderim ortamlarında aynı miktarda kontrol bilgisine ihtiyaç duyduğu için önceki senaryo ile benzer durumu sergilemektedir.



Şekil 4.25: Çoğa gönderim ortamında, karışık simülasyonda kontrol bilgisi kullanımı.

Çoğa gönderim ortamında algoritmalar üzerinde oluşan işlem yükü Şekil 4.26’da belirtilmiştir.

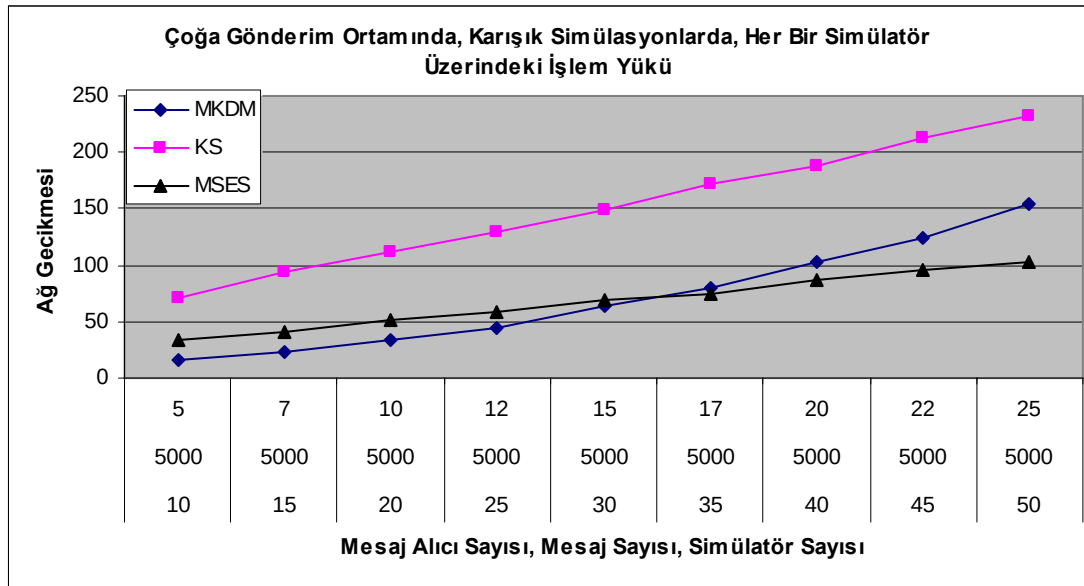


Şekil 4.26: Çoğa gönderim ortamında, karışık simülasyonda işlem yükü.

Daha önceki analizlerde KS algoritmasının MSES algoritmasından daha yavaş çalıştığı gösterilmiştir. Şekilde, KS algoritmasının işlem yükünün artış oranının MSES algoritmasının işlem yükünün artış oranından daha fazla olduğu

görülmektedir. KS algoritmasının mesaj gönderme mekanizmasında gerçekleştirdiği nedensellik vektörü küçültme işlemleri algoritmayı yavaşlatmaktadır. Ancak her iki algoritmanın simülatörler üzerinde neden oldukları işlem yükü parabolik olarak artmaktadır. MKDM algoritması ise diğer iki algoritmadan çok daha az işlem yüküne neden olmaktadır ve işlem yükü değişimi sadece mesaj alıcı sayısına bağlı olduğu için doğrusal artmaktadır. MKDM algoritması simülasyon büyüklüğüne karşı diğer iki algoritmaya göre çok daha ölçeklenebilir bir algoritmadır.

Yukarıdaki analizde çoğa gönderim ortamında, karışık simülasyonlarda, simülatörler üzerindeki toplam işlem yüküne bakılmıştır. Her bir simülatör üzerinde oluşan işlem yüküne bakıldığında ise Şekil 4.27’de sunulan sonuçlar elde edilmektedir.

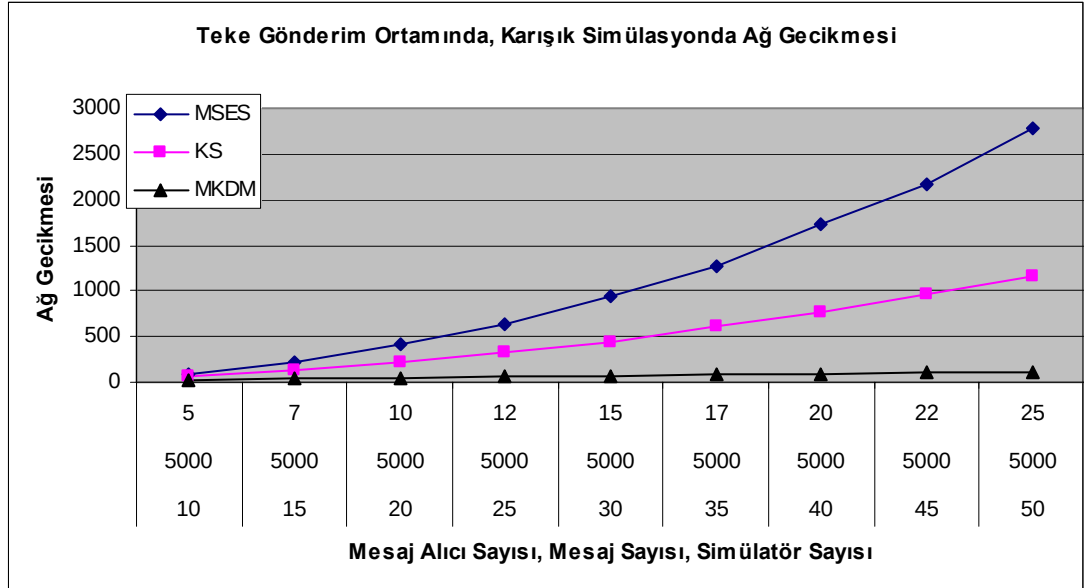


Şekil 4.27: Çoğa gönderimde, karışık simülasyonlarda, her bir simülatör üzerindeki işlem yükü.

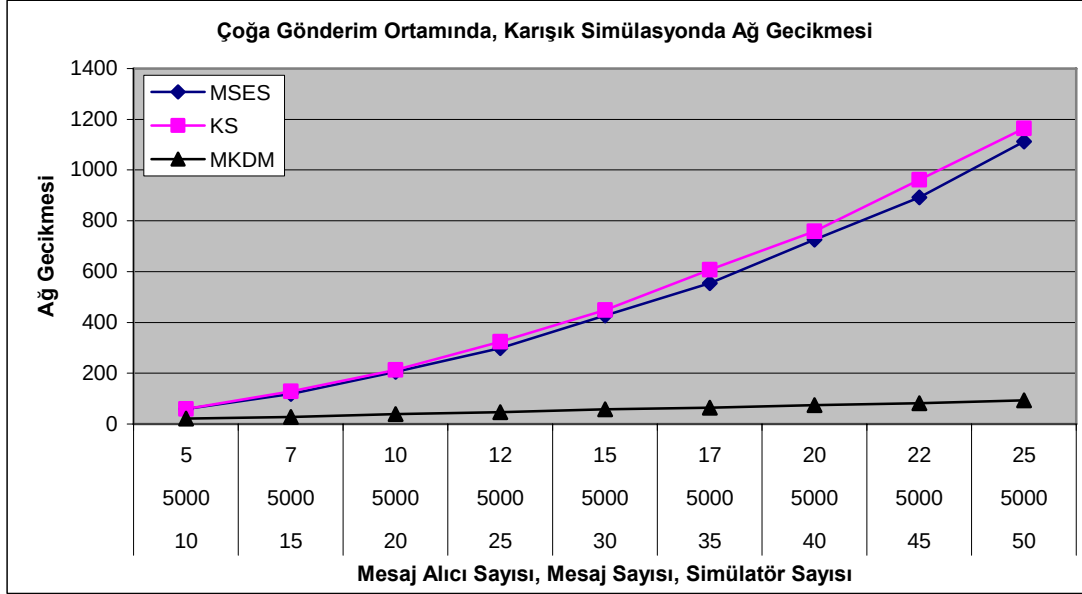
Bu şekilde, MSES ve KS algoritmalarındaki her bir simülatör üzerinde oluşan işlem yükü ile, MKDM algoritmasındaki yönetici simülatör üzerinde oluşan işlem yükleri karşılaştırılmıştır. Simülatörler üzerindeki toplam işlem yükleri parabolik bir şekilde artan KS algoritmasının her bir simülatörü üzerindeki işlem yükü artışı ise doğrusaldır. Bu değişim şu şekilde açıklanabilir. Her bir simülatör üzerinde oluşan işlem yükü mesaj sayısına ve nedensellik vektörünün büyüklüğüne bağlıdır. Mesaj

alıcı sayısının sabit bir şekilde arttığı göz önünde bulundurulursa, doğrusal artışı koruyabilmek için nedensellik vektörünün büyüklüğü de her bir örnekte $O(N)$ 'i geçmemelidir. Bu da KS algoritmasında, karışık simülasyon örneklerinde nedensellik vektörünün büyüklüğünün $O(N)$ seviyesinde kaldığını göstermektedir. MSES algoritmasında ise her bir simülâtörün üzerindeki işlem yükü diğer algoritmalarından daha alt seviyeden başlamış olmasına rağmen parabolik bir artış göstererek hızla yükselmekte ve MKDM algoritmasındaki yönetici simülâtörünün neden olduğu işlem yükü miktarını geçmektedir. Parabolik davranış sergilemesi algoritmanın kullandığı nedensellik vektörünün karışık simülasyon için $O(N^2)$ büyüklüğünde olduğunu göstermektedir. MKDM algoritmasındaki yönetici simülâtör ise mesaj alıcı sayısındaki artışa paralel olarak doğrusal olarak artmakta ve büyük simülasyon modelleri için en iyi sonucu vermektedir. Dolayısıyla MKDM algoritmasında simülasyon içindeki simülâtörler arasında en fazla kaynak tüketimine neden olan yönetici simülâtörün, diğer algoritmalarındaki herhangi bir simülâtörden büyük simülasyonlar için daha az kaynak tükettiği söylenebilir.

Algoritmaların teke ve çoğa gönderim ortamlarında ağ üzerinde yarattıkları gecikmeler ise Şekil 4.28 ve Şekil 4.29'te gösterilmektedir.



Şekil 4.28: Teke gönderim ortamında, karışık simülasyonda ağ gecikmesi.



Şekil 4.29: Çoğa gönderim ortamında, karışık simülasyonda ağ gecikmesi.

KS ve MSES algoritmalarının ağ üzerinde yarattıkları gecikmeler simülasyonun büyümesi ile birlikte parabolik olarak artmaktadır. MSES algoritması çoğa gönderim ortamında, teke gönderim ortamına göre mesaj kullanımını ve dolayısıyla ağ gecikmesini düşürebilse bile artış oranı hala paraboliktir.

MKDM algoritması ise her iki ortamda da en az ağ gecikmesine sebebiyet veren algoritmadır. Algoritmanın ağ üzerinde yarattığı gecikme miktarı simülasyon büyüklüğüne bağlı olarak doğrusal bir şekilde artmaktadır.

Bu bölümde MSES, KS ve MKDM algoritmaları üzerinde deneysel analizler yapılmıştır. Algoritmalar simülatör sayısının, mesaj sayısının, mesaj alıcı sayısının, yayın ortamlarının değiştiği çeşitli senaryolarda; kullandıkları kontrol bilgisi, simülatörlerde neden oldukları işlem yükü ve ağ üzerinde oluşturdıkları gecikmeler bakımından karşılaştırılmış ve nedenleri üzerinde konuşulmuştur. Sonraki bölümde elde edilen sonuçlar bir özet şeklinde anlatılacaktır.

5. SONUÇLAR

MSES, KS ve MKDM algoritmaları nedensel sıralı mesajlaşma algoritmalarıdır. MSES ve KS, dağıtık sistemlerde mesajların mantıksal sıralarının bozulmadan işlenmesini sağlamak üzere geliştirilmişlerdir. MKDM algoritması ise tez kapsamında önerilen mesaj sıralama algoritmasıdır. Mesaj iletiminde kullanılan kontrol bilgisi açısından istemci-sunucu mimarisine göre tasarlanan bir NSM algoritmasıdır.

Algoritmalar 4 ana kıstas altında 2 farklı ortamda test edilmiştir. Bu kıstaslar gevşek simülasyon modelleri, sıkı simülasyon modelleri, alıcı sayısının artırıldığı modeller ve karışık simülasyon modelleridir. Her bir model de teke ve çoğa gönderim ortamları için ayrı ayrı çalıştırılmıştır.

Gevşek bağlı simülasyonlar simülatörlerin birbirleri hakkında çok fazla bilgiye sahip olmadığı, mesajlaşma oranının düşük olduğu simülatörlerdir. Bu tarz simülasyonlarda MSES algoritması KS algoritmasından daha az kontrol bilgisi kullanmaktadır. Bunun en önemli nedeni ise MSES algoritmasının fazladan kullandığı VT zaman vektörünün algoritmaya daha etkin nedensellik vektörü oluşturma imkanı vermesidir. KS algoritması ise gevşek bağlı simülasyonlarda kötü performans sergilemektedir çünkü simülatörlerin birbirleri hakkında fazla bilgiye sahip olmamaları ağ üzerinde daha fazla kontrol bilgisinin akmasına neden olmaktadır. MKDM algoritması ise iki farklı mesaj kullanmaktadır. İlki simülatörler arasındaki haberleşmeyi sağlamak için kullanılan haberleşme mesajlarıdır. İkincisi ise haberleşme mesajlarını sıralamak için kullandığı sıralama mesajıdır. MKDM, haberleşme mesajları içinde kontrol bilgisi barındırmaz. Kontrol bilgileri sadece sıralama mesajları içinde akar ve diğer algoritmaların kontrol bilgilerine kıyasla çok küçüktür. Gevşek bağlı simülasyonlarda en az kontrol bilgisi kullanan MKDM algoritmasıdır.

MSES algoritması bir mesajın bütün alıcılarına aynı kontrol bilgisini göndermektedir. Dolayısıyla çoğa gönderim ortamlarının avantajını kullanarak ağ üzerinde akan mesaj sayısını azaltabilmektedir. KS ve MKDM algoritmaları ise hem teke hem de çoğa gönderim de aynı miktarda kontrol bilgisi kullanmaktadır.

Sıkı bağılı simülasyonlarda KS algoritması MSES algoritmasından teke gönderim ortamlarında daha iyi sonuçlar vermektedir. KS algoritmasının mesajı göndermeden önce eleme yaklaşımı, sıkı bağılı simülasyonlarda çok verimli olmakta ve kontrol bilgisini ciddi oranda düşürmektedir. Bu ortamda yine de en iyi sonuçları yakalayan MKDM algoritmasıdır. Yönetici simülatör üzerinden kontrol bilgisi ile mesaj sıralama yaklaşımı hem kontrol mekanizmasının sadece yönetici simülatöre konmasını hem de bu mekanizmanın çok basit olmasını sağlamıştır. Çoğa gönderim ortamında MSES ve KS birbirine çok yakın sonuçlar vermektedir.

Sadece mesaj alıcılarının değiştirildiği ortamda KS algoritması MSES algoritmasına göre daha ölçeklenebilir bir grafik çizmektedir. MSES ve MKDM algoritmaları alıcı sayısının artması ile doğru orantılı olarak kontrol bilgisi kullanımını arttırmakta ve ağ üzerinde daha fazla gecikmeye sebebiyet vermektedir. KS algoritması ise önce artan sonra azalan bir grafik sergilemekte ve simülasyondaki simülatörlerin arasındaki iletişim seviyesi arttıkça daha az kontrol bilgisi kullanmaya ve ağ üzerinde daha az gecikmeye sebebiyet vermeye başlamaktadır. Ancak KS algoritmasının mesajı göndermeden önce kontrol bilgisi üzerinde eleme yapması onu diğer algoritmalarından daha yavaş bir algoritma yapmaktadır. Dolayısıyla ağ üzerine çıkan kontrol bilgisini MSES'ten daha küçültebilmesine karşılık, MSES'ten daha yavaş çalışmakta ve simülatör üzerine düşen işlem yükü daha büyük olmaktadır.

Algoritmalar son olarak farklı boyutlarda hazırlanmış karışık simülasyonlarda test edilmiştir. MSES ve KS algoritmalarının simülasyonun hem simülatör sayısı hem de üzerinde akan mesaj sayısı ile büyüdüğü durumlarda kararlı olmadığını, parabolik olarak kontrol bilgisi ve işlem gücü kullandıklarını göstermiştir. Bu senaryoda simülatör sayısı ve mesaj sayısı artışına birlikte doğrusal olarak tepki veren MKDM algoritması en iyi sonuçları yakalamıştır. MKDM algoritması sadece mesaj alıcı sayısına göre kontrol bilgisi oluşturmaktadır. Dolayısıyla simülasyonun büyümesi, toplam simülatör sayısının artması algoritmayı etkilememekte, çıkan sonuçlarda sadece mesaj alıcı sayısındaki artış doğrusal olarak yansımaktadır.

Sonuç olarak MKDM algoritması farklı simülasyon modellerinde kullanılan kontrol bilgisi, simülatörler üzerinde oluşan işlem yükü ve oluşturduğu ağ gecikmesi bakımından en iyi sonuçları veren algoritmadır. Algoritmanın ağ üzerinde kullandığı

kontrol bilgisi büyüklüğünün sadece mesaj alıcı sayısına bağlı olması onu simülasyonların büyümesine karşı ise ölçeklenebilir olan tek algoritma yapmaktadır. Gerçeklenmesi son derece basit olan yaklaşım günümüz simülasyon altyapı yazılımları üzerine bir arakatman servisi yada doğrudan altyapıların içine mesaj sıralama servisi olarak gerçekleştirilebilir. Elde edilen deneysel sonuçlar tez kapsamında önerilen MKDM algoritmasının diğer algoritmalara göre çok daha iyi bir alternatif çözüm olabileceğini göstermiştir.

6. ÖNERİLER

Bu tez kapsamında, iki en uygun nedensel sıralı mesajlaşma algoritması ile yine tez kapsamında geliştirilen mesaj sıralama algoritması dağıtık bir sistemi benzeten yazılım üzerinde gerçekleştirilmiş ve farklı kıstaslar altında incelenmiştir. Yapılan kapsamlı teorik ve deneysel analizler çalışmaya eklenmiştir. Bu çalışma baz alınarak ileride yapılabilecek araştırma önerileri şu şekilde sıralanabilir:

- Algoritmaların üzerinde koşturulduğu DOS ortamının, tüm dünyada kabul gören ortam simülatörleri (örn: OMNET++, OPNET++) ile değiştirilerek daha gerçekçi simülasyon ortamları oluşturulması
- Ağ üzerinde oluşan gecikmelerin özellikle hangi bağlantılardan kaynaklandığının tespit edilmesi
- Algoritmalar farklı kütüphaneler altında gerçekleştirilerek mesaj işleme hızlarındaki değişimin ölçülmesi
- Yönlendiricilere mesajları depolama kabiliyeti eklenerek daha gerçekçi mesaj gecikme modelleri oluşturulması
- Farklı mesaj sıralama algoritmalarının problemin çözümündeki etkinliklerinin denenmesi
- Performans kriterlerinden biri olarak algoritmaların mesajları ne kadar beklettiğinin ölçülmesi
- Yönlendiricilerin yerleştirileceği ve MKDM algoritmasındaki sunucunun konulabileceği nokta seçimini optimize edebilen bir yapının eklenmesi

7. KAYNAKLAR DİZİNİ

- [1] A.D. Kshemkalyani and M. Singhal, “*Necessary and Sufficient Conditions on Information for Casual Message Ordering and Their Optimal Implementation*”, Distributed Computing, 11, 2, pp. 91-111, 1998
- [2] W. Cai, S.J. Turner, B. Lee, and J. Zhou, “An Alternative Time Management Mechanism for Distributed Simulations”, ACM Transactions on Modeling and Computer Simulation, Vol. 15, No.2, 109-137, April 2005
- [3] Fujimoto, R. M., Weatherly, R. M., “*Time Management in the DoD High Level Architecture*”, IEEE 10th Workshop on Parallel and Distributed Simulation, 60-67, 1996
- [4] L. Lamport, “*Time, Clocks, and the Ordering of Events in a Distributed System*”, Communications of the ACM, 21, no.7, (July): 558-565, 1978
- [5] P. Chandra, P. Gambhire, A.D. Kshemkalyani, “*Performance of the Optimal Casual Multicast Algorithm: A Statistical Analysis*”, IEEE Transactions on Parallel and Distributed Systems, Vol. 15, No. 1, 2004
- [6] A. Schiper, J. Eggli, A. Sandoz, “*A new algorithm to implement causal ordering*”, In Proceedings of the 1989 Workshop on Distributed Algorithms LNCS 392, 219–232, 1989
- [7] M. Raynal, A. Schiper, S. Toueg, “*The causal ordering abstraction and a simple way to implement it*”, Information Processing Letters, volume 39, issue 6, September, 343–350, 1991
- [8] K. Birman, A. Schiper, P. Stephenson, “*Lightweight causal and atomic group multicast*”, ACM Transactions on Computing. Systems (TOCS), 9 no.3, (August): 272–314, 1991
- [9] R. Prakash, , M. Raynal, , M. Singhal, “*An adaptive causal ordering algorithm suited to mobile computing environments*”, Journal of Parallel and Distributed Computing, 41, no.2, (March): 190–204, 1997
- [10] K. Birman, T. Joseph, “*Reliable Communication in Presence of Failures*”, ACM Trans. on Computer Systems, 5(1), 47-76, 1987
- [11] “*High Level Architecture Run Time Infrastructure RTI 1.3-Next Generation Programmer’s Guild v5*”, Department of Defense, Defense Modeling and Simulation Office, 1999

8. ÖZGEÇMİŞ

Tolga Baştürk, 1980 yılında Çanakkale’de doğdu. İlkokul öğrenimini Gazi İlk İkokulu’nda, Orta ve Lise öğrenimini Çanakkale Milli Piyango Anadolu Lisesi’nde tamamladı. 1999 yılında Marmara Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği bölümüne girmeye hak kazandı. Buradaki öğrenimini 2004 yılında tamamladı. 2005 yılında Gebze Yüksek Teknoloji Enstitüsü’nde Bilgisayar Mühendisliği Anabilim Dalında yüksek lisans öğrenimine başladı. 2005 yılından beri TÜBİTAK, Marmara Araştırma Merkezi, Bilişim Teknolojileri Enstitüsü’nde araştırmacı olarak çalışmaktadır. Çalışma alanları simülasyon teknolojileri, simülasyon altyapıları ve dağıtık sistemlerde sıralı mesajlaşma yaklaşımlarıdır.

