
GAZİ ÜNİVERSİTESİ
Gazi
1926

AVİYONİKTE KULLANILAN YÜKSEK HIZLI AĞLARIN KARŞILAŞTIRILMASI

Osman Raşit KÜLTÜR

YÜKSEK LİSANS TEZİ
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANA BİLİM DALI

GAZİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

KASIM 2021

ETİK BEYAN

Gazi Üniversitesi Fen Bilimleri Enstitüsü Tez Yazım Kurallarına uygun olarak hazırladığım bu tez çalışmasında;

- Tez içinde sunduğum verileri, bilgileri ve dokümanları akademik ve etik kurallar çerçevesinde elde ettiğimi,
- Tüm bilgi, belge, değerlendirme ve sonuçları bilimsel etik ve ahlak kurallarına uygun olarak sunduğumu,
- Tez çalışmasında yararlandığım eserlerin tümüne uygun atıfta bulunarak kaynak gösterdiğimi,
- Kullanılan verilerde herhangi bir değişiklik yapmadığımı,
- Bu tezde sunduğum çalışmanın özgün olduğunu,

bildirir, aksi bir durumda aleyhime doğabilecek tüm hak kayıplarını kabullendiğimi beyan ederim.

Osman Raşit KÜLTÜR

05/11/2021

AVİYONİKTE KULLANILAN YÜKSEK HIZLI AĞLARIN KARŞILAŞTIRILMASI

(Yüksek Lisans Tezi)

Osman Raşit KÜLTÜR

GAZİ ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

Kasım 2021

ÖZET

Uçaklar ve helikopterler, gövdeleri boyunca dağılmış birçok sensör ve uç sistemlere sahiptir. Sürekli artan sensör yükü (radar, elektronik harp, radyo, kamera vb.) nedeniyle, uçakların daha fazla bant genişliğine ve daha yüksek bank genişlikli iletişim omurgalarına ihtiyaçları vardır. Bu uç sistemlerin arasındaki iletişimde kullanılan ARINC-429 ve MIL-STD-1553 gibi eski protokoller, günümüz uçaklarının artan bant genişliği gereksinimlerini karşılamada yetersiz kalmaktadır. Buna rağmen, bu eski ağlar, aviyonik sistemlerin gerektirdiği, güvenilirlik ve yüksek hata toleransı gibi özelliklere sahip olduklarından hâlen kullanılmaktadırlar. Bu nedenle, bu eski ağlardaki gibi yüksek hizmet kalitelerine sahip olan ve kritik sistemlere uygunlukları bulunan yüksek bant genişliğine sahip farklı teknolojiler geliştirilmiştir. Airbus patentli ARINC-664 Bölüm 7'nin özel bir uygulaması olan Aviyonik Tam Çift Yönlü Anahtarlamalı Ethernet (Avionics Full-Duplex Switched Ethernet, AFDX) protokolü son yıllarda gündeme gelmiştir. Bir diğer öne çıkan çözüm, TTTech tarafından patenti alınan Zaman Tetiklemeli Ethernet (Time Triggered Ethernet, TTEthernet) protokolüdür. Bu çalışma, bu iki protokolü aviyonik perspektifinden OMNET++ simülasyon ortamında modelleyerek simüle etmekte ve karşılaştırmaktadır. Simülasyon sonuçları, yeni nesil uçaklarda TTEthernet ağlarının kullanılmasının AFDX'e nazaran daha yüksek iletişim kalitesi sağlayacağını göstermiştir.

Bilim Kodu : 90523

Anahtar Kelimeler : Aviyonik, OMNET++, ARINC-429, AFDX, TTEthernet

Sayfa Adedi : 67

Tez Danışmanı : Prof. Dr. Hasan Şakir BİLGE

COMPARISON OF HIGH SPEED NETWORKS IN AVIONICS

(M. Sc. Thesis)

Osman Raşit KÜLTÜR

GAZİ UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

November 2021

ABSTRACT

Airplanes and helicopters have many sensors and terminal systems scattered throughout their fuselage. Aircrafts need more bandwidth and higher broadband communication backbones due to the ever-increasing sensor load (radar, electronic warfare, radio, camera, etc.). Legacy protocols such as ARINC-429 and MIL-STD-1553 used in the communication of these edge systems are insufficient to meet the increasing bandwidth requirements of today's aircrafts. Despite this, these legacy networks are still in use because they have the features required by avionic systems, such as reliability and high fault tolerance. For this reason, different technologies with high bandwidth have been developed, which have high quality of service and are suitable for critical systems, as in these old networks. Avionics Full-Duplex Switched Ethernet (AFDX) protocol, which is a special implementation of Airbus patented ARINC-664 Part 7, has come to the fore in recent years. Another prominent solution is the Time Triggered Ethernet (TTEthernet) protocol patented by TTTech. This study simulates and compares these two protocols from the perspective of avionics by modeling them in the OMNET++ simulation environment. The simulation results have shown that TTEthernet networks will be more efficient than AFDX in new generation aircrafts.

Science Code : 90523

Key Words : Avionics, OMNET++, ARINC-429, AFDX, TTEthernet

Page Number : 67

Supervisor : Prof. Dr. Hasan Şakir BİLGE

TEŞEKKÜR

Tez çalışmalarım sırasında bana her daim vakit ayıran, akademik ve teknik açıdan yol gösteren danışmanım Prof. Dr. Hasan Şakir BİLGE hocama saygılarımı ve şükranlarımı sunarım. Ayrıca, destekleyici ve cesaretlendirici yönlendirmelerinden dolayı ASELSAN'daki takım arkadaşlarıma ve yöneticilerime teşekkür ederim.

İÇİNDEKİLER

Sayfa

Sayfa

ÇİZELGELERİN LİSTESİ

Çizelge **Sayfa**

ŞEKİLLERİN LİSTESİ

Şekil **Sayfa**

Şekil **Sayfa**

SİMGELER VE KISALTMALAR

Bu çalışmada kullanılmış kısaltmalar, açıklamaları ile birlikte aşağıda sunulmuştur.

Kısaltmalar	Açıklamalar
AFDX	Aviyonik Tam Çift Yönlü Anahtarlamalı Ethernet
ARINC	Birleşik Havacılık Radyosu (Aeronautical Radio Incorporated)
BAG	Bant Genişliği Tahsis Boşluğu (Band Allocation Gap)
BE	En İyi Çaba (Best Effort)
Gbps	Saniyedeki Gigabit (Gigabit per seconds)
Jitter	Seğirme
Kbps	Saniyedeki Kilobit (Kilobit per seconds)
Latency	İletim Gecikmesi
Mbps	Saniyedeki Megabit (Megabit per seconds)
MIL-STD-1553	Askeri Standart (Military Standard) 1553
NED	Ağ Tanımlama (Network Description)
RC	Bant Sınırlı (Rate Constraint)
TSN	Zaman Duyarlı Ağ (Time Sensetive Networking)
TT	Zaman Tetiklemeli (Time Triggered)
TTP	Zaman Tetiklemeli Protokol (Time Triggered Protocol)
TTEthernet	Zaman Tetiklemeli Ethernet
VL	Sanal Bağlantı (Virtual Link)

1. GİRİŞ

Aviyonik (avionics) kelimesi, havacılık (aviation) ve elektronik (electronics) kelimelerinin birleşimi ile oluşan, uçakların, helikopterlerin, uyduların ve uzay araçlarının her türlü elektronik aksamı olarak tanımlanmaktadır. 18. yüzyılın başlarında geliştirilmeye başlanan aviyonik teknolojilerinin gelişimi 1940'larda 2. Dünya Savaşının da etkisiyle çok daha olgun seviyelere gelmiştir. Özel kablolarla birbirine bağlanan seyrüsefer, RADAR, iletişim ekipmanları ve kokpit ekranları gibi birkaç ayrı analog sistem de kullanılarak geliştirilmeye devam eden uçaklar gitgide daha kompleks hale gelmeye başlamıştır. Bu sistemlerde bağlantılar hem noktadan noktaya olduğundan hem de analog haberleşme yapıldığından yeni bir aviyonik birimin uçağa eklenmesinde kullanılan kablolar sebebiyle uçağın hem maliyeti hem de ağırlığı artmaktadır. Uçaklardaki analog ekipmanları daha kullanışlı hale getiren sayısal teknolojiler sayesinde bu ekipmanları birbirleriyle haberleşecek şekilde sayısal iletişim teknolojileri tasarlanmaya başlanmıştır. ARINC-429 ve MIL-STD-1553 sayısal iletişim protokolleri, bütün aviyonik ekipmanlarını analog iletişimden sayısal iletişime geçirerek uçaklardaki iletişim kalitesini arttırmıştır.

Uçak içindeki aviyonik birimlerin kablolu sayısal haberleşmelerinde öne çıkan iki adet kritik unsur vardır. Bunlar, verinin belirli ve bilinen bir zamanda bir birimden diğer birime aktarılması ile verinin yıldırım, soğuk/sıcak hava koşullarından etkilenmeyip bozulmadan aktarılmasıdır. ARINC-429 ve MIL-STD-1553, bu iki ana unsuru karşılayacak yapılara sahiptir. Bu yüzden, 1980'lerden bugüne hâlen günümüz uçaklarında kullanılmaktadırlar. Ancak, sık kullanılan bu iki aviyonik ağ protokolü yüksek hızlarda veri taşıyamadıkları için yeni nesil uçaklardaki sistem ihtiyaçlarını karşılayamamaktadır. 2000'li yıllara doğru, artan aviyonik sistem ihtiyaçları sebebiyle ARINC-429 ve MIL-STD-1553 iletişim protokolleri yeterli gelmemeye başlamış ve yeni çalışmalar sonucu Ethernet teknolojisinin aviyonikte kullanımı ile ilgili çözümler üretilmiştir. Bu kapsamda, günümüzde en çok öne çıkan teknolojiler Aviyonik Tam Çift Yönlü Anahtarlamalı Ethernet (AFDX) ve Zaman Tetiklemeli Ethernet'tir (TTEthernet).

AFDX ve TTEthernet eski nesil aviyonik veri ağlarına göre uçaklardaki haberleşme hizmet kalitesini önemli bir ölçüde arttırmıştır. Bu tez çalışmasında, eski nesil protokoller ve bu iki protokolün farklılıkları karşılaştırılarak yeni nesil uçaklarda TTEthernet'in seçilmesinin

diğer protokollere göre avantajları ele alınmıştır. Karşılaştırma yöntemi olarak ise modelleme ve simülasyon metodu seçilmiştir. AFDX ve TTEthernet haberleşme protokolleri modellenerek bir aviyonik mimarisi özelinde OMNET++ ağ simülasyon programında simüle edilmiştir. Yapılan simülasyonlar ile AFDX ve TTEthernet ağlarının aynı ağ topolojisindeki davranışları, veri iletimi gecikmeleri analizi bakış açısı ile incelenmiştir.

2. LİTERATÜR TARAMASI

Bu bölümde bu tezin kapsamında olan literatürdeki araştırmalara yer verilmiştir. 2000'li yıllardan itibaren geliştirilen AFDX ve TTEthernet protokollerinin aviyonik ağlarındaki davranışları akademik ve endüstriyel araştırmalara konu olmuştur. Örneğin NASA'nın yayınladığı bir teknik raporda uzay araçlarındaki aviyonik sistemlerde kullanılmak üzere MIL-STD-1553, TTP, FlexRay, Ethernet 802.3 ve AFDX gibi aviyonik ağlarının teknik özellikleri özetlenmiş ve tablo halinde karşılaştırılmıştır [1].

Dan Gunnarsson'un yaptığı tez çalışmasında FlexRay, Zaman Tetiklemeli İletişim Protokolü, AFDX ve MIL-STD-1553 protokolleri teorik olarak karşılaştırılmakta ve teorik bir TTP aviyonik ağı topolojisini donanım seviyesinde simüle ederek TTP ağının, diğer ağlar gibi aviyonik sistemlere uygun olup olmadığını değerlendirmektedir [2].

Nour el-din Safwat'ın yaptığı tez çalışmasında ARINC-429 ve AFDX ağı teorik olarak karşılaştırılmıştır [3]. Ayrıca OPNET [4] Simülasyon ortamı kullanılarak AFDX ve ARINC-429 modellenmiş ve simüle edilmiştir. Bu çalışma, AFDX'in ARINC-429'a olan üstünlüklerini öne çıkarmaktadır. Simülasyonda 7 adet uç birim ve 3 adet ağ anahtarına sahip bir ağ modeli kullanılmıştır. Aynı ağ modeli Henri Bauer ve ark. yaptıkları akademik çalışmada AFDX ağındaki veri trafiği gecikmelerinin saptanmasında da kullanılmıştır [5]. Yapılan çalışmadan teorik Ağ Hesabı (Network Calculus) yöntemleri kullanılarak AFDX protokolünün sunduğu trafik gecikmeleri doğrulanmıştır. Ağ hesabı yöntemlerini kullanarak yapılan başka bir çalışmada ise UPPAL [6] ağ modelleme programı tabanını kullanan bir yazılım geliştirilerek, bu yazılımın AFDX ağlarında uçtan uca gecikme sürelerinin hesabında kullanılabileceği gösterilmiştir [7]. Ağ hesabı yöntemlerine dayalı olarak yapılan bir çalışmada, Ling Guo ve ark. TTEthernet ağlarındaki uçtan uca veri iletim gecikmelerini incelemek için yeni bir yöntem geliştirilmiştir [8]. Bu yöntemde geliştirilen algoritma Java dilinde kodlanıp TTEthernet ağlarının hız kısıtlı trafik iletimlerindeki veri gecikmeleri analiz edilmiştir.

Owais Hamid ve Anas Vaqar'ın yaptıkları çalışmada TTEthernet ve AFDX ağları teorik olarak karşılaştırılmıştır [9]. Yapılan çalışmada TTEthernet'in AFDX'e olan üstünlükleri

vurgulanmış, her iki protokolün birbirlerine göre olan avantajları ve dezavantajlarına değinilmiştir.

Quinghua Yang ve ark. yaptıkları çalışmada OMNET++ [10] simülasyon programı kullanılarak, [8]'deki çalışmaya benzer şekilde, TTEthernet ağının hız kısıtlı trafik iletimi kullanılarak AFDX ağı elde edilmiştir [11]. Kurulan ağ topolojisinde AFDX ağı simüle edilerek ağdaki trafik gecikmeleri karşılaştırılmıştır.

Lin Zhao ve ark. yaptıkları çalışmada TTEthernet ile TSN protokollerini teorik olarak karşılaştırmaktadır, iletilen trafiklerdeki gecikmelerin zamansal duyarlılık açısından irdelemektedir ve aviyonik uygulamalarda TSN protokolünün TTEthernet'e göre daha uygun olabileceğini göstermektedir [12].

Nejla Rejeb ve ark. yaptıkları çalışmada [3] ve [5]'deki aynı ağ modelini baz alınarak OMNET++ simülasyon programı ve CORE4INET [13] kütüphanesi kullanılarak AFDX ağı simüle edilmiştir ve simülasyon sonuçları ağ hesabı yöntemlerinin verdiği sonuçlarla karşılaştırılarak AFDX protokolünün sunduğu özellikler doğrulanmıştır [14].

Bu tez çalışmasının literatüre olan en önemli katkısı OMNET++ programını kullanarak hem AFDX'i hem de TTEthernet'i aynı ağ topoloji üzerinde simüle etmesi ve uçtan uça veri iletimi analizi yaparak bu iki protokolü karşılaştırmasıdır. Bu tez çalışmasında AFDX'in ve TTEthernet'in hem teorik olarak hem de deneysel olarak karşılaştırılması ve birbirlerine olan üstünlükleri anlatılmıştır. Bu tezde, literatürde sık kullanıldığı için baz alınan ağ topolojisi, [3], [5] ve [14]'deki 7 adet uç birim ve 3 adet ağ anahtarından oluşan topolojinin aynısıdır. Geçmişteki ve gelecekteki akademik çalışmalardan ayrı düşmemek adına, bu tez çalışmasında literatürde sık kullanılan ağ topolojisi üzerinde simülasyon yapılması tercih edilmiştir. Ayrıca, Temmuz 2021'de bu tez çalışmasının çıktısı olarak 3 adet uç birim ve 1 adet ağ anahtarına sahip olan topolojideki AFDX ve TTEthernet simülasyonu sonuçları özelinde bu iki protokolü karşılaştıran bir çalışma uluslararası bir konferansta sunulmuştur [15].

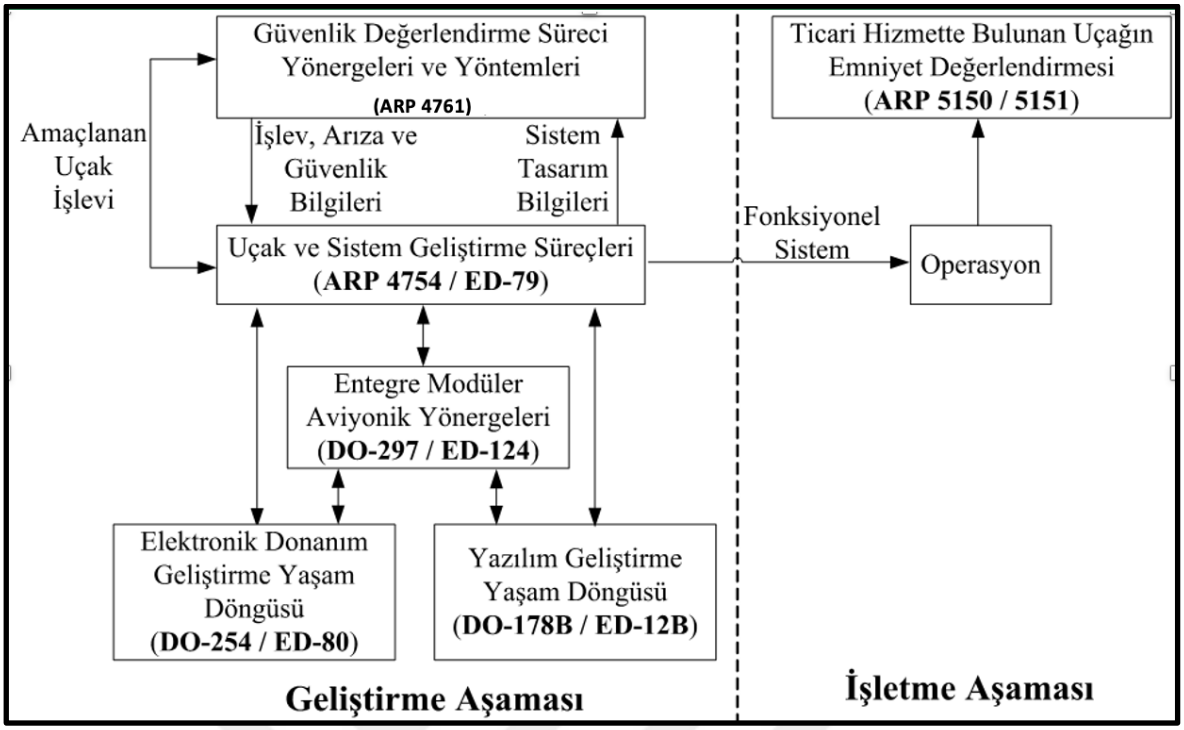
3. AVİYONİK SİSTEMLER

3.1. Aviyonik Sistemlerdeki Kılavuzlar ve Düzenlemeler

Aviyonik endüstrisi, uçakların geliştirilmesi ve tasarımı ile ilgili katı kurallar ve düzenlemeler konusunda uzun bir geçmişe sahiptir. Son yirmi yılda, aviyonik sistemlerinin artan karmaşıklığı ve entegrasyonu, güvenlikli ve düzgün çalışmayı garanti edebilmek için tasarım faaliyetlerinde bu konularda sertifikasyonlara olan ihtiyaç artmıştır.

Sertifikasyon yoluyla güvence, genellikle tasarım ve tasarım sürecinin test edilmesi, analizi ve gözden geçirilmesinin bir kombinasyonu ile sağlanmaktadır. Sivil hava taşıtları için nihai sertifikalandırmayı yapan otorite, Amerika Birleşik Devletleri'nde FAA ve Avrupa'da EASA'dır. Bu makamlar tarafından belirtilen kurallar ve düzenlemeler hemen hemen aynıdır. Ancak, değerlendirme ikisi arasında değişebilir. Bir uçağın sertifikasyondan geçmesi ve uçmasına izin verilmesi için nasıl çalışması gerektiğine dair kurallar ve düzenlemeler FAA ve EASA tarafından onaylanan regülasyon dokümanlarında mevcuttur.

Şekil 3.1'de gösterildiği üzere, sertifikasyona yardımcı olmak için, tatmin edici bir güvenilirlik düzeyi sağlamak adına donanım ve yazılımın tasarımının, uygulanmasının ve kullanımının nasıl yapılması gerektiğini belirten kılavuz belgeler vardır. Bu yönergeler uluslararası standartlardır ve uçak üretmek için uyulması gerekmektedir. Kılavuzlar, işlenen sistemlerin her bir parçasının belirli koşullar altında nasıl çalışması ve davranması gerektiğine dair gereksinimleri ayrıntılı olarak içermektedir.



Şekil 3.1. Geliştirme ve işletme aşamalarını kapsayan kılavuz belgeler [16]

Aviyonik sistemlerde Seviye A'dan Seviye E'ye kadar beş sistem geliştirme güvence düzeyi vardır: felaket, tehlikeli/çok büyük, büyük, küçük ve etkisiz. Çizelge 3.1, sistem tasarımı güvence seviyelerini, bu beş arıza durumu seviyesi ile ilişkilendirir ve ilgili tasarım güvence seviyelerini göstermektedir. Bu tezde tartışılan sistemler, güvenlik seviyesi A ve B olan kritik sistemler olup, bu tezin odak noktası, bu sistemler içindeki iletişimdir ve bu konu ile ilgili sertifikasyon yönergeleri şunlardır:

1. ARP4754 [16], genel hava aracı işletme gereksinimlerini tanımlar. Üretilecek hava araçları için işlevsel gereksinimleri de tanımlayarak, hava aracına kurulu yüksek düzeyde entegre veya karmaşık sistemlerin sertifikasyon sürecini anlatmaktadır. Yüksek düzeyde entegre sistemler, çok sayıda işlevlerin bir arada gerçekleştirildiği sistemleri ifade etmektedir. Karmaşık sistemler ise güvenlik seviyesi yalnızca test ile kanıtlanamayan ve analiz/simülasyon araçları olmadan işlevlerinin anlaşılması zor olan sistemleri tanımlamaktadır. ARP4754, uçak fonksiyonlarını uygulayan sistemlerin toplam yaşam döngüsünü ile ilgilenmektedir. Bu kılavuz, sistem, yazılım ve donanım tasarım süreçleri hakkında bilgi vermemektedir.
2. ARP4761 [17], hava araçlarında sistem gereksinimlerin oluşturulmasını ve doğrulanmasını içeren güvenlik değerlendirme sürecini açıklar. ARP4761, hava

araçlarında kullanılan fonksiyonları analiz etmede ve ilgili fonksiyonların muhtemel tehlikelerini saptamada kullanılan bir kılavuzdur.

3. RTCA/DO-178B [18] hava araçlarının yazılım yaşam döngülerini konu almaktadır. Yazılım tasarım sürecinin nasıl yönetileceğini ve yazılım çıktılarının sistem gereksinimlerini karşıladığının nasıl kanıtlanacağı ile ilgili bir kılavuzdur.
4. RTCA/DO-254 [19], güvenlik açısından kritik uygulamalar için karmaşık donanım geliştirmeye yönelik donanım tasarım güvencesi konusunda bir kılavuz belgesidir. Çizelge 3.1'de verilen sistem güvencelerine uygun bir şekilde donanım tasarım sürecinden bahseder.
5. RTCA/DO-160D [20], Çevresel gereklilikler için Çevre Koşulları ve Havadaki Ekipman için Test Prosedürleri kılavuz belgesidir. Çizelge 3.1'de verilen güvence seviyelerini karşılayabilmek için, donanımların geçmesi gereken testlerden bahseder.

Çizelge 3.1. Aviyonik sistemlerdeki güvence seviyeleri [16]

Sistem Güvence Seviyesi	Arıza Durumu Sınıflandırması	Arıza Durumu Açıklaması
A	Felaket	Devam eden güvenli uçuş ve inişi önleyecek arıza koşulları.
B	Tehlikeli / Çok Büyük	Uçağın fonksiyonlarını ve/veya uçuş ekibinin olumsuz çalışma koşullarıyla baş edebilme durumunu azaltacak arıza koşulları şunlardır: güvenlik hata paylarında veya işlevsel yeteneklerde yükse düzeyde bir düşüş, fiziksel tehlike veya daha yüksek iş yükü. Uçuş ekibinin görevlerini doğru veya tam olarak yerine getireceğine güvenilememesi veya bu yolcuların az bir kısmının ciddi veya potansiyel olarak ölümcül yaralanmalar dâhil olmak üzere yolcular üzerindeki olumsuz etkileri de B seviyesinde değerlendirilmektedir.
C	Büyük	Hava aracının kabiliyetini veya mürettebatın olumsuz çalışma koşullarıyla baş etme kabiliyetini, örneğin emniyet hata payında veya işlevsel yeteneklerde önemli bir düşüş, mürettebat iş yükünde veya mürettebatın verimliliğini bozan veya muhtemelen yaralanmalar da dâhil olmak üzere binada bulunanlara rahatsızlık veren koşullarda.
D	Küçük	Hava araçlarının güvenliğini önemli ölçüde düşürmeyecek ve mürettebat eylemlerinden kaynaklanan arıza koşullarını içerir. Küçük arıza koşulları, örneğin, güvenlik hata payında veya işlevsel yeteneklerde hafif bir azalmayı, rutin uçuş planı değişiklikleri gibi mürettebat iş yükünde hafif bir artışı veya yolcular için bazı rahatsızlıkları içermektedir.
E	Etkisiz	Hava araçlarının operasyonel ve fonksiyonel yeteneklerini etkilemeyen veya mürettebat iş yükünü artırmayan arıza durumları, etkisiz seviye olan E seviyesinde değerlendirilir.

3.2. Aviyonik Sistem Gereksinimleri

Bu bölümde, bir uçakta kullanılacak sistem ve bileşenlere ilişkin gereksinimlerden bahsedilmektedir. Bu gereksinimler birçok aviyonik veri ağları için geçerlidir. Ancak bazı gereksinimler sadece spesifik alanlarda geçerli olabilmektedir. Havacılık sektöründe geliştirilen sistemlerin, Çizelge 3.1'de verilen hedef sistem güvence seviyesine uygunluğunun sertifikalandırılması, sistemin operasyonel seviyede kullanılabilmesi için şarttır. Bu gerekliliklerin ve sertifikasyon sürecinin amacı, sertifikasyon otoritelerine bir hava aracına yerleştirilen her bir sistemin ve bileşenin birlikte sorunsuz çalışabildiğini, ilgili hava aracının uçuşa elverişli olduğunu ve uçmasına izin verilebileceğini kanıtlamaktır.

3.2.1. İşlevsel gereksinimler

İşlevsel gereksinimler, belirtilen koşullar altında bir aviyonik sistemin istenen performansını yakalayabilmek için gerekli olan gereksinimlerdir [16]. Bu gereksinimler şu hususları konu almaktadır: düzenleyici kısıtlamalar, müşteri istekleri, operasyonel kısıtlamalar ve uygulama senaryoları. Bu gereksinimler, ilgili aviyonik sistemin tüm önemli yönlerini incelemekte ve tanımlamaktadır.

Çizelge 3.2. Aviyonik sistemlerdeki işlevsel gereksinimler [2]

İşlevsel Gereksinimler	Açıklama
Operasyonel Gereksinimler	Uçuş ekibi ile her bir fonksiyonel sistem arasındaki ara yüzü tanımlar.
Ara yüz Gereksinimleri	Fiziksel sistem ve alt sistemler arasındaki bağlantıları ile veri yollarını içerir.
Fiziksel Gereksinimler	Boyut, güç, soğutma ve kullanım gibi özellikleri tanımlar.
Bakım Gereksinimleri	Ekipmanların periyodik ve plansız olarak yapılan bakımlarından bahseder.
Performans Gereksinimleri	Aviyonik veri ağındaki bant genişliği, düğüm sayısı, hız, gecikmeler ve yanıt süresi gibi faktörleri tanımlar.

Bu tez çalışması, aviyonik ağlarının performansı üzerine odaklandığı için birçok işlevsel gereksinim göz önünde bulundurulmamıştır.

3.2.2. Çevresel gereksinimler

Çevresel gereklilikler, spesifikasyonda aksi belirtilmedikçe, ekipmanın ve hava aracının hizmet ömrü boyunca yaşadığı çevresel koşullar altında normal ve bozulma olmadan çalışacağını belirtmektedir.

Çizelge 3.3. Aviyonik sistemlerdeki çevresel gereksinimler [2]

Çevresel Gereksinimler	Açıklama
Sıcaklık Gereksinimleri	Hem yerde hem de havadayken normal ve ağır koşulları içerir.
Jitter Gereksinimleri	Tüm ekipman tarafından ne tür bir jitter'ın ele alınması gerektiğini belirtir.
Elektromanyetik Uyumluluk (EMC) Gereksinimleri	Yıldırım etkileri altında sistemin davranışı ile ilgilenir. Doğrudan ve dolaylı olarak sisteme, yüksek yoğunluklu yayılan alanlar, geçici yüksek gerilimli dürtüler ve yavaş dalga formları verilerek testler yapılır. Bu gerekesimler belirlenirken RTCA/DO-160D kılavuzuna uyulur.
Bakım Gereksinimleri	Ekipmanların periyodik ve plansız olarak yapılan bakımlarından bahseder.
Ara yüz Gereksinimleri	Fiziksel sistem ve alt sistemler arasındaki bağlantıları ile veri yollarını içerir.

Güvenlik açısından kritik bir iletişim sistemi tasarlarken çevresel gereksinimler büyük önem taşımaktadır. Bir uçağın kanatlarına ve kuyruğuna kadar uzanan bir veri yolu veya iletişim bağlantısı, performansın yanı sıra güvenilirliği de azaltabilecek önemli elektromanyetik alanlara, sıcaklıklara ve jitterlere maruz kalmaktadır. Bu yüzden uçakların RTCA/DO-160D uyumluluğuna sahip olması uçağın aşırı koşullarda da çalışabileceğini göstermede şarttır.

3.2.3. Test gereksinimleri

Aviyonik sistemlerin, amaçlanan işlevlerini yerine getirdiğini göstermek için testler yapılmaktadır. Amaçlanan bir işlevin test edilmesi, sistem gereksinimlerine dayalı olarak geçti/kaldı kriterlerine göre değerlendirmeyi içermektedir. Her test için gerekli girdiler, gerekli eylemler ve beklenen sonuçlar toleranslı olarak belirtilmelidir. İlgili testler, bir aviyonik sistemin geliştirme aşamasında büyük önem taşır; ancak, sisteme her güç verildiğinde sistemin gerçekleştirdiği kendi kendine testler de önem arz etmektedir. Sonuç olarak, sistemdeki test kabiliyetlerinin çok olması sistemin güvenliğini arttıran bir durum olduğundan test gereksinimlerinin ayrıntılı olması faydalıdır.

3.2.4. Emniyet gereksinimleri

Emniyet gerekliliği, emniyet ve güvenilirlik açısından önemli olarak tanımlanabilecek tüm beklenmedik olayların ele alınması gerektiğine ilişkin talepleri içermektedir. Aviyonik sistemin her bir işlevi analiz edilmekte ve kritikliğine bağlı olarak uçuş saati başına maksimum hata oranı verilmektedir. Bu şekilde sayısal verilerden bahseden gereksinimlere nicel güvenlik gereksinimleri denir. Öte yandan, nitel gereksinimler ise tek nokta arızalarının felaket veya tehlikeli arıza koşullarına yol açmamasına yönelik gereksinimlerdir.

Aviyonik endüstrisinde yeni bir teknik ortaya çıktığında, tekniğin güvenilirliğinin kanıtlanması için bir yedekleme/fazlalık mekanizmasının da kullanılmasına ihtiyaç duyulmaktadır. Bu, zaten kanıtlanmış bir teknik kullanıldığında da sıklıkla görülen bir durumdur. Sık kullanılan bazı aviyonik veri protokolleri de emniyeti arttırmak için yedeklenebilirlik/fazlalık sistemini kullanırlar. Böylece, ağdaki bir hata durumunda diğer yedek/fazla hat kullanılarak sistemin emniyeti sağlanmış olmaktadır.

Bir aviyonik veri yolu söz konusu olduğunda belirgin olan temel bir sorun, iletişim ağındaki birimler birbirine bağlandığında, bir kısa devrenin iletişimi engelleyebilmesi veya veri yoluna bağlı birimleri yok edebilmesidir. Bu nedenle, fiziksel bağımsızlık ve yedekleme/fazlalık yaratmak için çoğaltılan birimleri uçağın farklı bölgelerine yerleştirmek emniyeti arttırmak için sık kullanılan bir yöntemdir.

3.2.5. Aviyonik veri yolu gereksinimleri

Aviyonik sistemlerde kullanılacak bir veri yolu için önemli olan gereksinimler, [2]'da da belirtildiği üzere, aşağıdaki gibi özetlenebilir:

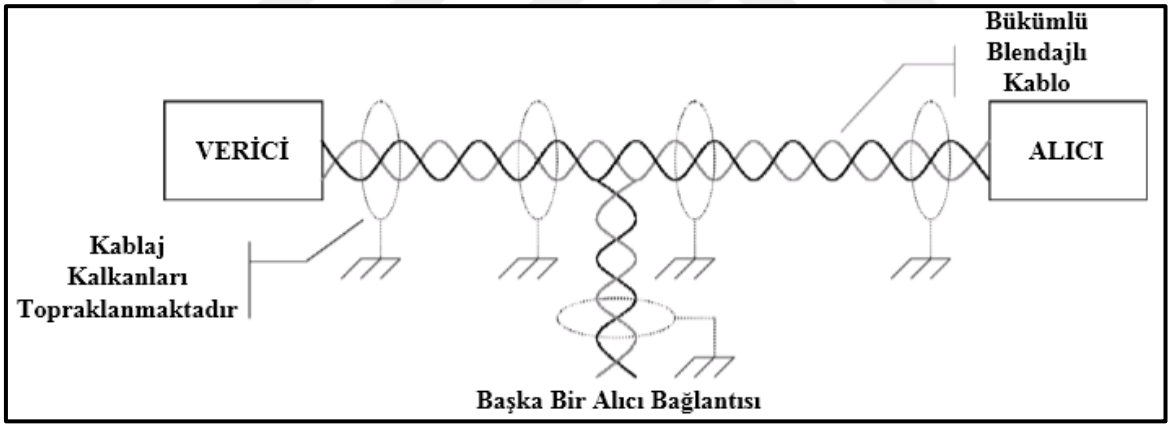
- Veri yolu, sınırlı (öngörülebilir) gecikme (latency) sağlamalıdır.
- Koşturulan uygulama için yeterli bant genişliği veri yolu tarafından sağlanmalıdır.
- Bir aviyonik veri yolunun, gelecekteki yeni muhtemel kullanımlar için en az %50 yedek bant genişliğine sahip olması, o veri yolunun kullanılmasında tercih sebebidir.
- Trafikte kayıp olabilen veya bozulabilen paketlerin bozulma olasılığı, en çok, Çizelge 3.1'de gösterilen, Sistem Güvence Seviyesi D ile tutarlı olmalıdır.
- Bir veri yolunda iletilen saptanamayan bozuk/kayıp veri olasılığı, Çizelge 3.1'de verilen tehlikeli sınıflandırma (Sistem Güvence Seviyesi B) ile tutarlı olmalıdır. Art arda veri bozulması meydana gelme olasılığı, belirtilen ortamın elektromanyetik girişim altında felaket olasılık sınıflandırması (Sistem Güvence Seviyesi A) ile tutarlı olmalıdır.
- Veri yolu standardı, yayın (broadcast) veya çok noktaya yayın (multicast) mesaj seçeneklerini desteklemelidir.
- Veri yolu, veri yoluna kontrollü erişim sağlayan arıza yalıtım yöntemleri sağlamalıdır. Fiziksel veri yolu kontrolcüsünün düşük bir arıza oranına sahip olması da, o veri yolunun kullanılmasında tercih sebebidir.
- Veri yolu, aviyonik sistemlerin operasyonel kalabilmeleri adına, piyasada en az 10 yıl kalabilecek donanımları kullanmalıdır.
- Fiziksel katman, ilgili aviyonik ağından bir sistem kaldırılsa bile veri yolu üzerinde iletişime imkân verebilmelidir.
- Fiziksel katmanın, en az 30 uç sistem kullanabilmesi ve en az 100 metrelik bir veri yolu uzunluğuna sahip olabilmesi tercih sebebidir.
- Fiziksel katmanın, her bir uç sistemde galvanik izolasyon sağlamak için transformatör kuplörlerini ihtiva edebilmesi, elektromanyetik girişime karşı tedbir sağladığından, tercih sebebidir.

4. AVİYONİK'TE KULLANILAN AĞLAR

Bu bölümde, geleneksel aviyonik ağ protokollerinden ARINC 429 [21] ve MIL-STD-1553 [22] ile yeni nesil aviyonik ağ protokollerinden AFDX [23] ve TTEthernet [24] hakkında temel bilgiler verilerek ileriki bölümlerde anlatılacak olan konuları desteklemek amacıyla bir arka plan sunulmaktadır.

4.1. ARINC-429

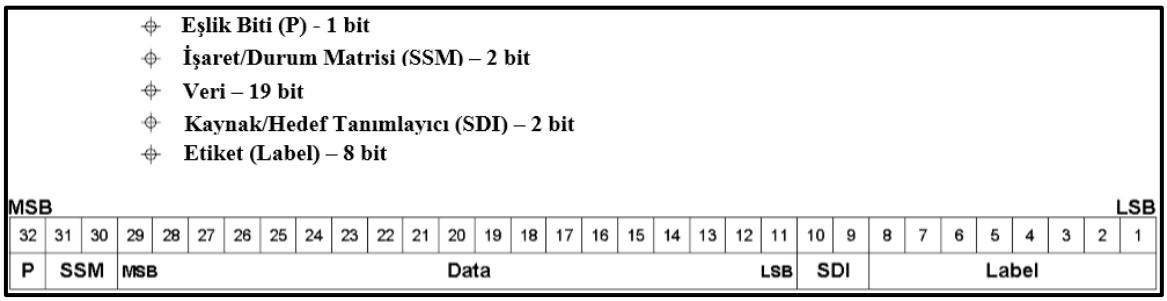
1970'li yıllarda, Amerika'da geliştirilen ARINC 429, havacılıkta en çok kullanılan ağ protokollerinden biridir. ARINC 429 protokolü fiziksel katmanda aviyonik birimleri bağlamak için 78 ohm bükümlü blendajlı kablo kullanır. EMI'a karşı dayanıklı olan diferansiyel sinyal aktarımının yanında, aşağıdaki şekilde olduğu gibi kabloların kalkanları her iki uçta ve her bir bağlantıda topraklanmaktadır.



Şekil 4.1. ARINC-429 ağ yapısı [21]

ARINC-429 fiziksel hat yapısı, bipolar sıfıra dönüşsüz (BNRZ) kodlamalı diferansiyel yapıdır. İki hat arasındaki potansiyel fark gerilimi yüksek, düşük ve boş seviyeler için sırasıyla +10V (7,25 ile 11V arası), -10V (-7,25 ile -11V arası) ve 0 V (-0,5 ile 0,5V arası) şeklindedir. 32 bitlik çerçeve paketindeki her bir bit, saat sinyali kullanmadan senkronizasyona yardımcı olan BNRZ'ye göre sıfır volttan başlayarak farklı bir seviye geçişine sahiptir.

ARINC-429 tek yönlü bir veri yolu olup, verici en az bir, en fazla 20 alıcıya tek yönlü çerçeve akışını başlatır. ARINC-429, 100 kbps olarak yüksek hızda ve 12-14,5 kbps olarak düşük hızda çalışmaktadır.



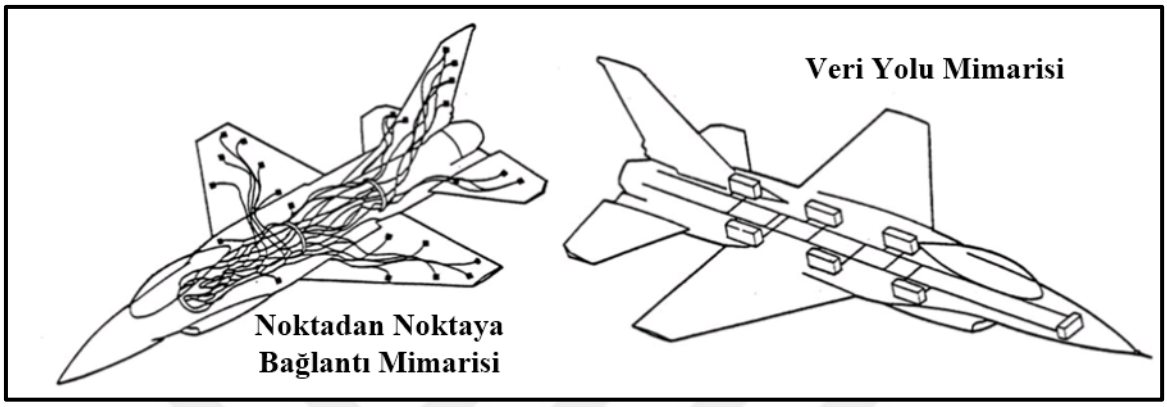
Şekil 4.2. ARINC-429'daki 32 bitlik çerçeve formatı [21]

Şekil 4.2'te verilen 32 bitlik çerçeve formatında 8 bitlik etiket, genellikle veri ve çerçeve formatını tanımlar. Etiket, alıcıları çerçevenin geri kalanı hakkında bilgilendiren ilk iletilen alandır. Çerçevenin bit sırası önce en az anlamlı bittir ancak bu sıra sadece 8. bitin ilk iletildiği etiket için farklılık gösterir. Kaynak/Hedef Tanımlayıcı (SDI), verinin kaynak birimini veya hedef alıcı birimini tanımlamak için kullanılır. Kullanımı zorunlu olmadığı için iletilen mesajların çözünürlüğünü arttırmak için SDI alanına veri bitleri de eklenebilmektedir. 19 bitlik Veri alanı 11. bitten başlar ve 29. bite kadar devam eder. Bu, hassas parametreler için çözünürlüğün yeterince yüksek olmamasına yol açabileceğinden bir dezavantaj olarak düşünülebilir. ARINC-429 standardı mesaj iletmede çok esnek olduğundan, mesajlar ondalık kodlu ikili (BCD), ikinin tümleyen ikili (BNR) ve ayrık veri formatları gibi farklı formatlarda kullanılabilir.

4.2. MIL-STD-1553

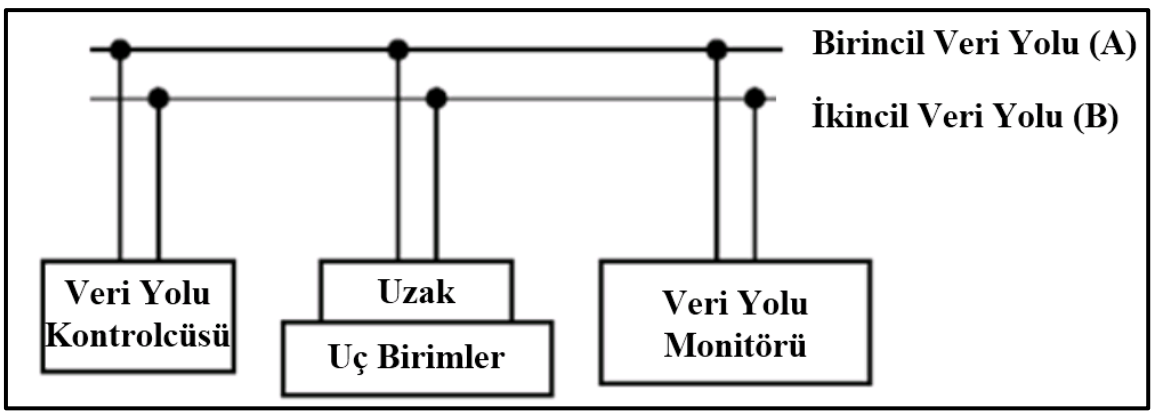
1970'lerin sonunda Amerika'daki F-16 jeti çalışmaları kapsamında MIL-STD-1553 veri yolu geliştirilmiştir. Bu çalışmalara liderlik yapan mühendis Erwin Gangl bu konu hakkında şunları söylemiştir: "Geçmişte tüm sinyallerin uçtan uca kablolarla bağlanıp büyük kablolar ve konektörlerle sonuçlandığı yerlerde, bunun dijital olarak yapılamaması benim için çok sinir bozucu oldu. Bu yüzden devre paylaşımlı dijital multipleks veri yolu konseptine gitmemizi tavsiye ettim. Dijital dönüştürme sensörde yapılır ve bilgisayara çift yedekli blendajlı bükümlü tel çifti aracılığıyla bir dijital sinyal gönderilir. Bu, bilgisayarın bir parçası olan analogtan dijitale dönüştürücü donanıma sahip olmanın yerini aldı ve

ayrıca bilgisayarı aviyoniklerin geri kalanına bağlamak için gereken kablo sayısını azalttı. Konseptin kabul edildiği ortaya çıktı ve bugün hala kullanımda olan MIL-STD-1553 veri yolu standardını geliştirdik.” [25]. Şekil 4.3 Erwin Gangl’ın bahsettiği mimari farkını göstermektedir.



Şekil 4.3. MIL-STD-1553’ün sunduğu veri yolu mimarisine tepeden bakış [22]

MIL-STD-1553, Dijital Dahili Zaman Bölmeli Komuta/Yanıt Çoklamalı Veri Yolu, silah sistemlerinin entegrasyonu için kullanılan temel araçlardan biri haline gelen askeri bir standarttır (şu anda revizyon B'de). Standart, veri yoluna bağlı alt sistemler için iletişim yöntemini ve elektriksel ara yüz gereksinimlerini açıklamaktadır. 1 Mbps seri haberleşme veri yolu “yarı dubleks” formdadır. Standart, diferansiyel sinyalleşme kullanır, burada veriler bükülü kablolar aracılığıyla iletilmektedir. Bu, otobüsün herhangi bir konuşma ve gürültüden etkilenmemesini sağlamaktadır. MIL-STD-1553 ağı için çift yedekli veri yolu ifadesi de kullanılabilmektedir. Ana veri yoluna Birincil veya “A” veri yolu ve diğerine İkincil veya “B” veri yolu adı verilir.

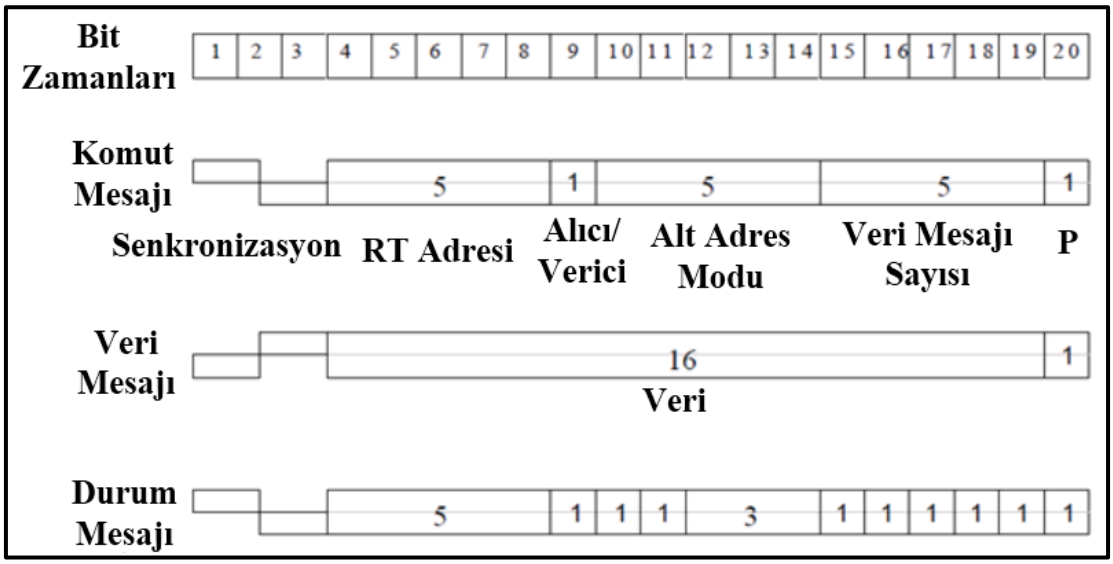


Şekil 4.4. MIL-STD-1553 ağ yapısı [22]

Şekil 4.4'te gösterildiği üzere MIL-STD-1553 ağında 3 temel unsur vardır:

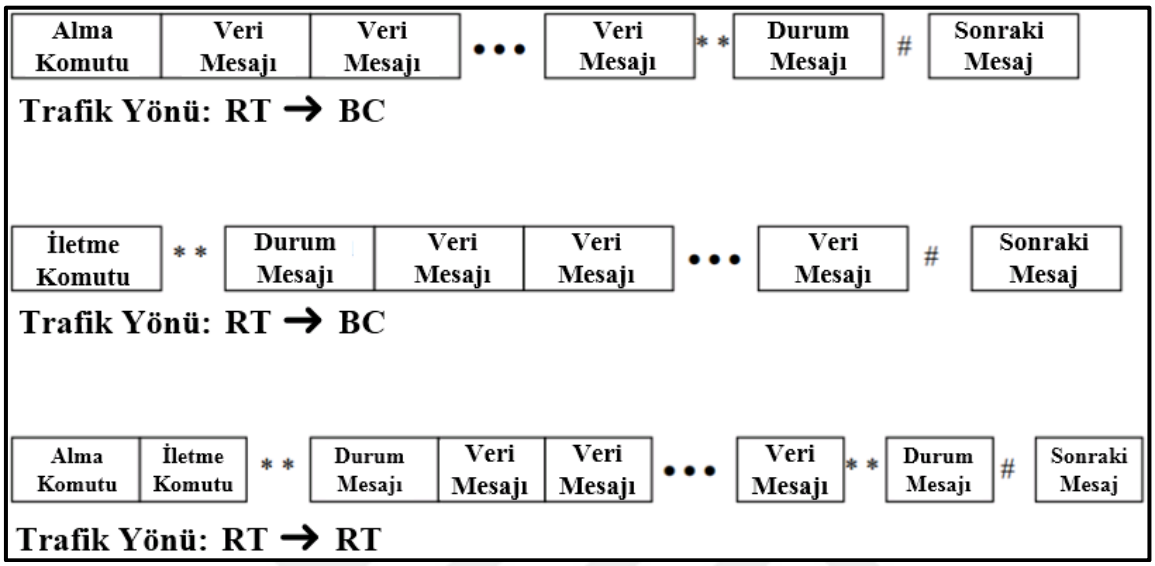
1. Veri Yolu Denetleyicisi (BC): Veri yolu denetleyicisinin ana işlevi, veri yolu üzerindeki tüm iletilen mesajlar için veri akışı kontrolü sağlamaktır.
2. Uzak Terminal(ler) (RT): Gömülü uzak terminal, doğrudan veri yoluna bağlı bir sensör veya alt sistem içinde bulunan ara yüz devresinden oluşur. Birincil görevi, veri yolu denetleyicisi tarafından kontrol edilen uç sistemin ilgili kısımlarına veri aktarımını gerçekleştirmektir. Uzak Uç Birimler olarak da adlandırılırlar.
3. Veri Yolu Monitörü (BM): Veri Yolu monitörü ağdaki tüm iletilen mesajları dinler ve ardından veri yolundan veri toplar. Bu çalışma modunun birincil uygulamaları şunları içerir: yerleşik toplu depolama veya uzaktan telemetri için veri toplama veya sistem ve alt sistemlerin durumunu ve çalışma modunu gözlemlemek için "sıcak" veya çevrimdışı bir yedekleme denetleyicisi içinde kullanım.

Mesajlar 16 bit kelime (komut, veri veya durum) üzerinden iletilir. Her kelime 3 µs senkronizasyon darbesi ile başlar ve bir eşlik biti ile biter. MIL-STD-1553B için eşlik biti tek bittir. Pratikte her kelime 20 bit olarak kabul edilebilir: 3 bit senkronizasyon için, 16 bit faydalı yük (asıl iletilen veri) için ve 1 bit eşlik kontrolü için. Bir mesajdaki kelimeler ardışık olarak iletilmektedir. Her kelime arasında 4 µs'lik bir gecikme süresi vardır. Uzak terminaller, veri yolu denetçisinden gelen mesajlara 12 µs içinde yanıt vermektedirler.



Şekil 4.5. MIL-STD-1553 mesaj formatı [22]

Şekil 4.5'te gösterildiği üzere standart 3 adet çerçeve formatı tanımlar: Komut Çerçevesi, Veri Çerçevesi, Durum Çerçevesi. Veri Yolu Kontrolcüsü, iletişimi her zaman bir komut sözcüğü ile başlatır.



Şekil 4.6.MIL-STD-1553 ağındaki veri iletimi [22]

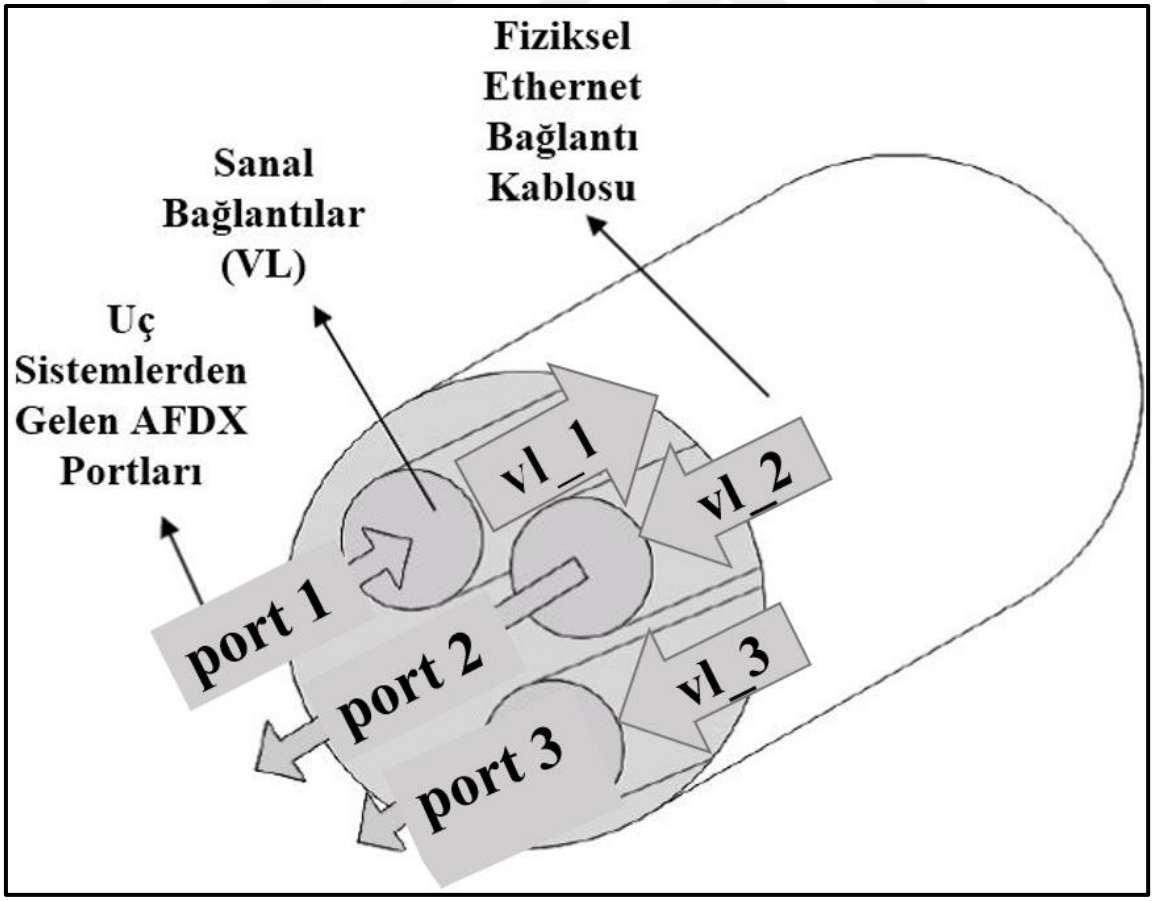
Şekil 4.6'da MIL-STD-1553 ağındaki uzak terminal (RT) ve Veri Yolu Kontrolcüsü (BC) arasındaki farklı mesaj iletimleri gösterilmiştir.

4.3. Aviyonik Tam Çift Yönlü Anahtarlamalı Ethernet: AFDX

1990'lı yıllarda, aviyonik birimler arasındaki iletişimi hızlandırma ve ağdaki hizmet kalitesini arttırmak adına IEEE 802.3 Ethernet standardının aviyonikte kullanılabilirliği üzerine çalışmalar yapılmıştır. İlk bakışta Ethernet protokolü, hem yüksek hızlı hem de Rafta Hazır Ürünler (COTS) ile düşük maliyetli bir şekilde kolayca operasyonel hale getirilebildiği için tercih sebebi olarak görülebilir. Ancak, Taşıyıcı Algılama ve Çarpışma Algılama Çoklu Erişim (CSMA/CD) gibi olasılıksal protokollere ve temel aviyonik ağ gereksinimlerinin belirleyiciliği ve güvenilirliğini karşılayamayan yedeksiz mimariye dayanmaktadır. Bu sebeplerden ötürü, Airbus firmasının geliştirdiği Aviyonik Tam Çift Yönlü Anahtarlamalı Ethernet (AFDX), artan aviyonik sistem gereksinimlerini karşılamak adına son yıllarda önde çıkan bir aviyonik ağ protokolüdür. MIL-STD-1553 ve ARINC-429 gibi geleneksel aviyonik veri yolu ağlarıyla karşılaştırıldığında, AFDX 100Mbps'ye kadar yüksek hızlara çıkabildiğinden, COTS Ethernet anahtarlarını kullanarak genel

maliyeti azalttığından ve esnek mimariye sahip olduğundan kullanım alanı günden güne artmaktadır. Bunun yanında 2000'li yıllardan itibaren popülerleşen, uçaktaki aviyonik birimlerin işlevlerini tek bir aviyonik bilgisayarında çalışan yazılım paketlerinde birleştiren Entegre Modüler Aviyonik (IMA) mimarisini desteklediği için de tercih edilen bir protokol olmuştur.

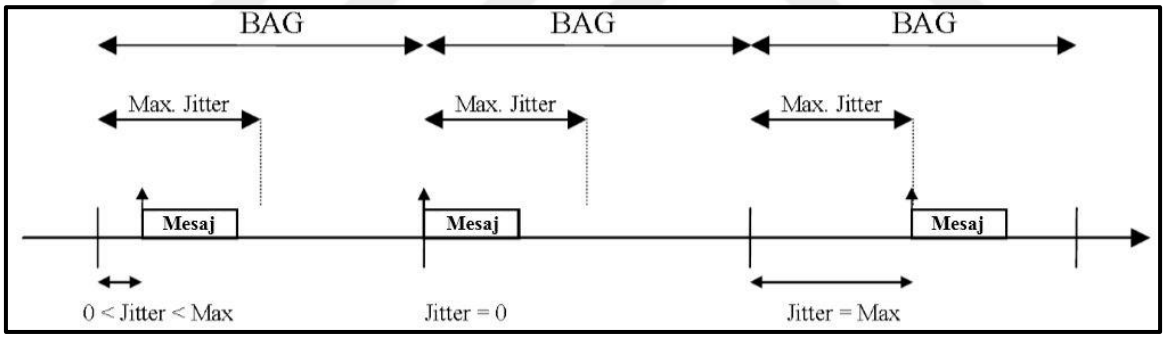
AFDX ağları üç ana parçadan oluşmaktadır: Uç Sistem (ES), anahtarlar (switch) ve fiziksel/sanal iletişim bağlantıları. Uç sistem, alt sistemler (örneğin uçuş kontrol bilgisayarı) ile AFDX ağı arasındaki ara yüzdür. AFDX anahtarı, AFDX ağının kaynak uç sistemini hedef uç sistemine bağlayan merkezi noktasıdır. AFDX ağındaki mesajlar elektro-manyetik girişime dayanıklı bükümlü çift kablolarda, AFDX protokolünün tanımladığı sanal bağlantılara (VL) bölünerek iletilmektedir. Sanal bağlantılar (VL), Şekil 4.7'de gösterildiği gibi bir kaynak uç sistemden bir veya daha fazla hedef uç sisteme giden mantıksal tek yönlü bağlantıları tanımlar.



Şekil 4.7. AFDX ağındaki fiziksel bağlantıların içindeki sanal bağlantılar [26]

Aslında AFDX, VL konseptini kullanarak anahtar üzerinden sanal bir ARINC-429 ağı kurmuş olmaktadır. Bant Genişliği Tahsis Boşluğu (BAG), Şekil 4.8'de gösterildiği gibi iki ardışık çerçeve arasındaki minimum boşluk süresini tanımlayarak VL'nin bant genişliğini sınırlayan bir zaman dilimidir. BAG değeri 1 - 128ms aralığında olmalı ve 2'nin kuvveti şeklinde olmalıdır. Belirli bir VL için BAG seçimi, VL tarafından bağlantı düzeyinde aktarım sağlanan AFDX bağlantı noktalarının gereksinimlerine bağlıdır.

Paket varış sürelerindeki değişime, sapmaya seğirme (jitter) denir. AFDX haberleşmesinde, son çerçeveden sonraki BAG süresi ile maksimum jitter süresi toplamından sonraki aralıkta, aynı VL'ye ait bir çerçeve gözlemlenmesi beklenmektedir. Bu, AFDX standardında her bir VL'deki jitter süresinin üst sınırının garanti edilmesi ile sağlanmaktadır. Aşağıdaki formülde jitter süresinin hesabına yönelik formül vardır. Görüldüğü üzere, her bir VL için olabilecek en yüksek jitter süresi 500 us ile sınırlandırılmıştır. BAG konseptinin yanında bu şekilde jitter süresine üst sınır tanımlanarak AFDX ağı deterministik yapıya kavuşmuştur.



Şekil 4.8. AFDX mesajlarının BAG içerisinde davranışları [23]

Bir sanal bağlantıda oluşabilecek maksimum seğirme (jitter), AFDX standardı tarafından aşağıdaki eşitlikteki (1) gibi tanımlanmıştır.

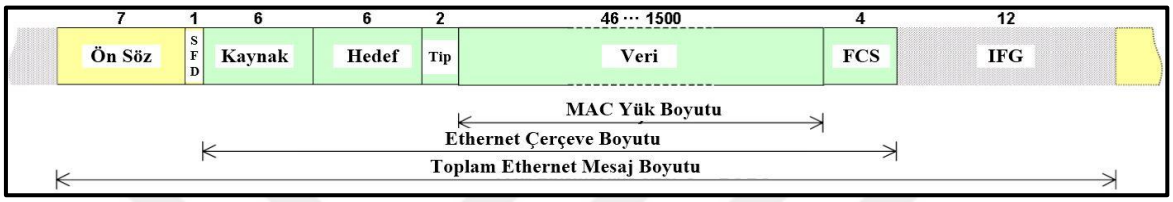
$$Jitter_{max} \leq 40\ \mu s + \frac{\sum(20Bytes + L_{max}Bytes)\ x\ \frac{8bits}{Bytes}}{\frac{N_{BW}bits}{seconds}} \qquad (1)$$

$$Jitter_{max} \leq 500\ \mu s$$

L_{max} maksimum çerçeve boyutudur ve N_{BW} belirli bir VL için bant genişliğidir. (1)'deki tipik sabit teknolojik gecikme τ= 40 μs alınmaktadır. O zaman belirli bir VL için maksimum gecikme süresi

$$Latency_{max} \leq BAG + Jitter_{max} + \tau \quad (2)$$

AFDX paket yapısı Şekil 4.9'da verilmiştir [23].



Şekil 4.9. AFDX paket yapısı [23]

Ethernet önsözü (7 bayt)

Ağ üzerinde yeni bir çerçevenin iletilmesini bildirmek için, ileten ES, asıl çerçevenin iletilmesinden önce önsöz adı verilen bir bayt akışı gönderir. Önsöz, alıcı ES'lere veri gönderiminin başlamasında oluşturulan 0 ve 1 bitlerinden oluşmaktadır. Giriş bölümünün sonunda, veri ileten ES, bu modeli kırmak ve Başlangıç Çerçevesi Ayırıcısı' ndan (SFD) hemen sonra gerçek çerçevenin başlangıcını bildirmek için 1 bit halinde SFD göndermektedir.

MAC adresleme

Sanal Bağlantılar tanımlanırken uç sistemlerin MAC hedef adresiyle tanımlanmaktadırlar. AFDX mesajlarında bu MAC adresi bildirilmelidir.

Protokol türü

MAC adresleri, Ethernet çerçevesinde hangi protokol türünün taşındığını belirtmek için 2 baytlık Ethernet Tipi (Ether Type) kullanılır. Burada, İnternet Protokolü Sürüm 4 (IPv4) için 0x0800 değeri kullanılmaktadır.

Ethernet yükü

Ether Type alanını takip eden Ethernet yükü, IP yapısını, UDP yapısını ve AFDX yükünü ve ardından Sıra Numarasını (SN) içeren Ethernet yüküdür. IP yapısı 20 bayt, UDP yapısı 8 bayt ve SN alanı 1 bayt uzunluğundadır. Ethernet çerçevesinin 64 ila 1518 bayt aralığında olduğu belirtildiğinden, ADFX yükü sonuç olarak 17 ila 1471 bayt aralığında olmalıdır.

Ethernet hata kontrolü

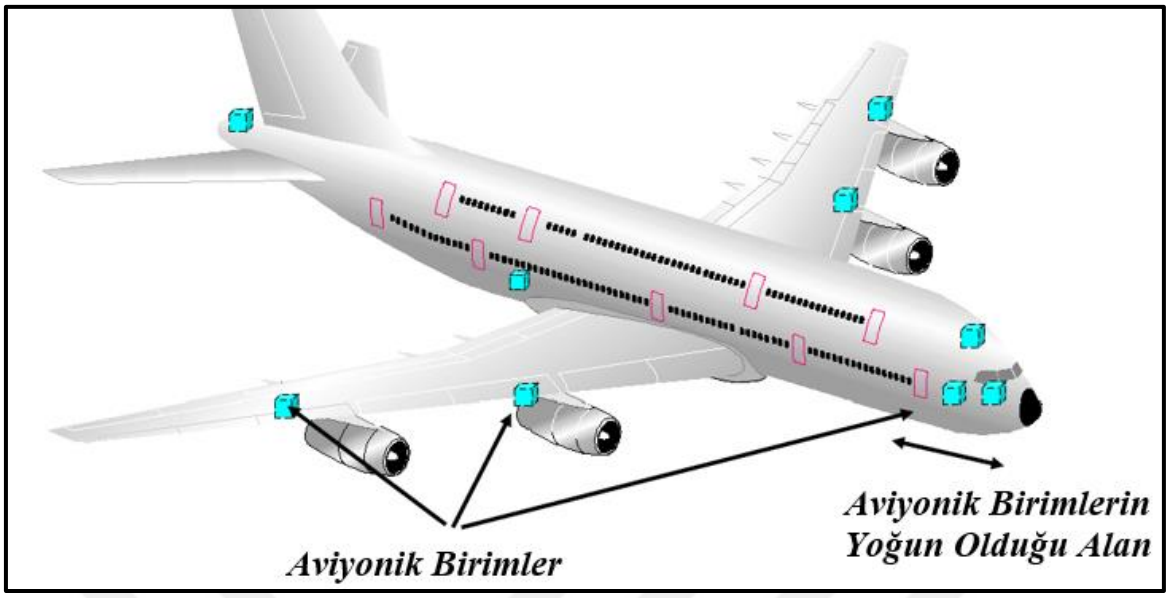
Ethernet çerçevesinin son alanı, 4 bayt uzunluğundaki Çerçeve Kontrol Sırasıdır (FCS). Verici ES, iletilen tüm çerçeve üzerinde Döngüsel Artıklık Sağlama Toplamı (CRC) algoritmasına dayalı olarak bir sağlama hesabı yapmaktadır. Bu daha sonra FCS alanı sonuna veri olarak eklenir. Alıcı ES de tıpkı verici ES gibi CRC algoritmasını kullanarak sağlama hesabı ve karşılaştırma yapar. Böylece iki ES'de CRC algoritması ile hesaplanan sağlama toplamlarının farklı olmasının anlaşılması halinde, alıcı ES alınan mesajı atar, yok sayar.

Ethernet arka sözü

Ethernet IEEE 802.1 tarafından kesinlikle gerekli olmayan çerçevelerin iletimleri arasında minimum bir boşta kalma süresini tanımlanmıştır. Bu süre Çerçeveler Arası Boşluk (IFG) olarak adlandırılmaktadır ve 12 bayttır. Ancak, Ethernet IEEE 802.1 ile AFDX protokollerinin uyumluluk isterleri nedenleriyle IFG, AFDX için de geçerlidir. IFG, ağ üzerinde 96 bit iletmek için geçen süre olarak tanımlanmıştır. 10Mbps ağda, IFG boşta kalma süresi böylece 9,6 mikro saniye olarak karşımıza çıkmaktadır. 100 Mbps hıza ise, IFG boşluk süresi 960 nano saniyedir.

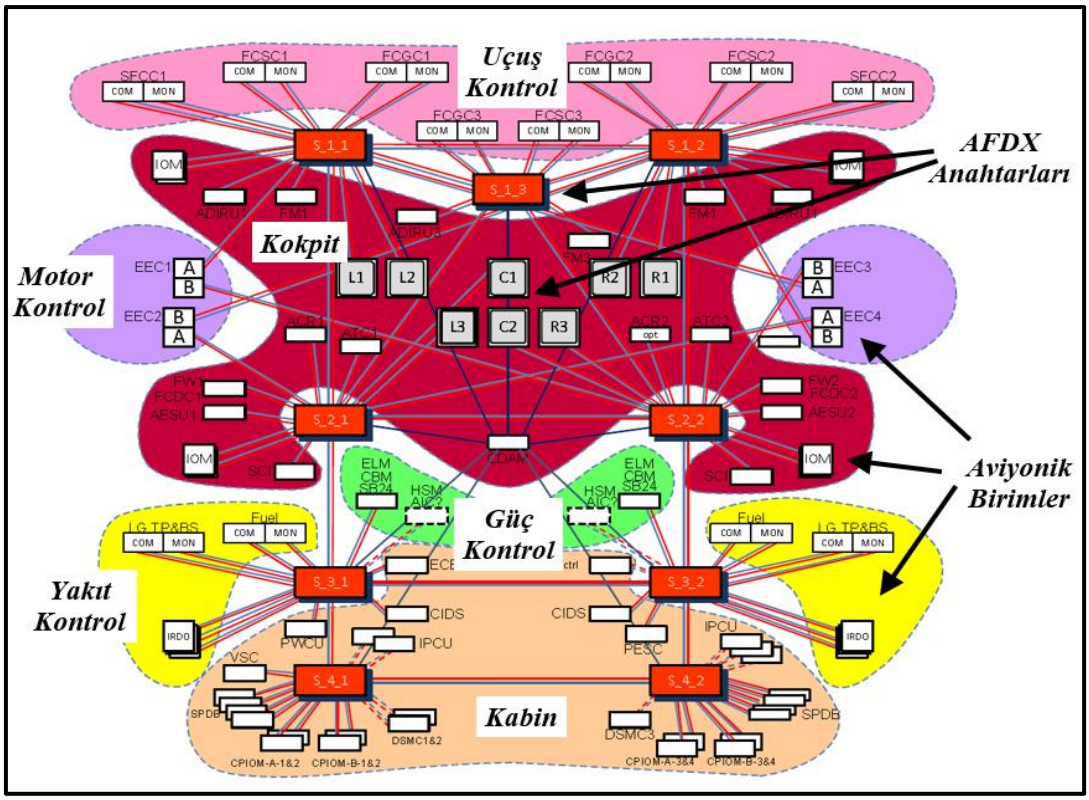
4.3.1 Airbus A380 AFDX ağ mimarisi

Uçaklarda Aviyonik birimlerin dağılımı genel olarak şu şekildedir:



Şekil 4.10. Aviyonik birimlerin uçak içinde dağılımı [27]

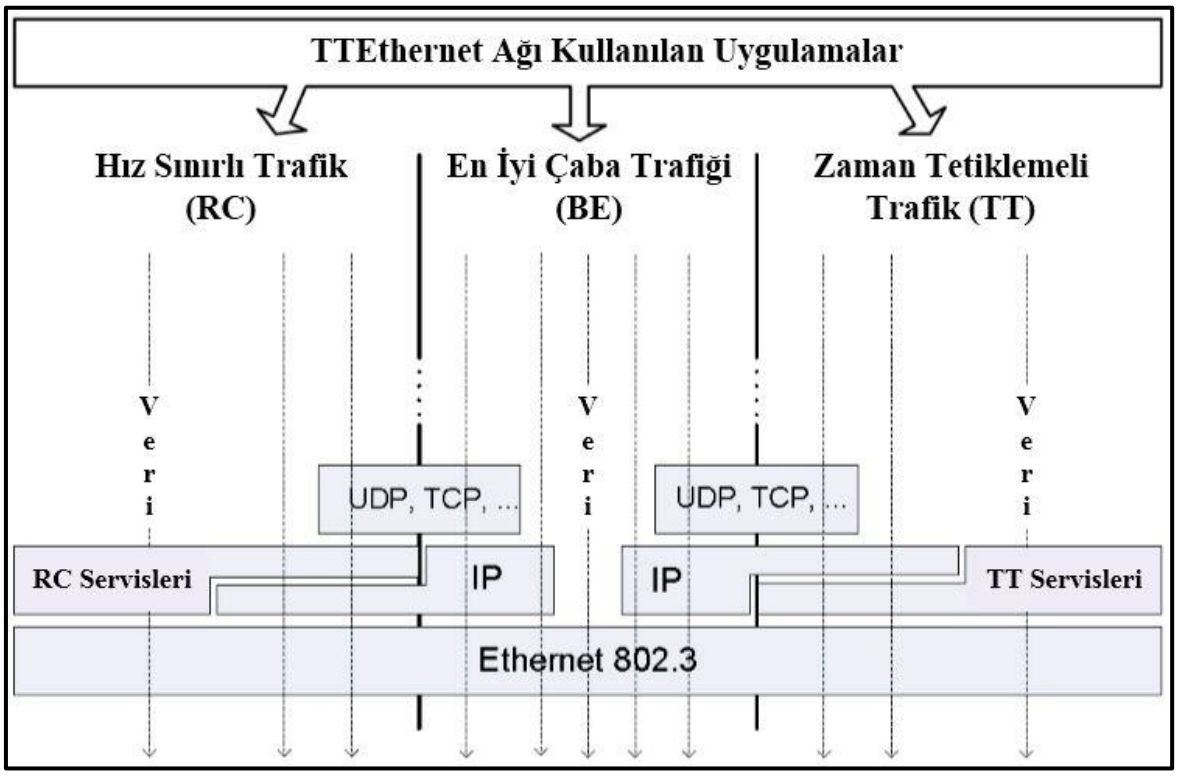
Airbus'ın A380 yolcu uçaklarında kullandığı AFDX ağı yapısı aşağıdaki şekildeki gibi duyurulmuştur. 4. Bölümde A380 mimarisinden çok daha basit bir örnek Aviyonik ağı modellenmiş ve hem AFDX ile hem de TTEthernet ile simüle edilerek sonuçları analiz edilmiştir.



Şekil 4.11. Airbus A380 uçağındaki AFDX ağı yapısı [28]

4.4. Zaman Tetiklemeli Ethernet

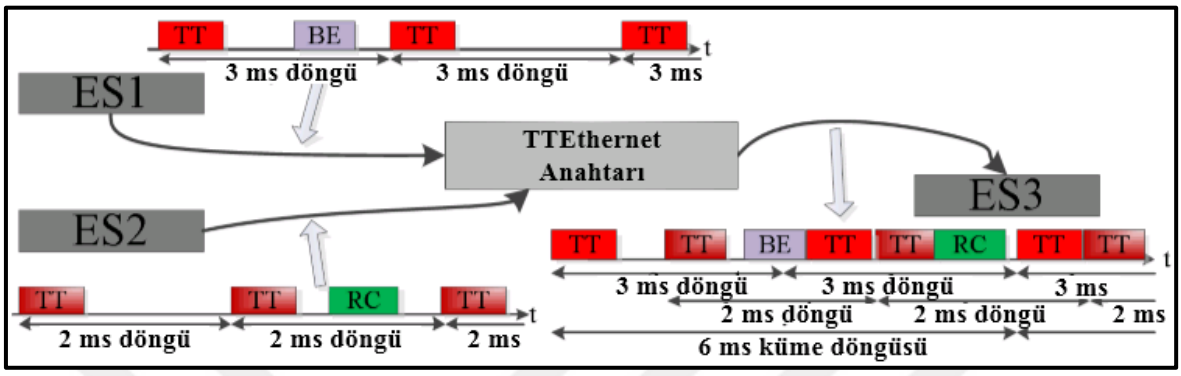
TTEthernet protokolü, Zaman Tetiklemeli (TT), Hız Kısıtlı (RC) ve En İyi Çaba (BE) olmak üzere üç farklı trafik sınıfını tanımlamaktadır. RC trafiği neredeyse AFDX trafiğiyle aynıdır, atanmış VL numaraları ve BAG değerleri vardır, ancak adları Kritik Trafik Kimliği ve BAG hesaplarıdır. BE trafiği, iyi bilinen IEEE 802.3 Ethernet trafiğidir. Şekil 4.12'de bu durum gösterilmiştir. Bu üç trafik de Şekil 4.9'da verilen AFDX çerçevesi gibi IEEE 802.3 Ethernet paket yapısını temel almaktadır.



Şekil 4.12. TTEthernet ağının kullandığı trafikler [24]

TTEthernet ağları uç sistemlerden ve anahtarlardan oluşur. Bir kaynak uç sistemden üretilerek trafiğe sunulan akışların, ağdaki anahtarlar aracılığıyla bir veya birden çok hedef uç sisteme gönderilmesi mümkündür. Her TT çerçevesi, önceden hesaplanarak belirlenen bir toplu küme döngüsünde periyodik olarak gönderilmektedir. Şekil 4.13'te gösterildiği gibi, ES1 -> Anahtar -> ES3 yolu boyunca ES1'den üretilen TT trafiği, iletim için 3 ms döngü süresine sahipken, ES2 -> Anahtar -> ES3 yolu boyunca ES2'den üretilen TT trafiği iletim için 2 ms döngü süresine sahiptir. Ağdaki TTEthernet anahtarında, iletilmek için rekabet eden TT trafik akışları için bir çizelge vardır ve çizelge periyodu, bu TT trafik

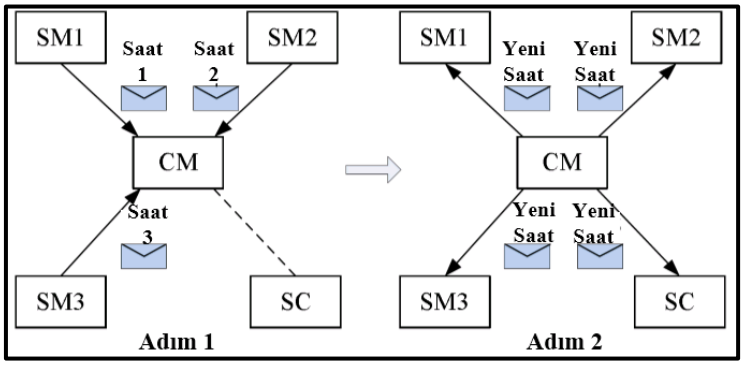
akışlarının döngülerinin en küçük ortak katı tarafından tanımlanmaktadır. Şekildeki örnekte TTEthernet anahtarına 2 ms ve 3 ms döngü sürelerine sahip 2 TT trafik aktığından, küme döngüsü süresi, 2 ms ve 3 ms'nin en küçük ortak katı olan 6 ms'dir.



Şekil 4.13. TTEthernet ağındaki ortak zaman [24]

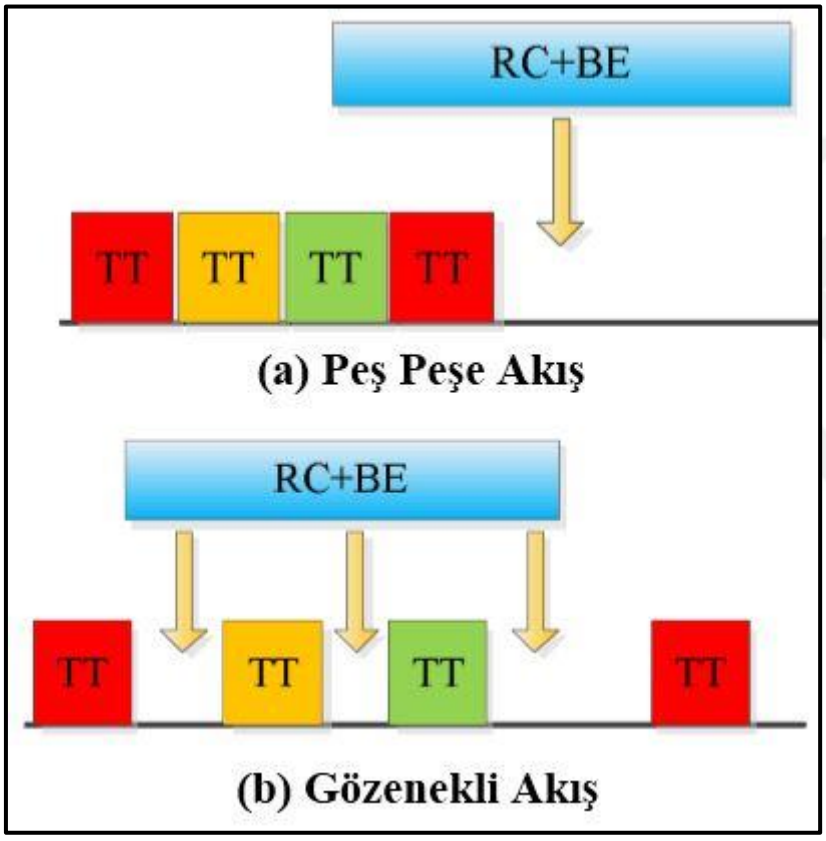
TTEthernet, TT ve RC trafiğinin yanı sıra BE trafiğini de aynı anda destekleyebilmektedir. TT trafiği, protokolün sunduğu en önemli özelliktir. Ağa bağlı tüm cihazların küresel senkronizasyonuna dayanan TTEthernet, TT çerçevelerinin Zaman Bölmeli Çoklu Erişim (TDMA) yapısıyla aktarılmasını mümkün kılmaktadır. Bu şekildeki örnekte RC ve BE trafik akışları belirli bir program dâhilinde değildir.

TTEthernet ayrıca cihazlar ve anahtarlar arasında senkronizasyonu sağlamak için kullanılan Protokol Kontrol Çerçevesi (PCF) adı verilen başka bir paket türü kullanmaktadır. TTEthernet, AFDX'e benzer bir ağ yapısına sahiptir, anahtarlar ve ES'ler vardır, ancak bunlar üç farklı parça olarak yapılandırılır: Senkronizasyon yöneticisi (SM), sıkıştırma yöneticisi (CM) ve senkronizasyon istemcisi (SC). Aşağıdaki şekilde bu yapı bünyesinde iki aşamada global zaman hesaplama yapısı gösterilmiştir.



Şekil 4.14. TTEthernet ağındaki saat senkronizasyonu [24]

SM, PCF'yi CM'ye gönderir ve ardından CM, alınan PCF'ye dayalı olarak global zamanı hesaplar ve hesaplanan global zaman, SC'ye ve diğer cihazlara güncellenmiş bir PCM olarak yayınlanmaktadır. Bu şekilde, hiçbir cihazda küresel zamanı takip etme konusunda bir çelişki olmaz ve seğirme, standart tanımlandığı gibi mikrosaniye cinsinden sabitlenmektedir. TDMA yapısına sahip bir mikrosaniye jitter'ı, tam gecikmeyi hesaplamamıza olanak tanıyarak, ağı kesin olarak belirleyici hale getirerek TTEthernet'in zaman açısından kritik uygulamalar için çok iyi bir çözüm olduğunu gösterektedir.

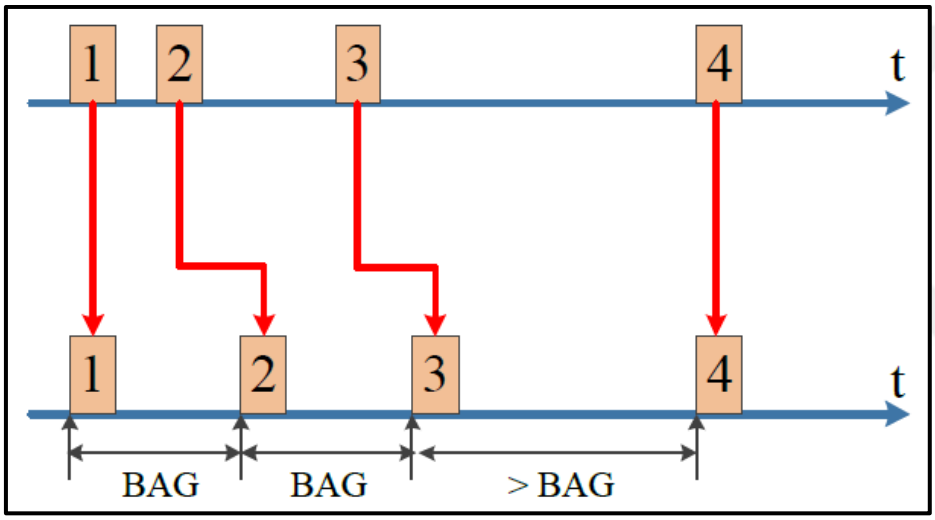


Şekil 4.15. TTEthernet ağındaki trafik akışı regülasyonu [12]

TTEthernet ağında, her TT çerçevesi, çevrimdışı bir TT planına (Zaman Tetiklemeli Plan) göre herhangi bir zaman karışıklığı olmadan iletilmektedir. Standart tanımlamaları gereği, RC ve BE trafik akışları, TT trafik akışında herhangi bir ek gecikmeye sebep olmamaktadır. Bu durum, tez çalışmamızın simülasyon sonuçları aşamasında da gösterilmiştir. TT trafik akışının tasarlamasında iki farklı yol vardır. TT Trafik, Şekil 4.15' te gösterildiği gibi peş peşe trafik akışı veya gözenekli biçimde trafik akışı gibi farklı biçimlere iletilebilir.

Kaynak uç sistemlerdeki RC trafik akışları, Şekil 4.16'da gösterildiği gibi Bant Genişliği Tahsis Boşluğu (BAG) süresi ile sınırlandırılır. Anahtar çıkış portunda, RC trafik akışları VL trafik denetleme stratejisi tarafından iletilir. Ağ anahtarında, her bir VL'nin trafik akışını düzenlemek için sızdıran kova (leaky bucket) ve jeton tabanlı kova (token bucket) gibi tıkanıklık açma algoritmaları kullanılır. Ayrıca, VL trafiklerini regüle etmek için ağ anahtarları Round-Robin algoritmasını da kullanmaktadır.

Kaynak uç sistemlerdeki BE trafik akışlarında ise trafik iletiminin zamanla garantisi ve hizmet kalitesi garantisi yoktur.

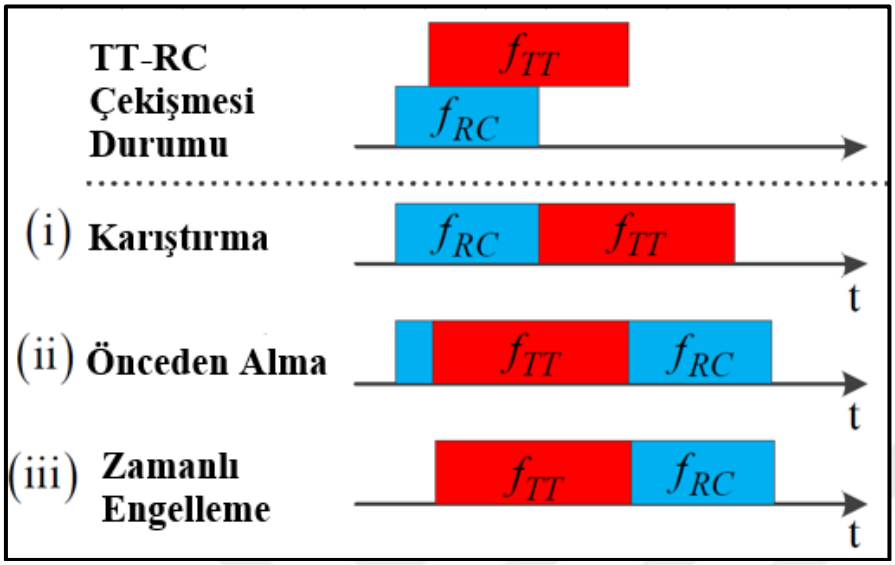


Şekil 4.16. TTEthernet ağındaki mesajların BAG tanımlarına uygun olarak akışı [12]

TTEthernet ağı, her bir TT mesajını iletebilmek için statik program yapılandırması gerektirmektedir. Yüksek güvenilirliğe sahip olan bu statik programın, TTEthernet ağına yeni bir uç sistem veya yeni bir trafik akışı eklenmesi durumunda, tüm uç sistemlerde değiştirilmesi gerektiğinden TTEthernet, bu tür yeni eklemeler için esnek değildir. Ancak yeni eklemelerden sonra yapılacak olan statik program ile sorunsuz yeni bir statik zaman programı elde edilebilir.

TT çerçeveleri statik zaman dilimlerinde planlanır ve kalan dilimlerde RC ve BE çerçeve iletimi için gözenekler kullanılır. TT trafik çizelgesinin gözenekliliği, RC trafik akışlarının ve BE trafik akışlarının iletim performansını etkileyebilme durumu söz konusudur. Bir RC çerçevesi zaten iletimde olduğu esnada yeni bir TT çerçevesi önceden hesaplanmış sabit çizelgede iletilmeye hazır olduğunda, oluşacak çakışmayı/çekişmeyi çözmek için üç

entegrasyon yöntemi vardır [24]. Bunlar, Şekil 4.17'de gösterildiği üzere, karıştırma, önceden alma ve zamanlı engelleme yöntemleridir.



Şekil 4.17. TTEthernet ağındaki trafik çakışması yönetimi [12]

Karıştırma yönteminde, TT çerçevesinin deterministik davranışının ihlal edilmesine sebebiyet verebilmektedir. Bu yöntemde, halihazırda iletilmekte olan RC çerçevesinin iletilmesinin bitmesi beklenir ve sonra TT çerçevesi iletilir. Bu sebepten ilgili TT çerçevesi planlanan zamanda gönderilemeyecektir. Bu yöntem, pratikte genel olarak dikkate alınmamaktadır ve uygulanmamaktadır. Önceden alma yöntemi, RC çerçevesinin önceden alınabileceği ve TT çerçeve iletimi bittikten sonra iletilmeye devam edilebileceği bir senaryo tanımlamaktadır. Zamanlı Engelleme yönteminde, daha sonra TT çerçevelerinin gelmesinden önce bir RC çerçevesinin iletimi tamamlanamazsa, RC çerçevesi bu TT çerçevelerinin iletimi bitene kadar bloke edilmektedir.

[1], [21], [22], [23], [24]'e göre aviyonik ağların karşılaştırılması Çizelge 4.1'de gösterilmiştir.

Çizelge 4.1. Aviyonik ağların karşılaştırılması [1], [21-24]

	Protokoller			
Metrikler	ARINC-429	MIL-STD 1553	AFDX	TTEthernet
Ortam Erişimi	Noktadan-noktaya bağlantı yapısı	Çok aktarmalı veri yolu yapısı	Tam çift yönlü anahtarlı yapı	Tam çift yönlü anahtarlı yapı, planlı zaman tetiklemeli yapı
Maksimum Veri Hızı	12.5 Kbps / 100 Kbps	1 Mbps	100 Mbps / 1Gbps	100 Mbps / 1 Gbps ve daha yüksek hızlara çıkmak mümkün
Fiziksel Katman	Diferansiyel, empedans kontrollü katman	Diferansiyel, empedans kontrollü katman	IEEE 802.3 fiziksel katmanı	IEEE 802.3 fiziksel katmanı
Bant Genişliği Kullanımı	El sıkışmaya dayanan Williamsburg / Buckhorn Protokolü	Zaman bölmeli çoğullama (TDM)	Bant-Kısıtlı İstatistiksel Çoğullama	BE ve RC trafik için Bant-Kısıtlı İstatistiksel Çoğullama, TT trafik için zaman bölmeli çoklu erişim (TDMA)
Deterministik (Belirleyici) Zamanlama	Ayarlanabilen gecikme ile çok deterministik	Ayarlanabilen gecikme ile çok deterministik	Üst limit ile sınırlandırılmış gecikme ile çok deterministik	Sabit gecikme ve mikrosaniye düzeyinde seğirme (jitter) ile kesin olarak deterministik
Yedekli Yapı	Yok	Çift yedekli yapı	Çift yedekli yapı	Çift, üçlü veya daha fazla yedekli yapı
Hata Toleranslı Saat	Yok	Yok	Yok	Var
Trafik Senkronizasyonu	Yok	Yok	Yok	Var
Kritik Görevlere Uyum	Var	Var	Var	Sadece TT ve RC trafik için var

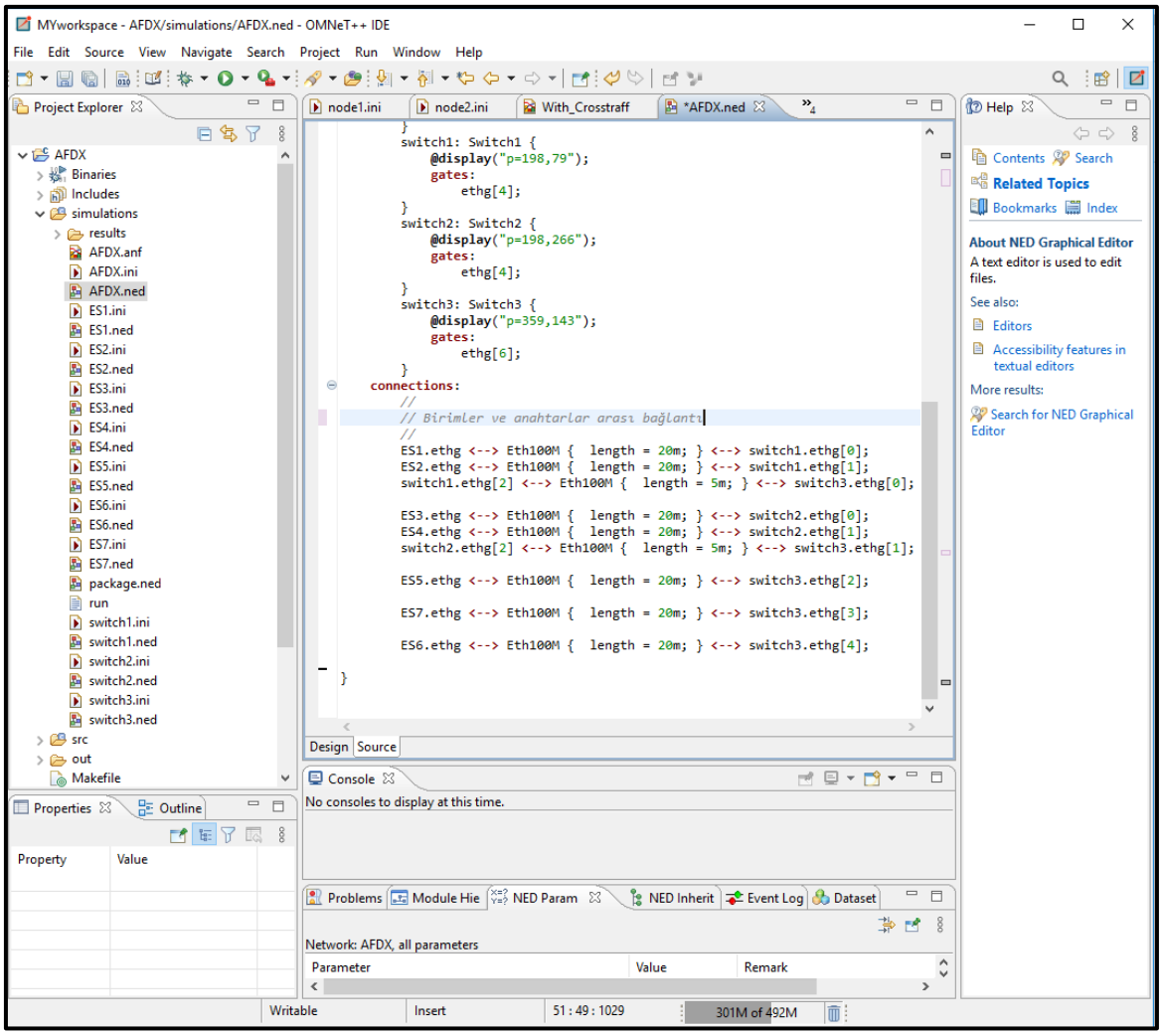
5. AVİYONİK AĞI MODELLENMESİ VE SİMÜLASYONU

Bu bölümde, OMNET++ [10] simülasyon ortamı kullanılarak bir aviyonik ağı seçilip modellenerek simüle edilmiş ve sonuçlar analiz edilmiştir.

5.1. Simülasyon Ortamı: OMNET++

OMNeT++, ayrık olay simülasyonu için bir geliştirme ortamıdır. Nesne yönelimli olması, farklı ağ senaryolarının kurulmasını ve sistemlerinin performanslarının değerlendirmelerinde kolaylık sağlamaktadır. İletişim ağlarını (hem kablolu hem de kablosuz) modellemenin yanı sıra, yeni bir protokol modellemesi, kuyruk oluşturma ve dağıtılmış donanım sistemlerini modellemek için de kullanılabilir. C++'ta yazılmış kütüphanelere sahip olduğu için yeni kütüphaneler tanımlamada miras alma yöntemi vesilesiyle ağ modellemede kolaylıklar sağlamaktadır.

Bu tez çalışmasında OMNET++'ta kullanılmak üzere açık kaynaklı olarak geliştirilmiş olan 2 popüler iskelet kütüphane kullanılmıştır: INET [29] İskeleti ve CORE4INET İskeleti.
Bu tez çalışmasında OMNET++ ortamının seçilme sebebi modellemenin INET ve CORE4INET [13] iskeletleri ile esnek bir şekilde yapılabilmesi ve OMNET++' ın akademik kullanımlar için herkes tarafından ücretsiz erişilebilmesidir. Şekil 5.1'de OMNET++ programının genel görüntüsü verilmiştir.



Şekil 5.1. OMNET++ simülasyon ortamı [10]

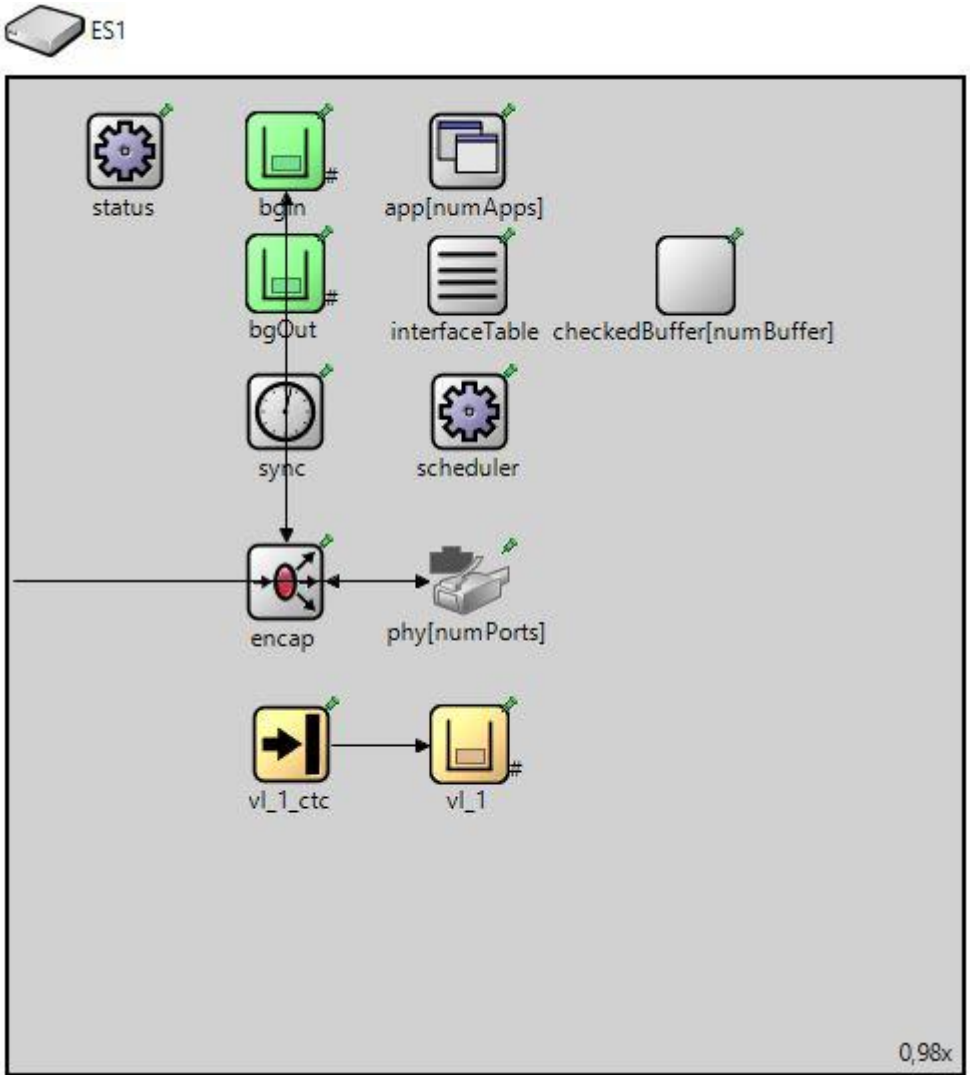
INET iskeleti (framework) kablolu/kablosuz iletişim ağlarının geliştirilmesini desteklemektedir. IEEE 802.3 Ethernet protokolünün alt elemanlarını C++'ta model oluşturarak hazır simülasyon yapıları sunmaktadır. Ücretsiz ve açık kaynak kodlu olarak sunulan INET iskeleti aşağıdaki yapıları sunan bir model kütüphanesidir:

- Tam çift yönlü veri iletimi
- 10Mbps and 100Mbps bağlantı hızları,
- Fast-Ethernet, Gigabit ve Multi-Gigabit-Ethernet Protokolleri
- Ethernet IEEE 802.1 MAC modeli (EtherMAC), LLC modeli (EtherLLC) ve çerçeve modeli (EtherFrame)
- Çoklu cihaz modelleri olan hub modeli (EtherHub), anahtar modeli (EtherSwitch), uç sistem modeli (EtherHost)

INET iskeletinin bu şekilde Ethernet IEEE 802.1 ağı modüllerini sunması, benzer ağların rahatça modellenebilmesinde kolaylıklar sunmaktadır.

CORE4INET İskeleti ise INET İskeleti üzerine Hamburg Üniversitesi'ndeki bir araştırma ekibi tarafından geliştirilmiştir. CORE4INET ile IEEE 802.1 Audio Video Bridging, IEEE 802.1Q / Time Sensitive Networking, CAN ve TTEthernet protokolünü OMNET++'ta modellemek mümkündür.

Simülasyonda kullanılan Uç Sistem (ES) modülünün içyapısı şu şekildedir:



Şekil 5.2. Uç sistem (ES1) modeli [13]

Şekil 5.2'de görülen modüller simülasyon yapılırken kolaylık olması adına CORE4INET kütüphanesinde C++ dilinde birer sınıf olarak tanımlanmıştır. Örneğin AFDX trafiğindeki

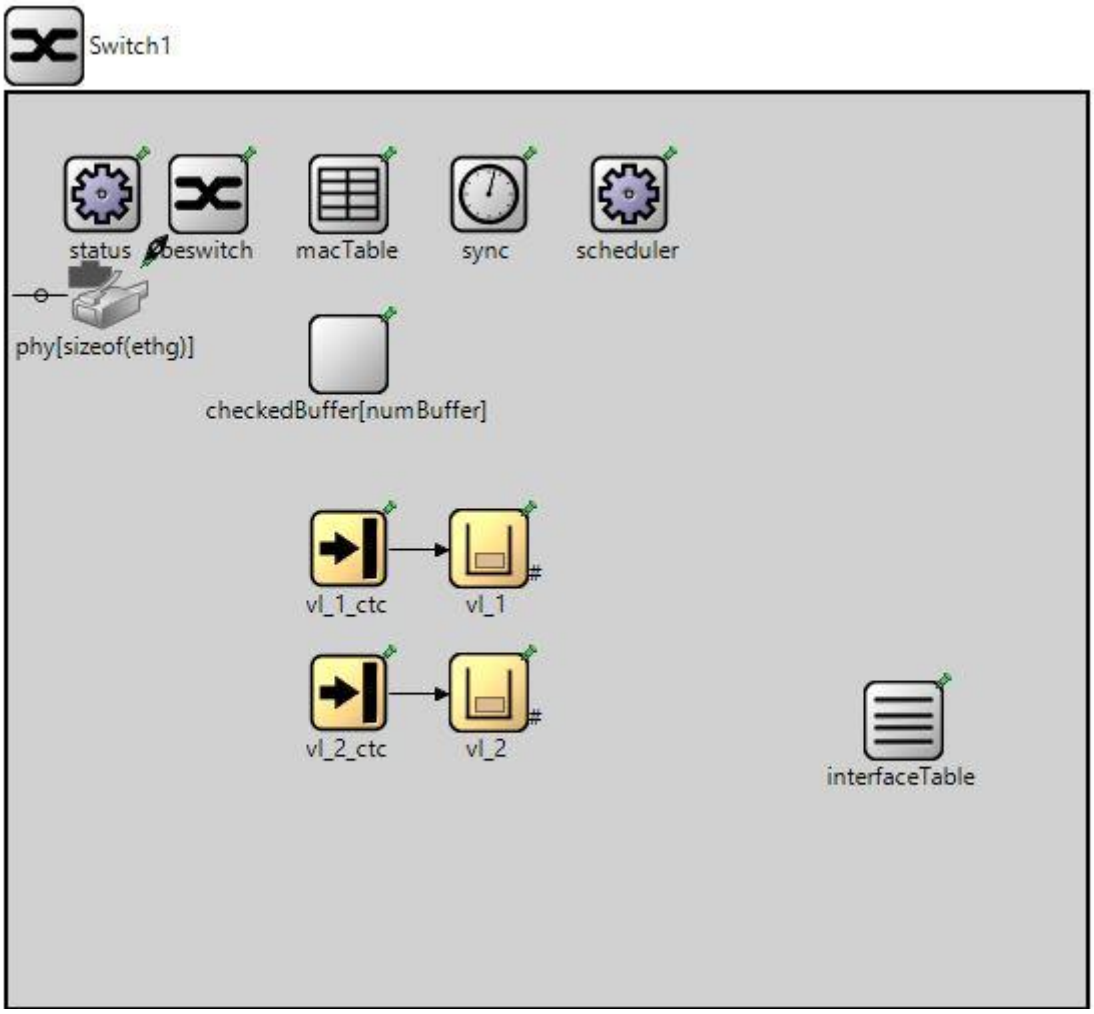
(RC Trafik)" sanal bağlantı 1'i (vl_1) oluşturmada kullandığımız ve "vl_1" diye isimlendirdiğimiz "RCQueueBuffer" modülünün kullandığı "RCBuffer" sınıfının kaynak kodu şu şekildedir:

```
#ifndef CORE4INET_RCBUFFER_H_
#define CORE4INET_RCBUFFER_H_
#include "core4inet/buffer/AS6802/CTBuffer.h"
#include "core4inet/utilities/classes/Timed.h"

namespace CoRE4INET {
/**
   * Hız kısıtlamalı (RC Trafik) bir sınıf oluşturmak için temel sınıfı tanımlar.
   * Gelen paket depolanır ve bir önceki BAG süresi bitince trafiğe sunulur.
   * BAG son kullanma zamanı geçmemişse çerçeve saklanır.
   * RCDoubleBuffer, RCQueueBuffer modülleri ile RC trafik modellemesi yapılmalıdır.
   * @yazar Till Steinbach
 */
class RCBuffer : public virtual CTBuffer, public virtual Timed
{
    private:
        // BAG süresinin dolduğunu gösteren Boole değeri
        bool bagExpired;
        // Hedef Port'lar tarafındaki sıfırlama mesajlarının sayısı.
        // BAG değeri yalnızca tüm Port'larda resetBag() çağrıldığında sıfırlanır.
        unsigned int numReset;
        // Son paketin saat bilgisini kaydededer.
        uint64_t lastSent;
        // Önbellekte tutulan BAG değerini gösterir.
        uint64_t bag;
        // Önbellekte tutulan jitter değerini gösterir.
        uint64_t jitter;

    public:
        // Yapıcı Fonksiyon (Constructor)
        RCBuffer();
        // Yapıcı Fonksiyon (Destructor)
        virtual ~RCBuffer() override = 0;
        // Bu fonksiyon, bir hedef port için BAG değerini sıfırlar.
        void resetBag();

    protected:
        using Timed::initialize;
        // timerMessage fonksiyonunu başlatır.
        virtual void initialize(int stage) override;
        // Bu modülün ihtiyaç duyduğu başlatma aşamalarının sayısını döndürür.
        virtual int numInitStages() const override;
        // RC trafik buffer'ının gelen ve giden mesajlarını işler
        virtual void handleMessage(cMessage *msg) override;
        // Herhangi bir parametrede değişiklik olup olmadığını belirtir.
        virtual void handleParameterChange(const char* parname) override;
        // Modülün, iletim için ihtiyaç duyduğu bant genişliğini hesaplar.
        virtual long getRequiredBandwidth() override;
};
} //namespace
#endif
```

Şekil 5.3. RCBuffer sınıfının C++ dilindeki kaynak kodu [13]

Görüldüğü üzere, CORE4INET kütüphanesi simülasyonlar için gerekli olabilecek her türlü fonksiyonu C++'ta uygulamıştır.

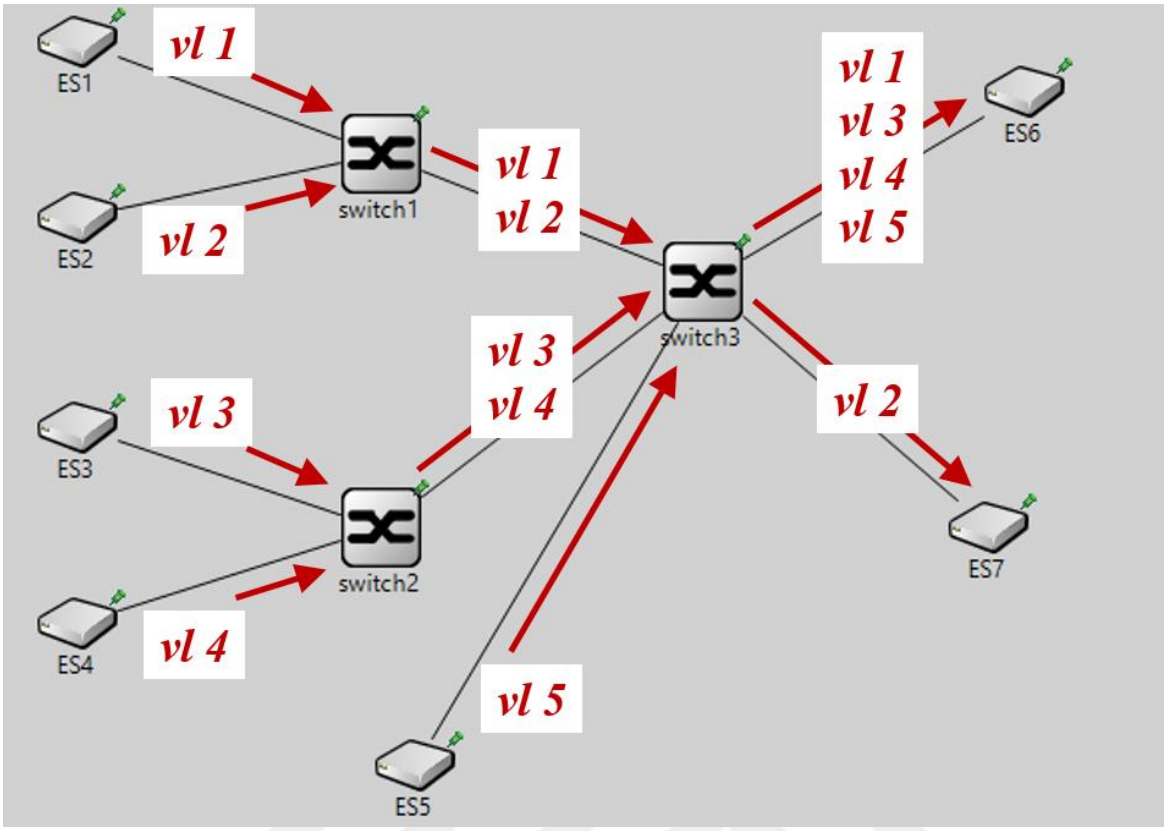
Simülasyonda kullanılan anahtar yapısı aşağıdaki şekilde modellenmiştir.



Şekil 5.4. Anahtar (Switch1) modeli [13]

5.2. OMNET++'ta Modellenen ve Simüle Edilen Mimari

OMNET++'ta modellenen ve simüle edilen Aviyonik ağ mimarisi Şekil 5.5'te verilmiştir. Burada, uç sistemlerden (ES) çıkan sanal bağlantılar (vl) gösterilmiştir. Bu sanal bağlantılar hem AFDX modelinde hem de TTEthernet modelinde mevcuttur.



Şekil 5.5. AFDX ağ modelinin şekilsel gösterimi [3], [5], [13-14]

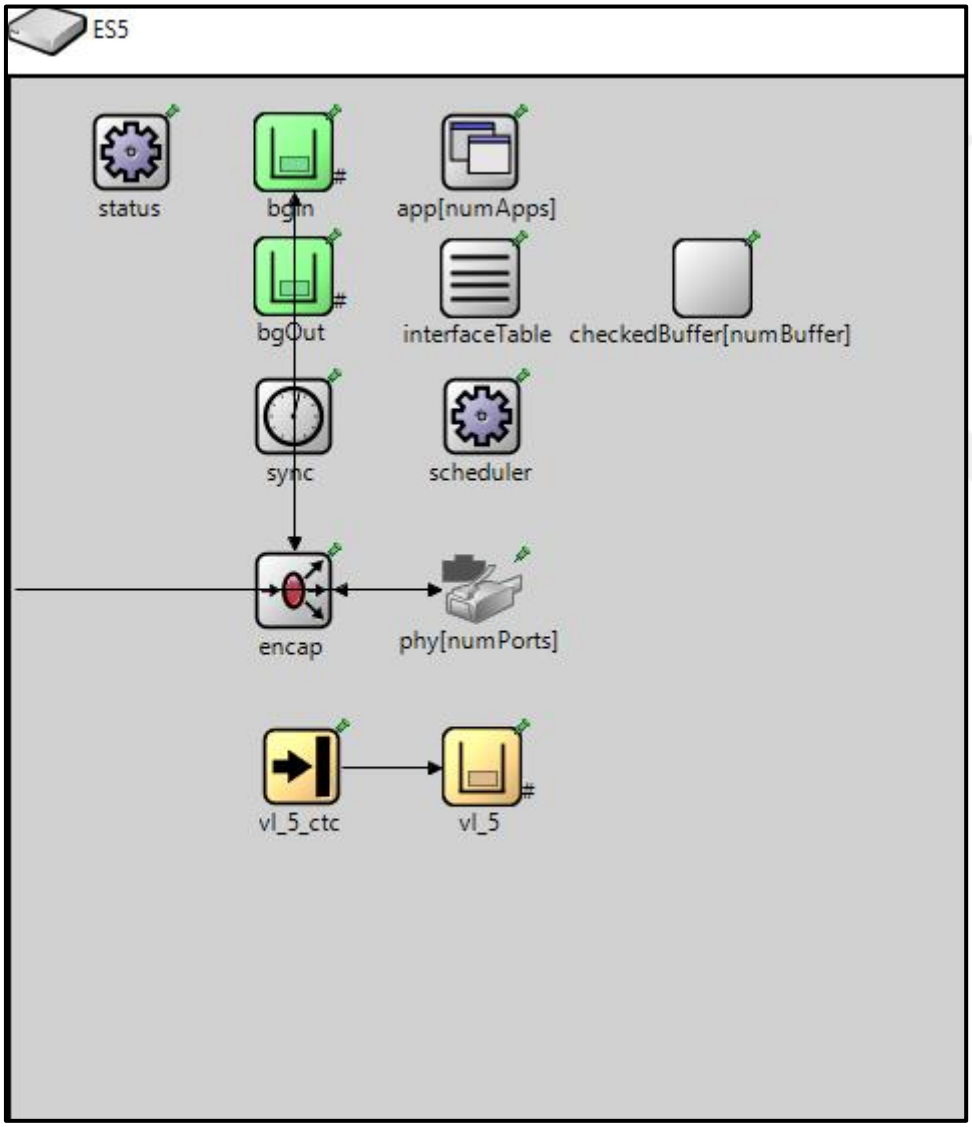
Bu mimari tanımlanırken, INET kütüphanesini miras alan CORE4INET kütüphanesindeki "ethernet" kütüphanesi kullanılmıştır. Kurduğumuz aviyonik ağ modelinin kaynak kodu ve konfigürasyonu aşağıdaki gibidir.

```
package afdx.simulations;
// Bu örnekte INET kütüphanesinin ethernet modülü kullanılmıştır"
import inet.node.ethernet.Eth100M;

network AFDX
{
    @display("bgb=629,419");
    submodules:
        ES1: ES1 {
            @display("p=46,23");
        }
        ES2: ES2 {
            @display("p=46,111");
        }
        ES3: ES3 {
            @display("p=46,216");
        }
        ES4: ES4 {
            @display("p=46,314");
        }
        ES5: ES5 {
            @display("p=215,385");
        }
        ES6: ES6 {
            @display("p=520,48");
        }
        ES7: ES7 {

            @display("p=502,266");
        }
        switch1: Switch1 {
            @display("p=198,79");
            gates:
                ethg[4];
        }
        switch2: Switch2 {
            @display("p=198,266");
            gates:
                ethg[4];
        }
        switch3: Switch3 {
            @display("p=359,143");
            gates:
                ethg[6];
        }
    connections:
        // Birimler ve anahtarlar arası bağlantı aşağıdaki gibidir.
        ES1.ethg <--> Eth100M {  length = 20m; } <--> switch1.ethg[0];
        ES2.ethg <--> Eth100M {  length = 20m; } <--> switch1.ethg[1];
        switch1.ethg[2] <--> Eth100M {  length = 5m; } <--> switch3.ethg[0];
        ES3.ethg <--> Eth100M {  length = 20m; } <--> switch2.ethg[0];
        ES4.ethg <--> Eth100M {  length = 20m; } <--> switch2.ethg[1];
        switch2.ethg[2] <--> Eth100M {  length = 5m; } <--> switch3.ethg[1];
        ES5.ethg <--> Eth100M {  length = 20m; } <--> switch3.ethg[2];
        ES7.ethg <--> Eth100M {  length = 20m; } <--> switch3.ethg[3];
        ES6.ethg <--> Eth100M {  length = 20m; } <--> switch3.ethg[4];
}
```

Şekil 5.6. AFDX ağ modelinin NED dilindeki kaynak kodu [13]

5.3. RC Trafik Senaryosu ile AFDX Mimarisi Simülasyonu

TTEthernet standardının tanımladığı RC trafik akışı AFDX standardının bire bir uygulamasıdır. Bu tez çalışmasında gerçeklenen AFDX ağ simülasyonunda CORE4INET yapısı kullanılmıştır. Modellemenin daha anlaşılır olması için vl_5 sanal bağlantısının ağdaki trafik konfigürasyonunu anlatan ES5, switch3 ve ES6 elemanlarının konfigürasyonları aşağıdaki gibi verilmiştir:



Şekil 5.7. AFDX ağ modelindeki uç sistem 5'in (ES5) modeli [13]

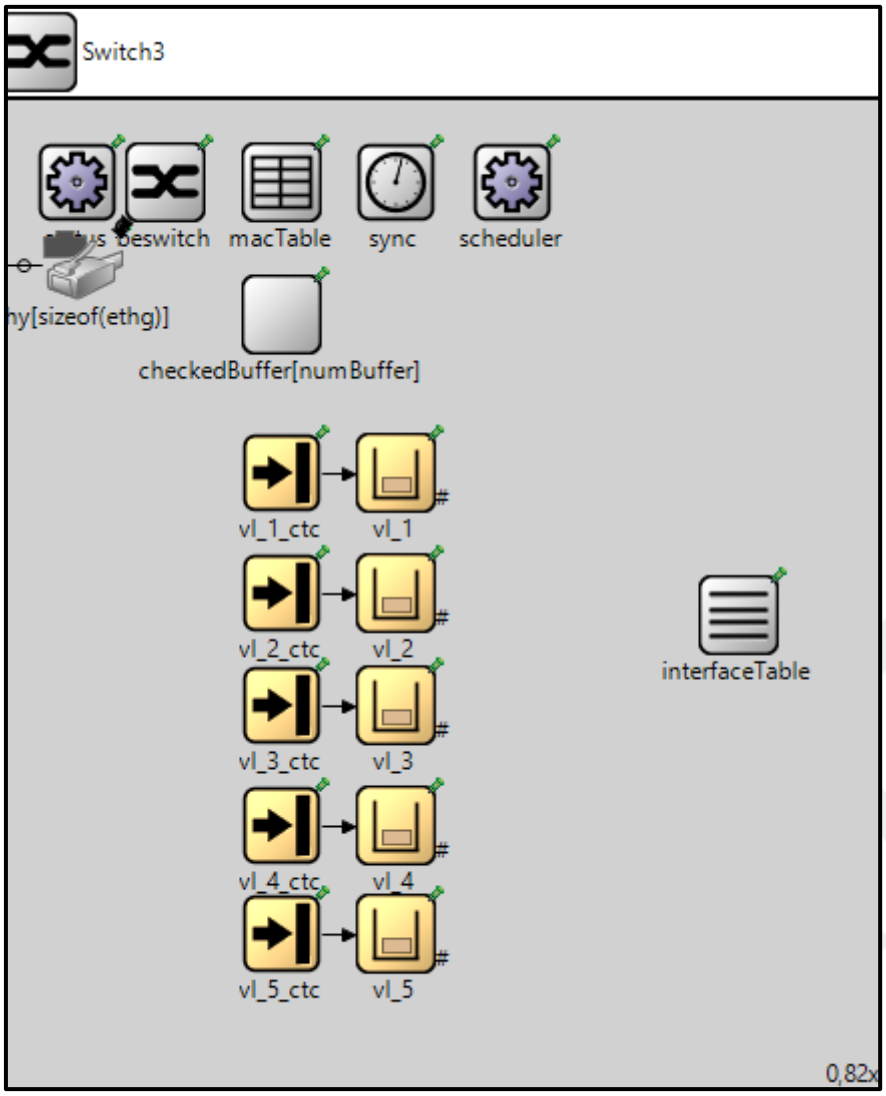
AFDX ağ senaryosunu simüle etmek için CORE4INET kütüphanesindeki RCBuffer modülleri (sarı ile gösterilen) kullanılmıştır.

```
network = AFDX

# ES5'in MAC Adresi
**.ES5.phy[*].mac.address = "0A-00-00-00-00-05"
# RC trafiği oluşturan ES5'in gerçek zamanlı
# çalışan uygulama sayısını tanımlar.
**.ES5.numApps = 1
# ES5'ten çıkan vl_5 isimili sanal bağlantıyı tanımlar.
**.ES5.app[0].buffers = "vl_5"
# RC trafiği oluşturan ES5'in gerçek zamanlı çalışan
# uygulamasını tanımlar.
**.ES5.app[0].typename = "RCTrafficSourceApp"
**.ES5.app[0].displayName = "vl_5"
**.ES5.app[0].interval = 5ms
**.ES5.app[0].payload = 500Byte

# Zaman-Kritik Trafik No (Sanal Bağlantı Numarası) = 1
**.ES5.app[0].ct_id = 1
# Connect the output vl_5 to the physical port phy[0].
**.ES5.vl_5.destination_gates = "phy[0].RCin"
# BAG değerini tanımlar.
**.ES5.vl_5.bag = sec_to_tick(4ms)
**.ES5.vl_5.priority = 0
**.ES5.vl_5.ct_id = 1
```

Şekil 5.8. AFDX ağ modelindeki uç sistem 5'in (ES5) NED dilindeki kaynak kodu [13]

Yukarıdaki şekilde ES5'in iç konfigürasyonu tanımlanmıştır.

OMNET++ programı C++'ta yazıldığı ve nesne yönelimli bir yapıya sahip olduğu için CORE4INET kütüphanesini kullanırken trafik tanımlama için gerekli bütün fonksiyonlara ulaşılabilmektedir. Örneğin, tanımlanan RC trafik uygulamasında "payload" fonksiyonunu kullanarak iletilen paketlerin boyutu belirlenebilmektedir. Aynı şekilde, AFDX standardının tanımladığı BAG değerini de belirtmek için "bag" fonksiyonunu kullanmak gerekmektedir.



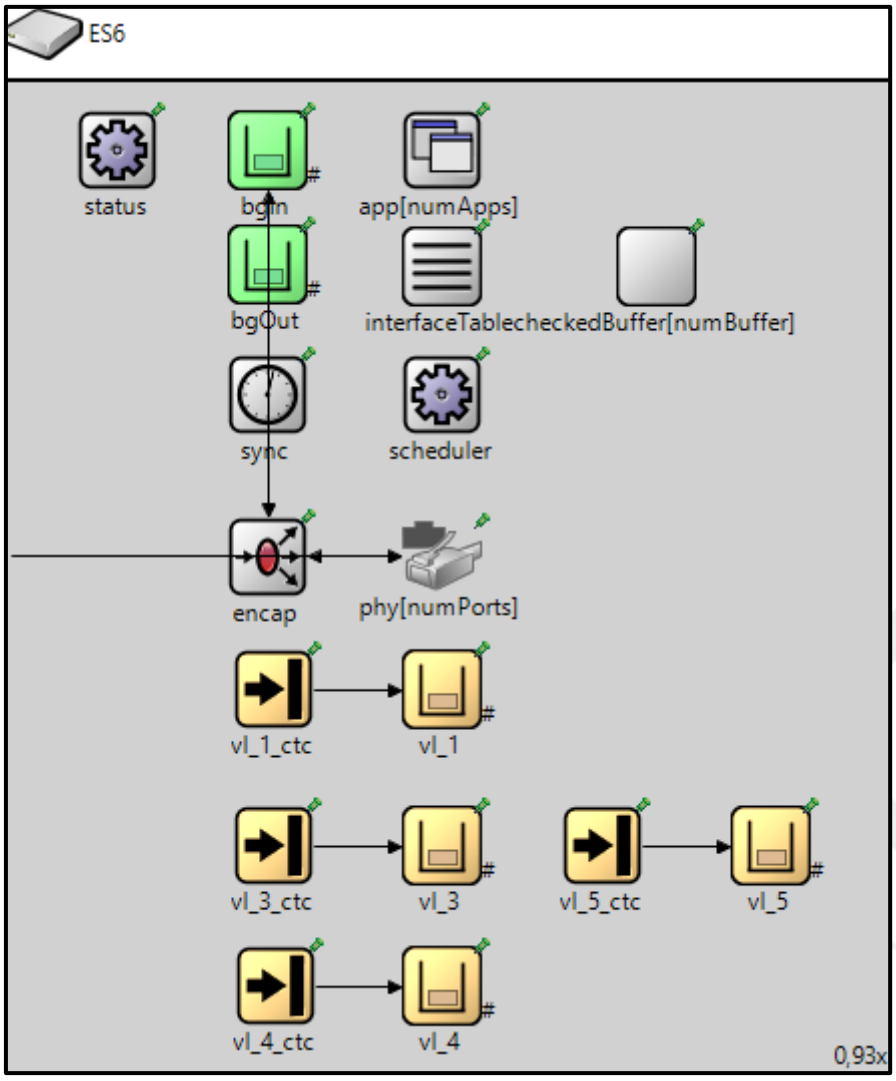
Şekil 5.9. AFDX ağ modelindeki anahtar 3'ün (Switch3) modeli [13]

Şekil 5.9, ağdaki Anahtar 3 (switch3)'ün modelini göstermektedir. Sarı renkli modüller RCBuffer modülleridir ve RC Trafik yapılandırılmasında kullanılmaktadır.

```
network = AFDX

# Sanal bağlantıların Anahtarın hangi portlarına
# gittiklerini ve hangi porttan geldiklerini tanımlar.
# VL1, VL2 => PHY0
**.switch3.phy[0].inControl.ct_incomings = "vl_1_ctc, vl_2_ctc"
**.switch3.vl_1_ctc.bag = sec_to_tick(4ms)
**.switch3.vl_2_ctc.bag = sec_to_tick(4ms)

# VL3, VL4 => PHY1
**.switch3.phy[1].inControl.ct_incomings = "vl_3_ctc, vl_4_ctc"
**.switch3.vl_3_ctc.bag = sec_to_tick(4ms)
**.switch3.vl_4_ctc.bag = sec_to_tick(4ms)

# VL5 => PHY2
**.switch3.phy[2].inControl.ct_incomings = "vl_5_ctc"
**.switch3.vl_5_ctc.bag = sec_to_tick(4ms)

# PHY[3] => VL2 => ES7
**.switch3.vl_2.destination_gates = "phy[3].RCin"
**.switch3.vl_2.bag = sec_to_tick(4ms)
**.switch3.vl_2.ct_id = 3

# PHY[4] => VL1, VL3, VL4, VL5 => ES6
**.switch3.vl_1.destination_gates = "phy[4].RCin"
**.switch3.vl_1.bag = sec_to_tick(4ms)
**.switch3.vl_1.ct_id = 2
**.switch3.vl_3.destination_gates = "phy[4].RCin"
**.switch3.vl_3.bag = sec_to_tick(4ms)
**.switch3.vl_3.ct_id = 4
**.switch3.vl_4.destination_gates = "phy[4].RCin"
**.switch3.vl_4.bag = sec_to_tick(4ms)
**.switch3.vl_4.ct_id = 5
**.switch3.vl_5.destination_gates = "phy[4].RCin"
**.switch3.vl_5.bag = sec_to_tick(4ms)
**.switch3.vl_5.ct_id = 1
```

Şekil 5.10. AFDX ağ modelindeki anahtar 3'ün NED dilindeki kaynak kodu [13]

Yukarıdaki şekilde anahtar 3'ün (switch 3) iç konfigürasyonu tanımlanmıştır. CORE4INET kütüphanesinin tanımladığı "ct_incomings" ve "destination_gates" fonksiyonları ile anahtardaki trafik giriş ve çıkış noktaları tanımlanabilmektedir.



Şekil 5.11. AFDX ağ modelindeki uç sistem 6'nın (ES6) modeli [13]

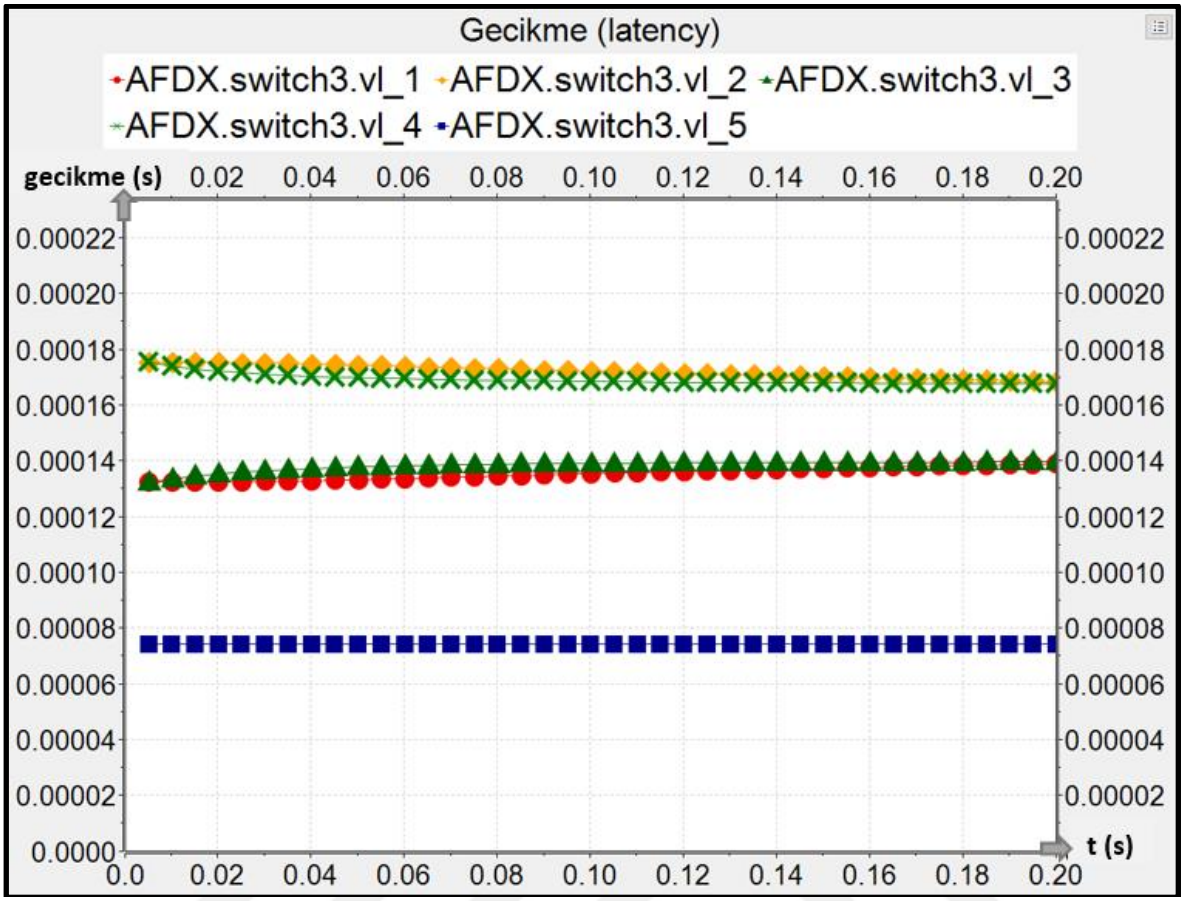
Şekil 5.9'da da gösterildiği üzere, Şekil 5.11'deki sarı renkli modüller RCBuffer modülleridir ve AFDX trafiği oluşturmada kullanılmaktadır.

```
[General]
network = AFDX

# ES6'nın MAC Adresi
**.ES6.phy[*].mac.address = "0A-00-00-00-00-06"
# RC trafiği oluşturan ES6'in gerçek zamanlı
# çalışan uygulama sayısını tanımlar.
**.ES6.numApps = 4

# ES6'ya gelen sanal bağlantıları tanımlar.
**.ES6.app[0].typename = "CTTrafficSinkApp"
**.ES6.app[0].displayName = "vl_1"
**.ES6.vl_1_ctc.bag = sec_to_tick(4ms)
**.ES6.vl_1.destination_gates = "app[0].RCin"
**.ES6.vl_1.ct_id = 2

**.ES6.app[1].typename = "CTTrafficSinkApp"
**.ES6.app[1].displayName = "vl_3"
**.ES6.vl_3_ctc.bag = sec_to_tick(4ms)
**.ES6.vl_3.destination_gates = "app[1].RCin"
**.ES6.vl_3.ct_id = 4

**.ES6.app[2].typename = "CTTrafficSinkApp"
**.ES6.app[2].displayName = "vl_4"
**.ES6.vl_4_ctc.bag = sec_to_tick(4ms)
**.ES6.vl_4.destination_gates = "app[2].RCin"
**.ES6.vl_4.ct_id = 5

**.ES6.app[3].typename = "CTTrafficSinkApp"
**.ES6.app[3].displayName = "vl_5"
**.ES6.vl_5_ctc.bag = sec_to_tick(4ms)
**.ES6.vl_5.destination_gates = "app[3].RCin"
**.ES6.vl_5.ct_id = 1

# ES6'ya gelen sanal bağlantıların ES6'nın phy[0]
# portuna geldiğini gösterir.
**.ES6.phy[0].inControl.ct_incomings = "vl_1_ctc, vl_3_ctc, vl_4_ctc, vl_5_ctc"
```

Şekil 5.12. AFDX ağ modelindeki uç sistem 6'nın (ES6) NED dilindeki kaynak kodu [13]

Yukarıdaki konfigürasyon ES6'nın iç yapısını tanımlamaktadır.

Bu konfigürasyonlardan anlaşıldığı üzere ES5, ürettiği RC trafiği vl_5 sanal bağlantısı aracılığıyla aradaki anahtar üzerinden ES6'ya iletir. Eğer aynı Aviyonik mimarisi AFDX yerine ARINC-429 ile gerçeklenseydi, arada anahtar olmayacaktır ve vl_5 sanal bağlantısı doğrudan olarak fiziksel bağlantı şeklinde tasarlanacaktır ve uçaktaki kablo yükü artacaktır.

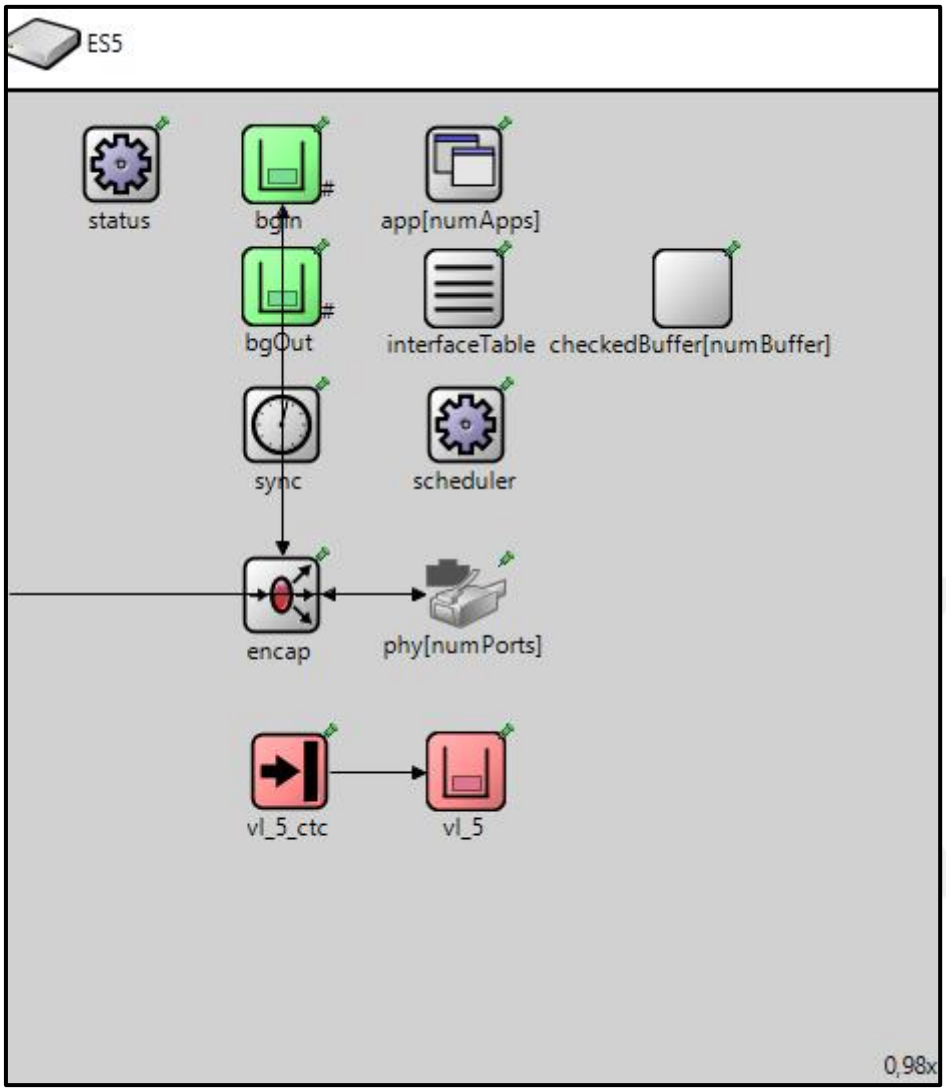


Şekil 5.13. AFDX ağ simülasyonu sonucunda ölçülen iletim gecikmeleri [10]

Yukarıdaki şekilde AFDX ağ simülasyonu sonucundaki sanal bağlantılardaki trafik gecikmeleri gösterilmiştir. Anahtar 3'e en yakın bağlantı vl_5 olduğu için en az gecikmeyi 750 milisaniye olarak vl_5'te ölçülmüştür. Diğer sanal bağlantılardaki gecikmenin vl_5'ten fazla olmasının sebebi ise onların anahtar 1 ve anahtar 2'den gelerek daha uzun mesafe kat etmeleridir.

5.4. Sadece TT Trafiğe Sahip Senaryo

TTEthernet protokolünün tanımladığı TT trafik, mikrosaniye mertebelerinde seğirme (jitter) garanti etmektedir. Hem bu sabit seğirmeyi hem de bunun sonucu olarak gelen sabit gecikme sürelerini, en başta tanımladığımız ağ topolojisini kullanarak görmek adına bütün sanal bağlantıları TT olarak konfigüre edilmiştir. Konfigürasyon parametrelerini daha iyi anlamak adına AFDX senaryosunda olduğu gibi vl_5 sanal bağlantısının izlediği yolun (ES5 -> switch3 -> ES7) konfigürasyonu aşağıda verilmiştir.



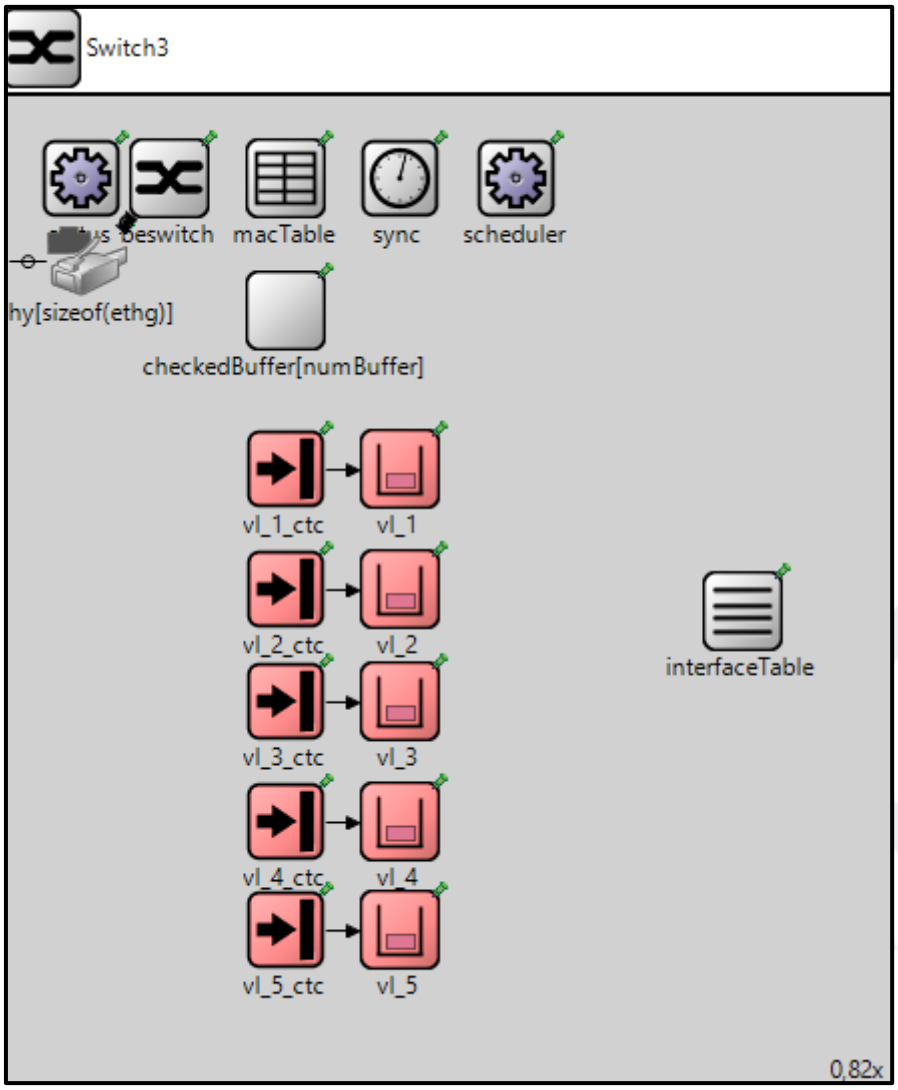
Şekil 5.14. TT trafikteki uç sistem 5'in (ES5) modeli [13]

ES5 modelinde görüldüğü üzere, vl_5 sanal bağlantısında AFDX'ten farklı olarak kullanılan modül "TTBuffer" yapısında bir modüldür. TTBuffer kaynak kodu aşağıda verilmiştir.

```
#include "core4inet/buffer/AS6802/TTBuffer.h"
#include "core4inet/buffer/AS6802/TTBufferEmpty_m.h"
#include "core4inet/scheduler/SchedulerMessageEvents_m.h"
#include "core4inet/utilities/ConfigFunctions.h"

using namespace CoRE4INET;
// TTBuffer sınıfı, zaman tetiklemeli kuyruk yapısını tanımlar
// @yazar Till Steinbach

TTBuffer::TTBuffer()
{

TTBuffer::~TTBuffer()
{

int TTBuffer::numInitStages() const
{

void TTBuffer::initialize(int stage)
{

// TT trafik buffer'ının gelen ve giden mesajlarını işler
void TTBuffer::handleMessage(cMessage *msg)
{

// Herhangi bir parametrede değişiklik olup olmadığını belirtir.
void TTBuffer::handleParameterChange(const char* parname)
{

uint64_t TTBuffer::nextSendWindowStart() const
{

// Modülün, iletim için ihtiyaç duyduğu bant genişliğini hesaplar.
long TTBuffer::getRequiredBandwidth()
{

// TT paketlerin gönderim zamanınının başlangıcını gösterir.
uint32_t TTBuffer::getSendWindowStart()
{

// TT paketlerin gönderim zamanınının bitişini gösterir.
uint32_t TTBuffer::getSendWindowEnd()
{
```

Şekil 5.15. TTBuffer sınıfının C++ dilindeki kaynak kodu [13]

TT trafik için kullanılan anahtarın (switch3) modeli aşağıdaki gibidir:



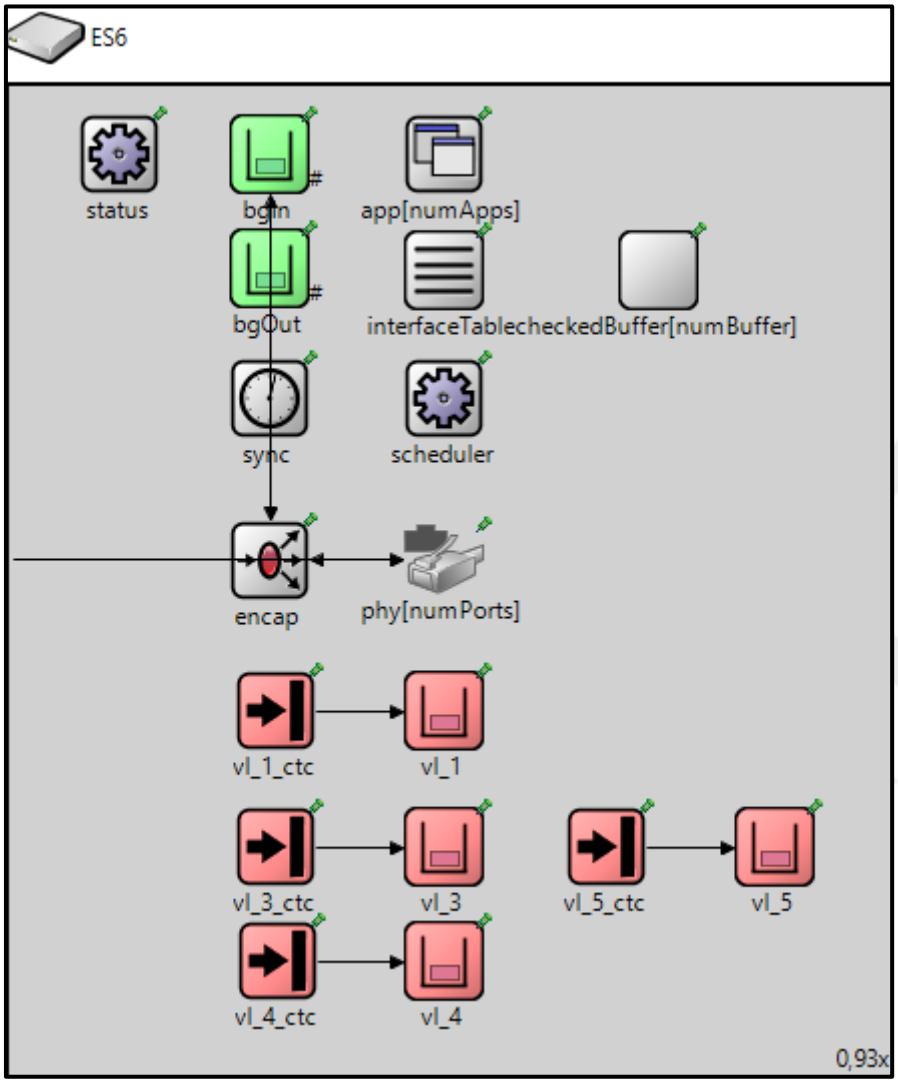
Şekil 5.16. TT trafikteki anahtar 3'ün (Switch3) modeli [13]

Görüldüğü üzere TT trafik ileten 5 adet sanal bağlantı bulunmaktadır. Bu sanal bağlantıların giriş ve çıkış port'larına olan atamaları ve TT trafiğin zaman-tetiklemeli ayarları aşağıdaki konfigürasyonda yapılmıştır.

```
[General]
network = TTE

# Sanal bağlantıların Anahtarın hangi portlarına
# gittiklerini ve hangi porttan geldiklerini tanımlar.

# VL1, VL2 => PHY0
**.switch3.phy[0].inControl.ct_incomings = "vl_1_ctc, vl_2_ctc"
# VL3, VL4 => PHY1
**.switch3.phy[1].inControl.ct_incomings = "vl_3_ctc, vl_4_ctc"
# VL5 => PHY2
**.switch3.phy[2].inControl.ct_incomings = "vl_5_ctc"

# PHY[3] => VL2 => ES7
**.switch3.phy[3].shaper.tt_buffers = "vl_2"
**.switch3.vl_2.destination_gates = "phy[3].TTin"
**.switch3.vl_2.ct_id = 3
# PHY[4] => VL1, VL3, VL4, VL5 => ES6
**.switch3.phy[4].shaper.tt_buffers = "vl_1, vl_3, vl_4, vl_5"
**.switch3.vl_1.destination_gates = "phy[4].TTin"
**.switch3.vl_1.ct_id = 2
**.switch3.vl_3.destination_gates = "phy[4].TTin"
**.switch3.vl_3.ct_id = 4
**.switch3.vl_4.destination_gates = "phy[4].TTin"
**.switch3.vl_4.ct_id = 5
**.switch3.vl_5.destination_gates = "phy[4].TTin"
**.switch3.vl_5.ct_id = 1

# Sanal Bağlantı 1'in zaman-tetikli konfigürasyonu
**.switch3.vl_1_ctc.receive_window_start = sec_to_tick(1020.120us)
**.switch3.vl_1_ctc.receive_window_end = sec_to_tick(1040.120us)
**.switch3.vl_1_ctc.permanence_pit = sec_to_tick(1040.120us)
**.switch3.vl_1.sendWindowStart = sec_to_tick(1049.960us)

# Sanal Bağlantı 2'in zaman-tetikli konfigürasyonu
**.switch3.vl_2_ctc.receive_window_start = sec_to_tick(1045.120us)
**.switch3.vl_2_ctc.receive_window_end = sec_to_tick(1065.120us)
**.switch3.vl_2_ctc.permanence_pit = sec_to_tick(1065.120us)
**.switch3.vl_2.sendWindowStart = sec_to_tick(1074.960us)

# Sanal Bağlantı 3'in zaman-tetikli konfigürasyonu
**.switch3.vl_3_ctc.receive_window_start = sec_to_tick(1070.120us)
**.switch3.vl_3_ctc.receive_window_end = sec_to_tick(1090.120us)
**.switch3.vl_3_ctc.permanence_pit = sec_to_tick(1090.120us)
**.switch3.vl_3.sendWindowStart = sec_to_tick(1099.960us)

# Sanal Bağlantı 4'in zaman-tetikli konfigürasyonu
**.switch3.vl_4_ctc.receive_window_start = sec_to_tick(1095.120us)
**.switch3.vl_4_ctc.receive_window_end = sec_to_tick(1115.120us)
**.switch3.vl_4_ctc.permanence_pit = sec_to_tick(1115.120us)
**.switch3.vl_4.sendWindowStart = sec_to_tick(1124.960us)

# Sanal Bağlantı 5'in zaman-tetikli konfigürasyonu
**.switch3.vl_5_ctc.receive_window_start = sec_to_tick(1120.120us)
**.switch3.vl_5_ctc.receive_window_end = sec_to_tick(1140.120us)
**.switch3.vl_5_ctc.permanence_pit = sec_to_tick(1140.120us)
**.switch3.vl_5.sendWindowStart = sec_to_tick(1149.960us)
```

Şekil 5.17. TT trafikteki anahtar 3'ün (Switch3) NED dilindeki kaynak kodu [13]

Aşağıda TT trafiği alan ES6 uç sisteminin modeli gösterilmiştir. Kırmızı renkli modüller, CORE4INET kütüphanesinde tanımlanan TTBuffer modülleridir.

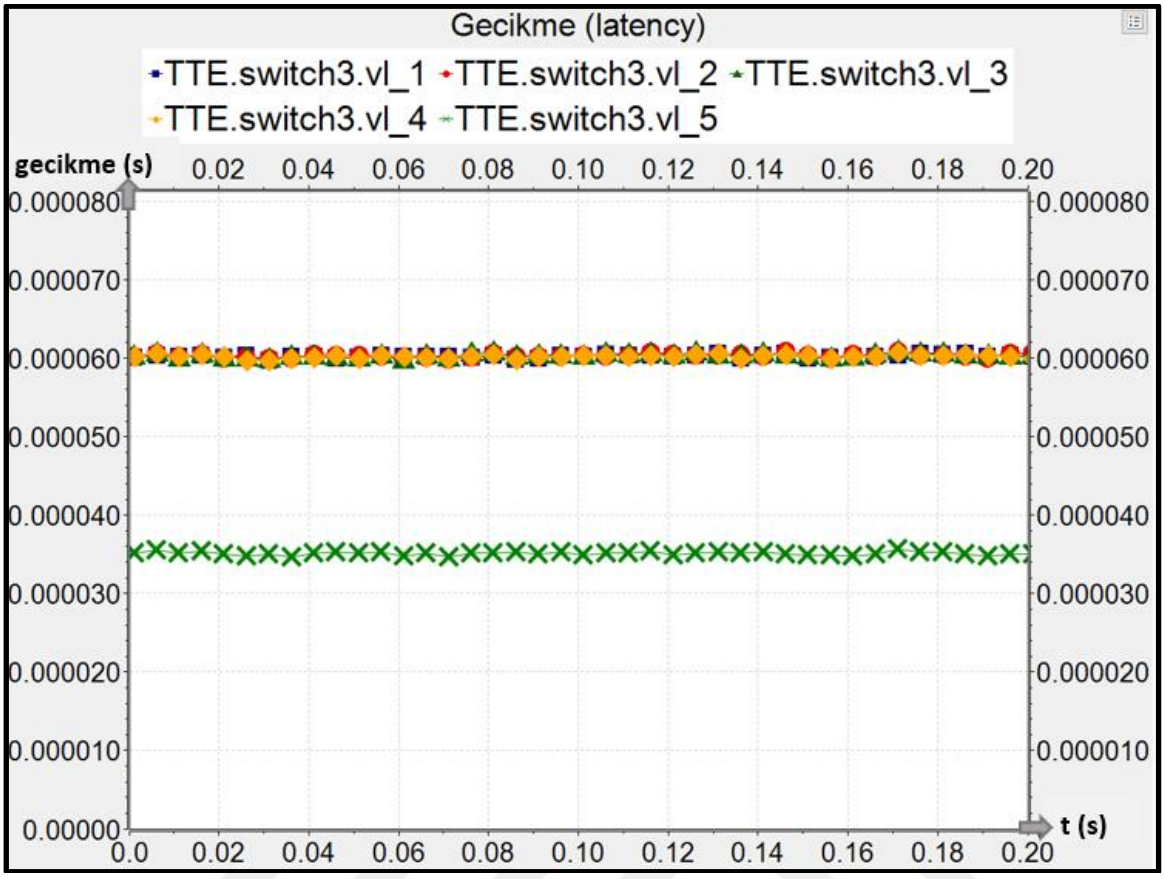


Şekil 5.18. TT trafikteki uç sistem 6'nın (ES6) modeli [13]

Yukarıdaki şekilde modellenen ES6 uç sisteminin konfigürasyonu aşağıdaki gibidir.

```
[General]
network = TTE

# ES6'nın MAC Adresi
**.ES6.phy[*].mac.address = "0A-00-00-00-00-06"
# RC trafiği oluşturan ES6'in gerçek zamanlı
# çalışan uygulama sayısını tanımlar.
**.ES6.numApps = 4

# ES6'ya gelen sanal bağlantıları tanımlar.
**.ES6.app[0].typename = "CTTrafficSinkApp"
**.ES6.app[0].displayName = "vl_1"
**.ES6.vl_1.destination_gates = "app[0].TTin"
**.ES6.vl_1.ct_id = 2

**.ES6.app[1].typename = "CTTrafficSinkApp"
**.ES6.app[1].displayName = "vl_3"
**.ES6.vl_3.destination_gates = "app[1].TTin"
**.ES6.vl_3.ct_id = 4

**.ES6.app[2].typename = "CTTrafficSinkApp"
**.ES6.app[2].displayName = "vl_4"
**.ES6.vl_4.destination_gates = "app[2].TTin"
**.ES6.vl_4.ct_id = 5

**.ES6.app[3].typename = "CTTrafficSinkApp"
**.ES6.app[3].displayName = "vl_5"
**.ES6.vl_5.destination_gates = "app[3].TTin"
**.ES6.vl_5.ct_id = 1

# Sanal Bağlantı 1'in zaman-tetikli konfigürasyonu
**.ES6.vl_1_ctc.receive_window_start = sec_to_tick(1045.120us)
**.ES6.vl_1_ctc.receive_window_end = sec_to_tick(1065.120us)
**.ES6.vl_1_ctc.permanence_pit = sec_to_tick(1065.120us)
**.ES6.vl_1.sendWindowStart = sec_to_tick(1074.960us)

# Sanal Bağlantı 3'ün zaman-tetikli konfigürasyonu
**.ES6.vl_3_ctc.receive_window_start = sec_to_tick(1095.120us)
**.ES6.vl_3_ctc.receive_window_end = sec_to_tick(1115.120us)
**.ES6.vl_3_ctc.permanence_pit = sec_to_tick(1115.120us)
**.ES6.vl_3.sendWindowStart = sec_to_tick(1124.960us)

# Sanal Bağlantı 4'ün zaman-tetikli konfigürasyonu
**.ES6.vl_4_ctc.receive_window_start = sec_to_tick(1120.120us)
**.ES6.vl_4_ctc.receive_window_end = sec_to_tick(1140.120us)
**.ES6.vl_4_ctc.permanence_pit = sec_to_tick(1140.120us)
**.ES6.vl_4.sendWindowStart = sec_to_tick(1149.960us)

# Sanal Bağlantı 5'in zaman-tetikli konfigürasyonu
**.ES6.vl_5_ctc.receive_window_start = sec_to_tick(1145.120us)
**.ES6.vl_5_ctc.receive_window_end = sec_to_tick(1165.120us)
**.ES6.vl_5_ctc.permanence_pit = sec_to_tick(1165.120us)
**.ES6.vl_5.sendWindowStart = sec_to_tick(1174.960us)

# ES6'ya gelen sanal bağlantıların ES6'nın phy[0]
# portuna geldiğini gösterir..
**.ES6.phy[0].inControl.ct_incomings = "vl_1_ctc, vl_3_ctc, vl_4_ctc, vl_5_ctc"
```

Şekil 5.19. TT trafikteki uç sistem 6'nın (ES6) NED dilindeki kaynak kodu [13]



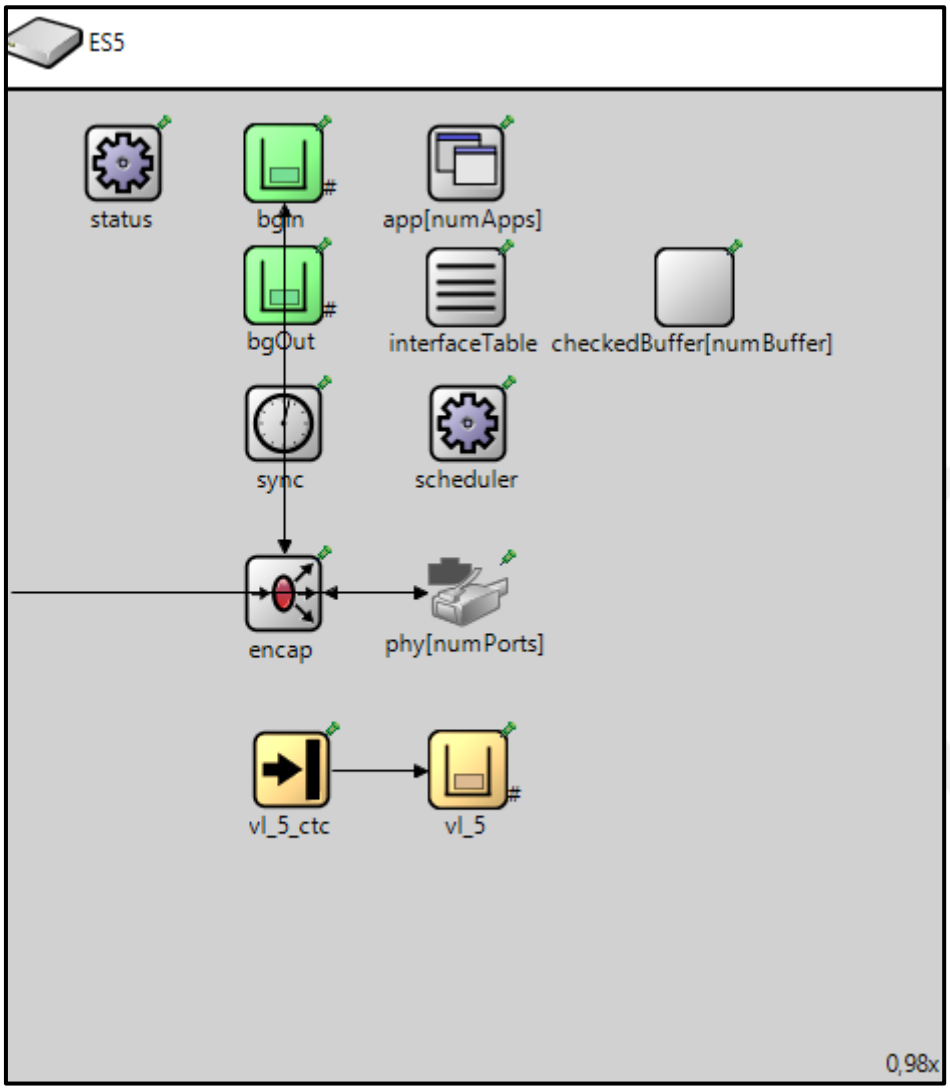
Şekil 5.20. TT trafik ağ simülasyonu sonucunda ölçülen iletim gecikmeleri [10]

TT trafiği simülasyonu sonucu gösteriyor ki, AFDX simülasyonunda olduğu gibi vl_5 sanal bağlantısının trafik gecikmesi en az 350 mili saniye çıkmaktadır. Sonuçlara göre, eğer TTEthernet tabanlı aviyonik ağı tamamen TT trafik olarak konfigüre edilirse AFDX tabanlı aviyonik ağı ile benzer şekilde trafik gecikmesine sahip olacaktır.

5.5. TT ve RC Trafiğe Sahip Senaryo

TTEthernet protokolü zaman-tetiklemeli trafik olan TT trafiği AFDX protokolünün aynısı olan bant-sınırlı RC trafiği aynı anda kullanabilmektedir. Bu senaryoda TT ve RC trafiğin aynı anda çalışacak şekilde modellenmesi konfigüre edilmesinden bahsedilmiştir ve simülasyon sonuçları analiz edilmiştir. Bir önceki bölümlerde gösterildiği üzere bu bölümde de vl_5 bağlantısı üzerinden örnek verilmiştir. ES2 uç biriminden çıkan vl_3 ve vl_4 sanal bağlantıları ile ES5 uç biriminden çıkan vl_5 bağlantısı RC trafik olarak, ES1 uç biriminden çıkan vl_1 ve vl_2 sanal bağlantısı da TT trafik olarak konfigüre edilmiştir.

ES5 uçbirimi modeli aşağıdaki gibidir.



Şekil 5.21. TT + RC trafikteki uç sistem 5'in (ES5) modeli [13]

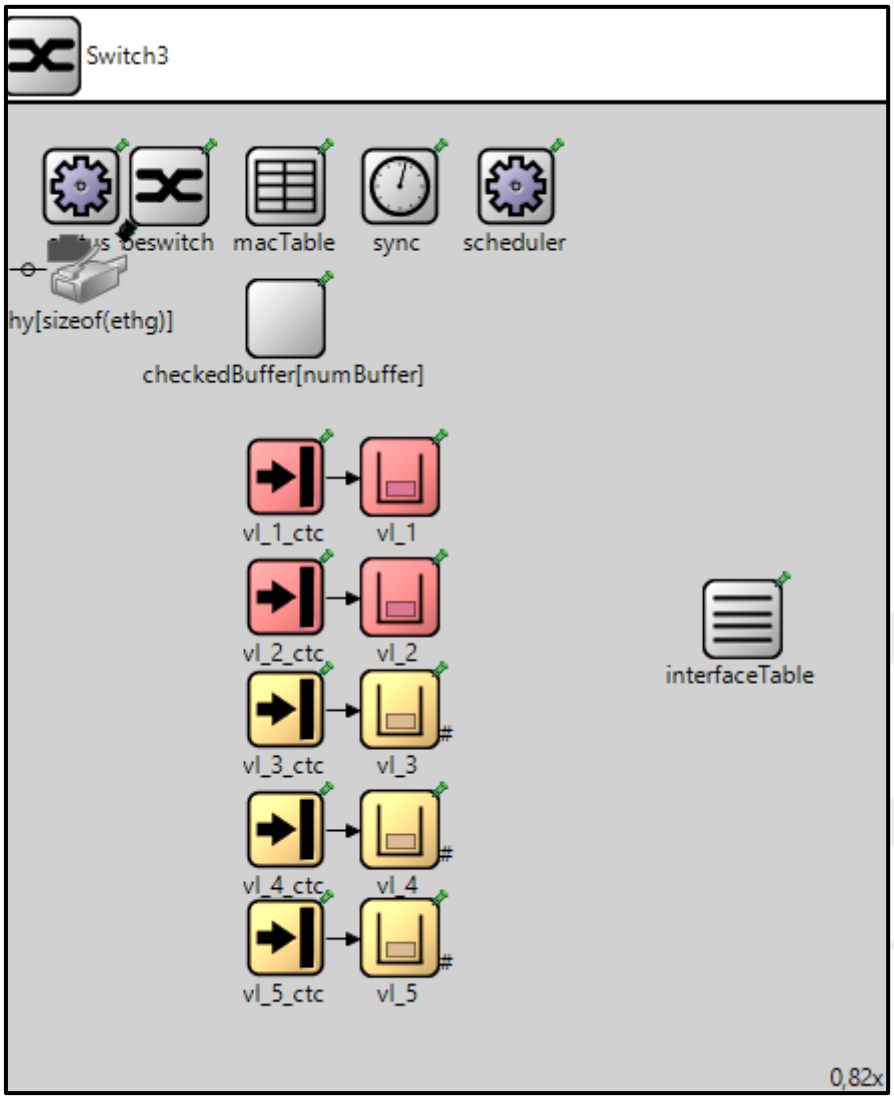
RC trafik kullanan ES5 uç biriminin konfigürasyonunda yine RCBuffer modülleri kullanılmıştır.

```
[General]
network = TTE

# ES5'in MAC Adresi
**.ES5.phy[*].mac.address = "0A-00-00-00-00-05"
# RC trafiği oluşturan ES5'in gerçek zamanlı
# çalışan uygulama sayısını tanımlar.
**.ES5.numApps = 1
# ES5'ten çıkan vl_5 isimili sanal bağlantıyı tanımlar.
**.ES5.app[0].buffers = "vl_5"
# RC trafiği oluşturan ES5'in gerçek zamanlı çalışan
# uygulamasını tanımlar.
**.ES5.app[0].typename = "RCTrafficSourceApp"
**.ES5.app[0].displayName = "vl_5"
**.ES5.app[0].interval = 5ms
**.ES5.app[0].payload = 46Byte

# Band-Sınırlı Trafik No (Sanal Bağlantı Numarası) = 1
**.ES5.app[0].ct_id = 1
# vl5'in çıkışını ilgii ES5'in fiziksel portu phy[0]'a bağlar.
**.ES5.vl_5.destination_gates = "phy[0].RCin"
# BAG değerini tanımlar.
**.ES5.vl_5.bag = sec_to_tick(4ms)
**.ES5.vl_5.priority = 0
**.ES5.vl_5.ct_id = 1
```

Şekil 5.22. TT + RC trafikteki uç sistem 5'in (ES5) NED dilindeki kaynak kodu [13]

ES5 uç sisteminin konfigürasyonu yukarıda tanımlanmıştır. ES5 uç sisteminden vl_5 sanal bağlantısı çıkmaktadır ve switch3 anahtarına girdikten sonra ES6 uç sistemine girmektedir.

Aşağıda RC ve TT trafiği kullanan switch3 anahtarının modeli verilmiştir.



Şekil 5.23. TT + RC trafikteki anahtar 3'ün (Switch3) modeli [13]

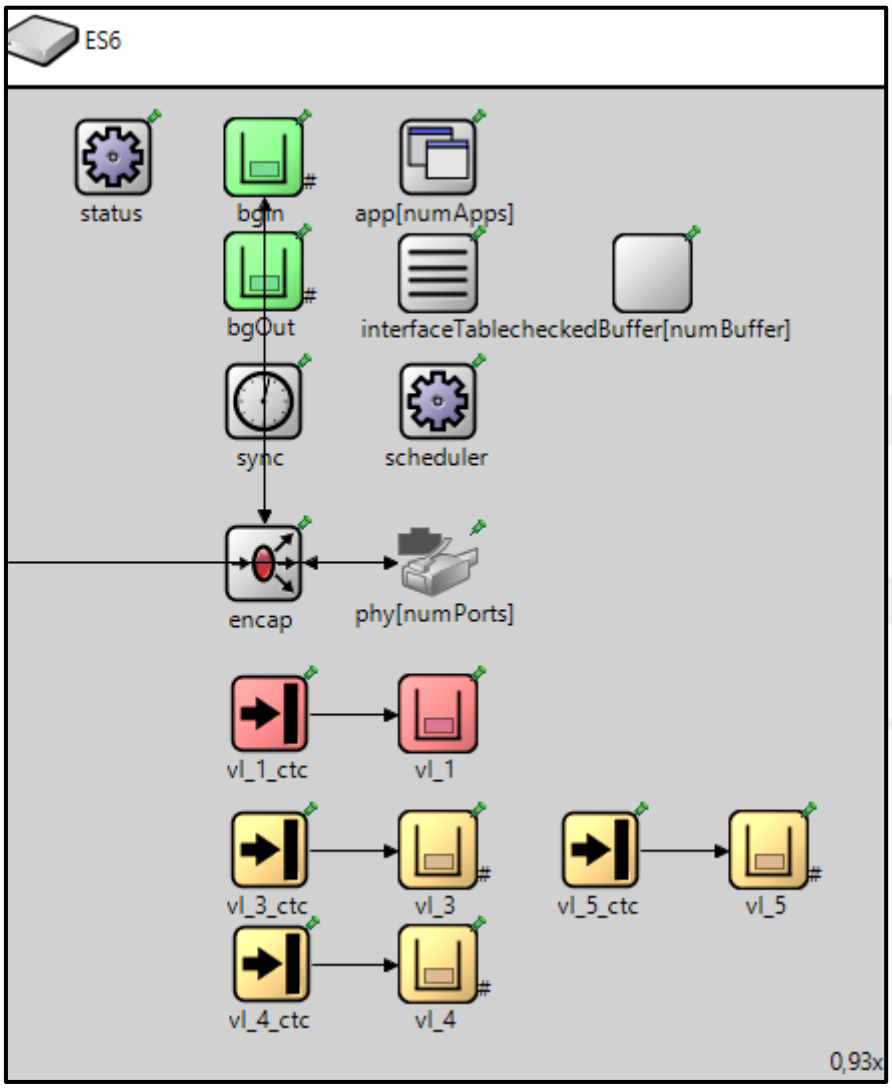
Switch3 anahtarının konfigürasyonu ise aşağıda verilmiştir. Burada, sarı ile boyalı modüller yukarıda kaynak kodu verilen RCBuffer modülüne ait ve kırmızı ile boyalı olan modüller de TTBuffer modülüne aittir.

```
[General]
network = TTE

# Sanal bağlantıların Anahtarın hangi portlarına
# gittiklerini ve hangi porttan geldiklerini tanımlar.

# VL1, VL2 => PHY0
**.switch3.phy[0].inControl.ct_incomings = "vl_1_ctc, vl_2_ctc"
# VL3, VL4 => PHY1
**.switch3.phy[1].inControl.ct_incomings = "vl_3_ctc, vl_4_ctc"
# VL5 => PHY2
**.switch3.phy[2].inControl.ct_incomings = "vl_5_ctc"

# PHY[3] => VL2 => ES7
**.switch3.phy[3].shaper.tt_buffers = "vl_2"
**.switch3.vl_2.destination_gates = "phy[3].TTin"
**.switch3.vl_2.ct_id = 3
# PHY[4] => VL1, VL3, VL4, VL5 => ES6
**.switch3.phy[4].shaper.tt_buffers = "vl_1"
**.switch3.vl_1.destination_gates = "phy[4].RCin"
**.switch3.vl_1.ct_id = 2
**.switch3.vl_3.destination_gates = "phy[4].RCin"
**.switch3.vl_3.ct_id = 4
**.switch3.vl_4.destination_gates = "phy[4].RCin"
**.switch3.vl_4.ct_id = 5
**.switch3.vl_5.destination_gates = "phy[4].RCin"
**.switch3.vl_5.ct_id = 1

# Sanal Bağlantı 1'in zaman-tetikli konfigürasyonu
**.switch3.vl_1_ctc.receive_window_start = sec_to_tick(1020.120us)
**.switch3.vl_1_ctc.receive_window_end = sec_to_tick(1040.120us)
**.switch3.vl_1_ctc.permanence_pit = sec_to_tick(1040.120us)
**.switch3.vl_1.sendWindowStart = sec_to_tick(1049.960us)

# Sanal Bağlantı 2'in zaman-tetikli konfigürasyonu
**.switch3.vl_2_ctc.receive_window_start = sec_to_tick(1045.120us)
**.switch3.vl_2_ctc.receive_window_end = sec_to_tick(1065.120us)
**.switch3.vl_2_ctc.permanence_pit = sec_to_tick(1065.120us)
**.switch3.vl_2.sendWindowStart = sec_to_tick(1074.960us)

# RC Trafik sanal bağlantıları BAG değerleri tanımlanmıştır.
**.switch3.vl_3_ctc.bag = sec_to_tick(4ms)
**.switch3.vl_3.bag = sec_to_tick(4ms)
**.switch3.vl_4_ctc.bag = sec_to_tick(4ms)
**.switch3.vl_4.bag = sec_to_tick(4ms)
**.switch3.vl_5_ctc.bag = sec_to_tick(4ms)
**.switch3.vl_5.bag = sec_to_tick(4ms)
```

Şekil 5.24. TT + RC trafikteki anahtar 3'ün (Switch3) NED dilindeki kaynak kodu [13]

Ağdaki anahtarları NED dilinde kodlarken dikkat edilmesi gereken en önemli husus sanal linklerin anahtarların hangi portlarına girdi olarak geldiği ve hangi portlarından çıktı olarak çıktığıdır.

vl_5 sanal bağlantısının son uğradığı yer ES6 uç sistemidir. vl_5 sanal bağlantısı RC trafik ilettiği için ES6'nın da RC trafik alabilecek şekilde modellenmesi ve konfigüre edilmesi gerekmektedir. Aşağıda ES6 uç sisteminin modeli mevcuttur.

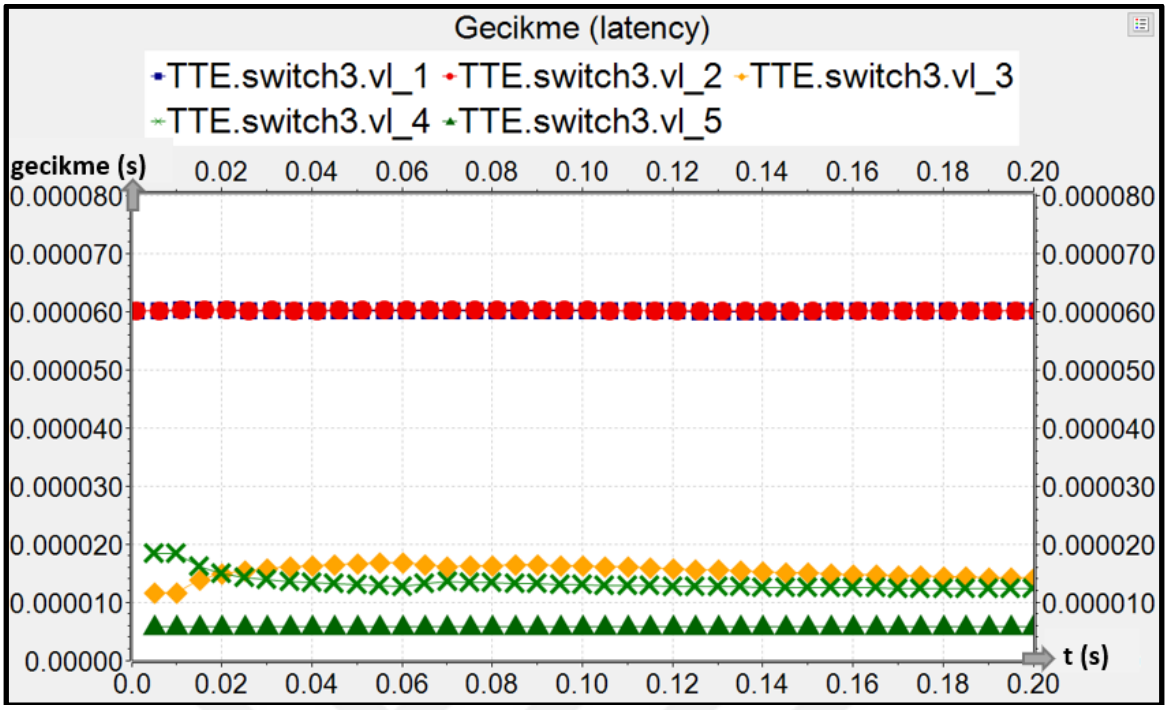


Şekil 5.25. TT + RC trafikteki uç sistem 6'nın (ES6) modeli [13]

```
[General]
network = TTE

# ES6'nın MAC Adresi
**.ES6.phy[*].mac.address = "0A-00-00-00-00-06"
# RC trafiği oluşturan ES6'in gerçek zamanlı
# çalışan uygulama sayısını tanımlar.
**.ES6.numApps = 4

# ES6'ya gelen sanal bağlantıları tanımlar.
**.ES6.app[0].typename = "CTTrafficSinkApp"
**.ES6.app[0].displayName = "vl_1"
**.ES6.vl_1.destination_gates = "app[0].TTin"
**.ES6.vl_1.ct_id = 2

**.ES6.app[1].typename = "CTTrafficSinkApp"
**.ES6.app[1].displayName = "vl_3"
**.ES6.vl_3_ctc.bag = sec_to_tick(4ms)
**.ES6.vl_3.destination_gates = "app[1].RCin"
**.ES6.vl_3.ct_id = 4

**.ES6.app[2].typename = "CTTrafficSinkApp"
**.ES6.app[2].displayName = "vl_4"
**.ES6.vl_4_ctc.bag = sec_to_tick(4ms)
**.ES6.vl_4.destination_gates = "app[2].RCin"
**.ES6.vl_4.ct_id = 5

**.ES6.app[3].typename = "CTTrafficSinkApp"
**.ES6.app[3].displayName = "vl_5"
**.ES6.vl_5_ctc.bag = sec_to_tick(4ms)
**.ES6.vl_5.destination_gates = "app[3].RCin"
**.ES6.vl_5.ct_id = 1

# Sanal Bağlantı 1'in zaman-tetikli konfigürasyonu
**.ES6.vl_1_ctc.receive_window_start = sec_to_tick(1045.120us)
**.ES6.vl_1_ctc.receive_window_end = sec_to_tick(1065.120us)
**.ES6.vl_1_ctc.permanence_pit = sec_to_tick(1065.120us)
**.ES6.vl_1.sendWindowStart = sec_to_tick(1074.960us)


# ES6'ya gelen sanal bağlantıların ES6'nın phy[0]
# portuna geldiğini gösterir..
**.ES6.phy[0].inControl.ct_incomings = "vl_1_ctc, vl_3_ctc, vl_4_ctc, vl_5_ctc"
```

Şekil 5.26. TT + RC trafikteki uç sistem 6'nın (ES6) NED dilindeki kaynak kodu [13]

RC ve TT trafiği aynı anda ihtiva eden aviyonik ağ senaryosunun simülasyon sonucunda sanal bağlantılardaki gecikme aşağıdaki grafikte verilmiştir.



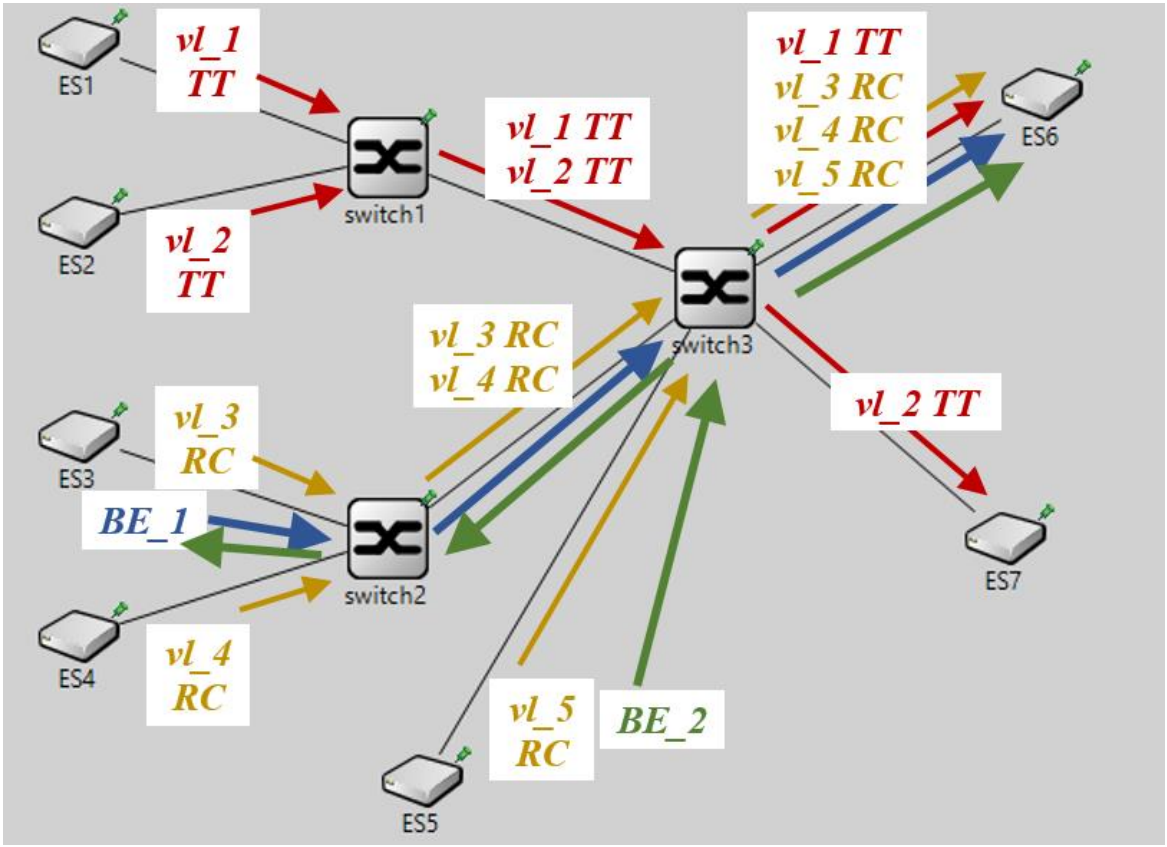
Şekil 5.27. TT + RC trafik ağ simülasyonu sonucunda ölçülen iletim gecikmeleri [10]

Yukarıdaki grafikten de anlaşılacağı üzere TT trafik gecikmesini gösteren vl_1 ve vl_2 sanal bağlantılarındaki gecikme sabittir (60 mikro saniye). Grafiğin alt kısmındaki RC trafiğe ait olan vl_3, vl_4 ve vl_5 sanal bağlantılarındaki gecikmeler de sabittir; ancak TT trafik gecikmesi ile karşılaştırılınca birkaç mikro saniye daha fazla seğirmeye (jitter) maruz kaldıkları anlaşılmaktadır. Aviyonik perspektifinden bakıldığında, bu seviyede mikro saniye mertebelerindeki seğirmeler aviyonik ağlarındaki belirlenirlik (deterministik) özelliğini olumsuz etkilememektedir. Ancak, daha sabit bir gecikme süresine sahip olan TT trafik kullanan TTEthernet ağının, AFDX ağına nazaran daha çok tercih edilmesi belirlenirlik açısından daha uygundur.

5.6. TT, RC ve BE Trafiğe Sahip Senaryo

Bu bölümde, bant-sınırlı AFDX trafiği olan RC trafik, zaman-tetiklemeli TT trafik ve en iyi çaba BE trafik aynı anda koşturularak aviyonik ağdaki performans analiz edilmiştir. En iyi çaba BE trafik, IEEE 802.3 Ethernet trafiğinin aynısıdır ve TTEthernet, zaman-kritik olmayan uygulamalarda BE trafiğinin de kullanılabilmesi üzerine çalışmaktadır.

Bu senaryoda, bir önceki TT + RC trafik senaryosuna ek olarak BE trafik eklenmiştir. Ağ topolojisi aşağıdaki şekilde gösterilmiştir. Yeşil oklar BE trafiği, sarı oklar RC trafiği ve kırmızı oklar TT trafiği göstermektedir.



Şekil 5.28. TT + RC + BE trafiğe sahip TTEthernet ağı şekilsel gösterimi [13]

BE trafik ES3 uç birimi, ES6 birimine göndermek üzere BE trafik üretmektedir (BE_1). ES5 uç birimi ise hem ES3 uç birimine hem de ES6 uç birimine BE trafik iletmektedir (BE_2). Bu senaryoya, uç sistemlerin ve anahtarların modellerini değiştirmeksizin sadece uç sistemlerin konfigürasyonlarını güncelleyerek ulaşmak mümkündür. Aşağıda, ilgili uç sistemlere uygulanan ek konfigürasyonlar gösterilmiştir.

ES3 Uç Sistemine Eklenen BE Konfigürasyonu

```
[Config With_Crosstraffic]
**.ES3.numApps = 3
# En-İyi-Çaba BE trafiğini tanımlar
**.ES3.app[1].typename = "BGTrafficSinkApp"
# BE_2'nin kaynak MAC adresi = ES5'in MAC adresi
**.ES3.app[1].srcAddress = "0A-00-00-00-00-05"
**.ES3.bgIn.destination_gates = "app[1].in"

# BE_1'in hedef MAC adresi = ES6'nın MAC adresi
**.ES3.app[2].typename = "BGTrafficSourceApp"
**.ES3.app[2].destAddress = "0A-00-00-00-00-06"
**.ES3.app[2].payload = intWithUnit(uniform(1500Byte, 1500Byte))
**.ES3.app[2].sendInterval = uniform(200us,500us)
```

ES5 Uç Sistemine Eklenen BE Konfigürasyonu

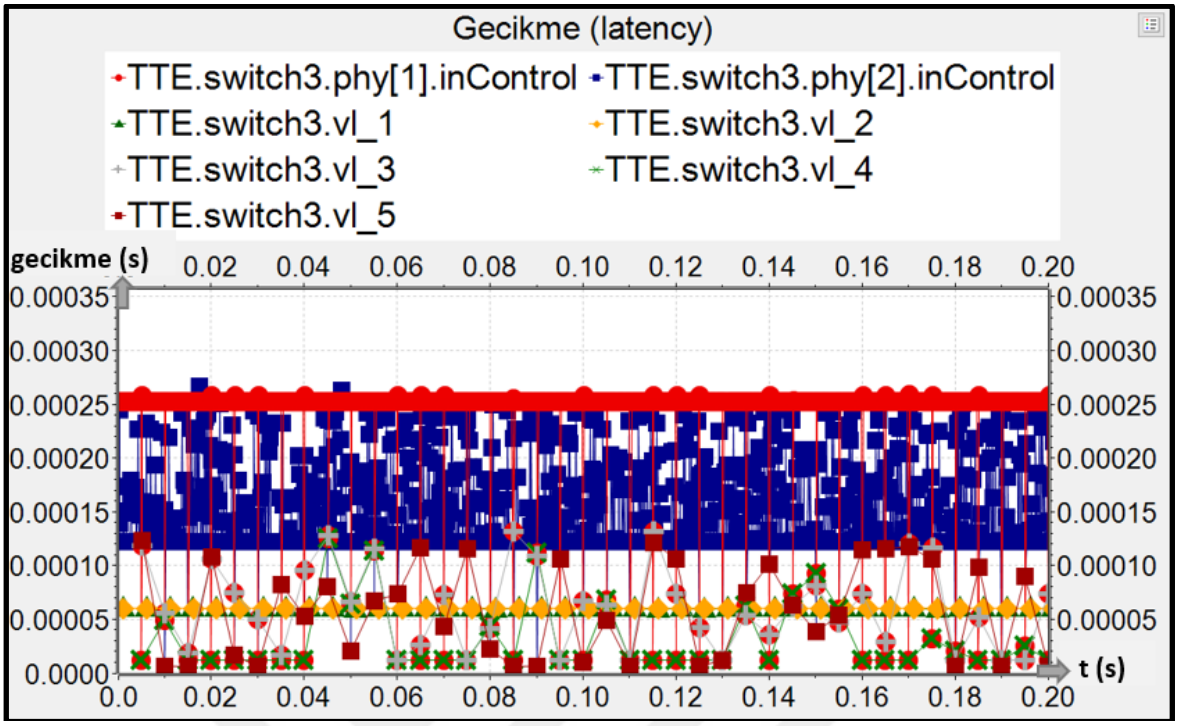
```
[Config With_Crosstraffic]
**.ES5.numApps = 3
# En-İyi-Çaba BE trafiğini tanımlar
**.ES5.app[1].typename = "BGTrafficSourceApp"
# BE_2'nin hedef MAC adresi = ES3'ün MAC adresi
**.ES5.app[1].destAddress = "0A-00-00-00-00-03"
**.ES5.app[1].payload = intWithUnit(uniform(1500Byte, 1500Byte))
**.ES5.app[1].sendInterval = uniform(200us,500us)

**.ES5.app[2].typename = "BGTrafficSourceApp"
# BE_1'in hedef MAC adresi = ES6'nın MAC adresi
**.ES5.app[2].destAddress = "0A-00-00-00-00-06"
**.ES5.app[2].payload = intWithUnit(uniform(1500Byte, 1500Byte))
**.ES5.app[2].sendInterval = uniform(200us,500us)
```

ES6 Uç Sistemine Eklenen BE Konfigürasyonu

```
[Config With_Crosstraffic]
**.ES6.numApps = 6
# En-İyi-Çaba BE trafiğini tanımlar
**.ES6.app[4].typename = "BGTrafficSinkApp"
# BE_1'in kaynak MAC adresi = ES3'ün MAC adresi
**.ES6.app[4].srcAddress = "0A-00-00-00-00-03"

**.ES6.app[5].typename = "BGTrafficSinkApp"
# BE_2'nin kaynka MAC adresi = ES5'in MAC adresi
**.ES6.app[5].destAddress = "0A-00-00-00-00-05"


**.ES6.bgIn.destination_gates = "app[4].in, app[5].in"
```

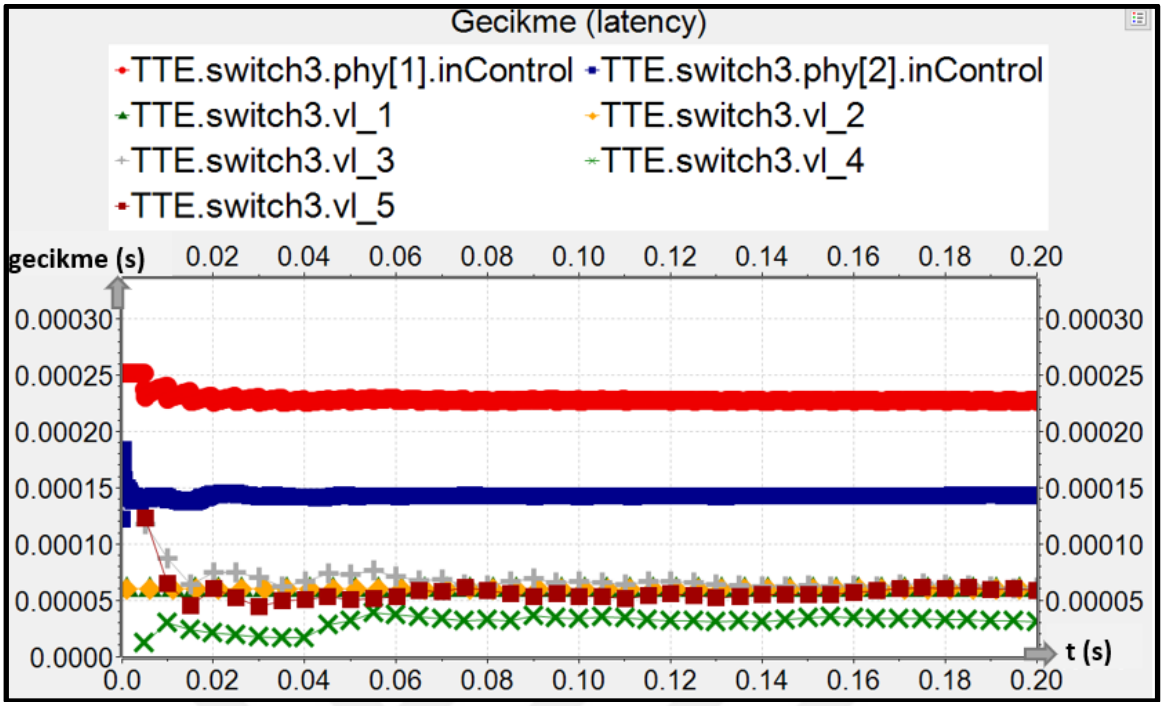
Şekil 5.29. TT + RC trafik üzerine eklenen BE trafiğin NED dilindeki kaynak kodu [13]

Simülasyon sonucunda ilgili ağ senaryosundaki uçtan uca iletim gecikmesi analizi aşağıdaki gibidir.



Şekil 5.30. TT + RC + BE trafik ağ simülasyonu sonucunda ölçülen iletim gecikmeleri [10]

Bu şekilde görüldüğü üzere, veri iletim gecikmesinde en çok varyasyona sahip (jitter) olan hatlar kırmızı ve mavi ile gösterilen BE_1 ve BE_2 en-iyi-çaba BE trafiklerine aittir. BE_1 ve BE_2 trafiklerinin veril iletim gecikmeleri 250 mili saniye ve 120 mili saniye arasında değişerek maksimum 130 mili saniyelik bir seğirmeye neden olmaktadırlar. TT trafik simülasyonunda gösterdiğimiz üzere, TTEthernet standardı TT trafik için maksimum olarak mikro saniyeler mertebesinde seğirme garanti etmektedir. Ancak, TTEthernet standardı paketlerin iletilmesinde her zaman TT paketlere öncelik verir. İkinci öncelik bant-sınırlı RC trafikte ve sonrasında ise en-iyi-çaba BE trafiktedir. Yani, TT, RC ve BE trafiğin aynı anda kullanıldığı senaryolarda RC ve BE trafik için sabit bir gecikme söz konusu değildir. Bu sebepten TTEthernet mimarisi kullanılacak uçaklarda, eğer aynı anda TT, RC ve BE trafik oluşturulacaksa RC ve BE trafik uygulamaları zaman-kritik uygulamalar olmamalıdır. Aksi takdirde aviyonik endüstrisinin isteri olan deterministik davranışa sahip olunamayacaktır. İletim gecikmelerindeki bu durumun daha rahat anlaşılabilmesi için yukarıdaki grafiğin ortalaması aşağıdaki gibi alınmıştır.



Şekil 5.31. TT + RC + BE trafik ağ simülasyonu sonucunda ölçülen ortalama iletim gecikmeleri [10]

Ortalaması alınmış grafikte görüldüğü üzere TT trafik taşıyan vl_1 ve vl_2 sanal bağlantılarındaki gecikme her zaman 50 mikro saniye mertebesinde sabit kalmıştır. Ancak, RC trafik taşıyan vl_3, vl_4 ve vl_5 sanal bağlantılarındaki gecikme 150 ve 20 mikro saniye arasında salınım yaptığı için TT trafik kadar başarılı bir sonuç vermemektedir. İletim gecikmelerindeki bu değişimlerden ötürü TT, RC ve BE trafiğin aynı anda kullanılması durumunda RC ve BE trafikte zaman-kritik verilerin taşınmaması gerekmektedir. Zaman-kritik veriler TT trafikte taşınarak RC ve BE trafiğe zaman-kritik olmayan uygulamaların verileri yönlendirilerek ağdaki bant genişliği verimli bir şekilde kullanılmış olmaktadır.

6. SONUÇ VE ÖNERİLER

Bu tez çalışmasında, 1970'lerden bu yana uçaklarda sıklıkla kullanılan MIL-STD-1553, ARINC-429 gibi gelenekselleşmiş aviyonik haberleşme ağlarının özelliklerinden ve bunların günümüzdeki uçakların işlevsel özelliklerini karşılamada niçin yetersiz kaldıklarından bahsedilmiştir. Aviyonik sistemlerde desteklenmesi istenen fonksiyon sayısı günden güne artmaktadır. Bununla beraber, gelişen görüntüleme tekniklerinin uçaklara entegre edilmesi ile uçakların da artık yüksek çözünürlüklü video kaydetmesi ve bunları işlemesi istenmektedir. Bütün bunlar, aviyonik haberleşme ağlarının daha yüksek bant genişliklerinde çalışması ihtiyacını doğurmaktadır. Bu probleme çözüm olarak 1990'lı yıllarda, yüksek bant genişliği sunduğu için, Ethernet teknolojisinin aviyonik endüstrisinde kullanılması düşüncesi ortaya atılmıştır.

Deterministik (belirleyici) olmayan IEEE 802.3 Ethernet teknolojisinde gönderilen paketlerin alıcıya ulaşma süreleri her zaman önceden hesap edilememektedir. Bu yüzden Ethernet protokolü zaman-kritik olan aviyonik uygulamaların ihtiyaçlarını karşılayamaz, Ethernet deterministik değildir. Ayrıca paket çarpışmalarının ve kayıplarının yaşanmasından ötürü Ethernet protokolü ile gönderilen her paket alıcıya ulaşmayabilmektedir, bu yüzden Ethernet güvenilir değildir. Bunlara çözüm olarak 2000'li yıllarda Airbus firmasın yaptığı AR-GE'ler neticesinde, ARINC-664 Part 7'nin özel bir uygulaması olarak standartlaştırılan, Aviyonik Tam Çift Yönlü Anahtarlamalı Ethernet (AFDX) protokolü geliştirilmiştir. 100 Mbps hızlarına çıkabilen AFDX protokolü, ağda izin verilen maksimum jitter'ın bir üst sınırını garanti eder, bu da sonuç olarak uçtan uca gecikmeyi sınırlar ve bunu bilinir hale getirir. Böylece, vericiden giden paketlerin alıcıya en geç ne zaman varacaklarının üst sınır bilindiği için AFDX ağları deterministiktir. Öte yandan 2000'li yıllarda Viyana Teknik Üniversitesi ve TTTech firmasın geliştirdiği ve AS6802 olarak standartlaştırılan, 1 Gbps hızlarına çıkabilen Zaman Tetiklemeli Ethernet (TTEthernet), mikro saniye mertebesinde minimum jitter sunarak sabit bir uçtan-uca gecikme tanımladığından kesinlikle deterministiktir. TTEthenet, sunduğu Zaman Tetiklemeli (TT) trafik ile mikrosaniye jitter mertebesinde deterministiktir, Bant-Kısıtlı (RC) trafik ile AFDX protokolünü de içine alabilmektedir ve AFDX protokolünü tam olarak taklit edebilir. Ayrıca, TT ve/veya RC trafikten arta kalan bant genişliğini de IEEE 802.3 standart Ethernet'i kullanarak değerlendirebilir. Böylelikle, TTEthernet, AFDX'e

nazaran çok daha kullanışlı, çok daha hızlı ve çok daha deterministik olduğu için gelecek nesil uçaklarda tercih edilmelidir.

Bu tez çalışmasında, AFDX ve TTEthernet protokolü OMNET++ ağ simülasyon programında modellenip simüle edilerek bu iki protokolün aynı ağ topolojisindeki trafik gecikmeleri analiz edilmiştir. Böylelikle, aviyonik endüstrisinin en çok önem verdiği ağ özelliği olan deterministik özelliği açısından bu iki protokol karşılaştırmalı olarak analiz edilmiştir.

Bu tez çalışmasındaki simülasyonlar, TTEthernet ağlarındaki TT trafiğinin tam olarak deterministik işlediğini göstermektedir. TTEthernet ağlarını yalnızca RC trafiğini kullanılacak şekilde konfigüre edildiğinde, ağ AFDX olarak kullanılmaktadır. RC trafiğin çalıştığı deney senaryosunda da görülmüştür ki, AFDX ağı da deterministiktir. Bununla birlikte, TT trafiğinin üzerine RC trafiği eklemek, TT trafik gecikmesini etkilemeden RC trafik gecikmesini olumsuz etkileyebilmektedir. Dahası, TTEthernet mimarisi TT, RC ve BE trafiğini aynı anda kullanabilmekte, ancak RC ve BE trafiklerinin gecikmelerinin düzeltilmesi garanti edilmemektedir. Bu nedenle, hangi ağların kullanılacağına karar verirken hizmet gereksinimlerinin kalitesi konusunda dikkatli olunmalı ve bant genişliği kullanılabilirliği ile zamansal duyarlılık arasındaki olumlu ve olumsuz yanlar göz önünde bulundurulmalıdır. Bu nedenle, AFDX veya MIL-STD-1553, ARINC-429 gibi klasik protokoller yerine TTEthernet'i seçmek, yeni nesil uçaklarda aviyonik platform gereksinimleri için daha uygun bir çözüm olacaktır.

KAYNAKLAR

1. İnternet: NASA, Comparison of Communication Architectures for Spacecraft Modular Avionics Systems. URL: http://spacewire.esa.int/WG/SpaceWire/SpW-SnP-WG-Mtg7-Proceedings/Reference%20Documents/NASA_TM_06_214431.pdf, Son Erişim Tarihi: 04.05.2021.

2. Gunnarsson, D. (2006). *Safety-Critical Communicationin Avionics*. Master Thesis, Lund Institute of Technology, Lund, İsveç, 245-246.

3. Safwat, N. (2014). *Modeling and simulation of aircraft data networks*. Master Thesis, Ain Shams University, Kahire, Mısır, 23-24.

4. İnternet: *OPNET*. URL: https://support.riverbed.com/content/support/software/opnet-model/modeler.html/. Son Erişim Tarihi: 20.02.2021

5. Bauer, H. Scharbarg, J. and Fraboul, C. (2009). Applying and optimizing trajectory approach for performance evaluation of AFDX avionics network. *2009 IEEE Conference on Emerging Technologies and Factory Automation*, Palma de Mallorca, Spain, 1-8.

6. İnternet: *UPPAAL*. URL: https://uppaal.org/. Son Erişim Tarihi: 20.02.2021

7. Efe, O. (2016). *Tool support for worst case end to end delay analysis of AFDX networks*. Master Thesis, Middle East Technical University, Ankara, 25-27.

8. Guo, L., Wan, B., Li, C., Zhou, K. and Li, X. (2018, July). *A flow-grained end-to-end delay analysis for RC traffic in TTEthernet. In 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, Tokyo, 904-913.

9. Hamid, O. and Vaqar, S. A.. (2020). *A comparison of Distributed Data Communications using Ethernet in Aircraft*, [M.S. - Master Thesis]. Owais Hamid Systems Design Engineering University, of Waterloo, 33-35.

10. İnternet: *OMNET*, URL: https://www.omnetpp.org/. Son Erişim Tarihi: 20.02.2021

11. Yang, Q., Lu, H., and Tu, X. (2020, November). Simulation and experiment of AFDX network based on OMNeT++. In *2020 Chinese Automation Congress (CAC)*, Shanghai, 5849-5854.

12. Zhao, L., He, F., Li, E. and Lu, J. (2018). Comparison of Time Sensitive Networking (TSN) and TTEthernet. In *IEEE/AIAA 37th Digital Avionics Systems Conference (DASC*), London.

13. İnternet: *CORE4INET*. URL: https://core4inet.core-rg.de/. Son Erişim Tarihi: 20.02.2021

14. Rejeb, N., Salem, A. K. B. and Saoud, S. B. (2017, January). AFDX simulation based on TTEthernet model under OMNeT++. In *2017 International Conference on Advanced Systems and Electric Technologies (IC_ASET)*, Hammamet, 423-429.

15. Kultur, Osman R., Bilge, Hasan S. (2021). "Comparative Analysis of Next Generation Aircraft Data Networks", *IEEE 19th International Conference on Smart Technologies* (EUROCON), Lviv, 22-36.

16. İnternet: Certification Considerations for Highly Integrated or Complex Aircraft Systems. URL: https://www.sae.org/standards/content/arp4754a/, Son Erişim Tarihi: 14.02.2021.

17. İnternet: Guidelines and methods for conduction the safety assessment process on civil airborne systems and equipment. URL: https://www.sae.org/standards/content/arp4761/, Son Erişim Tarihi: 05.03.2021.

18. İnternet: Software consideration in Airborne Systems and Equipment Certification. URL: https://my.rtca.org/NC__Product?id=a1B36000001IcmsEAC, Son Erişim Tarihi: 14.02.2021.

19. İnternet: Design Assurance Guidance for Airborne Electronic Hardware. URL: https://my.rtca.org/NC__Product?id=a1B36000001IcjTEAS, Son Erişim Tarihi: 14.02.2021.

20. İnternet: RTCA/DO-160D, Environmental Conditions and Test Procedures for Airborne Equipment, URL: https://my.rtca.org/NC__Product?id=a1B36000001IcnSEAS, Son Erişim Tarihi: 03.03.2021.

21. İnternet: Aeronautical Radio Incorporated. ARINC specification 429P3-19, Mark 33 digital information transfer system (DITS), Part 3 - File data transfer techniques, URL: https://global.ihs.com/doc_detail.cfm?document_name=ARINC%20429P3&item_s_key=00235028, Son Erişim Tarihi: 12.04.2021.

22. İnternet: United States Department of Defense. MIL-STD-1553B, military standard: aircraft internal digital time division command response multiplex data bus, URL: https://global.ihs.com/doc_detail.cfm?&item_s_key=00069743&item_key_date=811003, Son Erişim Tarihi: 17.04.2021.

23. Aeronautical Radio Incorporated. Aircraft Data Network. Standard 664, ARINC, Annapolis, URL: https://global.ihs.com/doc_detail.cfm?document_name=ARINC%20664%20P7&item_s_key=00465045, Son Erişim Tarihi: 02.04.2021.

24. İnternet: SAE - AS-2D Time Triggered Systems and Architecture Committee. Time-Triggered Ethernet (AS 6802), URL: https://global.ihs.com/doc_detail.cfm?document_name=SAE%20AS6802&item_s_key=00578029, Son Erişim Tarihi: 07.04.2021.

25. İnternet: Interview: Erv Gangl. URL: https://www.aviationtoday.com/2002/09/01/interview-erv-gangl/, Erişim Tarihi: 20.02.2021

26. Erdinç, E. (2010). *Soft afdx (avionics full duplex switced ethernet) end system implementation with standard pc and ethernet card.* Master Thesis, Middle East Technical University, Ankara, 43-45.

27. İnternet: Switched Ethernet Solutions for Embedded Real Time Networks. URL: http://etr2015.irisa.fr/images/presentations/Jean-Luc_Scharbarg_ETR2015.pdf, Son Erişim Tarihi: 22.03.2021.

28. İnternet: AFDX, From Component to Application, URL: https://www.slideshare.net/MENMikroElektronik/afdx-from-component-to-application, Son Erişim Tarihi: 20.02.2021.

29. İnternet: *INET*. URL: https://inet.omnetpp.org/. Erişim Tarihi: 20.02.2021

GAZİ GELECEKTİR..