BLM3051
VERİ İLETİŞİMİ VE BİLGİSAYAR AĞLARI
ÖDEV 1

Grup 7
Ebru Kılıç – 23011602
Rumeysa Nur Yasav – 22011059
Sude Elitok – 21011062
Andalib Goncharova – 22011935

Dr. Öğr. Üyesi Furkan Çakmak

Nisan 2025

Projenin çalıştırılabilmesi için gerekli exe dosyası C:\Users\Lenovo\Desktop\viba-20242-7\viba-20242-7\build\Desktop_x86_windows_msvc2019_pe_32bit-Release\release içerisinde viba-20242-7.exe ismiyle bulunmaktadır. dll dosyaları olmadan çalışmadığı için bu açıklamanın yapılmasına ihtiyaç duyulmuştur. Bilgilerinize sunarız.

Bu projede, OSI referans modeli 2.katmanı olan Data Link Layer'de gerçekleşen veri iletiminde oluşabilecek hataları simüle etmek amacıyla çerçeveleme (framing), CRC (Cyclic Redundancy Check), bit stuffing, checksum ve hata simülasyonu teknikleri uygulanmıştır. Alıcı (receiver) tarafında gönderilen verilerin doğruluğu kontrol edilmekte, hatalı paketler tespit edilerek yeniden iletim (retransmission) süreci tetiklenmektedir. Veri iletimi Stop&Wait protokolü ile gerçekleştirilmekte olup, tüm bu işlemler için grafiksel bir arayüz (GUI) üzerinden görsel ve takip edilebilir bir şekilde sunulmaktadır.

1. Dosya Seçme

Simülasyonun ilk adımı kullanıcının "Dosya Seç" butonu ile göndermek istediği .dat uzantılı dosyayı seçmesidir.

SelectFile() fonksiyonu sayesinde kullanıcıya dosya seçim penceresi açılır. Kullanıcı dosyayı seçtiğinde seçilen dosyanın yolu fileLabel adlı QLabel bileşenine yazılmakta ve böylelikle dosya yolu kullanıcıya ekran üzerinden gösterilmektedir.

2. Dosyanın İşlenmesi

Dosya seçiminin sonraki aşaması dosyaların işlenip frame'lere dönüştürülme sürecidir. "Dosyayı İşle" butonuna basıldığında öncelikle dosyanın uygunluğu kontrol edilmektedir. Eğer dosya boş değilse ve uygun formattaysa Sender sınıfında bulunan readFileAndCreateFrames() fonksiyonu ile işlenmektedir.

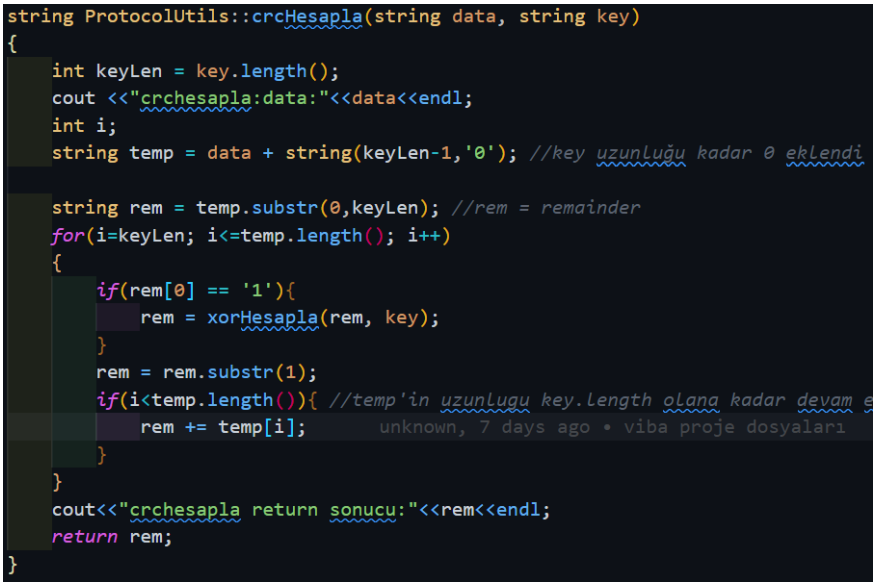
Dosya byte byte okunmaktadır. Okunan byte'lar 8 bitlik bir bit kümesine çevrilir, MSB'den LSB'ye doğru işlenerek bitStream adlı stringe eklenmektedir. Bütün dosya bu şekilde okunarak bütün karakterlerin bitlerini içeren bitStream stringi elde edilmektedir. bitStream bit stuffing yapılarak 100 bitlik framelere parçalanmakta, sığmayanlar bir sonraki frame'e aktarılmaktadır. Eğer son frame'de boşluk kalırsa 0 ile doldurulmaktadır. Her frame'in CRC kodu hesaplanmaktadır.

3. CRC (Cyclic Redundancy Check) ile Hata Tespiti:

Veriler, her biri 100 bit uzunluğundaki çerçevelere ayrıldıktan sonra x^16 + x^12 + x^5 + 1 polinomu (binary: 10001000000100001) kullanılarak veri iletiminde oluşabilecek bit düzeyindeki hataları tespit edebilmek için kullanılan CRC tekniği ile hata kodları işlenmiştir. CRC kodu göndericiden alıcıya gerçekleşen frame iletimi sırasında iletilmektedir. Bu iletim sırasında gönderilen bir frame'in yapısı Tablo 1'deki gibidir. Bu frame yapısındaki her bir bileşen string olarak tutulmaktadır ve alıcıya gönderilirken birbiri ardına eklenerek tek bir string halinde iletilmektedir. Flag olarak "01111110" kullanılmaktadır. Bunun tercih edilmesinin sebebi veri içerisinde ~ işaretine rastlanmaması ve HDLC gibi protokollerde de kullanılan bir flag olmasıdır.

Tablo 1 – Frame Yapısı

HEADER		DATA	TRAILER	
Adres (7 bit)	Frame No (1 bit)	Data (100 bit)	Flag (8 bit)	CRC Kodu (16 bit)



```
string ProtocolUtils::crcHesapla(string data, string key)
{
    int keyLen = key.length();
    cout <<"crchesapla:data:"<<data<<endl;
    int i;
    string temp = data + string(keyLen-1,'0'); //key uzunluğu kadar 0 eklendi

    string rem = temp.substr(0,keyLen); //rem = remainder
    for(i=keyLen; i<=temp.length(); i++)
    {
        if(rem[0] == '1'){
            rem = xorHesapla(rem, key);
        }
        rem = rem.substr(1);
        if(i<temp.length()){ //temp'in uzunlugu key.length olana kadar devam e
            rem += temp[i];       unknown, 7 days ago • viba proje dosyaları
        }
    }
    cout<<"crchesapla return sonucu:"<<rem<<endl;
    return rem;
}
```

Şekil 1 – CRC Hesaplama Kodu

Şekil 1'de kodu gösterilen CRC hesaplama fonksiyonunda, gönderilecek verinin sonuna key olarak belirtilmiş 17 bitlik (16. Dereceden) CRC polinomunun uzunluğunun bir eksiği kadar sıfır eklenmiştir. Daha sonra, bu veri her adımda kaydırılarak XOR işlemleri uygulanmış ve geriye kalan bit dizisi CRC değeri olarak belirlenmiştir. Bu işlem veri bütünlüğünün doğrulanmasında kullanılmıştır. Bu işlem sonucunda elde edilen CRC 16 bit boyutundadır.

4. Bit Stuffing ve Flag:

Veri iletişiminde, her bir frame'in içeriğinde adres, frame numarası, veri (data), flag ve CRC kodu yer almaktadır. Alıcı taraf, veri ve CRC kodunu birbirinden ayırmak zorundadır. Bu amaçla, frame içinde "01111110" olarak tanımlanan bir flag kullanılmaktadır. Ancak, eğer veri kısmında aynı bit sıralamasına sahip bir flag yer alırsa, alıcı taraf veri ile CRC kodunu ayırt edemez. Bu duruma data transparency denmektedir.

Bu sorunu çözmek için, veri içinde art arda altı tane "1" gelirse, beşinci "1"den sonra bir adet "0" eklenir. Bu sayede veri kısmında hiçbir zaman flag dizilimi oluşmaz ve alıcı taraf, veri ile CRC'yi doğru bir şekilde ayırabilir. Alıcı taraf, veri alırken bu ekstra eklenen "0" bitini tespit eder ve çıkarır. Böylece veri kaybı olmadan işlemler düzgün bir şekilde gerçekleştirilir.

5. Checksum Hesaplama ve Cheksum için Frame Yapısı:

Tüm frame'ler için oluşturulmuş CRC kodları toplanarak 8-bit'lik checksum değeri hesaplanmıştır. Bu sayede, verinin tamamının kontrolü amacıyla ek bir güvenlik katmanı oluşturulmuştur. Checksum hesaplamasında kullanılacak CRC kodlarını saklamak için hem gönderici (sender) hem de alıcı (receiver) tarafında vector<Frame> yapısı kullanılmıştır. Frame içerisinde crc, veri (data), frame numarası, frame boyutu, isPadding (veri bitleri padding (tamamlama) içeriyor mu bilgisi), gönderenin adresi gibi bilgiler tutulmaktadır. Şekil 2'de ödev için kodlanan cheksum fonksiyonu yer almaktadır. Bu fonksiyon kullanılarak gerçekleştirilen hesaplama işlemi sonunda, checksum değeri hem gönderici hem de alıcı tarafında karşılaştırılmaktadır. Eğer checksum değerleri eşleşmezse verinin tekrar gönderilmesi gerekmektedir.

```
string ProtocolUtils::calculateChecksum(vector<Frame> fr, int frmNum) {
    int n = 8;  // 8-bitlik parça
    string concat_data = "";
    string sum = "00000000"; // Başlangıçta sum 8-bit olarak 0

    // Tüm frame'lerin CRC'lerini concat_data'ya ekleyelim
    for (int i = 0; i < frmNum; i++) {
        concat_data += fr[i].crc.substr(0, fr[i].frameSize);  // CRC'yi frame boyutuna göre alıyoruz
    }

    // concat_data'yı 8-bitlik parçalara ayırıp toplama işlemi yapalım
    for (size_t i = 0; i < concat_data.size(); i += 8) {
        string chunk = concat_data.substr(i, 8);  // 8-bitlik parçayı alıyoruz
        if (chunk.length() < 8) {
            chunk = string(8 - chunk.length(), '0') + chunk;  // Eğer eksikse başa 0 ekleyip tamamlıy
        }

        // sum ile bu 8-bitlik chunk'ı topluyoruz
        sum = binaryAddStrings(sum, chunk);
    }
    // Son adımda 1 ekliyoruz
    sum = binaryAddStrings(sum, "00000001");
    // Eğer toplamda 8 bitten uzun bir sonuç olduysa, sadece son 8 bit kalacak şekilde kesiyoruz
    if (sum.length() > 8) {
        sum = sum.substr(sum.length() - 8, 8);
    }
    return sum;
}
```

Şekil 2 – Checksum Hesaplama Kodu

Cheksum gönderimi sırasında kullanılan frame'in yapısı veri iletimi sırasında kullanılandan farklıdır. Tablo 2'de kullanılan cheksum frame yapısı gösterilmektedir.

Tablo 2 – Cheksum Frame Yapısı

"01111110"	HEADER		TRAILER	"01111110"
Flag(8 bit)	Adres (7 bit)	Frame No (1 bit)	Checksum (8 bit)	Flag (8 bit)

Elde edilen checksum sonucu hexadecimal formatta arayüzde gösterilmektedir.

6. Stop &Wait Tekniği ve Hata Simülasyonu:

Stop&Wait Akış Kontrolü Tekniği, veri iletiminde her bir frame gönderildikten sonra göndericinin alıcıdan bir onay (ACK) veya hata (NACK) almasını beklediği bir yöntemdir. Bu teknik, her frame'in başarılı bir şekilde alıcıya ulaşıp ulaşmadığını kontrol etmek için kullanılmaktadır. Ödevde bizden istenen verinin iletimi konusunda yaşanabilecek hata senaryoları da bu bölümde açıklanmaktadır.

6.1. Gönderici Frame Gönderir:

Gönderici, belirli bir frame'i gönderme işlemi başlatır. Örneğin, sendFrame fonksiyonunda, gönderici, frame verisini hazırlar, CRC (hata kontrol kodu) hesaplar ve frame'i alıcıya iletir. Bu işlem, her bir frame için ayrı ayrı gerçekleştirilir. Projede gönderilen framelerin kontrolünü sağlamak amacıyla gönderici, alıcıdan gelen ACK onayını aldıktan sonra yeni frame'i göndermektedir. Alıcı frame'in hatalı olması durumunda NACK göndermektedir. Bu durumda gönderici aynı frame'i alıcıya tekrar gönderir. Aynı zamanda gönderilen ACK cevabının göndericiye ulaşmaması durumunda da aynı frame tekrar gönderilmektedir. Gönderici ACK yanıtını alana kadar yeni frame göndermez ve 200 ms bekledikten sonra (belirlenen timeout miktarı) veriyi tekrar gönderir.

6.2. Frame İletimi ve Hata Senaryoları:

Gönderici, gönderdiği frame'in başarıyla iletilip iletilmediğini belirlemek için çeşitli hata durumlarını simüle edebilmektedir. Bu işlemi gerçekleştirebilmek adına rand fonksiyonu kullanılmıştır ve gönderilen framelerin ödev dosyasında belirtilen oranlara uygun olacak şekilde bir kısmı hatalı/eksik gönderilir. Bu hata durumları, frame kaybı, frame bozulması ve ACK kaybolması gibi senaryoları içermektedir.

6.2.1. Frame Kaybı Senaryosu: %10 ihtimalle gönderilen frame kaybolur (frameLost durumu), bu durumda alıcıya ulaşmaz ve gönderici aynı frame'i yeniden gönderir.
6.2.2. Frame Bozulması: %20 ihtimalle gönderilen frame bozulur (bilincliHata durumu). İlk denemede bozulmuş bir veri gönderilir.
6.2.3. ACK Kaybolması: %15 ihtimalle ACK mesajı kaybolur (ackLost durumu) ve gönderici tekrar ACK almayı bekler.
6.2.4. Checksum Değerinin Bozulması: %5 ihtimalle checksum kodunda hata vardır ve gönderici ve alıcı taraftaki değerler birbirleriyle uyuşmaz, bu durumda bütün veriler en baştan gönderilir.

6.3. Tekrar Gönderim (Retransmission):

Eğer ACK alınamazsa veya bir hata oluşursa, gönderici aynı frame'i yeniden gönderir. Bu, Stop&Wait akış kontrolünün temel özelliğidir. Gönderici, her zaman bir frame'in başarılı bir şekilde iletildiğini doğrulamadan bir sonraki frame'i göndermez.

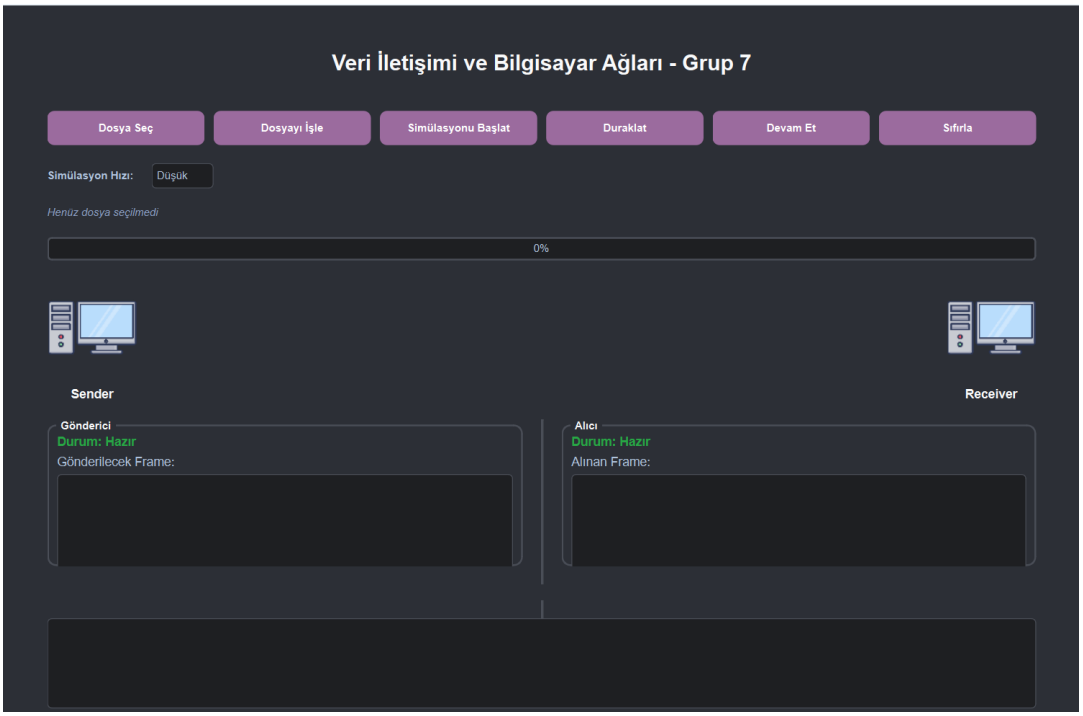
6.4 Verinin Başarıyla Gönderilmesi:

Eğer frame, hata durumları (frame kaybı, bozulma, ACK kaybolması) başarılı bir şekilde yönetilirse, frame başarılı bir şekilde gönderilir ve alıcı tarafından onaylanır (ACK gönderilir). Bu işlemden sonra "Frame X başarıyla alındı." mesajı gösterilir. Bu durumda, gönderici bir sonraki frame'i göndermeye başlar.

7. Arayüz

Arayüz için Qt Creator kullanılmıştır. Qt üzerinden arayüz tasarımı oluşturulmuştur. Arayüzde giriş ekranı ve simülasyon ekranı bulunmaktadır. Bu iki sayfa Şekil 3 ve 4'te bulunmaktadır.

Şekil 3 - Giriş Ekranı



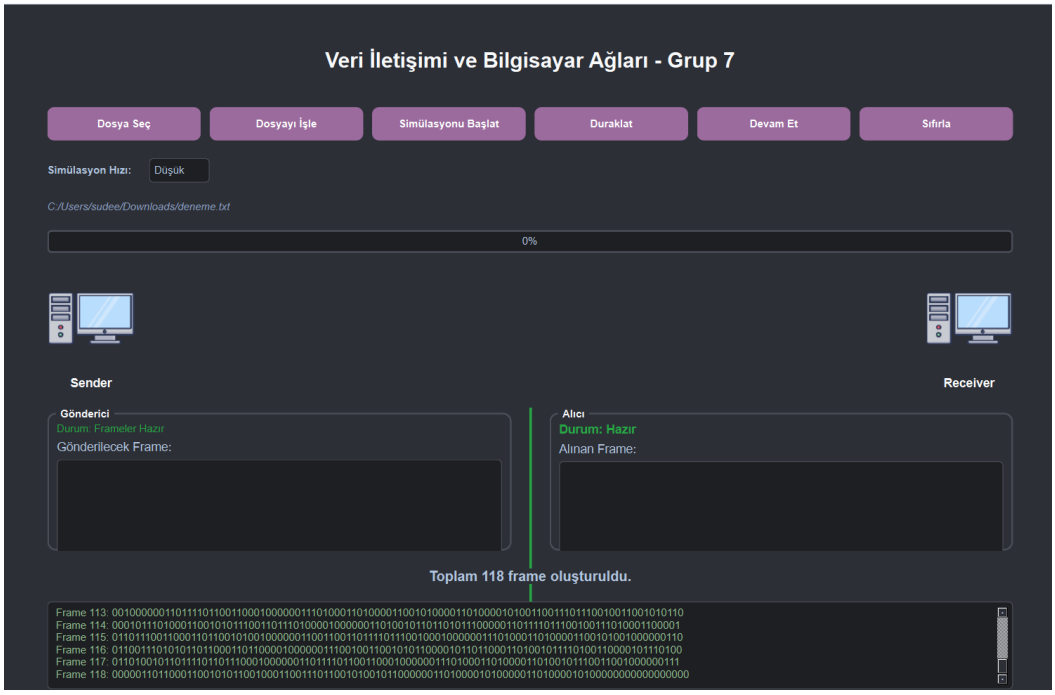
Şekil 4 - Simülasyon Ekranı

Simülasyon ekranında Simülasyon Hızı, Dosya Seç, Dosyayı İşle, Simülasyonu Başlat, Duraklat, Devam Et ve Sıfırla butonları bulunmaktadır. Bu butonlar sırasıyla şu işlevlere sahiptir:

7.1. Simülasyon Hızı Butonu: Simülasyonu düşük - orta - yüksek olmak üzere hızlarla çalıştırmaktadır. Düşük hız 3 saniye, orta hız 1 saniye, yüksek hız ise yarım saniyede bir frame'i göndermektedir.

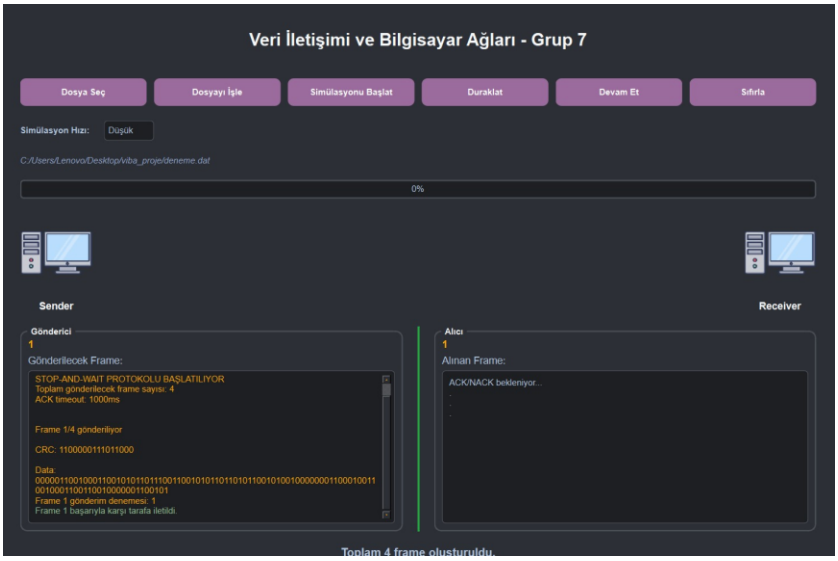
7.2. Dosya Seç Butonu: Dosya seçme işlemleri (dat dosyaları) buradan gerçekleşmektedir.

7.3. Dosyayı İşle Butonu: Seçilen dosyanın içerisinde bulunan text verisini frame'lere ayırır. Her frame 100 bitlik datadan oluşacak şekilde bölütleme yapılır. Bu aşamada kullanılacak flag'in "01111110" olduğundan ötürü veride data transparency'yi önlemek adına bit stuffing işlemi gerçekleştirilmektedir. Ekranın altında oluşturulan frame'ler ve sayısı gösterilmektedir. Şekil 5'te bu ana ait ekran görüntüsü bulunmaktadır.

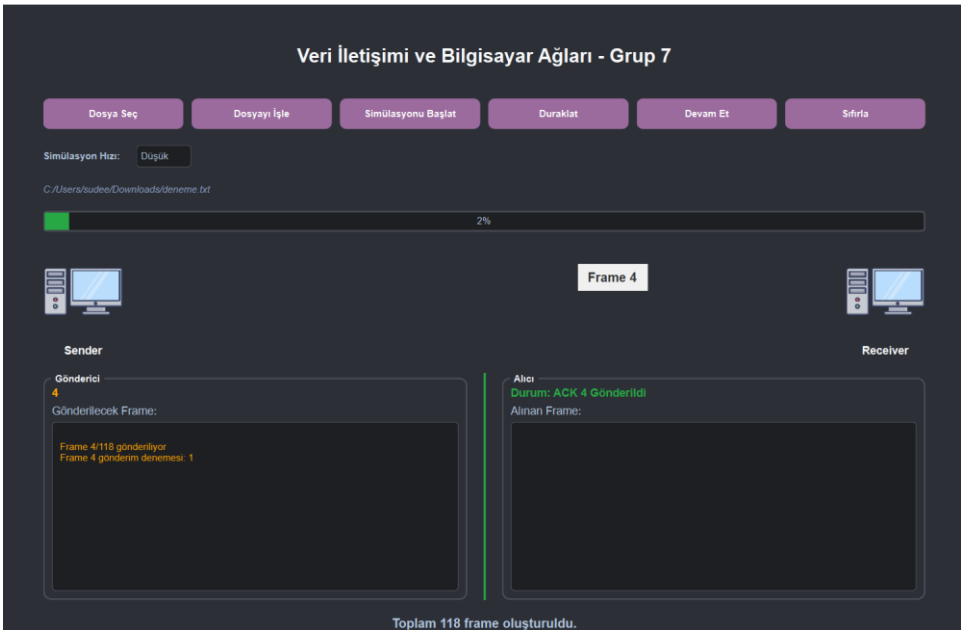


Şekil 5 – Frame'lerin oluşturulması ve ekrana yazılması

7.4. Simülasyonu Başlat: Verilere framing uygulandıktan sonra simülasyonu başlat butonuna tıklanır ve simülasyon başlatılır. Duraklat butonuyla simülasyon duraklatılır, duraklatılan işlemi devam ettirmek için devam et butonu kullanılmaktadır. Tüm işlem bittikten sonra sıfırla butonuna basılarak yeniden dosya seçme işlemi yapılabilir ve tekrar simülasyon başlatılabilmektedir. Her frame gönderilirken o frame'e ait CRC kodu ve data burada Gönderici'nin alanına yazılmaktadır. Şekil 6, 7, 8, 9, 10 ve 11'de simülasyonun işleyişine dair görseller bulunmaktadır.

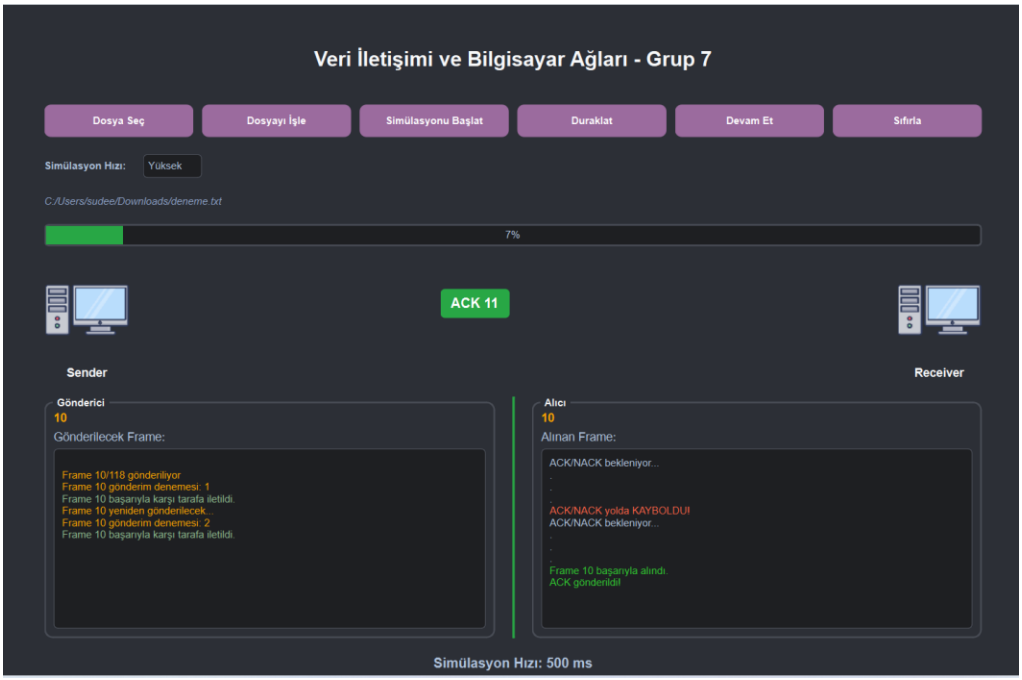


Şekil 6 – Frame'lerin gönderilmesi ve CRC'nin Ekranda Gösterimi

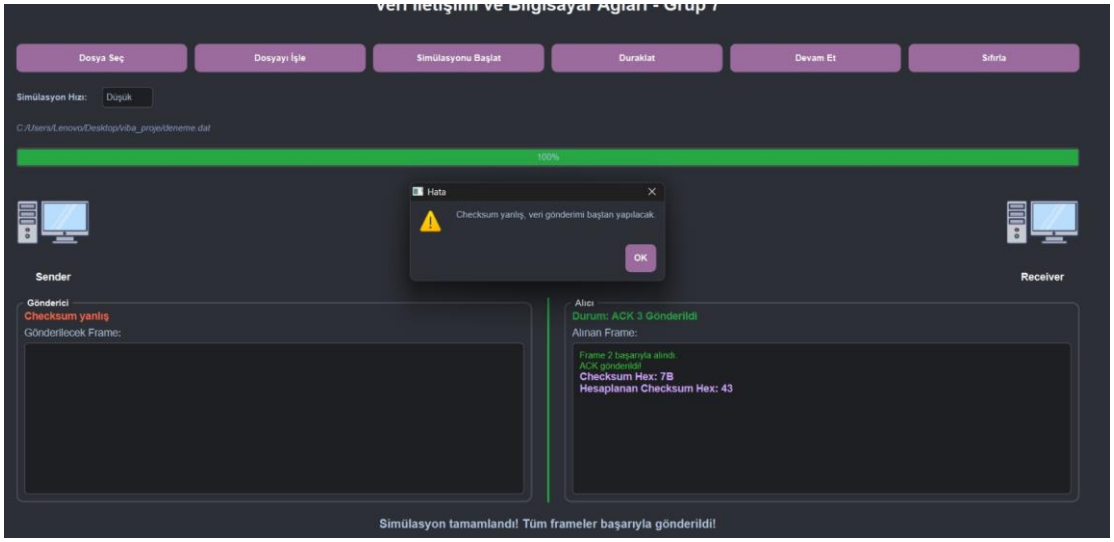


Şekil 7 – Frame'lerin gönderilmesi

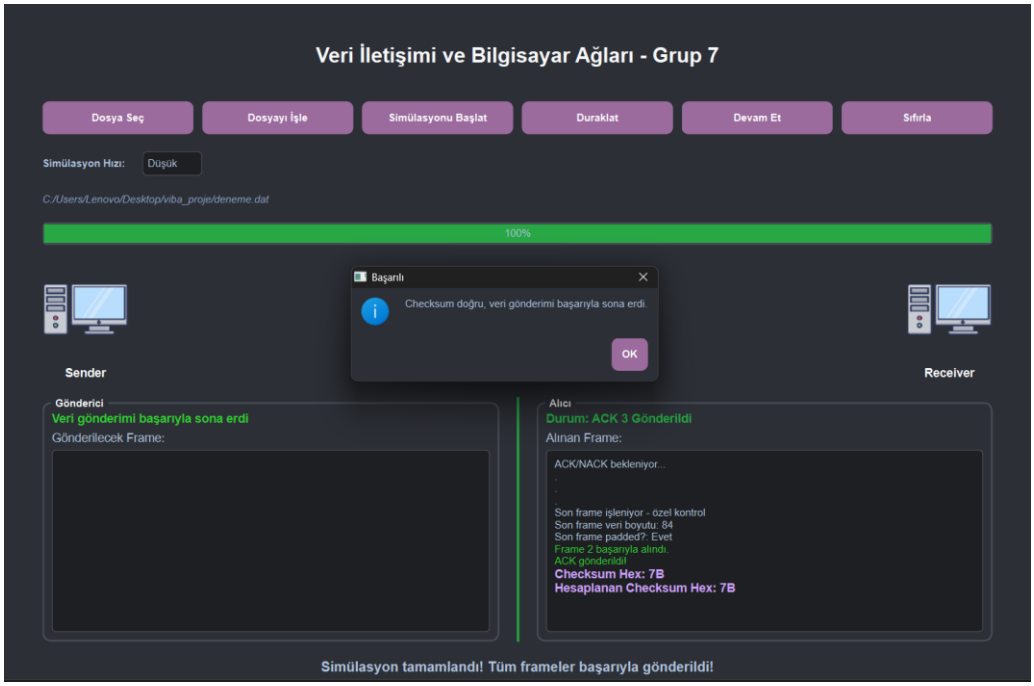
Şekil 8 - Hatalı Veri Gönderimi (CRC Sonuçları Uyuşmadığında)



Şekil 9 - Doğru Veri Gönderimi (CRC Sonuçları Aynı)



Şekil 10 - Tüm Framelerin Gönderilmesine Rağmen Checksum Sonucunun Hatalı Olmasından Ötürü Bildirim Oluşturulması



Şekil 11 - Simülasyon İşlem Sonucu