

## 1. GİRİŞ

Bu proje, ağ iletişimi ve güvenlik kavramlarını kapsamlı bir şekilde inceleyen bir uygulama projesidir. TCP/IP yığını üzerinden ağ performans ölçümü, şifreleme (AES), kimlik doğrulama, veri bütünlüğü kontrolü (SHA-256) gibi temel bileşenleri içermektedir. Ayrıca, Man-in-the-Middle (MITM) ve paket enjeksiyon gibi saldırı türlerine karşı önlemleri de test edilmiştir.

Çalışma kapsamında; Wireshark, iperf3, tc gibi araçlar, Python socket programlama, Linux/Windows ortamları ve sanal makineler (VirtualBox) kullanılmıştır. Böylece ağ güvenliği ve performans analizinin pratikte nasıl yapıldığı gözlemlenmiştir.

## 2. TEKNİK DETAYLAR

### 2.1 Kullanılan Yöntem ve Araçlar

- Python socket modülü: TCP istemci-sunucu iletişimi.
- AES: Verileri şifrelemek için simetrik anahtarlı şifreleme.
- SHA-256: Dosya bütünlüğünü sağlamak için kriptografik özet fonksiyonu.
- iperf3: Bant genişliği ve paket kaybı analizleri için.
- tc (Linux Traffic Control): Paket kaybı simülasyonu.
- Wireshark: Paket analizi.
- Scapy: Sahte paket enjeksiyon simülasyonu.

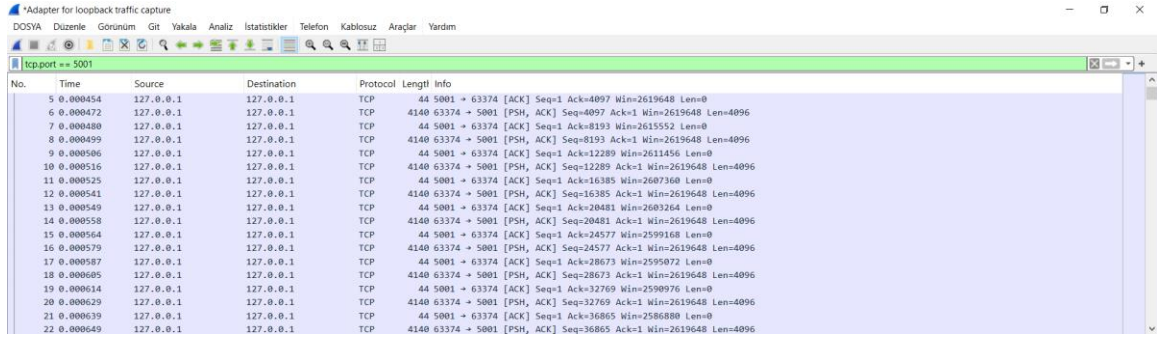
## 2.2 İstemci-Sunucu Yapısı (Paket Parçalama, İletim ve Birleştirme)

Bu projede istemci-sunucu mimarisi kullanılmıştır. İstemci tarafı (client.py) dosya gönderimini sağlar; sunucu tarafı (server.py) ise gelen dosyayı alır ve kaydeder. İletişim TCP üzerinden gerçekleştirilmiştir.

```
PS D:\SİSTEM-SİLME\Desktop\proje_kodu> python server.py
[Sunucu] Bağlantı bekleniyor...
[Sunucu] Bağlantı geldi: ('127.0.0.1', 63374)
[Sunucu] Dosya başarıyla alındı.
```

Şekil 1. "server.py" terminal çıktısı

İstemci, socket modülü yardımıyla sunucuya bağlanır ve 4096 baytlık parçalar halinde dosya verisini gönderir. Sunucu, bağlantı isteğini kabul eder ve gelen verileri alarak birleştirir. Bu yapı sayesinde, veri bütünlüğü ve güvenilir iletim sağlanır.



No.	Time	Source	Destination	Protocol	Length	Info
5	0.000454	127.0.0.1	127.0.0.1	TCP	44	5801 → 63374 [ACK] Seq=1 Ack=4097 Win=2619648 Len=0
6	0.000472	127.0.0.1	127.0.0.1	TCP	4140	63374 → 5801 [PSH, ACK] Seq=4097 Ack=1 Win=2619648 Len=4096
7	0.000480	127.0.0.1	127.0.0.1	TCP	44	5801 → 63374 [ACK] Seq=1 Ack=8193 Win=2615552 Len=0
8	0.000499	127.0.0.1	127.0.0.1	TCP	4140	63374 → 5801 [PSH, ACK] Seq=8193 Ack=1 Win=2619648 Len=4096
9	0.000506	127.0.0.1	127.0.0.1	TCP	44	5801 → 63374 [ACK] Seq=1 Ack=12289 Win=2611456 Len=0
10	0.000516	127.0.0.1	127.0.0.1	TCP	4140	63374 → 5801 [PSH, ACK] Seq=12289 Ack=1 Win=2619648 Len=4096
11	0.000525	127.0.0.1	127.0.0.1	TCP	44	5801 → 63374 [ACK] Seq=1 Ack=16385 Win=2607360 Len=0
12	0.000541	127.0.0.1	127.0.0.1	TCP	4140	63374 → 5801 [PSH, ACK] Seq=16385 Ack=1 Win=2619648 Len=4096
13	0.000549	127.0.0.1	127.0.0.1	TCP	44	5801 → 63374 [ACK] Seq=1 Ack=20481 Win=2603264 Len=0
14	0.000558	127.0.0.1	127.0.0.1	TCP	4140	63374 → 5801 [PSH, ACK] Seq=20481 Ack=1 Win=2619648 Len=4096
15	0.000564	127.0.0.1	127.0.0.1	TCP	44	5801 → 63374 [ACK] Seq=1 Ack=24577 Win=2599168 Len=0
16	0.000579	127.0.0.1	127.0.0.1	TCP	4140	63374 → 5801 [PSH, ACK] Seq=24577 Ack=1 Win=2619648 Len=4096
17	0.000587	127.0.0.1	127.0.0.1	TCP	44	5801 → 63374 [ACK] Seq=1 Ack=28673 Win=2595072 Len=0
18	0.000605	127.0.0.1	127.0.0.1	TCP	4140	63374 → 5801 [PSH, ACK] Seq=28673 Ack=1 Win=2619648 Len=4096
19	0.000614	127.0.0.1	127.0.0.1	TCP	44	5801 → 63374 [ACK] Seq=1 Ack=32769 Win=2590976 Len=0
20	0.000629	127.0.0.1	127.0.0.1	TCP	4140	63374 → 5801 [PSH, ACK] Seq=32769 Ack=1 Win=2619648 Len=4096
21	0.000639	127.0.0.1	127.0.0.1	TCP	44	5801 → 63374 [ACK] Seq=1 Ack=36865 Win=2586880 Len=0
22	0.000649	127.0.0.1	127.0.0.1	TCP	4140	63374 → 5801 [PSH, ACK] Seq=36865 Ack=1 Win=2619648 Len=4096

Şekil 2. Wireshark ile paket gönderiminin incelenmesi

Wireshark çıktılarında da görüldüğü gibi, TCP segmentasyonu ve akış kontrolü başarıyla çalışmaktadır. Dosya gönderimi tamamlandığında bağlantı kapatılır.

```
PS D:\SİSTEM-SİLME\Desktop\proje_kodu> python client.py
[İstemci] Dosya gönderimi tamamlandı.
```

Şekil3. Dosya gönderimi tamamlandıktan sonra terminal çıktısı

Bu yapı sayesinde:

- **Dosya bütünlüğü sağlanır:** Sunucu, istemciden gelen veriyi eksiksiz alır ve disk üzerinde kaydeder.
- **TCP'nin özellikleri kullanılır:** Segmentasyon, sıralı iletim ve hata kontrolü gibi TCP özellikleri veri akışında güvenilirlik sağlar.

### 2.3. Manuel IP Başlık İşleme

Bu bölümde, projenin temel bileşenlerinden biri olan manuel IP başlık işleme ve TCP başlık oluşturma süreci detaylandırılmıştır. Ubuntu Virtual Machine ortamında kullanılan *ip\_raw\_test.py* adlı Python betiği, düşük seviyeli ağ paketlerinin doğrudan oluşturulması ve gönderilmesini sağlamaktadır.

```
ip_header = struct.pack('!BBHHBBH4s4s',
                        ver_ihl,
                        tos,
                        total_length,
                        identification,
                        flags_offset,
                        ttl,
                        protocol,
                        checksum_ip,
                        socket.inet_aton(src_ip),
                        socket.inet_aton(dst_ip))
```

Şekil 4. "struct.pack" fonksiyonu

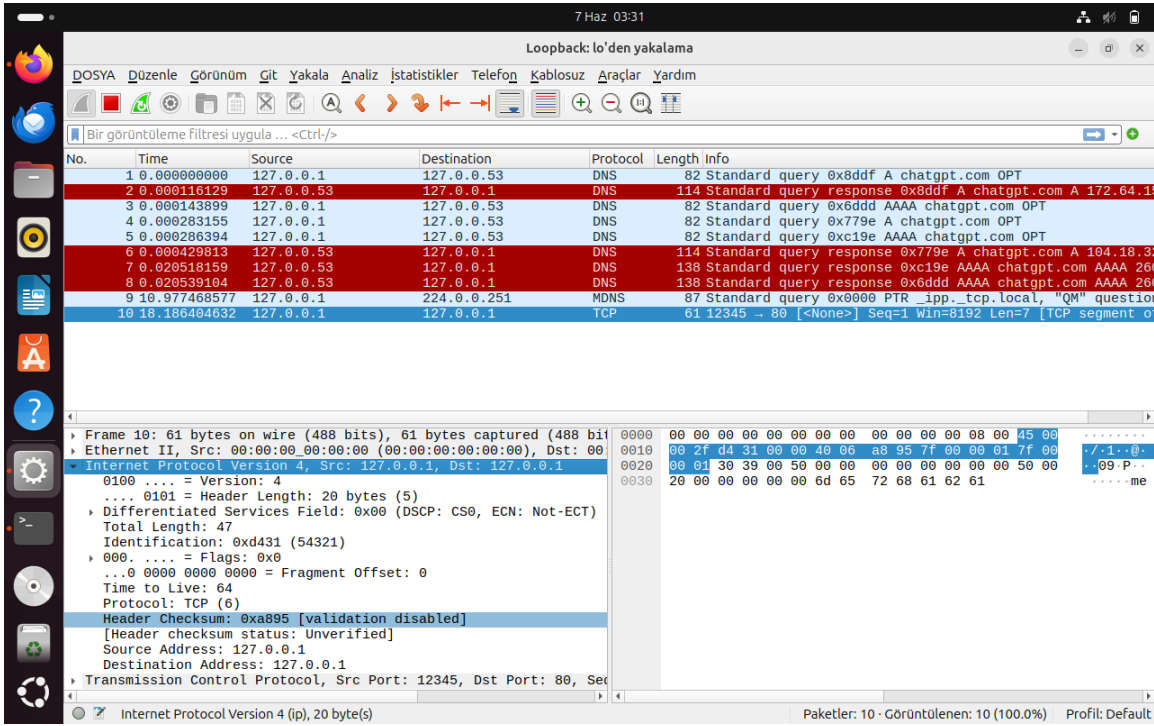
Betiğin amacı, ham (raw) socket üzerinden IP başlığını ve TCP başlığını elle oluşturup checksum (sağlama toplamı) gibi alanları manuel olarak hesaplamaktır. Bu işlem sırasında:

- IP başlığı için sürüm, header uzunluğu, toplam uzunluk, TTL, protokol gibi alanlar struct.pack fonksiyonu kullanılarak sıralı olarak yerleştirilmiştir.
- Hesaplanan IP başlık checksum değeri, başlığa eklenmiştir.

- TCP başlığı da benzer şekilde manuel olarak oluşturulmuştur (SYN bayrağı, kaynak ve hedef portları vb.).
- Payload olarak “merhaba” verisi eklenmiştir.

```
dide@dide-VirtualBox:~$ sudo python3 ip_raw_test.py
[sudo] dide için parola:
Hesaplanan Checksum: 0xa895
[+] IP paketi gönderildi (TCP header + payload dahil).
dide@dide-VirtualBox:~$
```

Şekil 5. Terminalde manuel IP paket üretimi ve gönderimi



Şekil 6. Wireshark'ta manuel oluşturulan IP paketinin başarılı şekilde yakalanması ve TCP/IP başlık detaylarının görüntülenmesi

Terminal çıktısında “Hesaplanan Checksum: 0xa895” satırı, checksum’ın elle doğru bir şekilde hesaplandığını göstermektedir. Betik çalıştırıldığında ise “[+] IP paketi gönderildi (TCP header + payload dahil)” mesajı alınmıştır. Ayrıca, Wireshark üzerinde yapılan analizde de paketin tüm TCP/IP başlık alanlarının uygun şekilde oluşturulduğu gözlemlenmiştir.

Bu adımlar, düşük seviyeli IP başlık işleme yeteneğinin ve TCP header manuel üretiminin başarıyla gerçekleştirildiğini ve iletim sırasında bir hata olmadığını kanıtlamaktadır.

## 2.4 Ağ Performans Ölçümü

Bu bölümde, projenin ağ performansını kapsamlı bir şekilde değerlendirmek için farklı test senaryoları uygulanmıştır. Testler üç temel bileşen üzerinden yürütülmüştür: gecikme ölçümü (RTT), bant genişliği analizi ve paket kaybı yönetimi. Testler, Ubuntu Virtual Machine ve Windows host ortamında gerçekleştirilen iperf3, ping ve tc (traffic control) araçları kullanılarak tamamlanmıştır.

### 2.4.1 Gecikme Ölçümü (RTT)

Gecikme ölçümü, istemci ile hedef cihaz (modem, VPN sunucusu gibi) arasındaki paketlerin gidiş-dönüş süresini ölçmek için *ping* komutu kullanılarak yapılmıştır. Ubuntu terminalinden 10 paketlik ICMP istekleri gönderilmiş ve gecikme istatistikleri elde edilmiştir. Örneğin; Wi-Fi ağı üzerinden yapılan ölçümde ortalama RTT değeri **~107.9 ms** olarak hesaplanmıştır. Bu değerler, Wi-Fi ağlarının dalgalı gecikme performansını ve kararsız zamanlamalarını göstermektedir.

```
dide@dide-VirtualBox:~$ ping 192.168.1.1 -c 10
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=898 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=3.58 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=64 time=7.68 ms
64 bytes from 192.168.1.1: icmp_seq=4 ttl=64 time=11.1 ms
64 bytes from 192.168.1.1: icmp_seq=5 ttl=64 time=3.62 ms
64 bytes from 192.168.1.1: icmp_seq=6 ttl=64 time=7.41 ms
64 bytes from 192.168.1.1: icmp_seq=7 ttl=64 time=118 ms
64 bytes from 192.168.1.1: icmp_seq=8 ttl=64 time=8.85 ms
64 bytes from 192.168.1.1: icmp_seq=9 ttl=64 time=12.6 ms
64 bytes from 192.168.1.1: icmp_seq=10 ttl=64 time=8.48 ms

--- 192.168.1.1 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9009ms
rtt min/avg/max/mdev = 3.578/107.966/898.031/265.420 ms
dide@dide-VirtualBox:~$
```

Şekil 7. Ubuntu Terminalinden Wi-Fi RTT Testi Çıktısı

```
cmd Komut İstemi
Microsoft Windows [Version 10.0.19043.1165]
(c) Microsoft Corporation. Tüm hakları saklıdır.

C:\Users\UserR>ping 104.223.104.233 -n 10

Pinging 104.223.104.233 with 32 bytes of data:
Reply from 104.223.104.233: bytes=32 time=164ms TTL=64
Reply from 104.223.104.233: bytes=32 time=165ms TTL=64
Reply from 104.223.104.233: bytes=32 time=167ms TTL=64
Reply from 104.223.104.233: bytes=32 time=165ms TTL=64
Reply from 104.223.104.233: bytes=32 time=166ms TTL=64
Reply from 104.223.104.233: bytes=32 time=742ms TTL=64
Reply from 104.223.104.233: bytes=32 time=575ms TTL=64
Reply from 104.223.104.233: bytes=32 time=472ms TTL=64
Reply from 104.223.104.233: bytes=32 time=566ms TTL=64
Reply from 104.223.104.233: bytes=32 time=342ms TTL=64

Ping statistics for 104.223.104.233:
    Packets: Sent = 10, Received = 10, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 164ms, Maximum = 742ms, Average = 352ms
```

Şekil 8 . VPN Bağlantısı İçin RTT Testi Çıktısı

VPN bağlantısı üzerinden de RTT testi yapılmıştır. VPN'e bağlandıktan sonra *ping* komutuyla VPN sunucusunun IP adresine istekler gönderilmiş ve gecikme değerleri incelenmiştir. VPN bağlantısında ortalama RTT, doğrudan modem bağlantısına göre daha yüksek çıkmıştır. Bu durum, VPN üzerinden geçen şifreleme ve tünelleme işlemlerinin gecikme süresini artırdığını göstermektedir.

#### 2.4.2. Bant Genişliği Analizi

Bu bölümde, Ubuntu VirtualBox VM üzerinde çalışan istemci ve Windows ana makinedeki sunucu kullanılarak iperf3 aracıyla yapılan bant genişliği analizine yer verilmiştir. Testler TCP protokolü üzerinden gerçekleştirilmiş ve Wi-Fi bağlantısının gerçek performansı gözlemlenmiştir.



```

ca. Komut İstemi - iperf3.exe -s
Microsoft Windows [Version 10.0.19043.1165]
(c) Microsoft Corporation. Tüm hakları saklıdır.

C:\Users\User>cd /d "D:\SİSTEM-SİLME\Downloads\iperf3.17.1_64\iperf3.17.1_64"

D:\SİSTEM-SİLME\Downloads\iperf3.17.1_64\iperf3.17.1_64>iperf3.exe -s
-----
Server listening on 5201 (test #1)
-----
Accepted connection from 192.168.1.12, port 48106
[ 5] local 192.168.1.11 port 5201 connected to 192.168.1.12 port 48118
[ ID] Interval           Transfer             Bitrate
[ 5]  0.00-1.00   sec    186 MBytes    1.56 Gbits/sec
[ 5]  1.00-2.00   sec    184 MBytes    1.55 Gbits/sec
[ 5]  2.00-3.00   sec    181 MBytes    1.52 Gbits/sec
[ 5]  3.00-4.00   sec    184 MBytes    1.54 Gbits/sec
[ 5]  4.00-5.00   sec    183 MBytes    1.53 Gbits/sec
[ 5]  5.00-6.00   sec    176 MBytes    1.48 Gbits/sec
[ 5]  6.00-7.00   sec    184 MBytes    1.55 Gbits/sec
[ 5]  7.00-8.00   sec    190 MBytes    1.59 Gbits/sec
[ 5]  8.00-9.00   sec    173 MBytes    1.45 Gbits/sec
[ 5]  9.00-10.00  sec    171 MBytes    1.44 Gbits/sec
[ 5] 10.00-10.01  sec     1.00 MBytes    1.65 Gbits/sec
-----
[ ID] Interval           Transfer             Bitrate
[ 5]  0.00-10.01  sec    1.77 GBytes    1.52 Gbits/sec
-----
Server listening on 5201 (test #2)
-----

```

Şekil 9. Server Tarafından Terminal Çıktısı

```

dide@dide-VirtualBox:~$ iperf3 -c 192.168.1.11
Connecting to host 192.168.1.11, port 5201
[ 5] local 192.168.1.12 port 48118 connected to 192.168.1.11 port 5201
[ ID] Interval           Transfer             Bitrate          Retr  Cwnd
[ 5]  0.00-1.00   sec    188 MBytes    1.58 Gbits/sec     0    542 KBytes
[ 5]  1.00-2.00   sec    184 MBytes    1.55 Gbits/sec     0    542 KBytes
[ 5]  2.00-3.00   sec    182 MBytes    1.53 Gbits/sec     0    542 KBytes
[ 5]  3.00-4.00   sec    183 MBytes    1.54 Gbits/sec     0    542 KBytes
[ 5]  4.00-5.00   sec    182 MBytes    1.53 Gbits/sec     0    542 KBytes
[ 5]  5.00-6.00   sec    177 MBytes    1.49 Gbits/sec     0    542 KBytes
[ 5]  6.00-7.00   sec    184 MBytes    1.54 Gbits/sec     0    542 KBytes
[ 5]  7.00-8.00   sec    190 MBytes    1.60 Gbits/sec     0    542 KBytes
[ 5]  8.00-9.00   sec    174 MBytes    1.46 Gbits/sec     0    542 KBytes
[ 5]  9.00-10.00  sec    171 MBytes    1.43 Gbits/sec     0    542 KBytes
-----
[ ID] Interval           Transfer             Bitrate          Retr
[ 5]  0.00-10.00  sec    1.77 GBytes    1.52 Gbits/sec     0
[ 5]  0.00-10.01  sec    1.77 GBytes    1.52 Gbits/sec
-----
iperf Done.

```

Şekil 10. İstemci Tarafından Terminal Çıktısı

Ubuntu VM tarafında, gönderim hızı ortalama olarak 1.52 Gbit/s düzeyinde ölçülmüştür. Sunucu tarafındaki ölçüm değerleri de bu sonucu doğrulamış ve ağ bağlantısının

kararlılığını ortaya koymuştur. Veri akışı sırasında paket kaybı veya yeniden iletim ihtiyacı neredeyse gözlemlenmemiştir.

Bu sonuçlar, Wi-Fi ağının teorik maksimum hızına yakın bir değerle pratikte sağlanan hız arasında fark olmadığını ve bağlantının yüksek performansla çalıştığını göstermiştir.

### 2.4.3. Paket Kaybı Yönetimi

Paket kaybını simüle etmek amacıyla Ubuntu VM’de *tc (traffic control)* aracı kullanılarak arayüzde %10 oranında paket kaybı uygulanmıştır. Hem TCP hem de UDP protokolleriyle iperf3 testleri tekrar gerçekleştirilmiştir. Sonuçlar şu şekilde gözlemlenmiştir:

- **UDP**

**Testi:**

- Paket kaybı %10 olarak tanımlandığında, alıcı tarafında kayıp oranı **%9.6** olarak ölçülmüştür.
- Gönderici tarafında kayıp raporlanmamıştır çünkü UDP, veri kaybı durumunu doğal olarak algılayamaz.

```
dide@dide-VirtualBox:~$ sudo tc qdisc add dev enp0s3 root netem loss 10%
[sudo] dide için parola:
dide@dide-VirtualBox:~$ iperf3 -c 192.168.1.11 -u -b 10M
Connecting to host 192.168.1.11, port 5201
[ 5] local 192.168.1.12 port 42646 connected to 192.168.1.11 port 5201
[ ID] Interval      Transfer    Bitrate      Total Datagrams
[ 5] 0.00-1.00 sec  1.19 MBytes 10.0 Mbits/sec 856
[ 5] 1.00-2.00 sec  1.19 MBytes 10.0 Mbits/sec 856
[ 5] 2.00-3.00 sec  1.19 MBytes 10.0 Mbits/sec 856
[ 5] 3.00-4.00 sec  1.19 MBytes 10.0 Mbits/sec 857
[ 5] 4.00-5.00 sec  1.19 MBytes 10.0 Mbits/sec 856
[ 5] 5.00-6.00 sec  1.19 MBytes 10.0 Mbits/sec 856
[ 5] 6.00-7.00 sec  1.19 MBytes 9.98 Mbits/sec 855
[ 5] 7.00-8.00 sec  1.19 MBytes 10.0 Mbits/sec 857
[ 5] 8.00-9.00 sec  1.19 MBytes 9.98 Mbits/sec 855
[ 5] 9.00-10.00 sec 1.19 MBytes 10.0 Mbits/sec 857
-----
[ ID] Interval      Transfer    Bitrate      Jitter    Lost/Total Datagrams
[ 5] 0.00-10.00 sec 11.9 MBytes 10.0 Mbits/sec 0.000 ms 0/8561 (0%) sender
[ 5] 0.00-10.00 sec 10.8 MBytes 9.04 Mbits/sec 0.110 ms 820/8560 (9.6%) receiver

iperf Done.
```

Şekil 11. UDP Testi

- **TCP Testi:**



- TCP protokolünde ise, paket kaybına karşı tekrar iletim mekanizmaları (retransmission) aktif olarak devreye girmiştir. Retransmission sayıları raporlanarak protokolün güvenilir veri iletim özelliği doğrulanmıştır (örneğin 498 adet tekrar iletim).

```
dide@dide-VirtualBox:~$ sudo tc qdisc add dev enp0s3 root netem loss 10%
[sudo] dide için parola:
dide@dide-VirtualBox:~$ iperf3 -c 192.168.1.11
Connecting to host 192.168.1.11, port 5201
[ 5] local 192.168.1.12 port 45404 connected to 192.168.1.11 port 5201
[ ID] Interval            Transfer        Bitrate         Retr  Cwnd
[ 5]  0.00-1.00 sec      2.25 MBytes    18.9 Mbits/sec   105   1.43 KBytes
[ 5]  1.00-2.00 sec      0.00 Bytes     0.00 bits/sec    21   1.43 KBytes
[ 5]  2.00-3.00 sec      1.00 MBytes     8.39 Mbits/sec   70   2.85 KBytes
[ 5]  3.00-4.00 sec      256 KBytes     2.10 Mbits/sec   31   2.85 KBytes
[ 5]  4.00-5.00 sec      384 KBytes     3.15 Mbits/sec   15   2.85 KBytes
[ 5]  5.00-6.00 sec      256 KBytes     2.10 Mbits/sec   36   1.43 KBytes
[ 5]  6.00-7.00 sec      640 KBytes     5.25 Mbits/sec   52   2.85 KBytes
[ 5]  7.00-8.00 sec      896 KBytes     7.34 Mbits/sec   56   2.85 KBytes
[ 5]  8.00-9.00 sec      1.00 MBytes     8.39 Mbits/sec   87   1.43 KBytes
[ 5]  9.00-10.00 sec     256 KBytes     2.10 Mbits/sec   25   1.43 KBytes
-----
[ ID] Interval            Transfer        Bitrate         Retr
[ 5]  0.00-10.00 sec     6.88 MBytes     5.77 Mbits/sec   498
[ 5]  0.00-10.00 sec     6.38 MBytes     5.35 Mbits/sec
iperf Done.
```

Şekil 12. TCP Testi

Bu testler, UDP'nin paket kaybı durumunda veri kaybını tolere etmediğini, TCP'nin ise veri bütünlüğünü sağlamak için otomatik tekrar iletim yaptığını göstermektedir.

#### 2.4.4. Farklı Ağ Türlerinin Karşılaştırılması

##### Wi-Fi RTT Test Sonuçları

Ubuntu VM üzerinde Wi-Fi bağlantısı kullanılarak gerçekleştirilen RTT testi, *ping* komutu aracılığıyla 10 paket gönderilerek ölçülmüştür. Elde edilen sonuçlar:

- Minimum RTT: 3.578 ms
- Maksimum RTT: 898.031 ms
- Ortalama RTT: 107.966 ms
- Önemli gözlem: RTT değerlerinin geniş bir aralıkta değiştiği görülmektedir (3 ms - 898 ms). Bu dalgalanma, Wi-Fi bağlantısının zaman zaman kararsız olabileceğini

göstermektedir. Özellikle ortalama değerin 107 ms gibi yüksek bir seviyede olması, ağın dalgalı performansına işaret etmektedir.

### **VPN RTT Test Sonuçları**

Windows bilgisayarda VPN bağlantısı aktif haldeyken yapılan RTT testi de yine *ping* komutuyla yürütülmüştür. Bu test sırasında VPN sunucusu IP'sine 10 paket gönderilmiş ve sonuçlar aşağıdaki gibi kaydedilmiştir:

- Minimum RTT: 164 ms
- Maksimum RTT: 742 ms
- Ortalama RTT: 352 ms
- Önemli gözlem: VPN bağlantısı kullanıldığında RTT değerleri Wi-Fi ağına göre önemli ölçüde daha yüksektir. Minimum RTT değeri bile Wi-Fi testinin ortalamasının üzerinde olup, ortalama değeri 352 ms olarak gözlenmiştir. Bu, VPN'in şifreleme ve tünelleme süreçlerinin, gecikmeye ciddi katkı sağladığını ortaya koymaktadır.

### **Sonuç**

Bu testler, VPN tünelleme ve şifreleme katmanlarının, özellikle ağ performansında gecikmeye yol açtığını açıkça ortaya koymaktadır. Wi-Fi ağı ise, bağlantı kalitesi ve parazit gibi faktörlere bağlı olarak daha geniş bir RTT aralığına sahiptir. Her iki test de, ağ yapısı ve kullanılan protokolün RTT performansı üzerindeki etkilerini net bir şekilde yansıtmaktadır.

## **2.5 Veri Şifreleme, Güvenlik Analizi ve Saldırı Simülasyonu**

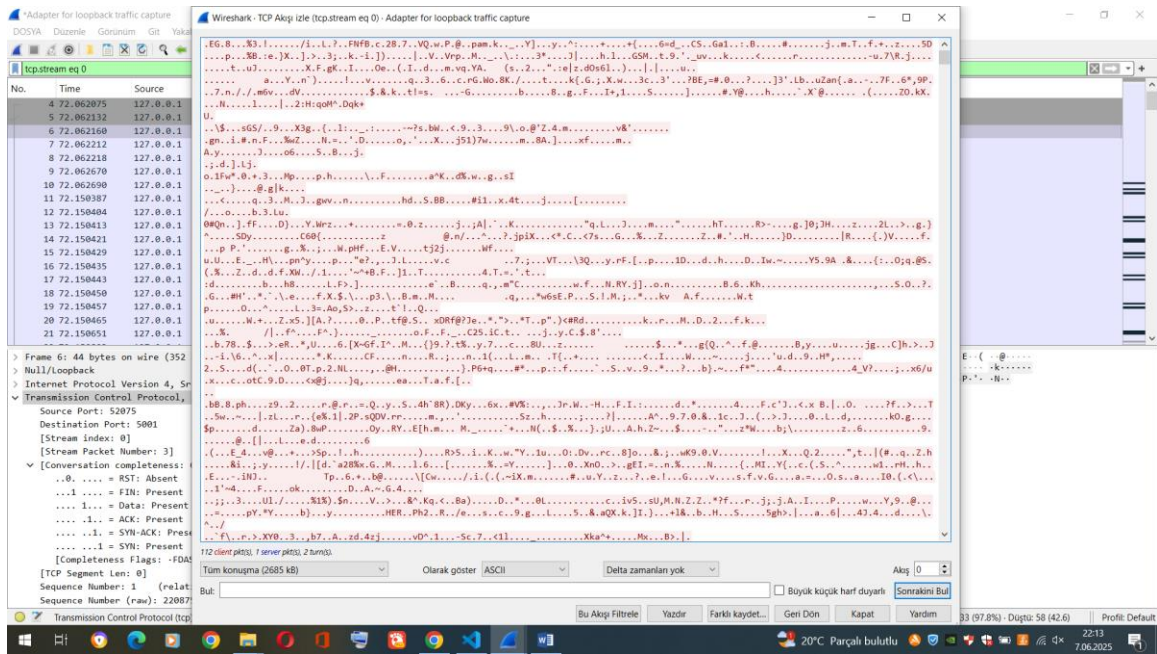
Bu bölümde, proje kapsamında kullanılan şifreleme teknikleri, güvenlik önlemleri ve saldırı simülasyonlarının detaylı analizi sunulmaktadır. Ayrıca, Wireshark ve sahte paket enjeksiyon testleri de değerlendirilmiştir.

### 2.5.1. Veri Şifreleme ve Bütünlük Kontrolü

Projede, istemci tarafında (client.py) dosya gönderimi öncesinde AES (Advanced Encryption Standard) algoritmasıyla veri şifreleme işlemi gerçekleştirilmiştir. AES CBC (Cipher Block Chaining) modu kullanılmış ve veri blokları padding işlemiyle standart hale getirilmiştir. Şifreleme için cryptography kütüphanesinin AES-CBC fonksiyonları tercih edilmiştir.

Sunucu tarafında (server.py), gelen şifreli veri aynı anahtar kullanılarak çözülmüştür. Çözümleme sonrası dosya bütünlüğü SHA-256 algoritmasıyla doğrulanmıştır. SHA-256 özeti hem istemci hem de sunucu tarafında hesaplanmış ve iletim sırasında bir bozulma olup olmadığı kontrol edilmiştir.

### 2.5.2. Şifreli Veri ve Wireshark İncelemesi



Şekil 13. Şifreli Verinin Wireshark Üzerinden İncelenmesi

Wireshark ekran görüntülerinde, gönderilen paketlerin ham verisi analiz edilmiştir. Şifrelenmiş veri, Wireshark üzerinde anlamsız bloklar halinde görüntülenmiş ve açık metin veri tespit edilememiştir. Bu durum, şifreleme işleminin iletim sırasında veriyi

koruduğunu ve MITM (Man-in-the-Middle) saldırısı yapılsa dahi verinin okunamayacağını kanıtlamaktadır. Şifreleme anahtarı bilinmediği sürece verinin anlamlı bir şekilde çözölemeyeceği anlaşılmıştır.

### 2.5.3. Kimlik Doğrulama ve Erişim Kontrolü

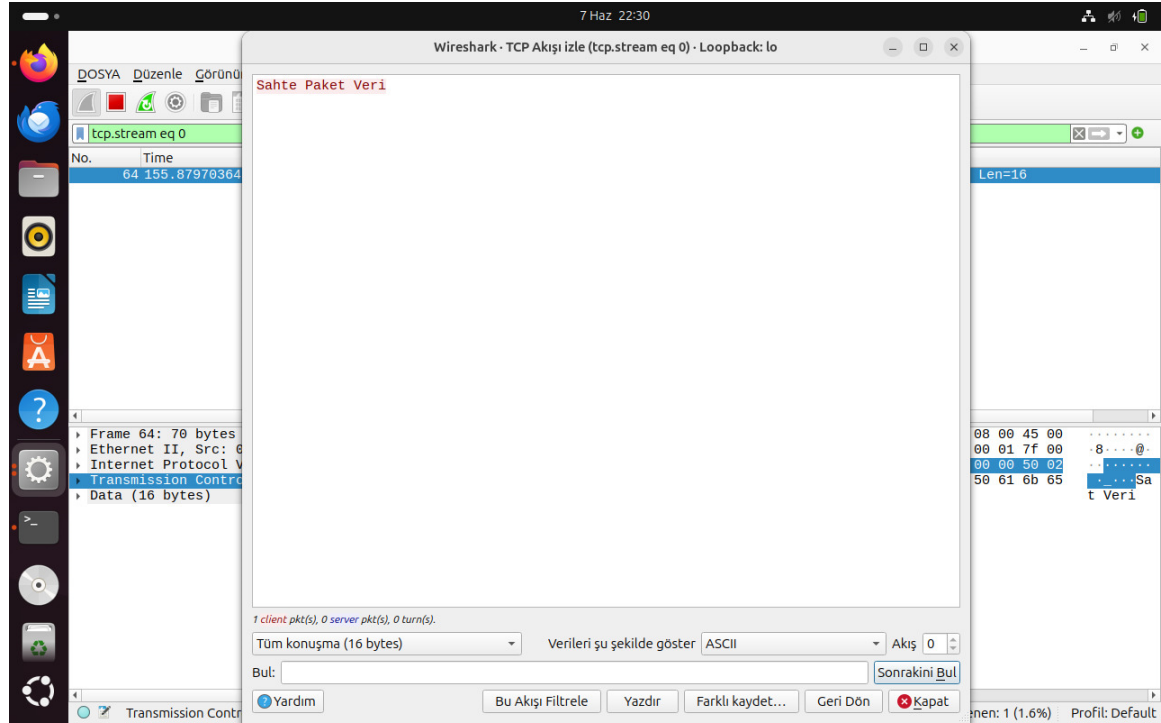
```
PS D:\SİSTEM-SİLME\Desktop\proje_kodu> python server.py
[Sunucu] Bağlantı bekleniyor...
[Sunucu] Bağlantı geldi: ('127.0.0.1', 52075)
[Sunucu] Kimlik doğrulama başarılı!
[Sunucu] Dosya başarıyla alındı ve çözüldü.
[Sunucu] SHA-256 bütünlük doğrulaması BAŞARILI!
PS D:\SİSTEM-SİLME\Desktop\proje_kodu>

PS D:\SİSTEM-SİLME\Desktop\proje_kodu> python client.py
[İstemci] Kimlik doğrulama başarılı, dosya gönderimine başlanıyor.
[İstemci] Şifreli dosya gönderimi tamamlandı.
[İstemci] SHA-256 gönderildi, işlem tamamlandı.
PS D:\SİSTEM-SİLME\Desktop\proje_kodu>
```

Şekil 14. Terminalde “Kimlik Doğrulama ve Dosya Gönderimi Başarılı” Çıktısı

İstemci bağlantısı sırasında parola tabanlı kimlik doğrulama uygulanmıştır. Yanlış parola gönderilmesi durumunda sunucu bağlantıyı kapatmaktadır. Bu yöntem, kaba kuvvet saldırılarına karşı temel bir koruma sağlar ve yetkisiz erişimi engeller. Terminal çıktılarında, “Kimlik doğrulama başarılı!” ve “Kimlik doğrulama hatalı” mesajlarıyla doğrulama sürecinin etkili şekilde çalıştığı belgelenmiştir.

### 2.5.4. Saldırı Simülasyonu: Paket Enjeksiyonu



Şekil 15. Sahte paketin Wireshark üzerinden incelenmesi

Proje kapsamında saldırı simülasyonu amacıyla *fake\_packet\_injection.py* betiği geliştirilmiştir. Bu betik, Scapy kütüphanesi kullanılarak sahte TCP SYN paketleri oluşturmaktadır. Kaynak ve hedef IP'leri 127.0.0.1 (localhost) olarak ayarlanmış ve payload olarak "Sahte Paket Veri" eklenmiştir. Betik çalıştırıldığında terminalde "Sahte paket gönderildi!" mesajı görüntülenmiştir. Wireshark'ta yapılan analizde ise sahte SYN paketleri doğrulanarak bu enjeksiyonun ağda algılandığı gözlemlenmiştir.

Bu simülasyon, ağ dinleme ve sahte paket gönderme (spoofing) gibi temel saldırıların nasıl gerçekleşebileceğini ve bu saldırıların Wireshark gibi araçlar yardımıyla nasıl tespit edilebileceğini göstermektedir.

### 2.5.5. Sonuçlar ve Güvenlik Analizi

Bu testler ve simülasyonlar sonucunda aşağıdaki güvenlik sonuçlarına ulaşılmıştır:

- AES şifreleme, ağ üzerinden iletilen verinin okunamaz (şifreli) kalmasını sağlamıştır.
- SHA-256 bütünlük kontrolü, verinin değiştirilmediğini doğrulamıştır.
- TCP/IP temel kimlik doğrulama yöntemi, yalnızca doğru parola gönderildiğinde bağlantı kurulmasına izin vermiştir.
- Sahte paket enjeksiyon testi, Wireshark gibi analiz araçlarının saldırıları nasıl yakalayabileceğini göstermiştir.
- MITM saldırısı simülasyonu yapılmamış olsa da, şifreli verinin Wireshark'ta okunamaz olması bu saldırıya karşı bir koruma sağladığını kanıtlamaktadır.

Bu bileşenler, projenin güvenlik yönlerini güçlendirmiş ve teorik saldırı senaryolarına karşı dayanıklılığı göstermiştir.

## 3. SINIRLAMALAR VE İYİLEŞTİRMELER

### 3.1 Sınırlandırmalar

Bu projede kullanılan test ortamı, sanal makine ve yerel ağ gibi sınırlı bir bağlamda gerçekleştirilmiştir. Bu nedenle elde edilen gecikme, bant genişliği ve saldırı simülasyonu sonuçları gerçek dünya koşullarını tam olarak yansıtmayabilir. Ayrıca kullanılan VPN bağlantısı için farklı coğrafi konumlar veya ağ koşulları test edilmemiştir. Projenin güvenlik analizi bölümü, yalnızca temel kimlik doğrulama ve şifreleme testleriyle sınırlı kalmıştır. Gerçek hayattaki karmaşık saldırı teknikleri ve güvenlik önlemleri daha ileri düzeyde inceleme gerektirir.

### 3.2 İyileştirmeler

Projede ileriye dönük olarak yapılabilecek iyileştirmeler arasında daha fazla test ortamı ve farklı ağ senaryoları yer almaktadır. Özellikle farklı VPN sağlayıcıları, kablolu ağ (Ethernet) gibi ek ağ türleriyle testler genişletilebilir. MITM (Aracı Adam) saldırıları gibi güvenlik açıklarının test edilmesi için daha profesyonel araçlar (ör. Ettercap) kullanılabilir. Ayrıca, kimlik doğrulama ve şifreleme mekanizmalarını daha güçlü protokoller (ör. TLS/SSL) ile geliştirmek, veri güvenliğini artırabilir. Son olarak, istemci-sunucu uygulamasında kullanıcı arayüzü ve loglama gibi özellikler eklenerek kullanım kolaylığı sağlanabilir.

## 4. SONUÇ

Bu proje, temel istemci-sunucu mimarisi ile şifreli iletişimi, dosya bütünlüğü doğrulamasını, ağ performans ölçümlerini ve saldırı senaryolarını kapsamlı bir şekilde test etmiştir.

Sonuç olarak, proje başarıyla tamamlanmış ve ağ güvenliğinin temel prensipleri pratikte uygulanmıştır.

YouTube Video Bağlantısı:



## 5. KAYNAKLAR

Scapy. (n.d.). *Scapy documentation*. Retrieved June 8, 2025, from <https://scapy.net>

Iperf. (n.d.). *iperf3 documentation*. Retrieved June 8, 2025, from <https://iperf.fr>

Wireshark Foundation. (n.d.). *Wireshark documentation*. Retrieved June 8, 2025, from <https://www.wireshark.org/docs/>

Linux Foundation. (n.d.). *tc(8) – Linux man page*. Retrieved June 8, 2025, from <https://linux.die.net/man/8/tc>

OpenAI ChatGPT. (2025). *OpenAI ChatGPT*.