

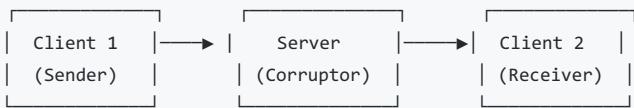
# Socket Programming Assignment – Data Transmission with Error Detection Methods

## Proje Özeti

Bu proje, veri iletişiminde kullanılan hata tespit yöntemlerini (Parity, 2D Parity, CRC, Hamming Code, Internet Checksum) pratik olarak gösteren bir socket programlama uygulamasıdır.

## 1. Sistem Mimarisı

Proje üç ana bileşenden oluşmaktadır:



Bileşenler:

1. **Client 1 (Data Sender)**: Kullanıcıdan veri alır, kontrol bilgisi üretir
2. **Server (Intermediate Node + Data Corruptor)**: Veriyi alır, hata enjekte eder, Client 2'ye ileter
3. **Client 2 (Receiver + Error Checker)**: Veriyi alır, kontrol bilgisini doğrular

## 2. Client 1 – Data Sender

Görevler:

- Kullanıcıdan metin girişi alır
- Seçilen yönteme göre kontrol bilgisi üretir
- Paketi DATA|METHOD|CONTROL\_INFORMATION formatında gönderir

Kod Örneği:

```

// client1_sender.cpp - Ana Fonksiyon

int main() {
    // Socket oluştur ve server'a bağlan
    int clientSocket = socket(AF_INET, SOCK_STREAM, 0);
    // ... bağlantı kodu ...

    // Kullanıcıdan veri al
    std::string data;
    std::cout << "Enter data to send: ";
    std::getline(std::cin, data);

    // Yöntem seçimi
    std::cout << "\nSelect error detection method:" << std::endl;
    std::cout << "1. Parity Bit" << std::endl;
    std::cout << "2. 2D Parity" << std::endl;
    std::cout << "3. CRC-16" << std::endl;
    std::cout << "4. Hamming Code" << std::endl;
    std::cout << "5. Internet Checksum" << std::endl;

    int choice;
    std::cin >> choice;

    std::string methodStr;
    std::string controlInfo;

    switch (choice) {
        case 1:
            methodStr = "PARITY";
            controlInfo = ErrorDetection::calculateParity(data, true);
            break;
        case 3:
            methodStr = "CRC16";
            controlInfo = ErrorDetection::calculateCRC16(data);
            break;
        // ... diğer yöntemler ...
    }

    // Paket oluştur ve gönder
    std::string packet = data + "|" + methodStr + "|" + controlInfo;
    send(clientSocket, packet.c_str(), packet.length(), 0);
}

```

## Paket Formatı:

HELLO|CRC16|87AF

## 3. Hata Tespit Yöntemleri

### 3.1. Parity Bit (Even/Odd Parity)

#### Çalışma Prensibi:

- Verinin binary temsilindeki 1'lerin sayısını sayar
- Even parity: Toplam 1 sayısı çift olmalı
- Odd parity: Toplam 1 sayısı tek olmalı

#### Kod Örneği:

```
// error_detection.h

static std::string calculateParity(const std::string& data, bool evenParity = true) {
    std::string binary = stringToBinary(data);
    int ones = countOnes(binary);
    bool parityBit = evenParity ? (ones % 2 != 0) : (ones % 2 == 0);
    return parityBit ? "1" : "0";
}
```

Örnek:

- Veri: "A" → Binary: 01000001 → 1'ler: 2 (çift) → Parity: 0

### 3.2. 2D Parity (Matrix Parity)

Çalışma Prensibi:

- Veriyi satır-sütun matrisine (örn: 8x8) böler
- Her satır ve sütun için parity hesaplar
- Satır ve sütun parity'lerini birleştirir

Kod Örneği:

```
// error_detection.h

static std::string calculate2DParity(const std::string& data, int rows = 8) {
    std::string binary = stringToBinary(data);
    int cols = (binary.length() + rows - 1) / rows;

    // Matris oluştur
    std::vector<std::vector<bool>> matrix(rows, std::vector<bool>(cols, false));

    // Binary veriyi matrise yerleştir
    for (size_t i = 0; i < binary.length(); i++) {
        int row = i % rows;
        int col = i / rows;
        matrix[row][col] = (binary[i] == '1');
    }

    // Satır parity'leri hesapla
    std::string rowParity;
    for (int i = 0; i < rows; i++) {
        int ones = 0;
        for (int j = 0; j < cols; j++) {
            if (matrix[i][j]) ones++;
        }
        rowParity += (ones % 2 == 0) ? "0" : "1";
    }

    // Sütun parity'leri hesapla
    std::string colParity;
    for (int j = 0; j < cols; j++) {
        int ones = 0;
        for (int i = 0; i < rows; i++) {
            if (matrix[i][j]) ones++;
        }
        colParity += (ones % 2 == 0) ? "0" : "1";
    }

    // Hex formatına çevir
    return binaryToHex(rowParity) + "|" + binaryToHex(colParity);
}
```

### 3.3. CRC-16 (Cyclic Redundancy Check)

#### Çalışma Prensibi:

- Polynomial division kullanır (Polynomial: 0x8005)
- 16-bit remainder kontrol bilgisi olur

#### Kod Örneği:

```
// error_detection.h

static std::string calculateCRC16(const std::string& data) {
    uint16_t crc = 0xFFFF;
    uint16_t polynomial = 0x8005;

    for (char c : data) {
        crc ^= (static_cast<uint16_t>(c) << 8);
        for (int i = 0; i < 8; i++) {
            if (crc & 0x8000) {
                crc = (crc << 1) ^ polynomial;
            } else {
                crc <<= 1;
            }
        }
    }

    std::stringstream ss;
    ss << std::hex << std::uppercase << std::setfill('0')
       << std::setw(4) << crc;
    return ss.str();
}
```

#### Örnek:

- Veri: "HELLO" → CRC-16: 87AF

---

### 3.4. Hamming Code

#### Çalışma Prensibi:

- Veriyi 4-bit bloklara böler
- Her blok için parity bitleri ekler (Hamming(7,4))
- Hata düzeltme yeteneği sağlar

#### Kod Örneği:

```

// error_detection.h

static std::string calculateHamming(const std::string& data) {
    std::string binary = stringToBinary(data);
    std::string hammingResult;

    // 4-bit bloklar halinde işle
    for (size_t i = 0; i < binary.length(); i += 4) {
        std::string block = binary.substr(i, 4);
        if (block.length() < 4) {
            block += std::string(4 - block.length(), '0');
        }

        // Parity bitleri hesapla (Hamming(7,4))
        bool p1 = (block[0] - '0') ^ (block[1] - '0') ^ (block[3] - '0');
        bool p2 = (block[0] - '0') ^ (block[2] - '0') ^ (block[3] - '0');
        bool p4 = (block[1] - '0') ^ (block[2] - '0') ^ (block[3] - '0');

        hammingResult += std::to_string(p1) + std::to_string(p2) +
                         block[0] + std::to_string(p4) + block.substr(1);
    }

    // Checksum hesapla
    int checksum = 0;
    for (char c : hammingResult) {
        checksum += (c - '0');
    }

    std::stringstream ss;
    ss << std::hex << std::uppercase << std::setfill('0')
       << std::setw(4) << checksum;
    return ss.str();
}

```

### 3.5. Internet Checksum (IP Checksum)

#### Çalışma Prensibi:

- Veriyi 16-bit kelime'lere böler
- Tüm kelime'leri toplar
- Carry bitlerini ekler
- One's complement alır

#### Kod Örneği:

```

// error_detection.h

static std::string calculateChecksum(const std::string& data) {
    uint32_t sum = 0;

    // 16-bit kelimeler halinde işle
    for (size_t i = 0; i < data.length(); i += 2) {
        uint16_t word = 0;
        if (i < data.length()) {
            word = static_cast<uint8_t>(data[i]) << 8;
        }
        if (i + 1 < data.length()) {
            word |= static_cast<uint8_t>(data[i + 1]);
        }
        sum += word;
    }

    // Carry bitlerini ekle
    while (sum >> 16) {
        sum = (sum & 0xFFFF) + (sum >> 16);
    }

    // One's complement
    uint16_t checksum = ~static_cast<uint16_t>(sum);

    std::stringstream ss;
    ss << std::hex << std::uppercase << std::setfill('0')
       << std::setw(4) << checksum;
    return ss.str();
}

```

## 4. Server – Intermediate Node + Data Corruptor

---

### Görevler:

- Client 1'den paketi alır
- Paketi parse eder (data, method, control\_info)
- Veriye hata enjekte eder
- Corrupted veriyi Client 2'ye gönderir (orijinal kontrol bilgisini koruyarak)

### Kod Örneği:

```

// server.cpp - Ana Fonksiyon

int main() {
    // Client 1'den paket al
    char buffer[4096];
    ssize_t bytesReceived = recv(client1Socket, buffer, sizeof(buffer) - 1, 0);
    buffer[bytesReceived] = '\0';
    std::string packet(buffer);

    // Paketi parse et: DATA|METHOD|CONTROL_INFORMATION
    size_t firstPipe = packet.find('|');
    size_t secondPipe = packet.find('|', firstPipe + 1);

    std::string data = packet.substr(0, firstPipe);
    std::string method = packet.substr(firstPipe + 1, secondPipe - firstPipe - 1);
    std::string controlInfo = packet.substr(secondPipe + 1);

    // Hata enjekte et
    std::string corruptedData = ErrorInjection::injectError(data);

    std::cout << "Original Data: " << data << std::endl;
    std::cout << "Corrupted Data: " << corruptedData << std::endl;

    // Corrupted paketi oluştur (orijinal kontrol bilgisini koru)
    std::string corruptedPacket = corruptedData + "|" + method + "|" + controlInfo;

    // Client 2'ye gönder
    send(client2Socket, corruptedPacket.c_str(), corruptedPacket.length(), 0);
}

```

## 5. Hata Enjeksiyon Yöntemleri

### 5.1. Bit Flip

Kod Örneği:

```

// error_injection.h

static std::string injectBitFlip(const std::string& data) {
    std::string binary = stringToBinary(data);
    std::mt19937& gen = getRandomGenerator();
    std::uniform_int_distribution<int> dis(0, binary.length() - 1);

    int pos = dis(gen);
    binary[pos] = (binary[pos] == '0') ? '1' : '0';

    return binaryToString(binary);
}

```

Örnek:

- Orijinal: "HELLO" → Binary'de bir bit değişir → Corrupted: Farklı karakter

### 5.2. Character Substitution

Kod Örneği:

```
// error_injection.h

static std::string injectCharSubstitution(const std::string& data) {
    if (data.empty()) return data;

    std::mt19937& gen = getRandomGenerator();
    std::uniform_int_distribution<> posDis(0, data.length() - 1);
    std::uniform_int_distribution<> charDis(32, 126); // Printable ASCII

    std::string corrupted = data;
    int pos = posDis(gen);
    corrupted[pos] = static_cast<char>(charDis(gen));

    return corrupted;
}
```

**Örnek:**

- Orijinal: "HELLO" → Corrupted: "HEZLO" (L → Z)

### 5.3. Character Deletion

**Kod Örneği:**

```
// error_injection.h

static std::string injectCharDeletion(const std::string& data) {
    if (data.empty()) return data;

    std::mt19937& gen = getRandomGenerator();
    std::uniform_int_distribution<> dis(0, data.length() - 1);

    std::string corrupted = data;
    corrupted.erase(dis(gen), 1);

    return corrupted;
}
```

**Örnek:**

- Orijinal: "HELLO" → Corrupted: "HELO" (bir karakter silindi)

### 5.4. Random Character Insertion

**Kod Örneği:**

```

// error_injection.h

static std::string injectCharInsertion(const std::string& data) {
    if (data.empty()) {
        std::mt19937& gen = getRandomGenerator();
        std::uniform_int_distribution<> charDis(32, 126);
        return std::string(1, static_cast<char>(charDis(gen)));
    }

    std::mt19937& gen = getRandomGenerator();
    std::uniform_int_distribution<> posDis(0, data.length());
    std::uniform_int_distribution<> charDis(32, 126);

    std::string corrupted = data;
    int pos = posDis(gen);
    corrupted.insert(pos, 1, static_cast<char>(charDis(gen)));

    return corrupted;
}

```

Örnek:

- Orijinal: "HELLO" → Corrupted: "HEaLLO" (a eklendi)
- 

## 5.5. Character Swapping

Kod Örneği:

```

// error_injection.h

static std::string injectCharSwapping(const std::string& data) {
    if (data.length() < 2) return data;

    std::mt19937& gen = getRandomGenerator();
    std::uniform_int_distribution<> dis(0, data.length() - 2);

    std::string corrupted = data;
    int pos = dis(gen);
    std::swap(corrupted[pos], corrupted[pos + 1]);

    return corrupted;
}

```

Örnek:

- Orijinal: "HELLO" → Corrupted: "HLELO" (E ve L yer değişti)
- 

## 5.6. Multiple Bit Flips

Kod Örneği:

```

// error_injection.h

static std::string injectMultipleBitFlips(const std::string& data) {
    if (data.empty()) return data;

    std::string binary = stringToBinary(data);
    std::mt19937& gen = getRandomGenerator();
    std::uniform_int_distribution<> countDis(2, 5); // 2-5 bit flip
    std::uniform_int_distribution<> posDis(0, binary.length() - 1);

    int flips = countDis(gen);
    for (int i = 0; i < flips; i++) {
        int pos = posDis(gen);
        binary[pos] = (binary[pos] == '0') ? '1' : '0';
    }

    return binaryToString(binary);
}

```

## 5.7. Burst Error

Kod Örneği:

```

// error_injection.h

static std::string injectBurstError(const std::string& data) {
    if (data.empty()) return data;

    std::mt19937& gen = getRandomGenerator();
    std::uniform_int_distribution<> burstDis(3,
        std::min(8, static_cast<int>(data.length())));
    std::uniform_int_distribution<> posDis(0,
        std::max(0, static_cast<int>(data.length()) - 3));
    std::uniform_int_distribution<> charDis(32, 126);

    int burstSize = burstDis(gen);
    int startPos = posDis(gen);

    std::string corrupted = data;
    for (int i = 0; i < burstSize && (startPos + i) < corrupted.length(); i++) {
        corrupted[startPos + i] = static_cast<char>(charDis(gen));
    }

    return corrupted;
}

```

Örnek:

- Orijinal: "HELLO WORLD" → Corrupted: "HELLO XyZLD" (3-8 karakter corrupt)

---

## 6. Client 2 – Receiver + Error Checker

Görevler:

1. Server'dan paketi alır
2. Paketi parse eder: `data, method, incoming_control = packet.split("|")`
3. Yönteme göre yeni kontrol bilgisi üretir
4. Gelen kontrol bilgisi ile hesaplanan kontrol bilgisini karşılaştırır
5. Sonucu ekrana yazdırır

Kod Örneği:

```

// client2_receiver.cpp - Ana Fonksiyon

int main() {
    // Server'dan paket al
    char buffer[4096];
    ssize_t bytesReceived = recv(serverSocket, buffer, sizeof(buffer) - 1, 0);
    buffer[bytesReceived] = '\0';
    std::string packet(buffer);

    // Paketi parse et: DATA|METHOD|CONTROL_INFORMATION
    size_t firstPipe = packet.find('|');
    size_t secondPipe = packet.find('|', firstPipe + 1);

    std::string receivedData = packet.substr(0, firstPipe);
    std::string methodStr = packet.substr(firstPipe + 1, secondPipe - firstPipe - 1);
    std::string incomingControl = packet.substr(secondPipe + 1);

    // Yönteme göre kontrol bilgisi hesapla
    ErrorDetectionMethod method = ErrorDetection::stringToMethod(methodStr);
    std::string computedControl;

    switch (method) {
        case ErrorDetectionMethod::PARITY:
            computedControl = ErrorDetection::calculateParity(receivedData, true);
            break;
        case ErrorDetectionMethod::CRC16:
            computedControl = ErrorDetection::calculateCRC16(receivedData);
            break;
        // ... diğer yöntemler ...
    }

    // Karşılaştır
    bool isCorrect = (incomingControl == computedControl);

    // Sonuçları yazdır
    std::cout << "Received Data : " << receivedData << std::endl;
    std::cout << "Method : " << methodStr << std::endl;
    std::cout << "Sent Check Bits : " << incomingControl << std::endl;
    std::cout << "Computed Check Bits : " << computedControl << std::endl;
    std::cout << "Status: " << (isCorrect ? "DATA CORRECT" : "DATA CORRUPTED")
        << std::endl;
}

```

## 7. Örnek Çalıştırma

### Senaryo:

1. Client 1: "HELLO" verisini CRC-16 ile gönderir
2. Server: Veriyi "HEYLLO" olarak corrupt eder (orijinal CRC'yi korur)
3. Client 2: Corrupted veriyi alır, yeni CRC hesaplar, karşılaştırır

### Çıktı Örneği:

```
==== Client 1: Data Sender ===
```

```
Enter data to send: HELLO
```

```
Select error detection method:
```

```
1. Parity Bit
```

```
2. 2D Parity
```

```
3. CRC-16
```

```
4. Hamming Code
```

```
5. Internet Checksum
```

```
Choice (1-5): 3
```

```
Generated Packet:
```

```
Data: HELLO
```

```
Method: CRC16
```

```
Control Information: 5189
```

```
Full Packet: HELLO|CRC16|5189
```

```
Packet sent successfully (17 bytes)
```

```
==== Server: Intermediate Node + Data Corruptor ===
```

```
Waiting for Client 1 on port 8080...
```

```
Client 1 connected!
```

```
Received packet from Client 1: HELLO|CRC16|5189
```

```
Parsed Packet:
```

```
Data: HELLO
```

```
Method: CRC16
```

```
Control Info: 5189
```

```
Error Injection Applied:
```

```
Original Data: HELLO
```

```
Corrupted Data: HEYLLO
```

```
Corrupted packet sent to Client 2 (17 bytes)
```

```
==== Client 2: Receiver + Error Checker ===
```

```
Waiting for server on port 8081...
```

```
Server connected!
```

```
Received packet from server: HEYLLO|CRC16|5189
```

```
==== Error Detection Results ===
```

```
Received Data : HEYLLO
```

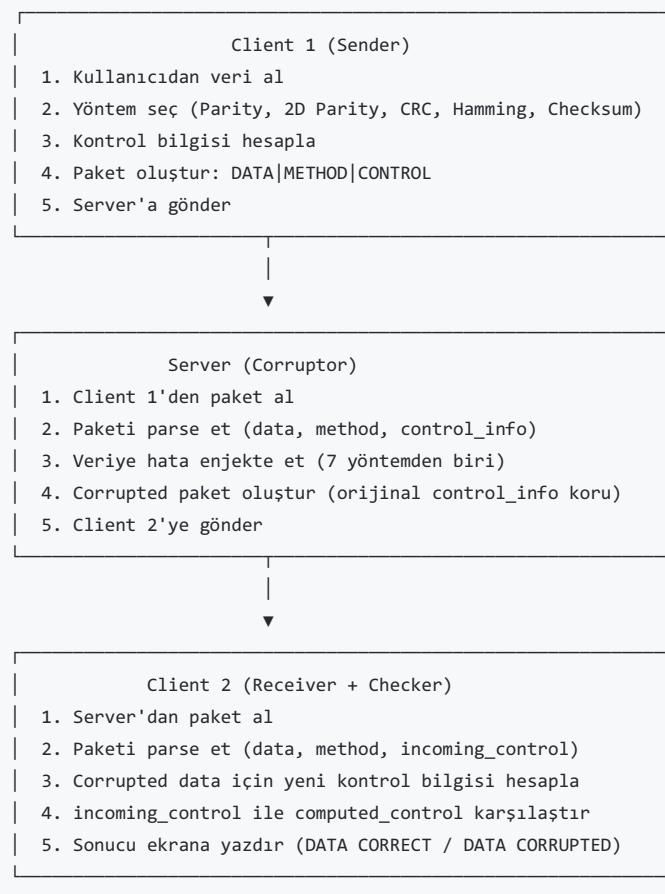
```
Method : CRC16
```

```
Sent Check Bits : 5189
```

```
Computed Check Bits : 8742
```

```
Status: DATA CORRUPTED
```

## 8. Sistem Akış Diyagramı



## 9. Teknik Detaylar

### 9.1. Socket Programlama

- Protokol: TCP/IP (SOCK\_STREAM)
- Portlar:
  - Client 1 → Server: 8080
  - Server → Client 2: 8081
- Platform: Linux/WSL (POSIX sockets)

### 9.2. Paket Formatı

DATA|METHOD|CONTROL\_INFORMATION

- DATA: Gönderilen/Corrupted veri
- METHOD: Hata tespit yöntemi (PARITY, PARITY2D, CRC16, HAMMING, CHECKSUM)
- CONTROL\_INFORMATION: Hesaplanan kontrol bilgisi (hex veya binary format)

### 9.3. Hata Tespit Yöntemleri Özeti

Yöntem	Kontrol Bilgisi Boyutu	Tespit Edebileceği Hatalar
Parity Bit	1 bit	Tek bit hataları
2D Parity	Row + Column parity	Tek ve çift bit hataları
CRC-16	16 bit	Çoklu bit hataları, burst errors
Hamming Code	4-bit blok başına 3 bit	Tek bit hataları (düzeltebilir)
Internet Checksum	16 bit	Çoklu bit hataları

## 10. Sonuç

Bu proje, veri iletişiminde kullanılan hata tespit yöntemlerini pratik olarak gösterir:

- ☒ 5 farklı hata tespit yöntemi implementasyonu
- ☒ 7 farklı hata enjeksiyon yöntemi ile test
- ☒ Socket programlama ile gerçek zamanlı veri iletimi
- ☒ Hata tespiti ve doğrulama mekanizması

Proje, network programming, error detection ve data integrity konularını birleştiren kapsamlı bir uygulamadır.

## Ek: Derleme ve Çalıştırma

Derleme:

```
make
```

Çalıştırma (3 ayrı terminal):

```
# Terminal 1  
./client2  
  
# Terminal 2  
./server  
  
# Terminal 3  
./client1
```

Proje Tarihi: Aralık 2025

Dil: C++17

Platform: Linux/WSL Ubuntu-22.04