**전송프로토콜 검증을 위한 자바 기반 네트워크 시뮬레이터 설계 및 구현**

The Design and Implementation of Network Simulator

2019. 9.

소의섭 ( creativezzanggu@naver.com )

목차

[1. 서론 5](#_Toc28156109)

[2. 네트워크 시뮬레이터 설계 7](#_Toc28156110)

[2.1 외부 구조 설계 7](#_Toc28156111)

[2.2 UDP 서버 설계 11](#_Toc28156112)

[2.3 UDP 클라이언트 설계 12](#_Toc28156113)

[2.4 TCP 서버 설계 13](#_Toc28156114)

[2.5 TCP 클라이언트 설계 15](#_Toc28156115)

[2.6 시스템의 전체 흐름도 17](#_Toc28156116)

[3. 네트워크 시뮬레이터 구현 19](#_Toc28156117)

[3.1. 개발환경 19](#_Toc28156118)

[3.2 설정화면 19](#_Toc28156119)

[3.4 모니터링 화면 20](#_Toc28156120)

[3.5 테스트 파일의 저장과 랜덤 발송 22](#_Toc28156121)

[3.6 패킷 발송 시간 간격과 발송 횟수 설정 24](#_Toc28156122)

[3.7 TCP 통신 시 클라이언트 연결 유무 표시 25](#_Toc28156123)

[3.8 네트워크 연결 해제 시 소켓과 입출력 클래스 객체들의 우아한 종료 26](#_Toc28156124)

[3.9 소프트웨어 병목현상 제거 27](#_Toc28156125)

[3.10 성능 테스트 29](#_Toc28156126)

[4. 결론 및 향후 과제 29](#_Toc28156127)

그림목차

[그림 1. 프레임의 기초구성 8](#_Toc28156137)

[그림 2. headPanel 설계1 9](#_Toc28156138)

[그림 3. headPanel 설계2 9](#_Toc28156139)

[그림 4. bodyPanel 설계1 10](#_Toc28156140)

[그림 5. bodyPanel 설계2 11](#_Toc28156141)

[그림 6. tailPanel 설계 11](#_Toc28156142)

[그림 7. UDPServer 알고리즘 12](#_Toc28156143)

[그림 8. UDPClient 알고리즘 14](#_Toc28156144)

[그림 9. TCPServer 알고리즘 14](#_Toc28156145)

[그림 10. TCPServerHandler 알고리즘 15](#_Toc28156146)

[그림 11. TCPClient 알고리즘 17](#_Toc28156147)

[그림 12. 네트워크시뮬레이터 작동 순서도 18](#_Toc28156148)

[그림 13. 초기 설정화면 21](#_Toc28156149)

[그림 14 UDP 통신 모니터링 화면 22](#_Toc28156150)

[그림 15. TCP 통신 모니터링 화면 23](#_Toc28156151)

[그림 16. 테스트파일 저장 24](#_Toc28156152)

[그림 17. 테스트파일 불러오기 25](#_Toc28156153)

[그림 18. 시간간격과 보낼 개수 설정 25](#_Toc28156154)

[그림 19. TCP통신 연결 상태 알림 26](#_Toc28156155)

[그림 20. 네트워크 통신 종료 27](#_Toc28156156)

[그림 21. 클라이언트 패킷 전송 반복문 28](#_Toc28156157)

[그림 22. UDPServer의 세부 알고리즘 29](#_Toc28156158)

# 1. 서론

네트워크를 이용하는 프로그램 개발 시 패킷(packet, 데이터 전송 최소 단위)의 정확한 송수신 검사가 필요하다. 현재 사용 가능한 장비로는 Smart-Bit을 포함한 다양한 장비들이 존재하는데 이런 장비를 구매하기 위해서는 많은 비용이 필요하기 때문에 보다 저렴하고 성능도 우수한, 실무환경에 적합한 장비를 소프트웨어적으로 구현하는 것이 필수적이다.

네트워크 프로그램 개발은 서버, 클라이언트가 함께 존재해야 한다. 예를 들어 서버를 개발하는 경우, 이를 검증하기 위한 클라이언트 프로그램도 필요하다. 이때 이미 검증된 네트워크 시뮬레이터를 통해 테스트한다면 디버깅(debugging, 오류 검출)을 빠르게 할 뿐만 아니라 정확한 개발에 도움이 된다.

프로그래밍언어로는 자바를 사용하였는데, 자바는 현재 안드로이드를 포함하여 실무에서 가장 광범위하게 사용되는 언어 중 하나이고 GUI(Graphic User Interface)에 강점을 가지고 있어 직관적으로 이해 가능한 프로그래밍 언어이다. 여러 통신 환경에 적용할 수 있도록 가장 많이 사용되는 전송프로토콜인 TCP(Transmission Control Protocol, 전송 제어 프로토콜), UDP(User Datagram Protocol, 사용자 데이터그램 프로토콜)를 테스트할 수 있는 환경을 구축하였다. 서버 기능 구현에서 에코(Echo, 수신된 데이터를 그대로 재전송하기)기능을 설정할 수 있도록 하였고, 클라이언트 기능 구현에서는 단순 전송기능과 수신기능을 선택적으로 사용하도록 구분하여 개발하였다. 송수신 메시지는 직관적으로 상세내용을 확인할 수 있고, 이를 저장, 불러오기를 통해 손쉽게 재사용할 수 있도록 구현하였다. 저장된 데이터를 통해 무작위(Random, 랜덤)로 메시지 전송 기능을 추가하여 실제와 유사한 환경을 만들어 전송할 수 있도록 하였다. 전송 데이터간 전송 간격, 전송 메시지 수를 사용자가 설정할 수 있도록 하여 다양한 환경 테스트를 가능하도록 하였다.

성능향상과 최적화를 위해 1000만개, 1억개 단위로 패킷을 보내 테스트를 실행하였고 이러한 수 차례의 테스트를 통해, 불필요한 코드와 소프트웨어 상의 병목현상을 배제할 수 있었다. TCP 통신의 경우 연결되었을 때와 통신이 끊겼을 경우 메시지를 통해 연결 여부를 표시하였고 본 문서에는 이러한 개발과 관련한 상세한 내용을 기술한다.

본 문서의 구성은 다음과 같다. 2장에서는 네트워크 시뮬레이터의 설계를 기술하고, 3장에서는 네트워크 시뮬레이터의 구현에 대한 상세 내용을 기술하고, 결론 및 향후 과제는 4장에서 기술한다.

# 2. 네트워크 시뮬레이터 설계

문서의 표기는 다음 표 1을 따른다.

표 . 문서의 표기

|  |  |
| --- | --- |
| 표기 | 설명 |
| 예시 | 그림 및 표상의 단어 |
| 예시 | 변수, 메서드() |
| 예시 | 클래스 |
| **ⓐ, ⓑ, ⓒ, …** | 그림이나 표상의 ⓐ, ⓑ, ⓒ,흐름이 없다. |
| **(1), (2), (3), …** | 그림이나 표상의 (1), (2), (3),흐름이 있다. |

## 2.1 외부 구조 설계

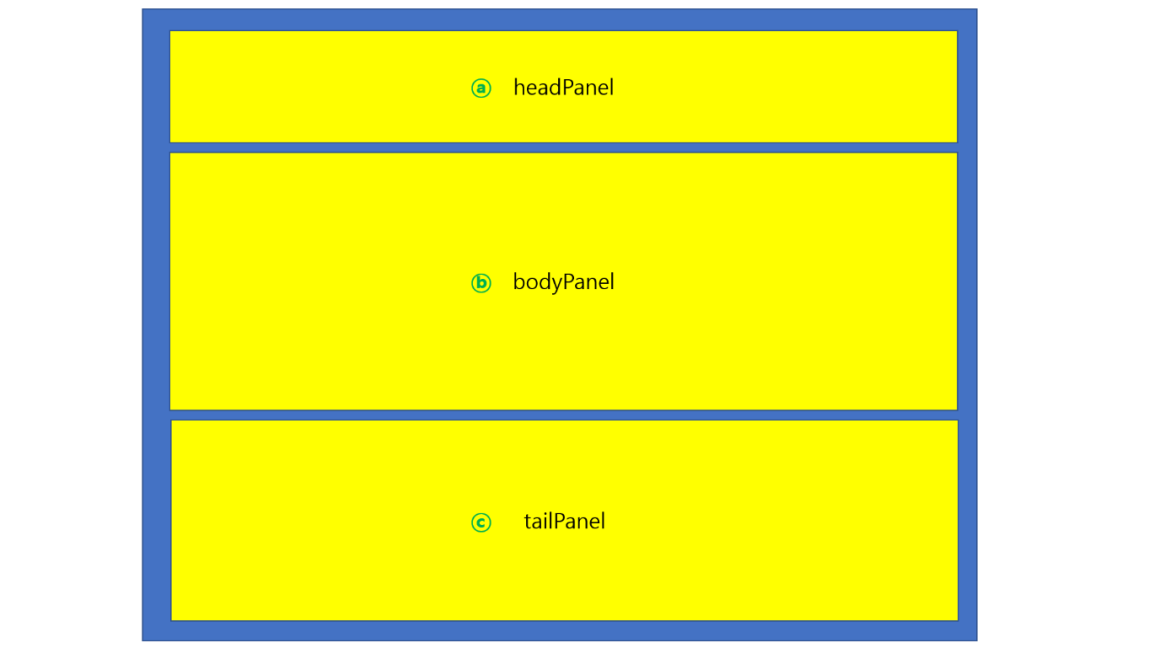


그림 . 프레임의 기초구성

그림 1은 **ⓐ, ⓑ, ⓒ** 3개의 패널로 구성된 프레임의 기초 설계를 보여준다. 전체적인 화면 구성은 중앙에 bodyPanel을 배치하고, 상단(북쪽), 하단(남쪽)에 각각 headPanel과 tailPanel을 배치하는 보더레이아웃(BorderLayout)을 기본 골격으로 한다.

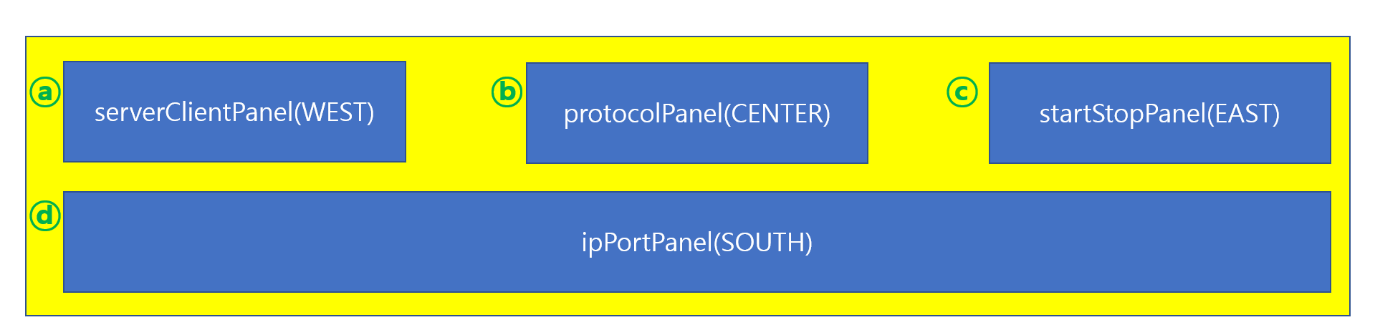


그림 . headPanel 설계1

그림 2는 headPanel의 내부구성을 보여준다. headPanel은 BorderLayout을 레이아웃으로 선택하여 **ⓐ** serverClientPanel을 서쪽(BorderLayout.WEST)에, **ⓑ** protocolPanel을 중앙(BorderLayout.CENTER)에, **ⓒ** startStopPanel을 동쪽(BorderLayout.EAST)에, **ⓓ** ipPortPanel을 남쪽(BorderLayout.SOUTH)에 배치한다.

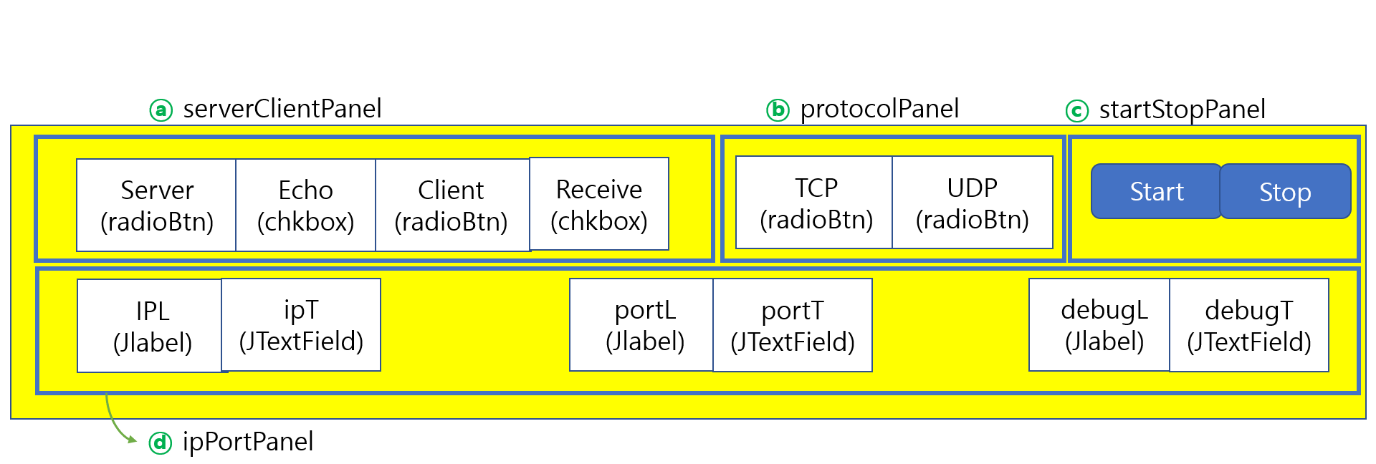


그림 . headPanel 설계2

그림 3은 headPanel의 내부 컴포넌트(component) 구성을 보여준다. **ⓐ** serverClientPanel에는 Server라는 이름의 JRadioButton과 Echo라는 이름의 JCheckBox, Client라는 이름의 JRadioButton, Receive라는 이름의 JCheckBox를 FlowLayout에 차례대로 배치했다. **ⓑ** protocolPanel에는 TCP와 UDP라는 이름으로 각각 JRadioButton를 배치했다. **ⓒ** startStopPanel은 Start라는 이름의 JButton, Stop이라는 이름의 JButton을 각각 배치했다. 마지막으로 **ⓓ** ipPortPanel에는 JLabel 객체 ipL, JTextField 객체 ipT, JLabel 객체 portL, JTextField 객체 portT, JLabel 객체 debugL, JTextField 객체 debugT를 순서대로 배치했다.

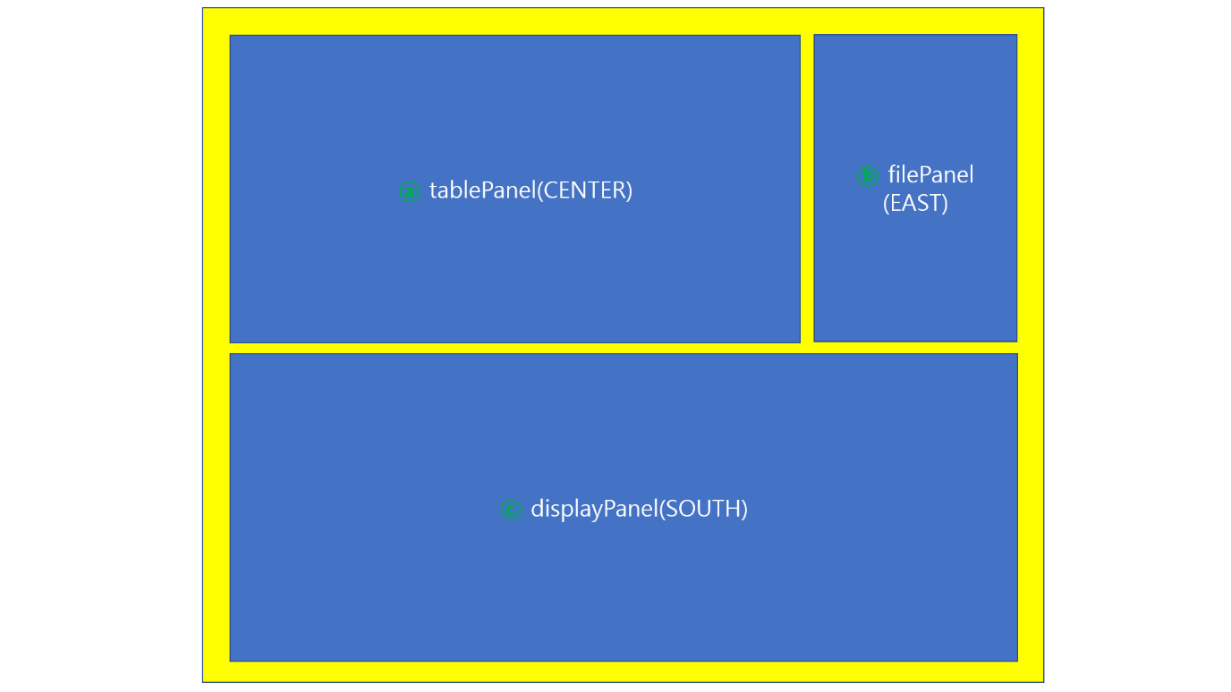


그림 . bodyPanel 설계1

그림 4는 bodyPanel의 내부 패널구성을 보여준다. bodyPanel은 BorderLayout을 레이아웃으로 선택하여 **ⓐ** tablePanel을 중앙에, **ⓑ** filePanel을 동쪽에, **ⓒ** displayPanel을 남쪽에 배치했다.

그림 5는 bodyPanel의 내부 컴포넌트(component) 구성을 보여준다. **ⓐ** displayPanel에는 JTable객체 table을 배치했다. **ⓑ** filePanel에는 Test File라는 이름의 버튼 여럿과 Save File버튼을 포함하는 JButton[]객체 testFileButtonArray를 배치했다. **ⓒ** displayPanel에는 JTextArea객체인 display를 배치했다.

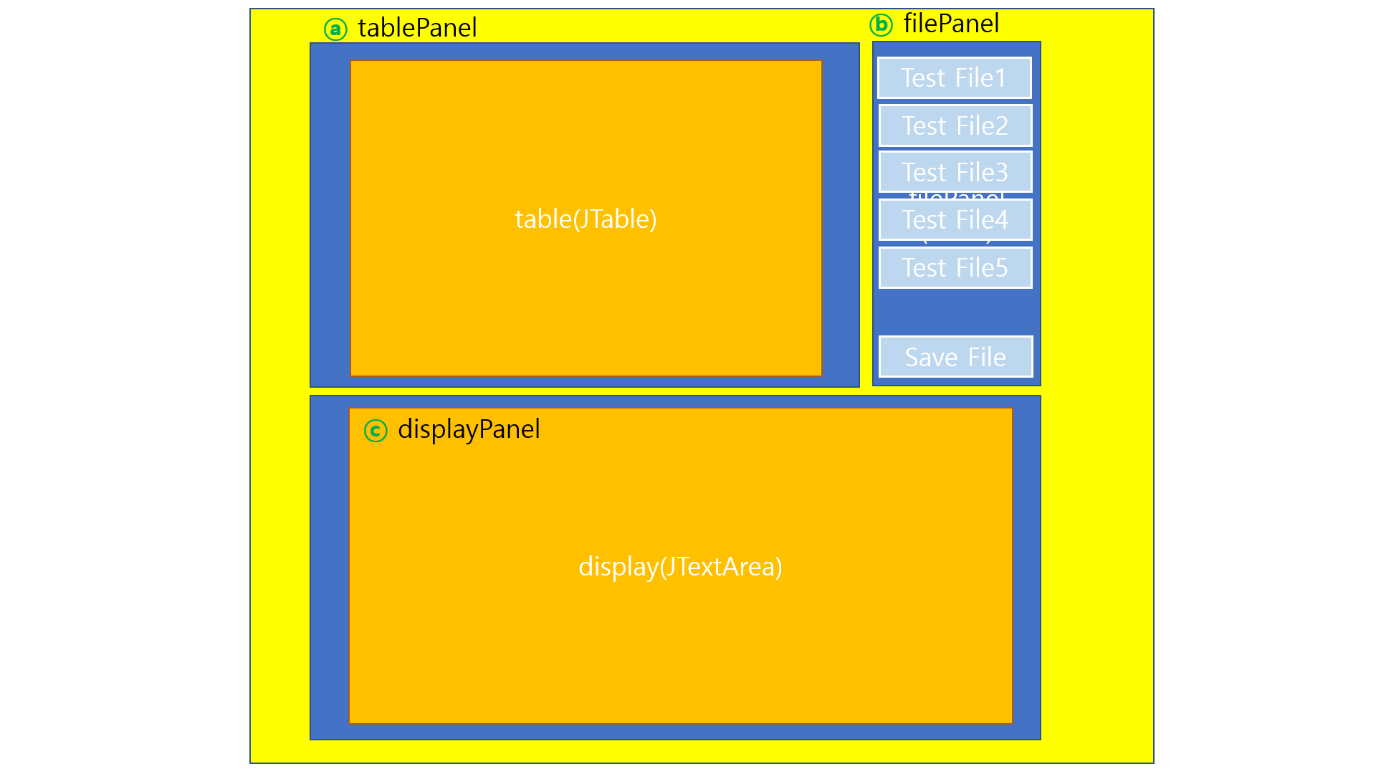


그림 . bodyPanel 설계2

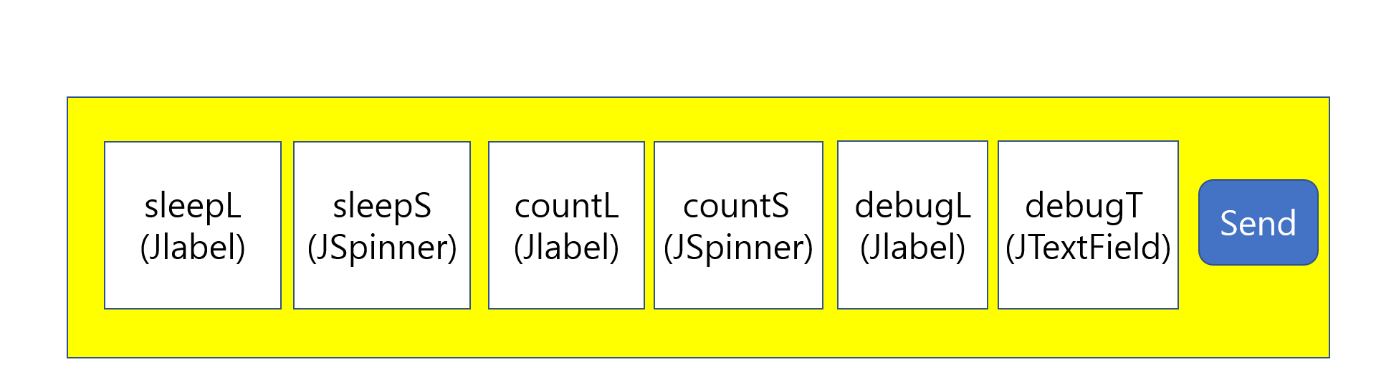


그림 . tailPanel 설계

그림 6은 tailPanel의 내부 컴포넌트(component) 구성을 보여준다. tailPanel에는 JLabel 객체 sleepL, JSpinner 객체 sleepS, JLabel 객체 countS, JSpinner 객체 countS, JCheckBox 객체 randomCheckBox, JButton 객체 sendBtn을 순서대로 배치했다.

## 2.2 UDP 서버 설계

|  |
| --- |
| public void receiveWithoutEcho()  {  while(UDP서버가 동작중인 경우)  {  //1. 패킷 받기 대기  //2. 패킷의 정보를 테이블에 넣는다.  }  }  public void receiveWithEcho()  {  while(UDP서버가 동작중인 경우)  {  //1. 패킷 받기 대기  //2. 패킷의 정보를 테이블에 담는다.  //3. 받은 정보를 패킷에 담아 클라이언트로 재전송한다.  //4. 보낸 패킷의 정보를 테이블에 담는다.  }  }  Public void returnResources()  {  //현재 사용중인 자원 반환  } |

그림 . UDPServer 알고리즘

그림 7는 UDPServer의 알고리즘을 간략하게 보여준다. 서버가 실행될 경우 Thread객체를 생성해 에코가 선택되어 있으면 receiveWithEcho()를 실행시키고, 선택되어 있지 않을 경우 receiveWithoutEcho()를 실행시킨다. 서버가 종료되어 메서드를 종료하는 경우 returnResources()를 호출하여 현재 사용중인 자원을 반환하고 종료한다.

## 2.3 UDP 클라이언트 설계

|  |
| --- |
| public void sendWithoutReceive()  {  int count = 메인 프레임에서 입력 받은 메시지 보낼 개수  while(count-- > 0)  {  if(메인 프레임의 By Random 체크했나?)  {  //체크되어 있으면 test file 중 랜덤으로 가져와 메시지 내용으로 한다.  }else{  //체크가 안 되었으면 display에 입력된 글을 메시지 내용으로 한다.  }  //1. 패킷 보내기  //2. 보낸 패킷의 정보를 테이블에 담는다.  }  }  public void sendWithReceive()  {  int count = 메인 프레임에서 입력 받은 메시지 보낼 개수  while(count-- > 0)  {  if(메인 프레임의 by random 체크했나?)  {  //체크되어 있으면 test file 중 랜덤으로 가져와 메시지 내용으로 한다.  } else{  //체크가 안 되었으면 display에 입력된 글을 메시지 내용으로 한다.  }  //1. 패킷 보내기  //2. 보낸 패킷의 정보를 테이블에 넣는다.  //3. 서버로부터 패킷을 받는다.  //4. 받은 패킷의 정보를 테이블에 넣는다.  }  }  public void returnResources()  {  //현재 사용중인 자원 반환  } |

그림 . UDPClient 알고리즘

그림 8는 UDPClient의 알고리즘을 간략하게 보여준다. 클라이언트가 실행되면 먼저 메인 프레임에서 display에 내용을 입력하거나 By Random을 체크하고, 메시지를 보낼 간격과 보낼 메시지 개수를 선택하고 sendBtn를 클릭한다. 그러면 새로운 Thread객체를 생성하여 리시브가 선택되어 있으면 sendWithReceive()를 실행시키고, 리시브가 선택되어 있지 않을 경우 sendWithoutReceive()를 실행시키게 한다. 클라이언트를 종료하는 경우 returnResources()를 호출하여 현재 사용중인 자원을 반환하고 종료한다.

## 2.4 TCP 서버 설계

|  |
| --- |
| @Override  public void run()  {  while(TCP서버가 동작중인 경우)  {  //1. 클라이언트 연결 대기  //2. TCPServerHandler 클래스를 만들어 연결된 클라이언트의 정보를 넘겨준다.  }  }  public void returnResources()  {  //현재 사용중인 자원 반환  } |

그림 . TCPServer 알고리즘

그림 9는 TCPServer의 알고리즘을 간략하게 표기했다. 서버가 실행될 경우 ServerSocket 객체를 생성하여 클라이언트가 연결될 때마다 커스텀 클래스인 TCPServerHandler의 객체를 생성하여 클라이언트의 정보를 전달한다. 서버를 종료하는 경우 returnResources()를 실행시켜 현재 사용중인 자원을 반환한다.

|  |
| --- |
| public void receiveWithoutEcho()  {  while(TCP서버가 동작중인 경우)  {  //1. 메시지 받기  //2. 클라이언트에서 온 정보를 테이블에 넣는다.  }  }  public void receiveWithEcho()  {  while(TCP서버가 동작중인 경우)  {  //1. 메시지 받기  //2. 클라이언트에서 온 정보를 테이블에 넣는다.  //3. 받은 정보를 클라이언트로 재전송한다.  //4. 전송한 내역을 테이블에 담는다.  }  }  public void returnResources()  {  //현재 사용중인 자원 반환  } |

그림 . TCPServerHandler 알고리즘

그림 10는 TCPServerHandler의 알고리즘을 간략하게 표기했다. TCPServer에서 새로운 클라이언트가 연결되면 우선 연결이 되었다는 메시지를 테이블에 저장하고 Thread객체를 생성한다. 이때 에코가 선택되어 있으면 receiveWithEcho()를 실행시키고, 에코가 선택되어 있지 않을 경우 receiveWithoutEcho()를 실행시킨다. 클라이언트 연결이 해제(NullPointerException 발생)되거나 서버가 종료되어 메서드를 종료할 경우, returnResources()를 실행시켜 현재 사용중인 자원을 반환하고 연결이 해제되었다는 메시지를 테이블에 저장한다.

## 2.5 TCP 클라이언트 설계

|  |
| --- |
| public void sendWithoutReceive()  {  int count = 메인 프레임에서 입력 받은 메시지 보낼 개수  while(count-- > 0)  {  if(메인 프레임의 By Random 체크했나?)  {  //체크되어 있으면 test file 중 랜덤으로 가져와 메시지 내용으로 한다.  }else{  //체크가 안 되었으면 display에 입력된 글을 메시지 내용으로 한다.  }  //1. 패킷 보내기  //2. 보낸 패킷의 정보를 테이블에 담는다.  }  }  public void sendWithReceive()  {  int count = 메인 프레임에서 입력 받은 메시지 보낼 개수  while(count-- > 0)  {  if(메인 프레임의 by random 체크했나?)  {  //체크되어 있으면 test file 중 랜덤으로 가져와 메시지 내용으로 한다.  } else{  //체크가 안 되었으면 display에 입력된 글을 메시지 내용으로 한다.  }  //1. 패킷 보내기  //2. 보낸 패킷의 정보를 테이블에 넣는다.  //3. 서버로부터 패킷을 받는다.  //4. 받은 패킷의 정보를 테이블에 넣는다.  }  }  public void returnResources()  {  //현재 사용중인 자원 반환  } |

그림 . TCPClient 알고리즘

그림 11는 TCPClient의 알고리즘을 간략하게 표기했다. 서버와 연결이 되면 연결이 되었다는 패킷 정보를 테이블에 표시한다. 그 다음 메인 프레임에서 display에 내용을 입력하거나 By Random을 체크하고, 메시지를 전송할 간격과 메시지 개수를 선택하여 sendBtn를 클릭한다. 그러면 새로운 Thread객체를 만들어 리시브가 선택되어 있으면 sendWithReceive()를 실행시키고, 리시브가 선택되어 있지 않을 경우 sendWithoutReceive()를 실행시키게 한다. 클라이언트가 종료되거나 서버와의 연결이 끊겨(NullPointerException 발생) 메서드를 종료할 경우, returnResources()를 실행시켜 현재 사용중인 자원을 반환하고 연결이 해제되었다는 메시지를 테이블에 저장한다.

## 2.6 시스템의 전체 흐름도



그림 . 네트워크시뮬레이터 작동 순서도

그림 12은 네트워크 시뮬레이터의 전체적인 흐름도를 보이고 있다. 그림 12에서 보는 바와 같이 구동환경(SETTING CHECK)을 설정하고 프로그램을 실행하면 프로토콜(TCP, UDP), ROLE(SERVER, CLIENT), ECHO(서버의 데이터 반송)/RECEIVE(클라이언트의 데이터 수신) 선택 여부에 따라 각각 RECEIVE DATA(데이터 수신 기능)와 SEND DATA(데이터 발송 기능)을 수행한 후 TABLE에 해당 내용을 출력하는 과정을 보여준다.

# 3. 네트워크 시뮬레이터 구현

## 3.1. 개발환경

표 2은 네트워크 시뮬레이터의 환경구성을 보여준다.

|  |  |
| --- | --- |
| 개발환경 | 환경 상세 |
| 운영체제 | Windows 10 |
| 사용언어 | 자바(JDK 11.0.3) |
| 개발도구 | Eclipse 2019-03 (4.11.0) |
| 서버 | Apache 2.2.14 |

표 . 네트워크 시뮬레이터 개발환경

## 3.2 설정화면

그림 13는 NetworkSimulatorFrame의 초기설정화면을 보여준다. 사용자는 **ⓐ**에서 시뮬레이터가 수행할 역할(서버, 클라이언트)은 물론 에코, 수신 기능도 선택, 사용할 수 있다. **ⓑ**에서는 프로토콜을 선택 가능하다. **ⓒ**에서 서버의 경우 자신의 IP(Internet Protocol)와 포트번호가 화면에 표시되고, 클라이언트의 경우 통신할 서버의 IP와 포트번호를 설정할 수 있다. **ⓓ**에서 Sleep(msec)(송신 시간간격)과 Count(송신할 메시지)의 개수도 설정 가능하다. By Random 체크박스 선택 시 미리 저장해 두었던 테스트 파일 내용을 무작위로 전송한다.

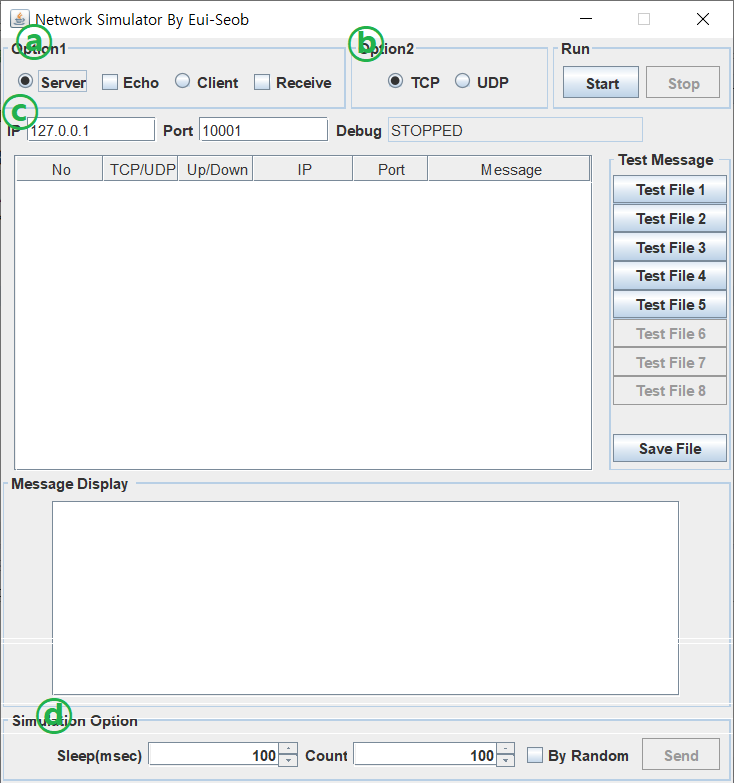


그림 . 초기 설정화면

## 3.4 모니터링 화면

그림 14는 UDP 프로토콜로 Echo와 Receive 기능 없이 통신하는 모습을 모여준다. 그림 15은 TCP 프로토콜로 Echo와 Receive 기능을 사용하여 통신하는 모습을 보여준다. 초기설정을 마치고 Start 버튼을 누르면 모니터링이 시작된다. 클라이언트에서 해당 서버로 패킷 전송을 시작하면 전송된 메시지의 정보가 테이블에 표현된다. 테이블을 통해 송수신데이터 내용이 표시되며, 오류 발생 시 디버그 창을 통하여 확인 가능하다. Echo와 Receive 기능을 선택하면 서버가 받은 데이터를 반송하고 이를 클라이언트가 다시 받는다.

송수신되는 데이터의 Message를 클릭하게 되면 Message Display화면에 송수신 메시지의 전체 내용이 표기된다. 스크롤은 자동으로 마지막 테이블 데이터를 보여주도록 설정하였다. Stop 버튼을 클릭하면 소켓 등 해당 데이터 자원을 반환하고 현재 화면상의 데이터는 유지된다. Start 버튼을 다시 클릭할 경우 통신을 재개한다.

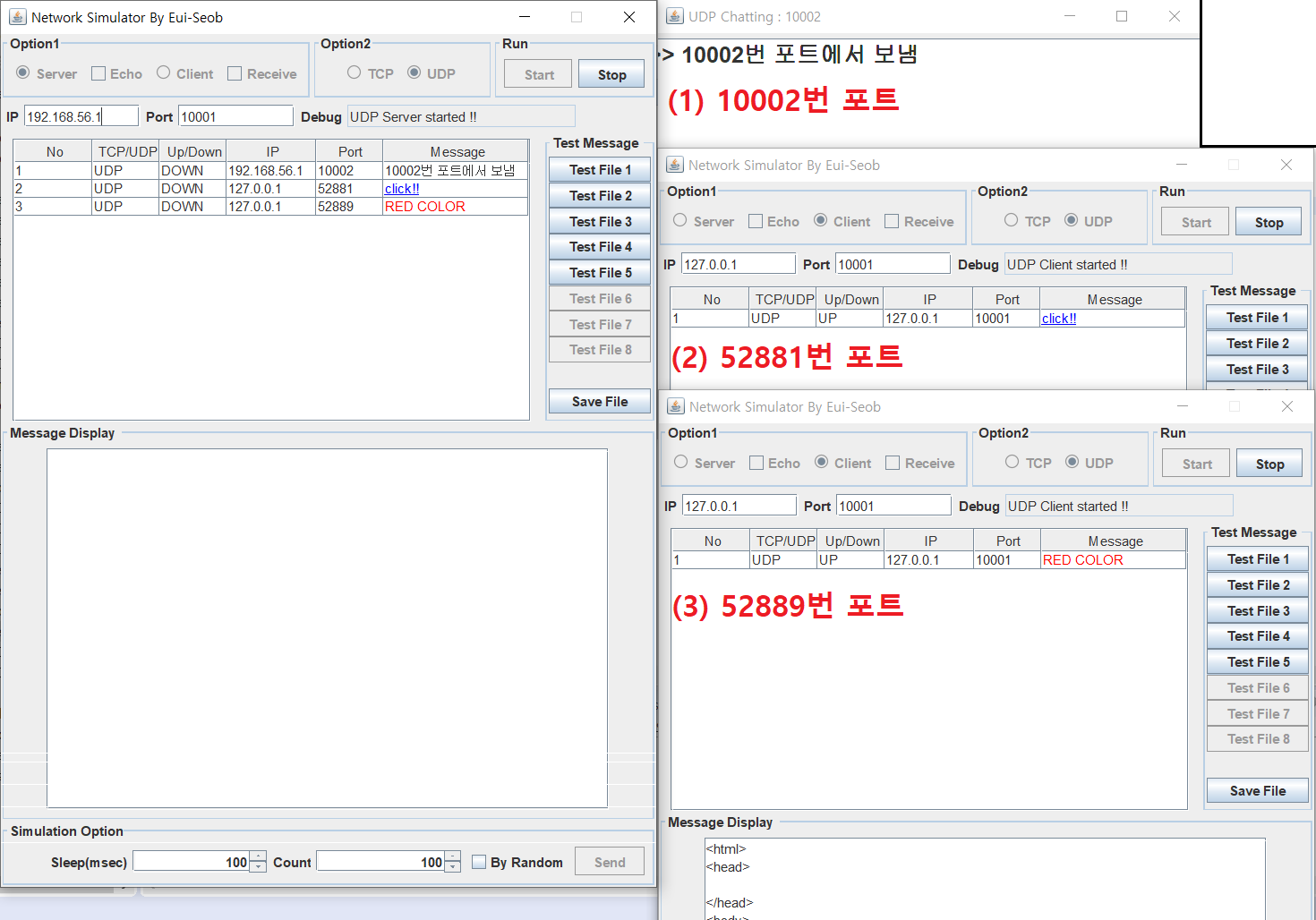


그림 UDP 통신 모니터링 화면

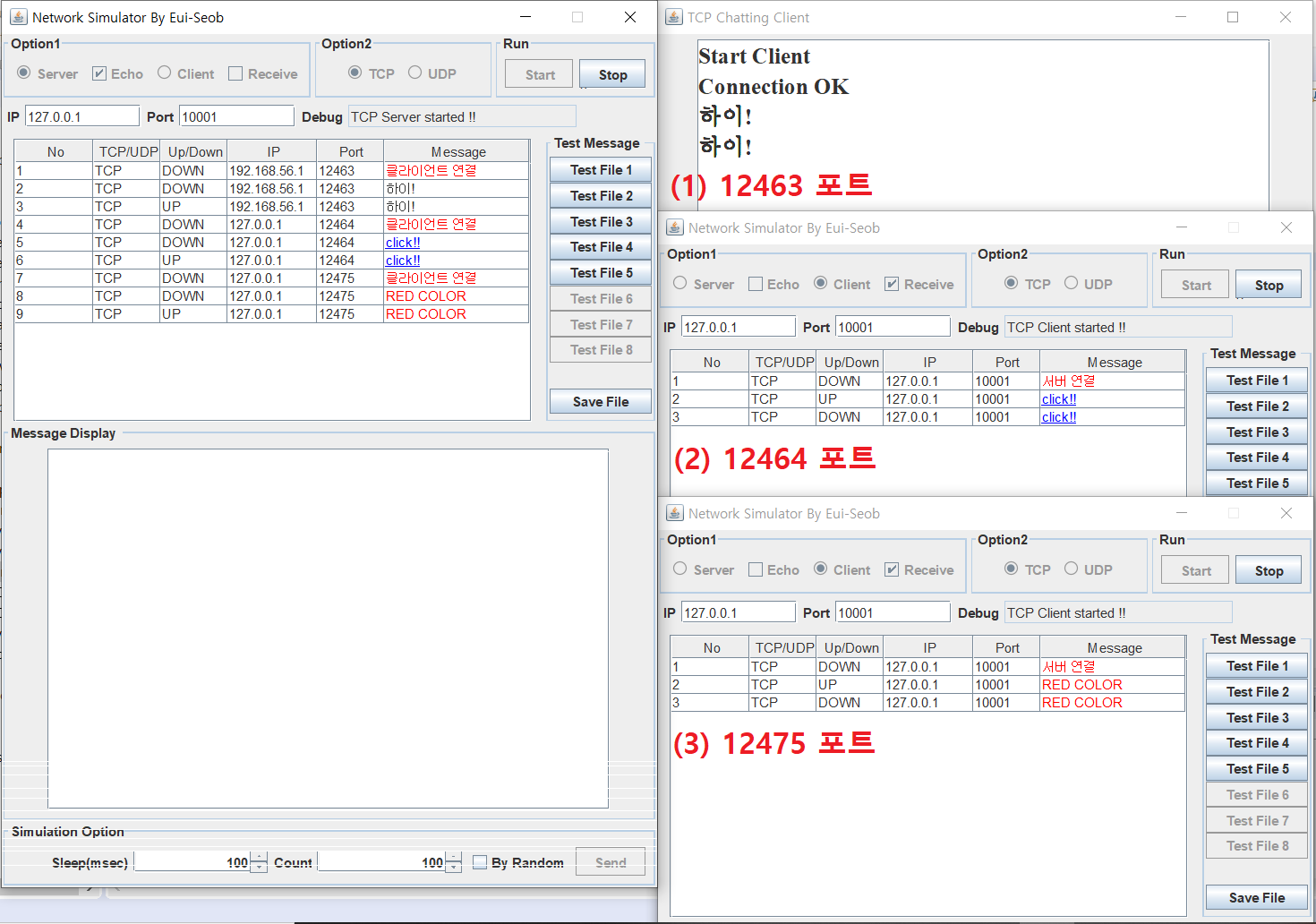


그림 . TCP 통신 모니터링 화면

## 3.5 테스트 파일의 저장과 랜덤 발송

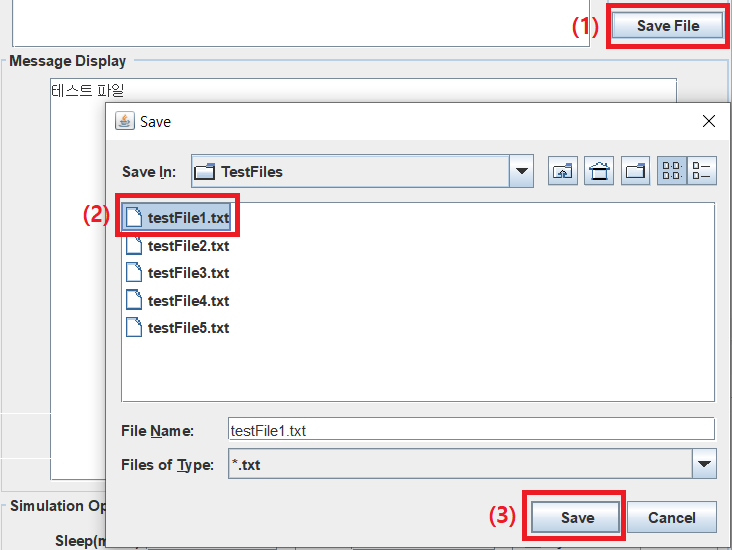


그림 . 테스트파일 저장

그림 16는 테스트 파일을 저장하는 방법을 순서대로 보여준다. Message Display에 작성한 메시지는 필요에 따라 Test File 1, 2, 3, 4 … 에 저장 가능하게 하였다. 메시지를 저장하기 위해서는 먼저 Message Display에 메시지를 입력하고, **(1)** Save File을 클릭한 후 **(2)** 원하는 테스트 파일을 선택하고 **(3)** Save를 클릭하여 저장된 내용을 갱신한다.

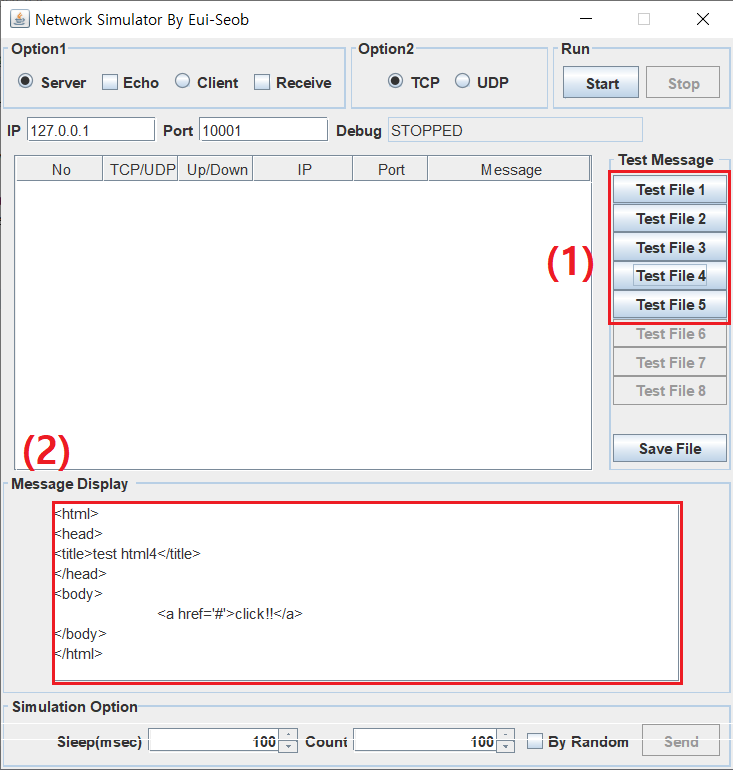


그림 . 테스트파일 불러오기

그림 17은 테스트파일을 불러오는 방법을 순서대로 보여준다. **(1)** Test File 1, 2, 3, 4 …중 하나를 클릭하면 해당 파일에 저장 되어있는 메시지가 **(2)** Message Display에 표시되도록 하였다.

## 3.6 패킷 발송 시간 간격과 발송 횟수 설정

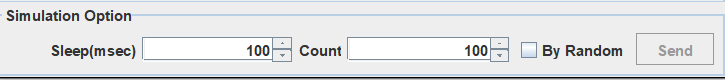


그림 . 시간간격과 보낼 개수 설정

그림 18은 메시지 시간간격과 보낼 개수를 설정하는 화면을 보여준다. Sleep(msec)은 0.001초(msec) 단위 설정으로 1이 증가하면 메시지 발송 사이의 간격이 0.001초 증가한다. Count는 전송할 메시지의 개수를 설정하는 도구이며, 해당 개수만큼 메시지가 전송된다. By Random 체크박스를 선택하고 보내기를 클릭하면 그림 16에서 저장한 메시지들을 무작위로 선택해서 보낼 수 있다. 이는 동일한 데이터를 반복적으로 보내게 되면 로컬리티의 특성으로 인해 시스템 성능을 정확히 측정할 수 없는 문제를 개선하려는 의도이다.

## 3.7 TCP 통신 시 클라이언트 연결 유무 표시

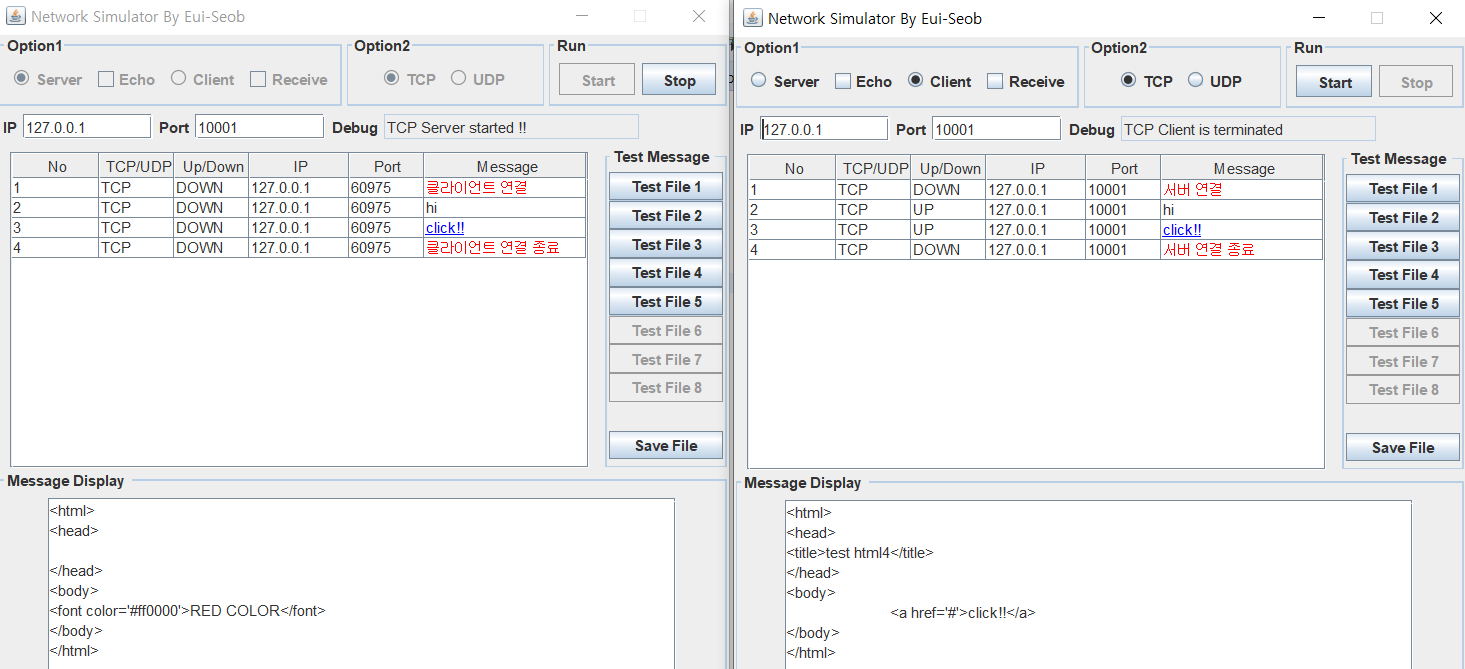


그림 . TCP통신 연결 상태 알림

그림 19은 메시지 시간간격과 보낼 개수를 설정하는 화면을 보여준다. TCP 통신 시 서버가 준비된 상태에서 클라이언트가 연결되면 HTML로 서버 측엔 “클라이언트 연결” 메시지를, 클라이언트 측엔 “서버 연결” 메시지를 TABLE에 보여준다. 반대로 연결이 끊어졌을 경우 서버에는 “클라이언트 연결 종료” 메시지를, 클라이언트 측에는 “서버 연결 종료” 메시지를 보여준다.

## 3.8 네트워크 연결 해제 시 소켓과 입출력 클래스 객체들의 우아한 종료

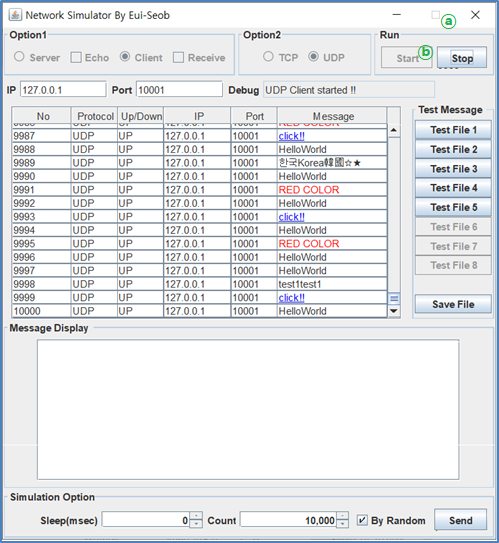


그림 . 네트워크 통신 종료

|  |
| --- |
| Public void sendWithoutReceive(){  int sendCnt = 메인에서 입력한 메시지 전송 횟수  while((sendCnt--)>0) {  //패킷 전송  }  }  Public void returnResources(){  if(BufferedReader객체가 null이 아닌 경우)  //BufferedReader객체를 닫아준다.  if(PrintWriter객체가 null이 아닌 경우)  //PrintWriter객체를 닫아준다.  if(Socket객체가 null이 아닌 경우)  //Socket객체를 닫아준다.  } |

그림 . 클라이언트 패킷 전송 반복문

그림 20는 프로그램을 종료하는 방법인 **ⓐ**와 **ⓑ**를 보여준다. 그림 21은 클라이언트에서 패킷을 앞에서 설정한 전송개수만큼 반복하여 전송하는 반복문을 보여준다. 그림 20에서 **ⓐ**나 **ⓑ**를 통해 프로그램을 종료할 때 전송중인 데이터가 있을 경우, 그림 21의 sendCnt를 0으로 만들어서 더 이상의 전송을 막고 반복문을 탈출하게 하였다. 그 다음 returnResources()라는 메서드를 호출하여 BufferedReader나 PrintWriter등의 입출력 클래스 객체나 Socket객체가 자원을 차지하고 있으면 닫아주었다. 이 일련의 과정들을 통해 불필요한 자원낭비를 최소화하였고 이 후에 발생할 수 있는 에러들을 미연에 방지하였다.

## 3.9 소프트웨어 병목현상 제거

|  |
| --- |
| public UDPServer(){  Runnable processBuffer = new Runnable() {  @Override  synchronized public void run() {  while(출력버퍼에 패킷이 남아있나?){  //1) 출력버퍼에서 패킷을 꺼낸다.  //2) 꺼낸 패킷정보를 테이블에 출력  }  };  }  public void receiveWithoutEcho(){  int bufferCnt = 0;  while(서버가 동작 중인가?){  try{  //1. 패킷을 받는다.  //2. 받은 패킷을 미리 만든 버퍼에 담는다.  bufferCnt 1증가  }catch(IOException e){ //받는 데이터가 없을 경우 설정한 시간마다 실행  if(bufferCnt != 0){  //쓰레드 풀에 만들어 놓은 쓰레드로 processBuffer를 처리  bufferCnt = 0  }  }  }  } |

그림 . UDPServer의 세부 알고리즘

그림 22는 UDPServer의 세부 알고리즘을 보여준다. 네트워크 통신의 성능을 저해하는 환경에는 대표적으로 병목현상(bottleneck)이 존재한다. 이 프로그램은 받은 패킷 정보를 테이블에 출력하는 시간이 오래 걸려, 테이블 처리가 UDP 통신의 신뢰성을 저하시키는 최대의 병목현상으로 진단하였다. 이를 해결하기 위해 ArrayList<Object>객체를 출력버퍼로 미리 지정하여 받은 패킷을 담아두고, 통신 중 받는 데이터가 없을 때 한번에 출력하게 설정하였다.

## 3.10 성능 테스트

표 3. 성능 테스트

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 보낸 패킷 | 수행 횟수 | 걸린 시간 평균 | 평균 에러개수 | 에러율 |
| 1천개 | 10회 | 1초 미만 | 0개 | 0% |
| 1만개 | 10회 | 1초 미만 | 0개 | 0% |
| 10만개 | 10회 | 3초 미만 | 24.1개 | 0.0241% |

표 3은 성능테스트의 수행 결과를 보여준다. 1천개, 1만개 단위로 패킷을 전송했을 때 에러 검출이 적었다. 다만 10만개이상 패킷 전송부터 에러 검출이 빈번해지기 시작했고, 100만개 이상씩 보낼 때는 일정 수행 횟수가 넘어가면 테이블에 담는 과정에서 이클립스 자체의 메모리 부족 에러가 발생했다. 이클립스의 스택 크기를 넓히는 등 여러 방법을 시도해봤지만 해결되지 않았다. 이는 추후 해결해야 할 문제이다.

# 4. 결론 및 향후 과제

네트워크 통신을 할 때 서버, 클라이언트가 패킷을 정확하게 송수신하는지 검사할 필요가 있는데, Smart-Bit 같은 장비를 사용하는 것은 비용적인 부담이 된다. 이와 같은 문제를 해결하기 위해, 네트워크 통신 프로그램 개발과정에서 성능과 안정성을 모두 테스트할 수 있는 프로그램인 네트워크 시뮬레이터를 개발하였다.

이 프로그램은 프로토콜, 서버/클라이언트, 에코/리시브 각각의 선택지에 따라 알맞은 역할을 수행할 수 있었고 부가적으로 메시지를 일정시간 간격으로, 개수를 지정하여 보낼 수 있는 기능과 통신 오류 발생 시 오류메시지를 출력하는 기능도 지원한다.

실제와 같은 환경으로 메시지를 보내기 위해 테스트 파일을 저장할 수 있는 기능을 추가하였고, 저장한 파일들을 무작위로 보낼 수 있는 기능도 구현하였다. 메시지를 보내는 중간에 프로그램을 종료해도 미리 설계된 알고리즘으로 사용중인 자원들을 모두 반환하고 안정적으로 종료될 수 있게 구성하였다.

이 프로그램은 실무에서 자바 소켓 통신을 개발할 때 좋은 테스트 툴로 사용될 수 있을 것이다. 다만 보완해야 될 점으로는 패킷 통신의 성능보다 안정성과 시각적으로 정확하게 확인할 수 있는 GUI(Graphic User Interface)에 비중을 보다 많이 두었기 때문에, 성능면에서 전문적인 장비보다 많은 시간을 소요하게 되는 점을 고려할 필요가 있다.

참고문헌

[1] “웹사이트의 제목”, <http://www.daum.net/a.php>, 2019.09

[2] “자바 프로그래밍 기초”, 홍길동, 홍대감 역, 우리출판사 2019.09