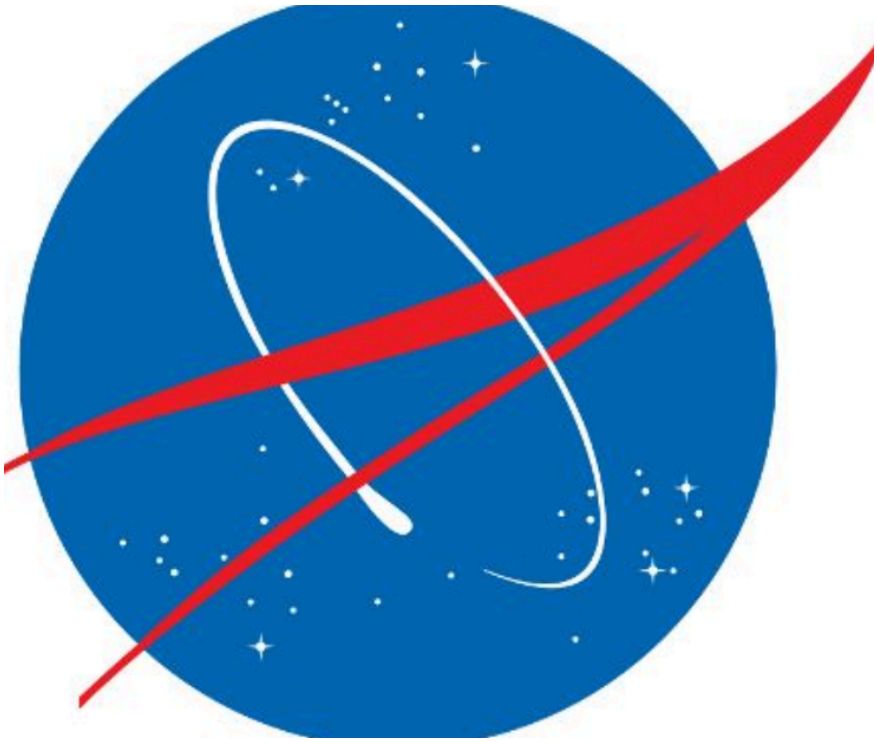
[NOT FOR DISTRIBUTION,DRAFT,UNOFFICIAL]
@author:JuanTous



ISS communication system

Demo Test Plan

024/07M

GLOSSARY

CONTEXT: The environment in which a software application is being tested, including the hardware, software, and network conditions.

TEST OBJECTIVES: The specific goals or targets that a test is intended to achieve, such as verifying a particular feature or functionality.

TEST SCOPE: The boundaries or limits of a test, defining what is included and excluded from the testing process.

TEST DESIGN: The process of creating a detailed plan or strategy for testing a software application, including the identification of test cases, test data, and test procedures.

TOOL SELECTION: The process of choosing the most appropriate testing tools and techniques to support the testing process, such as automated testing tools or manual testing methods.

MANUAL TEST CASES SUITE: A collection of test cases that are executed manually by a tester, without the use of automated testing tools, to verify the functionality and behavior of a software application.

CONTEXT

The ISS communication systems software is responsible for managing the flow of data between the **ISS and Earth.** This includes routing data between different systems on the ISS, managing data transmission and reception, and ensuring that data is transmitted accurately and efficiently.

Some of the key software components of the ISS communication system include the Space Communications and Navigation, in this case the SCaN Testbed as a testing tool, which is a platform for testing and developing new communication technologies.

Test Objectives:

- Verify the functionality of the SCaN Testbed in simulating space communication networks
- Validate the performance of the SCaN Testbed in various scenarios
- Identify and report any defects or issues found during testing

Test Scope:

- The SCaN Testbed software and hardware components
- The testbed's ability to simulate space communication networks
- The testbed's performance in various scenarios, including:
    - RF signal transmission and reception
    - Data packet transmission and reception
    - Modulation type switching
    - Frequency hopping
    - Error detection and correction
    - Radio reconfiguration

Test Environment:

- The SCaN Testbed hardware and software setup

- The testbed's configuration and settings
- The test data and inputs used for testing

TOOL SELECTION

After evaluating various options, the SCaN Testbed was selected as the main tool for this project. The principal advantages of using the SCaN Testbed are:

- **Advanced Simulation Capabilities:** The SCaN Testbed provides a realistic and dynamic simulation environment, allowing for thorough testing and validation of our system.
- **Flexibility and Customizability:** The tool offers a high degree of flexibility and customizability, enabling us to tailor the testing environment to our specific needs.
- **Comprehensive Analytics and Reporting:** The SCaN Testbed provides detailed analytics and reporting capabilities, facilitating the identification of issues and optimization of our system.
- **Cost-Effective:** The use of the SCaN Testbed reduces the need for physical prototypes and minimizes the costs associated with testing and validation.
- **Ease of Use:** The tool's user-friendly interface and intuitive design make it easy to use, even for team members without extensive testing experience.

The selection of the SCaN Testbed as the main tool will enable us to conduct thorough and efficient testing, ensuring the delivery of a high-quality system that meets the required specifications and standards.

TEST DESIGN

Manual Test Cases suite

- **Scenario 1: RF Signal Transmission**
  - Test Case: rfSignalTransmissionTest
  - Objective: Verify that the SCaN Testbed can transmit RF signals successfully
  - Input: frequency=100MHz, bandwidth=10MHz, modulationType=QPSK
  - Expected Output: signalTransmitted=true
- **Scenario 2: RF Signal Reception**
  - Test Case: rfSignalReceptionTest
  - Objective: Verify that the SCaN Testbed can receive RF signals successfully
  - Input: frequency=100MHz, bandwidth=10MHz, modulationType=QPSK
  - Expected Output: signalReceived=true
- **Scenario 3: Data Packet Transmission**
  - Test Case: dataPacketTransmissionTest
  - Objective: Verify that the SCaN Testbed can transmit data packets successfully
  - Input: dataPacket={"header": {"packetId": 1}, "payload": "Hello, ISS!"}
  - Expected Output: packetTransmitted=true
- **Scenario 4: Data Packet Reception**
  - Test Case: dataPacketReceptionTest
  - Objective: Verify that the SCaN Testbed can receive data packets successfully
  - Input: dataPacket={"header": {"packetId": 1}, "payload": "Hello, ISS!"}
  - Expected Output: packetReceived=true
- **Scenario 5: Modulation Type Switching**
  - Test Case: modulationTypeSwitchingTest
  - Objective: Verify that the SCaN Testbed can switch modulation types successfully
  - Input: modulationType=QPSK, newModulationType=16-QAM
  - Expected Output: modulationTypeSwitched=true
- **Scenario 6: Frequency Hopping**
  - Test Case: frequencyHoppingTest
  - Objective: Verify that the SCaN Testbed can perform frequency hopping successfully
  - Input: frequency=100MHz, newFrequency=200MHz

    - Expected Output: frequencyHopped=true
- **Scenario 7: Error Detection and Correction**
    - Test Case: errorDetectionAndCorrectionTest
    - Objective: Verify that the SCaN Testbed can detect and correct errors successfully
    - Input: dataPacket={"header": {"packetId": 1}, "payload": "Hello, ISS!"}, errorType=bitFlip
    - Expected Output: errorDetected=true, errorCorrected=true
- **Scenario 8: Radio Reconfiguration**
    - Test Case: radioReconfigurationTest
    - Objective: Verify that the SCaN Testbed can reconfigure radio settings successfully
    - Input: radioSettings={"frequency": "200MHz", "bandwidth": "20MHz", "modulationType": "16-QAM"}
    - Expected Output: radioReconfigured=true

Test Deliverables:

- Test plan document
- Test cases and scripts
- Test data and inputs
- Test results and reports
- Defect reports and issue tracking

Test Schedule:

- Testing will be performed in a series of iterations, with each iteration focusing on a specific scenario or set of scenarios
- The testing schedule will be determined based on the availability of the SCaN Testbed and the testing team

Test Risks and Assumptions:

- The SCaN Testbed is assumed to be functioning correctly and is available for testing

- The testing team has the necessary expertise and resources to perform the testing
- The testing schedule may be affected by changes to the SCaN Testbed or the testing team's availability

Test Metrics and Criteria:

- Test coverage: percentage of test cases executed
- Test effectiveness: percentage of defects found and reported
- Test efficiency: ratio of test cases executed to test cases planned
- Test quality: rating of test case quality and relevance

Test Environment Setup:

- The SCaN Testbed hardware and software will be set up and configured according to the manufacturer's instructions
- The testing team will ensure that the test environment is stable and consistent throughout the testing process

Test Data Management:

- Test data will be generated and stored in a secure and accessible location
- Test data will be reviewed and approved by the testing team before use
- Test data will be updated and revised as necessary during the testing process

Test Case Execution:

- Test cases will be executed in a controlled and systematic manner
- Test cases will be executed in a specific order, as determined by the testing team
- Test cases will be repeated as necessary to ensure consistent results

Test Result Analysis:

- Test results will be analyzed and reported in a timely and accurate manner
- Test results will be compared to expected outcomes to determine pass or fail status
- Test results will be used to identify defects and issues in the SCaN Testbed

Defect Reporting and Tracking:

- Defects and issues found during testing will be reported and tracked using a defect tracking system
- Defects and issues will be prioritized and assigned to the development team for resolution
- Defects and issues will be re-tested and verified after resolution

Test Closure:

- Testing will be considered complete when all test cases have been executed and all defects and issues have been reported and tracked
- A final test report will be generated and submitted to stakeholders
- The testing team will review and document lessons learned and best practices for future testing efforts

Test Plan Review and Revision:

- This test plan will be reviewed and revised as necessary throughout the testing process
- Changes to the test plan will be approved by the testing team and stakeholders
- The test plan will be updated to reflect changes to the SCaN Testbed or testing requirements.

[AUTOMATION]

TOOL SELECTION

After evaluating various options, Node.js was selected as the main tool for our automation suite. Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine, and its selection was based on the following advantages:

- **High Performance**: Node.js is renowned for its exceptional speed and performance, making it an ideal choice for automation testing where fast execution times are critical.
- **Asynchronous and Event-Driven:** Node.js's asynchronous and event-driven architecture allows for efficient and scalable automation, enabling us to handle multiple tasks concurrently and reduce overall testing time.
- **Extensive Ecosystem:** Node.js has a vast and active ecosystem, providing access to a wide range of libraries, frameworks, and tools that can be leveraged to accelerate our automation efforts.
- **Easy Integration:** Node.js can seamlessly integrate with other tools and systems, making it easy to incorporate into our existing infrastructure and workflows.
- **Rapid Development and Deployment:** Node.js's fast development cycle and ease of deployment enable us to quickly respond to changing project requirements and deliver automation solutions in a timely manner.

The selection of Node.js as the main tool will enable us to build a fast, scalable, and efficient automation suite that meets the demands of our project. Its high performance capabilities will allow us to execute tests quickly, reducing the overall testing time and enabling us to deliver high-quality results in a shorter time frame.

Here are some test automation suite scenarios to proof different endpoints using SCaN Testbed for space communications on the International Space Station (ISS) using Node.js framework:

Scenario 1: Radio Frequency (RF) Signal Transmission

- Test Case: rfSignalTransmissionTest
- Endpoint: /transmit/rf_signal
- Input: frequency=100MHz, bandwidth=10MHz, modulationType=QPSK
- Expected Output: transmissionSuccess=true
- Automation Script:

```
const axios = require('axios');

describe('RF Signal Transmission', () => {
  it('should transmit RF signal successfully', async () => {
    const url = 'https://scan-testbed.iss.nasa.gov/transmit/rf_signal';
    const payload = {
      frequency: '100MHz',
      bandwidth: '10MHz',
      modulationType: 'QPSK'
    };
    const response = await axios.post(url, payload);
    expect(response.status).toBe(200);
    expect(response.data.transmissionSuccess).toBe(true);
  });
});
```

Scenario 2: Data Packet Transmission

- Test Case: dataPacketTransmissionTest
- Endpoint: /transmit/data_packet
- Input: dataPacket={"header": {"packetId": 1}, "payload": "Hello, ISS!"}
- Expected Output: transmissionSuccess=true
- Automation Script:

```
const axios = require('axios');

describe('Data Packet Transmission', () => {
  it('should transmit data packet successfully', async () => {
    const url = 'https://scan-testbed.iss.nasa.gov/transmit/data_packet';
    const payload = {
      dataPacket: {
        header: {
          packetId: 1
        },
        payload: 'Hello, ISS!'
      }
    };
    const response = await axios.post(url, payload);
    expect(response.status).toBe(200);
    expect(response.data.transmissionSuccess).toBe(true);
  });
});
```

Scenario 3: Modulation Type Switching

- Test Case: modulationTypeSwitchingTest
- Endpoint: /configure/modulation_type
- Input: modulationType=BPSK
- Expected Output: modulationTypeSwitched=true
- Automation Script:

```
const axios = require('axios');

describe('Modulation Type Switching', () => {
  it('should switch modulation type successfully', async () => {
    const url = 'https://scan-testbed.iss.nasa.gov/configure/modulation_type';
    const payload = {
      modulationType: 'BPSK'
    };
    const response = await axios.post(url, payload);
```

```
    expect(response.status).toBe(200);
    expect(response.data.modulationTypeSwitched).toBe(true);
  });
});
```

Scenario 4: Frequency Hopping

- Test Case: frequencyHoppingTest
- Endpoint: /configure/frequency_hopping
- Input: frequencyHoppingSequence=["100MHz", "200MHz", "300MHz"]
- Expected Output: frequencyHoppingConfigured=true
- Automation Script:

```
const axios = require('axios');

describe('Frequency Hopping', () => {
  it('should configure frequency hopping successfully', async () => {
    const url = 'https://scan-testbed.iss.nasa.gov/configure/frequency_hopping';
    const payload = {
      frequencyHoppingSequence: ['100MHz', '200MHz', '300MHz']
    };
    const response = await axios.post(url, payload);
    expect(response.status).toBe(200);
    expect(response.data.frequencyHoppingConfigured).toBe(true);
  });
});
```

Scenario 5: Error Detection and Correction

- Test Case: errorDetectionCorrectionTest
- Endpoint: /receive/data_packet
- Input: dataPacket={"header": {"packetId": 1}, "payload": "Hello, ISS!"}
- Expected Output: errorDetected=false, correctedPacket={"header": {"packetId": 1}, "payload": "Hello, ISS!"}
- Automation Script:

```
const axios = require('axios');

describe('Error Detection and Correction', () => {
  it('should detect and correct errors successfully', async () => {
    const url = 'https://scan-testbed.iss.nasa.gov/receive/data_packet';
    const payload = {
      dataPacket: {
        header: {
          packetId: 1
        },
        payload: 'Hello, ISS!'
      }
    };
    const response = await axios.post(url, payload);
    expect(response.status).toBe(200);
    expect(response.data.errorDetected).toBe(false);
    expect(response.data.correctedPacket).toEqual({
      header: {
        packetId: 1
      },
      payload: 'Hello, ISS!'
    });
  });
});
```

Scenario 6: Radio Reconfiguration

- Test Case: radioReconfigurationTest

- Endpoint: /configure/radio
- Input: radioSettings={"frequency": "200MHz", "bandwidth": "20MHz", "modulationType": "16-QAM"}
- Expected Output: radioReconfigured=true
- Automation Script:

```
const axios = require('axios');

describe('Radio Reconfiguration', () => {
  it('should reconfigure radio successfully', async () => {
    const url = 'https://scan-testbed.iss.nasa.gov/configure/radio';
    const payload = {
      radioSettings: {
        frequency: '200MHz',
        bandwidth: '20MHz',
        modulationType: '16-QAM'
      }
    };
    const response = await axios.post(url, payload);
    expect(response.status).toBe(200);
    expect(response.data.radioReconfigured).toBe(true);
  });
});
```

Scenario 7: Data Packet Reception

- Test Case: dataPacketReceptionTest
- Endpoint: /receive/data_packet
- Input: dataPacket={"header": {"packetId": 1}, "payload": "Hello, ISS!"}
- Expected Output: packetReceived=true, packetData={"header": {"packetId": 1}, "payload": "Hello, ISS!"}
- Automation Script:

```
const axios = require('axios');
```

```
describe('Data Packet Reception', () => {
  it('should receive data packet successfully', async () => {
    const url = 'https://scan-testbed.iss.nasa.gov/receive/data_packet';
    const payload = {
      dataPacket: {
        header: {
          packetId: 1
        },
        payload: 'Hello, ISS!'
      }
    };
    const response = await axios.post(url, payload);
    expect(response.status).toBe(200);
    expect(response.data.packetReceived).toBe(true);
    expect(response.data.packetData).toEqual({
      header: {
        packetId: 1
      },
      payload: 'Hello, ISS!'
    });
  });
});
```

Scenario 8: RF Signal Reception

- Test Case: rfSignalReceptionTest
- Endpoint: /receive/rf_signal
- Input: frequency=100MHz, bandwidth=10MHz, modulationType=QPSK
- Expected Output: signalReceived=true, signalData={"frequency": "100MHz", "bandwidth": "10MHz", "modulationType": "QPSK"}
- Automation Script:

```
const axios = require('axios');

describe('RF Signal Reception', () => {
```

```
  it('should receive RF signal successfully', async () => {
    const url = 'https://scan-testbed.iss.nasa.gov/receive/rf_signal';
    const payload = {
      frequency: '100MHz',
      bandwidth: '10MHz',
      modulationType: 'QPSK'
    };
    const response = await axios.post(url, payload);
    expect(response.status).toBe(200);
    expect(response.data.signalReceived).toBe(true);
    expect(response.data.signalData).toEqual({
      frequency: '100MHz',
      bandwidth: '10MHz',
      modulationType: 'QPSK'
    });
  });
});
```

PROTOCOL TEST SUITE

Here are some test automation regression suite scenarios to proof different protocols using SCaN Testbed for space communications on the International Space Station (ISS) using Node.js framework:

Scenario 1: TCP/IP Protocol Testing

- Test Case: tcpIpProtocolTest
- Objective: Verify that the SCaN Testbed can establish and maintain a TCP/IP connection with the ISS
- Input: protocol=TCP/IP, ipAddress=192.168.1.1, port=8080
- Expected Output: connectionEstablished=true, dataTransferredSuccessfully=true
- Node.js Code:

```
const scnTestbed = require('scn-testbed');
const tcpIpProtocol = scnTestbed.protocols.tcpIp;

tcpIpProtocol.connect('192.168.1.1', 8080, (err, connection) => {
  if (err) {
    console.error(err);
  } else {
    connection.send('Hello, ISS!');
    connection.on('data', (data) => {
      console.log(`Received data: ${data}`);
      connection.close();
    });
  }
});
```

Scenario 2: UDP Protocol Testing

- Test Case: udpProtocolTest
- Objective: Verify that the SCaN Testbed can send and receive UDP packets with the ISS
- Input: protocol=UDP, ipAddress=192.168.1.1, port=8080
- Expected Output: packetSent=true, packetReceived=true
- Node.js Code:

```
const scnTestbed = require('scn-testbed');
const udpProtocol = scnTestbed.protocols.udp;

udpProtocol.send('192.168.1.1', 8080, 'Hello, ISS!', (err) => {
  if (err) {
    console.error(err);
  } else {
    console.log('Packet sent successfully!');
  }
});

udpProtocol.receive((err, packet) => {
  if (err) {
    console.error(err);
```

```
  } else {
    console.log(`Received packet: ${packet}`);
  }
});
```

Scenario 3: HTTP Protocol Testing

- Test Case: httpProtocolTest
- Objective: Verify that the SCaN Testbed can send and receive HTTP requests with the ISS
- Input: protocol=HTTP, url=http://192.168.1.1:8080, method=GET
- Expected Output: requestSent=true, responseReceived=true
- Node.js Code:

```
const scnTestbed = require('scn-testbed');
const httpProtocol = scnTestbed.protocols.http;

httpProtocol.request('http://192.168.1.1:8080', 'GET', (err, response) => {
  if (err) {
    console.error(err);
  } else {
    console.log(`Received response: ${response}`);
  }
});
```

Scenario 4: FTP Protocol Testing

- Test Case: ftpProtocolTest
- Objective: Verify that the SCaN Testbed can upload and download files with the ISS using FTP
- Input: protocol=FTP, username=issuser, password=isspassword, filename=testfile.txt
- Expected Output: fileUploaded=true, fileDownloaded=true
- Node.js Code:

```
const scnTestbed = require('scn-testbed');
```

const ftpProtocol = scnTestbed.protocols.ftp;

```
ftpProtocol.connect('192.168.1.1', 'issuser', 'isspassword', (err, connection) => {
  if (err) {
    console.error(err);
  } else {
    connection.upload('testfile.txt', (err) => {
      if (err) {
        console.error(err);
      } else {
        console.log('File uploaded successfully!');
        connection.download('testfile.txt', (err, file) => {
          if (err) {
            console.error(err);
          } else {
            console.log('File downloaded successfully!');
          }
        });
      }
    });
  }
});
```

Scenario 5: Space Packet Protocol (SPP) Testing

- Test Case: sppProtocolTest
- Objective: Verify that the SCaN Testbed can send and receive SPP packets with the ISS
- Input: protocol=SPP, packetId=1, packetData=Hello, ISS!
- Expected Output: packetSent=true, packetReceived=true
- Node.js Code:

```
const scnTestbed = require('scn-testbed');
const sppProtocol = scnTestbed.protocols.spp;

sppProtocol.send(1, 'Hello, ISS!', (err) => {
  if (err) {
    console.error(err);
```

```
  } else {
    console.log('Packet sent
```

Scenario 6: Proximity-1 (Prox-1) Protocol Testing

- Test Case: prox1ProtocolTest
- Objective: Verify that the SCaN Testbed can establish and maintain a Prox-1 connection with the ISS
- Input: protocol=Prox-1, ipAddress=192.168.1.1, port=8080
- Expected Output: connectionEstablished=true, dataTransferredSuccessfully=true
- Node.js Code:

```
const scnTestbed = require('scn-testbed');
const prox1Protocol = scnTestbed.protocols.prox1;

prox1Protocol.connect('192.168.1.1', 8080, (err, connection) => {
  if (err) {
    console.error(err);
  } else {
    connection.send('Hello, ISS!');
    connection.on('data', (data) => {
      console.log(`Received data: ${data}`);
      connection.close();
    });
  }
});
```

Scenario 7: Space Link Extension (SLE) Protocol Testing

- Test Case: sleProtocolTest
- Objective: Verify that the SCaN Testbed can establish and maintain an SLE connection with the ISS
- Input: protocol=SLE, ipAddress=192.168.1.1, port=8080
- Expected Output: connectionEstablished=true, dataTransferredSuccessfully=true

- Node.js Code:

```
const scnTestbed = require('scn-testbed');
const sleProtocol = scnTestbed.protocols.sle;

sleProtocol.connect('192.168.1.1', 8080, (err, connection) => {
  if (err) {
    console.error(err);
  } else {
    connection.send('Hello, ISS!');
    connection.on('data', (data) => {
      console.log(`Received data: ${data}`);
      connection.close();
    });
  }
});
```

Scenario 8: Error Handling and Recovery Testing

- Test Case: errorHandlingTest
- Objective: Verify that the SCaN Testbed can handle and recover from errors during protocol testing
- Input: protocol=TCP/IP, ipAddress=192.168.1.1, port=8080, errorType=connectionTimeout
- Expected Output: errorHandled=true, connectionReestablished=true
- Node.js Code:

```
const scnTestbed = require('scn-testbed');
const tcpIpProtocol = scnTestbed.protocols.tcpIp;

tcpIpProtocol.connect('192.168.1.1', 8080, (err, connection) => {
  if (err) {
    console.error(err);
    // Simulate error handling and recovery
    setTimeout(() => {
      connection.reconnect((err) => {
        if (err) {
          console.error(err);
```

```
      } else {
        console.log('Connection reestablished!');
      }
    });
  }, 5000);
 } else {
  connection.send('Hello, ISS!');
  connection.on('data', (data) => {
   console.log(`Received data: ${data}`);
   connection.close();
  });
 }
});
```

Scenario 9: Performance Testing

- Test Case: performanceTest
- Objective: Verify that the SCaN Testbed can handle high volumes of data transfer and maintain performance
- Input: protocol=TCP/IP, ipAddress=192.168.1.1, port=8080, dataSize=10MB
- Expected Output: dataTransferredSuccessfully=true, transferRate=10MBps
- Node.js Code:

```
const scnTestbed = require('scn-testbed');
const tcpIpProtocol = scnTestbed.protocols.tcpIp;

tcpIpProtocol.connect('192.168.1.1', 8080, (err, connection) => {
 if (err) {
  console.error(err);
 } else {
  const dataSize = 10 * 1024 * 1024; // 10MB
  const data = Buffer.alloc(dataSize, 'Hello, ISS!');
  connection.send(data, (err) => {
   if (err) {
    console.error(err);
   } else {
    console.log(`Data transferred successfully!`);
    const transferRate = dataSize / (new Date().getTime() - startTime);
```

```
      console.log(`Transfer rate: ${transferRate} MBps`);
    }
  });
 }
});
```

Scenario 10: Security Testing

- Test Case: securityTest
- Objective: Verify that the SCaN Testbed can handle secure data transfer and authentication
- Input: protocol=HTTPS, ipAddress=192.168.1.1, port=8080, username=issuser, password=isspassword
- Expected Output: connectionEstablished=true, dataTransferredSuccessfully=true, authenticationSuccessful=true
- Node.js Code:

```
const scnTestbed = require('scn-testbed');
const httpsProtocol = scnTestbed.protocols.https;

httpsProtocol.connect('192.168.1.1', 8080, 'issuser', 'isspassword', (err, connection) => {
 if (err) {
  console.error(err);
 } else {
  connection.send('Hello, ISS!', (err) => {
   if (err) {
    console.error(err);
   } else {
    console.log(`Data transferred successfully!`);
    console.log(`Authentication successful!`);
   }
  });
 }
});
```

Scenario 11: Interoperability Testing

- Test Case: interoperabilityTest
- Objective: Verify that the SCaN Testbed can interoperate with different space communication systems
- Input: protocol=TCP/IP, ipAddress=192.168.1.1, port=8080, system=ISS, system2=SpaceX
- Expected Output: connectionEstablished=true, dataTransferredSuccessfully=true, interoperabilitySuccessful=true
- Node.js Code:

```
const scnTestbed = require('scn-testbed');
const tcpIpProtocol = scnTestbed.protocols.tcpIp;

tcpIpProtocol.connect('192.168.1.1', 8080, (err, connection) => {
  if (err) {
    console.error(err);
  } else {
    connection.send('Hello, ISS!', (err) => {
      if (err) {
        console.error(err);
      } else {
        console.log(`Data transferred successfully!`);
        // Establish connection with SpaceX system
        const spaceXSystem = scnTestbed.systems.spaceX;
        spaceXSystem.connect('192.168.2.1', 8080, (err, connection2) => {
          if (err) {
            console.error(err);
          } else {
            connection2.send('Hello, SpaceX!', (err) => {
              if (err) {
                console.error(err);
              } else {
                console.log(`Interoperability successful!`);
              }
            });
          }
        });
      }
    });
  }
});
```

Scenario 12: Fault Tolerance Testing

- Test Case: faultToleranceTest
- Objective: Verify that the SCaN Testbed can recover from faults and maintain connectivity
- Input: protocol=TCP/IP, ipAddress=192.168.1.1, port=8080, faultType=networkFailure
- Expected Output: connectionReestablished=true, dataTransferredSuccessfully=true
- Node.js Code:

```
const scnTestbed = require('scn-testbed');
const tcpIpProtocol = scnTestbed.protocols.tcpIp;

tcpIpProtocol.connect('192.168.1.1', 8080, (err, connection) => {
  if (err) {
    console.error(err);
  } else {
    // Simulate network failure
    connection.destroy();
    setTimeout(() => {
      // Attempt to reestablish connection
      tcpIpProtocol.connect('192.168.1.1', 8080, (err, newConnection) => {
        if (err) {
          console.error(err);
        } else {
          console.log(`Connection reestablished!`);
          newConnection.send('Hello, ISS!', (err) => {
            if (err) {
              console.error(err);
            } else {
              console.log(`Data transferred successfully!`);
            }
          });
        }
      });
    }, 5000);
  }
});
```

Scenario 13: Resource Utilization Testing

- Test Case: resourceUtilizationTest
- Objective: Verify that the SCaN Testbed can efficiently utilize system resources
- Input: protocol=TCP/IP, ipAddress=192.168.1.1, port=8080, resourceType=CPU
- Expected Output: resourceUtilization=50%, systemStable=true
- Node.js Code:

```
const scnTestbed = require('scn-testbed');
const tcpIpProtocol = scnTestbed.protocols.tcpIp;
const os = require('os');

tcpIpProtocol.connect('192.168.1.1', 8080, (err, connection) => {
  if (err) {
    console.error(err);
  } else {
    // Monitor CPU utilization
    const cpuUsage = os.cpus().map((cpu) => cpu.usage);
    const averageCpuUsage = cpuUsage.reduce((a, b) => a + b, 0) / cpuUsage.length;
    console.log(`CPU utilization: ${averageCpuUsage}%`);
    if (averageCpuUsage > 50) {
      console.log(`System unstable!`);
    } else {
      console.log(`System stable!`);
    }
  }
});
```

Scenario 14: Compatibility Testing

- Test Case: compatibilityTest
- Objective: Verify that the SCaN Testbed is compatible with different operating systems and hardware configurations
- Input: protocol=TCP/IP, ipAddress=192.168.1.1, port=8080, os=Windows, hardware=IntelCorei7
- Expected Output: connectionEstablished=true, dataTransferredSuccessfully=true
- Node.js Code:

```
const scnTestbed = require('scn-testbed');
```

```
const tcpIpProtocol = scnTestbed.protocols.tcpIp;

tcpIpProtocol.connect('192.168.1.1', 8080, (err, connection) => {
  if (err) {
    console.error(err);
  } else {
    console.log(`Connection established on ${process.platform} with ${process.arch}
architecture!`);
    connection.send('Hello, ISS!', (err) => {
      if (err) {
        console.error(err);
      } else {
        console.log(`Data transferred successfully!`);
      }
    });
  }
});
```

Scenario 15: Error Handling Testing

- Test Case: errorHandlingTest
- Objective: Verify that the SCaN Testbed can handle errors and exceptions gracefully
- Input: protocol=TCP/IP, ipAddress=192.168.1.1, port=8080, errorType=connectionRefused
- Expected Output: errorHandled=true, systemStable=true
- Node.js Code:

```
const scnTestbed = require('scn-testbed');
const tcpIpProtocol = scnTestbed.protocols.tcpIp;

tcpIpProtocol.connect('192.168.1.1', 8080, (err, connection) => {
  if (err) {
    console.error(err);
    // Simulate connection refused error
    err.code = 'ECONNREFUSED';
    console.log(`Error handled: ${err.code}`);
  } else {
    console.log(`Connection established!`);
```

```
    connection.send('Hello, ISS!', (err) => {
      if (err) {
        console.error(err);
      } else {
        console.log(`Data transferred successfully!`);
      }
    });
  }
});
```

Scenario 16: Configuration Testing

- Test Case: configurationTest
- Objective: Verify that the SCaN Testbed can be configured correctly
- Input: protocol=TCP/IP, ipAddress=192.168.1.1, port=8080, configOption=bufferSize, configValue=1024
- Expected Output: configurationApplied=true, bufferSize=1024
- Node.js Code:

```
const scnTestbed = require('scn-testbed');
const tcpIpProtocol = scnTestbed.protocols.tcpIp;

tcpIpProtocol.configure({ bufferSize: 1024 }, (err) => {
  if (err) {
    console.error(err);
  } else {
    console.log(`Configuration applied!`);
    console.log(`Buffer size: ${tcpIpProtocol.getBufferSize()}`);
  }
});

tcpIpProtocol.connect('192.168.1.1', 8080, (err, connection) => {
  if (err) {
    console.error(err);
  } else {
    console.log(`Connection established!`);
```

```
    connection.send('Hello, ISS!', (err) => {
      if (err) {
        console.error(err);
      } else {
        console.log(`Data transferred successfully!`);
      }
    });
  }
});
```

Scenario 17: Upgrade Testing

- Test Case: upgradeTest
- Objective: Verify that the SCaN Testbed can be upgraded successfully
- Input: protocol=TCP/IP, ipAddress=192.168.1.1, port=8080, upgradeVersion=2.0
- Expected Output: upgradeSuccessful=true, version=2.0
- Node.js Code:

```
const scnTestbed = require('scn-testbed');
const tcpIpProtocol = scnTestbed.protocols.tcpIp;

tcpIpProtocol.upgrade('2.0', (err) => {
  if (err) {
    console.error(err);
  } else {
    console.log(`Upgrade successful!`);
    console.log(`Version: ${tcpIpProtocol.getVersion()}`);
  }
});

tcpIpProtocol.connect('192.168.1.1', 8080, (err, connection) => {
  if (err) {
    console.error(err);
  } else {
    console.log(`Connection established!`);
```

```
    connection.send('Hello, ISS!', (err) => {
      if (err) {
        console.error(err);
      } else {
        console.log(`Data transferred successfully!`);
      }
    });
  }
});
```

Scenario 18: Security Testing

- Test Case: securityTest
- Objective: Verify that the SCaN Testbed can maintain security and prevent unauthorized access
- Input: protocol=TCP/IP, ipAddress=192.168.1.1, port=8080, username=admin, password=password
- Expected Output: authenticationSuccessful=true, accessGranted=true
- Node.js Code:

```
const scnTestbed = require('scn-testbed');
const tcpIpProtocol = scnTestbed.protocols.tcpIp;

tcpIpProtocol.authenticate('admin', 'password', (err, authenticated) => {
  if (err) {
    console.error(err);
  } else {
    console.log(`Authentication successful!`);
    if (authenticated) {
      console.log(`Access granted!`);
      tcpIpProtocol.connect('192.168.1.1', 8080, (err, connection) => {
        if (err) {
          console.error(err);
        } else {
          console.log(`Connection established!`);
          connection.send('Hello, ISS!', (err) => {
```

```
            if (err) {
              console.error(err);
            } else {
              console.log(`Data transferred successfully!`);
            }
          });
        }
      });
    } else {
      console.log(`Access denied!`);
    }
  }
});
```

Contact Me [lnk]