Injeção de falhas

Ismael Coral Hoepers Heinzelmann, Marcos Tomaszewski, Matheus Paulon Novais, Sérgio Bonini

Falhas assumidas

- Perda aleatória de pacotes;
- Corrupção aleatória de dados.

Modelagem de falhas

- Inicialmente considerou-se modelar as falhas na parte do envio, porém em broadcast de rede, a falha ocorreria para todos os nós;
- Optou-se por injetar as falhas no nó destinatário das mensagens, onde cada nó pode aleatoriamente corromper ou perder um pacote de maneira individualizada;
- Com isso, para o mesmo pacote em um broadcast, um nó pode sofrer uma falha, enquanto outro nó não é afetado.

```cpp
bool Protocol::readDatagramSocket(Datagram *datagramBuff, int socketfd, sockaddr_in *senderAddr,
                                  std::vector<unsigned char> *buff, int dropChance, int corruptChance) {
    socklen_t senderAddrLen = sizeof(senderAddr);
    ssize_t bytes_received = recvfrom( fd: socketfd,  buf: buff->data(),  n: buff->size(),  flags: 0,
                                      reinterpret_cast<struct sockaddr *>(senderAddr), &senderAddrLen);
    if (bytes_received < 0)
        return false;
    buff->resize(bytes_received);
    // Generate fault returns true if the package should be dropped.
    if (generateFault(buff, dropChance, corruptChance))
        return false; // Package dropped

    unsigned int checkSum = TypeUtils::buffToUnsignedInt(*buff,  i: 20);
    unsigned int computedCheckSum = computeChecksum(buff);
    if (checkSum != computedCheckSum) {
        Logger::log( message: "Packet received is now corrupted and will not be responded.", LogLevel::FAULT);
        // Package corrupted
        return false;
    }
    bufferToDatagram( [&]*datagramBuff, *buff);
    const std::vector dataVec( first: buff->begin() + 24,  last: buff->begin() + 24 + datagramBuff->getDataLength());
    datagramBuff->setData( dataVec);
    return true;
}
```

```cpp
// Will return true if the packet should be dropped.
bool Protocol::generateFault(std::vector<unsigned char>* data, int dropChance, int corruptChance) {
    // Return true if package should be dropped.
    if (FaultInjector::returnTrueByChance(dropChance)) {
        Logger::log( message: "Packet received will be dropped and ignored.", LogLevel::FAULT);
        return true;
    }
    // If the package is not dropped, verify if the package should be corrupted.
    if (FaultInjector::returnTrueByChance(corruptChance)) {
        FaultInjector::corruptVector(data);
    }
    return false;
}
```

```cpp
void FaultInjector::corruptVector(std::vector<unsigned char>* data) {
    if (!data || data->empty()) {
        return;
    }
    std::random_device rd;
    std::mt19937 gen( sd: rd());
    // Select some indexes of the datagram data.
    std::uniform_int_distribution<> indexDis( a: 0, b: data->size() - 1);
    // Select a number of corruptions between 1 and half of the data.
    std::uniform_int_distribution<> numCorruptions( a: 1, b: data->size()/2);
    int corruptionCount = numCorruptions([&]gen);

    // Iterate over the selected corrupted bytes, inverting them.
    for (int i = 0; i < corruptionCount; ++i) {
        int index = indexDis([&]gen);
        unsigned char originalValue = (*data)[index];
        unsigned char corruptedValue = ~originalValue;
        (*data)[index] = corruptedValue;
    }
}
```

Testes

Código de teste

- Para testar as implementações realizadas nas últimas etapas, um código de teste foi implementado, o qual envia 100 mensagens sendo ela unicasts ou broadcasts;
- Os testes foram executados 5 vezes para cada perfil, para aumentar a confiança dos mesmos;
- Foi definido um grupo de **três nós** para os testes.

Unicast

```cpp
if (type == "1") {
    std::string idString = std::string();
    std::cout << "Choose which node you want to send the message:" << std::endl;
    std::cin >> idString;
    std::cin.ignore(); // Remove this if necessary
    int success = 0;
    for (int i = 0; i < 100; i++) {
        std::string message = "Node: " + std::to_string(strtol(argv[1], nullptr, 10)) + " | Message: " + std::to_string(i);
        std::vector<unsigned char> messageBytes(message.begin(), message.end());

        bool sent = rb.send(strtol(argv[1], nullptr, 10), messageBytes);
        if (sent)
            success++;
    }
    std::cout << "Success in " + std::to_string(success) + " of 100 messages." << std::endl;
    std::cout << "Messages contents:" << std::endl;
    if (idString == std::to_string(strtol(argv[1], nullptr, 10))){
        for (int i = 0; i < success; i++) {
            auto receivedMessage = rb.receive();
            std::string str(receivedMessage.second.begin(), receivedMessage.second.end());
            std::cout << str << std::endl;
        }
    }
}
```

Broadcast

```
else if (type == "2") {
    int success = 0;
    for (int i = 0; i < 100; i++) {
        std::string message = "Node: " + std::to_string(strtol(argv[1], nullptr, 10)) + " | Message: " + std::to_string(i);
        std::vector<unsigned char> messageBytes(message.begin(), message.end());
        bool sent = rb.sendBroadcast(messageBytes);
        if (sent){
            success++;
        } else {
            std::cout << "Message failed." << std::endl;
        }
    }
    std::cout << "Success in " + std::to_string(success) + " of 100 messages." << std::endl;
    std::cout << "Messages contents:" << std::endl;
    for (int i = 0; i < success; i++) {
        auto receivedMessage = rb.receive();
        std::string str(receivedMessage.second.begin(), receivedMessage.second.end());
        std::cout<<str<<std::endl;
    }
}
```

Perfis de teste

- Considerando os diversos perfis de rede, foi realizada uma pesquisa para definir perfis de rede, e assim foram decididos os perfis:
    - **1% de perda de pacote com 0% de corrupção;**
    - **1% de perda de pacote com 1% de corrupção;**

Logs

- Sistema de log implementado, com o tipo de log *FAULT*.

Drop:

```
[2024-11-11 10:29:50] [FAULT] Packet received will be dropped and ignored.
```

Corrupted:

```
[2024-11-11 10:29:50] [FAULT] Packet received is now corrupted and will not be responded.
```

Resultados

Teste perfil 1% de perda com 0% de corrupção

```
Message type: 1 for unicast test and 2 for broadcast test
2
[2024-11-11 10:12:21] [FAULT] Packet received will be dropped and ignored.
[2024-11-11 10:12:21] [FAULT] Packet received will be dropped and ignored.
[2024-11-11 10:12:21] [FAULT] Packet received will be dropped and ignored.
[2024-11-11 10:12:21] [FAULT] Packet received will be dropped and ignored.
[2024-11-11 10:12:21] [FAULT] Packet received will be dropped and ignored.
[2024-11-11 10:12:21] [FAULT] Packet received will be dropped and ignored.
[2024-11-11 10:12:21] [FAULT] Packet received will be dropped and ignored.
[2024-11-11 10:12:21] [FAULT] Packet received will be dropped and ignored.
[2024-11-11 10:12:22] [FAULT] Packet received will be dropped and ignored.
[2024-11-11 10:12:22] [FAULT] Packet received will be dropped and ignored.
[2024-11-11 10:12:22] [FAULT] Packet received will be dropped and ignored.
[2024-11-11 10:12:22] [FAULT] Packet received will be dropped and ignored.
[2024-11-11 10:12:22] [FAULT] Packet received will be dropped and ignored.
[2024-11-11 10:12:23] [FAULT] Packet received will be dropped and ignored.
[2024-11-11 10:12:23] [FAULT] Packet received will be dropped and ignored.
[2024-11-11 10:12:23] [FAULT] Packet received will be dropped and ignored.
[2024-11-11 10:12:23] [FAULT] Packet received will be dropped and ignored.
[2024-11-11 10:12:24] [FAULT] Packet received will be dropped and ignored.
[2024-11-11 10:12:24] [FAULT] Packet received will be dropped and ignored.
[2024-11-11 10:12:24] [FAULT] Packet received will be dropped and ignored.
```

Teste perfil 1% de perda com 0% de corrupção

```
Success in 100 of 100 messages.
Messages contents:
Node: 0 | Message: 0
Node: 0 | Message: 1
Node: 0 | Message: 2
Node: 0 | Message: 3
Node: 0 | Message: 4
Node: 0 | Message: 5
Node: 0 | Message: 6
Node: 0 | Message: 7
Node: 0 | Message: 8
Node: 0 | Message: 9
Node: 0 | Message: 10
Node: 0 | Message: 11
Node: 0 | Message: 12
```

Teste perfil 1% de perda com 1% de corrupção

```
Message type: 1 for unicast test and 2 for broadcast test
[2024-11-11 10:15:24] [FAULT] Packet received is now corrupted and will not be responded.
2
[2024-11-11 10:15:26] [FAULT] Packet received will be dropped and ignored.
[2024-11-11 10:15:26] [FAULT] Packet received is now corrupted and will not be responded.
[2024-11-11 10:15:26] [FAULT] Packet received is now corrupted and will not be responded.
[2024-11-11 10:15:26] [FAULT] Packet received will be dropped and ignored.
[2024-11-11 10:15:26] [FAULT] Packet received will be dropped and ignored.
[2024-11-11 10:15:27] [FAULT] Packet received is now corrupted and will not be responded.
[2024-11-11 10:15:27] [FAULT] Packet received will be dropped and ignored.
[2024-11-11 10:15:27] [FAULT] Packet received is now corrupted and will not be responded.
[2024-11-11 10:15:27] [FAULT] Packet received is now corrupted and will not be responded.
[2024-11-11 10:15:28] [FAULT] Packet received will be dropped and ignored.
[2024-11-11 10:15:28] [FAULT] Packet received will be dropped and ignored.
[2024-11-11 10:15:28] [FAULT] Packet received will be dropped and ignored.
[2024-11-11 10:15:28] [FAULT] Packet received is now corrupted and will not be responded.
[2024-11-11 10:15:28] [FAULT] Packet received will be dropped and ignored.
[2024-11-11 10:15:28] [FAULT] Packet received is now corrupted and will not be responded.
[2024-11-11 10:15:29] [FAULT] Packet received will be dropped and ignored.
[2024-11-11 10:15:29] [FAULT] Packet received will be dropped and ignored.
```

Teste perfil 1% de perda com 1% de corrupção

```
Success in 100 of 100 messages.
Messages contents:
Node: 0 | Message: 0
Node: 0 | Message: 1
Node: 0 | Message: 2
Node: 0 | Message: 3
Node: 0 | Message: 4
Node: 0 | Message: 5
Node: 0 | Message: 6
Node: 0 | Message: 7
Node: 0 | Message: 8
Node: 0 | Message: 9
Node: 0 | Message: 10
Node: 0 | Message: 11
Node: 0 | Message: 12
Node: 0 | Message: 13
Node: 0 | Message: 14
Node: 0 | Message: 15
Node: 0 | Message: 16
Node: 0 | Message: 17
Node: 0 | Message: 18
```

Demais resultados

Com objetivo de testar a biblioteca, foram realizados testes também com:

- **3% de perda de pacote e 1% de corrupção**
- **3% de perda de pacote com 2% de corrupção**

com resultados similares aos anteriores.

Teste de ordem

- Utilizando o teste de 1% de perda de pacotes com 1% de corrupção, foi realizado um teste onde dois realizam o teste de broadcast ao mesmo tempo, a fim de testar a ordem das mensagens.

Teste de ordem

```
Message content: Node: 1 | Message: 84
Message content: Node: 1 | Message: 85
Message content: Node: 0 | Message: 93
Message content: Node: 1 | Message: 86
Message content: Node: 1 | Message: 87
Message content: Node: 1 | Message: 88
Message content: Node: 0 | Message: 94
Message content: Node: 0 | Message: 95
Message content: Node: 0 | Message: 96
Message content: Node: 1 | Message: 89
Message content: Node: 0 | Message: 97
Message content: Node: 0 | Message: 98
Message content: Node: 0 | Message: 99
Message type: 1 for unicast and 2 for broadcast
Message content: Node: 1 | Message: 90
Message content: Node: 1 | Message: 91
Message content: Node: 1 | Message: 92
Message content: Node: 1 | Message: 93
Message content: Node: 1 | Message: 94
Message content: Node: 1 | Message: 95
Message content: Node: 1 | Message: 96
Message content: Node: 1 | Message: 97
Message content: Node: 1 | Message: 98
Message content: Node: 1 | Message: 99
```

```
Message content: Node: 1 | Message: 84
Message content: Node: 1 | Message: 85
Message content: Node: 0 | Message: 93
Message content: Node: 1 | Message: 86
Message content: Node: 1 | Message: 87
Message content: Node: 1 | Message: 88
Message content: Node: 0 | Message: 94
Message content: Node: 0 | Message: 95
Message content: Node: 0 | Message: 96
Message content: Node: 1 | Message: 89
Message content: Node: 0 | Message: 97
Message content: Node: 0 | Message: 98
Message content: Node: 0 | Message: 99
Message content: Node: 1 | Message: 90
Message content: Node: 1 | Message: 91
Message content: Node: 1 | Message: 92
Message content: Node: 1 | Message: 93
Message content: Node: 1 | Message: 94
Message content: Node: 1 | Message: 95
Message content: Node: 1 | Message: 96
Message content: Node: 1 | Message: 97
Message content: Node: 1 | Message: 98
Message type: 1 for unicast and 2 for broadcast
Message content: Node: 1 | Message: 99
```

Conclusão

- A biblioteca em seu estado atual possui tolerância a falhas por perda de pacote e corrupção de dados;
- Um caso excepcional foi identificado onde o fluxo do remetente é interrompido em casos com maior número de falhas foi identificado e será investigado.