# Содержание

[Содержание 1](#_Toc105341507)

[1 Передача Goose потоков и его структура 2](#_Toc105341508)

[Приложение А 6](#_Toc105341511)

# Передача Goose потоков и его структура

Обмен логической информацией терминалами между собой и АРМ (автоматизированным рабочим местом) релейщика производится по протоколу GOOSE. GOOSE (англ Generic Object Oriented Substation Event) (стандарт МЭК 61850-8-1) – протокол передачи данных о событиях на подстанции.

Устройство-отправитель передает по сети Ethernet информацию в широковещательном диапазоне. В сообщении присутствует адрес отправителя и адреса, по которым осуществляется его передача, а также значение сигнала (например, «0» или «1»). Устройство-получатель получит сообщение, а все остальные устройства его проигнорируют.

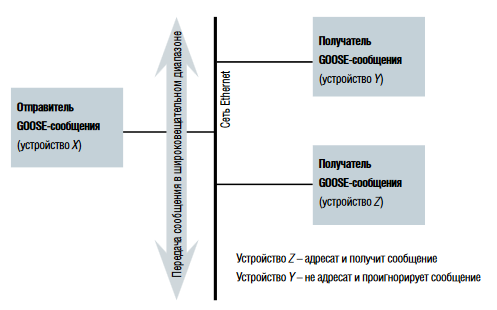


Рисунок 1 – Принцип передачи GOOSE-сообщения

Поскольку передача GOOSE-сообщений осуществляется в широковещательном диапазоне, т.е. нескольким адресатам, подтверждение факта получения адресатами сообщения отсутствует. По этой причине передача GOOSE-сообщений в установившемся режиме производится с определенной периодичностью. При наступлении нового события в системе (например, КЗ и, как следствие, пуска измерительных органов защиты) начинается спонтанная передача сообщения через увеличивающиеся интервалы времени (например, 4 мс, 8 мс, 16 мс и т.д.). Интервалы времени между передаваемыми сообщениями увеличиваются, пока не будет достигнуто предельное значение, определяемое пользователем (например, 2 с). Далее, до момента наступления нового события в системе, передача будет осуществляется именно с таким периодом. Технология повторной передачи не только гарантирует получение адресатом сообщения, но также обеспечивает контроль исправности линии связи и устройств – любые неисправности будут обнаружены по истечении максимального периода передачи GOOSE-сообщений (с точки зрения эксплуатации практически мгновенно). В случае передачи сигналов традиционным образом неисправность выявляется либо в процессе плановой проверки устройств, либо в случае неправильной работы системы РЗА.

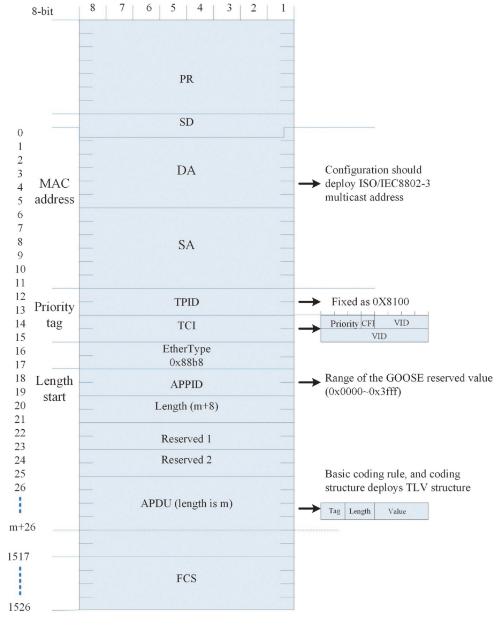


Рисунок 2 – Структура GOOSE пакета

* PR – используется для синхронизации приемопередатчиков.
* DA – адрес устройства, которому направляется сообщение.
* SA – уникальный адрес передающего интеллектуального устройства.
* Идентификатор протокола (TPID, Tag Protocol Identifier) – указывает тип использованного протокола.
* Идентификатор сообщения (Ethertype) – указывает тип сообщений.
* Идентификатор положения (APPID, Application Identifier) – служит для разделения сообщений.
* Длина данных (Length) – суммарная длина полей APPID
* Reserved 1 и reserved 2 – зарезервированные поля.
* Протокол данных (APDU, Application Protocol Data Unit) – содержит информацию.
* Контрольная сумма (Frame check sequence) – контрольное значение. С помощью контрольной суммы получатель может определить не был ли поврежден фрейм во время передачи.

Данные в сообщении GOOSE содержатся в Protocol Application Unit (PDU) и кодируются в соответствии со стандартом абстрактного синтаксического обозначения ONE (ASN.1) для сетей передачи данных и связи в открытых системах IEC 61850-8-1. PDU содержит основную информацию пакета, ниже представлены его атрибуты:

* GoCBRef – ссылка на блок управления GOOSE – это уникальное имя пути экземпляра блока управления
* TimeAllowedtoLive – время, информирует подписчиков о том, как долго ждать следующего повторения сообщения.
* DatSet – ссылки на набор данных, значения членов которого должны быть переданы.
* GoID – GOOSE ID – это атрибут, который позволяет пользователю назначить идентификацию для сообщения GOOSE.
* t (отметка времени) – время, в которое атрибут StNum был увеличен.
* StNum (номер состояния) – это счетчик, который увеличивается каждый раз при отправке сообщения GOOSE и обнаружении изменения значения в наборе данных, указанном в DatSet. Начальное значение должно быть 1. Значение 0 зарезервировано.
* SqNum – текущий порядковый номер отчетов. Он будет увеличиваться каждый раз при отправке сообщения GOOSE. После изменения StNum счетчик SqNum должен быть установлен в значение 0.
* Test (тестовый бит) – если true, сообщение и, следовательно, его значение были выданы модулем моделирования и не являются реальными значениями.
* ConfRev –Число должно представлять количество раз, когда DataSet изменялась.
* NdsCom – указывает в сообщении, что необходим некоторый ввод в эксплуатацию (требуется комиссия). Если TRUE требует дальнейшей настройки.
* NumDatSetEntries – количество записей набора данных
* allData – список определяемой пользователем информации.

При потере нескольких пакетов при передаче, эту потерю можно будет отследить по двум счётчикам StNum и SqNum.

# программная реализация Goose потоков

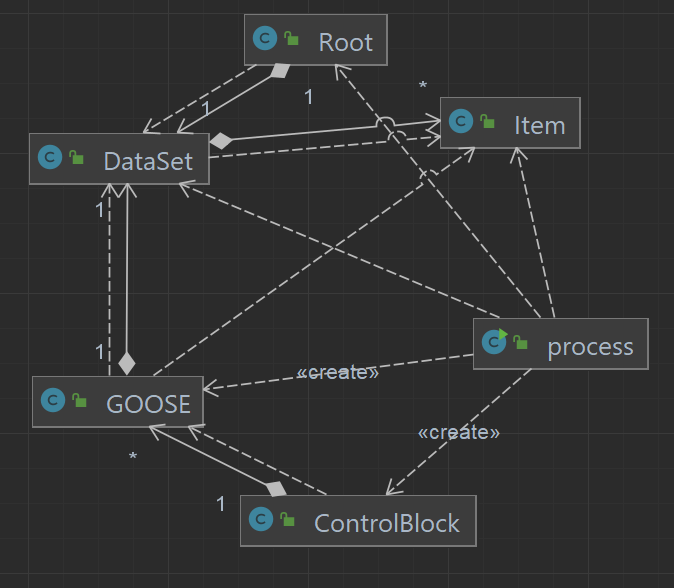


Рисунок 3 – Классы проекта

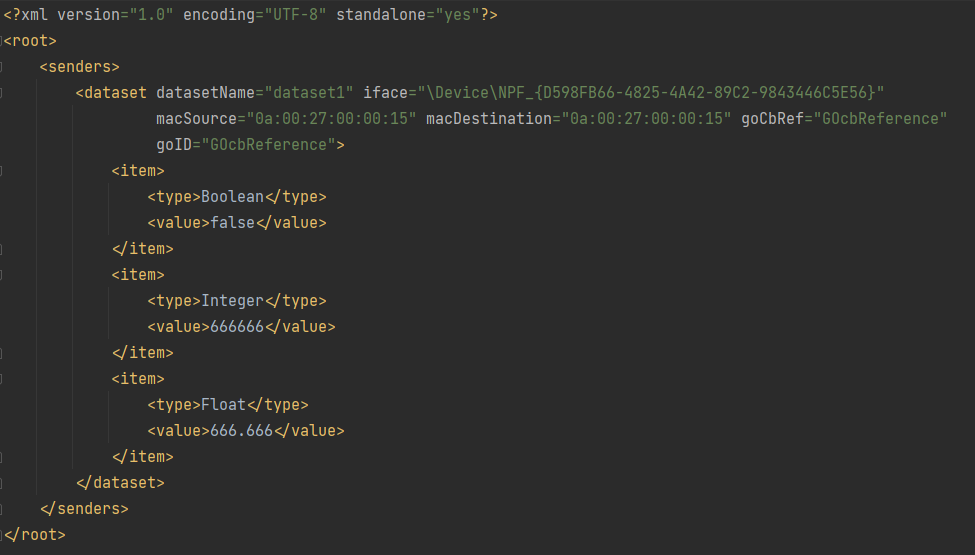


Рисунок 4 – Отправляемый XML

# Приложение А

1 import com.fasterxml.jackson.dataformat.xml.XmlMapper;

2 import lombok.SneakyThrows;

3

4 import java.io.File;

5 import java.util.ArrayList;

6 import java.util.List;

7 import java.util.concurrent.Executors;

8 import java.util.concurrent.TimeUnit;

9

10 public class process {

11

12 @SneakyThrows

13 public static void main(String[] args) {

14 XmlMapper xmlMapper = new XmlMapper();

15 Root value = xmlMapper.readValue(new File("src/main/resources/Cfg.xml"), Root.class);

16 ControlBlock controlBlock = new ControlBlock();

17 List<GOOSE> gse = new ArrayList<>();

18 controlBlock.setGooseList(gse);

19 value.getSenders().forEach(dataSet -> gse.add(new GOOSE(dataSet)));

20

21 Executors.newSingleThreadScheduledExecutor().schedule(

22 () -> {

23 value.getSenders().get(0).getItems().get(0).setValue("true");

24 value.getSenders().get(0).getItems().get(1).setValue("228");

25 value.getSenders().get(0).getItems().get(2).setValue("14.2");

26 controlBlock.changeGoose(gse.get(0));

27 }

28 , 10, TimeUnit.SECONDS);

29

30 controlBlock.sender();

31

32 }

33 }

34

1 import lombok.Data;

2

3 import java.util.ArrayList;

4 import java.util.List;

5

6 @Data

7 public class ControlBlock {

8

9 private List<GOOSE> gooseList = new ArrayList<>();

10

11 public void sender() {

12 while (!gooseList.isEmpty()) {

13 gooseList.stream()

14 .filter(GOOSE::isUnPaused)

15 .forEach(GOOSE::send);

16 }

17 }

18

19 public void changeGoose(GOOSE gse) {

20 gooseList.stream()

21 .filter(goose -> goose == gse)

22 .findFirst()

23 .ifPresent(GOOSE::setData);

24 }

25 }

26

1 import lombok.Data;

2 import lombok.SneakyThrows;

3 import org.pcap4j.core.PcapHandle;

4 import org.pcap4j.core.PcapNetworkInterface;

5 import org.pcap4j.core.Pcaps;

6 import org.pcap4j.packet.EthernetPacket;

7

8 import java.nio.ByteBuffer;

9 import java.nio.charset.StandardCharsets;

10 import java.time.Instant;

11 import java.util.List;

12

13 @Data

14 public class GOOSE implements Runnable {

15 private byte[] destination = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};

16 private byte[] source = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};

17 private final byte[] type = {(byte) 0x88, (byte) 0xB8};

18

19 private final byte[] appID = {0x00, 0x01};

20 private final byte[] length = {0x00, 0x00};

21 private final byte[] reserved1 = {0x00, 0x00};

22 private final byte[] reserved2 = {0x00, 0x00};

23 private final byte[] goosePdu = {0x61, (byte) 0x81, (byte) 0x8A};

24 private final byte[] goCBRef = {(byte) 0x80, 0x00};

25 private byte[] timeAllowedToLive = {(byte) 0x81, 0x05};

26 private final byte[] datSet = {(byte) 0x82, 0x00};

27 private final byte[] goID = {(byte) 0x83, 0x00};

28 private final byte[] t = {(byte) 0x84, 0x08};

29 private final byte[] stNum = {(byte) 0x85, 0x05};

30 private final byte[] sqNum = {(byte) 0x86, 0x05};

31 private final byte[] simulation = {(byte) 0x87, 0x01};

32 private final byte[] confRev = {(byte) 0x88, 0x05};

33 private final byte[] ndsCom = {(byte) 0x89, 0x01};

34 private final byte[] numDatSetEntries = {(byte) 0x8A, 0x05};

35 private final byte[] allDate = {(byte) 0xAB, 0x00};

36

37 private final byte[] bool = {(byte) 0x83, 0x01};

38 private final byte[] int32 = {(byte) 0x85, 0x05};

39 private final byte[] float32 = {(byte) 0x87, 0x05};

40

41 private byte[] valueGoCBRef;

42 private byte[] valueDatSet;

43 private byte[] valueGoID;

44

45 private ByteBuffer valueStNum = ByteBuffer.allocate(5);

46 private ByteBuffer valueSqNum = ByteBuffer.allocate(5);

47 private boolean valueSimulation = false;

48 private ByteBuffer valueConfRev = ByteBuffer.allocate(5);

49 private boolean valueNdsCom = false;

50

51 private byte[] valueBool = {0x01};

52 private byte[] valueInt32 = {0x00, 0x00, 0x00, 0x00, 0x00};

53 private byte[] valueFloat32 = {0x00, 0x00, 0x00, 0x00, 0x00};

54

55 private ByteBuffer valueTimeAllowedToLive = ByteBuffer.allocate(5).put(new byte[]{0x00, 0x00, 0x00, 0x00, 0x06});

56 private ByteBuffer valueT = ByteBuffer.allocate(8);

57 private ByteBuffer valueNumDatSetEntries = ByteBuffer.allocate(5);

58

59 private DataSet dataSet;

60

61 private ByteBuffer buffer;

62 private ByteBuffer headerBuffer;

63 private ByteBuffer dataBuffer;

64

65 private int[] delays = new int[]{0, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2000};

66 private long startTime;

67 private long previousTime;

68

69 private int lenGoose;

70 private int lenAllowTime;

71 private int lenSq;

72 private int lenData;

73 private int lenT;

74 private int lenSt;

75

76 private boolean unPaused = true;

77

78 private int conf;

79 private int sq;

80 private int st;

81

82 private List<PcapNetworkInterface> ifs;

83 private PcapHandle sendHandle;

84 private EthernetPacket packet;

85

86 @Override

87 @SneakyThrows

88 public void run() {

89 sendHandle.sendPacket(packet);

90 valueSqNum = valueSqNum.putInt(1, ++sq);

91 valueTimeAllowedToLive = valueTimeAllowedToLive

92 .putInt(1, (int) (delays[Math.min(sq + 1, delays.length - 1)] \* 1.5));

93 packet = EthernetPacket.newPacket(buffer.put(lenAllowTime, valueTimeAllowedToLive.array())

94 .put(lenSq, valueSqNum.array()).array(), 0, lenGoose);

95 }

96

97 @SneakyThrows

98 public GOOSE(DataSet dataSet) {

99 ifs = Pcaps.findAllDevs();

100 PcapNetworkInterface activeInterface = null;

101 for (PcapNetworkInterface pcapIface : ifs) {

102 if (pcapIface != null && pcapIface.getName().contains(dataSet.getIface())) {

103 activeInterface = pcapIface;

104 break;

105 }

106 }

107 assert activeInterface != null;

108 sendHandle = activeInterface.openLive(65536,

109 PcapNetworkInterface.PromiscuousMode.PROMISCUOUS, 50);

110 this.dataSet = dataSet;

111 createHeader(dataSet);

112 createData(dataSet);

113 createMessage();

114 }

115

116 @SneakyThrows

117 private void createMessage() {

118 valueT = valueT

119 .putInt((int) (Instant.now().getEpochSecond()))

120 .putInt(Instant.now().getNano());

121

122 headerBuffer.put(destination)

123 .put(source)

124 .put(type)

125 .put(appID)

126 .put(length)

127 .put(reserved1)

128 .put(reserved2)

129 .put(goosePdu)

130 .put(goCBRef)

131 .put(valueGoCBRef)

132 .put(timeAllowedToLive)

133 .put(valueTimeAllowedToLive.array())

134 .put(datSet)

135 .put(valueDatSet)

136 .put(goID)

137 .put(valueGoID)

138 .put(t)

139 .put(valueT.array())

140 .put(stNum)

141 .put(valueStNum.array())

142 .put(sqNum)

143 .put(valueSqNum.array())

144 .put(simulation)

145 .put(booleanToByte(valueSimulation))

146 .put(confRev)

147 .put(valueConfRev)

148 .put(ndsCom)

149 .put(booleanToByte(valueNdsCom))

150 .put(numDatSetEntries)

151 .put(valueNumDatSetEntries)

152 .put(allDate);

153

154 dataSet.getItems().forEach(this::typeValue);

155

156 buffer.put(headerBuffer.array())

157 .put(dataBuffer.array());

158

159 packet = EthernetPacket.newPacket(buffer.array(), 0, lenGoose);

160 }

161

162 private void typeValue(Item e) {

163 switch (e.getType()) {

164 case "Boolean" -> dataBuffer.put(bool)

165 .put(booleanToByte(Boolean.valueOf(e.getValue())));

166

167 case "Integer" -> dataBuffer.put(int32)

168 .put(new byte[]{0x00})

169 .putInt(Integer.parseInt(e.getValue()));

170

171 case "Float" -> dataBuffer.put(float32)

172 .put(new byte[]{0x08})

173 .putFloat(Float.parseFloat(e.getValue()));

174

175 }

176 }

177

178 private void createHeader(DataSet dataSet) {

179 valueConfRev = valueConfRev.putInt(1, ++conf);

180 valueStNum = valueStNum.putInt(1, ++st);

181 valueDatSet = dataSet.getDatasetName().getBytes(StandardCharsets.UTF\_8);

182

183 for (int i = 0; i < destination.length; i++) {

184 destination[i] = (byte) Integer.parseInt(dataSet.getMacDestination().split(":")[i], 16);

185 }

186 for (int i = 0; i < source.length; i++) {

187 source[i] = (byte) Integer.parseInt(dataSet.getMacSource().split(":")[i], 16);

188 }

189

190 valueGoCBRef = dataSet.getGoCbRef().getBytes(StandardCharsets.UTF\_8);

191 valueGoID = dataSet.getGoID().getBytes(StandardCharsets.UTF\_8);

192

193 datSet[datSet.length - 1] = (byte) valueDatSet.length;

194 goCBRef[goCBRef.length - 1] = (byte) valueGoCBRef.length;

195 goID[goID.length - 1] = (byte) valueGoID.length;

196 }

197

198 private void createData(DataSet dataSet) {

199 valueNumDatSetEntries = valueNumDatSetEntries.putInt(1, dataSet.getItems().size());

200

201 lenAllowTime = destination.length +

202 source.length +

203 type.length +

204 appID.length +

205 length.length +

206 reserved1.length +

207 reserved2.length +

208 goosePdu.length +

209 goCBRef.length +

210 valueGoCBRef.length +

211 timeAllowedToLive.length;

212

213 lenT = lenAllowTime +

214 valueTimeAllowedToLive.array().length +

215 datSet.length +

216 valueDatSet.length +

217 goID.length +

218 valueGoID.length +

219 t.length;

220

221 lenSt = lenT +

222 valueT.array().length +

223 stNum.length;

224

225 lenSq = lenSt +

226 valueStNum.array().length +

227 sqNum.length;

228

229 lenData = lenSq +

230 valueSqNum.array().length +

231 simulation.length +

232 booleanToByte(valueSimulation).length +

233 confRev.length +

234 valueConfRev.array().length +

235 ndsCom.length +

236 booleanToByte(valueNdsCom).length +

237 numDatSetEntries.length +

238 valueNumDatSetEntries.array().length +

239 allDate.length;

240

241 dataSet.getItems().forEach(this::lengthValue);

242

243 dataBuffer = ByteBuffer.allocate(lenGoose);

244 headerBuffer = ByteBuffer.allocate(lenData);

245

246 lenGoose += lenData;

247

248 buffer = ByteBuffer.allocate(lenGoose);

249

250 length[length.length - 1] = (byte) (lenGoose - destination.length - source.length - type.length);

251 }

252

253 private void lengthValue(Item e) {

254 switch (e.getType()) {

255 case "Boolean" -> {

256 lenGoose += bool.length + valueBool.length;

257 allDate[allDate.length - 1] += bool.length + valueBool.length;

258 }

259 case "Integer" -> {

260 lenGoose += int32.length + valueInt32.length;

261 allDate[allDate.length - 1] += int32.length + valueInt32.length;

262 }

263 case "Float" -> {

264 lenGoose += float32.length + valueFloat32.length;

265 allDate[allDate.length - 1] += float32.length + valueFloat32.length;

266 }

267 }

268 }

269

270 private byte[] booleanToByte(Boolean bool) {

271 if (bool) {

272 return new byte[]{0x01};

273 } else {

274 return new byte[]{0x00};

275 }

276 }

277

278 @SneakyThrows

279 public void setData() {

280 unPaused = false;

281 sq = 0;

282

283 dataBuffer.clear();

284 dataSet.getItems().forEach(this::typeValue);

285

286 valueSqNum = valueSqNum.putInt(1, sq);

287 valueTimeAllowedToLive = valueTimeAllowedToLive

288 .putInt(1, 6);

289 valueStNum = valueStNum.putInt(1, ++st);

290

291 valueT = valueT.clear()

292 .putInt((int) (Instant.now().getEpochSecond()))

293 .putInt(Instant.now().getNano());

294

295 headerBuffer.put(lenAllowTime, valueTimeAllowedToLive.array())

296 .put(lenSq, valueSqNum.array())

297 .put(lenT, valueT.array())

298 .put(lenSt, valueStNum.array());

299

300 buffer.clear()

301 .put(headerBuffer.array())

302 .put(dataBuffer.array());

303

304 packet = EthernetPacket.newPacket(buffer.array(), 0, lenGoose);

305 unPaused = true;

306 }

307

308

309 public void send() {

310 startTime = System.nanoTime();

311 if (startTime - previousTime >= delays[Math.min(sq, delays.length - 1)] \* 1000000L) {

312 run();

313 previousTime = startTime;

314 }

315 }

316 }

317

1 import com.fasterxml.jackson.dataformat.xml.annotation.JacksonXmlElementWrapper;

2 import com.fasterxml.jackson.dataformat.xml.annotation.JacksonXmlRootElement;

3 import lombok.Data;

4

5 import java.util.List;

6

7 @Data

8 @JacksonXmlRootElement

9 public class Root {

10

11 @JacksonXmlElementWrapper

12 private List<DataSet> senders;

13

14 }

15

1 import com.fasterxml.jackson.dataformat.xml.annotation.JacksonXmlElementWrapper;

2 import com.fasterxml.jackson.dataformat.xml.annotation.JacksonXmlProperty;

3 import lombok.Data;

4

5 import java.util.List;

6

7 @Data

8 public class DataSet {

9

10 @JacksonXmlProperty(isAttribute = true)

11 private String datasetName;

12 @JacksonXmlProperty(isAttribute = true)

13 private String iface;

14 @JacksonXmlProperty(isAttribute = true)

15 private String macSource;

16 @JacksonXmlProperty(isAttribute = true)

17 private String macDestination;

18 @JacksonXmlProperty(isAttribute = true)

19 private String goCbRef;

20 @JacksonXmlProperty(isAttribute = true)

21 private String goID;

22

23 @JacksonXmlProperty(localName = "item")

24 @JacksonXmlElementWrapper(useWrapping = false)

25 private List<Item> items;

26

27 }

28

1 import lombok.Data;

2

3 @Data

4 public class Item {

5

6 private String type;

7 private String value;

8

9 }

10