ProviewR

OPEN SOURCE PROCESS CONTROL

ProviewR redundancy

2020 08 26

Table of Contents

Introduction

This document describes the implementation of redundancy in ProviewR.

Note! Currently redundancy is still a beta version that is not thoroughly tested in production.

With redundancy here means how the function of a process node is taken over by another identical node at failure. The first node is called *primary node* and the second *secondary node*. At startup the primary node is *active* and executing the plc program and reading and writing to the I/O modules. The secondary node is *passive*, waiting for a failure to occur in the primary node. After a fail over the roles are reversed, the secondary node is active, and the primary node, after recovery, is passive ready to take over.

To make it possible for a passive node to take over from the active node the following is required:

- Content of objects handled by the plc program and I/O handler, and possibly by applications, has to be transferred cyclically from the active to the passive node. Reading of the objects in the active node has to be synchronized with the execution of the plc program to make sure that data is consistent.
- A supervision function that detects when a fail over should occur. The reason for a fail over can be failure in the active node, timeout of the cyclic transfer or on command of the operator.
- After a fail over the passive node starts the execution of the plc code and applications, and starts reading and writing to the I/O modules.
- Also a switching of nethandler and messagehandler communication with operator stations and other nodes will occur at fail over.

Redundancy will also make it easier to implement minor changes in the program as the secondary node temporary can take over while the primary node is updated and restarted.

Update of database

For the passive node to be able to start the execution of the plc program, it is required that the objects in the database are updated with data from the active node. Data that has to be transferred are data read from the IO modules, data calculated by the plc and applications, also internal data to detect edges and define states, and data set by operators. Attributes that is to be transferred has the bit ReduTransfer set in Flags field in the class definition.

In order not to lose any events, data has to be transferred every plc scan. Furthermore the transfer has to be made synchronous with the execution of plc threads and applications. Thus every plc thread and application itself handles the collection of data in the active node. Data handled by a thread or application is gathered into a packet that is sent to the corresponding thread in the passive node, where it’s unpacked and distributed to the database. The unpacking is also performed by the plc thread in order to be synchronous with a possible start up of the execution at fail over.

Some plc threads are executed at high frequency and high priority, executing a minor amounts of code, while other are executing at lower frequency and lower priority, executing larger amounts of code. This will also be reflected in the packet size where larger amount of code will imply larger amount of data and larger packets. As the packets are sent with the scan time of the thread, the scan time and packet size has to be adapted to the network capacity.

It is possible to set a priority of the packets so that smaller high prioritized packets will precede and even interrupt larger and lower prioritized packets.

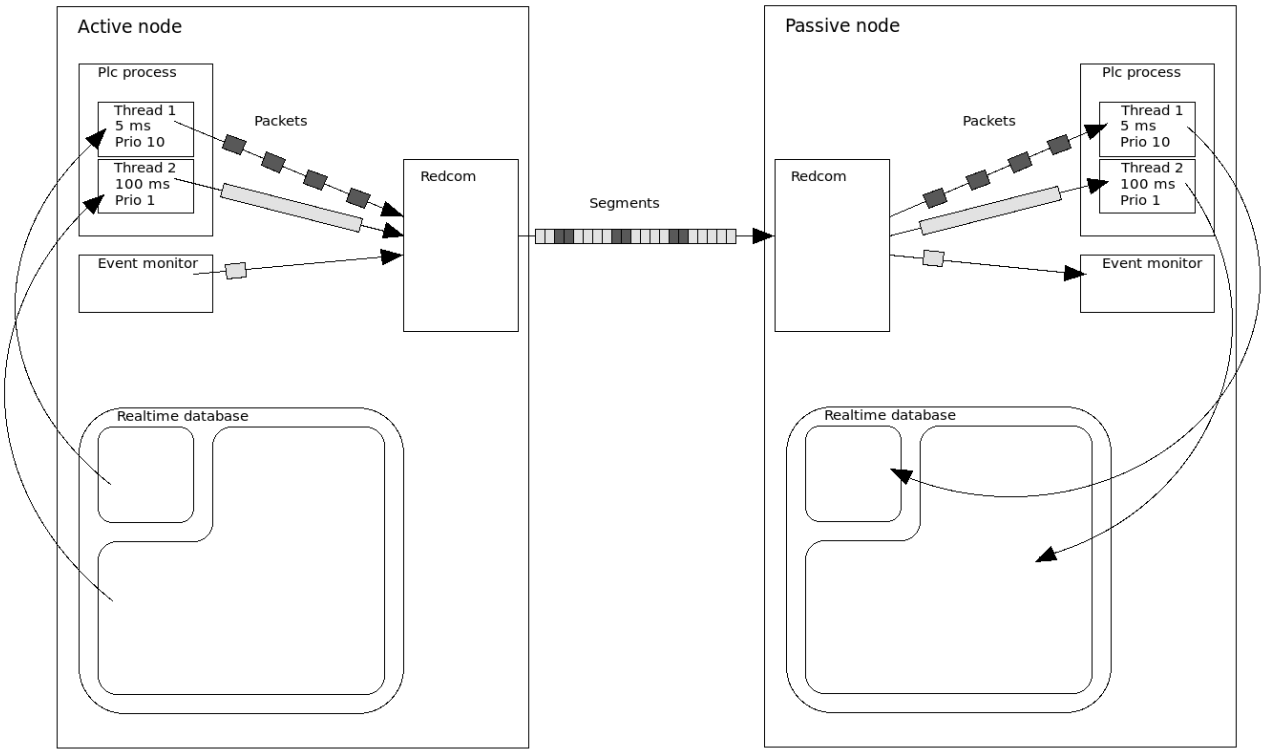

Fig Transfer of object data and events from active to passive node

Packet

A packet is configured with a *RedcomPacket* object. In the attribute *Prio* the priority of the packet is set in the interval 0 – 10 where 0 is the lowest priority and 10 the highest. The attribute *Hierarchies* is an array where the object hierarchies that is to be transferred by this thread is stated.

The *RedcomPacket* object is placed under the *PlcThread* object for the thread.

Table packet

The cyclic packets for a thread are preceded by a table packet that is sent from the active to the passive node at startup or after a fail over. The packet contains information about the structure of the coming cyclic packets, and where the packet data is to be distributed. In both the active and the passive node a list is created with pointers to the handled objects to optimize the packing and unpacking. The creation of the list and the table packet will take some time and therefor the start of the fail over supervision is delayed at startup. This delay time may have to be adjusted in the *RedcomConfig.StartupTimeout* attribute. If the startup sequence is yet not finished after this time, the passive node will initiate a fail over.

Transfer sequence

The sequence for transfer of data for a thread is as follows.

At startup each plc thread in the active node creates a list of all attributes that is to be contained in the cyclic packet. All objects under the hierarchies specified in Hierarchies in the *RedcomPacket* object is searched, and attributes with the *RedcomTransfer* bit set is added to the list. Then a table packet is sent to the plc thread of the passive node, containing a description of what the cyclic data

packets will contain, so that data can be distributed to the right place. The passive node builds a corresponding list to optimize the distribution of the data in the packet.

Then the sending the cyclic packets starts. It is executed by the plc thread in the active node, with the cycle time of the execution. The thread will execute the code, collect the data from the attribute list, and send the packet to the corresponding thread in the passive node. The thread in the passive node unpacks the packet and distributes the data in the database.

Event handling

Also alarms and events lists in the passive node has to be updated. This is done by a packet sent from the event monitor in the active node to the event monitor in the passive node.

Redcom

The communication between the active and passive node is executed by a server process, rt_redcom. The plc threads in the active node sends their packets to the redcom server. The packets are divided into segments (with a default size of 8192 bytes) and segments with higher priority are preceding segments with lower priority. In this way packets with higher priority will precede and interrupt packets with lower priority. The redcom server in the passive node receives the segments and restores the packet that are forwarded to the target thread.

The redcom server also handles the supervision of the node and decides if a switch from passive to active node or vice versa should be done.

The redcom server is configured with a *RedcomConfig* object that is placed in the node hierarchy.

Fail over

If a failure is detected in the active node, the passive node transits to active state. The reason can be

- EmergencyBreak. The EmergencyBreak attribute in the node object is set. The cause for this can be that some IO module doesn’t respond, or time out from a plc thread.
- SystemStatus. Error indication in system status is caused by timeout or error indication in any system process of application.
- Communication timeout. If the packets from the active node hasn’t arrived within the timeout time. The timeout time is configured in the RedcomConfig object.
- Manuel transition. A transition can be initiated manually from for example the object graph for the RecomConfig object.

Which of these reasons that should be able to cause a transition can be configured in the *RedcomConfig* object.

Operator station communication

Connected operator stations are also affected by a transition. Qcom in the operator station connects to both the primary and secondary node. All messages for net handler and event handler are channeled to the currently active node.

I/O

Modbus TCP

Modbus communication is performed with request from the master and respond to the requesting node from the slaves. Thus it is possible to communicate with the slaves without any further configuration.

PSS9000

Remote rack replies to the calling node and is able to communicate with both primary and secondary node without any additional configuration.

For QBUS rack it is possible to place both primary and secondary node in the same rack. To be able to switch between the nodes a minor modification of the IO cards is needed.

Configuration

Primary node

The primary node is configured as an ordinary process station with a NodeConfig object in the directory volume.

Secondary node

The secondary node is configured in the NodeConfig object for the primary node under *SecondaryNode*. Node name, boot node and IP address should be stated here.

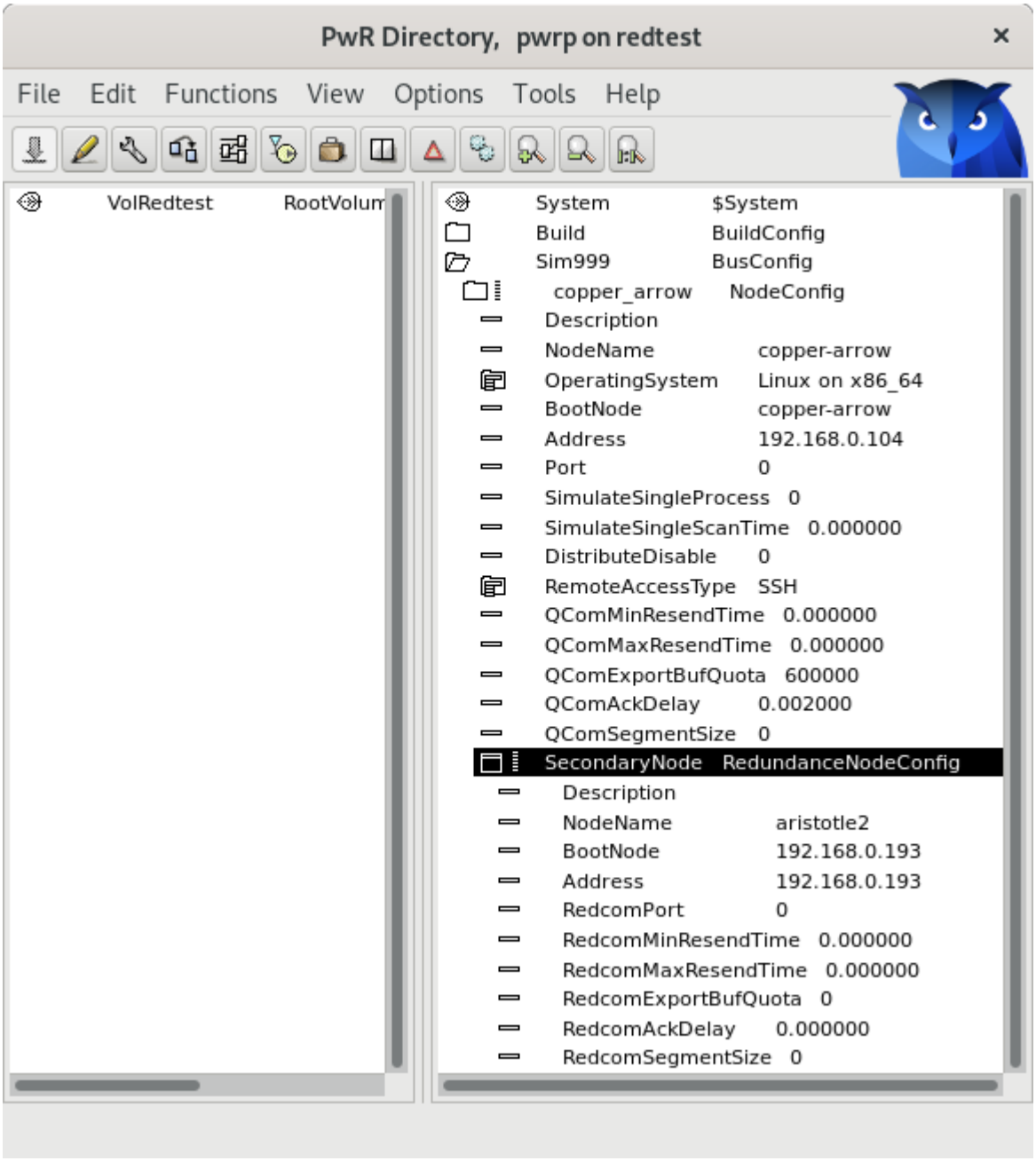

Fig Configuration of plc thread packets

Plc thread packets

The packets for the plc threads are configure with a *RedcomPacket* object under each *PlcThread* object. The packet priority is stated in *Prio*, a value between 0 and 10 where 0 is low priority and 10 high. The object hierarchies that are handled by the thread and that should be included in the packet, is stated in the *Hierachies* array.

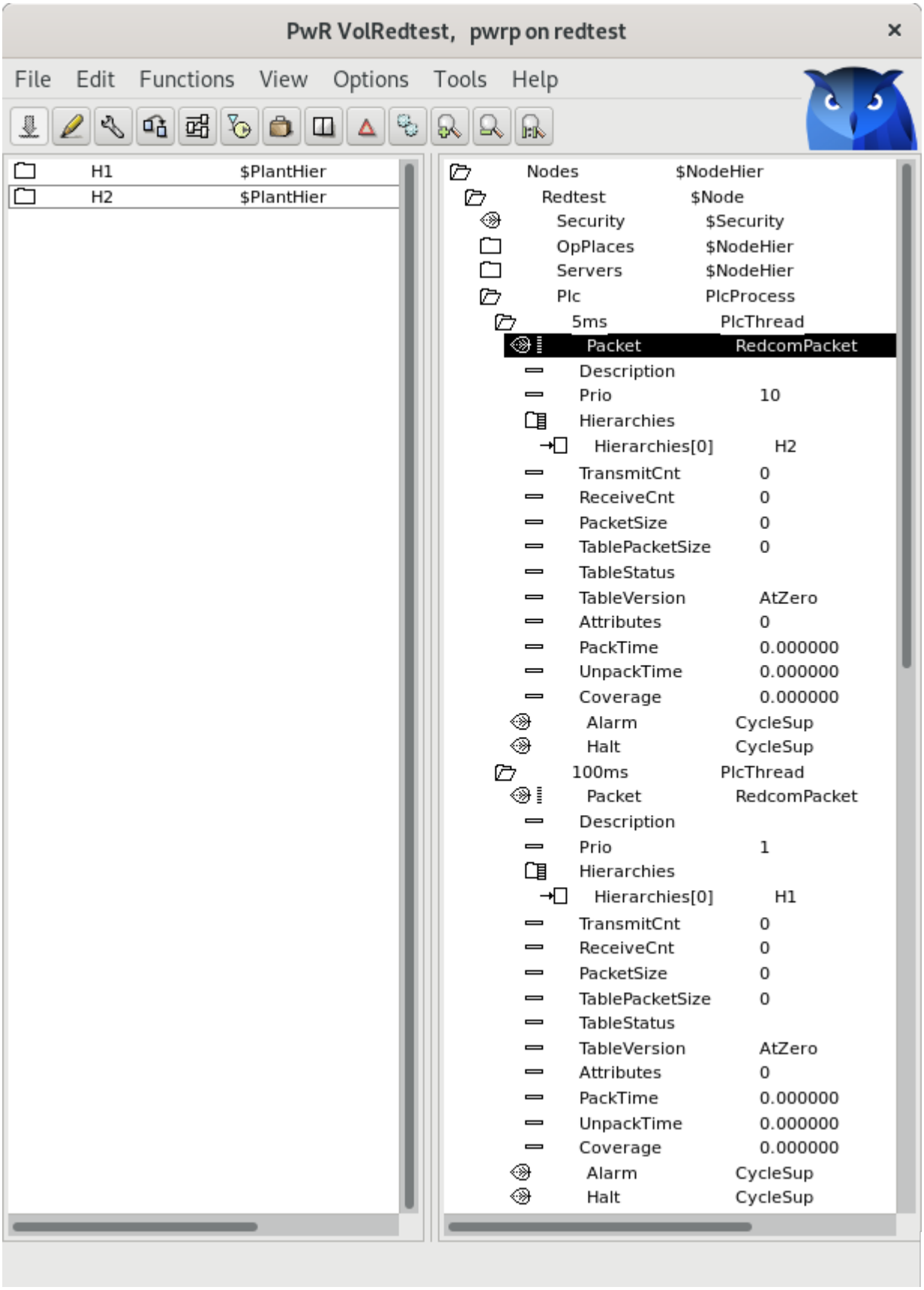

Fig Configuration of plc thread packets

Redcom server

The redcom server process is configured with a *RedcomConfig* object in the node hierarchy.

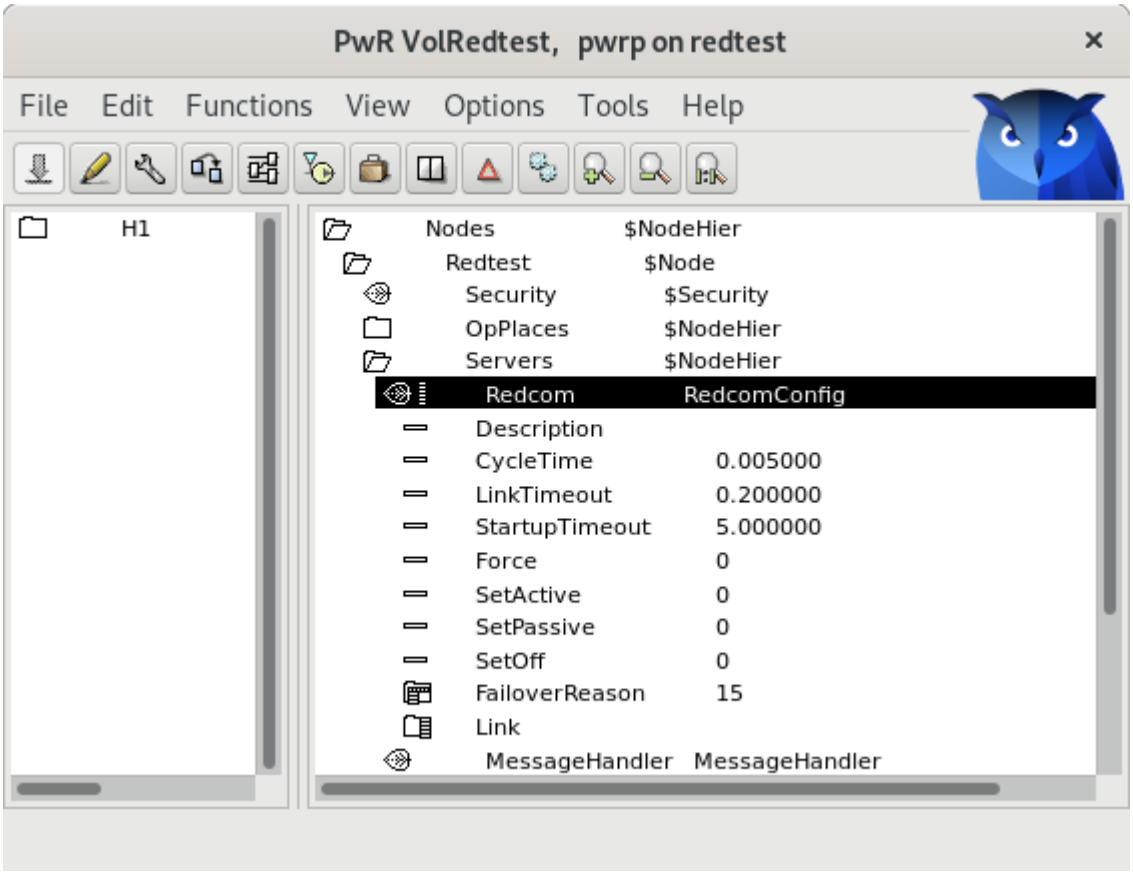

Fig Configuration of redcom server

Applications

Applications can transfer data with the redu_appl API. It's defined in rt_redu.h and contains the functions

```
pwr_tStatus redu_appl_init( redu_tCtx* ctx,
                            pwr_sClass_RedcomPacket* packetp);
pwr_tStatus redu_appl_send( redu_tCtx ctx,
                            void* msg,
                            int size,
                            pwr_tTime version,
                            unsigned int msg_id);
pwr_tStatus redu_appl_receive(redu_tCtx ctx,
                            unsigned int timeout,
                            void** msg,
                            int* size);
```

redu_appl_init() will intialize the application return a redu context. redu_appl_send() will send a message with data from the active node to the passive, and redu_appl_receive() will receive in the

message in the passive node.

The message should contain a redu_sMsgHeader, eg

```
typedef struct {
  redu_sMsgHeader h;
  float data1;
  float data2;
} sApplMessage;
```

The header will be filled in by the send function.

In passive mode the applications will just receive the message and store the data. In active mode it will execute its tasks and then send the message.

```
void scan()
{
  if (nodep->RedundancyState == pwr_eRedundancyState_Passive) {
    sApplMessage *rmsg;
    int tmo = 1000;
    pwr_tTime version = pwr_cNTime;

    sts = redu_appl_receive(ctx, tmo, &rmsg, &size);
    if (ODD(sts) {
      data1 = rmsg->data1;
      data2 = rmsg->data2;
      qcom_Free(&sts, rmsg);
    }
  } else {
    static unsigned int msgid = 0;
    sApplMessage msg;
    struct timespec scantime = {1, 0};

    // Do some calculations
    …
    msg.data1 = data1;
    msg.data2 = data2;
    sts = redu_appl_send(ctx, &msg, sizeof(msg), version, msgid++);

    nanosleep(&scantime, NULL);
  }
}
```

Building

Both the primary and secondary nodes will be present in the build node list and can be built individually. They will share the same volume, but some files like ld_boot, ld_node and plc executable will be different. A new configuration file for the redcom server will be added (ld_redcom).

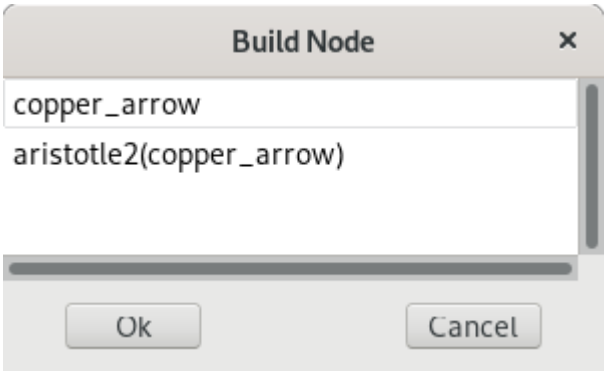

Fig Build node list with primary and secondary node

Both nodes are also present in the distribution list and are distributed separately.

Runtime

If a node is currently active or passive is displayed in the node graph, in the upper left corner, or in the object graph for the *RedcomConfig* object. The current state is stored in the *RedundancyState* attribute in the node object.

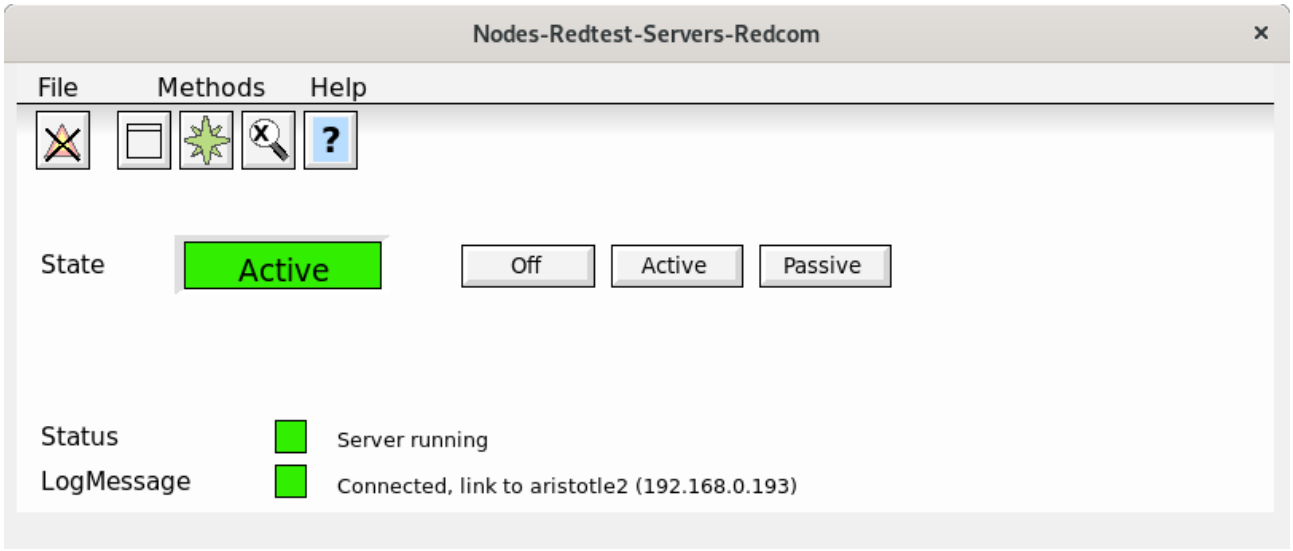

Fig Object graph for RedcomConfig object

From this graph, also a manual transition can be made with the *Active* and *Passive* buttons.

The *Link[0]* attribute in the *RedcomConfig* object contains the state of the link to the other node.

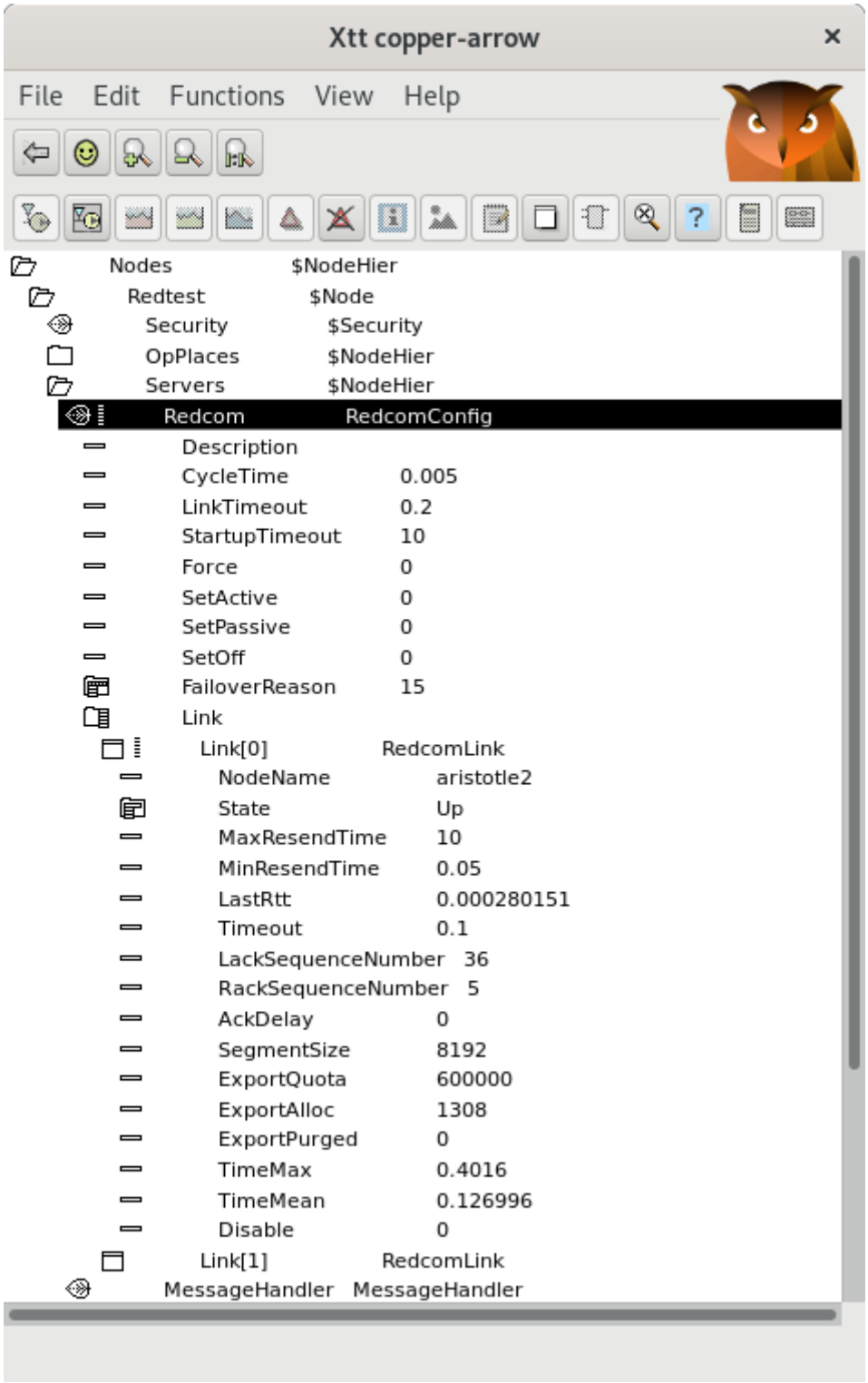

Fig Link info in the active node

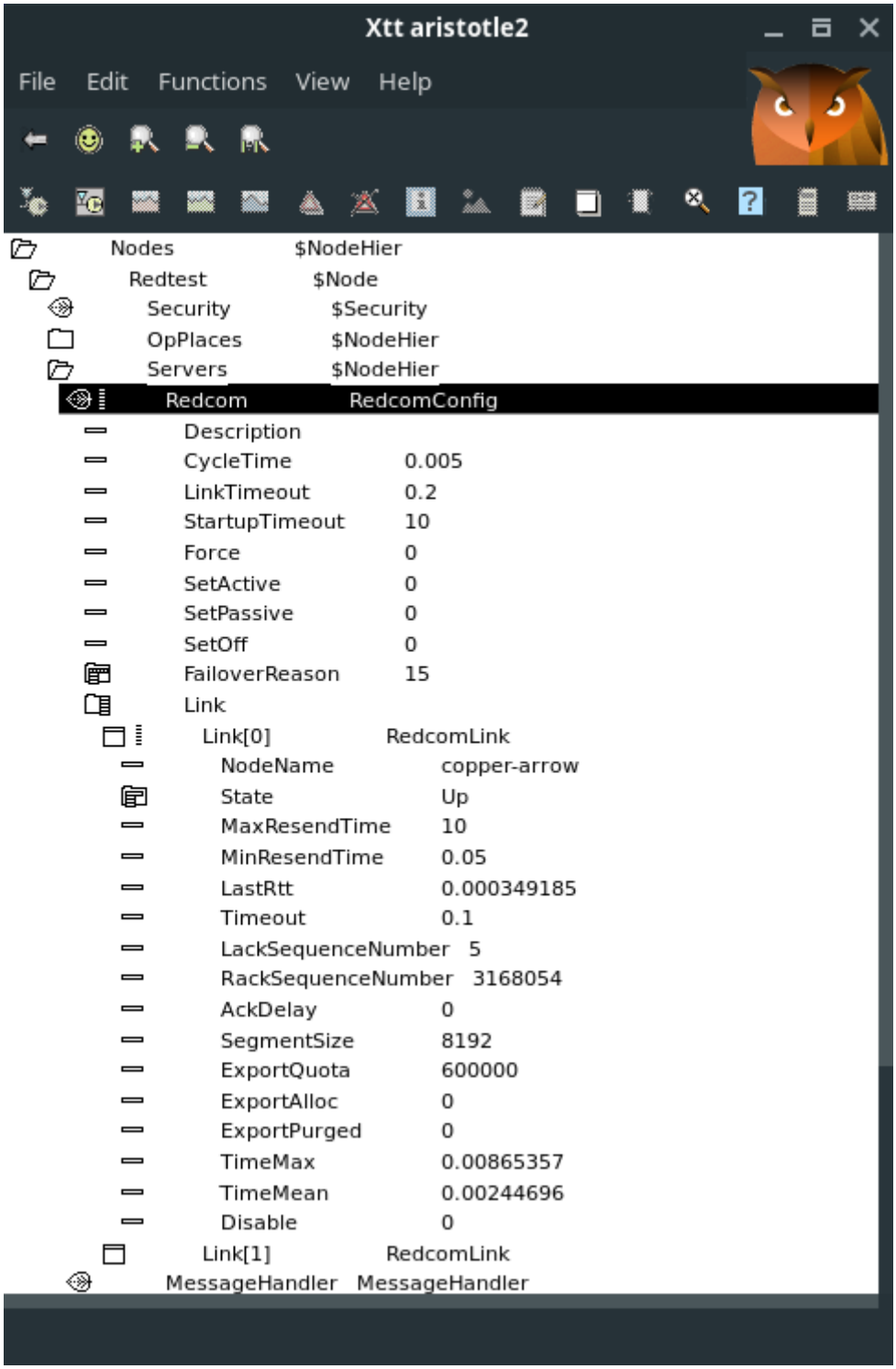

Fig Link info in the passive node

Information about the packet transfer is showed in the *RedcomPacket* objects.

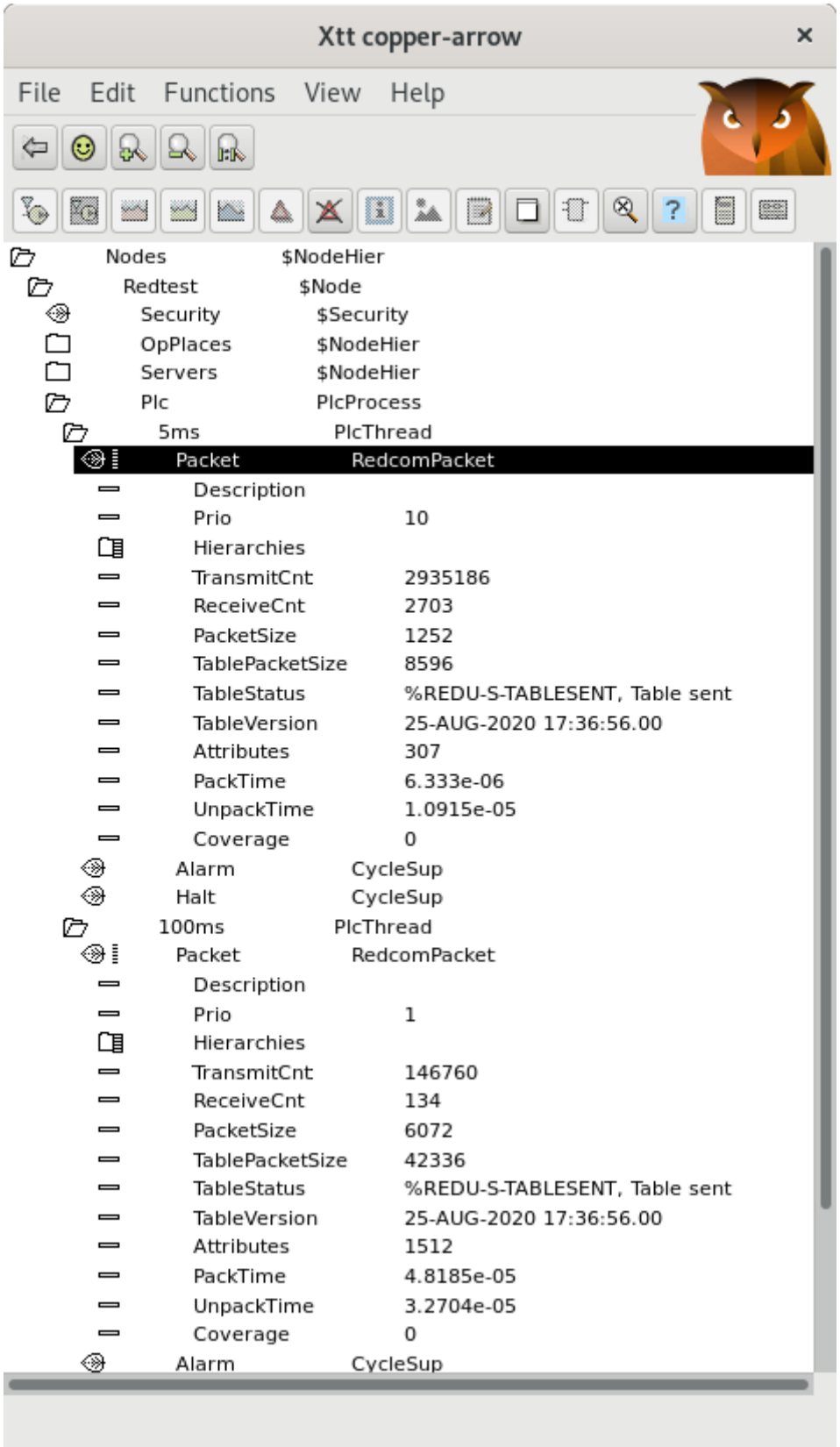

Fig RedcomPacket objects in active node

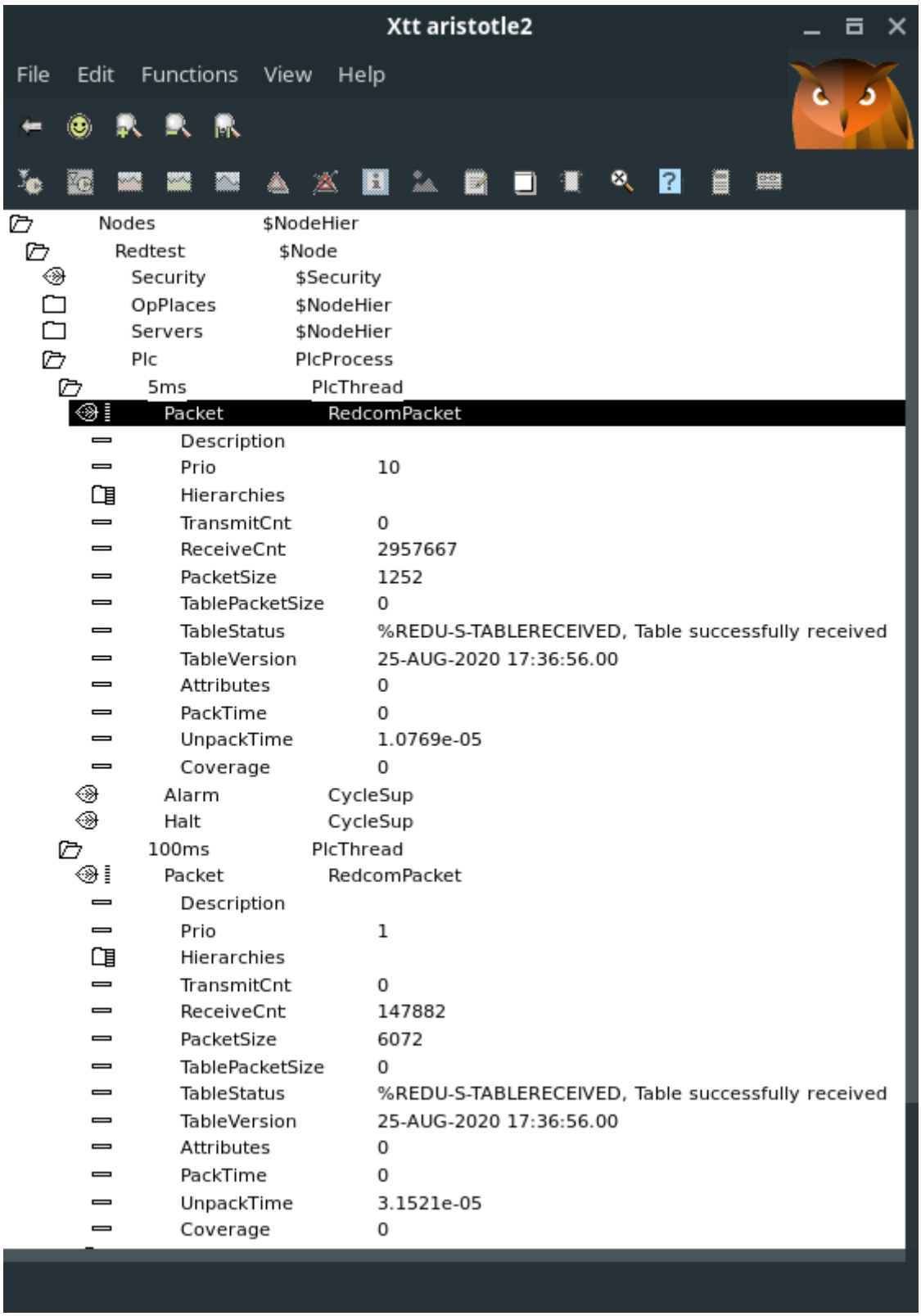

Fig RedcomPacket objects in passive node

Program updates

Minor changes in the program can be made without affecting the process. Distribute the changes to the passive node and restart this node. The active node will continue to send also after the restart

and when the activity is switched to the passive node, the modifications will be started. After a distribution and restart of the now passive node, the updated is completed.

Some consideration has to be made though. When the first node is restarted, it will receive recom data from the other node, and this from the old program. New objects will not be affected, but changed configuration data in old objects will be overwritten. In some cases this data can be inserted manually (or with a script) when the switch is preformed.

Known bugs

- NodeConfig.SecondaryNode.RedundantSegmentSize has no default value and has to be set to 8192. Fixed in V5.7.2.
- Statistics in the PlcThread object is not always updated in the passive node. Fixed in V5.7.2.
- PlcThread.Coverage in passive node is not calculated correctly and shows a to large value.