Openhab smaenergymeter

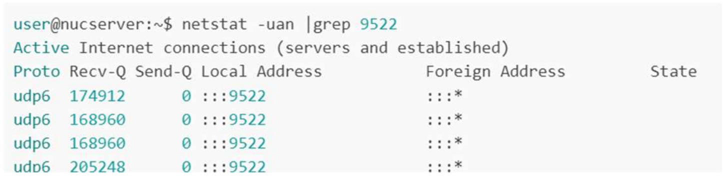
Several issues were detected in a network with following hardware:

- SMA Homemanger 2.0. Mcast 239.12.255.254, port 9522, SN1
- SMA Energymeter 20, Mcast 239.12.255.254, port 9522, SN2
- SMA Energymeter 20, Mcast 239.12.255.254, port 9522, SN3

Most of it is described in the thread: https://community.openhab.org/t/sma-energy-meter-binding-yields-unplausible-values/128180/137

Starting from the code at https://github.com/ConnectorIO/openhab-addons.git , /Branch: sma-fixes-4.2, I investigated this issues, and implemented some extra logging and code to improve the read-outs
(tried out on Raspberry Pi 4B, with openHAB-4.2.0-snapshot build #3900

1/ Delays in receiving data:

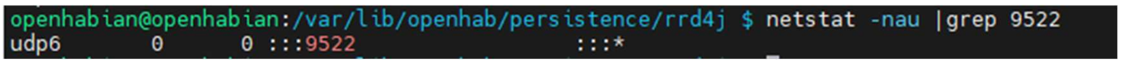


```
user@nucserver:~$ netstat -uan |grep 9522
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
udp6  174912      0 :::9522                 :::*
udp6  168960      0 :::9522                 :::*
udp6  168960      0 :::9522                 :::*
udp6  205248      0 :::9522                 :::*
```

Binding was joining 4 x the same multicast group.
1 joinGroup happens during thing discovery, 3 others by each meter => 4 in total
Proposed fix: modify isOpen() in PacketListener.java to

```
    public boolean isOpen() {
        return socket != null && !socket.isClosed();
    }
```

As all SMA meters send data to the **same** multicast group, and the multicast group is open all the time, the input buffer fills up with a lot of data. SMA meters send packets every second. The binding itself only read 1 packet every 30 second (default value). The buffer is FIFO. Delays go up to 20-30 mins.
Proposed fix: move socket.JoinGroup(address) and socket.LeaveGroup() to ReceivingTask run. See code below.
After fix: 1x multicast group, no Recv-Q queue => no delay in measurements



```
openhabian@openhabian:/var/lib/openhab/persistence/rrd4j $ netstat -nau |grep 9522
udp6       0      0 :::9522                 :::*
```

2/ The binding is polling every 30 minutes. So, if 3 meters, every meter is sampled on average every 90 seconds.
Only 1 packet was read every 30 seconds by the existing binding

```
socket.receive(msgPacket);     in ReceivingTask run
```

I modify the binding to read a minimum of 3 samples per run, and also ignores all packets with a size less then 600. (to get rid of broadcast frames)
This modification is an ugly fix, as it only works with my 3 meters. Rework by a Java specialist is required.

3/ Distinguish between the Home manager 2.0 (HM) and the Energymeter 20 (EM).
The binding decodes only the 608-byte HM dataframes in a correct way.
The EnergyMeter dataframe is 8 bytes shorter (no Frequency data at position 156 to 163)
Proposed update: modify energymeter.parse to insert 8 zeros in case of EM dataframes. (see code below)

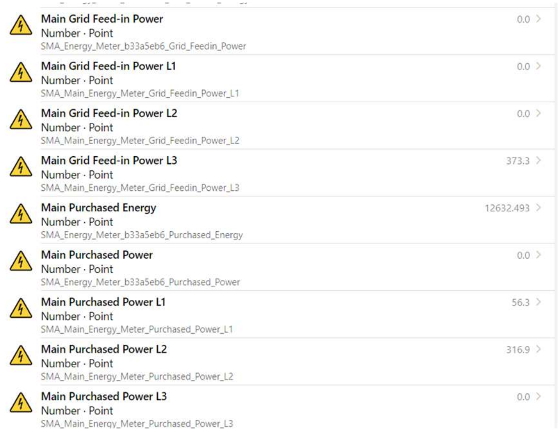
I have the updated binding running now, and everything looks fine:

Some debug loggings:

Every 30 seconds: 1 update of every meter

```
openhabian@openhabian:~ $ grep Multi /var/log/openhab/o*log
2024-02-13 16:34:05.767 [DEBUG] [.packet.PacketListener$ReceivingTask] - [Multicast UDP message received] meter 1 >> 0 600
2024-02-13 16:34:05.851 [DEBUG] [.packet.PacketListener$ReceivingTask] - [Multicast UDP message received] meter 2 >> 0 600
2024-02-13 16:34:05.904 [DEBUG] [.packet.PacketListener$ReceivingTask] - [Multicast UDP message received] meter 3 >> 0 608
2024-02-13 16:34:35.762 [DEBUG] [.packet.PacketListener$ReceivingTask] - [Multicast UDP message received] meter 1 >> 0 600
2024-02-13 16:34:35.841 [DEBUG] [.packet.PacketListener$ReceivingTask] - [Multicast UDP message received] meter 2 >> 0 600
2024-02-13 16:34:35.895 [DEBUG] [.packet.PacketListener$ReceivingTask] - [Multicast UDP message received] meter 3 >> 0 608
2024-02-13 16:35:05.752 [DEBUG] [.packet.PacketListener$ReceivingTask] - [Multicast UDP message received] meter 1 >> 0 600
```

Openhab Homemanager: sum of all power L1,L2,L3 = sum of feed-in + purchased power. Now =0 due to battery



Main Grid Feed-in Power 0.0
Number · Point
SMA_Energy_Meter_b33a5eb6_Grid_Feedin_Power

Main Grid Feed-in Power L1 0.0
Number · Point
SMA_Main_Energy_Meter_Grid_Feedin_Power_L1

Main Grid Feed-in Power L2 0.0
Number · Point
SMA_Main_Energy_Meter_Grid_Feedin_Power_L2

Main Grid Feed-in Power L3 373.3
Number · Point
SMA_Main_Energy_Meter_Grid_Feedin_Power_L3

Main Purchased Energy 12632.493
Number · Point
SMA_Energy_Meter_b33a5eb6_Purchased_Energy

Main Purchased Power 0.0
Number · Point
SMA_Energy_Meter_b33a5eb6_Purchased_Power

Main Purchased Power L1 56.3
Number · Point
SMA_Main_Energy_Meter_Purchased_Power_L1

Main Purchased Power L2 316.9
Number · Point
SMA_Main_Energy_Meter_Purchased_Power_L2

Main Purchased Power L3 0.0
Number · Point
SMA_Main_Energy_Meter_Purchased_Power_L3

PacketListener.java

```java
/**
 * Copyright (c) 2010-2024 Contributors to the openHAB project
 *
 * See the NOTICE file(s) distributed with this work for additional
 * information.
 *
 * This program and the accompanying materials are made available under the
 * terms of the Eclipse Public License 2.0 which is available at
 * http://www.eclipse.org/legal/epl-2.0
 *
 * SPDX-License-Identifier: EPL-2.0
 */
package org.openhab.binding.smaenergymeter.internal.packet;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.MulticastSocket;
import java.util.List;
import java.util.concurrent.CopyOnWriteArrayList;
import java.util.concurrent.ScheduledFuture;

import org.openhab.binding.smaenergymeter.internal.handler.EnergyMeter;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * The {@link PacketListener} class is responsible for communication with the SMA devices.
 * It handles udp/multicast traffic and broadcast received data to subsequent payload handlers.
 *
 * @author Łukasz Dywicki - Initial contribution
 */
public class PacketListener {
    private final Logger logger = LoggerFactory.getLogger(ReceivingTask.class);
    private final DefaultPacketListenerRegistry registry;
    private final List<PayloadHandler> handlers = new CopyOnWriteArrayList<>();

    private String multicastGroup;
    private int port;

    public static final String DEFAULT_MCAST_GRP = "239.12.255.254";
    public static final int DEFAULT_MCAST_PORT = 9522;

    private MulticastSocket socket;
    private ScheduledFuture<?> future;
    private InetAddress address;

    public PacketListener(DefaultPacketListenerRegistry registry, String multicastGroup, int port) {
        this.registry = registry;
        this.multicastGroup = multicastGroup;
        this.port = port;
    }

    public void addPayloadHandler(PayloadHandler handler) {
        handlers.add(handler);
    }

    public void removePayloadHandler(PayloadHandler handler) {
        handlers.remove(handler);

        if (handlers.isEmpty()) {
            registry.close(multicastGroup, port);
        }
    }

    public boolean isOpen() {
        return socket != null && !socket.isClosed();
    }

    public void open(int intervalSec) throws IOException {
        if (isOpen()) {
            logger.debug("no need to bind socket second time");
            return;
        }
        socket = new MulticastSocket(port);
        socket.setSoTimeout(5000);
        address = InetAddress.getByName(multicastGroup);
        logger.debug("JoinGroup: " + multicastGroup);
        socket.setReuseAddress(true);
        // socket.joinGroup(address);

        future = registry.addTask(new ReceivingTask(socket, address, multicastGroup + ":" + port, handlers),
                intervalSec);
    }

    void close() throws IOException {
        if (future != null) {
            future.cancel(true);
        }

        address = InetAddress.getByName(multicastGroup);
        socket.leaveGroup(address);
        logger.debug("LeaveGroup: " + multicastGroup);
        socket.close();
    }

    public void request() {
        registry.execute(new ReceivingTask(socket, address, multicastGroup + ":" + port, handlers));
    }

    static class ReceivingTask implements Runnable {
        private final Logger logger = LoggerFactory.getLogger(ReceivingTask.class);
        private final MulticastSocket socket;
        private final InetAddress address;
        private final String group;
        private final List<PayloadHandler> handlers;

        ReceivingTask(MulticastSocket socket, InetAddress address, String group, List<PayloadHandler>
handlers) {
            this.socket = socket;
            this.address = address;
            this.group = group;
            this.handlers = handlers;
        }

        public void run() {
            try {
                // byte[] bytes = new byte[608];
                byte[] bytes1 = new byte[608];
                byte[] bytes2 = new byte[608];
                byte[] bytes3 = new byte[608];
                // DatagramPacket msgPacket = new DatagramPacket(bytes, bytes.length);
                DatagramPacket msgPacket1 = new DatagramPacket(bytes1, bytes1.length);
                DatagramPacket msgPacket2 = new DatagramPacket(bytes2, bytes2.length);
                DatagramPacket msgPacket3 = new DatagramPacket(bytes3, bytes3.length);

                socket.joinGroup(address);

                do {
                    socket.receive(msgPacket1);
                } while (msgPacket1.getLength() < 600);

                try {
                    EnergyMeter meter1 = new EnergyMeter();
                    // meter.parse(bytes);
                    logger.debug("[Multicast UDP message received] meter 1 >> " + msgPacket1.getOffset() + "
"
```

```
                            + msgPacket1.getLength());
                    meter1.parse(msgPacket1.getData(), msgPacket1.getLength());

                    for (PayloadHandler handler : handlers) {
                        handler.handle(meter1);
                    }
                } catch (IOException e) {
                    logger.info("Unexpected payload received for group {}, meter 1", group, e);
                }

                do {
                    socket.receive(msgPacket2);
                } while (msgPacket2.getLength() < 600);

                try {
                    EnergyMeter meter2 = new EnergyMeter();
                    // meter.parse(bytes);
                    logger.debug("[Multicast UDP message received] meter 2 >> " + msgPacket2.getOffset() + "
"

                            + msgPacket2.getLength());
                    meter2.parse(msgPacket2.getData(), msgPacket2.getLength());

                    for (PayloadHandler handler : handlers) {
                        handler.handle(meter2);
                    }
                } catch (IOException e) {
                    logger.info("Unexpected payload received for group {}, meter 2", group, e);
                }
                do {
                    socket.receive(msgPacket3);
                } while (msgPacket3.getLength() < 600);

                socket.leaveGroup(address);

                try {
                    EnergyMeter meter3 = new EnergyMeter();
                    // meter.parse(bytes);
                    logger.debug("[Multicast UDP message received] meter 3 >> " + msgPacket3.getOffset() + "
"

                            + msgPacket3.getLength());
                    meter3.parse(msgPacket3.getData(), msgPacket3.getLength());

                    for (PayloadHandler handler : handlers) {
                        handler.handle(meter3);
                    }
                } catch (IOException e) {
                    logger.info("Unexpected payload received for group {}, meter 3", group, e);
                }
            } catch (IOException e) {
                logger.warn("Failed to receive data for multicast group {}", group, e);
            }
        }
    }
}
```

EnergyMeter.java

```java
/**
 * Copyright (c) 2010-2024 Contributors to the openHAB project
 *
 * See the NOTICE file(s) distributed with this work for additional
 * information.
 *
 * This program and the accompanying materials are made available under the
 * terms of the Eclipse Public License 2.0 which is available at
 * http://www.eclipse.org/legal/epl-2.0
 *
 * SPDX-License-Identifier: EPL-2.0
 */
package org.openhab.binding.smaenergymeter.internal.packet;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.MulticastSocket;
import java.util.List;
import java.util.concurrent.CopyOnWriteArrayList;
import java.util.concurrent.ScheduledFuture;

import org.openhab.binding.smaenergymeter.internal.handler.EnergyMeter;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * The {@link PacketListener} class is responsible for communication with the SMA devices.
 * It handles udp/multicast traffic and broadcast received data to subsequent payload handlers.
 *
 * @author Łukasz Dywicki - Initial contribution
 */
public class PacketListener {
    private final Logger logger = LoggerFactory.getLogger(ReceivingTask.class);
    private final DefaultPacketListenerRegistry registry;
    private final List<PayloadHandler> handlers = new CopyOnWriteArrayList<>();

    private String multicastGroup;
    private int port;

    public static final String DEFAULT_MCAST_GRP = "239.12.255.254";
    public static final int DEFAULT_MCAST_PORT = 9522;

    private MulticastSocket socket;
    private ScheduledFuture<?> future;
    private InetAddress address;

    public PacketListener(DefaultPacketListenerRegistry registry, String multicastGroup, int port) {
        this.registry = registry;
        this.multicastGroup = multicastGroup;
        this.port = port;
    }

    public void addPayloadHandler(PayloadHandler handler) {
        handlers.add(handler);
    }

    public void removePayloadHandler(PayloadHandler handler) {
        handlers.remove(handler);

        if (handlers.isEmpty()) {
            registry.close(multicastGroup, port);
        }
    }

    public boolean isOpen() {
        return socket != null && !socket.isClosed();
    }

    public void open(int intervalSec) throws IOException {
        if (isOpen()) {
            logger.debug("no need to bind socket second time");
            return;
        }
        socket = new MulticastSocket(port);
        socket.setSoTimeout(5000);
        address = InetAddress.getByName(multicastGroup);
        logger.debug("JoinGroup: " + multicastGroup);
        socket.setReuseAddress(true);
        // socket.joinGroup(address);

        future = registry.addTask(new ReceivingTask(socket, address, multicastGroup + ":" + port, handlers),
                intervalSec);
    }

    void close() throws IOException {
        if (future != null) {
            future.cancel(true);
        }

        address = InetAddress.getByName(multicastGroup);
        socket.leaveGroup(address);
        logger.debug("LeaveGroup: " + multicastGroup);
        socket.close();
    }

    public void request() {
        registry.execute(new ReceivingTask(socket, address, multicastGroup + ":" + port, handlers));
    }

    static class ReceivingTask implements Runnable {
        private final Logger logger = LoggerFactory.getLogger(ReceivingTask.class);
        private final MulticastSocket socket;
        private final InetAddress address;
        private final String group;
        private final List<PayloadHandler> handlers;

        ReceivingTask(MulticastSocket socket, InetAddress address, String group, List<PayloadHandler>
handlers) {
            this.socket = socket;
            this.address = address;
            this.group = group;
            this.handlers = handlers;
        }

        public void run() {
            try {
                // byte[] bytes = new byte[608];
                byte[] bytes1 = new byte[608];
                byte[] bytes2 = new byte[608];
                byte[] bytes3 = new byte[608];
                // DatagramPacket msgPacket = new DatagramPacket(bytes, bytes.length);
                DatagramPacket msgPacket1 = new DatagramPacket(bytes1, bytes1.length);
                DatagramPacket msgPacket2 = new DatagramPacket(bytes2, bytes2.length);
                DatagramPacket msgPacket3 = new DatagramPacket(bytes3, bytes3.length);

                socket.joinGroup(address);

                do {
                    socket.receive(msgPacket1);
                } while (msgPacket1.getLength() < 600);

                try {
                    EnergyMeter meter1 = new EnergyMeter();
                    // meter.parse(bytes);
                    logger.debug("[Multicast UDP message received] meter 1 >> " + msgPacket1.getOffset() + "
"
                            + msgPacket1.getLength());
                    meter1.parse(msgPacket1.getData(), msgPacket1.getLength());

                    for (PayloadHandler handler : handlers) {
                        handler.handle(meter1);
                    }
                } catch (IOException e) {
                    logger.info("Unexpected payload received for group {}, meter 1", group, e);
                }

                do {
                    socket.receive(msgPacket2);
                } while (msgPacket2.getLength() < 600);

                try {
                    EnergyMeter meter2 = new EnergyMeter();
                    // meter.parse(bytes);
                    logger.debug("[Multicast UDP message received] meter 2 >> " + msgPacket2.getOffset() + "
"
```

```
                            + msgPacket2.getLength());
                    meter2.parse(msgPacket2.getData(), msgPacket2.getLength());

                    for (PayloadHandler handler : handlers) {
                        handler.handle(meter2);
                    }
                } catch (IOException e) {
                    logger.info("Unexpected payload received for group {}, meter 2", group, e);
                }
                do {
                    socket.receive(msgPacket3);
                } while (msgPacket3.getLength() < 600);

                socket.leaveGroup(address);

                try {
                    EnergyMeter meter3 = new EnergyMeter();
                    // meter.parse(bytes);
                    logger.debug("[Multicast UDP message received] meter 3 >> " + msgPacket3.getOffset() + "
"
                            + msgPacket3.getLength());
                    meter3.parse(msgPacket3.getData(), msgPacket3.getLength());

                    for (PayloadHandler handler : handlers) {
                        handler.handle(meter3);
                    }
                } catch (IOException e) {
                    logger.info("Unexpected payload received for group {}, meter 3", group, e);
                }
            } catch (IOException e) {
                logger.warn("Failed to receive data for multicast group {}", group, e);
            }
        }
    }
}
```