AXIOMWARE SYSTEMS, INC.

Netrunr Series

Netrunr Edge Quick Start Guide

AXIOMWARE SYSTEMS, INC.

Netrunr Edge Quick Start Guide

229 Polaris Ave • Suite 15
Mountain View, CA 94043
Phone +1-408-675-8382 • www.axiomware.com

1. Overview.

Netrunr Edge Server (NES) is designed to provide connectivity, compute and storage functionality for your IoT network deployments. NES is a general-purpose Linux computer, capable of providing extensive customization to support various end-user requirements.

An overview of Netrunr Edge server hardware is provided in the Netrunr Edge Hardware Guide. An overview of major software components is provided in the Netrunr Edge Software Manual.

2. Hardware Setup.

Before powering up the Netrunr Edge server, attach the antennas: The dual-band antenna is of smaller size and should to attached to ANT1 port.

Use the 12V DC power supply to power up the NES. The LED1 will blink with heartbeat rhythm after the successful boot-up of the OS (approximately 25 seconds from power-up).

3. Connecting to NES.

Netrunr Edge server is shipped with Debian 9 pre-installed. The factory image has two login accounts:

a) root
b) axm

axm is a user account with normal privileges. Since Debian 9 does not allow SSH login directly into root account, you can use the axm account for remote SSH login. You can also create additional accounts as required. The initial passwords are sent through email to customers.

There are two major methods to connect to the NES:

1. The RS232C port can be used to monitor the boot-up process and to directly login to NES. After boot-up process, a shell console is presented. No network connection is required when using this method. The serial port settings are as follows: 115200 kbps 8N1 with no flow control. A Null modem cable must be used to connect to a PC.
2. Network connection to WAN/LAN1/LAN2/Wi-Fi.
    a. WAN port: Use an ethernet cable to connect the WAN port to a router. Monitor the router to find out the IP address assign to NES.
    b. LAN1/LAN2 port: Use an ethernet cable to connect your computer directly to LAN1 or LAN2 port. Set the computer to obtain IP address from NES by DHCP. The IP address of NES will be 192.168.3.1
    c. Wi-Fi Access Point: NES will be advertising as an access point with hostname as SSID (check the bottom label for hostname). Connect to the NES SSID. The default Wi-Fi password is **MyWiFiPassword (**Please refer to the NES software manual (Section 3.2) for information to change the default password). Once you connect to the NES Wi-Fi access point, the IP address of NES will be 192.168.3.1

For access through network connection, you can use a terminal program or PUTTY to ssh into NES:

```
# connect using SSH. Change the IP address as appropriate
ssh axm@192.168.3.1
```

4. Initial Setup.

Netrunr Edge server is shipped with Debian 9 pre-installed. You may want to modify the following parameters:

- User account passwords
- Wi-Fi Access point password (see section 3.2 of NES SW manual)
- Set date and time zone (see below)

To set time zone on Debian 9, you can `use timedatectl` command as follows:

```
# Example of setting time zone to US EST
# get name of US EST time zone
timedatectl list-timezones | grep -i new

# set to the new time zone
sudo timedatectl set-timezone America/New_York

# Verify if the time is set properly
date
```

- `Check Network connection (see below)`

```
# Check the state of wireless interfaces
# sudo iwconfig

# Check the state of all interfaces
# sudo ifconfig

# Verify that a valid IP address has been assigned to the upstream interface.

# Check connection by pinging to external
ping 8.8.8.8
```

5. Run a simple Bluetooth application.

Netrunr Edge server is fully equipped to operate as a Bluetooth gateway. In the example below, we execute all operations locally, including operating the MQTT broker on the NES (refer to Figure 4 in NES SW manual).

The Bluetooth radio on NES is configured by the file `/etc/btc/gwconfig`. Please see below for an example:

```
> more /etc/btc/gwconfig
{
        "mqtt_user": "",
        "mqtt_mode": 0,
        "mqtt_tls": 0,
        "mqtt_data_event_out": "user/BT000DB94E1560/6",
        "mqtt_data_report_out": "user/BT000DB94E1560/4",
        "mqtt_client_id": "BT000DB94E1560_clid",
        "scan_hardware_filter": 1,
        "host": "192.168.3.1",
        "scan_period": 10,
        "mqtt_data_in": "",
        "mqtt_pwd": "",
        "application": "gapi",
        "mqtt_data_out": "user/BT000DB94E1560/2",
        "mqtt_data_in": "user/BT000DB94E1560/1",
        "scan_active": 0,
        "port": 1883
}
```

This gwconfig file is set to transfer data to host: 192.168.3.1 (which is local host). The Bluetooth radio service will publish or subscribe to four MQTT topics:

Topic type	Topic Name	Direction
mqtt_data_in	user/BT000DB94E1560/1	subscribe
mqtt_data_out	user/BT000DB94E1560/2	publish
mqtt_data_report_out	user/BT000DB94E1560/4	publish
mqtt_data_event_out	user/BT000DB94E1560/6	publish

We will use MQTT client tools, `mosquitto_sub` and `mosquitto_pub` to directly interact with the Bluetooth radio. The command line tools use the JSON formatted MQTT commands as specified in the API (https://www.axiomware.com/netrunr-mqtt/). Two terminal windows are required or terminal multiplexing software like tmux can be used for this demonstration.

```
# Login into NES in terminal window 1 and subscribe all the topics by
# the radio. Check the topics names in /etc/btc/gwconfig
# Monitor this window for results for commands issued on terminal window 2
# Subscribe to 3 topics published by the radio
mosquitto_sub -d -h 127.0.0.1 -t user/BT000DB94E1560/2 -t  user/BT000DB94E1560/4 -t
user/BT000DB94E1560/6
```

```
# Login into NES in terminal window 2 and publish commands to the radio
# Check the topics names in /etc/btc/gwconfig
# Issue a command to perform active scan for 2 seconds
```

```
mosquitto_pub -d -h 127.0.0.1 -t user/BT000DB94E1560/1 -m '{"c":1, "active":0,
"period":1, "filter":2}'
```

The results of the scan appear in terminal window 1. An example snippet of the output is show below. You can use JSON pretty-print to format the output. In this output, we are interested in a specific device high-lighted below in red color:


```
{"nodes":[{"did":"07d9bb2be971","dtype":1,"ev":4,"rssi":-68, "tss":1544638272,
"tsus":74607, "adv":[{"t":1,"v":"1a"}, {"t":255,"v":"4c001005031827b0ea"}],
"rsp":[]},{"did":"277ebd0ead6f","dtype":1,"ev":4,"rssi":-85,"tss":1544638271,
"tsus":992563,"adv":[{"t":1,"v":"06"},{"t":255,"v":"4c0010020108"}],"rsp":[]},{"did":
"2dd92986ee08","dtype":1,"ev":3,"rssi":-75,"tss":1544638272, "tsus":129651,
"adv":[{"t":255,"v":"060001092002f6d57d34be8d90d2bcedc7b387b443933754473d58cdf3"}],"r
sp":[]},{"did":"381111d7b47f","dtype":1,"ev":4,"rssi":-93,"tss":1544638271,
"tsus":495814,"adv":[{"t":1,"v":"1a"},{"t":255,"v":"4c001005031c50c82c"}],"rsp":[]},
{"did":"3b265a0baec6","dtype":1,"ev":4,"rssi":-51,"tss":1544638272, "tsus":152887,
"adv":[{"t":25,"v":"0000"},{"t":1,"v":"06"},{"t":255,"v":"a305493f0200"},{"t":9,"v":"
AXMS5A263B"}],"rsp":[{"t":7,"v":["7d1c82979a359a964f426c0c010090a4"]}]},
{"did":"d49312b3b5db","dtype":1,"ev":4,"rssi":-59,"tss":1544638271,"tsus":646988,
"adv":[{"t":1,"v":"06"}], "rsp":[{"t":9,"v":"Smart Humigadget"}]},
{"did":"dd0cb09818f0", "dtype":0,"ev":0,"rssi":-92,"tss":1544638271,"tsus":772766,
"adv":[{"t":1,"v":"06"}, {"t":255,"v":"4c0010020708"}],"rsp":[]},
{"did":"f4f35ac90659","dtype":1,"ev":0,"rssi":-93,"tss":1544638271, "tsus":898381,
"adv":[{"t":1,"v":"06"},{"t":255,"v":"4c000c0e0094601865c75c0019366833818a"}],"rsp":[
]}],"result":200,"subcode":0,"report":1}
```


We would like to connect to the highlighted device with device ID : "**3b265a0baec6".**

```
# Execute this command in terminal window 2
# Connect to a device if specific connection parameters
mosquitto_pub -d -h 127.0.0.1 -t user/BT000DB94E1560/1 -m '{"c":4, "node":
"3b265a0baec6","dtype": 1, "interval_min": 16, "interval_max": 80, "latency": 0,
"timeout": 200, "wait": 10}'
```

If the connection is successful, get list of primary services:

```
# Execute this command in terminal window 2
# Get the list of primary services
mosquitto_pub -d -h 127.0.0.1 -t user/BT000DB94E1560/1 -m '{"c":9, "node":
"3b265a0baec6","primary": 1}'
```

An example snippet of output terminal window 1 displayed below. We would like to see the GATT characteristics under the custom service highlighted below:


```
{"c":9,"m":0,"node":"3b265a0baec6","services":[{"handle":"0001","end":"0009","uuid":"
0018","primary":1},{"handle":"000a","end":"000a","uuid":"0118","primary":1},{"handle"
```

:"000b","end":"ffff","uuid":"7d1c82979a359a964f426c0c010090a4","primary":1}],"result":200,"subcode":0}

```
# Execute this command in terminal window 2
# Get the list of charactristics using service handles
mosquitto_pub -d -h 127.0.0.1 -t user/BT000DB94E1560/1 -m '{"c":14, "node": "3b265a0baec6","sh": "000b", "eh": "ffff", "uuid": "0328"}'
```

An example snippet of output terminal window 1 displayed below. We would like to see the GATT characteristics under the custom service highlighted below. Let us enable notifications on characteristic handle highlighted in red.

{"c":14,"m":0,"node":"3b265a0baec6","characteristics":[{"handle":"000c","end":"000e","uuid":"7d1c82979a359a964f426c0c020090a4","properties":"12","vh":"000d"},{"handle":"000f","end":"0010","uuid":"7d1c82979a359a964f426c0c030090a4","properties":"0a","vh":"0010"},{"handle":"0011","end":"0012","uuid":"7d1c82979a359a964f426c0c040090a4","properties":"0a","vh":"0012"},{"handle":"0013","end":"0015","uuid":"7d1c82979a359a964f426c0c070090a4","properties":"12","vh":"0014"},{"handle":"0016","end":"0018","uuid":"7d1c82979a359a964f426c0c080090a4","properties":"12","vh":"0017"},{"handle":"0019","end":"001b","uuid":"7d1c82979a359a964f426c0c090090a4","properties":"12","vh":"001a"},{"handle":"001c","end":"ffff","uuid":"7d1c82979a359a964f426c0c0a0090a4","properties":"12","vh":"001d"}],"result":200,"subcode":0}

```
# Execute this command in terminal window 2
# Let us enable notifications on characteristic handle “001c”
mosquitto_pub -d -h 127.0.0.1 -t user/BT000DB94E1560/1 -m '{"c":18, "node": "3b265a0baec6","ch": "001c", "notify": 1}'
mosquitto_pub -d -h 127.0.0.1 -t user/BT000DB94E1560/1 -m '{"c":27, "node": "3b265a0baec6","ch": "001c", "notify": 1, "event": 1}'
```

The Bluetooth device should be periodically sending out notification events. You can use the following commands to show all the devices connected to the gateway, stop notifications and disconnect any connected device.

```
# Execute this command in terminal window 2
# Show all devices connected to the gateway
mosquitto_pub -d -h 127.0.0.1 -t user/BT000DB94E1560/1 -m '{"c":2, "enable":1}'

# Stop notifications

mosquitto_pub -d -h 127.0.0.1 -t user/BT000DB94E1560/1 -m '{"c":18, "node": "3b265a0baec6","ch": "001c", "notify": 0}'

# Disconnect all devices from the gateway
mosquitto_pub -d -h 127.0.0.1 -t user/BT000DB94E1560/1 -m '{"c":5, "node":null}'
```

The Mosquitto client tools are great for quickly exploring the Bluetooth functionality. It is easy to use the same API in JavaScript, Java, C/C++, Python, Scala, etc., to create production programs. Axiomware will be releasing examples on our Github reposistory.