Veseo

vNet

Network Virtualization Layer

User Guide

VT-D03

Originally written by Dario Di Maio

Printed in Italy

Veseo

Napoli

Revision Control

Description	Author	Date	Rev.
First draft	Dario Di Maio	2011 Nov 24	A
Updated as per release A2.2	Dario Di Maio	2012 Mar 09	1
Updated as per release A4	Dario Di Maio	2013 May 10	2

Index

1 Purpose and Intended Audience

This document help developers to proper set a vNet network, is intended for people with a basic knowledge of electronics and telecommunication, focused on: wireless and wired networks, IPv4 networks.

1.1 Download Source Code

Source code is included in Souliss project available for download at website http://www.souliss.net

1.2 Credits

The vNet code is developed by *Veseo* and contributors, is distributed as open source you can use and modify it as per General Public License GPL v3.

2 Overview

vNet is a network virtualization layer, that provide a single set of APIs across different communication interfaces, including routing and bridging functionality between nodes. Frames that involve different communication interfaces or wireless extension over the listening range, are routed and bridged automatically without any action on the application side, creating a network of devices running over different communication medias, but connected together.

The vNet layer is coded mainly in C and is based on the Chibiduino stack, extended for multiple wired and wireless interfaces and including a software IP stack that can be included for Ethernet based devices.

2.1 Supported Hardware

vNet is developed for Arduino / AVR based devices and has Atmel Atmega328P as target microcontroller and support the followings media controller:

- Atmel AT86RF230 for Wireless 2.4 GHz point-to-point,
- Microchip ENC28J60 for UDP/IP and MACRAW,
- Wiznet W5100 for UDP/IP and MACRAW,
- Microchip MRF24WB0MA for WiFi (in ZG2100 compatibility)

Communication behind vNet is peer-to-peer and can be applied and extended for media communication that has collision detection either in hardware or software.

In case of Wiznet W5100 is used the in-hardware IP stack of the chip and can be used also for bridging UDP/IP and MACRAW, up to two additional UDP/IP and TCP/IP

communication socket can be used out of vNet.

In case of Microchip ENC28J60 and MRF24WB0MA is used a software IP stack based on Contiki uIP, is not actually (release A4) supported the bridging between UDP/IP and MACRAW.

In case of Atmel AT86RF230 the Chibiduino stack is used, the wireless communication can be bridged to Ethernet based one for both W5100 and ENC28J60, generally vNet allow automatically bridging over different media controller.

2.2 Peer to Peer Network

vNet is intended as a software layer for P2P communication, so all the drivers for the available communication media are forced as P2P., There is no concept of server/client, master/slave or similar. The applications on the top of vNet can use out of P2P also server/client, master/slave or similar concept.

In vNet a total of five drivers can be attached, as now three are included in the base distribution. A suitable driver shall include its own addressing and filtering rules, and work in P2P.

2.3 Supported Network Architectures

Basically vNet works with all the architectures that you can have with the supported media, the configuration may require different configurations based on the number of nodes and their interconnection.

2.3.1 Single Media Networks

A network is defined “Single Media” if all the nodes use the same communication media interface, in this case the vNet layer is just a go-through between the APIs and

the media drivers.

2.3.2 Bridged Networks

A network is defined “Bridged” if there is more than one communication media interface and data need to be shared within nodes using different communication interfaces.

A bridge node has two or more communication interfaces and vNet cares of the bridging between the same. More than one bridge can be used in one network.

All the nodes can communicate directly with the nodes with the same communication interface, rather frames are bridged when the destination is a node with a different interface. Each interface has an address range, so from the address is known which interface is refereed to.

A node can act as bridge if is defined as SuperNode in the compiling options.

2.3.3 Routed Network

A network is defined “Routed” if there are nodes that share the same communication interface but are not able or shall not communicate directly. As instance, two wireless node that are out of the listening range need a middle node to route the frames.

The routing path can be identified automatically based on the addressing or may require definition of routing tables in case of more complex networks.

A node can act as router if is defined as SuperNode in the compiling options.

2.3.4 Bridged and Routed Networks

A network is bridged and routed, if there is more than one communication media

interface and data need to be shared within nodes using different communication interfaces; furthermore within one or more media is required a routing of frames to extend the listening range.

3 Network Configuration Rules

Bridging and routing facility are provided assuming the use of simple network configuration rules, these allow a node to identify how bridge or route a message and if these actions are required without additional configuration.

3.1 Addressing Rules

The addressing rules used for vNet are mainly the same of IPv4 with one difference, the length of the address. In vNet are used only two bytes (instead of 4 bytes as in IPv4).

Furthermore, in vNet the addresses are split by media, this make easy identify which media has to be used for the communication.

In the following table are shown the available ranges for all the media, the code support up to five different media (more can be added). A comparison of an address with these ranges is used to find out the media of the relevant address.

Media	Range Start	Range End
M1 – UDP/IP over LAN	0x0000	0x00FF
M1 – UDP/IP in User Mode	0x0100	0x64FF
M2 – Wireless 2.4 GHz IEEE 802.15.4	0x6500	0xAAFF
M3 – MACRAW over LAN	0xAB00	0xBBFF
M4 – Not Used	*0xBC00*	*0xCCFF*
M5 – Not Used	*0xCE00*	*0xFEFF*

Each media has an associated subnet mask, using a bit-wise AND between address and subnet mask, is obtained the subnet of the device. The subnet mask is fixed for M1

medias as 0x0000 and is user configurable for others, using 0xFF00 as default. Some configuration may request fixed subnet mask.

3.2 Address Translation

The address translation is carried out in the vNet layer when the used communication interface has addressing rules that doesn't allow a direct use of the vNet address into the interface.

In the actual release of vNet an address translation is provided for IPv4 based devices that require a 4 byte address instead of the 2 byte address of vNet. From a math point of view, this translation cannot be unique, so some bytes should be forced using a “Base IP Address”. The IPv4 address is obtained with a bit-wise adding of the “Base IP Address” and the vNet address. Due to the address translation, IP based vNet frames couldn't go out of their own IP Subnet via IP routing or NATting.

An additional address translation mode called “User Mode” is available for devices that require operation behind a NAT, the “User Mode” is build for user interfaces that shall access the vNet network from the internet and not only the home network LAN. When using the “User Mode” the IP address is saved for next communication, this mode require a first connection from the user interface to the board in order to allow the IP address recording, is not allowed the use of “User Mode” between the boards because doesn't allow a full peer-to-peer.

3.3 Routing and Bridging Rules

Each communication interface has an its own subnet mask and is part of the relevant subnet. A bit-wise AND operation between the address and the subnet mask give the subnet address.

Device within the same subnet must be in listening range, the concept of listening range should be referred to the communication media that are used for the device. As instance, for device that use wireless communication a listening range is the area where the radio are able to send and receive message; for device that use a wired or wireless Ethernet a listening range is the switched/routed LAN network and the bridging is not requested.

A frame that has a destination out of the own subnet, is sent to a device with routing and/or bridging capability. In this case this device is called SuperNode (different from the IP Gateway) that will bridge or route the message. In the configuration of each node is specified the own SuperNode address.

A device compiled as SuperNode is able to route and/or bridge frames without additional configurations, the output path for a routed or bridged frame is evaluated assuming the followings:

- All devices share the same subnet mask;
- Each subnet has a SuperNode with address *"subn & 0x0001";*

If these requirement are not meet, the routing and bridging tables allow frame routing via preferred paths or not standard addressing, these are configured into routing and bridging tables.

3.4 IP Based vNet and Standard TCP/IP Connections

The vNet IP connectivity is based on UDP/IP frames that encapsulate the vNet frame itself, all nodes are always listening on port 230 for incoming data. A communication between two nodes means send an UDP frame to the IP address of the target node on port 230. In case of User Mode communication, the listening port for data is 23000.

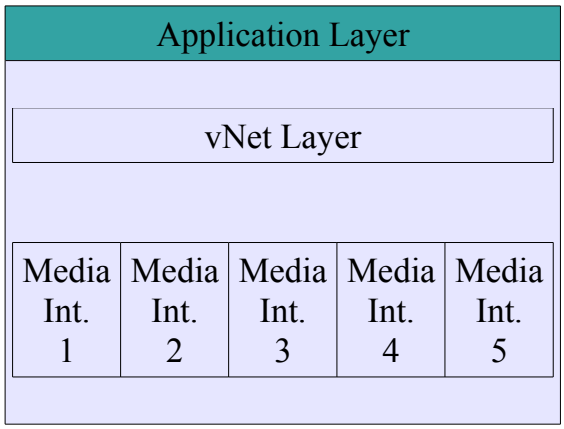
As all nodes are listening on port 230, there is no need to reply to a request using the IP source port, to identify different data streams a vNet port can be used, this port is shared over all medias (not only IP). As result any vNet frames (doesn't matter if is coming from a request or answer to the top layer protocol) is sent with destination IP port 230 using a random source IP port.

In vNet are available API for bulding standard TCP/IP connection external to the vNet layer it self. If the IP software stack is used each listener needs a dedicated handler.

4 Application Methods

As described before, every application build over vNet can be used regardless the communication media without any change in the code.

The following schematic shown the vNet structure, whatever is running on the "Application Layer" can access one or more "Media Interface" through the "vNet Layer".



The "vNet Layer" is not able to auto identify the available hardware, so an additional configuration is required to enable the available media and define node behavior.

The main functions that are used in the "Application Layer" are the followings:

- *U8 vNet_SendData(U16 addr, U8 *data, U8 len, U8 port);*
- *U8 vNet_DataAvailable();*
- *U8 vNet_RetriveData(U8 *data);*

The *vNet_Send* method send a message to a node identified with its address (*addr*), the frame is automatically routed or bridged (if required).

The *vNet_DataAvailable* method looks for data available for any of the available media, it will return the number of media that have at least one incoming message.

The *vNet_RetriveData* retrieve data from the buffer in order of media, should runs one for each media that have at least one incoming message.

Other methods are available for setting up the address and routing/bridging rules for complex architectures.

4.1 Output Frame oFrame

As additional tool in vNet is defined the oFrame structure, this is used over the Souliss project for handling of payload between Souliss layers saving RAM, oFrame are not mandatory if using vNet.

The oFrame is a way have a frame fragmented over different areas of the RAM, without the need of moving all data in a common consecutive space, is build over a recursive pointer structure (a chain) that point out the areas where data are allocated, a set of APIs get as serial output the data that are sent to the transceiver.

4.2 vNet Drivers

For each media interface supported is available a drivers, that is a collection of code that care of communication protocol between the microcontroller and the relevant communication media interface. Using compatible hardware with the listed media controller may need setup of pinouts, generally the default configuration is compatible with most common hardware.

List of available drivers:

- Wireless 2.4 GHz point-to-point Atmel AT86RF230,

- Ethernet UDP/IP Wiznet W5100,
- Ethernet MACRAW Wiznet W5100,
- Ethernet UDP/IP Microchip ENC28J60,
- Ethernet MACRAW Microchip ENC28J60,
- WiFi UDP/IP Microchip MRF24WB0MA (in ZG2100 compatibility).

4.3 vNet Frames Header

Frames in the vNet network are composed of six byte of header and a variable payload for a frame length, max frame length depends on implementation.

Length (1 byte)	Port (1 byte)	Final Destination Address (2 byte)	Original Destination Address (2 byte)

The header on the top together with the payload is what is parsed from the vNet layer, this is passed to the media driver that include additional information and encapsulate the vNet frame into the final communication frame.

4.3.1 vNet at Media Driver

The vNet frames can be transported over different “ways” that could be different from the electrical and telecommunication point of view. All drivers add an additional byte with the overall length, so the frames before be encapsulated into the final communication frame looks as the one before with an additional byte at begin of values Lenght+1, this will be parsed out at the receiving stage.

A Wireshark capture of MaCaco over vNet data stream between two nodes is available as reference for additional details, looking at a capture of a vNet over IP frame the

result is:

Ethernet II, Src: 1b:a6:49:6b:00:13 (1b:a6:49:6b:00:13), Dst: 1b:a6:49:6b:00:12 (1b:a6:49:6b:00:12)
Internet Protocol Version 4, Src: 192.168.1.18 (192.168.1.18), Dst: 192.168.1.17 (192.168.1.17)
User Datagram Protocol, Src Port: geognosis (4326), Dst Port: 230 (230)
Data (12 bytes)

An Ethernet framet that start from MAC address 1b:a6:49:6b:00:13 for 1b:a6:49:6b:00:12, that include and IP frame starting from 192.168.1.18 for 192.168.1.17, that contains and UDP frame from source port 4326 (random value) to 230 (vNet listening port) and contains a 12 bytes data payload. The MAC addresses are both broadcast ones only for capturing purposes.

Those payload is the vNet frame transported over UDP/IP, that it self contains the payload, a data protocol for the vNet application.

If we look into those 12 bytes, the result is: 0c:0b:17:11:00:12:00:05:6d:02:cb:08 these are the bytes transferred, using the structure discussed before,

Lenght+1 (1 byte)	Length (1 byte)	Port (1 byte)	Final Destination Address (2 byte)	Original Destination Address (2 byte)	Payload (Lenght – 6) bytes
0x0C	0x0B	0x17	0x0011	0x0012	0x05, 0x6D, 0x02, 0xCB, 0x08

As you can note, the data representation of the addresses (and all data build over more than one byte) are little-endian as per AVR architecture.

5 vNet Configuration and Debugging

The vNet configuration parameters are contained in file “vNet_Config.h” where the user can set the main options, each parameter has a dedicated description that make easier the configuration job.

As starting point, the examples provided with vNet contains in the header description the sets required for a proper configuration.

Followings options are the most important:

```
//vNet_Config.h

/*****************************************************/
/*!
If enabled allow broadcast support for incoming frames, to work properly
the same should be supported also by the communication media drivers.

      Value     SuperNode
      0x0       Disable (Default)
      0x1       Enable
*/
/*****************************************************/
#define VNET_BRDCAST                  0


/*****************************************************/
/*!
A SuperNode is a device with routing and bridging capability, if Enable
message from Nodes of the own subnet can be sent out to other subnet.

SuperNode address should be the first usable address of the subnet.

      Value      SuperNode
      0x0       Disable (Default)
      0x1       Enable
*/
```

```
/*********************************************************/
#define VNET_SUPERNODE     0


/*********************************************************/
/*!
A Node or a Supernode can have one or more media interfaces active at
same time. A device (either Node or Supernode) automatically  act  as  a
bridge between the active media.

      Value       Media
       0x0        Disable (Default)
       0x1        Enable

*/
/*********************************************************/
#define VNET_MEDIA1_ENABLE  0
#define VNET_MEDIA2_ENABLE  1
#define VNET_MEDIA3_ENABLE  0
#define VNET_MEDIA4_ENABLE  0
#define VNET_MEDIA5_ENABLE  0
```

The VNET_BRDCAST allow receiving broadcast message, send broadcast messages is always possible even if the option is not set.

The VNET_SUPERNODE allow routing/bridging functionality for a device.

The VNET_MEDIAn_ENABLE enable the n-media. If more than one media is activated, the VNET_SUPERNODE option is required to bridge the networks.

5.1 vNet Drivers

Each vNet driver has its own configuration file, this file contains all the parameters for a proper operation of the communication interface, each parameter has a dedicated

description that make easier the configuration job. As starting point, the examples provided with vNet contains in the header description the sets required for a proper configuration.

In the below table are listed the configuration files for the supported communication interfaces.

Media Number	Description	Driver Configuration File
1	Ethernet UDP/IP	ethUsrCfg.h
2	Chibiduino - 2.4 GHz IEEE 802.15.4	chbUsrCfg.h
3	Ethernet MACRAW	ethUsrCfg.h
4	Not used	-
5	Not used	-

5.2 Configuration Examples

A proper configuration is required for proper operation of vNet, the guidelines described in this document are applied in the examples included in the library. In the followings each example is described in the configuration parameters and addresses.

5.2.1 Wireless Routed Media Network (Example 1)

The vNet network is composed of two wireless subnets, with 0xFF00 mask: 0x6500 and 0x6600. Communication between subnets is feasible only if each subnet has its own Supernode. In this case there are two Supernode, one for each subnet.

Subnet 0x6500 is composed of only one node, the Supernode it-self; rather the 0x6600 is composed of two nodes.

Filename	Node Type	Address and Subnet	Subnet Mask	My SuperNode
vNet_example1_chibi_RX	Supernode	0x6501 / 0x6500	0xFF00	-
vNet_example1_chibi_SN	Supernode	0x6601 / 0x6600	0xFF00	-
vNet_example1_chibi_TX	Node	0x6612 / 0x6600	0xFF00	0x6601

Table 1: Node types, all nodes has subnet mask 0xFF00

The configuration of the two Supernodes require to set: VNET_SUPERNODE, VNET_MEDIA2_ENABLE in *vNet_Config.h* to load the proper code into the node. Address and subnet mask are as per table.

The configuration of the node require to set only VNET_MEDIA2_ENABLE, out of addresses and subnet mask also the MySuperNode parameter shall be set.

This example allow communication between Wireless 2.4 GHz nodes

5.2.2 Ethernet Single Media Network (Example 2)

The vNet network is composed of only two Ethernet devices, both under the same vNet and IP subnet. When planning the use of Ethernet devices in vNet the first step is find out usable addresses in own IP network.

As base recap, the IP subnet is obtained with a bit-wise AND operation between the IP address and the subnet mask. In Table 2 are shown some examples of home networks addresses and relevant subnets.

Example IP Address	Subnet Mask	Subnet
192.168.10.11	255.255.255.0	192.168.10.0
192.168.1.11	255.255.255.0	192.168.1.0
192.168.0.11	255.255.255.0	192.168.0.0

Table 2: Some home networks examples

In most of the cases, the subnet is composed of 255 and 0 that mean neutral and null values in the byte-wise operation of the binary algebra, so find out the subnet mask is a no brain activity.

The Ethernet configuration in vNet require the following configuration in *ethCfg.h*: DEFAULT_BASEIPADDRESS, DEFAULT_SUBMASK, DEFAULT_GATEWAY and MAC_ADDRESS. The base IP address is the the subnet address and not the complete IP address, the other bytes of the address came from the vNet address; the MAC address is typically provided with the Ethernet board, and shall be different for each loaded board, an option for using automatically defined local administered MAC address is available.

As example, if the DEFAULT_BASEIPADDRESS is {192, 168, 0, 0} and the vNet address is {0x000B}, the sum give the final IP address {192, 168, 0, 11}. Its noted that vNet is typically written as HEX value, this because looks easy find out the bytes structure. Obviously, using 11 as vNet address, give the same result.

To include the drivers of the W5100, the flag VNET_MEDIA1_ENABLE in *vNet_Config.h* shall be set.

5.2.3 Bridged Network (Example 3 and 4)

The vNet network is composed of three devices over two networks using two different medias. Data are exchanged between the nodes via the bridge, below the list of the required hardware and address configuration.

Filename	Node Type	Example Board
vNet_example3_chb_eth_RX	Ethernet	Arduino Ethernet or Arduino with Ethernet Shield
vNet_example3_chb_eth_TX	2.4 Ghz Wireless	Freaklabs Freakduino
vNet_example3_chb_eth_BR	Bridge	Freakduino with Ethernet Shield

Table 3: Supported Hardware

Filename	Address	Subnet Mask	My SuperNode
vNet_example3_chb_eth_RX	0x0012	0xFF00	0x0011
vNet_example3_chb_eth_TX	0x6512	0xFF00	0x6511
vNet_example3_chb_eth_BR	0x0011 0x6511	0xFF00 0xFF00	- -

Table 4: Address configurations

The guidelines for the code setup can be found in 5.2.1 for the Wireless configuration and in 5.2.2 for the Ethernet one.

5.2.4 Routed and Bridged Network (Example 5)

The vNet network is composed of three subnets and two different communication media, so the network is bridged and routed at same time. The difference between this example and the previous two is in the listening range of the wireless communication, using a routed network there is a middle wireless node able to repeat the data.

Filename	Address	Subnet Mask	My SuperNode
vNet_example5_chb_eth_RX	0x0012	0xFF00	0x0011
vNet_example5_chb_eth_TX	0x6612	0xFF00	0x6601
vNet_example5_chb_eth_BR	0x0011 0x6511	0xFF00 0xFF00	- -
vNet_example5_chb_eth_SN	0x6601	0xFF00	-

Table 5: Address configurations

The configuration of Table 5 allow communication only from the Ethernet node 0x0012 to the 0x6612 but not the viceversa. If node 0x012 would send a message to 0x6612, it will find-out that the destination node is not in its own subnet, that means readdress the frame to the SuperNode 0x0011.

Starting from this point, is the network and bridging structure that shall care of the frame coming from 0x0012. The bridge node receive the frame, its destination is 0x6612, rather the wireless interface for the bridge is on the 0x6500 subnet. It assume that a 0x6601 node exist and route the frame to it, rather from 0x6601 the frame finally is in 0x6612 because both are in the same subnet.

The reverse path from 0x6612 to 0x0012, result in a routed frame to 0x6601 that at this time has no information on how route the frame to other networks. It has a frame for 0x0012 that is on another media, it doesn't have a way to identify that the frame shall be router at 0x6511.

The problem can be solved adding an entry into the routing table of 0x6601, where all the frames with destination in subnet 0x0000 are routed via 0x6511.

Destination Subnet	Route to	Index in Routing table
0x0000	0x6511	0

Table 6: Routing table

For the supported hardware and software configuration follows 5.2.3.

5.3 Debugging Tools

Setting up a vNet network is quite easy and the examples included in the code allow an easy setup of similar cases. When the communication is not established, using the vNet debugging tools let easy understand the data flow between vNet nodes.

Debugging can be enabled using the flag VNET_DEBUG and enabling the serial port in the sketch, this let print all the incoming and outgoing messages involving the node. The printout contains the full frame, where the first six bytes are the header and the others the payload.