International Journal on Recent and Innovation Trends in Computing and Communication
Volume: 3 Issue: 5

ISSN: 2321-8169
2569 - 2573

Implementation of IPv6 in Embedded Device using LWIP TCP/IP Stack

Mohsin Kesarani
VLSI & Embedded Systems Design
Gujarat Technological University PG School
Ahmedabad, India
mfkesarani@gmail.com

Mr. Vikas Shirvastava
Project Leader
L&T Technology Services Ltd
Mumbai, India
vikasetc@gmail.com

Abstract—It is becoming trend of making every embedded devices connected wirelessly with each other. Due to large scale of deployment of IPv4 throughout internet, address spaces allowed by IPv4 is saturating and thus there is a strong need for implementing IPv6 into embedded devices. Despite the fact that there are numerous TCP/IP implementations for embedded and minimal systems, little research has been conducted in the area. Also currently, Allen Bradley's Power Flex Drives based on Bacnet card implements TCP/IP stack that do not support IPv6. Thus, IPv6 using LWIP Stack has been implemented and tested on Bacnet card (Renesas H8S/2556 microcontroller). This paper covers brief overview of LWIP TCP/IP Stack. It also presents a design method to implement IPv6 in embedded device. Furthermore it presents LWIP stack flow, Tx/Rx packet process in Ethernet controller and actual implementation results of IPv6 in Bacnet card based Power flex drive.

Keywords- *TCP/IP, IPv6, Renesas H8S, LWIP, Embedded Systems, Bacnet.*

*****

I. INTRODUCTION

Internet Protocol (IP) is the basic building block on which all Internet protocols are built. Applications that we take for granted today, such as Web browsers and e-mail, all use TCP and UDP, which in turn are built on top of IP. Because of new IoT concept, almost all the embedded devices are getting connected wirelessly with each other. In the last decade, the vast spread of network devices exhausted the IPv4 addresses, which consists of only 32 bits for one IP address. Therefore, IPv6 is designed as the solution. However, due to the large scale of IPv4 deployment, it is no doubt that the transition will take a long time. IETF IPng transition working group proposed three main approaches so as to ensure a smooth transition: dual stack, translation and tunneling [1]. Among them, dual stack is considered as most straight-forward way [2]. Therefore, it is widely adopted for embedded systems to support IPv4 and IPv6.

Currently there have been a number of TCP/IP stacks designed for embedded systems. Especially, LwIP [3] is considered as one of the most popular TCP/IP stack for embedded systems. It focuses on reducing the resource usage, while trying to preserve a full set of functions of TCP. Normally, it just takes up 40KB of ROM and less than 10KB of RAM during execution. Besides, LwIP can be deployed independent of operating systems. However, current implementation of LWIP on Bacnet card is just IPv4 based despite of some code for IPv6 which is commented [4].

This paper proposes an implementation of LWIP TCP/IP Stack supporting dual IPv4-IPv6 feature simultaneously on Bacnet card which incorporates Renesas H8S/2556 microcontroller in its core. Our contribution can be summarized as follows:

- First added platform specific IPv6 features into the stack preserving IPv4 at the same time.
- Proprietary RTOS has been used and interfaced with LWIP Stack for multiple threads handling into the stack.
- Porting has been done onto H8S/2556 microcontroller along with LAN9221 Ethernet controller.
- Finally, both TCP and UDP based webserver has been implemented over IPv6.

Experimental results show that this implementation takes up about 2591 bytes of Stack and 113942 bytes of Heap Memory during execution. It proves that it can effectively satisfy most embedded applications with limited resource.

The rest of this paper is organized as follows: Section II gives a brief description of LWIP TCP/IP Stack. Section III gives the implementation details of the stack on Bacnet card. Experimental results are shown in Section IV followed by conclusion in Section V.

II. LWIP TCP/IP STACK OVERVIEW

The lightweight Internet Protocol (lwIP) is a small independent implementation of the network protocol suite that has been initially developed by Adam Dunkels [3]. The focus of the lwIP network stack implementation is to reduce memory resource usage while still having a full scale TCP. This makes lwIP suitable for use in embedded systems with tens of kilobytes of free RAM and room for around 40 kilobytes of code ROM. lwIP supports the following protocols:
ARP, IPv4 and IPv6, TCP, UDP, DNS, DHCP, ICMP, IGMP, SNMP, PPP and PPPoE.

LwIP offers three different APIs designed for different purposes [5]:

- **Raw API** is the core API of lwIP. This API aims at providing the best performances while using a minimal code size. One drawback of this API is that it handles asynchronous events using callbacks which complexify the application design.
- **Netconn API** is a sequential API built on top of the Raw API. It allows multi-threaded operation and therefore requires an operating system. It is easier to use than the Raw API at the expense of lower performances and increased memory footprint.
  - **BSD Socket API** is a Berkeley like Socket implementation (Posix/BSD) built on top of the Netconn API. Its interest is portability. It shares the same drawback than the Netconn API.

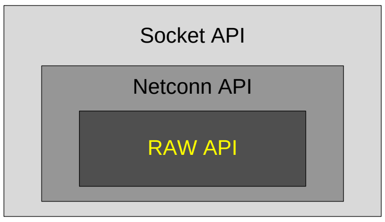


Figure 1. LWIP API Structure [5]

III. Implementation Details

First of all for lwIP stack implementation, *lwipopts.h* must be defined in the project's root folder by user that will contain various stack related options. Options not configured in lwipopts.h will be taken care by opt.h file of the stack. To enable IPv6 feature, *LWIP_IPV6* must be set in these header files. Also, since lan9221 ethernet controller has been used, its MAC control registers must be set to *pass all multicast packets* in order to allow IPv6 packets. To initialize the lwIP stack, the user application has to perform two functions calls from the main program loop:

- lan9221_if_input() to treat incoming packets (function defined in the network interface LAN9221 driver)

- sys_timers() to refresh and trigger the lwIP timers (defined in the user application)

A. BSD Socket API

The lwIP socket API is built on top of the Netconn API and offers portability for BSD socket based applications. In our implementation, we have used BSD Socket API. So, we will discuss BSD socket flow in lwIP. To enable BSD style socket support; the lwIP configuration file must define *LWIP_SOCKET* and *LWIP_COMPAT_SOCKETS*.

Fig. 2 gives an overview of an input packet processing while using the Socket API (Netconn based). Depending on the thread priorities, a minimum of 4 context switches is required to process one single TCP packet. If the user application requires maximum performances Raw API should be considered instead of the Netconn API [3].

As opposed to the Raw API approach, the LAN9221 driver does not process the TCP packet directly. Instead by calling the tcpip_input() function it notifies the lwIP core thread using the tcpip "mbox" mailbox that a packet is ready for processing. Then the lwIP core thread wakes up, reads the tcpip "mbox" message and starts the packet processing using the Raw API (calling ethernet_input() function, as shown in **Figure 3**). When a valid TCP packet is found, the lwIP core thread notifies the corresponding Netconn socket using the "recvmbox" mailbox.

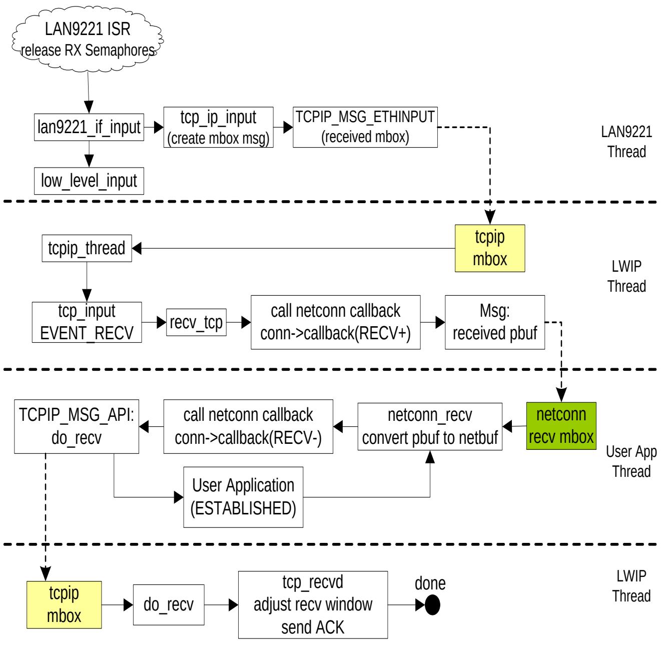


Figure 2. Input Packet Processing using Socket API [5]

From the user application point of view, when calling the netconn_recv() function, the user application thread waits for a message in the recvmbox to know if a TCP input packet has arrived. The user application can wait forever or for the specified amount of time if *LWIP_SO_RCVTIMEO* has been defined in the configuration file. When the "recvmbox" message is available, the user application thread wakes up and sends a notification message to the tcpip "mbox" to give a chance to acknowledge the packet and to adjust the receive window if necessary. During that time the user application thread waits for a notification semaphore only released by the lwIP core thread when the operation is completed. Finally, the netconn_recv() function returns the netbuf structure containing the TCP packet data to the user application.

The following mailbox sizes must be defined in the lwIP configuration file when using the Socket/Netconn API:

- TCPIP_MBOX_SIZE: size of the core tcpip mailbox.

ISSN: 2321-8169
2569 - 2573

- DEFAULT_ACCEPTMBOX_SIZE: size of the accept mailbox.
- DEFAULT_TCP_RECVMBOX_SIZE: size of the TCP recv mailbox.

B. LWIP Receive Stream

Fig. 3 shows the lwIP receive flow from the lan9221_if_input() function to the appropriate input protocol function in the lwIP core. The lan9221_if_input( ) function should typically be called from the main program loop.

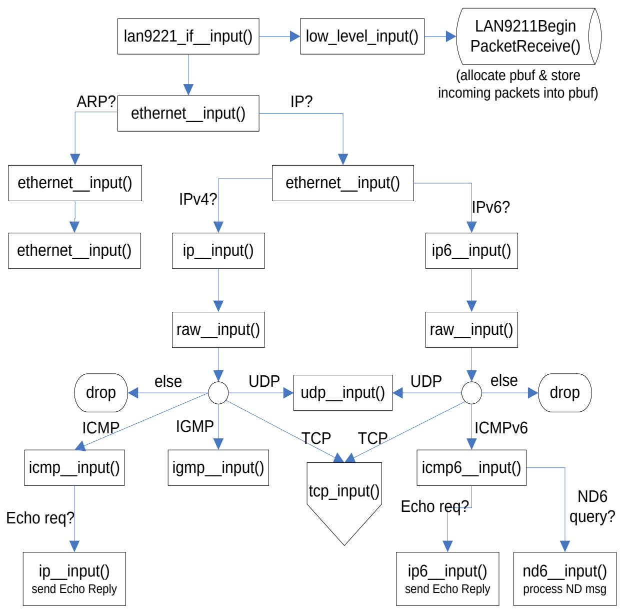


Figure 3. LWIP Receive Stream

Initially, ethernet_input() function is called from lan9221 driver for processing incoming ethernet packets. If received Ethernet frame is ARP then etharp_arp_input() is called and if frame is of IP type then etharp_ip_input() is called. Based on IPv4 or IPv6, corresponding input functions, ip_input() or ip6_input() checks for a valid IP checksum and ensures that the packet is addressed to the device. The raw_input() function tries to find a raw protocol control block (PCB) to handle the incoming packet. Raw PCBs are used to implement custom network protocols. If there is no raw PCB to be used, the appropriate input protocol function (ICMP, IGMP, TCP, UDP and ICMPv6) is called using the protocol field from the IP header.

C. LWIP TCP Input Flow

Once the received Ethernet frame is properly parsed and validated, tcp_input() function is called if protocol field in IP header is of TCP type. Then after processing of TCP segment is done. As shown in Fig. 4, the tcp_input() function tries to find the PCB keeping track of the connection used by the incoming packet (using IP and port numbers). The TCP checksum is verified, then depending on the incoming packet, the tcp_input() function will eventually inform the user application on specific events (like data sent, data received, etc) using the previously registered callbacks.

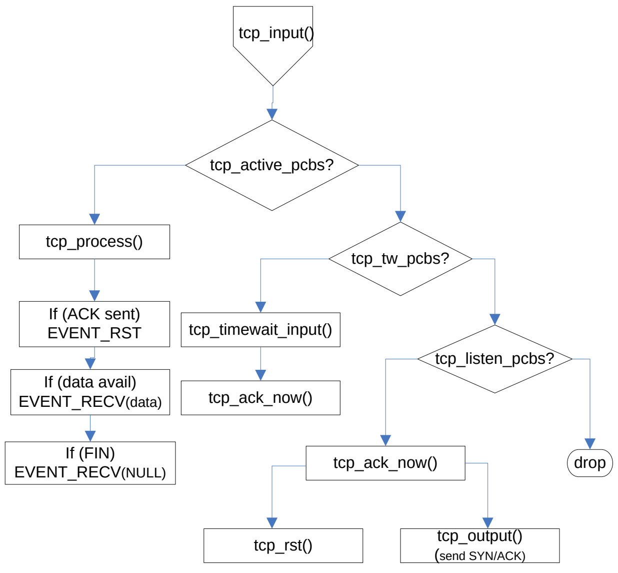


Figure 4. TCP Input flow of LWIP [5]

D. LWIP TCP Output Flow

The lwIP network stack provides the tcp_write() function for sending data to a remote host, as shown in Fig. 5.

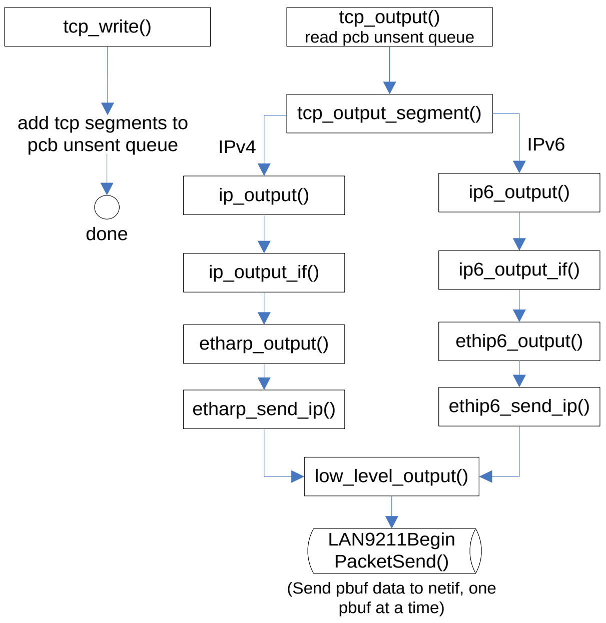


Figure 5. TCP Output flow of LWIP

It takes a pointer to the PCB structure (representing the active connection to write to), data buffer, data size and API flags as parameters. This function attempts to build TCP segments from the user data buffer. The TCP segments are then added to the PCB's unsent queue. Alternatively, the user data can be directly prepended to the last enqueued TCP segment if its remaining size does not exceed the *TCP_MSS* value.

The TCP segments are only sent when a call to the tcp_output() function is made, as shown in Fig. 5. This function is also automatically triggered by lwIP in the following cases:

- Inside the tcp_input() function (when TCP acknowledgement has to be sent right away)
- Inside the slow and fast timers (where retransmitting TCP segments can be required)

At this stage, the TCP segment gets encapsulated with the IPv4 or IPv6 header (ip_output() and ip6_output() function) and Ethernet header (etharp_output() and ethip6_output() function). Finally, the Ethernet frame is sent to the LAN9221 via the low_level_output() function located in the lwIP netif driver.

IV. IMPLEMENTATION RESULTS

Fig. 6 shows the proposed design structure which includes Power Flex drive. For IPv6 support in this drive, LWIP stack has been implemented and finally this Dual Stack supported device will be able to communicate with either only IPv4 enabled host or only IPv6 enabled.

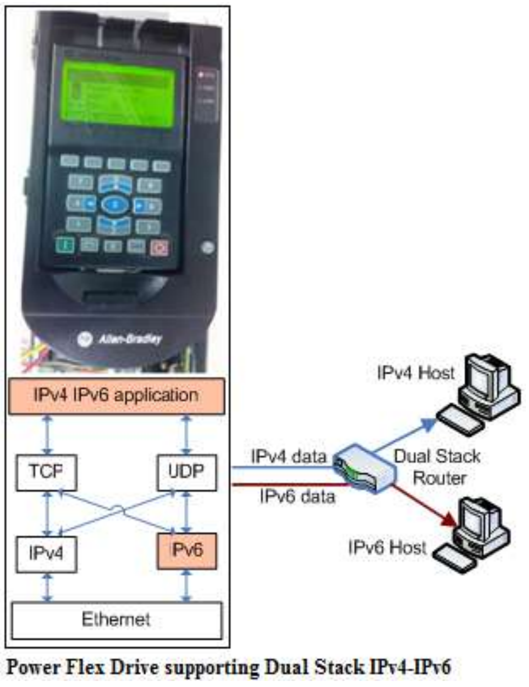


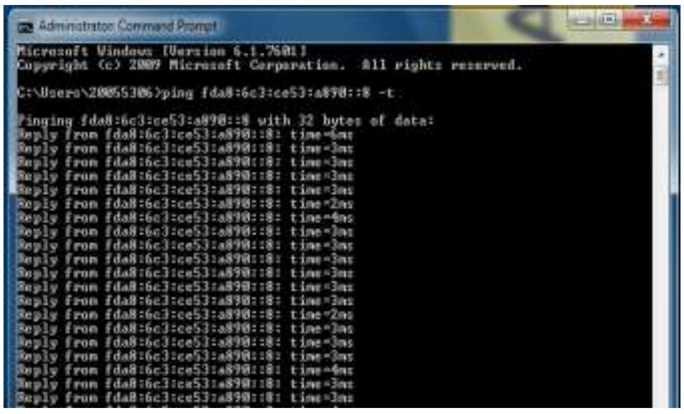
Figure 6. Proposed Design Structure.

A. Ping Test

LWIP Stack already has a function for creating link-local IPv6 address. But for configuring manual IPv6 address in Bacnet card, ***"netif_add_ip6_address( )"*** function has been defined which verifies available index in netif and adds the manual address. Thus, the IP address assigned to the device are:

IPv4 address of drive:	***10.9.208.210***
Link-local IPv6 address of drive:	***fe80::200:21ff:fe12:3224***
Manual IPv6 address assigned:	***fda8:6c3:ce53:a890::8***

In order to test IPv6 connectivity, drive has been connected with Windows-7 based PC and "ping" test was carried for both IPv4 and IPv6 simultaneously. The ping test results obtained are shown in Fig. 7 and Fig. 8 below:

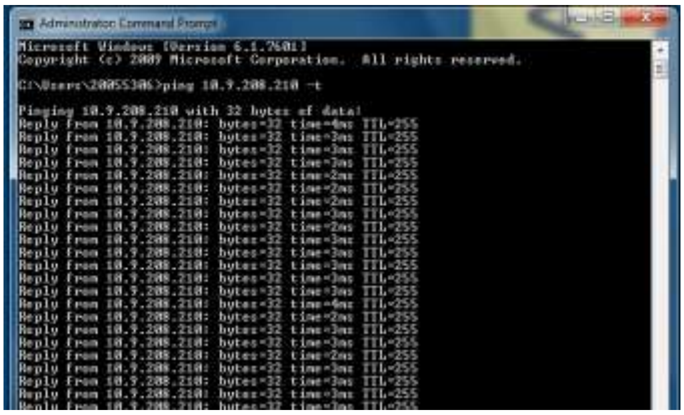


```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\20055306>ping fda8:6c3:ce53:a890::8 -t

Pinging fda8:6c3:ce53:a890::8 with 32 bytes of data:
Reply from fda8:6c3:ce53:a890::8: time=6ms
Reply from fda8:6c3:ce53:a890::8: time=3ms
Reply from fda8:6c3:ce53:a890::8: time=3ms
Reply from fda8:6c3:ce53:a890::8: time=3ms
Reply from fda8:6c3:ce53:a890::8: time=3ms
Reply from fda8:6c3:ce53:a890::8: time=2ms
Reply from fda8:6c3:ce53:a890::8: time=4ms
Reply from fda8:6c3:ce53:a890::8: time=3ms
Reply from fda8:6c3:ce53:a890::8: time=3ms
Reply from fda8:6c3:ce53:a890::8: time=3ms
Reply from fda8:6c3:ce53:a890::8: time=3ms
Reply from fda8:6c3:ce53:a890::8: time=3ms
Reply from fda8:6c3:ce53:a890::8: time=3ms
Reply from fda8:6c3:ce53:a890::8: time=2ms
Reply from fda8:6c3:ce53:a890::8: time=3ms
Reply from fda8:6c3:ce53:a890::8: time=3ms
Reply from fda8:6c3:ce53:a890::8: time=3ms
Reply from fda8:6c3:ce53:a890::8: time=4ms
Reply from fda8:6c3:ce53:a890::8: time=3ms
Reply from fda8:6c3:ce53:a890::8: time=3ms
```

Figure 7. IPv6 ping test of PF Drive (Bacnet card).



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\20055306>ping 10.9.208.210 -t

Pinging 10.9.208.210 with 32 bytes of data:
Reply from 10.9.208.210: bytes=32 time=4ms TTL=255
Reply from 10.9.208.210: bytes=32 time=3ms TTL=255
Reply from 10.9.208.210: bytes=32 time=3ms TTL=255
Reply from 10.9.208.210: bytes=32 time=3ms TTL=255
Reply from 10.9.208.210: bytes=32 time=2ms TTL=255
Reply from 10.9.208.210: bytes=32 time=2ms TTL=255
Reply from 10.9.208.210: bytes=32 time=2ms TTL=255
Reply from 10.9.208.210: bytes=32 time=3ms TTL=255
Reply from 10.9.208.210: bytes=32 time=2ms TTL=255
Reply from 10.9.208.210: bytes=32 time=3ms TTL=255
Reply from 10.9.208.210: bytes=32 time=3ms TTL=255
Reply from 10.9.208.210: bytes=32 time=3ms TTL=255
Reply from 10.9.208.210: bytes=32 time=3ms TTL=255
Reply from 10.9.208.210: bytes=32 time=3ms TTL=255
Reply from 10.9.208.210: bytes=32 time=4ms TTL=255
Reply from 10.9.208.210: bytes=32 time=2ms TTL=255
Reply from 10.9.208.210: bytes=32 time=3ms TTL=255
Reply from 10.9.208.210: bytes=32 time=3ms TTL=255
Reply from 10.9.208.210: bytes=32 time=2ms TTL=255
Reply from 10.9.208.210: bytes=32 time=3ms TTL=255
Reply from 10.9.208.210: bytes=32 time=3ms TTL=255
Reply from 10.9.208.210: bytes=32 time=3ms TTL=255
Reply from 10.9.208.210: bytes=32 time=3ms TTL=255
```

Figure 8. IPv4 ping test of PF Drive (Bacnet card).

B. Webserver Results over IPv6

As mentioned earlier, webserver has also been implemented using BSD Socket API of LWIP Stack. But currently at a time only one IP version works for webserver (TCP based). However ICMP is working for both IPv4 and IPv6. Webpages obtained using IPv6 address of drive is shown in the Fig. 9 below:

ISSN: 2321-8169
2569 - 2573

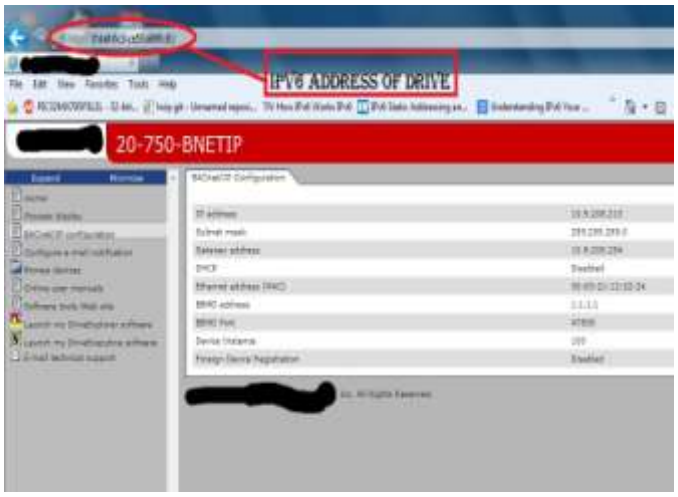
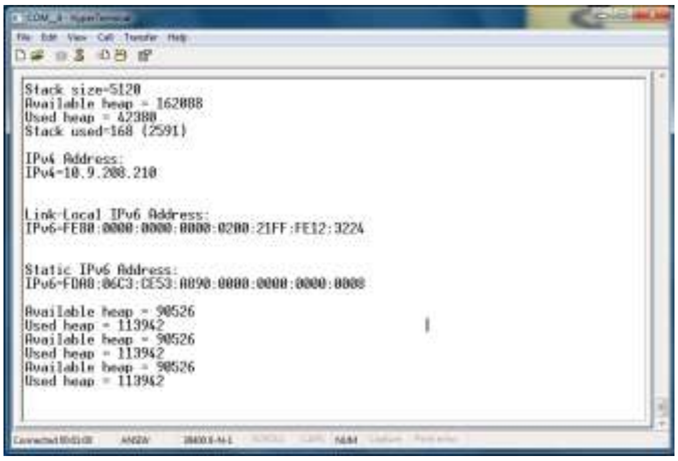


Figure 9. Snapshot of Webserver on Bacnet over IPv6

C. *Code Memory Results*

In order to analyse the memory consumed by the Stack, we are continuously monitoring the Stack memory and Heap memory obtained through UART logs. Thus the memory results obtained during Stack execution is show in Fig. 10 below:



```
Stack size=5120
Available heap = 162088
Used heap = 42380
Stack used=168 (2591)

IPv4 Address:
IPv4=10.9.208.210


Link-Local IPv6 Address:
IPv6=FE80:0000:0000:0000:0200:21FF:FE12:3224


Static IPv6 Address:
IPv6=FDA8:06C3:CE53:A090:0000:0000:0000:0008

Available heap = 90526
Used heap = 113942
Available heap = 90526
Used heap = 113942
Available heap = 90526
Used heap = 113942
```

Figure 10. Memory consumed during Stack execution

This output shows that out of total 5120 bytes of Stack Memory, only 2591 bytes are used whereas in case of Heap memory, just 113942 bytes are consumed.

V. CONCLUSION

This paper proposes design and implementation of LWIP TCP/IP Stack in embedded device especially Bacnet card. The Stack together with port and application codes is compiled and run under Bacnet card which is configured by Renesas H8S/2556 core [6] with built-in Ethernet support. Current implementation supports at a time any one of the IP version for webserver. However ICMP works concurrently for both IPv4 and IPv6 as shown in Fig. 8 and Fig. 9 of the ping test results.

Our future work includes support for dual version of IP in webserver so that webserver can be operated by both IPv4 and IPv6 simultaneously. Also, tunneling mechanism will be implemented using same LWIP Stack for IPv4 mapped IPv6 address [7] compatibility.

REFERENCES

[1] Ioan Raicu, Sheraili Zeadally, “Evaluating IPv4 to IPv6 transition mechanisms”, in 10th ICT, Tahiti, Papeete, French Polynesia,2003.

[2] Jiann-Liang Chen, Yao-Chung Chang and Chien-Hsiu Lin, “Performance investigation of IPv4/IPv6 transition mechanisms”, ICACT, Jan 2003,Gangwon-Do, Korea.

[3] A lightweight TCP/IP stack, Jan 2015. [Online] Available: http://savannah.nongnu.org/projects/ lwip/.

[4] Rockwell Automation, PowerFlex 753 Drives Manual revision 1.011. [Online] Available: http://www.rockwellautomation.com/support.

[5] Atmel Corporation, “AT04055: Using the lwIP Network Stack [APPLICATION NOTE]”, Initial document release, 42233A, March 2014.

[6] Renesas Technology Corp., “Renesas 16-Bit Single-Chip Microcomputer Hardware Manual H8S/2556 Group, H8S/2552 Group, H8S/2506 Group”, 1st Edition, March, 2003 Rev.5.00, September 27, 2007.

[7] Marc E. Fiuczynski, Vincent K. Lam, Brian N. Bershad, “The design and implementation of an IPv6/IPv4 network address and protocol translator”, in the Proceedings of USENIX Annual Technical Conference, Jun 1998, New Orleans, Louisiana