Latency Measurement Testbed for Wireless Sensor Networks

Sergio Lembo
Aalto University - School of Science and Technology
Department of Communications and Networking
P.O.Box 13000, 00076 Aalto, Finland

Abstract

In this paper we propose a testbed for measuring latency in Wireless Sensor Networks (WSN). The testbed was constructed using sensor nodes running the open-source 6LoWPAN stack Nanostack v1.1. The testbed is completed by connecting a workstation to the WSN through a gateway. The measurements are initiated from the workstation but the actual latency is measured inside the WSN.

I. Introduction

Advances in microelectronics, low-power electronics and micro-electro-mechanical systems (MEMS), along with low manufacturing costs, have led to the development of wireless sensor networks (WSN). These networks consist of individual devices that have been interconnected wirelessly in order to perform diverse tasks. Each one of these devices is referred to as a node, and in general it consists of a radio transceiver, microcontroller, energy source and one or more sensors.

Wireless sensor nodes are generally arranged in a wireless sensor network. Some network topologies are such that it is necessary to introduce a suitable routing algorithm in order to communicate nodes separated by more than one hop.

The time delay introduced by a WSN may become critical in certain scenarios, like in real-time control systems, and may vary according to the network topology and routing algorithm in use.

In this paper we propose a testbed for measuring latency in Wireless Sensor Networks. A testbed for measuring latency is necessary because latency measured by the nodes themselves can not be stored for posterior use in the nodes; nodes usually have a small and volatile memory. Therefore it is necessary to introduce a suitable scenario to measure latency, that also provides the means to initiate and store the measurement in an automated way, and without affecting the measured times. The testbed was constructed using sensor nodes communicating with IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) [1], and running the open-source 6LoWPAN stack Nanostack v1.1 [2][3] on on top of a real time kernel (FreeRTOS [4]).

In the following sections we first introduce the latency measurement testbed (Section II), then explain the latency measurement mechanism (Section III), next explain implementation details (Section IV) and finally conclude the paper in Section V.

II. Latency measurement Testbed

The testbed is constituted by one workstation and a wireless sensor network (WSN). The WSN consists of one or more generic nodes and one gateway (GW) node, Fig. 1. In our

testbed all the nodes implement the 6LoWPAN stack Nanostack v1.1. Two particular layers of this stack are of our interest in the proposed testbed, namely, the application layer and the MAC layer.

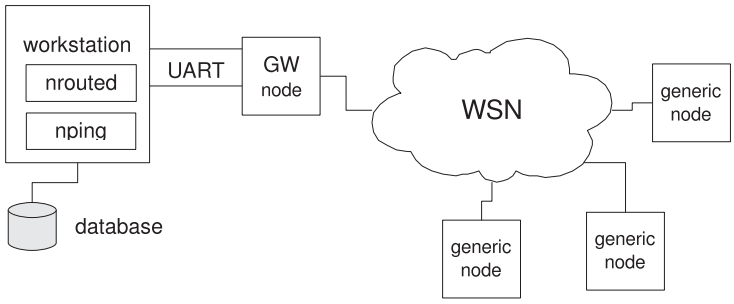

Fig. 1. Testbed scenario

The workstation is connected to the WSN through the GW and it is used for initiating the latency measurements, retrieving the measured times and storing these in a local database for posterior processing. The physical connection between the workstation and the GW is by means of a UART (universal asynchronous receiver/transmitter) interface.

In the GW the application layer performs a double task, first, receiving and sending packets from/to the WSN and second, communicating data through the UART interface. In this case, the application layer is actually a layer referred to as NRP layer. The layer was named NRP due that it uses a Network Routing Protocol (NRP) [5] to communicate the GW node with the external world. In practice the GW communicates with the workstation via the UART interface through the NRP layer in the stack, and packets circulating through the UART are encapsulated using the Network Routing Protocol (Fig. 2). The main idea is that a process running in the workstation (*nrouted* process) attends the other end of the UART communication line and helps to transmit and receive packets to the WSN. In Fig. 2 we depict a possible representation of the arrangement of a node acting as gateway, the UART communication line and the workstation.

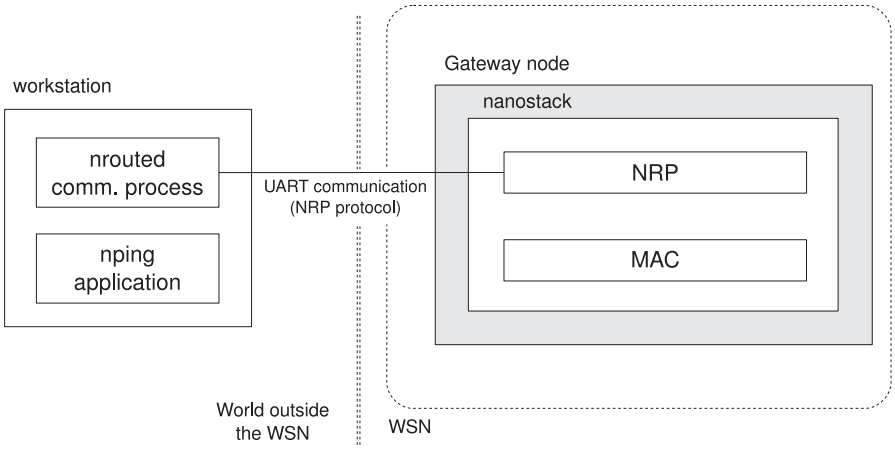

Fig. 2. Node implementing Nanostack with NRP layer, and UART communication to a workstation using NRP protocol

III. Latency measurement mechanism

Latency is measured in terms of the round trip time (RTT) measured from a packet leaving from the application layer of a GW node (actually NRP layer), traveling to a destination node where it is then echoed back by the application layer and returned to the application layer of the GW, Fig. 3.

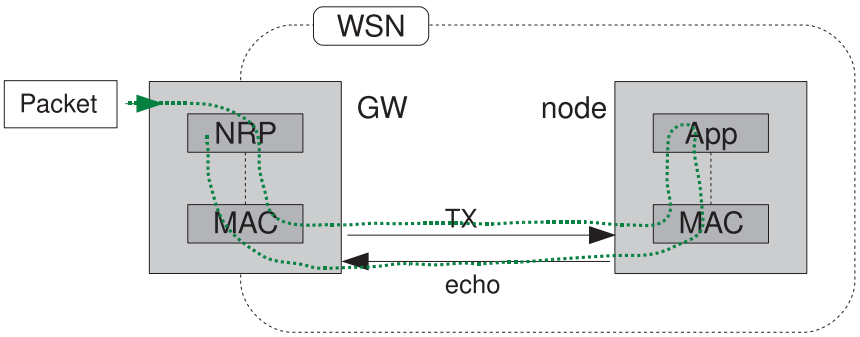

Fig. 3. Latency measurement scenario

The RTT is measured by adding a *start-time* timestamp in the data payload of a packet when the packet is leaving the NRP layer of the GW, and later adding an *end-time* timestamp and a *delta-time* indicator (*delta-time* = *end-time* − *start-time*) in the same data payload when the packet returns back to the GW, Fig. 4.

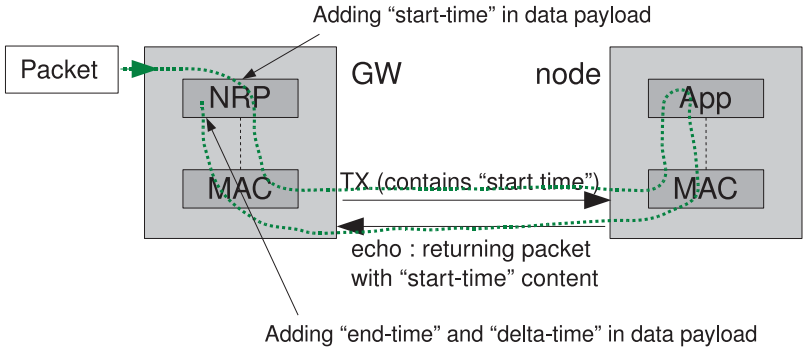

Fig. 4. Details of timestamps added to payload

Actually the *start-time* is added in the NRP layer, but the the packet is originated in the workstation, outside of the WSN . The *end-time* is also added in the GW, before sending back the packet to the workstation. In this sense the latency introduced by the UART communication is not added to the the latency measured in the WSN.

The 6LoWPAN packet is generated in the *nrouted* process but it originates in response to an external application, in this case the *nping* application (Fig. 2). *nping* provides to *nrouted* the addresses, port numbers and payload to be transmitted to the WSN. Then *nrouted* composes the 6LoWPAN packet and delivers it to the GW (Fig. 5).

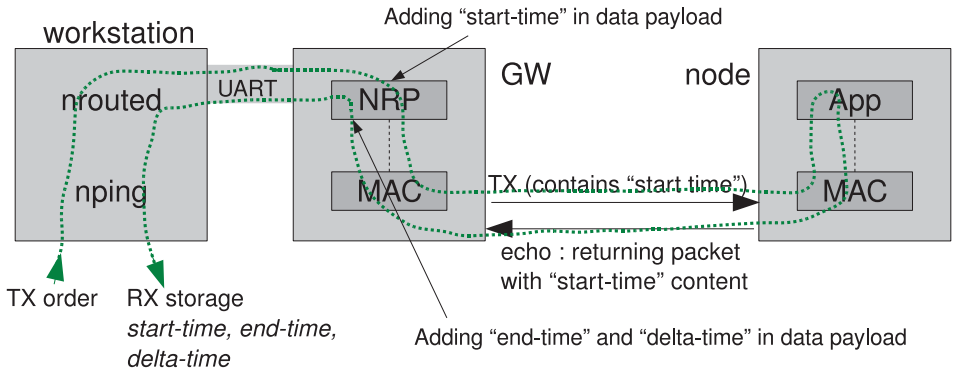

Fig. 5. Complete testbed scenario

Once the payload of the echoed packet arrives to the *nping* application, it contains in the payload the *start-time*, *end-time* and *delta-time*. These times are stored in a local database for posterior analysis of the latency times.

IV. Implementation

The delivered data payload contains an indicator in the first octet (0x50) that is recognized by the NRP layer in the GW when the packet is in the outgoing and incoming direction. This octet is added in the payload by *nping*.

The data in the packet is arranged as shown in Fig. 6. In the figure we observe the initial octet 0x50, used to indicate that we intend to measure latency in the given testbed. Then the payload is followed by the *start-time*, *end-time* and *delta-time* separated by 0xFF octets. In addition the payload contains a scaling factor to convert time expressed in system ticks to milliseconds.

Fig. 6. Payload format

Actually all the times in the operating system are expressed in ticks [6]. A tick is a counter incremented by a timer interrupt at a predefined rate. In our tests we used the tick rate 0x0A = 10, so all the times should be multiplied by the conversion factor 10 to express these in *ms*. In the example shown in Fig. 6 the round-trip time is 0x16 * 0x0A = 220 *ms*. The ninth octet in the payload contains the conversion factor (portTICK_RATE_MS) to convert ticks to *ms*.

The times are represented with only 16 bits in 32 bits words, with the 16 most significant bits set to 0x00. This comes from a configuration in the stack using the setting configUSE_16_BIT_TICKS. The alternative option, timer of 32 bits, is not completely supported in the implementation. Times were not converted to *ms* to avoid complexity in the structure

of the packet and additional delay introduced by the conversion. Note that multiplying a 16 bit number by 5 or 10 to convert ticks to *ms* will require more than 16 bits for the result, or some mapping to use 16 bits; we avoided such complications by leaving the times in ticks.

The code to support this measurement was totally implemented in the NRP layer, file nrp.c (available in example nano_router_n60x in Nanostack v1.1 distribution). The patch to this file is shown in appendix A. In our implementation the location of the data in the payload is fixed and only works with compressed UDP ports.

The destination node needs a minor implementation at the application layer to echo back a packet, optionally recognizing the indicator 0x50, but no additional packet handling is necessary in the node.

The program tool *nping* is used to trigger the generation of the packet with the patch provided in Appendix B.

V. Conclusion

In this paper we proposed a testbed for measuring latency in Wireless Sensor Networks (WSN) using the 6LoWPAN stack Nanostack v1.1. With this testbed we can analyze the performance of different network topologies and routing algorithms in terms of latency. The testbed measures the latency time in terms of round trip time of a packet leaving a gateway and reaching a target destination node. The packet is originated outside the WSN but the actual latency measurement is within the WSN. The measured times are stored in a database outside the WSN for posterior analysis.

References

[1] Montenegro, G., Kushalnagar, N., Hui, J., Culler, D.: Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944 (Proposed Standard) (September 2007)
[2] NanoStack 6lowpan: Nanostack v1.1 (2008) http://sourceforge.net/projects/nanostack/.
[3] Lembo, S., Kuusisto, J., Manner, J.: Internal Map of the Nanostack 6LoWPAN Stack. In Meghanathan, N., Boumerdassi, S., Chaki, N., Nagamalai, D., eds.: Recent Trends in Networks and Communications. Volume 90 of Communications in Computer and Information Science. Springer Berlin Heidelberg (2010) 619–633
[4] FreeRTOS: The FreeRTOS project (2009) http://www.freertos.org/.
[5] Sensinode Ltd.: nRoute Protocol Specification, v0.7 (2006) http://sourceforge.net/projects/nanostack/ (distributed with Nanostack v1.1 source code).
[6] FreeRTOS: The RTOS Tick (2010) http://www.freertos.org/implementation/a00011.html.

Appendix A

Patch for file nrp.c in Nanostack v1.1 distribution incorporating the timestamps for the time measurement.

```
diff --git a/Examples/nano_router_n60x/nrp.c b/Examples/nano_router_n60x/nrp.c
index 190df09..c7ecd41 100644
--- a/Examples/nano_router_n60x/nrp.c
+++ b/Examples/nano_router_n60x/nrp.c
@@ -87,6 +87,9 @@ extern portCHAR nrp_check( buffer_t *buf );
 #define NRP_FILTERS 4
 #endif

+
+#define TKK_LATENCY_MEAS
+
 uint8_t nrp_router_mode;
 uint8_t nrp_router_advertise_period;

@@ -208,6 +211,98 @@ portCHAR nrp_handle( buffer_t *buf )
                         }
                         else
                         {
+
+                                /* TKK_LATENCY_MEAS */
+#ifdef TKK_LATENCY_MEAS
+
```

```
+                               uint8_t *ptr;
+                               portTickType curtime=0;
+
+#if( configUSE_16_BIT_TICKS == 1 )
+                               uint16_t ucurtime=0;
+                               uint16_t usenttime=0;
+                               uint16_t udeltatime=0;
+#else
+                               uint32_t ucurtime=0;
+                               uint32_t usenttime=0;
+                               uint32_t udeltatime=0;
+#endif
+
+                               /* pointing to start of data payload
+                                * hardcoded !, note that it assumes compressed UDP ports */
+                               ptr = buf->buf + 28;
+
+                               /* if the packet contains indicator 0x50 then add remaining info
+                                * end-time and delta-time in the payload */
+                               if(*ptr == 0x50)
+                               {
+
+                                       /* jump the start-time and write after it */
+                                       ptr+= 7;
+
+                                       /* cur time */
+                                       curtime = xTaskGetTickCount();
+
+                                       /* write in payload end-time and delta-time [TKK] */
+#if( configUSE_16_BIT_TICKS == 1 )
+                                       *ptr++ = 0xFF; /* not used */
+                                       *ptr++ = portTICK_RATE_MS;
+                                       *ptr++ = 0xFF; /* not used */
+                                       ucurtime = curtime;
+                                       *ptr++ = 0x00;
+                                       *ptr++ = 0x00;
+                                       *ptr++ = (ucurtime >> 8);
+                                       *ptr++ = (ucurtime);
+
+                                       *ptr++ = 0xFF; /* not used */
+                                       *ptr++ = 0xFF; /* not used */
+
+                                       /* Get start-time = sent-time */
+                                       /* note, hardcoded positions in buf, assuming compressed udp ports */
+                                       usenttime= (*(buf->buf + 33)) * 256 + *(buf->buf + 34);
+
+                                       /* boundary condition when the tick counter resets */
+                                       if(usenttime > ucurtime)
+                                       {
+
+                                               /* delta time */
+                                               udeltatime = (0xFFFF  - usenttime) + ucurtime;
+                                       }
+                                       else
+                                       {
+                                               udeltatime = ucurtime - usenttime;
+                                       }
+
+                                       /* Delta time */
+                                       *ptr++ = 0x00;
+                                       *ptr++ = 0x00;
+                                       *ptr++ = (udeltatime >> 8);
+                                       *ptr++ = (udeltatime);
+
+                                       *ptr++ = 0xFF;
+
+#else  /* 32 bit ticks not used */
+                                       *ptr++ = 0xFF;
+                                       *ptr++ = 0xFF;
+                                       ucurtime = curtime;
+                                       *ptr++ = (ucurtime >> 24);
+                                       *ptr++ = (ucurtime >> 16);
+                                       *ptr++ = (ucurtime >> 8);
+                                       *ptr++ = (ucurtime);
+
+                                       *ptr++ = 0xFF;
+                                       *ptr++ = 0xFF;
+
+                                       /* Delta time not implemented for this case*/
+
+                                       *ptr++ = 0x00;
+                                       *ptr++ = 0x00;
+                                       *ptr++ = 0x00;
+                                       *ptr++ = 0x00;
+#endif
+                               }
+#endif /* TKK_LATENCY_MEAS */
+
                                nrp_tx_add(buf);
                        }
                        break;
@@ -512,10 +612,60 @@ void nrp_tag_parse(void)
```

```
        {
                case NRP_PDATA:
                        {
+#ifdef TKK_LATENCY_MEAS
+                               portTickType curtime=0;
+
+#if( configUSE_16_BIT_TICKS == 1 )
+                               uint16_t ucurtime=0;
+#else
+                               uint32_t ucurtime=0;
+#endif
+#endif /* TKK_LATENCY_MEAS */
+
                                uint8_t *ptr = nrp_rx_buffer->buf;
                                ptr += nrp_rx_buffer->buf_end;
                                memcpy(ptr, nrp_tag_buffer, nrp_tag_len);
                                nrp_rx_buffer->buf_end += nrp_tag_len;
+
+#ifdef TKK_LATENCY_MEAS
+
+                               /* if the packet contains protocol 0x50 (latency time meas.)
+                                * add current time [TKK] */
+                               ptr = nrp_rx_buffer->buf;
+
+                               if(*ptr == 0x50)
+                               {
+
+                                       /* cur time = start-time */
+                                       curtime = xTaskGetTickCount();
+
+                                       /* add start-time */
+#if( configUSE_16_BIT_TICKS == 1 )
+                                       *ptr++ = 0x50;
+                                       *ptr++ = 0xFF; /* not used */
+                                       *ptr++ = 0xFF; /* not used */
+                                       ucurtime = curtime;
+                                       *ptr++ = 0x00;
+                                       *ptr++ = 0x00;
+                                       *ptr++ = (ucurtime >> 8);
+                                       *ptr++ = (ucurtime);
+                                       *ptr++ = 0x00;
+
+#else /* 32 bit ticks not used */
+                                       *ptr++ = 0x50;
+                                       *ptr++ = 0xFF;
+                                       *ptr++ = 0xFF;
+                                       ucurtime = curtime;
+                                       *ptr++ = (ucurtime >> 24);
+                                       *ptr++ = (ucurtime >> 16);
+                                       *ptr++ = (ucurtime >> 8);
+                                       *ptr++ = (ucurtime);
+                                       *ptr++ = 0x00;
+#endif
+                               }
+#endif /* TKK_LATENCY_MEAS */
+
                        }
                        break;
```

APPENDIX B

Patch for file nPing.c in Nanostack v1.1 distribution incorporating the code to trigger the generation of timestamps.

After adding the patches execute *nping* as follows, using a suitable address for the destination node.

```
./nPing -v -l 24 -t 00:00:00:00:00:00:00:00
```

```
diff --git a/Tools/nPing/nPing.c b/Tools/nPing/nPing.c
index f945eb6..57bd19a 100644
--- a/Tools/nPing/nPing.c
+++ b/Tools/nPing/nPing.c
@@ -140,6 +142,9 @@ int main(int argc, char **argv)

        unsigned short int source_port = 253;
        unsigned short int dst_port = 254;
+
+       /* protocol number in use */
+       unsigned int uiProtocolNumber = 0;

        unsigned char *nPing_packet;
 /*     And the length of the packet. */
```

```
@@ -273,6 +279,11 @@ int main(int argc, char **argv)

                        bzero(argbuf, 64);
                }
+               /* measure round trip time */
+               else if(strncmp(argbuf, "-t", 2) == 0)
+               {
+                       uiProtocolNumber = 50;
+               }
                else
                {
                        /*      Target address. */
@@ -372,13 +385,21 @@ int main(int argc, char **argv)
 /*     And now lets fill in the "random data" to the payload. */
        for(plen=0;plen<nPing_opts.payload;plen++)
        {
-               nPing_packet[plen] = 0xED;
+               /* nPing_packet[plen] = 0xED; */
+               nPing_packet[plen] = plen;
        }
+
+       /* compose payload according to protocol in use */
+    if(uiProtocolNumber == 50)
+       {
+               nPing_packet[0] = 0x50;
+       }

+
        while(counter < nPing_opts.count || nPing_opts.count == -1)
        {
                /*      Two first bytes are the sequence number. */
-               nPing_packet[0] = (seq >> 8);
+               /*nPing_packet[0] = (seq >> 8); */
                nPing_packet[1] = (seq & 0xff);

                /*      Create the nRP data packet using the nPing_packet[] as the data field. */
@@ -465,9 +488,13 @@ re_read:
 re_process:
                                        libnrp_get_data(rbuf, (unsigned int)i, data, &dlen);

-                                       rcv_seq = (data[0] << 8);
+                                       /*rcv_seq = (data[0] << 8); */
+                                       rcv_seq = 0;
+
                                        rcv_seq += data[1];

+                                       /* Add Database storage function call here */
+
                                        if(rcv_seq == seq - 1)
                                        {
                                                if(libnrp_get_source_address(rbuf, i, &source_addr_type, \\
                                                source_addr_bytes, source_addr_string) == 1)
```