Nigerian Journal of Technology (NIJOTECH)
Vol. 35, No. 1, January 2016, pp. **144 – 154**
Copyright© Faculty of Engineering, University of Nigeria, Nsukka,
Print ISSN: 0331-8443, Electronic ISSN: 2467-8821
www.nijotech.com
http://dx.doi.org/10.4314/njt.v35i1.22

PRECISE DELTA EXTRACTION SCHEME FOR REPROGRAMMING OF WIRELESS SENSOR NODES

E. M. Eronu[1],* and S. Misra[2]
[1] DEPARTMENT OF ELECTRICAL/ELECTRONIC ENGINEERING, UNIVERSITY OF ABUJA, GWAGWALADA, ABUJA. NIGERIA
[2] DEPARTMENT OF COMPUTER SCIENCE, COVENANT UNIVERSITY, OTTA, OGUN STATE. NIGERIA.
E-mail addresses: *[1] majieronu2007@hotmail.com, [2] ssopam@gmail.com*

ABSTRACT

In this paper, we present a precise delta extraction scheme and tool for use in wireless sensor network reprogramming processes. Our approach involves the use of a novel algorithm based on SET theory and the unique pattern of the Execution Link File (ELF) structure to extract delta from two distinct firmware (original and the modified). The delta consist of two set of unique values: one set clearly indicate the address of where the change has occurred and the second relays the change Data content. In addition, we developed a set of metrics that relays the degree of modification made with respect to the original file. The scheme capabilities, when compared with similar utilities referred in literature, shows an appreciable capacity to reduce energy consumption rate as well as effect a reduction in the amount of memory space used during reprogramming processes.

Keywords- *reprogramming; operating system, wireless sensor network, Delta*

1. INTRODUCTION

Wireless Sensor Networks (WSN) are mostly deployed in remote locations and are compose of a large number of sensor nodes that are memory constrained and deficient in sustainable energy supply. WSN has been helpful in several applications where remote monitoring and control are the needful [1, 2]. Interestingly, the need to alter a WSN application initial set objectives is a common occurrence in real time. To effect these changes, the wireless sensor nodes will have to be reprogrammed via serial communication or other known direct approaches. However, this is not possible because sensor nodes in most cases once deployed in challenging and inaccessible terrain cannot be easily reached. The only option appropriate and commonly in use is the Over-the-Air remote reprogramming method.

In several works [3, 4, 5], the Over-the-Air reprogramming method has been effectively implemented via the use of embedded operating systems. Jun-Zhao in [6] as well as Dunkiels *et al.* in [7] categorized the reprogramming processes into three, namely*: full code image replacement* approach, the *loadable module* approach and the *incremental-differential* approach. Each approach evolved from the need to improve upon the drawbacks of the others.

This work addresses energy consumption rate related issues attributed to certain utilities employed in reprogramming processes based on the *incremental-differential* approach. Section two provides detailed background information on the issues being addressed based on reviewed selected works of literature. A description of the materials and methods employed in designing, realising and evaluating the Precise Delta Extraction (PDE) scheme are conveyed in section three. In section four, we discuss the results obtained from these evaluations; and in additions; highlight some of the advantages the PDE scheme has over existing delta extraction utilities. Section five concludes the paper.

2 REVIEW OF RELATED LITERATURE

The *full image* replacement approach involves overwriting existing system image currently running on the sensor nodes with a compiled binary image of the new application software and the operating system inclusive. Examples of sensor networks reprogramming software employing this approach implemented in TinyOS [8, 9] are XNP and Deluge [10]. Next in line is the *loadable module* based

***Corresponding author, Tel: +234-803-392-7733.**

approach implemented on modularized operating system architectures (e.g. LiteOS [11] and Contiki [7, 12, 13]). It entails the transmission of only modified modules that are then linked and loaded by the operating system. In comparison to the *full image replacement* based approach, the *loadable module* based approach is more suitable for over-the-air reprogramming because only updated modules are transmitted. It also allows for the addition and removal of new application task packaged in modular form. However, use of large memory space, demand for more processing time that invariably translates to higher power consumption and slow system execution are drawbacks associated with the *loadable module* based approach. The *incremental-differential* based approach scheme transmits only the delta thereby reducing the amount of data needed to be transferred (especially when only small changes are involved like bug-fixes).

A review of existing reconfiguration scheme currently in use shows that the *incremental-differential* approach method is more promising when compared to others [14]. A further review of the *incremental-differential* based approach reveals that in some cases, instead of smaller deltas being generated, larger ones were rather produced [3]. A problem largely attributed to the traditional *incremental-differential* utilities employed in delta generation (Rsync [15] employed in [3, 16] and Clone Detection [17] used in [18]).

The majority of *incremental-differential* reconfiguration approach employs software applications that extract differences between the original source code and the modified application source codes. Most rely on the Rsync algorithm [15], though several extension or modifications have been made to the original work meant for larger system's networks to suit the WSN platform [19, 20] . The utilities were inherently not designed to handle file structures well-matched for sensor network data transmission and dissemination.

The PDE scheme presented in this paper provides a software tool and set of metrics for extracting precise delta information to address the issues attributed to the Rysnc utility and its variants. The Scheme has measuring capabilities that reports the degree of changes at various sections or segments. Hence, it is used in measuring the extent of firmware modification resulting from the addition or removal of software coding elements (variables, constants, functions, etc.). By extension, it is helpful in detecting firmware cloning.

3 METHODOLOGY

The approach adopted involves the use of a novel algorithm based on SET theory and the unique pattern of the Execution Link File (ELF) structure to design the scheme. The design enables the extraction of the delta from two distinct firmware (original and the modified) express as functions of their constituent set of bytes. The delta consists of two unique values. One clearly indicates the address of where the change has occurred and the second relays the change noticeable in the Content/Data. In addition, we developed a set of metrics that relays the degree of modification made with respect to the original file. The entire scheme is implemented using the Language Integrated Query (LINQ) and the Microsoft C# programming tool [21]. LINQ is a programming model that introduces queries as a first-class concept into any Microsoft.NET Framework language. LINQ provides a methodology that simplifies and unifies the implementation of any data access [22, 23].

3.1 Design and Implementation

Program modification can occur in any of the ways listed below:

- Adding new functionalities or data (for example, constants, variables, program constructs)
- Removing no longer needed functionalities and related data.
- Updating existing functions or data content.

Let

$$\mathrm{A} = \left\{x \middle| \begin{matrix} all\ bytes\ making\ up\ the\ firmware\ of\ the\ original \\ source\ code \end{matrix}\right\}$$

and

$$\mathrm{B} = \left\{x \middle| \begin{matrix} all\ bytes\ making\ up\ the\ firmware\ of\ modified \\ source\ code \end{matrix}\right\}$$

Now,

$$\Delta^{+} = B \backslash A$$

$$\therefore\ \Delta^{+} \Rightarrow\ Added\ set\ of\ code\ with\ significant\ increase\ in\ |B|$$

Also,

$$\Delta^{-} = A \backslash B$$

$$\therefore\ \Delta^{-} \Rightarrow Removed\ \ set\ of\ codes\ with\ significant\ decrease\ in\ |A|$$

However, modification could take place without a significant change in the number of elements contained in either A or B. Such occurrences can be represented as $\Delta^{\mp}$.

Descriptions of the symbols used in the mathematical modelling of the PDE scheme are given in Table 1 and Table 2 respectively. The symbols used were based on the structure of the Execution Link File (ELF) format [24].

Table 1: Description of ELF membership type symbols [24]

Member Types	Description
$PH.ptype$	Type of segment this array element describes
$SH.sh_{addr}$	Section's physical address
$PH.p_{addr}$	Segment's physical address
$PH.p_{filez}$	The number of bytes in the file image of the segment
$SH.sh_{filsize}$	The number of bytes in the file image of the section
$SH.sh_{flags}$	Flags relevant to the segment

Table 2: Description of ELF memberships' attributes type symbols [24]

Attributes	Description
PH_{T_LOAD}	The array element specifies a loadable segment
SHF_{Alloc}	The section occupies memory during process execution
$SHF_{EXECINSTR}$	The section contains executable machine instructions

Let SEG = a collection of seg_j with the *PH.p_type* attributes $=PH_{T\_LOAD}$:

$$SEG = \left\{ \bigcup_{j=0}^{n-1} seg_j \middle| PH.ptype = PH_{T_{LOAD}} \right\} \quad (1)$$

Where PH ⇒ Program Header and *n = number of segments:*

And let

$$A = sec_i.SH.sh_{addr} \in [seg_{j.}PH.p_{addr}, seg_{j.}PH.p_{addr} + seg_{j.}PH.p_{filez}] \quad (2)$$

$$B = sec_i.SH.sh_{filsize} \neq 0 \quad (3)$$

$$C = sec_i.SH.sh_{flags} = SHF_{Alloc} \quad (4)$$

$$D = sec_i.SH.sh_{flags} = SHF_{EXECINSTR} \quad (5)$$

Where *SH* is the Section Header and m is the number of sections then the elements of seg_j consists of a collection of sections sec_i expressed thus:

$$seg_j = \left\{ \bigcup_{i=0}^{m-1} sec_i \mid A \& B \ V \ C \ VD \right\} \quad (6)$$

From each section contained in seg_j, a unique address value ($Uaddr_k$) is derived for each instruction code/data by concatenating values of segment number (j), section number (i) and the position (p) of each instruction/data (D_k).

$$Uaddr_k \quad = j + i + p \quad for\ k = 0 \rightarrow \sum_{j=0}^{m-1} |seg_j| \quad (7)$$

The addressing scheme uniquely identifies an associated instruction code/data contained in the entire loadable file. In order to identify changes (Δ^+, Δ^- *and* $\Delta^\mp$) resulting from reprogramming or reconfiguration processes, seg_j are obtained for the original file's ELF (F_{orig}) and the modified version (F_{mod}) respectively. Subsequently, while using $Uaddr_k$ as a reference, each D_k within sec_i of respective seg_j are compared and where there are differences, they are reported as either modified set of codes ($\Delta^\mp$), added set of codes (Δ^+) or removed set of codes (Δ^-) appropriately. Algorithm 1 listing shows the algorithm employed for the PDE.

Algorithm 1: Precision Delta Extraction (PDE) Implementation

```
From SEG obtain a collection of seg
{
For each seg, obtain a collection of sec
{
For each sec collection
    {
    Compare associated contents ($D_k$) of F$_{orig}$ and F$_{mod}$ as addressed by
        unique address value ($Uaddr_k$)
    {
    Case (contents = equal )   :  ignore
    Case (contents = different) : report as modified, note address, count number of
        occurrence(s)
    Case ($Uaddr_k$ contained in F$_{orign}$ does not exist in F$_{mod}$) :  a deletion of code(s) has taken
        place , note address, count number of occurrence(s)
    Case ($Uaddr_k$contained in F$_{mod}$ does not exist in F$_{orig}$) :  an addition of code(s) has taken
        place, note address, count number of occurrence(s)
                                        }
                                }
                        }
        }
```

3.2 Measuring the degree of Δ^+, Δ^- and $\Delta^\mp$ in relation to the original firmware size (Distortion Metrics)

Let m, n and *p* represent the total number of segments, sections and bytes/words respectively, Likewise:

$Tsec_i$ is the Total Number of bytes /words contained in a section, $Tseg_j$ is the Total Number of bytes /words contained in a segment and T_f is the Total Number of bytes /words contained in the file. These terms can be obtained thus:

$$\mathrm{Tsec_i} = |sec_i| \tag{8}$$

$$\mathrm{Tseg_j} = \sum_{i=0}^{n-1} |sec_i| \tag{9}$$

$$\mathrm{T_f} = \sum_{j=0}^{m-1} |seg_j| \tag{10}$$

Where δ represents the degree of changes effected, the value δ can be obtained thus:

$$\delta = \left(\frac{T_f\left(F_{orig}\right) - T_f\left(F_{mod}\right)}{T_f\left(F_{orig}\right)} \right) \tag{11}$$

Based on the value of δ, the following can be inferred:

i. When($\delta < 0$), it implies that a set of codes has been added and possibly some of the original codes could have been modified as well.

ii. When($\delta > 0$), it implies that a set of codes has been removed and possibly some of the original codes could have been modified as well.

iii. When($\delta = 0$), it implies that no change has taken place. However, it is possible that some of the original codes could have been modified as well.

3.3 PDE Evaluation

We evaluated the PDE by employing Sample application source codes' ELF files derived from the GNU C compiler customised for the Contiki operating system. Each of the sample files' source codes was altered or modified in response to changes emanating from evolving application needs. Typically, these changes could involve or span over variables, constants, function names, libraries and other source code constructs. However, in this work the changes were confined to variation involving constants, variables and Function names only.

Having implemented these changes, the modified files were then recompiled to obtain new ELF files. Each pair of generated ELF files (original and modified) were further processed using the PDE. The PDE, by design, outputs a dataset, which contains a collection of delta (the data difference(s) between the original and modified files) and their respective address or addresses where applicable. In addition, the PDE produces three reports: the first and second reports are printouts of ELF constituents (available sections, data contents, and their respective addresses) of both the original and modified files respectively. The third report relays the changes detected in the two files. Figure 1 shows the front end of the PDE application developed using C-sharp programming tools, and Figure 2 shows an additional form that displays ELF profile information of application firmware.

As indicated in the Figure 1, the original and modified application's ELF constituents (generated unified address, physical address, data, list of loadable segments and segments related addresses and size) as well as the generated delta are displayed using the list view object components labelled as 'Original' , 'Modified' and 'Delta' respectively.

4. RESULTS AND DISCUSSION

In consonant with section 3.3 we demonstrate the performance of the PDE using an application sample 'remotepowerswitch.c' built on the Contiki OS [25]. Changes effected at various source code' program structure were applied to each application's source code, each of the ensuing modified files paired with the original was compiled and their subsequent ELF files fed into the PDE. The Delta obtained, and other relevant information provided on the ELF profile form, are presented under related subsections 4.1, 4.2 and 4.3.

Table 3: List of 'remotepowerswitch.elf' ELF constituents

Segment Number	Number of Sections	Segment Byte size	Segment Flags
0	465	79, 988	Execute , Read
1	65	2,152	Execute , Read
2	4	1,788	Write, Read
3	1	0	Read
4	15	8,344	Write, Read
5	2	912	Execute , Read
6	1	36	Execute , Read
7	1	4	Execute , Read
8	1	4	Execute , Read
9	1	4	Execute , Read
10	1	4	Execute , Read
Total bytes contained in File		93,236	

4.1 ELF Profile of the 'Remote Power Switch' Sample Application

Using the ELF profile front end of the PDE, the constituents of the generated 'remotepowerswitch.elf' form in its original state (without any modifications) are presented in Table 3. The Profile's front end as shown in Figure 3 indicates where these constituents were obtained from. In addition, The ELF profile front end provides the following information:

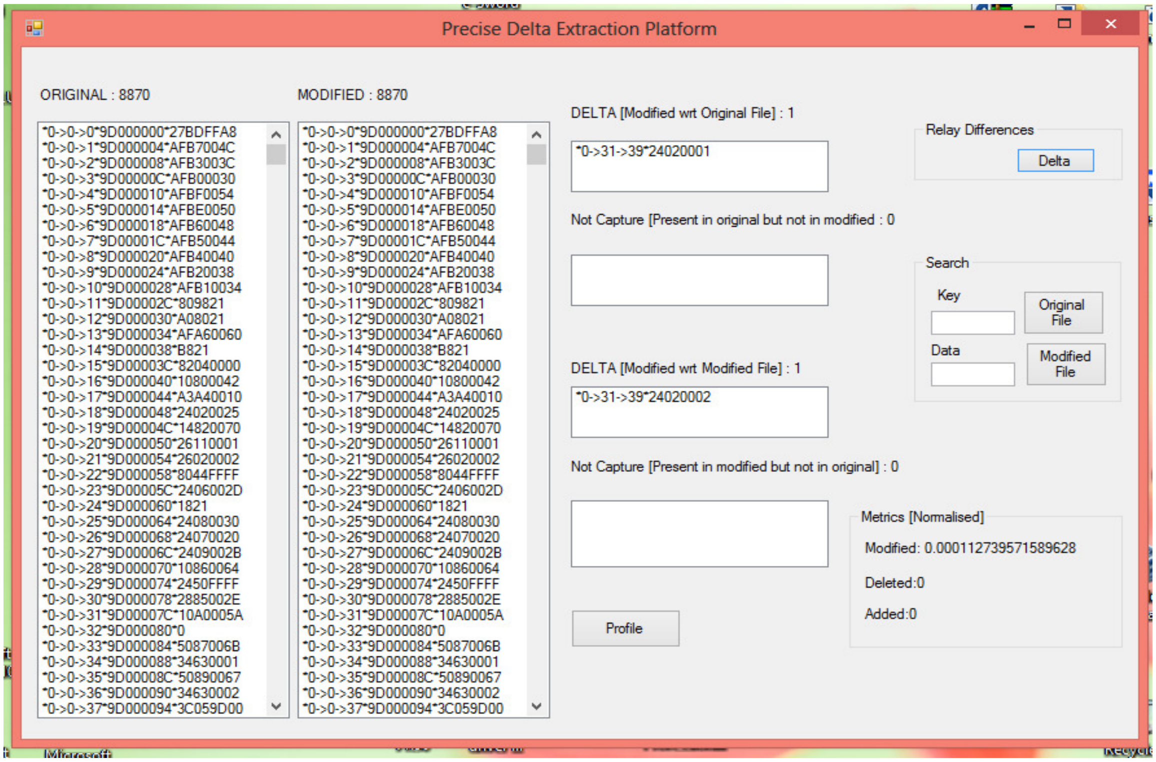
i. A list of all loadable segments contained in the file. The information is obtained using Equation (9).

ii. The virtual and physical start address of each segment.

iii. The total byte size of each segment

iv. Whether 'Execute', 'Read' or 'Write' operations are allowed in the listed segments.

v. It also indicates the unified Addressing scheme obtained to identify uniquely each data content within the ELF file.



ORIGINAL : 8870

```
*0->0->0*9D000000*27BDFFA8
*0->0->1*9D000004*AFB7004C
*0->0->2*9D000008*AFB3003C
*0->0->3*9D00000C*AFB00030
*0->0->4*9D000010*AFBF0054
*0->0->5*9D000014*AFBE0050
*0->0->6*9D000018*AFB60048
*0->0->7*9D00001C*AFB50044
*0->0->8*9D000020*AFB40040
*0->0->9*9D000024*AFB20038
*0->0->10*9D000028*AFB10034
*0->0->11*9D00002C*809821
*0->0->12*9D000030*A08021
*0->0->13*9D000034*AFA60060
*0->0->14*9D000038*B821
*0->0->15*9D00003C*82040000
*0->0->16*9D000040*10800042
*0->0->17*9D000044*A3A40010
*0->0->18*9D000048*24020025
*0->0->19*9D00004C*14820070
*0->0->20*9D000050*26110001
*0->0->21*9D000054*26020002
*0->0->22*9D000058*8044FFFF
*0->0->23*9D00005C*2406002D
*0->0->24*9D000060*1821
*0->0->25*9D000064*24080030
*0->0->26*9D000068*24070020
*0->0->27*9D00006C*2409002B
*0->0->28*9D000070*10860064
*0->0->29*9D000074*2450FFFF
*0->0->30*9D000078*2885002E
*0->0->31*9D00007C*10A0005A
*0->0->32*9D000080*0
*0->0->33*9D000084*5087006B
*0->0->34*9D000088*34630001
*0->0->35*9D00008C*50890067
*0->0->36*9D000090*34630002
*0->0->37*9D000094*3C059D00
```

MODIFIED : 8870

```
*0->0->0*9D000000*27BDFFA8
*0->0->1*9D000004*AFB7004C
*0->0->2*9D000008*AFB3003C
*0->0->3*9D00000C*AFB00030
*0->0->4*9D000010*AFBF0054
*0->0->5*9D000014*AFBE0050
*0->0->6*9D000018*AFB60048
*0->0->7*9D00001C*AFB50044
*0->0->8*9D000020*AFB40040
*0->0->9*9D000024*AFB20038
*0->0->10*9D000028*AFB10034
*0->0->11*9D00002C*809821
*0->0->12*9D000030*A08021
*0->0->13*9D000034*AFA60060
*0->0->14*9D000038*B821
*0->0->15*9D00003C*82040000
*0->0->16*9D000040*10800042
*0->0->17*9D000044*A3A40010
*0->0->18*9D000048*24020025
*0->0->19*9D00004C*14820070
*0->0->20*9D000050*26110001
*0->0->21*9D000054*26020002
*0->0->22*9D000058*8044FFFF
*0->0->23*9D00005C*2406002D
*0->0->24*9D000060*1821
*0->0->25*9D000064*24080030
*0->0->26*9D000068*24070020
*0->0->27*9D00006C*2409002B
*0->0->28*9D000070*10860064
*0->0->29*9D000074*2450FFFF
*0->0->30*9D000078*2885002E
*0->0->31*9D00007C*10A0005A
*0->0->32*9D000080*0
*0->0->33*9D000084*5087006B
*0->0->34*9D000088*34630001
*0->0->35*9D00008C*50890067
*0->0->36*9D000090*34630002
*0->0->37*9D000094*3C059D00
```

DELTA [Modified wrt Original File] : 1

```
*0->31->39*24020001
```

Not Capture [Present in original but not in modified : 0

DELTA [Modified wrt Modified File] : 1

```
*0->31->39*24020002
```

Not Capture [Present in modified but not in original] : 0

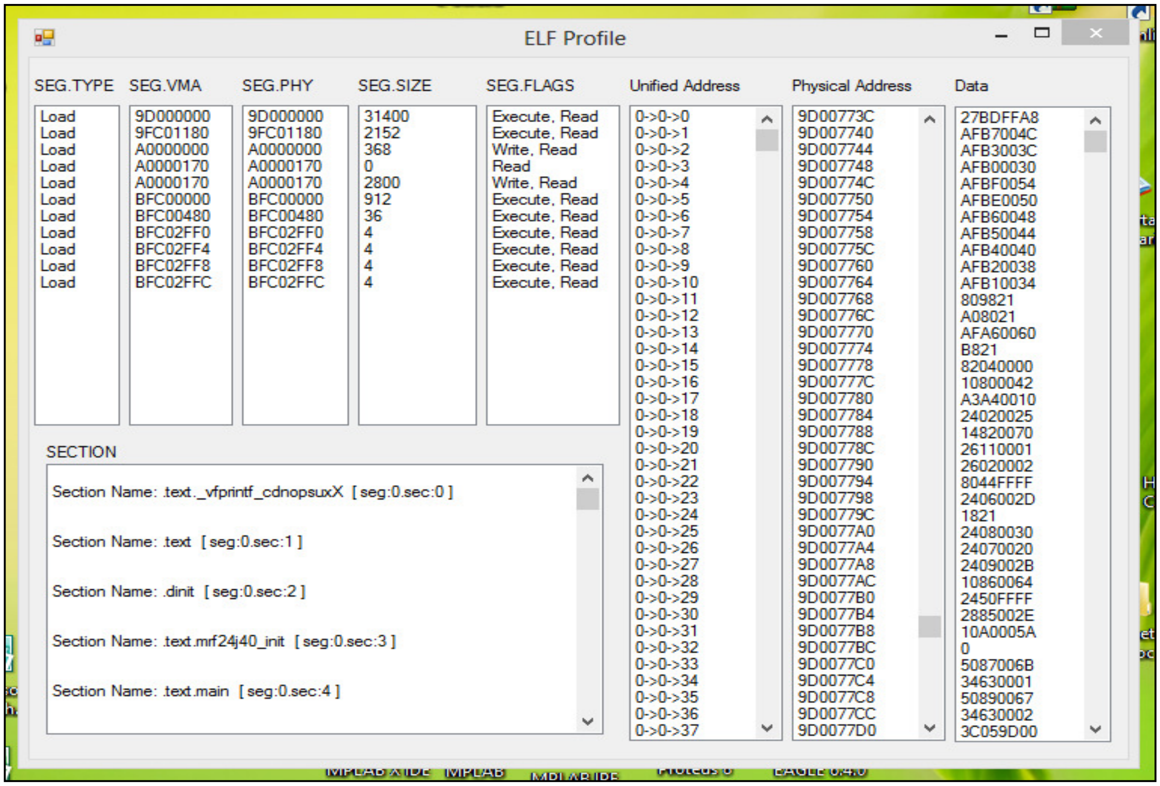
Metrics [Normalised]

Modified: 0.000112739571589628

Deleted:0

Added:0

Figure 1: Delta Extraction Front End



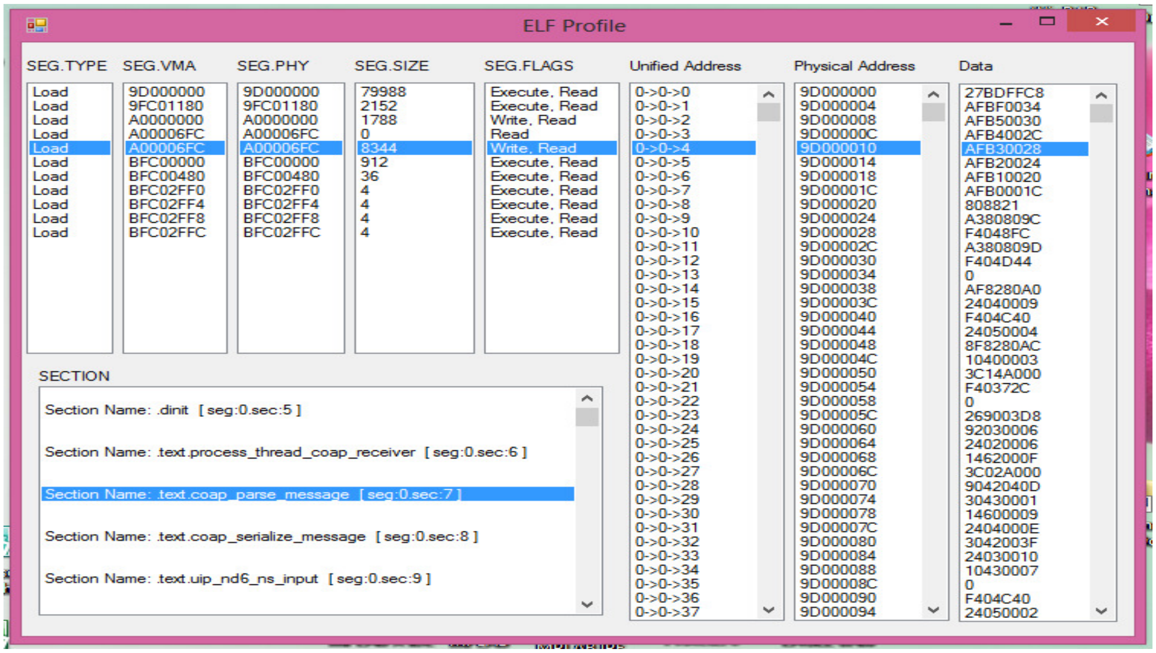
```
SEG.TYPE  SEG.VMA    SEG.PHY    SEG.SIZE  SEG.FLAGS
Load      9D000000   9D000000   31400     Execute, Read
Load      9FC01180   9FC01180   2152      Execute, Read
Load      A0000000   A0000000   368       Write, Read
Load      A0000170   A0000170   0         Read
Load      A0000170   A0000170   2800      Write, Read
Load      BFC00000   BFC00000   912       Execute, Read
Load      BFC00480   BFC00480   36        Execute, Read
Load      BFC02FF0   BFC02FF0   4         Execute, Read
Load      BFC02FF4   BFC02FF4   4         Execute, Read
Load      BFC02FF8   BFC02FF8   4         Execute, Read
Load      BFC02FFC   BFC02FFC   4         Execute, Read
```

SECTION

```
Section Name: .text._vfprintf_cdnopsuxX  [ seg:0.sec:0 ]

Section Name: .text  [ seg:0.sec:1 ]

Section Name: .dinit  [ seg:0.sec:2 ]

Section Name: .text.mrf24j40_init  [ seg:0.sec:3 ]

Section Name: .text.main  [ seg:0.sec:4 ]
```

```
Unified Address   Physical Address   Data
0->0->0           9D00773C           27BDFFA8
0->0->1           9D007740           AFB7004C
0->0->2           9D007744           AFB3003C
0->0->3           9D007748           AFB00030
0->0->4           9D00774C           AFBF0054
0->0->5           9D007750           AFBE0050
0->0->6           9D007754           AFB60048
0->0->7           9D007758           AFB50044
0->0->8           9D00775C           AFB40040
0->0->9           9D007760           AFB20038
0->0->10          9D007764           AFB10034
0->0->11          9D007768           809821
0->0->12          9D00776C           A08021
0->0->13          9D007770           AFA60060
0->0->14          9D007774           B821
0->0->15          9D007778           82040000
0->0->16          9D00777C           10800042
0->0->17          9D007780           A3A40010
0->0->18          9D007784           24020025
0->0->19          9D007788           14820070
0->0->20          9D00778C           26110001
0->0->21          9D007790           26020002
0->0->22          9D007794           8044FFFF
0->0->23          9D007798           2406002D
0->0->24          9D00779C           1821
0->0->25          9D0077A0           24080030
0->0->26          9D0077A4           24070020
0->0->27          9D0077A8           2409002B
0->0->28          9D0077AC           10860064
0->0->29          9D0077B0           2450FFFF
0->0->30          9D0077B4           2885002E
0->0->31          9D0077B8           10A0005A
0->0->32          9D0077BC           0
0->0->33          9D0077C0           5087006B
0->0->34          9D0077C4           34630001
0->0->35          9D0077C8           50890067
0->0->36          9D0077CC           34630002
0->0->37          9D0077D0           3C059D00
```

Figure 2: ELF Profile Display Front End



SEG.TYPE	SEG.VMA	SEG.PHY	SEG.SIZE	SEG.FLAGS
Load	9D000000	9D000000	79988	Execute, Read
Load	9FC01180	9FC01180	2152	Execute, Read
Load	A0000000	A0000000	1788	Write, Read
Load	A00006FC	A00006FC	0	Read
Load	A00006FC	A00006FC	8344	Write, Read
Load	BFC00000	BFC00000	912	Execute, Read
Load	BFC00480	BFC00480	36	Execute, Read
Load	BFC02FF0	BFC02FF0	4	Execute, Read
Load	BFC02FF4	BFC02FF4	4	Execute, Read
Load	BFC02FF8	BFC02FF8	4	Execute, Read
Load	BFC02FFC	BFC02FFC	4	Execute, Read

SECTION

Section Name: .dinit [seg:0.sec:5]

Section Name: .text.process_thread_coap_receiver [seg:0.sec:6]

Section Name: .text.coap_parse_message [seg:0.sec:7]

Section Name: .text.coap_serialize_message [seg:0.sec:8]

Section Name: .text.uip_nd6_ns_input [seg:0.sec:9]

Unified Address	Physical Address	Data
0->0->0	9D000000	27BDFFC8
0->0->1	9D000004	AFBF0034
0->0->2	9D000008	AFB50030
0->0->3	9D00000C	AFB4002C
0->0->4	9D000010	AFB30028
0->0->5	9D000014	AFB20024
0->0->6	9D000018	AFB10020
0->0->7	9D00001C	AFB0001C
0->0->8	9D000020	808821
0->0->9	9D000024	A380809C
0->0->10	9D000028	F4048FC
0->0->11	9D00002C	A380809D
0->0->12	9D000030	F404D44
0->0->13	9D000034	0
0->0->14	9D000038	AF8280A0
0->0->15	9D00003C	24040009
0->0->16	9D000040	F404C40
0->0->17	9D000044	24050004
0->0->18	9D000048	8F8280AC
0->0->19	9D00004C	10400003
0->0->20	9D000050	3C14A000
0->0->21	9D000054	F40372C
0->0->22	9D000058	0
0->0->23	9D00005C	269003D8
0->0->24	9D000060	92030006
0->0->25	9D000064	24020006
0->0->26	9D000068	1462000F
0->0->27	9D00006C	3C02A000
0->0->28	9D000070	9042040D
0->0->29	9D000074	30430001
0->0->30	9D000078	14600009
0->0->31	9D00007C	2404000E
0->0->32	9D000080	3042003F
0->0->33	9D000084	24030010
0->0->34	9D000088	10430007
0->0->35	9D00008C	0
0->0->36	9D000090	F404C40
0->0->37	9D000094	24050002

Figure 3: ELF profile of the 'remotepowerswitch.elf' file

4.2 Case Study 1: Effecting Changes to 'Constant Data '

Program Code Listing 1 and 2 shows the highlighted section of the 'Led.c' source code where the change was made. In this case, the label definition 'LEDS_RED' used in the original source code has a value of '#2' as indicated in the header file 'Led.h' in Program Code Listing 1. The label definition was altered to take on a new value of '#4' represented by 'LEDS_YELLOW'. The two source codes (the original and the altered) were compiled and their generated ELF fed into the PDE. The delta obtained are illustrated in Figures 6, 7 and 8.

Program Code Listing 1: Extract from 'Led.h' showing values assigned to constant definitions used in 'Led.c'

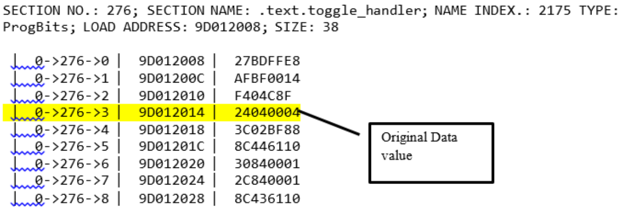
```
#ifndef LEDS_GREEN
#define LEDS_GREEN  1
#endif/* LEDS_GREEN */
#ifndef LEDS_YELLOW
#define LEDS_YELLOW  2
#endif/* LEDS_YELLOW */
#ifndef LEDS_RED
#define LEDS_RED  4
#endif/* LEDS_RED */
#ifndef LEDS_BLUE
#define LEDS_BLUE  LEDS_YELLOW
#endif/* LEDS_BLUE */
```

Program Code Listing 2: Extract from 'Led.C' file showing original Constant Assignment (Case study1)

```
void
toggle_handler(void* request, void* response, uint8_t
*buffer, uint16_t preferred_size, int32_t *offset)
{
 leds_toggle(LEDS_RED);

 PORTEbits.RE0 = !PORTEbits.RE0;
}
```

Program Code Listing 3: Extract from 'Led.C' file showing modified Constant Assignment (Case study 1)

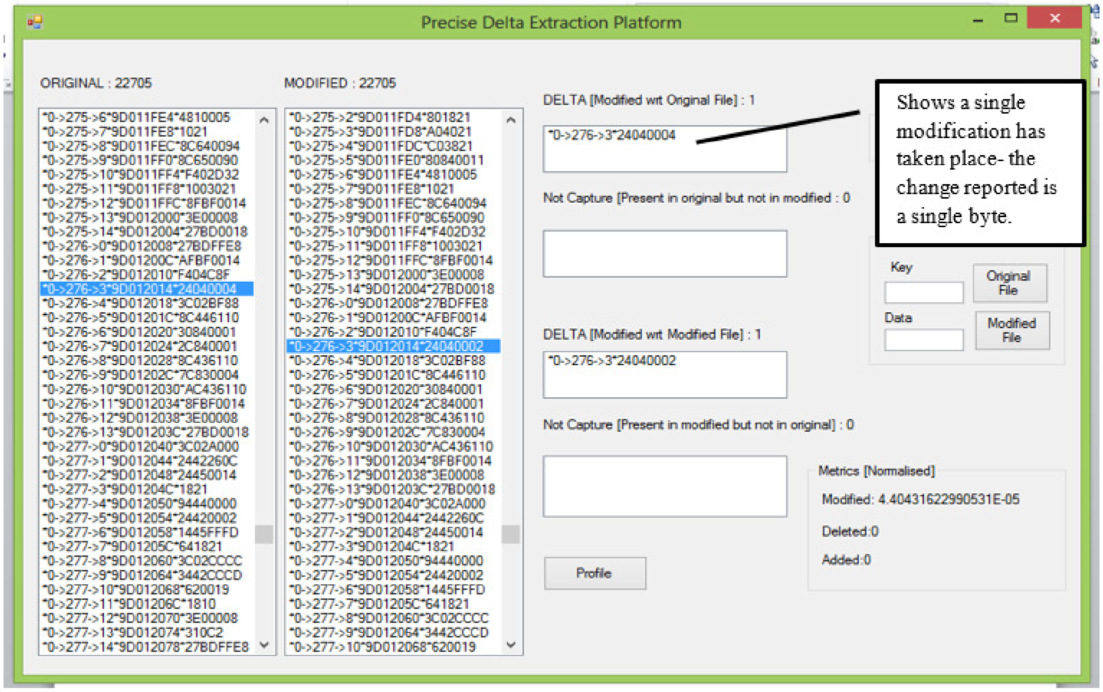
```
void
toggle handler(void* request, void* response, uint8_t
*buffer, uint16_t preferred_ size, int32_t *offset)
{
 leds_toggle (LEDS_YELLOW);
 PORTEbits.RE0 =! PORTEbits.RE0;
}
```



```
SECTION NO.: 276; SECTION NAME: .text.toggle_handler; NAME INDEX.: 2175 TYPE:
ProgBits; LOAD ADDRESS: 9D012008; SIZE: 38

 |__0->276->0 |   9D012008 |   27BDFFE8
 |__0->276->1 |   9D01200C |   AFBF0014
 |__0->276->2 |   9D012010 |   F404C8F
 |__0->276->3 |   9D012014 |   24040004
 |__0->276->4 |   9D012018 |   3C02BF88
 |__0->276->5 |   9D01201C |   8C446110
 |__0->276->6 |   9D012020 |   30840001
 |__0->276->7 |   9D012024 |   2C840001
 |__0->276->8 |   9D012028 |   8C436110
```

Figure 4: Original Data value of the file before effecting changes (Case study1)

The Delta listing in Figure 6 was obtained from the ‘modifiedRpt.txt’ and the initial values as presented in the ‘originalRpt.txt’ file is shown in Program Code Listing 4. Program Code Listing 4 depicts the alteration in the data content to be exactly one byte in size. The change occurs at unified address location ‘0->276->3' and has a physical address value of ‘9D012014’. The extent of change does not affect the size of the entire firmware, and it is confined to just a segment in the program hence its orientation is of the segment confined type.



ORIGINAL : 22705

```
*0->275->6*9D011FE4*4810005
*0->275->7*9D011FE8*1021
*0->275->8*9D011FEC*8C640094
*0->275->9*9D011FF0*8C650090
*0->275->10*9D011FF4*F402D32
*0->275->11*9D011FF8*1003021
*0->275->12*9D011FFC*8FBF0014
*0->275->13*9D012000*3E00008
*0->275->14*9D012004*27BD0018
*0->276->0*9D012008*27BDFFE8
*0->276->1*9D01200C*AFBF0014
*0->276->2*9D012010*F404C8F
*0->276->3*9D012014*24040004
*0->276->4*9D012018*3C02BF88
*0->276->5*9D01201C*8C446110
*0->276->6*9D012020*30840001
*0->276->7*9D012024*2C840001
*0->276->8*9D012028*8C436110
*0->276->9*9D01202C*7C830004
*0->276->10*9D012030*AC436110
*0->276->11*9D012034*8FBF0014
*0->276->12*9D012038*3E00008
*0->276->13*9D01203C*27BD0018
*0->277->0*9D012040*3C02A000
*0->277->1*9D012044*2442260C
*0->277->2*9D012048*24450014
*0->277->3*9D01204C*1821
*0->277->4*9D012050*94440000
*0->277->5*9D012054*24420002
*0->277->6*9D012058*1445FFFD
*0->277->7*9D01205C*641821
*0->277->8*9D012060*3C02CCCC
*0->277->9*9D012064*3442CCCD
*0->277->10*9D012068*620019
*0->277->11*9D01206C*1810
*0->277->12*9D012070*3E00008
*0->277->13*9D012074*310C2
*0->277->14*9D012078*27BDFFE8
```

MODIFIED : 22705

```
*0->275->2*9D011FD4*801821
*0->275->3*9D011FD8*A04021
*0->275->4*9D011FDC*C03821
*0->275->5*9D011FE0*80840011
*0->275->6*9D011FE4*4810005
*0->275->7*9D011FE8*1021
*0->275->8*9D011FEC*8C640094
*0->275->9*9D011FF0*8C650090
*0->275->10*9D011FF4*F402D32
*0->275->11*9D011FF8*1003021
*0->275->12*9D011FFC*8FBF0014
*0->275->13*9D012000*3E00008
*0->275->14*9D012004*27BD0018
*0->276->0*9D012008*27BDFFE8
*0->276->1*9D01200C*AFBF0014
*0->276->2*9D012010*F404C8F
*0->276->3*9D012014*24040002
*0->276->4*9D012018*3C02BF88
*0->276->5*9D01201C*8C446110
*0->276->6*9D012020*30840001
*0->276->7*9D012024*2C840001
*0->276->8*9D012028*8C436110
*0->276->9*9D01202C*7C830004
*0->276->10*9D012030*AC436110
*0->276->11*9D012034*8FBF0014
*0->276->12*9D012038*3E00008
*0->276->13*9D01203C*27BD0018
*0->277->0*9D012040*3C02A000
*0->277->1*9D012044*2442260C
*0->277->2*9D012048*24450014
*0->277->3*9D01204C*1821
*0->277->4*9D012050*94440000
*0->277->5*9D012054*24420002
*0->277->6*9D012058*1445FFFD
*0->277->7*9D01205C*641821
*0->277->8*9D012060*3C02CCCC
*0->277->9*9D012064*3442CCCD
*0->277->10*9D012068*620019
```

DELTA [Modified wrt Original File] : 1

```
*0->276->3*24040004
```

Not Capture [Present in original but not in modified : 0

DELTA [Modified wrt Modified File] : 1

```
*0->276->3*24040002
```

Not Capture [Present in modified but not in original] : 0

Metrics [Normalised]

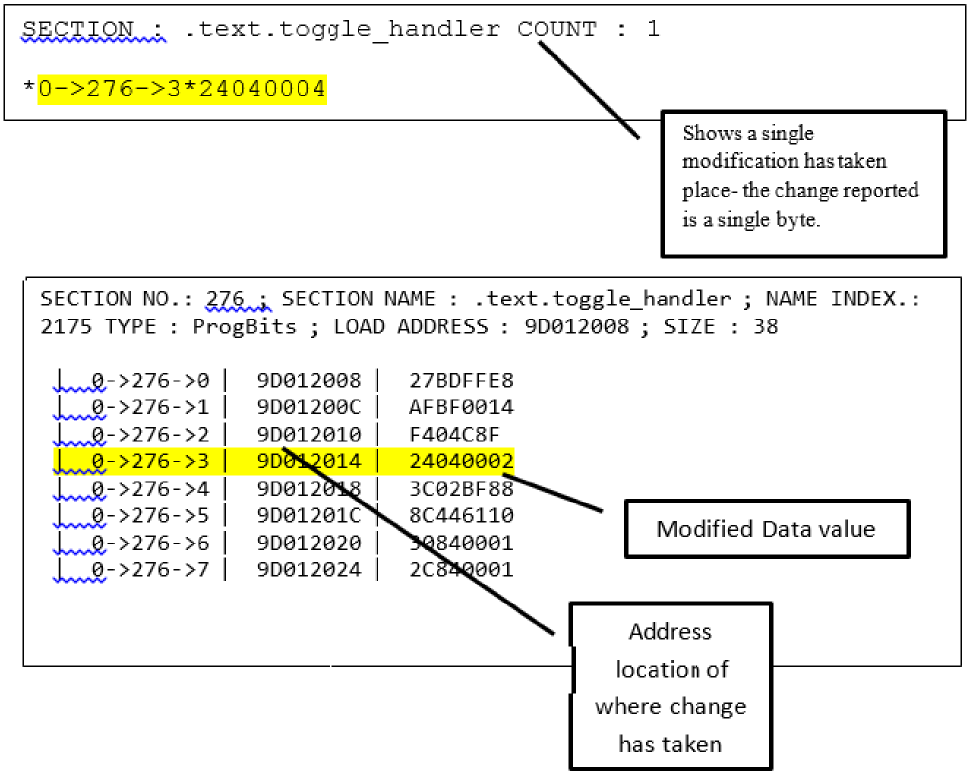
Modified: 4.40431622990531E-05

Deleted:0

Added:0

Shows a single modification has taken place- the change reported is a single byte.

Figure 5: PDE display delta results obtained from Case study 1



```
SECTION : .text.toggle_handler COUNT : 1

*0->276->3*24040004
```

Shows a single modification has taken place- the change reported is a single byte.

```
SECTION NO.: 276 ; SECTION NAME : .text.toggle_handler ; NAME INDEX.:
2175 TYPE : ProgBits ; LOAD ADDRESS : 9D012008 ; SIZE : 38

 |  0->276->0 |   9D012008 |   27BDFFE8
 |  0->276->1 |   9D01200C |   AFBF0014
 |  0->276->2 |   9D012010 |   F404C8F
 |  0->276->3 |   9D012014 |   24040002
 |  0->276->4 |   9D012018 |   3C02BF88
 |  0->276->5 |   9D01201C |   8C446110
 |  0->276->6 |   9D012020 |   30840001
 |  0->276->7 |   9D012024 |   2C840001
```

Modified Data value

Address location of where change has taken

Figure 6: Modified value of Data after effecting changes (Case study 1)

Program Code Listing 4: Extract from 'Led.C' file showing the insertion of a 'Flow of Control' code construct (Case study 2)

```
toggle handler(void* request, void* response,
uint8_t *buffer, uint16_t preferred_size,
int32_t *offset)
{
int decide = 0;
if (decide = 1)
 {
       leds_toggle(LEDS_RED);
             PORTEbits.RE0 = !PORTEbits.RE0;
       }
else
       {
       leds_toggle(LEDS_YELLOW);
             PORTEbits.RE0 = !PORTEbits.RE0;
             }
}
```

```
SECTION NO.: 150 ; SECTION NAME :
.text.process_thread_remote_power_switch ;
NAME INDEX.: 142D TYPE : ProgBits ; LOAD
ADDRESS : 9D00F268 ; SIZE : 84
 |  0->150->0 |  9D00F268 |  27BDFFE8
 |  0->150->1 |  9D00F26C |  AFBF0014
 |  0->150->2 |  9D00F270 |  AFB00010
 |  0->150->3 |  9D00F274 |  94820000
 |  0->150->4 |  9D00F278 |  10400006
 |  0->150->5 |  9D00F27C |  808021
|  0->150->6 |  9D00F280 |  24030059
 |  0->150->7 |  9D00F284 |  54430014
 |  0->150->8 |  9D00F288 |  A4800000
 |  0->150->9 |  9D00F28C |  B403CB3
 |  0->150->10 |  9D00F290 |  24020059
 |  0->150->11 |  9D00F294 |  F4043E3
 |  0->150->12 |  9D00F298 |  0
 |  0->150->13 |  9D00F29C |  3C02BF88
 |  0->150->14 |  9D00F2A0 |  8C436100
 |  0->150->15 |  9D00F2A4 |  7C030004
 |  0->150->16 |  9D00F2A8 |  AC436100
 |  0->150->17 |  9D00F2AC |  3C02BF88
 |  0->150->18 |  9D00F2B0 |  8C436110
 |  0->150->19 |  9D00F2B4 |  7C030004
 |  0->150->20 |  9D00F2B8 |  AC436110
 |  0->150->21 |  9D00F2BC |  3C04A000
 |  0->150->22 |  9D00F2C0 |  F4042CF
 |  0->150->23 |  9D00F2C4 |  2484257C
|  0->150->24 |  9D00F2C8 |  24020059
 |  0->150->25 |  9D00F2CC |  A6020000
 |  0->150->26 |  9D00F2D0 |  B403CB7
 |  0->150->27 |  9D00F2D4 |  24020001
 |  0->150->28 |  9D00F2D8 |  24020003
 |  0->150->29 |  9D00F2DC |  8FBF0014
 |  0->150->30 |  9D00F2E0 |  8FB00010
 |  0->150->31 |  9D00F2E4 |  3E00008
 |  0->150->32 |  9D00F2E8 |  27BD0018
```

Figure 7: PDE display delta results obtained from Case study 2

4.3 Case Study 2: Effecting Changes to 'Flow of Control'

Similar procedures carried out in the previous sub-section were repeated for a scenario where 'flow of control' construct is introduced in the main application's source codes. Program Code Listing 4 shows highlights of the introduced 'flow of control' construct. The isolated delta obtained were presented in Figure 7. In addition, Figure 7 depicts the delta distribution in the modified file.

4.4 Case Study 3: Effecting Changes to 'Function's Name'

In this case, changes were made to the original code by altering some selected function names in the application's source code. The deltas obtained were quite large and were unevenly distributed in the program memory map. These changes as reported by the PDE are depicted in Figure 8.

4.5 Discussion of Results

A summary of the results obtained in the three case studies earlier presented above is shown in Table 4. The results were categorised under the following: the delta(s) size, the physical address range of the delta(s), related ELF segments where the delta resides, delta orientation and the number of segment(s) involved.

The PDE isolates delta codes and provides information on the location in memory where appropriate changes are to be made in the new firmware. The information illustrated in Table 4. is useful in determining the size of delta involved and nature or characteristic of their distribution in the program memory.

In case study one, it is observed that the size of the delta is a single byte, this very small change can mean a lot in real WSN applications. One typical example involves altering the rate at which a sensor samples data in the field or taking an average of the number of samples acquired.

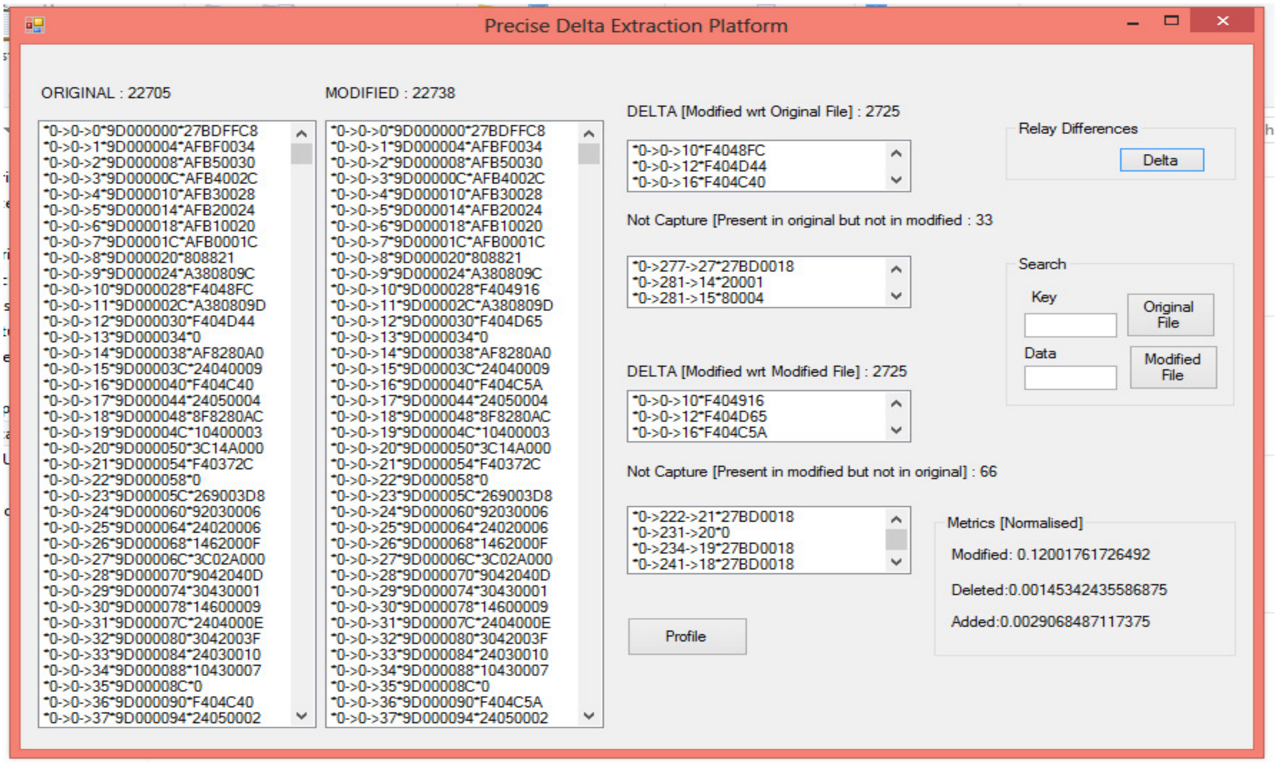
These changes in most cases are limited to single byte size or integer size. Using the conventional approach will involve the erasure and rewriting of the entire program memory space or a substantial amount of the memory space if a loadable reprogramming approach is employed. In case study three, the delta distribution among segments in the flash memory is highly fragmented. These changes spread over five ELF segments, namely: (.text, .vector, .data, .reset, and .config_BFC02FF0). Even though the total number of bytes involved is relatively small (2725) compared to the actual memory size (128KB) of the PIC32MX320F128H microcontroller, the disjointed nature of the delta is best handled by reprogramming the entire available memory space. The observations inferred from the above case studies can be instrumental in devising an inference engine for the fuzzy logic subsystem employed in the context-based reconfiguration system for WSN.

The limitations attributed to the *difference-based* approach as highlighted in Section 2 were resolved using the precision delta extraction scheme. The precision delta extraction scheme generate a unified address scheme, which concatenates the segment number, section number and the position of each data contained in the original image and the modified image file separately. The unified address scheme gives each set of data contained in the two files unique reference numbers that are similar. Hence, when any of the set of data is missing, its corresponding unified address ceases to exist, though its physical address might still exist, it will definitely point to another data. Similarly, when a set of new data is added, these new data acquire new unified addresses and invariably become easier to isolate.

This approach rules out the need to generate the pair (Checksum, MD5 hash) for each block of the old image and new image for comparison, which subsequently reduces the cost of implementing expensive computations in the base station. Though Panta, Bagchi and Midkiff in [19] tried to justify the use of the host computer in implementing their modified algorithm, issues of degrading performance occasioned by delay in Delta dissemination can arise (especially in real-time applications). Other variants of the Rysnc algorithm have been proposed and implemented: RDIFF [26], VCDIFF [20] and BSDIFF [27]. However, since they are derivatives of the original Rsync algorithm, the lapses highlighted here are very much applicable.

5. CONCLUSIONS

The PDE Metric tool developed is an improvement over existing similar tools like the Rysnc and its variants. The PDE does not need tuning in order to reduce the overheads associated with Rysnc and its variants. The PDE provides concise physical address and the virtual address of deltas. This information is useful for targeting Delta locations and allowing reconfiguration procedures to be confined to a single segment of the Flash memory thereby saving an enormous amount of energy expended when an entire program memory is reprogrammed.



ORIGINAL : 22705

```
*0->0->0*9D000000*27BDFFC8
*0->0->1*9D000004*AFBF0034
*0->0->2*9D000008*AFB50030
*0->0->3*9D00000C*AFB4002C
*0->0->4*9D000010*AFB30028
*0->0->5*9D000014*AFB20024
*0->0->6*9D000018*AFB10020
*0->0->7*9D00001C*AFB0001C
*0->0->8*9D000020*808821
*0->0->9*9D000024*A380809C
*0->0->10*9D000028*F4048FC
*0->0->11*9D00002C*A380809D
*0->0->12*9D000030*F404D44
*0->0->13*9D000034*0
*0->0->14*9D000038*AF8280A0
*0->0->15*9D00003C*24040009
*0->0->16*9D000040*F404C40
*0->0->17*9D000044*24050004
*0->0->18*9D000048*8F8280AC
*0->0->19*9D00004C*10400003
*0->0->20*9D000050*3C14A000
*0->0->21*9D000054*F40372C
*0->0->22*9D000058*0
*0->0->23*9D00005C*269003D8
*0->0->24*9D000060*92030006
*0->0->25*9D000064*24020006
*0->0->26*9D000068*1462000F
*0->0->27*9D00006C*3C02A000
*0->0->28*9D000070*9042040D
*0->0->29*9D000074*30430001
*0->0->30*9D000078*14600009
*0->0->31*9D00007C*2404000E
*0->0->32*9D000080*3042003F
*0->0->33*9D000084*24030010
*0->0->34*9D000088*10430007
*0->0->35*9D00008C*0
*0->0->36*9D000090*F404C40
*0->0->37*9D000094*24050002
```

MODIFIED : 22738

```
*0->0->0*9D000000*27BDFFC8
*0->0->1*9D000004*AFBF0034
*0->0->2*9D000008*AFB50030
*0->0->3*9D00000C*AFB4002C
*0->0->4*9D000010*AFB30028
*0->0->5*9D000014*AFB20024
*0->0->6*9D000018*AFB10020
*0->0->7*9D00001C*AFB0001C
*0->0->8*9D000020*808821
*0->0->9*9D000024*A380809C
*0->0->10*9D000028*F404916
*0->0->11*9D00002C*A380809D
*0->0->12*9D000030*F404D65
*0->0->13*9D000034*0
*0->0->14*9D000038*AF8280A0
*0->0->15*9D00003C*24040009
*0->0->16*9D000040*F404C5A
*0->0->17*9D000044*24050004
*0->0->18*9D000048*8F8280AC
*0->0->19*9D00004C*10400003
*0->0->20*9D000050*3C14A000
*0->0->21*9D000054*F40372C
*0->0->22*9D000058*0
*0->0->23*9D00005C*269003D8
*0->0->24*9D000060*92030006
*0->0->25*9D000064*24020006
*0->0->26*9D000068*1462000F
*0->0->27*9D00006C*3C02A000
*0->0->28*9D000070*9042040D
*0->0->29*9D000074*30430001
*0->0->30*9D000078*14600009
*0->0->31*9D00007C*2404000E
*0->0->32*9D000080*3042003F
*0->0->33*9D000084*24030010
*0->0->34*9D000088*10430007
*0->0->35*9D00008C*0
*0->0->36*9D000090*F404C5A
*0->0->37*9D000094*24050002
```

DELTA [Modified wrt Original File] : 2725

```
*0->0->10*F4048FC
*0->0->12*F404D44
*0->0->16*F404C40
```

Not Capture [Present in original but not in modified : 33

```
*0->277->27*27BD0018
*0->281->14*20001
*0->281->15*80004
```

DELTA [Modified wrt Modified File] : 2725

```
*0->0->10*F404916
*0->0->12*F404D65
*0->0->16*F404C5A
```

Not Capture [Present in modified but not in original] : 66

```
*0->222->21*27BD0018
*0->231->20*0
*0->234->19*27BD0018
*0->241->18*27BD0018
```

Key:

Data:

Metrics [Normalised]

Modified: 0.12001761726492

Deleted:0.00145342435586875

Added:0.0029068487117375

Figure 8: PDE display delta results obtained from Case study 3

Table 4: A summary of results obtained for the three case studies

Case study	Title	Size of changes in byte	Physical Address Range(s)		ELF segment Name	Orientation of Change in Memory(Delta Orientation)	Number of Segment
			Start	End			
1	Effecting Changes to 'Constant' Data	2	9D012014	9D012014	.text	Segment confined	1
2	Effecting Changes to 'Flow of Control	3	9D00F280	9D00F2C8	.text	Segment confined	1
3	Effecting Changes to 'Function's Name'	2725	9D000028 9FC01280 A00025FC BFC00014 BFC02FF0	9D013884 9FC01984 A0002784 BFC00194 BFC02FF0	.text .vector .data .reset .config_BFC02FF0	Segments Disjointed	5

6. REFERENCES

[1] J. Yick, B. Mukherjee and D. Ghosal, "Wireless Sensor Network Survey," *Journal of Computer Networks,* vol. 52, no. 2008, pp. 2292-2330, 2008.

[2] J. Burrell, T. Brooke and R. Beckwith, "Vineyard Computing: Sensor Networks in Agricultural Computing," *IEEE Pervasive Computing,* pp. 38-45, 2004.

[3] P. R. Krishna, S. Bagchi and P. S. Midkiff, "Zephyr: Efficient Increamental Reprogramming of Sensor Nodes using Function call Indirections and Difference Computation," in *Proceedings of the Annual Technical Conference (USENIX)*, San Diego, CA,USA, , June, 2009.

[4] C. Han, R. Kumar, R. Shea, E. Kohler and M. Srivastava, "A dynamic operating system for sensor nodes," in *Proceedings of the 3rd International Conference on Mobile systems, Applications , and Services (MobiSys)*, 2005.

[5] S. Misra and E. M. Eronu, "Implementing Reconfigurable Wireless Sensor Networks: The Embedded Operating System Approach, Embedded Systems," in *High Performance Systems, Applications and Projects*, 2012, pp. 221-232.

[6] S. Jun-Zhao, "OS-based Reprogramming Techniques in Wireless Sensor Networks: A Survey," in *Ubi-media Computing (U-Media), 3rd IEEE International Conference*, July, 2010.

[7] A. Dunkels, N. Finne, J. Eriksson and T. Voigt, "Run-time Dynamic linking for Reprogramming Wireless Sensor Networks," in *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, New York, USA, 2006.

[8] J. Hill, "TinyOS: An Operating System for Sensor Networks," in *Ambient Intelligence, Springer-Velag Berlin Heidelberg*, Netherland, 2005.

[9] J. Jeong, S. Kim and A. Broad , "Network reprogramming, tinyos documentation," [Online]. Available: http://www.tinyos.net/tinyos-1-x/doc/xnp.pdf. [Accessed 10 01 2012].

[10] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Cullar and K. Pister, "System architecture directions for networked sensors," in *Proceedings of the 9th International Confernce on Architectural Support for Programming Languages and Operating Systems*, November, 2004.

[11] Q. Cao, T. Abdelzaher, J. Stankovic and T. He, "The LiteOS Operating System: Towards Unix Like Abstraction for Wireless Sensor Networks," in *Proceedings of the 7th International Conference on Information Processing in Sensor Networks (IPSN 2008),*, St. Louis, MO, USA, 2008.

[12] C. Han, R. Kumar, R. Shea, E. Kolher and M. Srivastava, "A Dynamic Operating System for Sensor Nodes," in *Proceedings of the 3rd International Conference on Mobile Systems Applications and Services, ACM*, New York.

[13] K. Kulkarni, S. Sanyal, H. Al-Qaheri and S. Sanyal, "Dynamic Reconfiguration of Wireless Sensor Networks," *International Journal of Computer Science and Applications,* vol. 6, no. 4, pp. 16-42, 2009.

[14] E. M. Eronu, S. Misra and M. Aibinu, "Reconfiguration Approaches in Wirless Sensor Network: Issues and Chanllenges," in *2nd IEE Conference on Emerging &Sustanable Technologies for Power & ICT in a Developing Society (NIGERCON)*, Owerri, Nigeria, 2013.

[15] A. Tridgeell and P. Mackerras, "The Rsync Algorithm," The Australian National University., Canberra, Australia, 1996.

[16] S. Kim, J. Lee, K. Hur, K. Hwang and D. Eom, "Tiny Module–Linking for Energy–Efficient Reprogramming in wireless sensor networks," *Transactions on Consumer Electronics,* vol. 55, no. 4, pp. 1914-1920., 2009.

[17] F. V. Rysselberghe and S. Demeyer, "Evaluating Clone Detection Techniques," in *Proceedings of the International Workshop on Evolution of Large Scale Industrial Applications, ELISA*, 2003.

[18] N. B. Shafi, "Efficient Over-the-air Remote Reprogramming of Wireless Sensor Networks," Ontario, Canada, 2011.

[19] R. K. Panta, S. Bagchi and S. P. Midkiff, "Efficient Incremental Code Update for Sensor Networks, ACM," *Transactions on Sensor Networks,* vol. 7, no. 4, pp. 30.1-30.32., 2011.

[20] D. Korn, J. MacDonald, J. Mogul and K. Vo, "The VCDIFF Generic Differencing and Compression Data Format. RFC 3284 (Proposed Standard)," 2002.

[21] J. Sharp , Microsoft Visual C# 2013, Sebastopol,, California 95472: O'Reilly Media, Inc., 2013.

[22] P. Paolo and R. Marco, Programming Microsoft LINQ in Microsoft.Net 4 Framework, Sebastopol, California 95472: O'Reilly Media, Inc., 2010.

[23] A. Freeman and J. C. Rattz, Pro LINQ: Language Integrated Query in C# 2010, B. Ewan, Ed., New York: Apress , 2010.

[24] TIS Committee, "Tool Interface Standard (TIS) Executable and Linking (ELF) Specification," May,1995.

[25] P. Giovanni, "Remote Power Switch Example for the Seed-Eye Board," 24 01 2013. [Online]. Available: https://github.com/contiki-os/contiki/blob/master/examples/seedeye/powerswitch/remotepowerswitch.c. [Accessed 01 10 2014].

[26] S. Milosh, P. J. Cuijipers and J. J. Lukkien, "Efficient reprogramming of wireless sensor networks using incremental updates and data compression," in *International conference of Pervasive Computing and Communications Workshops (PERCOM Workshops),* 2013.

[27] C. Percival, "Naive differences of executable code. Technical Report," Oxford Computing Laboratory, University of Oxford., UK, 2003.