R3: Optimizing Relocatable Code for Efficient Reprogramming in Networked Embedded Systems

Wei Dong
Biyuan Mo
Chao Huang
Yunhao Liu
Chun Chen
The title: R3: Optimizing Relocatable Code for Efficient Reprogramming in Networked Embedded Systems

- Reprogramming: change/upgrade the software after deployment

- Relocatable code: kind of binary code format having special structure
Introduction

- System software for networked embedded systems often needs to be updated
  - We regularly face requirements of update in GreenOrbs during Dec. 2010~April, 2011.
- The current update approaches are under-satisfactory
  - Serial programming needs collecting all nodes back
  - Deluge requires transferring a large binary
The basic idea of incremental reprogramming

- Traditional reprogramming vs incremental reprogramming
Two factors impacting the delta size

- Differencing algorithm
  - Given the old code and new code, generate a small delta
  - E.g. Rsync [SECON’04], RMTD [MASS’09]

- Problems: (1) do not ensure optimal result, or (2) incur large computation overhead
Two factors impacting the delta size

- Similarity preserving method
  - Given the old code, generate a new code with large similarity
  - E.g. UCC [PLDI’07], Zephyr [USENIX’09], Hermes [INFOCOM’09], R2 [INFOCOM’11]

- Problems: (1) similarity is limited, or (2) large metadata overhead
Contributions

- R3: a holistic reprogram system
  - An optimized relocatable code format for preserving code similarity with a small metadata overhead
  - An optimal and efficient differencing algorithm (optimality is proved)
  - Evaluate the performance of existing methods including Zephyr/Hermes, R2 and R3.
R3 overview

PC side

- R3sim
  - Source files
- R3diff
  - Old binary file
- Dissemination
- R3cons
  - Old binary file
- Loader

Node side

- New binary file

Similarity preserving: basic idea

- **Observation**: If data or function address changes, all references to these addresses would also change.
- **Key idea**: how to keep these reference instructions unchanged
- **Approaches**
  - Slop region [EWSN’05]
    Problem: does not handle the case when function increases beyond the slop region
  - Indirection table, e.g. Zephyr [USENIX’09], Hermes [INFOCOM’09]
    Problems: (1) does not handle instruction other than call; (2) does not handle data shifts
Similarity preserving: slop region

Problems: (1) does not handle the case when function increases beyond the slop region (2) inefficient use of program flash
Similarity preserving: indirection table

- Problems: (1) does not handle instruction other than call; (2) does not handle data shifts
Similarity preserving based on relocatable code

- Relocatable code

<table>
<thead>
<tr>
<th>Raw code</th>
<th>Relocatable code</th>
<th>Relocation entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>offset₁: call 5000; &lt;led0On&gt;</td>
<td>offset₁: call 0000; &lt;led0On&gt;</td>
<td>rela₁ (offset₁, 5000)</td>
</tr>
<tr>
<td>...</td>
<td>offset₂: call 0000; &lt;led0On&gt;</td>
<td>rela₂ (offset₂, 5000)</td>
</tr>
<tr>
<td></td>
<td>offset₃: call 0000; &lt;led0On&gt;</td>
<td>rela₃ (offset₃, 5000)</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- The main benefits of using relocatable code in embedded system is to generate code modules where some functions are external and their addresses are unknown when compiling the current module.
R2: makes all references the same

Problems: incur a large metadata (i.e. relocation entries) overhead
R3: make references to a unique symbol the same

- Key idea of R3: optimize the representation of a relocation entry (offset, addr)
  - Offset is omitted. Instead, a bitmap is used to indicate which memory location needs relocation.
  - The address fields in reference instructions are inflated with symbol indices => # of entries ~ # unique symbols, instead of # of reference instructions.
Similarity preserving: Comparison of R2 and R3

- Eliminate overhead of offset by introducing a small bitmap overhead (C/16)
- Reduce # of entries to # of unique symbols
Benefits of R3sim

- R3 unifies the approach for handling function shifts and data shifts. R3 preserves a large code similarity as R2.
- R3 does not cause memory segmentation and makes efficient use of memory.
- R3 effectively reduces R2’s metadata overhead.
Differencing algorithm

- Delta file format
  - Copy command (cost = $\beta = 5$)
    \[ \text{COPY}_1 \ <n>_2 \ <\text{old_addr}>_2 \]
  - Add command (cost = $\alpha+n = 3+n$)
    \[ \text{ADD}_1 \ <n>_2 \ <\text{BYTE1 ... BYTEn}>_n \]

- Differencing algorithm
  - Problem: Determine which bytes to be copied or added in order to minimize the delta size
R3diff algorithm

R3diff () {
    Generatefootprint();
    Opt[0] = OptA[0] = OptC[0] = 0;
    for (i=1; i<=nsize; i++) {
        k = findk(i-1);
        OptA[i] = min(OptA[i-1]+1, OptC[i-1]+α+1);
        if (k>i-1) OptC[i] = LARGE_INTEGER;
        else OptC[i] = Opt[k]+β;
        Opt[i] = min(OptA[i], OptC[i]);
    }
}
Theorem 1. The R3diff algorithm generates the smallest delta size under a given encoding cost of $\alpha$ and $\beta$.

The worst-case time complexity of the algorithm is $O(n^3)$, and the average time complexity of the algorithm is $O(n^2)$. The space complexity is $O(n)$. 
Dissemination

- Dissemination protocol
  - Similar to Deluge
- Multiple image switching
  - Deluge as a standalone app installed on external flash.
  - Switch to reprogramming state when needed
- Benefits: reduce the app image size
Reconstruction and loading: R3cons and R3boot

- The delta interpreter in the reprogramming image reconstructs the new code according to the old code and delta
- The loader performs relocating before loading the code onto the program flash
Analysis: an example

```
    | COPY  |       |
---|-------|-------|
 Op 0x5000 | ADD   | Op 0x6000 |
    | COPY  |       |
 Op 0x5100 | ADD   | Op 0x6100 |
    |       |       |
 Op 0x5c00 | ADD   | Op 0x6c00 |
    | COPY  |       |
```
Notations

- $C$ denotes the code size in bytes
- $A$ denotes the total number of instructions
- $n$ denotes the number of reference instructions
- $m$ denotes the number of unique symbols
- $k$ denotes the number of call instructions
## Summary of Analysis

<table>
<thead>
<tr>
<th>Approach</th>
<th>Delta size</th>
<th>Approx value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native inc</td>
<td>$10n+5$</td>
<td>$0.864C+5$</td>
</tr>
<tr>
<td>Zephyr</td>
<td>$10(n-k)+5+M[2m]$</td>
<td>$0.54C+5$</td>
</tr>
<tr>
<td>R2</td>
<td>$5+M[4n]$</td>
<td>$5+0.17C$</td>
</tr>
<tr>
<td>R3</td>
<td>$5+M[2m+C/16]$</td>
<td>$5+0.054C$</td>
</tr>
</tbody>
</table>
Implementation

- Based on TinyOS 2.1.1
- R3sim: Perl and C
- R3diff: C
- Dissemination protocol: nesC
- R3cons and R3boot: nesC
- Google code: http://code.google.com/p/r3-dongw
Evaluation

- Change cases
  - Three from TinyOS distribution
  - Three from GreenOrbs
- Evaluation
  - R3diff
  - R3sim
  - Dissemination
  - R3cons and R3boot
- Energy model
## Differencing algorithm: execution time

<table>
<thead>
<tr>
<th>Change case</th>
<th>RMTD-core</th>
<th>RMTD-CS</th>
<th>R3diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.01</td>
<td>0.2</td>
<td>0.01</td>
</tr>
<tr>
<td>2</td>
<td>0.01</td>
<td>0.19</td>
<td>0.01</td>
</tr>
<tr>
<td>3</td>
<td>0.33</td>
<td>6.04</td>
<td>0.04</td>
</tr>
<tr>
<td>4</td>
<td>5.61</td>
<td>55.47</td>
<td>0.64</td>
</tr>
<tr>
<td>5</td>
<td>5.27</td>
<td>54.46</td>
<td>0.32</td>
</tr>
<tr>
<td>6</td>
<td>5.86</td>
<td>55.66</td>
<td>0.29</td>
</tr>
</tbody>
</table>

Execution time (s) of RMTD and R3diff
### Differencing algorithm: Memory consumption

<table>
<thead>
<tr>
<th>Change case</th>
<th>RMTD-core</th>
<th>RMTD-Table</th>
<th>R3diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>19,198</td>
<td>916,318</td>
<td>54,084</td>
</tr>
<tr>
<td>2</td>
<td>19,402</td>
<td>913,614</td>
<td>53,988</td>
</tr>
<tr>
<td>3</td>
<td>168,794</td>
<td>26,149,500</td>
<td>305,116</td>
</tr>
<tr>
<td>4</td>
<td>168,794</td>
<td>26,149,500</td>
<td>305,116</td>
</tr>
<tr>
<td>5</td>
<td>749,626</td>
<td>237,517,190</td>
<td>877,092</td>
</tr>
<tr>
<td>6</td>
<td>742,434</td>
<td>227,116,532</td>
<td>852,556</td>
</tr>
</tbody>
</table>

Memory consumption (bytes) of RMTD and R3diff
Delta size: with and without compression
Dissemination time
R3cons and R3boot overhead

<table>
<thead>
<tr>
<th>Case</th>
<th>R3cons</th>
<th>R3cons-opt</th>
<th>R3boot</th>
<th>Tosboot</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.405</td>
<td>0.395</td>
<td>0.2</td>
<td>0.148</td>
</tr>
<tr>
<td>2</td>
<td>0.48</td>
<td>0.406</td>
<td>0.194</td>
<td>0.148</td>
</tr>
<tr>
<td>3</td>
<td>4.14</td>
<td>2.8</td>
<td>1.08</td>
<td>0.744</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>7.5</td>
<td>2.59</td>
<td>1.72</td>
</tr>
<tr>
<td>5</td>
<td>7.47</td>
<td>6.19</td>
<td>2.62</td>
<td>1.75</td>
</tr>
<tr>
<td>6</td>
<td>8.6</td>
<td>6.54</td>
<td>2.66</td>
<td>1.8</td>
</tr>
</tbody>
</table>

Execution times (s) of R3cons and R3boot
Conclusion

- R3: a holistic reprogram system
- An optimized relocatable code format for preserving code similarity with a small metadata overhead
- An optimal and efficient byte-level differencing algorithm (optimality is proved)
- As a future work, we would like to port R3 to other embedded platforms and OSes
Q & A

Thanks