Lessons learned from “RLC FEC Scheme for FECFRAME” specification at TSVWG: PRNG
Park-Miller PRNG (pseudo random number gen.)

- convenient even if better solutions exist
- I-D used to rely on Park Miller Linear Congruential PRNG

- seed the sequence
- each PR number $I_{j+1}$ is computed with:
  $$I_{j+1} = A \times I_j \pmod{M}$$
  with $A=16807$, $M=2^{31}-1$
- scale it between 0 and $maxv - 1$ (inclusive)

- not an issue with RFC 5170 (LDPC-staircase specs.)
  - because:
    - ✓ we generate many PR Numbers from the same seed
    - ✓ we scale with large $maxv$ values
Park-Miller PRNG (2)

- yet it’s totally inappropriate with RLC
  - ex. produce two repairs symbols from the \textit{same} encoding window
    - application chooses seeds in sequence (s, s+1, \ldots)
      - the obvious strategy (that many implementations will use)

- when scaling to [0; 255]

<table>
<thead>
<tr>
<th>seed</th>
<th>=&gt;</th>
<th>0</th>
<th>33</th>
<th>192</th>
<th>116</th>
<th>135</th>
</tr>
</thead>
<tbody>
<tr>
<td>seed=1   =&gt;</td>
<td>0</td>
<td>67</td>
<td>130</td>
<td>233</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>seed=2   =&gt;</td>
<td>0</td>
<td>100</td>
<td>68</td>
<td>95</td>
<td>152</td>
<td></td>
</tr>
<tr>
<td>seed=3   =&gt;</td>
<td>0</td>
<td>134</td>
<td>5</td>
<td>212</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>seed=4   =&gt;</td>
<td>0</td>
<td>19</td>
<td>96</td>
<td>13</td>
<td>127</td>
<td>171</td>
</tr>
<tr>
<td>seed=10000 =&gt;</td>
<td>19</td>
<td>129</td>
<td>206</td>
<td>244</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>seed=10001 =&gt;</td>
<td>19</td>
<td>163</td>
<td>143</td>
<td>106</td>
<td>188</td>
<td></td>
</tr>
<tr>
<td>seed=10002 =&gt;</td>
<td>19</td>
<td>163</td>
<td>143</td>
<td>106</td>
<td>188</td>
<td></td>
</tr>
</tbody>
</table>
Park-Miller PRNG (3)

- [...]  
  - even worse when scaling to [0; 15] (needed for sparse equations)  
    - high probability of duplicated coefficients across repair symbols

- conclusion: if we want to let applications freely select seeds, P-M PRNG is not the right choice
TinyMT32 PRNG

- Tiny Mersenne Twister, 32-bit version
  - compact version of the renown/widely used Mersenne Twister PRNG
    - see https://en.wikipedia.org/wiki/Mersenne_Twister
  - provable quality 😊
  - comes with a reference C implementation
    - lightly edited version added in appendix to draft-ietf-tsvwg-rlc-fec-scheme-05

- solved all problems
- question: performance impacts?
TinyMT32 PRNG (2)

- Compulab ARM Cortex- A15@1.5GHz CPU

```
#seeds=1000000  #coefs per seed=20
P-M: duration=6.646820
TinyMT32: duration=3.477315
  tiny / P-M = 0.523 = 1 / 1.912

#seeds=1000000  #coefs per seed=100
P-M: duration=32.957823
TinyMT32: duration=14.058373
  tiny / P-M = 0.426 = 1 / 2.347

#seeds=1  #coefs per seed=1000000
P-M: duration=0.334906
TinyMT32 duration=0.134338
  tiny / P-M = 0.400 = 1 / 2.5
```

TinyMT32 more than 2 times faster
TinyMT32 PRNG (3)

- sometimes it’s the opposite: MacBookPro 15p
  - here TinyMT32 is upto 1.7 times slower than Park-Miller PRNG
  - … probably a sub-optimal instruction set usage/compiler problem
  - we didn’t investigate to find exact reason

- in any case initialization is a bit long, but production of PR numbers with TinyMT32 is fast 😊
TinyMT32 PRNG (4)

- NB: we fixed 3 internal parameters:
  - TINYMT32_MAT1_PARAM 0x8f7011ee
  - TINYMT32_MAT2_PARAM 0xfc78ff1f
  - TINYMT32_TMAT_PARAM 0x3793fdff
    ✓ those are good official values, but many other triples could be used, leading to different PR number sequences

- when RLC_for_FECFRAME I-D will be published as RFC, this modern PRNG will be easily reusable in other documents