Lessons learned from “RLC FEC Scheme for FECFRAME” specification at TSVWG: PRNG

Vincent Roca, Inria PRIVATICS, vincent.roca@inria.fr

IETF102, Montreal, July 19th, 2018

informatiques mathématiques

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 Ij+1 is computed with:
   Ij+1 = A * Ij (modulo 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 ***same*** encoding window
    - ✓ **application chooses seeds in sequence (s, s+1, …)**
      - the obvious strategy (that many implementations will use)
  - when scaling to [0; 255]

```
seed=1     =>  0   33  192  116  135
seed=2     =>  0   67  130  233   16
seed=3     =>  0  100   68   95  152
seed=4     =>  0  134    5  212   33
[…]
seed=10000 => 19   96   13  127  171
seed=10001 => 19  129  206  244   52
seed=10002 => 19  163  143  106  188
```

coefs. for 1st repair
coefs. for 2nd repair
…

first source symbol of encoding window is not/badly protected ☹

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**