Thread-safe High Performance Pseudo-random Number Generation
Agenda

- Introduction
- Use cases
- API
- Pre-19.08 Implementation
- Goals for 19.08
- Implementation
- Initial Seeding
- Bounded PRNG
- Questions
Pseudo-random Number Generator (PRNG)

• An algorithm for generating a sequence of numbers
• Each number approximates a truly random number
• Deterministic
• Sequence determined by initial state – the seed
Use Cases

- Network protocol implementations
- Functional and performance testing
- Important aspects
  - Good statistical properties (e.g. uniformity and passing various tests)
  - Thread safety
  - Efficiency
  - Security
PRNG API

- `<rte_random.h>`
- `uint64_t rte_rand(void);`
  - Generates 64-bit pseudo-random number
- `void rte_srand(uint64_t seed);`
  - Provides opportunity for a user-specified seed
Pre-19.08: Implementation

- `rte_rand()`/`rte_srand()` wrappers around `lrand48()`/`srand48()`
- *`rand48()` is a part of the Single UNIX Specification and glibc
  - Uniform distribution
  - Range \([0 \rightarrow 2^{31}-1]\]
  - Not thread-safe (shared state) [DPDK bug 114]
  - Implemented with a 48-bit linear congruential generator (LCG)
- `rte_rand()` uses two `lrand48()` calls – high and low bits
- Only generates 62-bit of randomness
  - Bit 31 and 63 are always zero [DPDK bug 276]
19.08: Goals

1. Provide thread safety to allow use from lcore threads
2. Keep things simple and API/ABI backward-compatible
3. Higher-quality generated values (including truly 64-bit values)
4. Improved performance
5. Improved API documentation
6. Improved initial seeding
7. DPDK performance test suite extension
8. Unbiased, bounded PRNG
19.08: Out of Scope

- No `rte_rand32()`
  - Faster (~35% less overhead), but 64-bit still very fast
- No support for multiple distributions - stay uniform-only
- No support for multiple PRNG algorithms
- …for advanced functionality, use external library
19.08: General Design

- `rte_rand()` / `rte_srand()` API remains unchanged
- Introduce per-lcore PRNG
  - No `rte_rand_r()` or any user-managed state
  - Makes `rte_rand()` MT safe (for lcore worker threads)
- `rte_srand()` still MT unsafe
  - For use during application initialization
  - Seed used by each lcore is `<global-seed>`+<lcore_id>
- `<rte_random.h>` functions are moved from “static inline” in header file to regular, non-static functions
  - ABI addition, but backward compatible
19.08: Tausworthe Generator

- DPDK-native implementation
- Maximally equidistributed combined Tausworthe generator
  - Also known as Linear Feedback Shift Register (LFSR)
  - Well-known and well-analyzed
  - Described in this paper, with this errata paper
- Allows performant software implementation
- Five sequences per instance/lcore (40 bytes state)
- Natively producing a 64-bit number
- Fairly common, including Linux kernel usage, for similar purposes
  - DPDK code not based on this implementation
- Tausworthe sequences seeded using a LCG

```c
struct rte_rand_state {
  uint64_t z1;
  uint64_t z2;
  uint64_t z3;
  uint64_t z4;
  uint64_t z5;
} __rte_cache_aligned;
```
19.08: Initial Seeding

- Pre-19.08 relies purely on monotonic CPU wall-time clock (TSC)
- 19.08 improves initial seeding
  - Primary: getentropy() syscall to retrieve “truly” random value
    - Requires Linux libc 2.25+ and kernel 3.17+ or FreeBSD 12
  - 1st Fallback: HW seeding by rdsseed x86 instruction
    - x86_64 Broadwell or later
  - 2nd Fallback: TSC register (or equivalent)
    - All systems/hardware
rte_rand() Performance

System: Skylake @ 2.9 GHz
19.08: Bounded PRNG

- `rte_rand() % UPPER_BOUND -> range [0 - UPPER_BOUND-1]`
  - Constant power-of-2 `UPPER_BOUND` (i.e. $2^N$): Very fast and uniform
  - Constant non-power-of-2: Fast, but biased (reduced uniformity)
  - Variable `UPPER_BOUND`: Slower, often biased
  - Bias only significant for large `UPPER_BOUNDS`
- `rte_rand_max()` produces *unbiased* pseudo-random numbers with an upper bound
Bounded PRNG

Latency [clock cycles/op]

- **rte_rand() % CONSTANT**: 37
- **rte_rand() % variable**: 22
- **rte_rand_max() best case**: 60
- **rte_rand_max() worst case average**: 47
- **lrand48()**: 27
- **tausworthe**: 69

(Bar chart showing the latency comparison between different PRNG methods.)
Questions?

Mattias Rönnblom
<mattias.ronnblom@ericsson.com>