IEEE P802.11
Wireless Access Method and Physical Layer Specifications

Title: **Standard Randomness, or**
A proposed change to Section 5.2.5 in the draft IEEE p802.11-93/20b1

Author: Barry A. Dobyns
Photonics Corporation
2940 North First Street
San Jose, CA 95134
408-955-7930
408-955-7940 fax
76527,266@compuserve.com or dobyns@acm.org

Abstract: **What's Wrong:**
The editor's note in section 5.2.5 at the top of page 60 of IEEE p802.11-93/20b1 reads:
[Need definition for Random() function. JES.]

How To Fix It:
Supply a definition, like the one in this document.

Motions:
Resolved, that the proposed text changes in 11-94/181 be incorporated into the draft standard IEEE p802.11-93/20b1 section 5.2.5 in it's next revision by the editors.

Proposed Change

The proposed text changes to define a suitable Random() function is shown below, with change bars, in the affected section from the draft standard, IEEE p802.11-93/20b1. Because the draft standard did not include a definition for Random(), the definition that *uses* Random() must be brought into conformance with the supplied definition.

5.2.5. Random Backoff Time

STA desiring to initiate transfer of asynchronous MPDUs shall utilize the carrier sense function to determine the state of the media. If the media is busy, the STA shall defer until after a DIFS gap is detected, and then generate a random backoff period for an additional deferral time before transmitting. This process resolves contention between multiple STA that have been deferring to the same MPDU occupying the medium.

~~Backoff Time = CW * Random() * Slot time~~

Backoff Time = (Random() modulo CW) * Slot time

where:

CW = An integer between CW_{min} and CW_{max}

```
Random() = procedure:
    begin
        constant integer32      a := 16807
        constant integer32      m := 2147483647
        constant integer32      q := 127773
        constant integer32      r := 2836
        static integer32        seed
        var integer32           lo, hi, test

        lo := seed / q
        hi := seed % q
        test := (a * lo) - (r * hi)
        if (test > 0) seed := test
        else seed := test + m

        random := seed
    end
```

Note that the value *seed* must be retained from one invocation of the Random() function to the next. Furthermore, the seed in the random number generator must be initialized to a unique value at MAC startup time, typically via MAC management.[1] If the seed is always initialized to the same value at startup time in all MACs, then there exists a substantial probability that the MAC in stations which are operating in the same BSS will synchronize their Random() machinery, thereby guaranteeing collisions.

Slot Time = Transmitter turn-on delay + medium propagation delay + medium busy detect response time.

[1]This seed value should be chosen by the implementors of a system to be a value for which they have strong expectations of uniqueness for each station. A good value to use which is known to be unique is the Globally Administered Unique MAC layer address.

Rationale

In order for the random backoff technique in the MAC to operate properly, stations which desire to transmit "at the same time" must have substantial probability of choosing different random backoff times.

This is best insured by choosing a random number generator with several desirable features, all offered by the proposed one.

1. Long Period.
   The offered random number generator has period 2^32. The period of a random number generator is the number of samples which must be taken from the generator before it repeats the same sequence. Given the period of the offered generator two stations have probablity of roughly 2^31 of synchronizing their generators. Generator synchronzation is always a problem to be wary of with deterministic random number generators.

2. Uniform distribution
   The offered random number generator is a member of the family of random number generators known as "Linear Congruential Random Number Generators", which are well documented in the literature to have excellent distribution characteristics. Distribution is the charateristic whereby subsequent random numbers differ from one another. (e.g. a generator with long period but monotonic distribution is easy to construct: a counter)

3. Uniform Randomness in all bits
   The offered random number generator has good randomness in all the bits of the produced, so that the use of only the low order bits still yields good randomneess.

4. "Inexpensive" implementation
   The offered random number generator is fully implementable with only 32 bit integer arithmetic. This is substantially less complex than floating point arithmetic at any reasonable bit width.

   The definition of BackoffTime in IEEE p802.11-93/20b1 reads:
   *BackoffTime = CW * Random() * SlotTime*
   Which strongly implies that Random() is a (0...1) floating point function. The offered random number generator will fulfill the same functional purpose as the implied floating point function in the draft, and will also result in a less complicated implementation, at possible reduced cost.

Sample Random() Implementation in 'C'

Somebody else can provide a sample implementation in VHDL.

```
/*  32-bit integer Linear Congruential Random Number generator
    with period 2^32, uniform distribution, and good uniformity
    of randomness in all bits
*/

static long seed = 1;

void SetSeed(lj)
long lj;
{
    seed += lj;
    }

long Random()
{
static long a = 16807;
static long m = 2147483647;
static long q = 127773;
static long r = 2836;

    long lo, hi, test;

    lo = seed / q;
    hi = seed % q;
    test = (a * lo) - (r * hi);
    if (test > 0) seed = test;
    else seed = test + m;

    return (seed);
    }
```