SAND2011-7533C

Tradeoffs in Targeted Fuzzing of Cyber Systems by Defenders and Attackers

12 October 2011

Jackson R. Mayo and Robert C. Armstrong
Sandia National Laboratories, Livermore, CA

Benjamin G. Davis
University of California, Davis, CA

Sandia National Laboratories is a multiprogram laboratory operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Fuzzing can locate faults and vulnerabilities in complex cyber systems

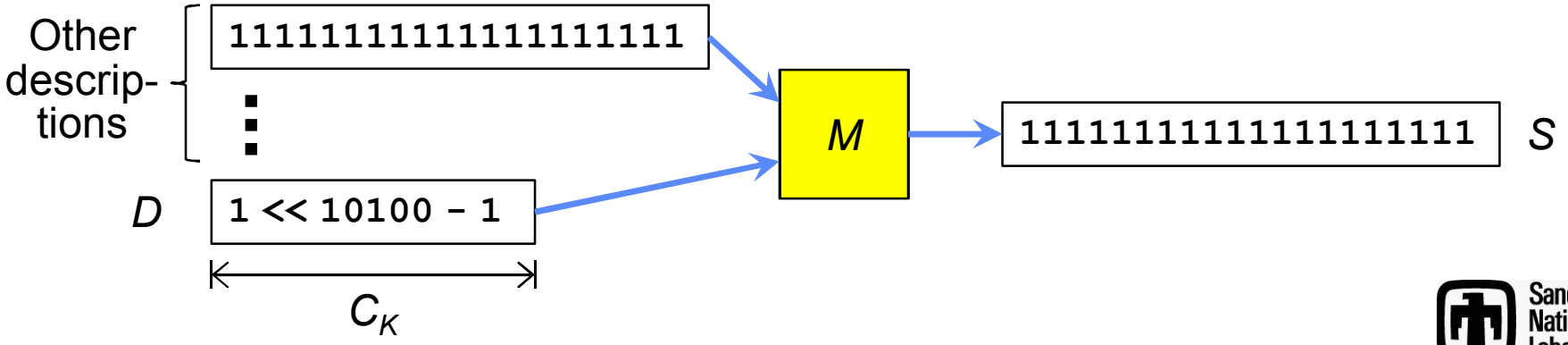
- **Fuzzing is automated randomized testing that samples vast input spaces to assess security**
  - Widely applicable, including for smart-grid devices
- **Fuzzing can operate on a black-box system but can also benefit from system understanding**
  - Complements analytical approaches that *reason* about the system (formal methods and complexity theory)
- **Effective fuzzing uses a targeted (non-uniform) distribution of test inputs**
- **Key questions:**
  - How should the sampling be targeted?
  - How can the resulting confidence be quantified?

Seek a relevant measure to organize the input space

- **A cyber system is taken as a general program**
- **Ultimately, the inputs of greatest interest – likely faults – are determined by the *semantics* of the program**
- **Some inputs are "simpler" than others in the natural representation that the program induces**
- **These simpler inputs are expected to be the most valuable fuzzing targets for defender and attacker**
  - Precedent 1: fuzzing by mutation of normal inputs
  - Precedent 2: cracking of low-entropy passwords
- **There is a rigorous notion of how "simple" a bit string is…**

Kolmogorov complexity distinguishes "natural" bit strings from noise

- **Given some universal Turing machine *M*: Kolmogorov complexity C_K of string *S* is length of shortest string *D* that, when fed to *M*, produces *S***
- ***D* is shortest "description" of *S* with respect to *M***
  - *D* is a "program" that runs on machine *M* to yield *S*
- **C_K is non-computable, but almost all *N*-bit strings have $C_K \approx N$ ("algorithmically random")**
- **Notional example showing low-complexity string:**



Sandia National Laboratories

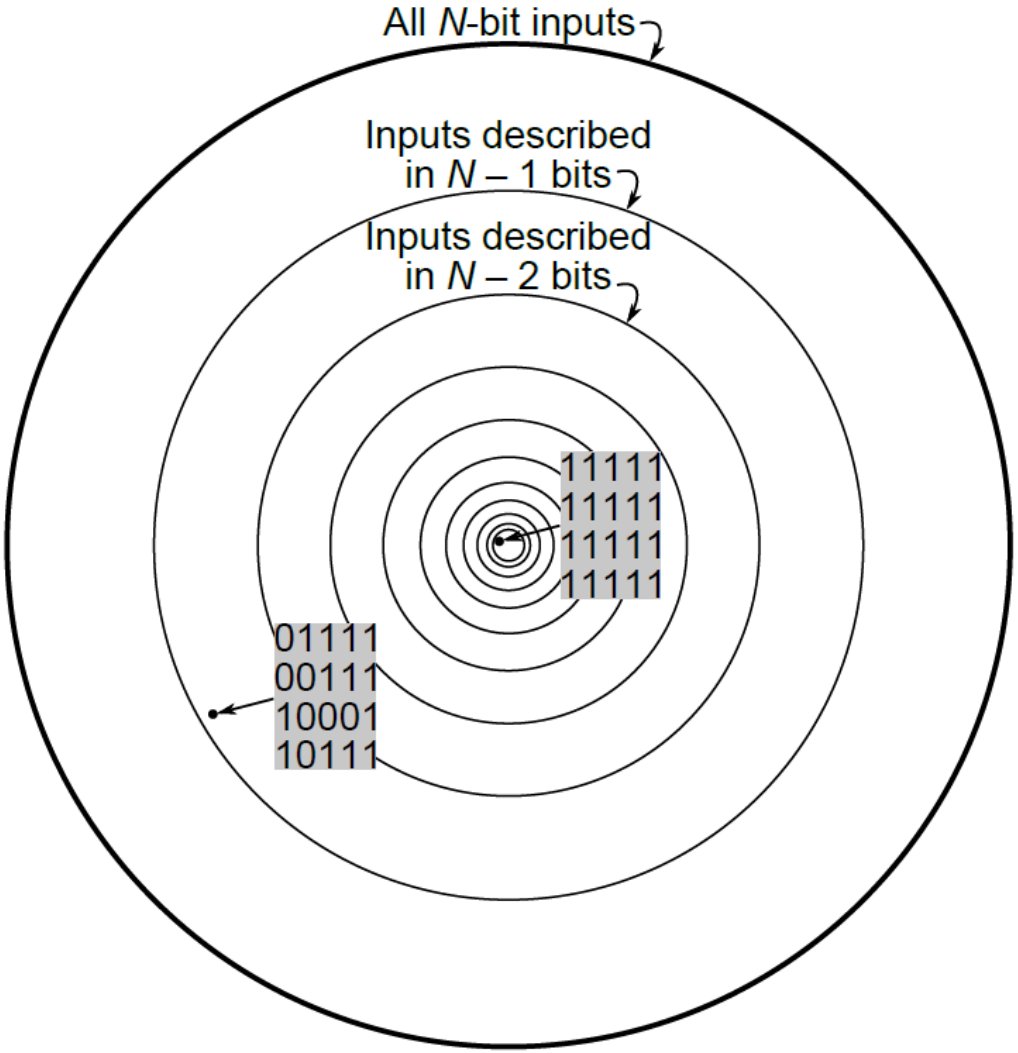
Kolmogorov complexity is relative but "asymptotically absolute"

- **View *S* as an input to the program being fuzzed**
  - Low C_K means *S* is "simple" and favored for sampling
- **Value of C_K depends on the Turing machine *M*, which is a "decompressor" that should reflect a natural input encoding for the program – e.g.:**
  - Databases: *M* most efficiently encodes SQL commands
  - Passphrases: *M* most efficiently encodes English words
- ***M* gives a semantically meaningful representation but, in the limit of long strings, C_K becomes independent of this representation**
  - A universal Turing machine can emulate any other via a *finite* interpreter (e.g., an English word table)

To quantify confidence in a generic program, must bound what attacker knows

- **If attacker grasps program semantics in a way defender does not, then attacker can embed this "key" in *M* and zero in on promising inputs that seem *high-complexity* (unguessable) to defender**
  - Shows the limits of fuzzing (effective undecidability)
- **Henceforth assume that defender and attacker share the same semantic understanding and thus employ essentially the same decompressor *M***
  - This seems plausible for very complex programs, and enables fuzzing to characterize security statistically
- **Define Kolmogorov complexity of inputs via this common representation**

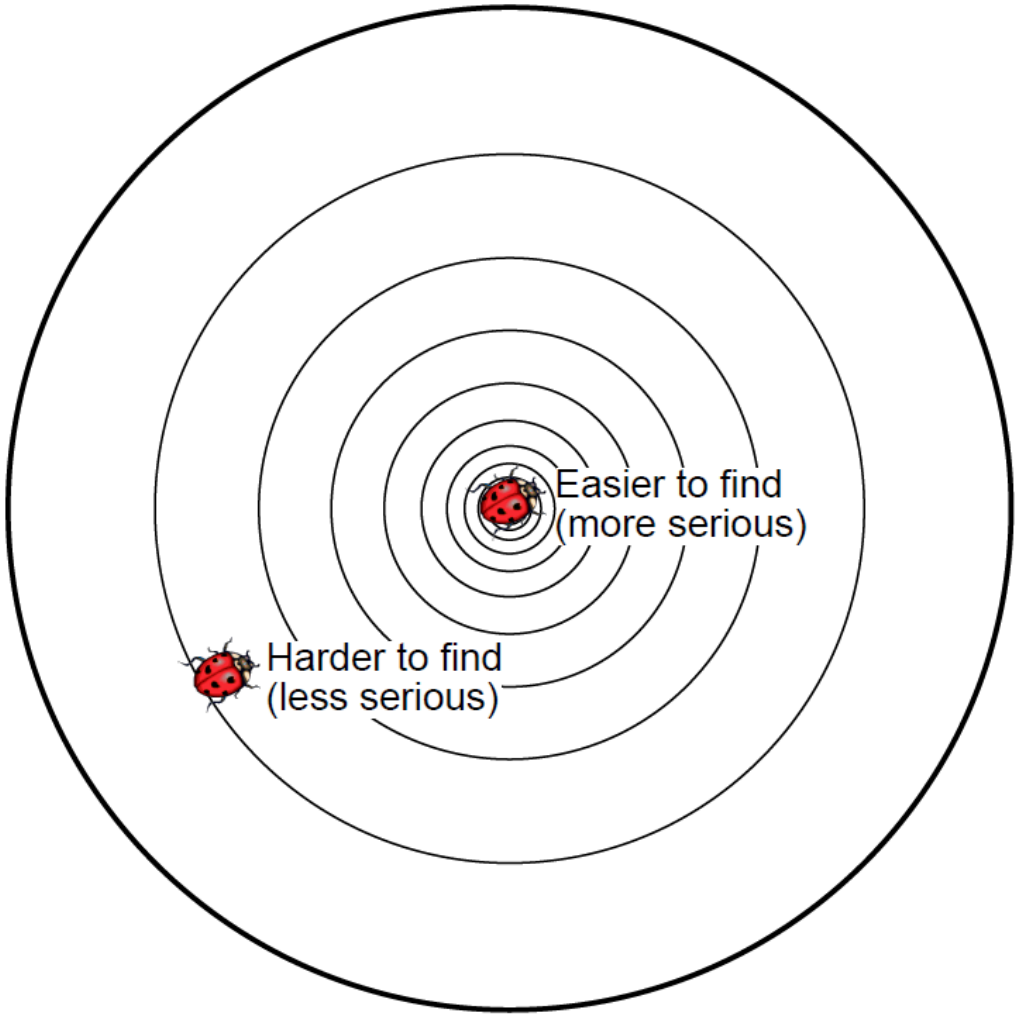
Kolmogorov complexity organizes the input space



- **Inputs that have a simple description (relative to available information) should be targeted for defender coverage because they form a smaller "corner" space (also more attractive to attacker)**

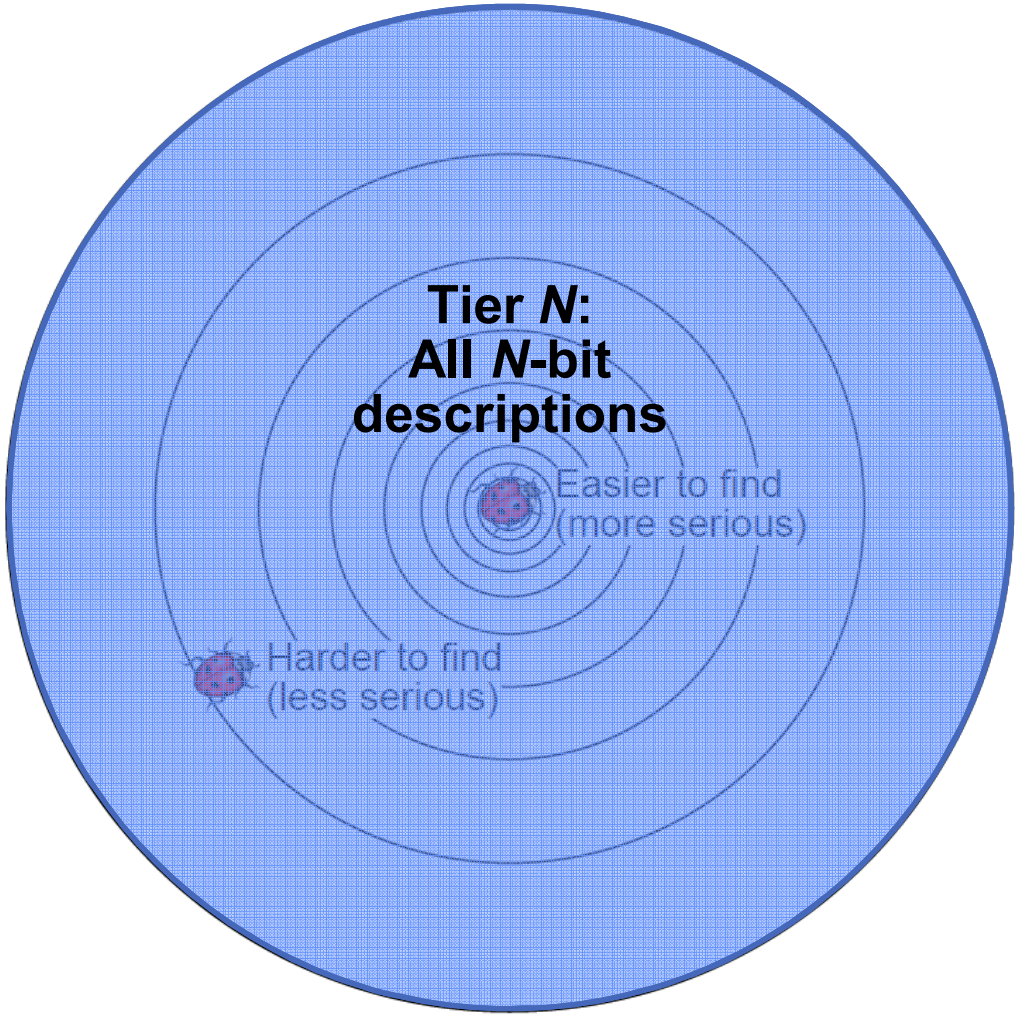
Sandia National Laboratories

Kolmogorov complexity organizes the input space



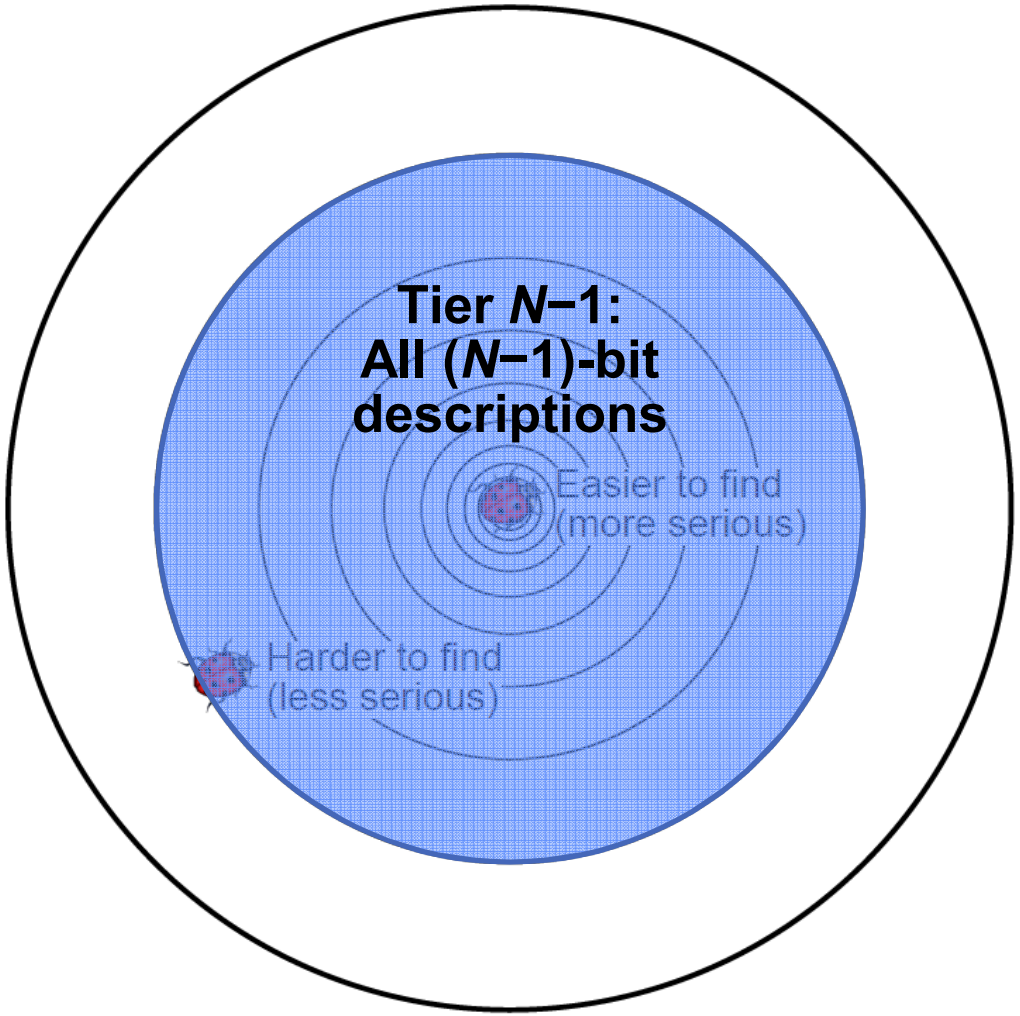
- Inputs that have a simple description (relative to available information) should be targeted for defender coverage because they form a smaller "corner" space (also more attractive to attacker)

Complexity-based sampling generates a "wedding cake" distribution



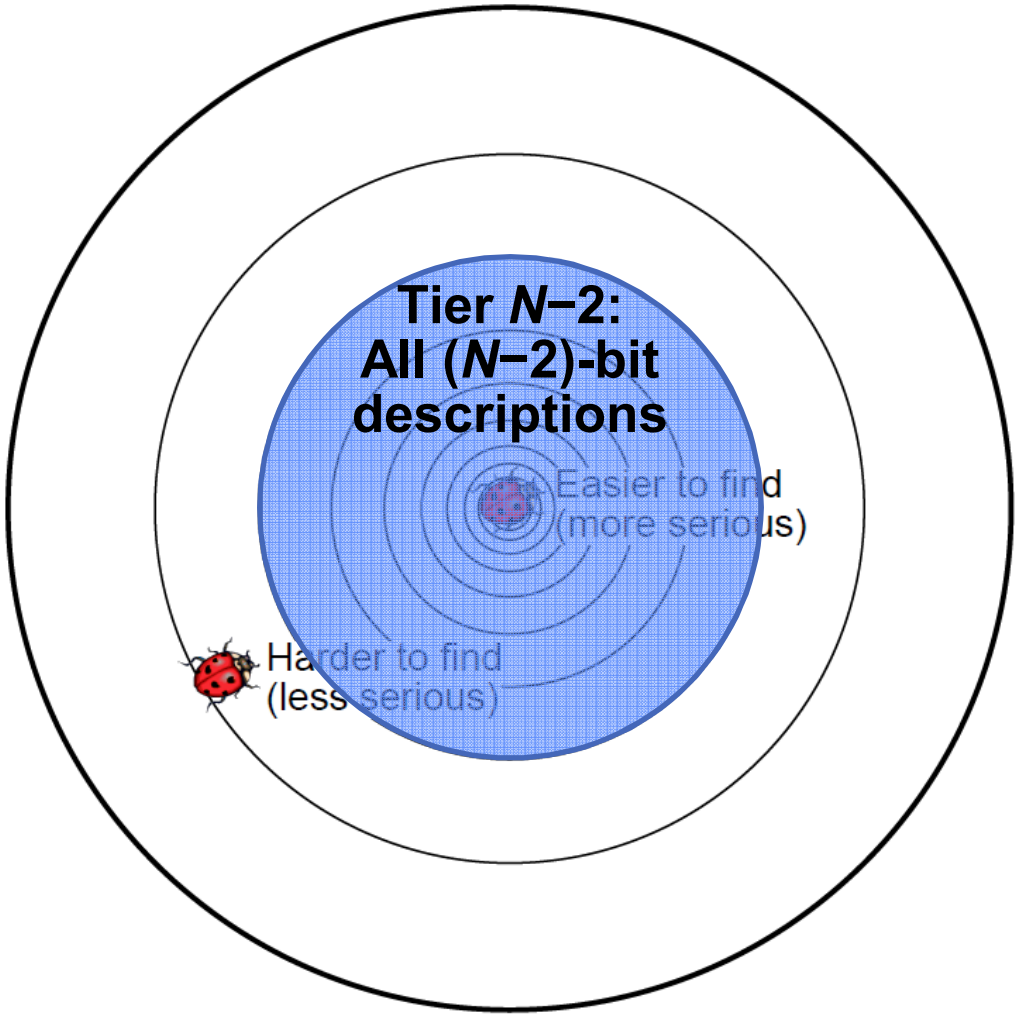
- Although C_K is non-computable, descriptions of length *k* can be sampled, yielding inputs with $C_K \leq k$
- Because low C_K is a small part of the space, there is no advantage to *under*sampling there; instead, tiers pile up

Complexity-based sampling generates a "wedding cake" distribution



- Although C_K is non-computable, descriptions of length k can be sampled, yielding inputs with $C_K \leq k$
- Because low C_K is a small part of the space, there is no advantage to *under*sampling there; instead, tiers pile up

Complexity-based sampling generates a "wedding cake" distribution



- Although C_K is non-computable, descriptions of length k can be sampled, yielding inputs with $C_K \leq k$
- Because low C_K is a small part of the space, there is no advantage to *under*sampling there; instead, tiers pile up

Sandia National Laboratories

A statistical strategy for fuzzing illustrates ability to quantify confidence

- Defender and attacker both can sample *N*-bit inputs from tiers $k = 0, 1, \ldots, N$ to locate faults
- Once program is deployed, attacker will find tier with highest fault rate and focus fuzzing there
- So defender wants to focus fault-patching effort on *minimizing* the *maximum* tier fault rate
- In tier k, after R_k random tests have yielded F_k faults of which P_k have been patched, defender's Bayesian estimate of tier fault rate is

$$\langle f_k \rangle = \frac{2^k - R_k}{2^k} \frac{F_k + 1}{R_k + 2} + \frac{F_k - P_k}{2^k}$$

A statistical strategy for fuzzing illustrates ability to quantify confidence

- **Defender samples each successive test from tier with current highest estimated fault rate**
  - Estimated fault rates decline due to increasing statistics and elimination of faults
  - High confidence (even for large *k*) can be obtained well short of exhaustive (2^k) fuzzing effort
- **Patching has negligible effect for large *k* but can dramatically reduce fault rates at small *k***
  - Purging "weak passwords" from the system

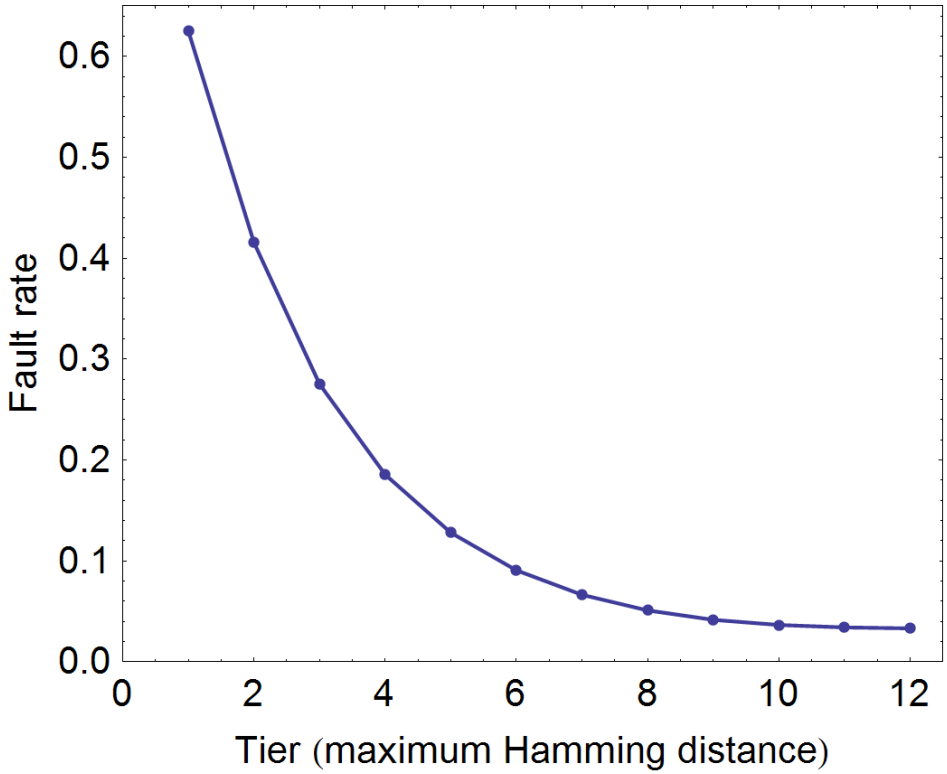
$$\langle f_k \rangle = \frac{2^k - R_k}{2^k} \frac{F_k + 1}{R_k + 2} + \frac{F_k - P_k}{2^k}$$

Experiments with programs developed by machine learning seem to corroborate

- **Boolean networks (BNs) are a flexible representation of digital logic**
- **Create BN circuits to perform string recognition: output 1 for a particular "gold" input string and 0 for all other inputs**
  - A fault is a non-gold input that produces a 1
- **Ensure objectivity and number of programs sufficient to gather meaningful statistics**
  - Create the programs automatically by machine learning
  - Using a genetic algorithm (mutation and recombination) to arrive at implementations with small but non-zero fault rates representative of more complex programs

These simple "grown" programs give a preliminary example of fault statistics

- 16-bit string recognizer has small enough input space for *exhaustive* fuzzing
- Model for machine *M*: an edit function based on the gold string, initially using bitwise edits (approximate C_K by Hamming distance)
- As expected, faults are most common close to the gold string



Sandia National Laboratories