Inter-Packet Network Delays as a Time-Series Random Generator

Micah Thornton, Neha Joshi, Qiliang Zeng

April 26, 2017

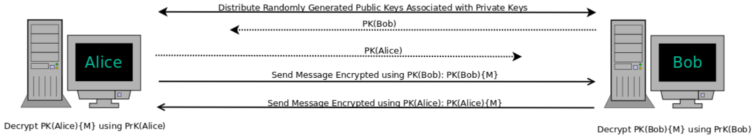
History of Random

FIGURE 1: ANCIENT BONE DIE USED BY ROMAN EMPIRE CIRCA 100-300 AD

- In ancient times, random values used for **gambling, forecasting and fortune-telling**
- In modern times, they are used for **Security and Simulation**
- Process random iff all outcomes have equal odds of occurrence. (Shannon)

Modern Applications of Random Values (1)

FIGURE 2: EXAMPLE APPLICATION OF RANDOM VALUES TO PUBLIC KEY CRYPTOGRAPHY



- In cryptography:
  - RSA: RNs are used to generate primes (No RNG specified)
  - 3-DES: RNs used as key-bundle (Specific RNG ANSI x9.31)
  - Blowfish: RN used a 52-bit key (No RNG specified)
  - Twofish: RN used as up to 256-bit key (No RNG specified)
  - AES: RNs used as key-IV-salt bundle (NIST specified RNG)
- In science:
  - Statistics: Taking random sample
  - Analysis: Extraction of signal from noise
  - Simulation: Providing a spectrum inputs

Modern Applications of Random Values (2)

Figure 3: Famous casino & resort in Las Vegas

- In Gambling:
  - Card Games: Poker, Black Jack, etc...
  - Die Games: Craps
  - Wheel Games: Roulette
  - Slot Machines: 'fair' results (usually biased)
- In CSE 7344 Research Projects:
  - Generation of Random IP addresses from a range (Mirai)
  - Sending random packets (Domino)

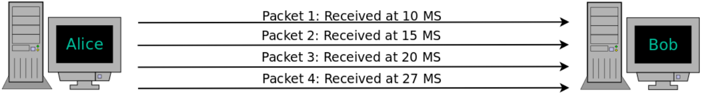
Approaches to Random Generation

FIGURE 4: GIUSEPPE LODOVICO LAGRANGIA

- Pseudo-Random Number Generators (PRNGs)
  - Shift Registers (LFSR, NLFSR) - Golomb (1948)
  - Linear Congruential Generators (LCG) - D. H. Lehmer (1949)
  - Blum Blum Shub (BBS) - Blum,Blum, and Shub (1986)
  - Mersenne Twister (MT) - Matsumoto & Nishimura (1997)
- True Random Number Generators (TRNGs)
  - Atmospheric Noise (random.org)
  - Radioactive Decay (hotbits.org)

Entropy Extractors for TRNGs

FIGURE 5: EXAMPLE ENTROPY EXTRACTION (THE HOTBITS WAY)



$$T_1 = P_2 - P_1 = 15 - 10 = 5$$

$$T_2 = P_4 - P_3 = 27 - 20 = 7$$

if T1 > T2:
    record one
if T1 < T2:
    record zero
if T1 = T2:
    record nothing

A Posteriori Extraction Method

Given X such that $X = \{x_1, x_2, x_3, ..., x_n\}$

$$Q_2 = \{x \in X | P(X > x) = P(X < x) = 0.5\}$$

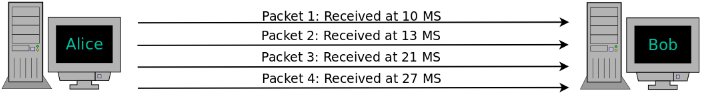
$$R_\psi(x_i) = r_i = \begin{cases} 1 & x_i > Q_2 \\ 0 & x_i < Q_2 \end{cases}$$

Hence, the entropy is extracted into the binary value: $r_1 r_2 r_3 r_4 ... r_n$

Note: alternative measures of center can be used in the place of Q_2 but only Q_2 maximizes the extracted entropy

A Posteriori Extractor for Inter-Packet Delays Example

Figure 6: Example Entropy Extraction (A Posteriori method)



$$T_1 = P_2 - P_1 = 13 - 10 = 3$$

$$T_2 = P_3 - P_2 = 21 - 13 = 8$$

$$T_3 = P_4 - P_3 = 27 - 21 = 6$$

$$Q_2 = 6$$

```
for T_i:
    if T_i > Q_2:
        record one
    else:
        record zero
```

A Posteriori Maximizes Shannon's Entropy (1)

[PROOF:]

Given a *supposedly* random sample

$$X = \{x_1 \in \mathbb{R}, x_2 \in \mathbb{R}, x_3 \in \mathbb{R}, \ldots, x_n \in \mathbb{R}\}$$

We define the random variable α in terms of the median (or second quartile) of X

$$\alpha : \mathbb{R} \to \mathbb{B}$$

$$P(\alpha = 0) = p_0(\alpha) = \frac{|\{x|x < Q_2(X)\}|}{|X|} = \frac{1}{2}$$

$$P(\alpha = 1) = p_1(\alpha) = \frac{|\{x|x > Q_2(X)\}|}{|X|} = \frac{1}{2}$$

The formula for the entropy of a string of Bernoulli trials (or a 'bitstring') is given:

$$H(p_0(b), p_1(b)) = -(p_0(b)\log_2(p_0(b)) + p_1(b)\log_2(p_1(b)))$$

We can maximize the Entropy function as so:

$$\nabla H(p_0, p_1) = \Big(\frac{\partial H}{\partial p_0}, \frac{\partial H}{\partial p_1}\Big) = \Big(-\frac{\ln(p_0)+1}{\ln(2)}, -\frac{\ln(p_1)+1}{\ln(2)}\Big)$$

Maximizing we find

$$\frac{-\ln(p_0) - 1}{\ln(2)} = 0 \implies \ln(p_0) = -1 \implies p_0 = \frac{1}{e}$$

$$\frac{-\ln(p_1) - 1}{\ln(2)} = 0 \implies \ln(p_1) = -1 \implies p_1 = \frac{1}{e}$$

A Posteriori Maximizes Shannon's Entropy (2)

This seemingly odd result is because there is an *inherent* dependence among these two values, expressed mathematically as $p_0 + p_1 = 1$, in our first maximization attempt, we neglected to account for the hard-restraint $p_0 + p_1 = 1$ In constraining the original optimization we have the following system:

$$\frac{-\ln(p_1) - 1}{\ln(2)} = 0 = \frac{-\ln(p_0) - 1}{\ln(2)}$$

$$p_1 = 1 - p_0$$

$$\frac{-\ln(1 - p_0) - 1}{\ln(2)} = \frac{-\ln(p_0) - 1}{\ln(2)} \implies 1 - p_0 = p_0$$

$$\implies p_0 = 0.5 \implies p_1 = 1 - 0.5 = 0.5$$

Because $p_0(\alpha) = p_1(\alpha) = 0.5$ by definition, we have maximized the entropy function for the constraint

$p_1 + p_0 = 1.$ $\square$

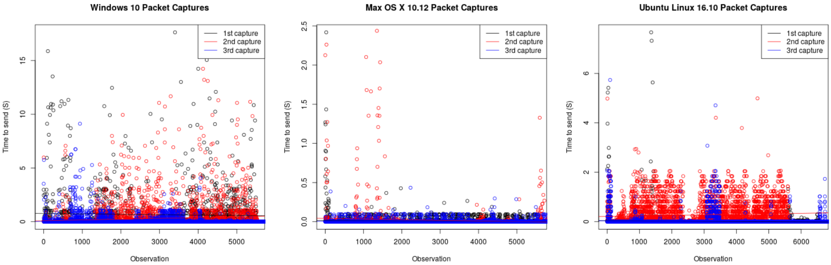
Experimental Set-Up

- Inter-Packet Timings: time differences between packet arrivals
- Arrival times (in μs) captured by Wireshark & TCPdump
- Five machines used:

Machine	OS	CPUs	RAM	Speed
1	Windows 10	2	8 Gb	2.35 GHz
2	MacOS 10.12	2	8 Gb	2.6 GHz
3	Ubuntu 16.10	8	16 Gb	2.6 GHz
4	Ubuntu 17.04	8	16 Gb	2.8 GHz
5	Ubuntu 17.04	8	32 Gb	3.2 GHz

Initial Packet Capture Timings

Figure 7: Initial Packet Captures (Wireshark)



- 3 Captures done on machines 1-3 $\approx$ 5000 packets ea.
- Significant fraction of data is below 100 ms
- Note: Scales are different (Mac OSX streaming during cap.)

ENT results

FIGURE 8: FOURMILABS ENT RESULTS

	Entropy		Arithmetic Mean		Serial Correlation		Compression		
	bits/byte	*bits/bit*	*by byte*	*by bit*	*by byte*	*by bit*	*size (b)*	*comp. size (b)*	*ratio (%)*
Windows 10									
Capture 1	6.666783	0.999995	126.5384	0.4986	-0.104782	0.207267	2200	2200	0
Capture 2	7.196086	1.000000	125.5073	0.4998	0.083004	0.209302	5504	5504	0
Capture 3	7.362089	1.000000	126.5384	0.4997	0.374569	0.232915	11472	11472	0
Mac OS X 12.10									
Capture 1	7.198828	1.000000	125.7977	0.5000	0.326609	0.070809	5536	5536	0
Capture 2	7.229747	1.000000	126.3883	0.5000	0.249999	0.119658	6552	6552	0
Capture 3	3.843544	0.980664	71.4336	0.4183	0.397400	0.018324	11680	11563	1
Ubuntu Linux 16.10									
Capture 1	7.229747	1.000000	126.3883	0.5000	0.249999	0.119658	6552	6552	0
Capture 2	6.973394	0.999999	127.8026	0.4993	0.313810	0.307581	5592	5592	0
Capture 3	5.304753	1.000000	128.1367	0.5001	-0.003104	0.682508	50080	50080	0
Averages									
Linux	6.503	1.0	127.4	0.4998	0.188971	0.3699	-	-	0
Mac	6.091	0.9936	107.87	0.4728	0.3247	0.06960	-	-	0.3333
PC	7.075	1.0	126.2	0.4994	0.11760	0.2165	-	-	0
Reference									
Hotbits	7.916369	0.999995	128.7873	0.4987	0.031555	0.000973	16320	16320	0
Ideal	8.0	1.0	127.5	0.5	0.0	0.0	-	-	0
Cross Platform Average	6.5563	0.997867	120.49	0.49067	0.21042367	0.21867	-	-	10

Before and After on an Idle Network

Figure 9: Idle Before Hashing

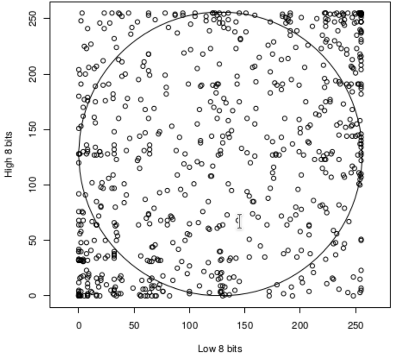
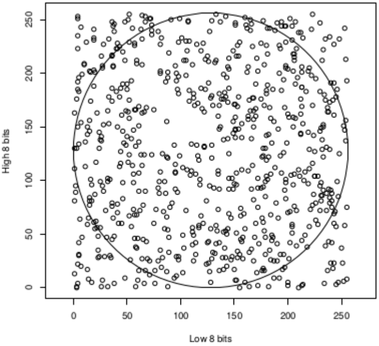


Figure 10: Idle After Hashing



Before and After on Busy Network

Figure 11: Busy Before Hashing

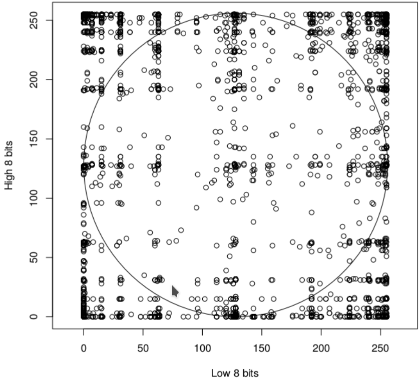
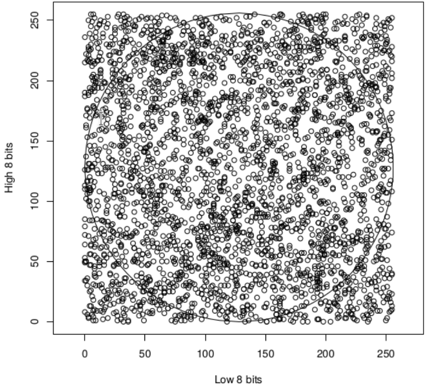


Figure 12: Busy After Hashing



Pros & Applicability

- Most modern-day PCs have network cards
- Could offer extra source of entropy
- Mixing with other sources is possible
- Two or more sources is shown to have better properties
- Could also be injected into dev/random in Linux

```
2543709069 7939612257 1429894671 5435784687 8861444581
2314593571 9849225284 7160504922 1242470141 2147805734
5510500801 9086996033 0276347870 8108175450 1193071412
2339086639 3833952942 5786905076 4310063835 1983438934
1596131854 3475464955 6978103829 3097164651 4384070070
7360411237 3599843452 2516105070 2705623526 6012764848
3084076118 3013052793 2054274628 6540360367 4532865105
7065874882 2569815793 6789766974 2205750596 8344086973
```

FIGURE 13: RANDOM NUMBERS?

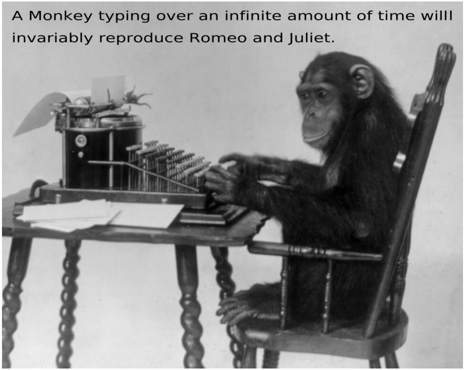
Potential Improvement & Future Work

National Institute of
Standards and Technology
U.S. Department of Commerce

- We have accumulated $\approx$ 8 Mb of random data
- We have preliminary NIST STS results and Dieharder Results
- Test strategy on different networks and devices
- Attempt method with different time series
- Intend to use Hummingbird 2 python script written during this project, in place of hash
- Suggestions?

Thankyou for your Attention: Questions?

FIGURE 15: MONKEY ON TYPEWRITER



References