COINMETRICS

THE SIGNAL & THE NONCE

TRACING ASIC FINGERPRINTS TO RESHAPE OUR UNDERSTANDING OF BITCOIN MINING

By Karim Helmy, Lucas Nuzzi, Alex Mead, Kyle Waters, and the Coin Metrics Team

SUMMARY

This paper introduces a methodology for determining the market share of every major Bitcoin mining machine, or ASIC, by analyzing nonce patterns and incorporating data sourced from real-world ASICs. Using novel data produced by our analysis, we can generate estimates to Bitcoin's electricity consumption and network efficiency with far greater accuracy over the existing body of research. By knowing the ASIC distribution over time, we can estimate not only the average efficiency, which is crucial for miners' competitiveness but also the e-waste generated by Bitcoin mining, a contentious issue in its own right. This approach paves the way for us to extract new insights into some of Bitcoin's most critical issues.

Introduction

Blockchains are often hailed for their transparency. By their very definition, blockchains are used to capture all financial operations related to a cryptoasset. That said, while they provide an abundance of readily available data, much of that data is opaque or outright illegible. Most often, sourcing on-chain data is insufficient— in order for it to be truly useful and insightful, it must first undergo cleansing, normalization, and contextualization through various methodologies and supporting data sets, all of which happen off-chain.

In this article, we'll discuss one such methodology: a breakthrough in the field of mining data that has enabled us to determine the market share of every major machine, or ASIC, used to mine Bitcoin. This is more than just an exercise in data analytics, though. By estimating the distribution of specific ASICs used in Bitcoin mining over time, we can gain more accurate insights into the Bitcoin mining industry and bring more objective data to the ongoing dialogue among mining industry experts, Bitcoin users, and policymakers.

Like Coin Metrics' previous miner identification methodology, our findings and the resulting data set are based on *nonce analysis*. The word "nonce" is short for "number used once" and it is essentially the special number Bitcoin miners are searching for during the mining process. As we previously disclosed, there are patterns in those numbers that enable us to associate them with specific ASIC models.

However, unlike previous attempts at hardware fingerprinting, this methodology does not rely heavily on circumstantial evidence like the timing of an ASIC release coinciding with the appearance of a new pattern on-chain. Instead, this new methodology incorporates off-chain data sourced directly from real-world ASICs, making it more generalizable and thus reducing the barrier to onboarding new model types. As a result, this new methodology is more accurate and easily extensible to newer hardware models.

What makes this data set particularly exciting is not only the ability to determine the predominance of specific ASICs mining Bitcoin but also the plethora of additional metrics that can be derived from that. Chief among these metrics is a substantially more accurate assessment of Bitcoin's electricity consumption, one of the industry's most contentious topics. Previous attempts at assessing Bitcoin's power draw missed a critical element that can only be attained with this type of ASIC-level data: hardware efficiency. As the mining industry has evolved, ASICs have become substantially more efficient, generating more hashes per second and per unit of power drawn. That dynamic is not captured fully by previous methodologies, which at times leads to considerable overestimations of power draw.

What's more, if you know the distribution of ASICs at a point in time, you can get a good idea of the average efficiency, which is a key metric for miners looking to assess the competitiveness of their operations. Finally, we can use the hardware distribution to estimate the e-waste generated by Bitcoin mining, yet another controversial topic.

What is a nonce, anyway?

The word 'mining' has become ubiquitous when describing the process through which new blocks are appended to a blockchain and new coins are issued. The analogy is fitting: miners allocate resources to this activity and, with some luck and adequate equipment, they are able to 'extract' new coins. The analogy, however, is not very helpful in contextualizing what is actually happening under the hood, and what miners are actually doing in this process. In order to understand that process more deeply, perhaps another analogy would be more appropriate.

Under the hood, what miners are doing more closely resembles the work of a locksmith trying to open a door for which there is no key. But instead of one locksmith, Bitcoin features thousands of them who are all simultaneously trying to produce a key that can open this door. Locksmiths with more resources, such as specialized machines, are able to produce keys faster and are thus more likely to find a valid key. Locksmiths can also join forces and work together in teams, or *pools*, to increase the likelihood of one of them finding a valid key.

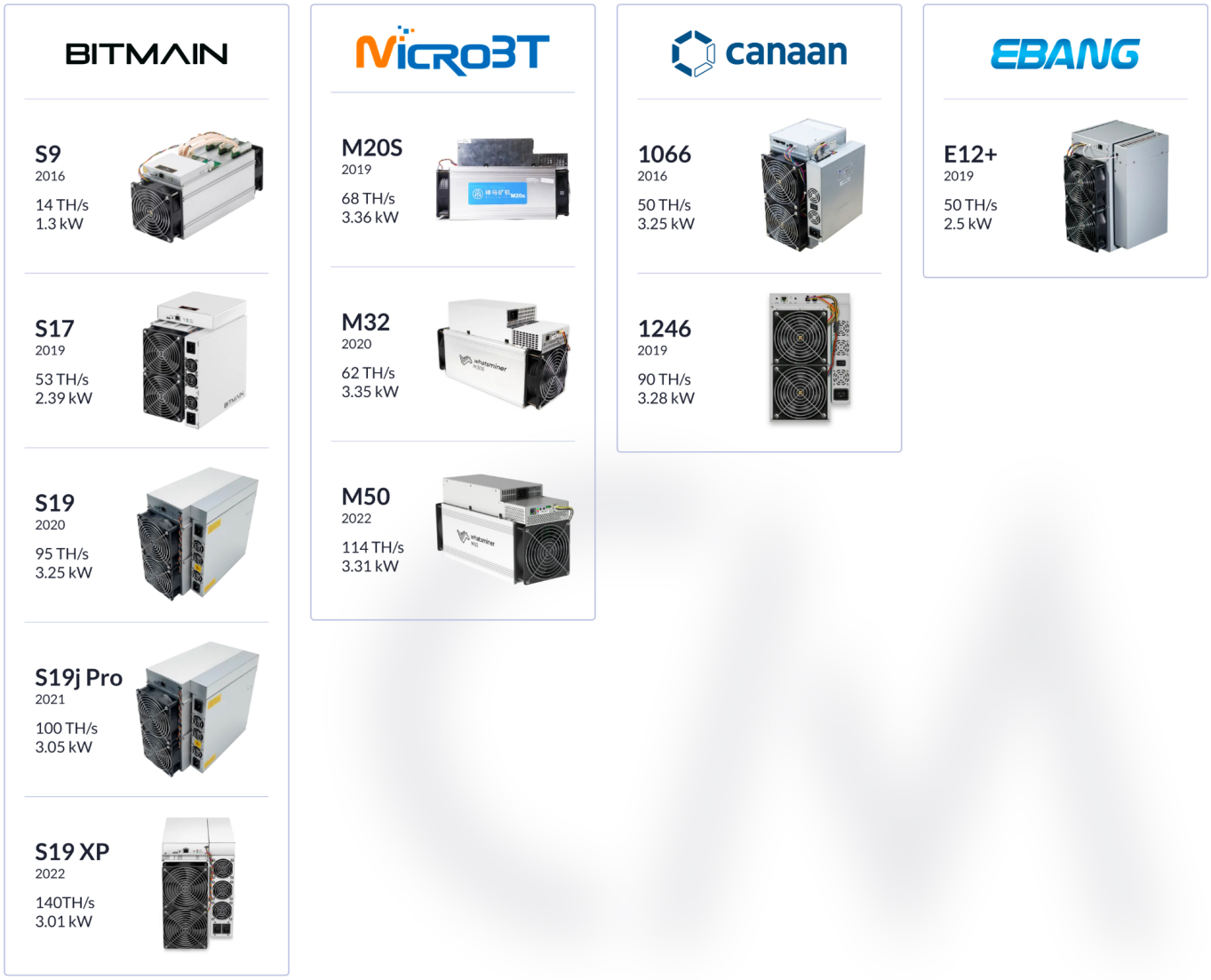
In this analogy, the keys that locksmiths are mass producing are nonces. Depending on the machine and process used to produce each key, that key will have a distinct pattern, perhaps in its shape or material. These are essentially the patterns this analysis attempts to identify. In Bitcoin vernacular, the key that is ultimately able to open the door is called a *golden nonce*. This key is shared with all other locksmiths who can then verify if it in fact opens the door. If the door opens, the key itself is proof that a locksmith has allocated resources to finding the correct one. This is why this process is called Proof-of-Work.

Collecting and Parsing Nonces

As alluded to earlier, the backbone of this methodology are nonces sourced directly from real-world Bitcoin ASICs (application-specific integrated circuit, or hardware built specifically for Bitcoin mining). A wide range of models were used, including bleeding-edge miners such as the S19XP and the M50 (both released in 2022). In this initial version of the analysis, 11 different ASIC models were

considered, produced by 4 distinct manufacturers. They range in their release year, ability to generate as many terahashes per second (TH/s) as possible, as well as their subsequent power draw in kilowatts (kW). The table below shows the range of miner performance and power draw across the models—and also allows us to see the distribution of ASIC electric *efficiency* (TH/s generated per kW of power draw). While this analysis did not consider certain ASIC models like Intel's Blockscale and Bitfury's Clarke, they represent a minor portion of the overall network hashrate and their exclusion does not substantially influence our methodology and final results.

FIG 1: Bitcoin ASIC Model Specifications



Source: *Coin Metrics, ASIC Miner Value*

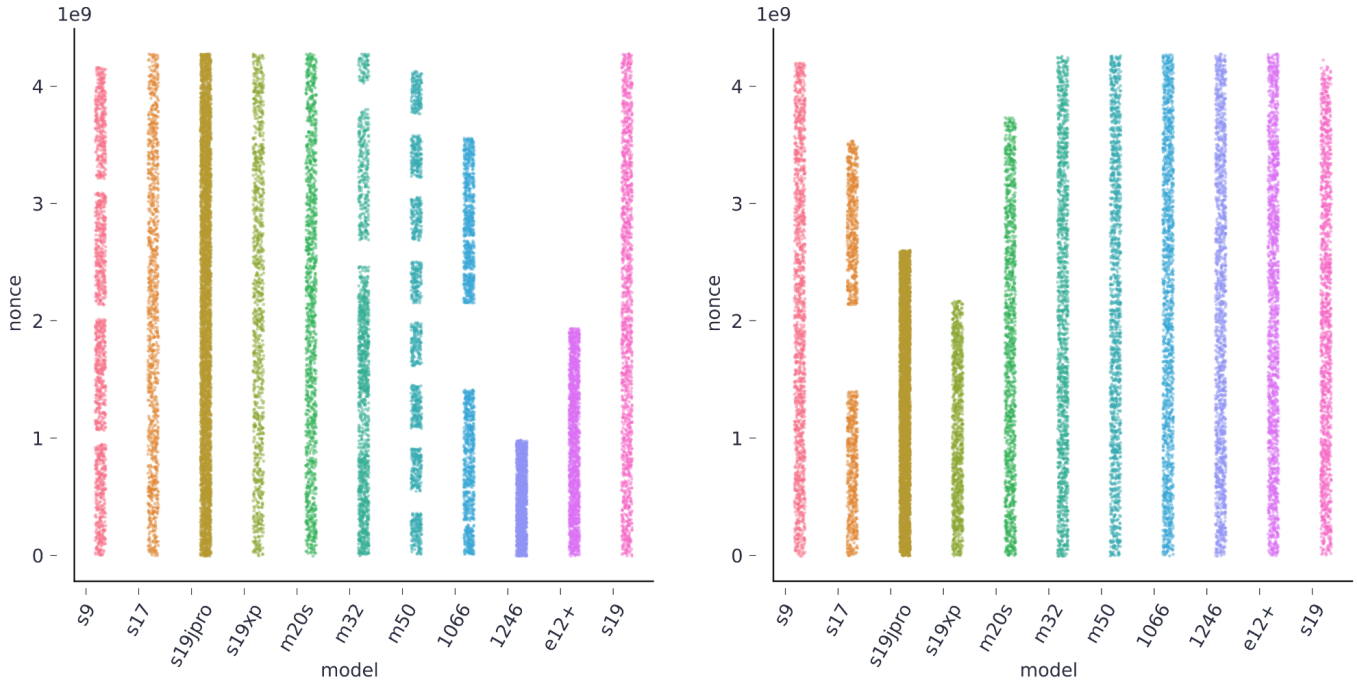
Each model was then connected to a proxy pool running Luxor Relay, which intercepted and logged a large sample of nonces generated by each ASIC. This is a substantial improvement over the previous method we used to source nonces, which was limited to mined blocks. To put things into perspective, consider that in a typical month, there are roughly 4 thousand unlabeled data points that can be sourced on-chain, whereas this methodology allowed us to collect a dataset of over 27 thousand data points, labeled by the model that produced them.

Once nonces are collected, they need to be encoded, or converted, to different formats so that patterns can be more easily recognized. Each nonce is encoded using two distinct methods to store data, namely, little-endian and big-endian notation. In essence, these are two ways to represent and store data in computer memory. In big-endian representation, the most significant byte (the one with the highest value) is stored first whereas the least significant byte (the one with the lowest value) is stored last. To put it very simply, imagine you're reading a number out loud – in big-endian, you'd say the most significant digit first (e.g. "three hundred and sixty-five"), while in little-endian you'd say the least significant digit first (e.g. "five sixty-three hundred").

In Bitcoin, little-endian notation is widely used to represent data. Natively within the Bitcoin protocol itself, nonces are a 32-bit unsigned (non-negative) integer that's encoded in little-endian format. In Stratum, the most used protocol for communications between miners and mining pool operators, nonces are represented in hex format. These are basically different ways to represent numbers and each has its own set of trade-offs. But as it turns out, representing nonce data using different methods can expose different patterns hidden in the nonce.

There are several additional details around little and big-endian notation, but the general idea is rather simple: in order to identify all possible ASIC patterns, each nonce is represented in two integer dimensions: little-endian and big-endian. This method enables us to identify unique patterns using either representation. Remarkably, we have found that, through this method, the overwhelming majority of ASICs leave an identifiable pattern in at least one of these spaces, even when different operating systems are used.

FIG 2: Nonce Patterns in Big-Endian *(Left)* and Little-Endian *(Right)*

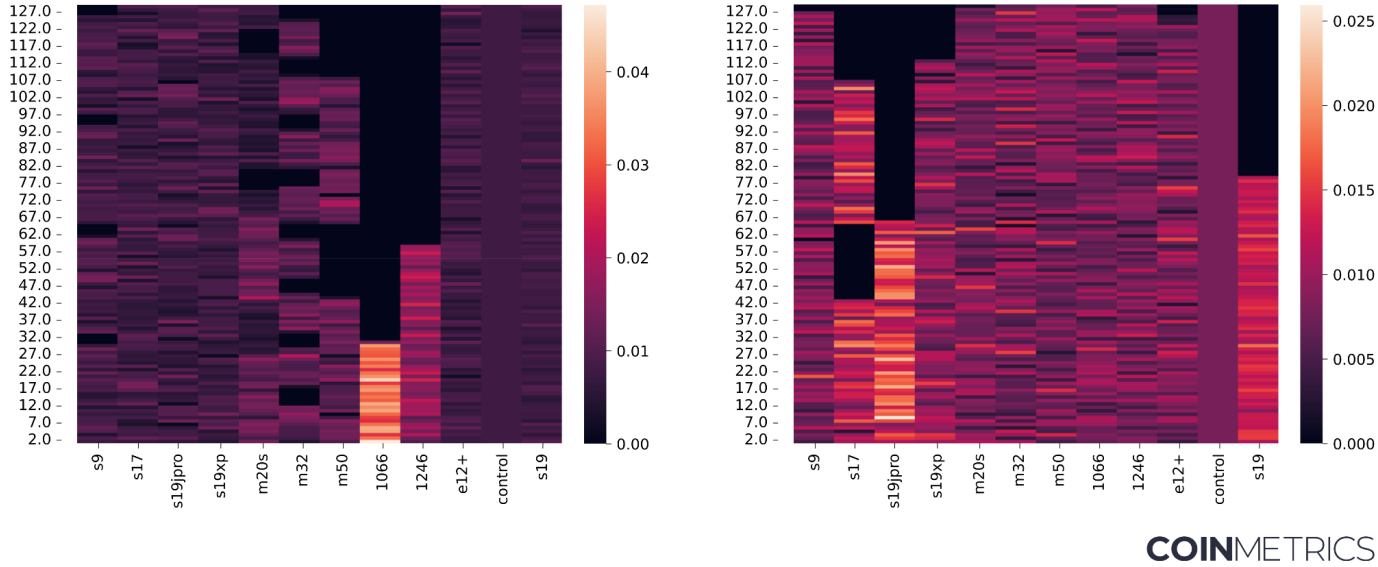


COINMETRICS

Remember that the nonce is a 32-bit value. Each bit is binary – in each of the 32 bits that make up a nonce, there can only be two values, either 0 or 1. This makes the universe of all possible nonces equal to 2^32, which is a little less than 4.3 billion. The magnitude of possible nonces ends up complicating the process of pattern identification. For this reason, our methodology does not look for patterns in the entire 32-bit space. Instead, we limit our search to the leading 7 bits of the nonce in both big and little-endian space. This is equivalent to only evaluating the beginning of the nonce using both little-endian and big-endian notation.

Doing so drastically reduces the universe of possible nonce values per dimension to 2^7 or 128, which is a much more reasonable search field. You can think of this reduced search universe as 128 distinct groups or *buckets*. Every time we produce a nonce using a specific ASIC model, the 7 leading bits of that nonce will fall into 1 of the 128 distinct buckets. Through this classification method, we have found that the very same ASIC models tend to fall into the same bucket. Thus, there is a high probability that nonces produced with that pattern were produced by the identified ASIC. Our hypothesis is that this pattern is intrinsic to the chip design used in that ASIC model. As you can see below, the patterns in each model is distinct, either in big-endian or in little-endian.

FIG 3: Buckets of ASICs in Big-Endian *(left)* and Little-Endian *(right)*



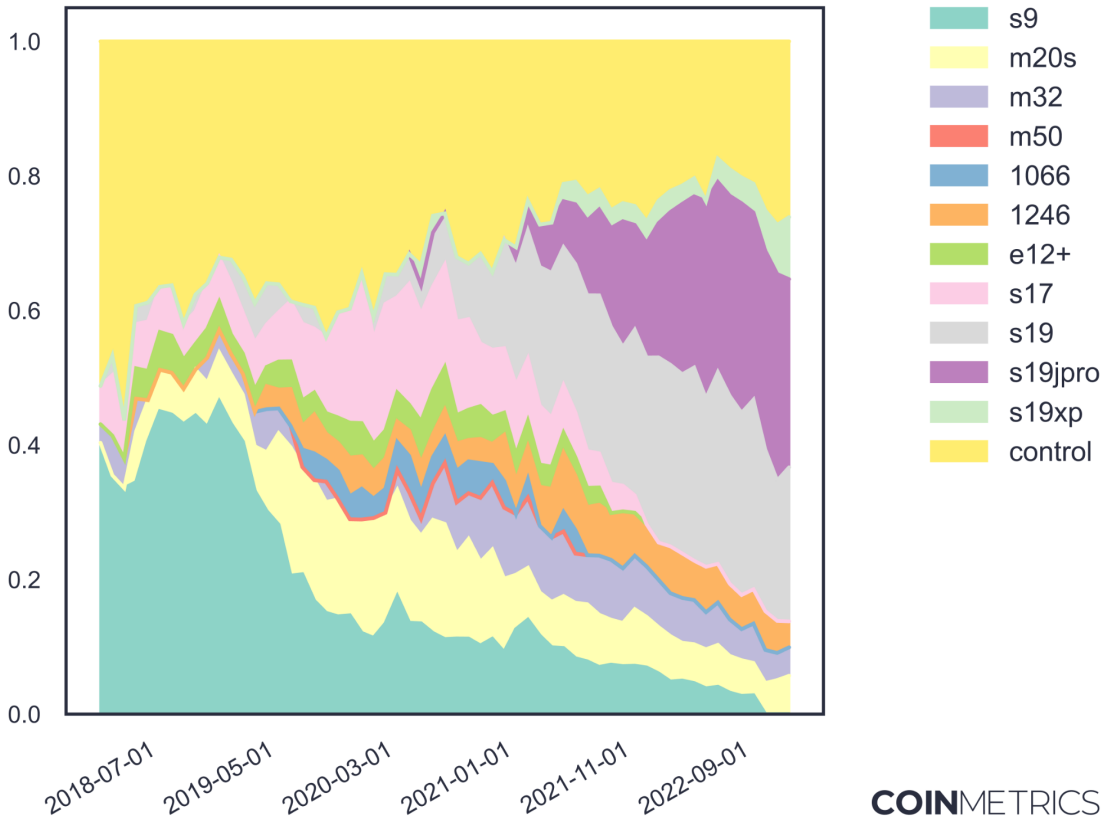
These patterns are generated at a hardware level by the mining chip itself. Typically, multiple models will be released with the same chip, often with different performance or form factors (e.g. physical size, shape, and configuration). For example, the S9, S9i, T9, and T9+ all use the same BM1387B chip but are sold as different models. While our estimates of machine hashrate will technically track chip dominance rather than rig dominance, we'll refer to the fingerprints by the name of one of their more common models instead of the less-frequently-referenced chip name throughout this article for convenience.

From Nonces to Model Hashrate

With ASIC fingerprints in hand, we're now tasked with mapping them to percentages of network hashrate. We'll use a technique called expectation-maximization (EM) to do this. Without going into too many of the details (see appendix for a formal specification of the analysis), we can gain an intuition for EM with a rough overview of the workflow. We start with an initial guess of the distribution of ASICs, then at each time period, we look at the data and update our estimates based on the likelihood we would see this data given our current estimate. With each iteration, we eventually converge to a point within our pre-set tolerance level (0.1%) or hit the maximum number of iterations of the algorithm.

To estimate the likelihood that each model could have produced the nonce, we perform the same conversion and binning procedure as we did with the nonces from the share submissions. We then calculate likelihoods in big- and little-endian space and multiply them to reach our result, implicitly assuming independence of the dimensions–which isn't strictly true but is sufficient for our purposes.

FIG 4: Lower Bound Dominance (%) of ASIC Models with Control Group

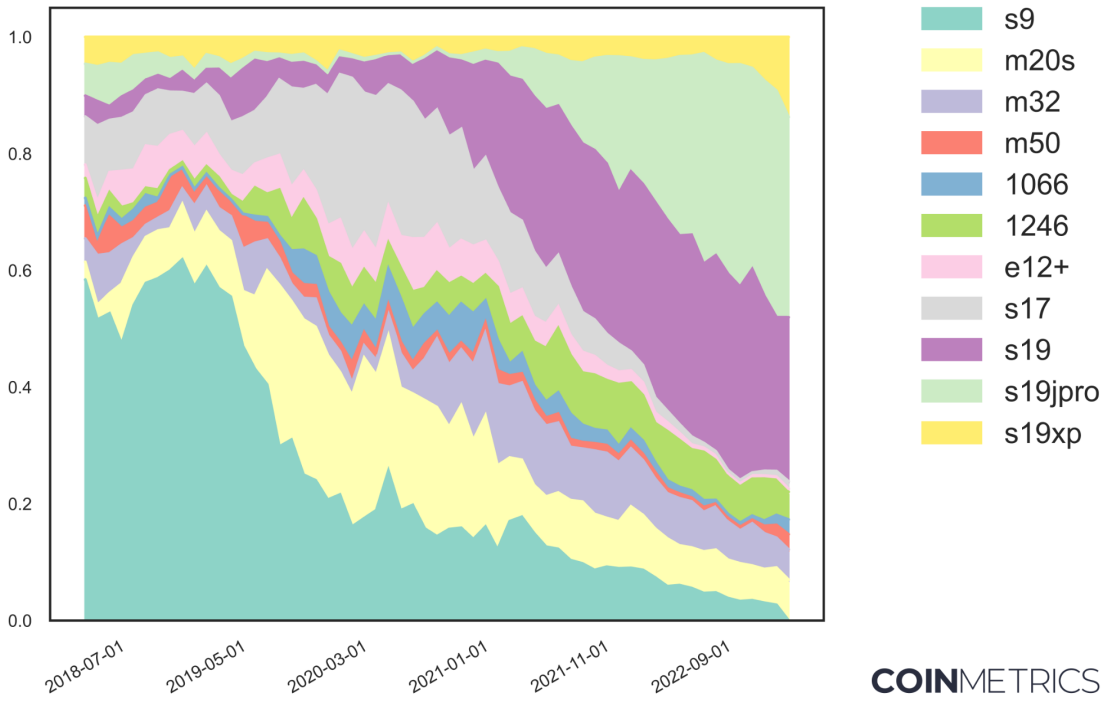


As you can see, throughout our calculations, we also include a random uniform control term, whose prior probability is set to a somewhat arbitrary value of 20% and resets at the start of every time period. By attaching to noise in the nonce distribution, this term acts as a proxy for confidence in the estimate: a large value indicates that our coverage isn't representative, while a smaller value means that most hardware on the network is accounted for in our model. Given that the percentage of hashrate attributed to the control term generally trends down over time, we can see that most hardware models in the present day are accounted for in our model.

We'll use this to set our confidence threshold dynamically, at 2.5% of 100% minus the control term, zeroing out values that are smaller than this in our postprocessing and moving them into the control term. Ultimately, this gives us a lower bound estimate for the percentage of hashrate by hardware type over time.

Keeping only the zeroed-out values in the control term as a proxy for unknown hardware, removing the rest of the control, and normalizing the results gives us our best-guess estimate for network hashrate by model:

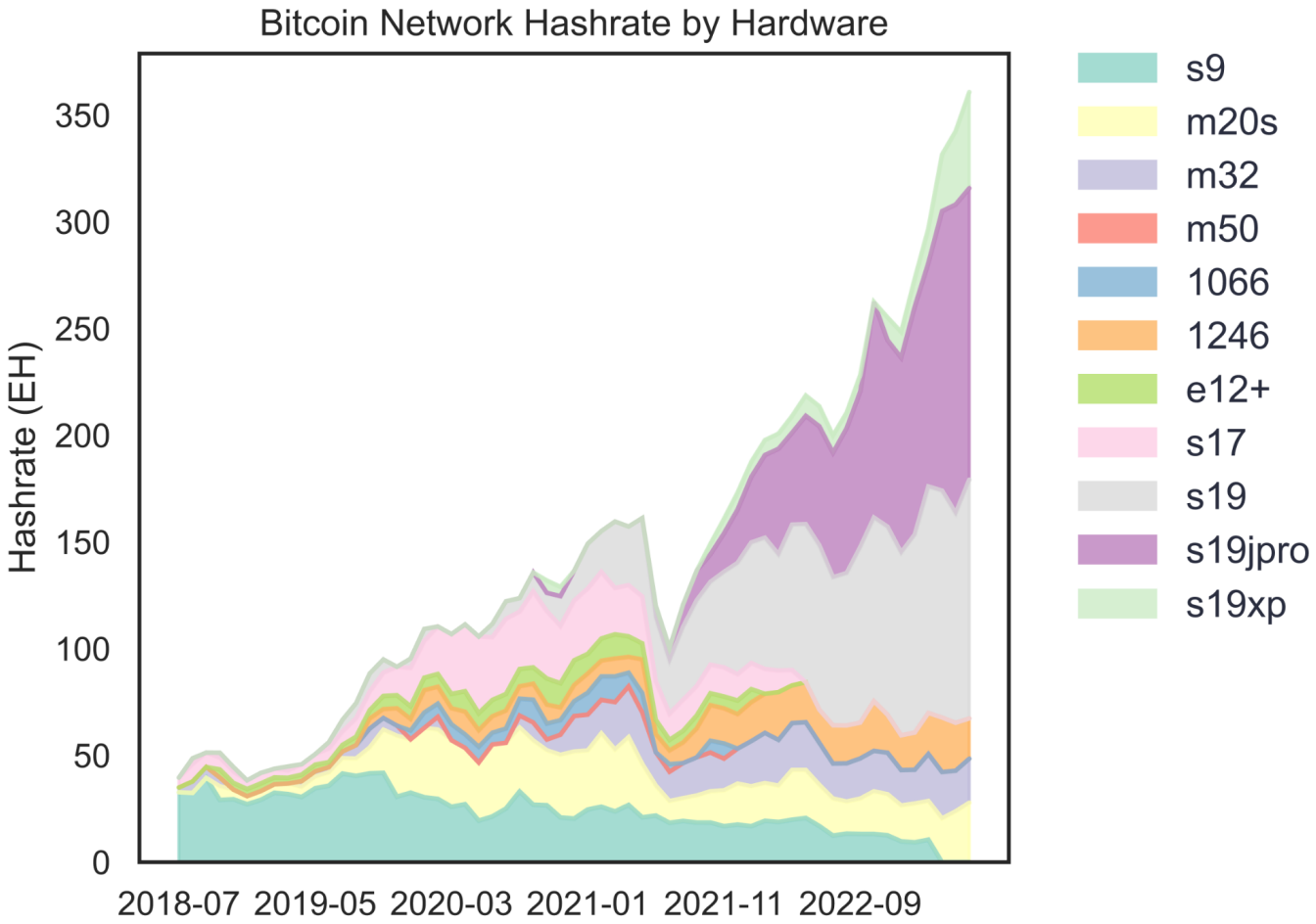
FIG 5: Adjusted Dominance (%) of ASIC Models



The distribution of models over time shows several interesting trends: Bitmain's sustained dominance in the hardware market is the first thing to stick out. The surprising resilience of the S9, which officially premiered in 2016 and only recently dropped below the threshold for detectability is also notable. This can be contrasted with the rapid decline of the S17, a notoriously non-performant machine with a large fraction of malfunctioning units due to a design flaw in its control board.

Also of note is the rapid rise in dominance of the S19jPro and related models, in contrast to the anemic initial growth of newer models like the S19XP (which was overpriced for an extended period after release and has also suffered from design flaws) and the M50 in the ongoing bear market. While S19XP growth has ticked up lately, M50s remain below the tolerance level for detection. This plot can also be represented in terms of total hashrate, which makes the gradual attrition in S9 dominance even more visible.

FIG 6: Bitcoin Network Hashrate by Hardware Model



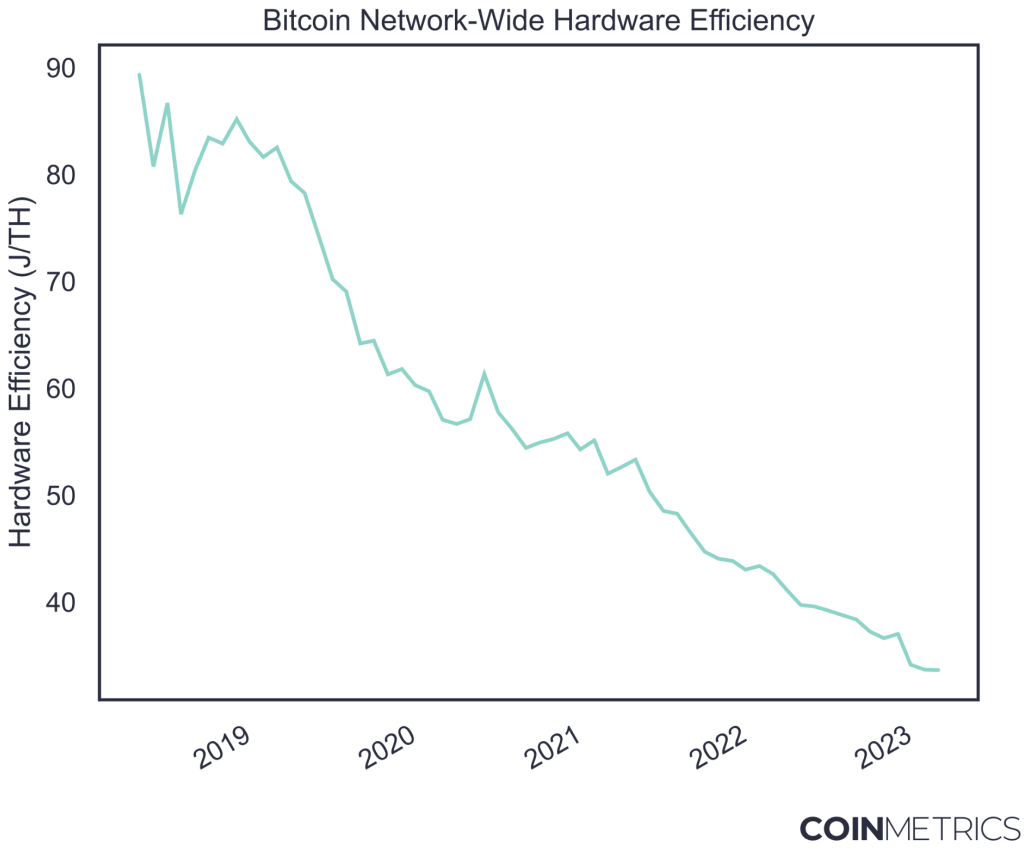
This metric is not without its flaws: in particular, it's unreliable for models with very little hashrate online, since these will mine fewer blocks and as a result won't leave their characteristic fingerprints

on the network as visibly. It's also possible for it to pick up signals that aren't necessarily there, for example from the M50 prior to its release. Finally, its model coverage isn't perfect, especially in the distant past. Still, its results are broadly in line with analyst estimates, and it offers the first continuous, comprehensive estimate for the network's mining hardware distribution.

Toward a Network-Wide Average Efficiency

Bitcoin mining rigs vary substantially in the amount of hashpower they produce per watt of electricity: as a general rule, newer models are more efficient than older ones. With our estimates for the occurrence of each hardware model, it's straight-forward to estimate a network-wide average efficiency by multiplying the distribution by manufacturer-specified hardware performance, giving us an average value in J/TH.

FIG 7: Network-Wide Efficiency



A miner's location on the efficiency curve is a major component in determining their profitability under volatile market conditions. Having this information allows miners to make better decisions in expanding, contracting, or relocating their operations. What's more, without a good understanding of what kinds of machines are running on the network and in what proportion, accurately estimating aggregate energy consumption from hashrate is impossible.

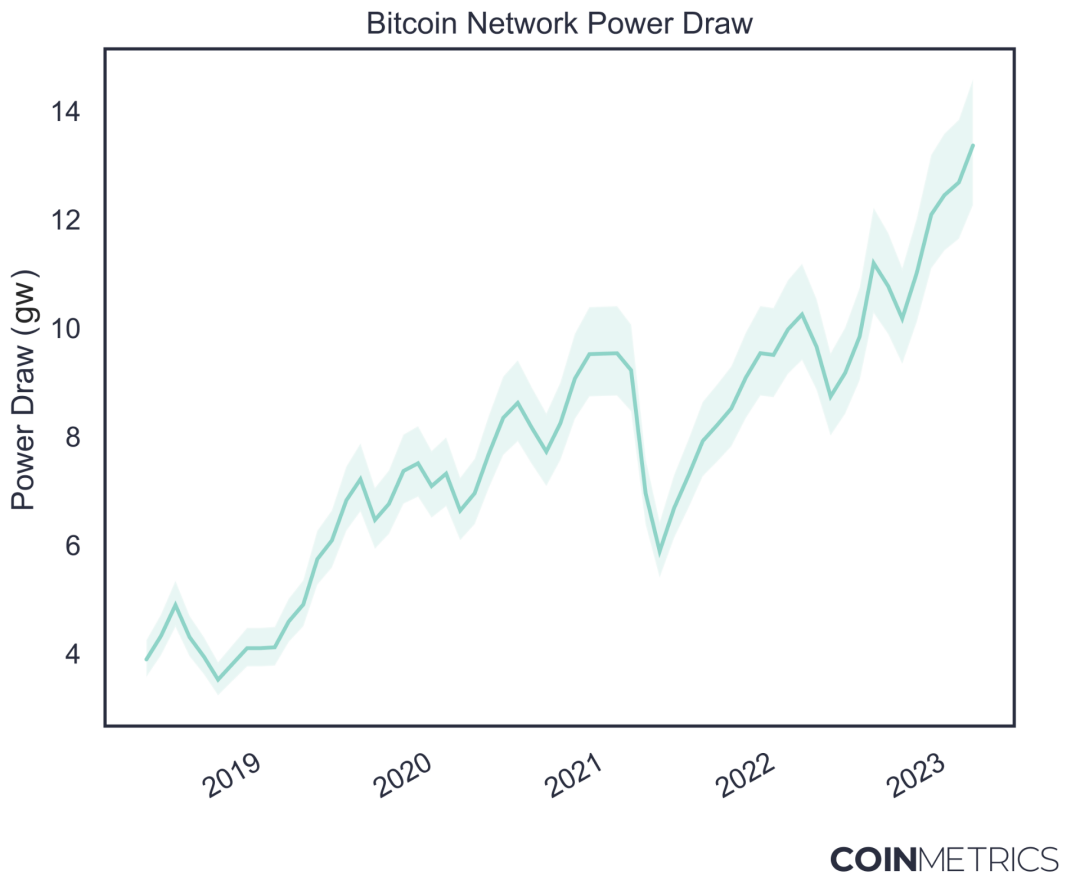
In addition to the limitations with our ASIC distribution methodology, our efficiency estimates are constrained in their accuracy by not knowing exactly the conditions under which the chips are hashing: for example, machines may be running custom firmware, or may be overclocked, underclocked, or immersed, all of which would affect efficiency. We also can't be sure of a specific machine's performance: models are typically sold under several different nameplate performances based on their empirical performance, for example, S9s were sold in a range from 11.5TH to 14TH. Finally, we can't distinguish between related models like M30Ss and M32s, which use the same chip but have different performance profiles.

Estimating Electricity Consumption

While various teams have attempted to determine Bitcoin's energy use, with the Cambridge Center for Alternative Finance's approach widely deemed the gold standard, network-wide miner efficiency was until now a bottleneck on accurate estimation. Cambridge's power draw estimate is based on evenly weighting marginally-profitable hardware models released in the last five years. This is a relatively blunt tool: it's deeply exposed to temporary fluctuations in bitcoin's price, relies on the assumption of a uniform network-wide electricity price, and isn't particularly accurate during bull markets. Still, the figures produced by Cambridge were groundbreaking, and the methodology presented in this report is at its core a refinement of this existing work.

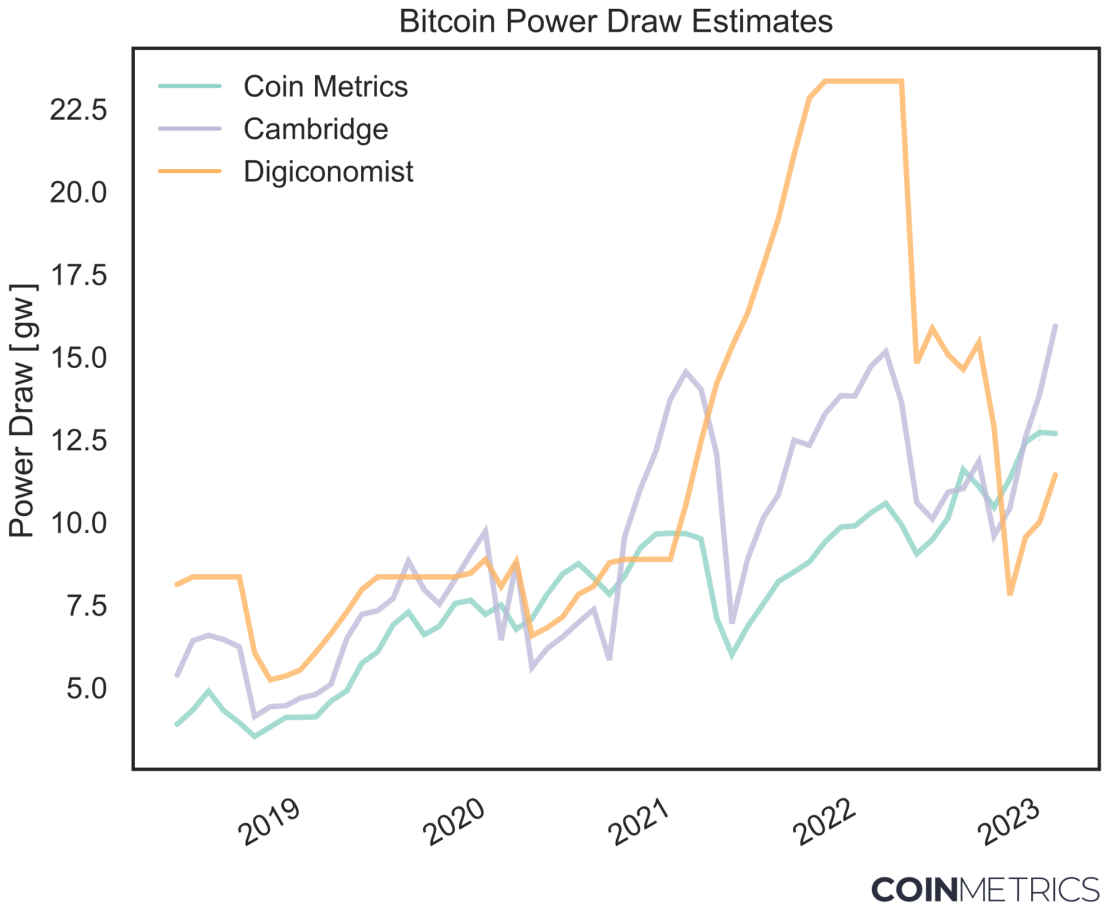
To derive an energy consumption estimate, we simply multiply the network-wide average efficiency by network hashrate. We also include Power Usage Effectiveness (PUE) factors of 1.20 (upper bound), 1.10 (average case) and 1.01 (lower bound) in our calculations, matching the figures used by Cambridge. We estimate the network's power draw at roughly 13.4 GW, or about 16% less than Cambridge's estimate of 15.9 GW for the month of May 2023, the latest data available.

FIG 8: Bitcoin Electricity Consumption with Efficiency Overlay



This newer methodology has several advantages over the existing body of research, including the Cambridge Bitcoin Electricity Consumption Index. Because these new results include a model hardware composition, the resulting energy consumption figures are also substantially more accurate than existing estimates. The model is the first that doesn't factor in the price of bitcoin as an input, which renders it much less volatile; it also doesn't require a network-wide energy price as an input, which is an oversimplification and can be difficult to estimate.

FIG 9: Comparison of Bitcoin Electricity Consumption Estimates

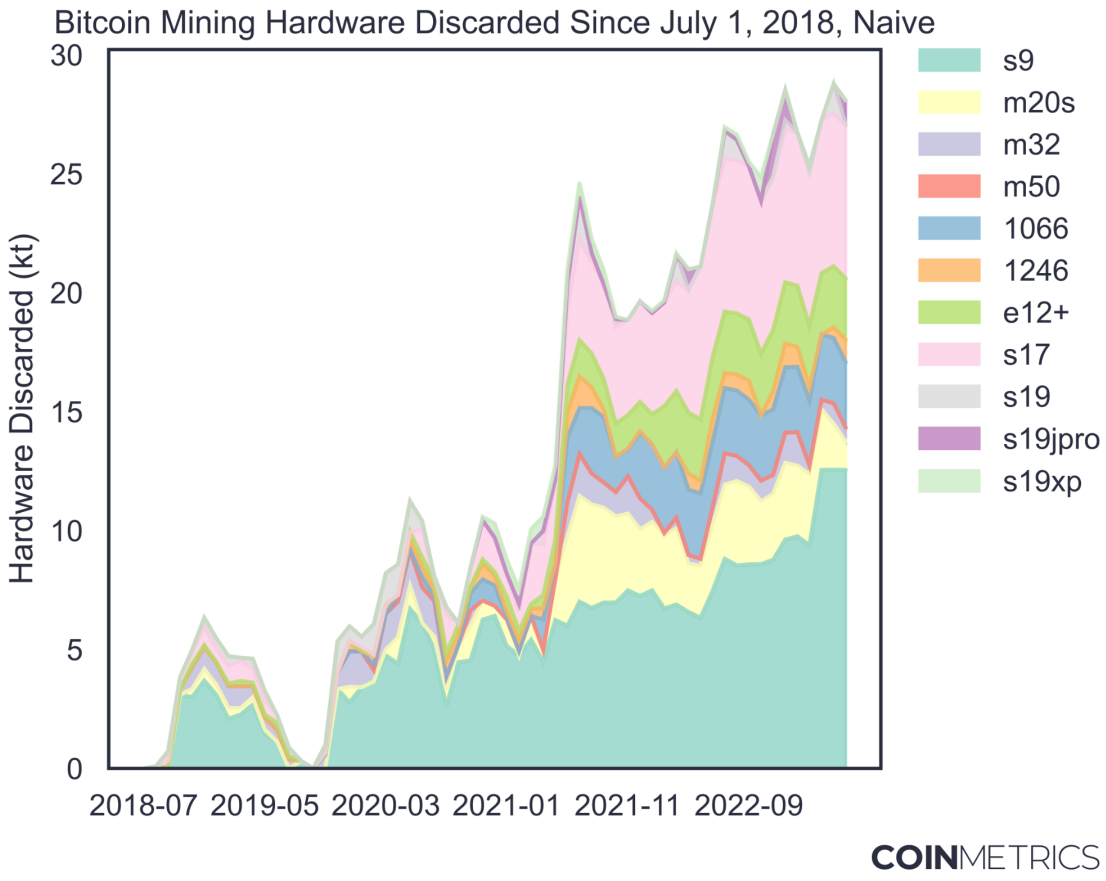


The limitations of our hardware distribution and efficiency estimates carry over into this model, and the PUE bounds are quite wide. Still, it provides a significant improvement over the current state of the art, especially during bull markets, when price can dislocate significantly from actual network conditions.

E-Waste Estimation

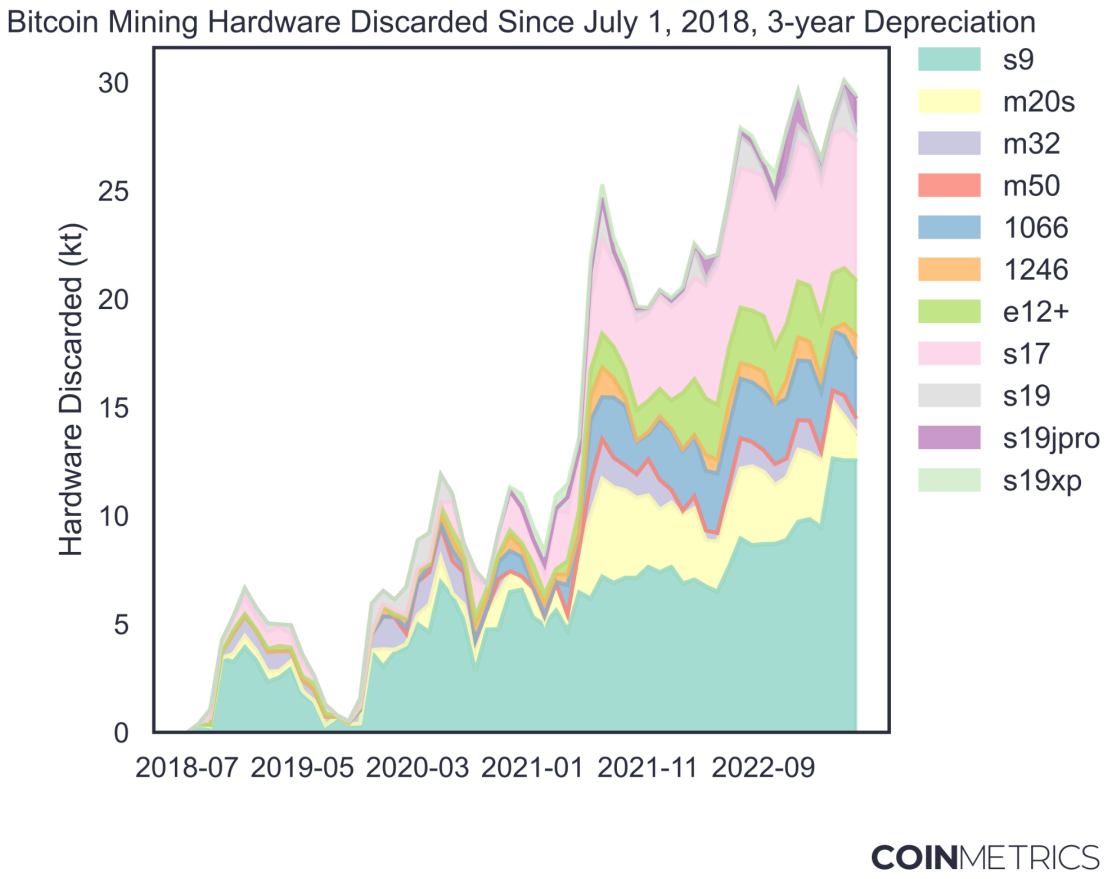
Another conversation point, though without much high-quality research behind it, has been the e-waste generated by Bitcoin mining (while most of the body of a mining rig is aluminum, and therefore is recyclable and doesn't actually qualify as e-waste, we'll adopt that terminology since it's become the standard in the industry). With our hardware distribution, we can come up with a simple heuristic to estimate e-waste: after the then-all-time-high hashrate produced by a certain type of hardware, assume every drop is caused by machines falling permanently offline; then, compare this figure to the manufacturer-specified hashrate and weight per machine to see how much e-waste has been produced:

FIG 10: Assessment of ASICs Discarded



In some regards, this methodology overestimates e-waste, since mining hardware can and does come back online when market conditions have improved or after exogenous shocks like the Chinese mining ban have resolved. That said, this naive methodology does not account for internal churn: for example, one S19jPro could fail, be thrown out, and be replaced by another. However, even adopting an aggressive 2.78% monthly churn in online hardware corresponding to a 3-year depreciation schedule, the plot looks very similar:

FIG 11: Assessment of ASICs Discarded, 3-Year Depreciation



The major blind spot with this methodology, in addition to the shortcomings in determining the network hardware distribution, is that some machines are manufactured but never plugged in in the first place. Estimating this value would likely require production figures from the manufacturers themselves.

Conclusion

In this post, we introduced a new methodology to fingerprint the predominance of each of the major ASICs in Bitcoin over time. We hope this data can shed some light on widely-debated topics within Bitcoin, such as its electricity consumption, resulting carbon footprint, and e-waste production. Given the importance of ASICs in the security of Bitcoin, we also hope this data can help contextualize security in Bitcoin over time. Finally, we hope this data set helps inform miners about where they sit on the efficiency curve, which can in turn function as a benchmark.

Acknowledgments

We would like to thank Celia Wan, Will Foxley, Edwan Chiam, Elijah Hendrickson, and Nic Carter for their contributions to the content and/or data presented in this report.

APPENDIX

Technical Appendix 1

Formal Specification of Bitcoin Nonce Analysis

Alex R. Mead

Introduction

This Appendix attempts to layout a formal specification for the complete process of ASIC proportion estimation presented here.

The appendix is divided into several parts, including: characterization of each ASIC used in mining (Section 1 and 3), characterization of the Bitcoin blockchain data (Section 2), and a detailed explanation of the Expectation-Maximization (EM) Algorithm (Section 4).

1 Application Specific Integrated Circuits (ASICs)

The ASIC chips[1] considered in this analysis are represented by the set,

$$ASICs = \{s9,\ m20s,\ m32,\ m50,\ 1066,\ 1246,\ e12+,\ s17,\ s19,\ s19jpro,\ s19xp\}. \tag{1}$$

A single ASIC is referred to as $asic \in ASICs$, for example $asic = s9$. For this analysis, each $asic \in ASICs$ has corresponding manufacturer specifications, expressed here as the function,

$$asic\_specs : ASICs \longrightarrow specs, \tag{2}$$

where,

specs = "key:value set" = {
"power draw" : [Joules @ 100% operation],
"hash rate" : [terahash],
"release date" : [YYYY-MM-DD],
"weight" : [kg]
"efficiency" : [Joules/terahash]
"weight efficiency" : [kg/terahash] }.

2 Bitcoin Data

The Bitcoin chain with the "most work" is referred to here as ₿. However, the analysis is done on a continuous subset, $\mathbf{B} \subset$ ₿, from height $\mathbf{B}_{min} = 529,967$ to $\mathbf{B}_{max} = 792,268$ [July 1, 2018 - May 31, 2023].

[1] ASIC chips are often refereed to as 'miners'.

For this analysis, an individual block, $b \in \mathbf{B}$ is a 4-tuple defined as,

$$b = (height,\ timestamp,\ difficulty,\ nonce), \tag{3}$$

where,

height	=	"standard Bitcoin notion of block height from genesis"
timestamp	=	"UTC consensus timestamp"
difficulty	=	"network mining difficulty, updated every 2016 blocks"
nonce	=	"4-byte (32-bit) value, the solution of mining."

Examining the *nonce*, we partition it into its base four bytes, where,

$$nonce = [\ byte_0 \mid byte_1 \mid byte_2 \mid byte_3\ ]. \tag{4}$$

with,

$byte_0$	=	"the lowest addressed byte of nonce in memory"
	and	
$byte_3$	=	"the highest addressed byte of nonce in memory"

with the notational convenience of $nonce_0 = byte_0,\ \ldots,\ nonce_3 = byte_3$.

2.1 Key Nonce Observation

The nonce can be thought of as a *random variable* with a corresponding probability density function (pdf) of the form,

$$pdf_{nonce} : nonce \times ASICs \longrightarrow [0,\ 1], \tag{5}$$

written equivalently in standardized stochastic nomenclature as,

$$\mathbb{P}(nonce \mid asic) = \text{"}given\ an\ asic\ made\ the\ nonce\ it\ did\ so\ with\ this\ probability\text{"}. \tag{6}$$

A key observation of this work is the nonce can be separated into two, *independent* random variables. These random variables are $byte_0$ and $byte_3$ of the nonce itself, expressed as $nonce_0$ and $nonce_3$.

One can verify this independence themselves by examining $\ddot{\mathbb{B}}$ and plotting $nonce_0$ versus $nonce_3$ on an X-Y plane. The points form a non-patterned cloud, exactly as expected for independent variables. Further, the authors conducted several statistical analysis, including correlation and Pearson tests, to examine the independence claim more rigorously. Further validation of this claim can be provided on request.

Thus, for each nonce, we have two random variables and their corresponding marginal pdf's can be expressed as,

$$\mathbb{P}_{nonce_0}(nonce_0 \mid asic)\ \ and\ \ \mathbb{P}_{nonce_3}(nonce_3 \mid asic) \tag{7}$$

With $nonce_0$ and $nonce_3$ being independent variables, we can now get the joint density function via the well known general relationship of pdf's (i.e. $\mathbb{P}(A \cap B) = \mathbb{P}(A)\mathbb{P}(B)$) as,

$$\mathbb{P}(nonce_0, nonce_3 \mid asic) = \mathbb{P}_{nonce_0}(nonce_0 \mid asic)\mathbb{P}_{nonce_3}(nonce_3 \mid asic). \tag{8}$$

2.2 Practical Nonce Processing

Practically speaking, $nonce_0$ and $nonce_3$ can be "extracted" from the observed *nonce* by using a simple two step procedure. First, using both a big and little endian encoding, the uint32 representation of the 4-byte nonce is calculated, $nonce^{be},\ nonce^{le} \in [0,\ 4,294,967,295]$. Next, a "binning" process is used to group the respective integers into 128 evenly spaced "buckets." That is, for some $u \in uint32$, buckets are defined as,

$$\begin{aligned}\text{buckets} = \{ & \\ & \text{bucket}_0 = \{u \mid u \in [0,\ \tfrac{2^{32}}{128})\}, \\ & \text{bucket}_1 = \{u \mid u \in [\tfrac{2^{32}}{128},\ 2\tfrac{2^{32}}{128})\}, \\ & \dots \\ & \text{bucket}_{127} = \{u \mid u \in [127\tfrac{2^{32}}{128},\ 2^{32})\} \\ \}&\end{aligned}$$

and the random variable of interest is "compressed" to an integer with value $0, \dots, 127$.
Depending on the encoding of big or little endian for $u \in uint32$, two sets of buckets exist,

$$buckets^{be} \ \ and \ \ bucket^{le}. \tag{9}$$

Now, we introduce a function *bin* to return the name of a bucket given a *nonce* as,

$$bin : nonce \times \{be, le\} \longrightarrow buckets_i^e, \tag{10}$$

where $e \in \{be, le\}, i \in 0, \dots, 127$ and $buckets_i^e$ is not the bucket itself, but the bucket's unique identifier. This function can can then be used to organize a set of nonces into their respective *buckets* or used with a single nonce to determine it's $nonce_0$ and $nonce_3$ "compressed" integer representation by simply looking at the integer of the bucket this function returns for a given *nonce* and its encoding $e \in \{be, le\}$.

Careful examination of $buckets^{be}$ for a *nonce* derived with a big endian encoding will in fact reveal itself as $nonce_0$ from the observed *nonce*. Similarly, for a little endian encoding for a *nonce*, $buckets^{le}$ is in fact $nonce_3$.

Note: the attuned reader will observe the buckets only use bits 1-7 of $nonce_0$ and $nonce_3$, not the full bits 0-7.

It should also be noted: $nonce_0$ and $nonce_3$ can be accessed in an equally valid manner by bit-masking and bit-shifting. The presented big and little endian interpretation is provided for both historical reasons (i.e. how the author's "discovered" the nonce patterns) and because it can be easier for some readers to think in integers and binning than bits and bytes.

With the nonce formally described and interpreted as two independent random variables, the next step is to express their respective pdf's based on measured data.

3 ASICs Training Data

For each $asic \in ASICs$, a large set of *nonces* were generated in a simulated mining environment[2]. This process created a set of labeled *nonces* for each *asic* which are now used as *training data* for the estimation of the respective probability distribution functions (pdf's). These sets of nonces can be described $\forall\ asic \in AISCs$ as,

$$nonces_{asic} = \{nonce \mid \text{"}nonce\ was\ made\ by\ asic''\}. \tag{11}$$

The process of building the respective pdfs starts by using the helper function 10 defined above, these sets of *nonces* are "binned" to create their respective sets of "buckets," $buckets_{asic}^{be}$ and $buckets_{asic}^{le}$. Recall from above, the sets of "binned" nonces in reality represent the random variable of the first and

[2]Technically speaking this was a "live" mining operation in the ₿ mining network.

fourth byte (i.e. $nonce_0$, $nonce_3$) produced by a respective miner and are now in a *labeled* set.

Next, $bucket^{be}_{asic}$ and $bucket^{le}_{asic}$ are used to create the marginal pdf's defined above in equation 7. Recall, big endian (be) is $nonce_0$ and little endian (le) is $nonce_3$. This means, the marginal pdf for $nonce_0$ which can be represented by some "compressed" integer $j \in 0, \ldots, 127$, for a given $asic \in ASICs$ can be defined as,

$$\mathbb{P}^{be}_{asic}(j) = \frac{|buckets^{be}_{asic,j}|}{\sum_{i=0}^{127} |buckets^{be}_{asic,i}|}, \tag{12}$$

where $buckets^{be}_{asic,i}$, $buckets^{be}_{asic,j}$ represents $bucket_i$, $bucket_j \in bucket^{be}_{asic}$. Intuitively, this is the proportion of times a specific $asic \in ASICs$ made a $nonce$ with $nonce_0 = j$ divided by the total number of nonces made by that $asic$ in the training data. Thus, the marginal pdf for each asic is derived using a simple proportion of each type of nonce as compared to the total number of nonces made in the training data.

This process is repeated for both $nonce_0$ and $nonce_3$ (i.e. big and little endian "binning" respectfully), as well as for each $asic$. Taken in total, this produces a set of two marginal pdfs for each ASIC considered, which we refer to moving forwards as,

$$\mathbb{P}^{be}_{asic}(nonce^{be}) \;\; and \;\; \mathbb{P}^{le}_{asic}(nonce^{le}) \tag{13}$$

where $asic \in ASICs$.

3.1 Control Group

For reasons that will become clear in the analysis presented below, a final pdf is added as a "control" term. This control term is a uniform random variable, as opposed to the previous empirically defined pdfs. Intuitively this can be thought of as a "collector" for any blocks not produced by an $asic \in ASICs$ which has been measured for this analysis.

Practically, this results in the expansion of the set $ASICs$ to include $control$ and the definition of two pdfs, $\mathbb{P}^{be}_{control}$ and $\mathbb{P}^{le}_{control}$. Due to their uniformity, $\forall$ $nonce$,

$$\mathbb{P}^{be}_{control}(nonce^{be}) \;=\; \mathbb{P}^{le}_{control}(nonce^{le}) \;=\; \frac{1}{128} \tag{14}$$

4 Expectation-Maximization Algorithm

The details of the block data set partitioning and subsequent ASIC proportion estimation are described next.

4.1 Partitioning the Blockchain Data

To begin, the continuous subset $\mathbf{B} \subset \ddot{\mathbb{B}}$ is first partitioned into "windows" based on the calendar month of each block's respective UTC timestamp[3]. For this analysis, non-overlapping window lengths of one month are used, however, other window lengths are also valid. For $\mathbf{B}$, these windows can be expressed formally as,

$$\begin{aligned}
\text{windows} = \{ & \\
& \text{window}_{Jul2018} = \{\, \forall\, b \in \mathbf{B} \mid b.timestamp \in \text{`}July\;\; 2018\text{'} \,\}, \\
& \text{window}_{Aug2018} = \{\, \forall\, b \in \mathbf{B} \mid b.timestamp \in \text{`}August\;\; 2018\text{'} \,\}, \\
& \qquad \ldots \\
& \text{window}_{May2023} = \{\, \forall\, b \in \mathbf{B} \mid b.timestamp \in \text{`}May\;\; 2023\text{'} \,\} \\
\}, &
\end{aligned}$$

[3]See section 2 above for formal definitions of $\ddot{\mathbb{B}}$, $\mathbf{B} \subset \ddot{\mathbb{B}}$, and $b \in \mathbf{B}$.

where,

$$\bigcup_{windows} window = \mathbf{B} \quad and \quad \bigcap_{windows} window = \{\}. \tag{15}$$

Intuitively, this means we assume the proportion of ASIC chips used on the network does not change within a month, only between months. While this is clearly not true in general, the difficulty changes approximately every 2 weeks - and is still "agile" enough to match the hashrate network changes - which gives us faith the one month window is appropriate [4].

4.2 Running the Algorithm

With all of the above machinery in place (i.e. pdfs in Section 3 and now the partitioned blockchain data), the *asic* proportionality estimation can finally be explained.

4.2.1 Initial Conditions

The first step in the EM Algorithm is to provide an initial "guess" as to the distribution we are estimating. That is, what proportion of the Bitcoin Mining Network hashrate is currently made up by each $asic \in ASICs$. These values are encoded in a vector, $weights \in [0,\ 1]^{|ASICs|}$, such that,

$$\sum_{w \in weights} w = 1 \tag{16}$$

Further algorithm parameters are also specified, including:

depth	=	"maximum iterations used in estimation-maximization"	=	30
delta	=	"percentage change threshold for convergence"	=	0.1 %

4.2.2 Expectation-Maximization Algorithm

Changing gears from set-theory formalisms, the EM-Algorithm will be presented using a Pythonic syntax. The goal here is to explicitly define the mathematics used in the estimate.

We begin by looping over each $window \in windows$,

Looping Over Windows

```
windows = [window_Jul2018, window_Aug2018, ..., window_May2023]
asic_estimates = [] # Container for stored estimates
weight_initial = [0.69, 0.01,..., 0.01, 0.2] # Initial Guess

for window in windows:
    # Estimate for the current window.
    weight_estimate = em_algorithm(weight_initial, window, depth=30, delta=0.001)

    # Filter weight estimation.
    weight_final = filter_weights(window, weight_estimate, weight_initial, asic_spec)

    # Store estimate for post processing.
    asic_estimates.append(weight_final)

    # Use current window as initial condition of next window.
    weight_initial = weight_final
```

[4] To jump ahead, one month window results match intuition as well as other estimates within reason. Lending more evidence this assumption is valid.

With the high level iterative process shown above, the actual estimation process is show here:

Expectation-Maximization Within Each Window

```
def em_algorithm(weight_initial, window, depth, delta):
    """Recursive EM Implementation."""
    # Marginal Probabilities
    pdf_be_mar = [ [ asic.pdf_be(block.nonce) for asic in ASICs ] \
                                                    for block in window ]
    pdf_le_mar = [ [ asic.pdf_le(block.nonce) for asic in ASICs ] \
                                                    for block in window ]

    # Joint Probabilities
    pdf_jnt = [ [pdf_be_blk_asic * pdf_le_mar[idx_pdf_be_blk][idx_pdf_be_blk_asic] \
                for idx_pdf_be_blk_asic, pdf_be_blk_asic in enumerate(pdf_be_blk)] \
                    for idx_pdf_be_blk, pdf_be_blk in enumerate(pdf_be_mar)]
    # Expectation
    observed = []
    for block_est in pdf_jnt:
        for idx, asic_est in enumerate(block_est):
            asic_est = (asic_est * weight_initial[idx]) \
                                / dot(block_est, weight_initial) # Dot product
        observed.append(block_est)

    # Weight by block difficulty
    for idx, block_est in enumerate(observed):
        difficulty = window[idx].difficulty
        for asic_est in block_est:
            asic_est = asic_est * difficulty

    # Average each block in the window
    average_asic_est = zeros(len(ASICs))
    for block_est in observed:
        for idx, asic_est in enumerate(block_est):
            average_asic_est[idx] += asic_est
    for avg in average_asic_est:
        avg = avg / len(observed)

    # Normalize estimate
    weight = [ avg / sum(average_asic_est) for avg in average_asic_est ]

    # Recursion -or- Exit conditions
    if depth == 0 or vector_converge(weight_initial, weight, delta):
        return weight
    else:
        return em_algorithm(weight, window, depth=depth-1, delta=delta)
```

Finally, two functions used within EM are presented:
First, the convergence criteria for the estimation vector, second, the "re-set" procedure for each *window* estimate:

Converge Criteria

```
def vector_converge(weight_initial, weight_new, delta):
    """Convergence check on ASIC proportional estimate."""
    converged = True
    for idx, weight in weight_initial:
        if abs(weight - weight_new[idx]) > delta:
            converged = False
    return converged
```

Estimation Re-Set

```
def filter_weights(window, weight_estimate, weight_initial, asic_spec):
    """Adjust estimates after EM convergence for next round"""
    weight_final = zeros(len(ASICs))

    # Reset control weight for every window
    weight_estimate['control'] = 0
    weight_final['control'] = weight_initial['control']

    # If asic not publically released for more than 1 year
    for asic in ASICs:
        if asic_spec('release_data') > window.month - 1_year:
            weight_estimate[asic] = 0 # asics to re-set
            weight_final[asic] = weight_initial[asic] # value asics are re-set to.

    # Normalize weights within vector, normalize vector between vectors, then
    addition.
    weight_final = normalize(weight_estimate) * (1 - sum(weight_final)) + weight_final

    # Return value for next round.
    return weight_final
```

4.2.3 Examples of Variable Values

For the reader's convenience, some variables presented in the algorithm are expressed in a more clear syntax below.

Joint Probability Function for Nonces

$$\begin{aligned} \forall\, window \;\in windows = \{window_{Jul2018}, \ldots, window_{May2023}\},\ and \\ \forall\, blk \;\in\; window = \{blk_0, \ldots, blk_n\}, \\ where\ blk_i.n\ is\ the\ nonce\ value \end{aligned}$$

$$pdf\_jnt = \left[\begin{bmatrix} \mathbb{P}^{be}_{s9}(blk_0.n^{be})\mathbb{P}^{le}_{s9}(blk_0.n^{le}) \\ \mathbb{P}^{be}_{m20s}(blk_0.n^{be})\mathbb{P}^{le}_{m20s}(blk_0.n^{le}) \\ \cdots \\ \mathbb{P}^{be}_{control}(blk_0.n^{be})\mathbb{P}^{le}_{control}(blk_0.n^{le}) \end{bmatrix}, \quad \cdots, \quad \begin{bmatrix} \mathbb{P}^{be}_{s9}(blk_n.n^{be})\mathbb{P}^{le}_{s9}(blk_n.n^{le}) \\ \mathbb{P}^{be}_{m20s}(blk_n.n^{be})\mathbb{P}^{le}_{m20s}(blk_n.n^{le}) \\ \cdots \\ \mathbb{P}^{be}_{control}(blk_n.n^{be})\mathbb{P}^{le}_{control}(blk_n.n^{le}) \end{bmatrix} \right]$$

Intuitively, each vector in pdf_jnt is a block and each row in the respective vectors are the *pdf* estimation for each $asic \in ASICs$, including the control term.

THE SIGNAL & THE NONCE

TRACING ASIC FINGERPRINTS TO RESHAPE OUR UNDERSTANDING OF BITCOIN MINING

By Karim Helmy, Lucas Nuzzi, Alex Mead, Kyle Waters, and the Coin Metrics Team

COINMETRICS

To view more from Coin Metrics Research go to coinmetrics.io/pubs or subscribe to State of the Network