Modeling, Identification and Control, Vol. 38, No. 4, 2017, pp. 157–168, ISSN 1890–1328

Nonlinear Feedback Control and Stability Analysis of a Proof-of-Work Blockchain

G. Hovland [1] J. Kučera [2]

[1] *Department of Engineering Sciences, University of Agder, N-4898 Grimstad, Norway*

[2] *Bismuth Foundation, Lead Developer, District Ostrava-City, Czech Republic*

Abstract

In this paper a novel feedback controller and stability analysis of a blockchain implementation is developed by using a control engineering perspective. The controller output equals the difficulty adjustment in the mining process while the feedback variable is the average block time over a certain time period. The computational power (hash rate) of the miners is considered a disturbance in the model. The developed controller is tested against a simulation model with constant disturbance, step and ramp responses as well as with a high-frequency sinusoidal disturbance. Stability and a fast response is demonstrated in all these cases with a controller which adjusts it's output at every new block. Finally the performance of the controller is implemented and demonstrated on a testnet with a constant hash rate as well as on the mainnet of a public open source blockchain project.

Keywords: Nonlinear, control system, blockchain, feedback, stability, disturbance rejection

1. Introduction

The enabling technology in a cryptocurrency is a blockchain. A blockchain works by linking cryptographic information from previous blocks with the current block. The most common method currently used in blockchain implementations to calculate the cryptographic puzzle linking the blocks is called proof-of-work (POW), first coined and formalized in Jakobsson and Juels (1999). In modern blockchain implementations POW is performed by computers called miners, typically utilizing massive parallelization in graphical processing units (GPUs) or Application-Specific Integrated Circuits (ASICs).

In most blockchain implementations it is desirable to have a near constant number of blocks generated per day to ensure timely execution of transactions. For example, if the goal is to generate blocks every 60 seconds, then the average number of blocks per day is 1440. The blockchain implementation must try to keep the number of blocks generated per day near constant even if there is large variation in computing power, also called hash rate, provided by the miners. Most POW blockchain implementations achieve a near constant generation of blocks per day by adjusting a parameter called "difficulty".

The difficulty relates to the complexity of the cryptographic puzzle which the miners have to solve. Hence, if the difficulty level increases proportionally with the combined hash rate of the miners, then the average block time should stay constant. Examples of recent algorithms for difficulty calculcations are described in Booth (2017) and Sechet (2017).

In Sechet (2017) some of the goals for the developed difficulty adjustment algorithm are stated as: 1) avoid sudden changes in difficulty when hash rate is fairly stable, 2) adjust difficulty rapidly when hash rate changes rapidly and 3) avoid oscillations from feedback between hash rate and difficulty. The algorithm in Sechet (2017) calculates an estimated hash rate, and then uses that as the basis of calculating a target. Booth (2017)

doi:10.4173/mic.2017.4.1

presents an alternative algorithm where the difficulty starts with the difficulty target of the previous block, and then "nudges" it up or down, depending on the observed timestamps of past blocks. In this way the algorithm in Booth (2017) acts as a "feedback" mechanism.

Even though developers of recent difficulty adjustment algorithms use terms such as stability, oscillations, rapid changes and feedback there are currently, to the authors knowledge despite an extensive literature review, no publications available adressing the blockchain difficulty adjustment problem from a feedback control engineering perspective.

In this paper a novel nonlinear feedback controller and blockchain stability analysis is presented. The controller is tested and the performance is demonstrated in simulations as well as in both the testnet and mainnet of a real blockchain implementation. The selected blockchain implementation for the developed controller is the Bismuth project programmed in Python. The open source code of the Bismuth project is available at Kučera (2017).

The approach taken in this paper to develop a blockchain difficulty adjustment controller and to study the stability of the closed-loop system from a control engineering perspective may become a standard approach in the future.

2. Bismuth Blockchain Overview

Fig. 1 shows an overview of the proof-of-work algorithm used by the Bismuth blockchain. The sha224 cryptographic hash function is used. For a description of this function, see for example Penard and van Werkhoven (2007). A hash function is a mathematical algorithm which maps data of any length to a bit string of a fixed size. A hash function is also designed to be a one-way function, ie. a function which is infeasible to invert.

The algorithm in Fig. 1 makes use of nonces, see for example Rogaway (2004). In cryptography, a nonce is a pseudo-random number used only once. Pseudo-random nonce generators are often seeded by computers (milli- or nanosecond) clocks to make the generated nonces different on two or more computers running the same code. The proof-of-work algorithm in Fig. 1 works as follows:

- The last stored block hash in the blockchain, the last stored miner address and the last stored nonce are hashed with sha224 generating a string of length 28 bytes or 224 bits.
- In Bismuth version 4.2.1.9 the 28 bytes hash is run through a binary conversion function which doubles the number of bits to 448, see Table 1.
- A mining condition string is created by using the first D (difficulty) bits of the 448-bit string from the previous step.
- Several miners (computers) running in parallel on the network are using the last block hash stored on the blockchain, their own miner address and pseudo-random generated nonces to create unique sha224 hashes converted to 448 bit strings using the binary conversion in Table 1. The total number of such 448-bit strings generated per second in the entire network of computers is denoted H (hash rate) in this paper.
- The first miner to generate a 448-bit string which contains the difficulty adjusted mining condition substring becomes the winner. This miner's block hash, address and nonce are stored on the blockchain and the entire process repeats.

Hex	Binary	Hex	Binary
0	0011 0000	8	0011 1000
1	0011 0001	9	0011 1001
2	0011 0010	a	0110 0001
3	0011 0011	b	0110 0010
4	0011 0100	c	0110 0011
5	0011 0101	d	0110 0100
6	0011 0110	e	0110 0101
7	0011 0111	f	0110 0110

Table 1: *1-char (4-bits) hex value to 8-bits binary conversion.*

3. Simulation Model

In this section a simulation model for the Bismuth blockchain in Matlab/Simulink is developed. The developed model differs from typical simulation models, since the discrete time step in the model is selected as one block. In other words, the Simulink model is discretized and the simulation steps $1, 2, \cdots, N$ correspond to block numbers in the blockchain. The actual time as measured in seconds between the blocks is given by the following equation.

$$T = \frac{2^{\text{floor}(D/2)}}{H \cdot \text{ceil}(28 - D/16)} \tag{1}$$

where T is the estimated block time in seconds, D is the current difficulty in bits and H is the current hash rate in hashes per second. The reason why D is divided by two in the function above, is to reverse the binary conversion given by Table 1. $2^{\text{floor}(D/2)}$ then represents the

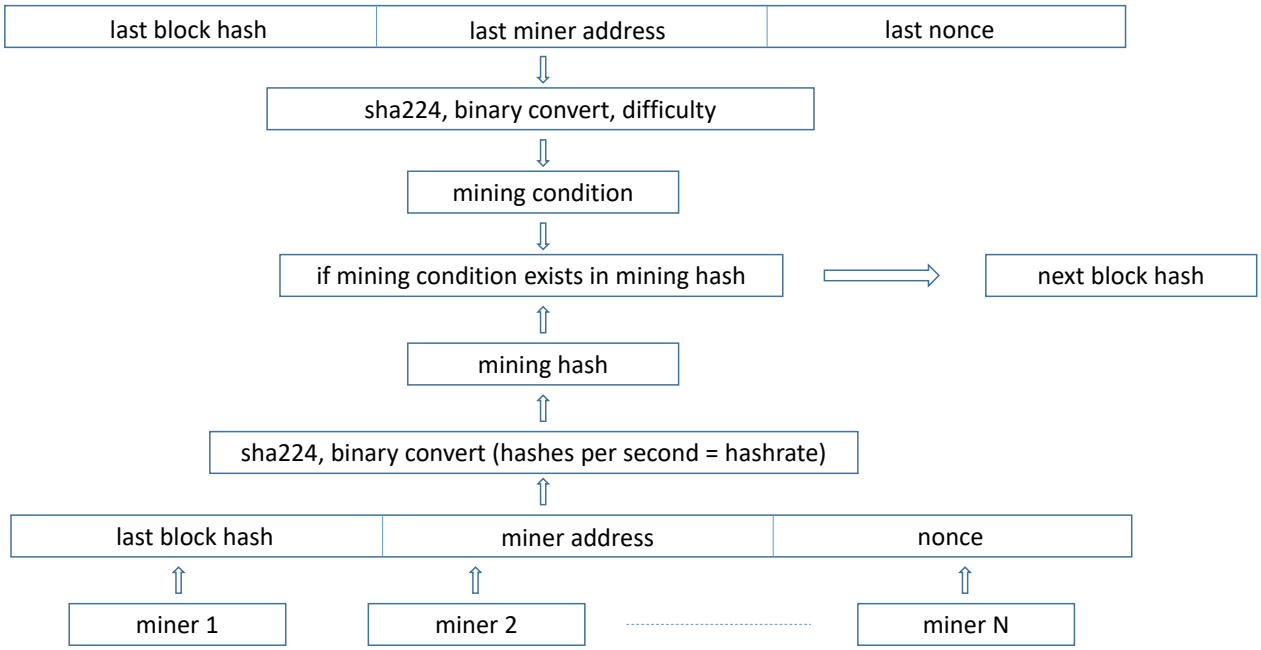

Figure 1: *Overview of the proof-of-work algorithm used by the Bismuth blockchain.*

total number of different possible mining condition bitstrings. The expression ceil$(28 - D/16)$ is explained as follows: every 16 in difficulty adds one byte to the mining condition. Total hash is 28 bytes (sha224, 224/8) and to fit the mining condition into it, it has to start at least $D/16$ bytes early. Only whole bytes are possible, therefore the ceil function is used. Fig. 2 shows the implemented Matlab function in Simulink.

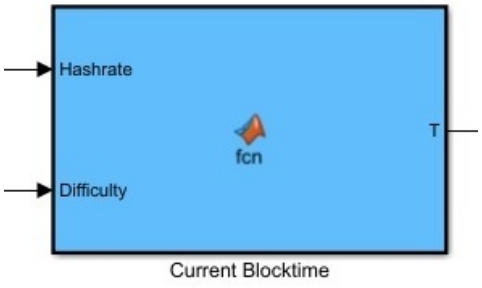

Figure 2: *Block time T as a function of hash rate H and difficulty D.*

The floor() function in eq. (1) makes the system highly nonlinear and the model linearization for the frequency analysis becomes dependent on the perturbation levels used in the linearization. Hence, for the simulation and controller development in this paper the block time calculation is approximated by the following Matlab code:

```
function T = fcn(H,D)
T = 2^(D/2)/(H*ceil(28-D/16));
```

The effect of this approximation will be studied later in the paper. The developed controller will be designed with stability margins for robustness against the approximation. The ceil() function in eq. (1) results in a much smaller sensitivity with respect to variations in D and is kept in the simulated model.

The estimated current block time from Fig. 2 is fed into a moving average filter implemented by the following Matlab function:

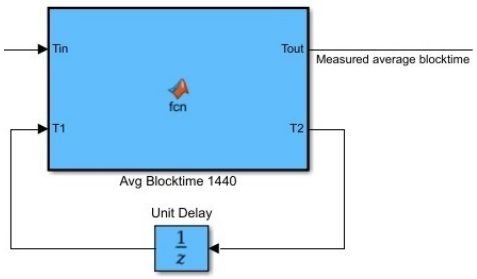

Figure 3: *Calculation of the average block time from the last 1440 blocks using a moving average filter.*

```
function [Tout,T2] = fcn(Tin,T1)
N=1440;
```

```
T2=zeros(N,1);
T2(1:N-1)=T1(2:N);
T2(N)=Tin;
Tout=mean(T2);
```

With an average block time of 60 seconds, there will be 1440 blocks per 24 hours. Hence, the moving average filter outputs the average block time over this period. In the function above Tin is the current block time while Tout is the averaged block time over the last 1440 blocks. The vectors T1 and T2 store the previous block times. The unit delay in Fig. 3 connects the two vectors T1 and T2. The initial condition of the unit delay when the simulation starts is a vector of size 1440 with all values equal to 60 seconds. The moving average filter is the most common filter in digital signal processing, mainly because it is the easiest digital filter to understand and use (implement), see Smith (1999).

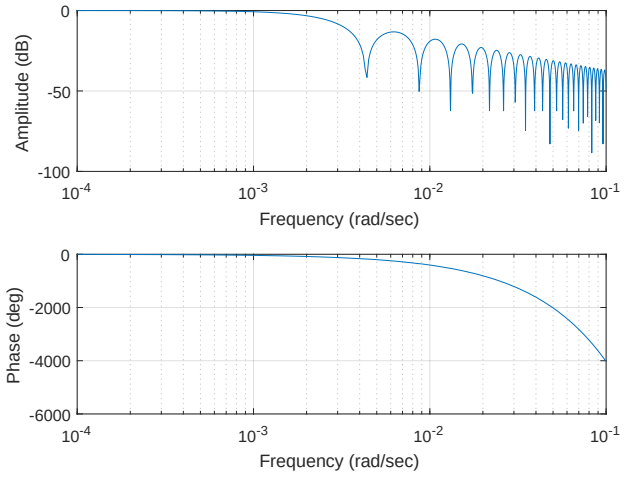

Figure 4: *Frequency response of the moving average filter of length 1440.*

Fig. 4 shows the frequency response (Bode) plot of the moving average filter. As noted by Smith (1999): *In spite of its simplicity, the moving average filter is optimal for a common task: reducing random noise while retaining a sharp step response. This makes it the premier filter for time domain encoded signals. However, the moving average is the worst filter for frequency domain encoded signals, with little ability to separate one band of frequencies from another. Relatives of the moving average filter include the Gaussian, Blackman, and multiple- pass moving average. These have slightly better performance in the frequency domain, at the expense of increased computation time.* The closed-loop control system to be presented in Section 4 of this paper operates in frequencies where the moving average filter has a gain close to 1 (0dB). Hence, the drawback mentioned in Smith (1999) is not a major concern for the developed controller.

4. Feedback Controller

The proposed controller is illustrated in Fig. 5. The constant T_d is the desired average block time equal to 60 seconds, T is the current average block time from the moving average filter presented in the previous section. T_1 is the average block time from the previous step, calculcated by the unit delay function. D is the difficulty and also the controller output. The Matlab

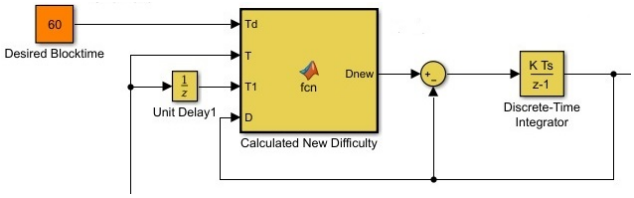

Figure 5: *Proposed controller for the difficulty adjustment.*

function named 'Calculated New Difficulty' shown in Fig. 5 contains the following nonlinear code, which includes a model inversion and a derivative term:

```
function Dnew = fcn(Td,T,T1,D)
H = 2^(D/2)/(T*ceil(28-D/16));
Dnew = 2/log(2)*log(H*Td*ceil(28-D/16));
Kd=10;
Dnew = Dnew - Kd*(T-T1);
```

In the code above H is the estimated network hash rate calculated from the current average block time T and the current difficulty D. D_{new} is the approximated new difficulty to achieve the desired block time of $T_d = 60$ seconds. The formula is an approximation because the current block time D is kept on the right-hand side of the equation. The effect of this approximation is small because of the ceil() function and also the division of 16, but it could have an effect at certain integer values of D. The effect of this approximation will be discussed in Section 5. The controller also contains a derivative term with a gain factor K_d which is multiplied by the difference between the current and the previous average block time, T and T_1. This term is subtracted from D_{new} because of negative feedback.

In series with the difficulty estimation and the derivative term, there is a gain K and a discrete-time integrator with a feedback loop which is an easy and practical way to implement a lowpass filter. In the continuous time domain, the closed-loop transfer function of a gain and an integrator with a feedback loop is:

$$G(s) = \frac{K/s}{1 + K/s} = \frac{K}{s + K} = \frac{1}{\frac{s}{K} + 1} \qquad (2)$$

Hence, the gain factor K becomes the cutoff frequency for a first-order lowpass filter. In practice, this low-

pass filter is used to limit the frequency range of the derivative term previously described to low frequencies only.

In summary, the controller consists of a nonlinear model inversion term which estimates the next difficulty level from the desired block time, the current average block time and the previous difficult level. In addition there is a lowpass filtered derivative term with controller parameters K and K_d.

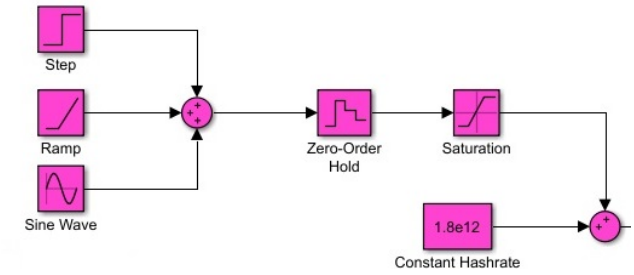

Figure 6: *The overall hash rate from the miners modelled as a disturbance into the system.*

The hash rate of all the miners in the network combined is modelled as a disturbance into the system. In order to analyse the closed-loop performance of the simulated system Fig. 6 shows the disturbance modelled as a constant hash rate of 1.8 gigahashes per second (GH/s) plus three optional terms which allow to vary the disturbance as a step, ramp and sinusoidal function, respectively.

Fig. 7 shows the overall control system model used in the Bismuth blockchain, version 4.2.1.9. The blocks marked in blue represent the blockchain system, the blocks marked in yellow represent the control system (difficulty adjustment calculation plus lowpass filtered derivative term) while the blocks marked in magenta represent the disturbance into the system (variation in the overall hash rate of the miners). The simulation step time defined in the zero-order hold function and the discrete-time integrator is set to 1, corresponding to one block in the blockchain.

The closed-loop system's bandwidth is defined as the frequency where the closed-loop gain equals -3dB, which occurs at 0.00202 rad/sec in Fig. 8 (blue curve). The frequency response plot was generated using Matlab's Control Systems Toolbox. The frequency 0.00202 rad/sec corresponds to $B_{bw} = 2\pi/0.00202 = 3111$ blocks or approximately 2.16 days with 1440 blocks per day. The red curve shows the closed-loop frequency response when the floor() function in eq. (1) is included. As can be seen from Fig. 8 the closed-loop bandwidth increases when the floor() function is included. A study of the closed-loop eigenvalues of the system, by using Matlab's Control Systems Toolbox, shows that the stability margin is reduced when the floor() function is included. The amount of stability margin reduction depends on the perturbation levels used when linearizing the model.

Fig. 9 shows the transfer function from the hash rate (disturbance) to the average block time (blue curve). The amplitude of this transfer function is always below -200dB, which is equivalent to a disturbance rejection of at least 10^{-10}. Hence a sinusoidal disturbance hash rate of 10GH/s will cause a disturbance in the average block time of less than 1 second. The red curve shows the same frequency response with the floor() function in eq. (1) included. Interestingly, the disturbance rejection is improved by approximately 50dB, or a factor of more than 300, for low frequencies when the floor() function is included.

The absolute values of all the eigenvalues of the discrete state space model were found using Matlab's Control System Toolbox and the feedback control system is found to be stable. The largest absolute value of the eigenvalues when $K = \frac{1}{720}$ and $K_d = 10$ was found to be 0.9991 which is inside the unit circle and guarantees that the closed-loop system is stable, see Table 2. The controller parameter K can be increased by a factor 4.7 before the largest absolute eigenvalue is on the unit circle. Hence, the gain margin of the closed-loop system with respect to K is $4.7 = 13.4dB$. The parameter K_d can be increased to a large value without causing instability but the drawback is a reduction in closed-loop bandwidth when K_d increases.

$\mathbf{K}$	$\mathbf{K_d}$	$\lambda_{\mathbf{max}}$	$\omega_{\mathbf{b1}}$ (rad/s)	$\omega_{\mathbf{b2}}$ (blocks)
$\frac{1}{720}$	0	0.9995	0.00210	2992
$\frac{1}{720}$	10	0.9991	0.00202	3111

Table 2: *Closed-loop performance as a function of the controller parameters K and K_d.*

5. Simulation Results

This section contains simulation results using the Simulink model in Fig. 7. Fig. 10 shows the simulated average block time when the hash rate (disturbance) is doubled from 1.8 terrahashes ($1.8 \cdot 10^{12}$) per second (TH/s) to 3.6TH/s at block number 10,000. The blue and the red curves are with $K_d = 0$ and $K_d = 10$, respectively. The simulation results show that the block time oscillates between 39 and 65 seconds, the settling time is approximately 4,000 blocks, there is zero steady-state error for a step disturbance and that the differentiation term of $K_d = 10$ increases phase margin / reduces overshoot. The initial condition for the diffi-

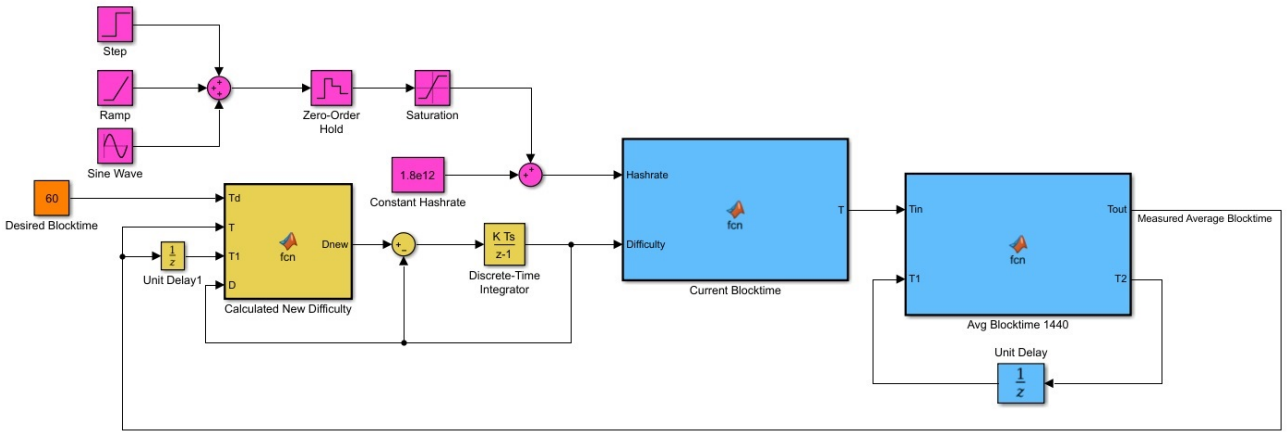

Figure 7: *Feedback control system used in the Bismuth blockchain, version 4.2.1.9.*

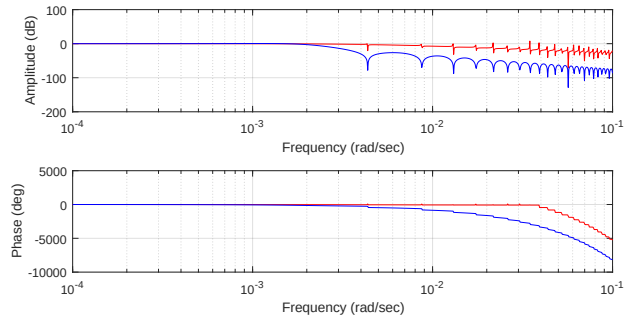

Figure 8: *Closed-loop frequency response from desired block time T_d to moving-average block time T with controller parameters $K = \frac{1}{720}$ and $K_d = 10$. The red and blue curves are with and without the floor() function in eq. (1), respectively.*

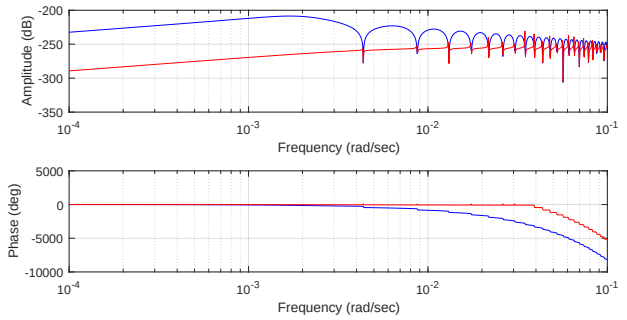

Figure 9: *Frequency response from disturbance (hash rate) H to moving-average block time T with controller parameters $K = \frac{1}{720}$ and $K_d = 10$. The red and blue curves are with and without the floor() function in eq. (1), respectively.*

culty D (discrete-time integrator) was set to 102 in the simulation. Fig. 11 shows how the difficulty D (controller output) increases from 102 to a new steady-state value of 104.2.

Fig. 12 shows the simulation results when the hash rate is ramped up from 1.8TH/s to 5TH/s at a rate of 0.1GH/s per block. The ramp disturbance causes the block time to drop initially, but the error goes to zero as time goes to infinity, also if the ramp was not stopped at block 50,000. Fig. 13 shows the corresponding controller output (difficulty level for the mining condition).

Fig. 14 shows the simulated block time when a high-frequency sinusoidal disturbance with amplitude of 0.2TH/s and period of 100 blocks is added to the hash rate. 100 blocks correspond to a frequency of $\frac{2\pi}{100} = 0.0628 rad/sec$ in the plot. At this frequency the disturbance rejection in Fig. 9 equals approximately $-250dB$ or a gain of $3.16 \cdot 10^{-13}$. This gain multiplied by 0.2TH/s gives a disturbance in the block time of 0.06s, which corresponds to the oscillations observed in Fig. 14. As can be seen in the figure, the derivative term $K_d = 10$ has no negative impact on the high-frequency disturbance rejection. Fig. 15 shows the corresponding controller output.

In eq. (1) and implemented in Fig. 2 there is the expression ceil(28-D/16). When developing the controller it was assumed that this term was constant, but at certain difficulty levels this will not be the case. For example, if D=112 we get ceil(28-D/16)=21, while D=111.99 results in ceil(28-D/16)=22. These discontinuities in the system model will affect the closed-loop control system. Consider a case when the overall hash rate (disturbance) is set to a constant of 55.9TH/s and the initial difficulty D=110. Fig. 16 then shows the response in the average block time. With $K_d = 0$ stable oscillations with amplitude 2.18 seconds around the desired block time occur, while with $K_d = 10$ these

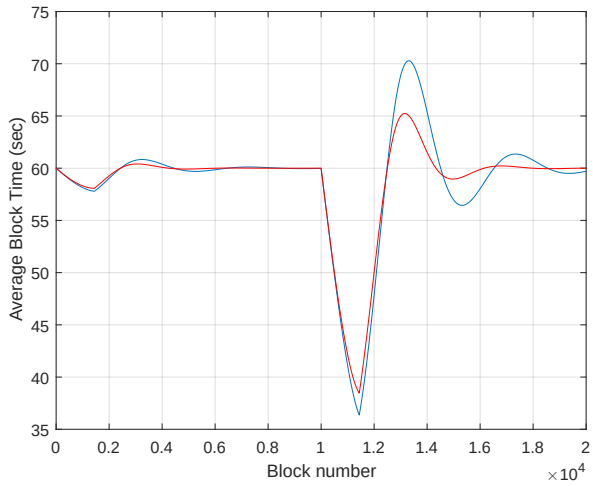

Figure 10: *Block time when hash rate is doubled from 1.8TH/s to 3.6TH/s at block 10000.* $K_d = 0$ *for the blue curve, while* $K_d = 10$ *for the red curve.*

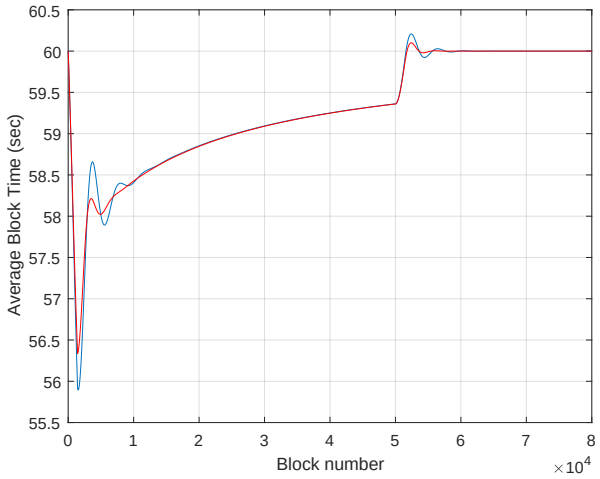

Figure 12: *Simulated average block time when hash rate ramps up from 1.8TH/s to 5TH/s at a rate of 0.1GH/s per block. The ramp stops at block 50,000.* $K_d = 0$ *for the blue curve, while* $K_d = 10$ *for the red curve.*

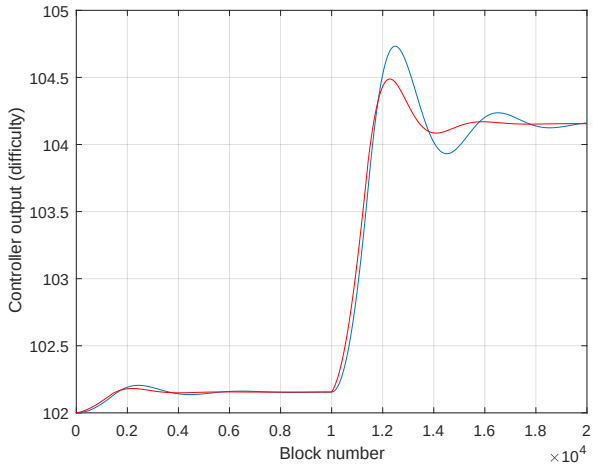

Figure 11: *Difficulty when hash rate is doubled from 1.8TH/s to 3.6TH/s at block 10000.* $K_d = 0$ *for the blue curve, while* $K_d = 10$ *for the red curve.*

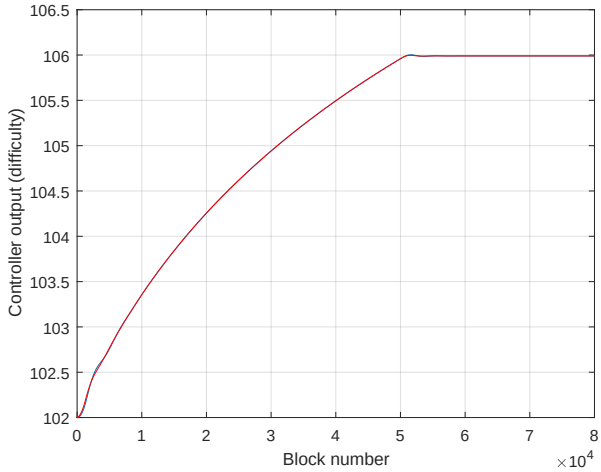

Figure 13: *Controller output (difficulty) when hash rate ramps up from 1.8TH/s to 5TH/s at a rate of 0.1GH/s per block. The ramp stops at block 50,000.* $K_d = 0$ *for the blue curve, while* $K_d = 10$ *for the red curve.*

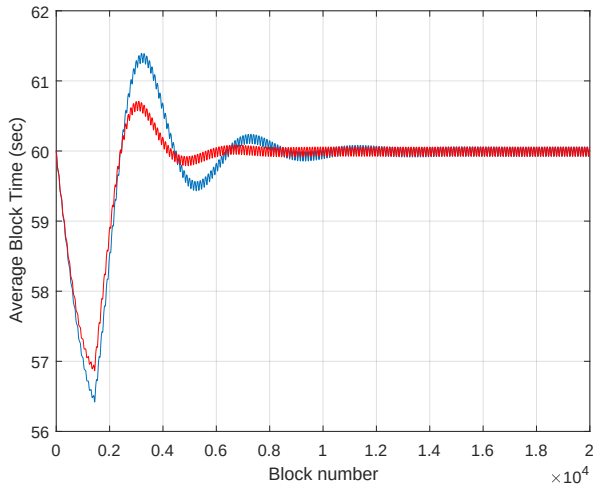

Figure 14: *Simulated average block time when a high-frequency sinusoidal disturbance with amplitude of 0.2TH/s and period of 100 blocks is added to the hash rate.* $K_d = 0$ *for the blue curve, while* $K_d = 10$ *for the red curve.*

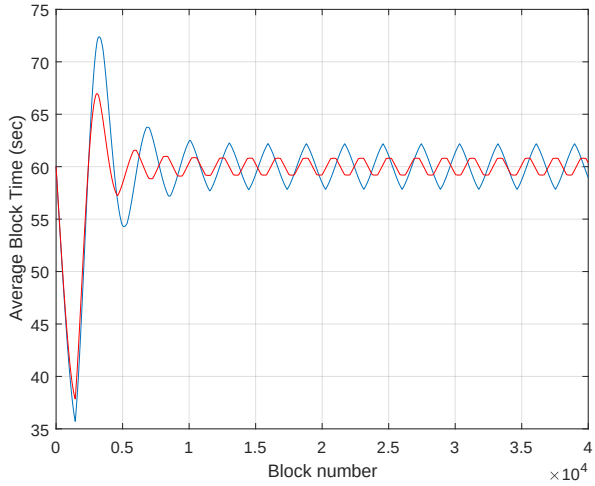

Figure 16: *Stable oscillations in average block time between 58.3s and 61.7s when hash rate is constant at 55.9 TH/s.* $K_d = 0$ *for the blue curve, while* $K_d = 10$ *for the red curve.*

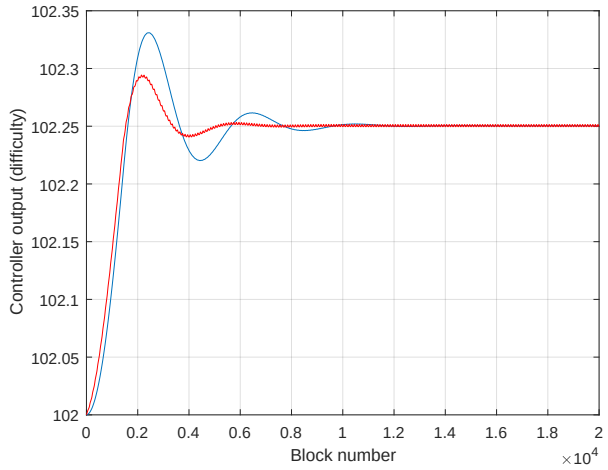

Figure 15: *Controller output (difficulty) when a high-frequency sinusoidal disturbance with amplitude of 0.2TH/s and period of 100 blocks is added to the hash rate.* $K_d = 0$ *for the blue curve, while* $K_d = 10$ *for the red curve.*

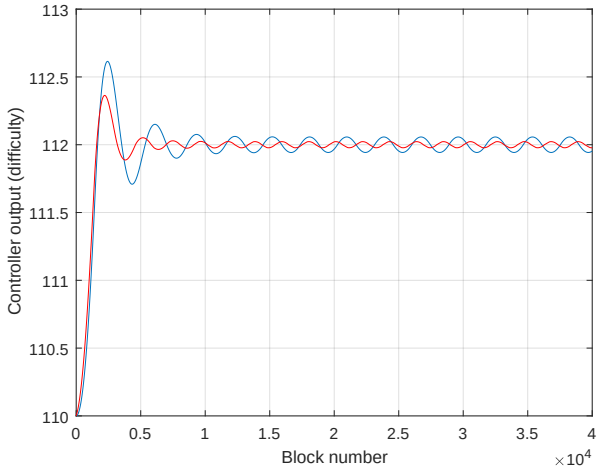

Figure 17: *Stable oscillations in controller output (difficulty) between 111.98 and 112.02 when hash rate is constant at 55.9 TH/s.* $K_d = 0$ *for the blue curve, while* $K_d = 10$ *for the red curve.*

oscillcations are reduced to an amplitude of 0.81 seconds. In other words, the derivative term K_d increases the damping in the closed-loop system and reduces the oscillation in this particular case by a factor 2.7. Fig. 17 shows the corresponding controller output (difficulty) as it initially starts at 110 and goes towards 112. The stable oscillations in the controller output are caused by the discontinuous ceil() function in the system model. This behaviour of the closed-loop system will occur at the following discrete set of difficulty levels $D \in \{8, 16, 24, \cdots, 432\}$.

6. Blockchain Network Results

In addition to simulations the difficulty adjustment feedback controller developed in this paper is implemented in the Bismuth blockchain as of version 4.2.1.9. The code in Fig. 24 shows the actual code implemented in Python.

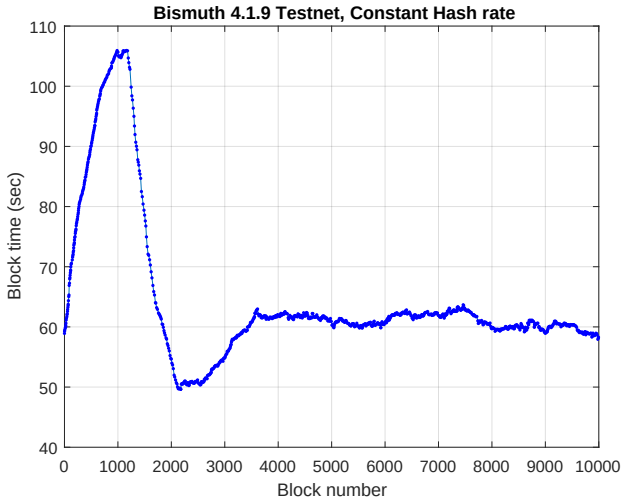

Figure 18: *Block time as a function of block number on testnet. Initial difficulty equals 86.26.*

Fig. 18 shows the block time when the developed controller is running on testnet with a constant hash rate. In this example testnet consisted only of one single node and two miners. Fig. 19 shows the corresponding difficulty level which initially started at 86.25. Since the initial difficulty in this test was too high for the available hash rate from the miners, the block time increases from 60 seconds to about 105 seconds, before the feedback controller reduces the difficulty and brings it back to the desired level of 60 seconds. Since the miners only generate solutions with integer difficulties, it is interesting to notice from Figs. 18 and Fig. 19 that the best level of control is achieved when the difficulty is close to the integer value of 84. This occurs near block

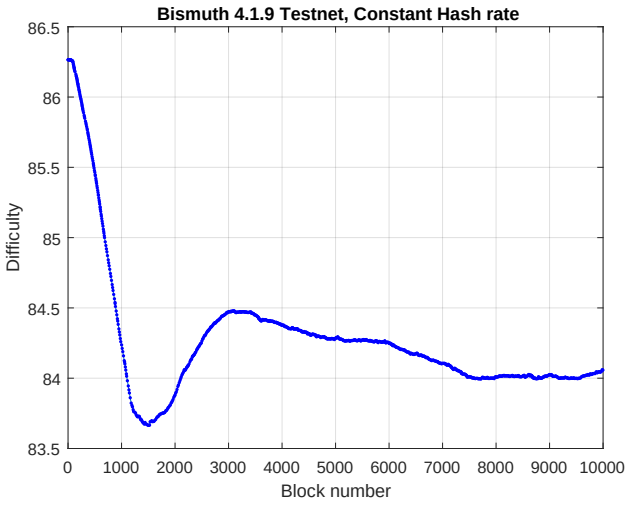

Figure 19: *Difficulty as a function of block number on testnet. Initial difficulty equals 86.26.*

number 7,500. If the block time goes above the desired value of 60 seconds, then the controller adjusts the difficulty to slightly below 84.0. Similarly, if the difficulty goes below the desired value of 60 seconds, then the controller adjusts the difficulty to slightly above 84.0. From block 8,000 to 10,000 in Fig. 18 the mean value of the block time is 59.80 seconds while the standard deviation is 0.66 seconds, which is considered a good result. Mining and generation of blocks is a stochastic process, hence on testnet the block time will not settle exactly at the desired reference level as it does in the simulations when the hash rate is constant.

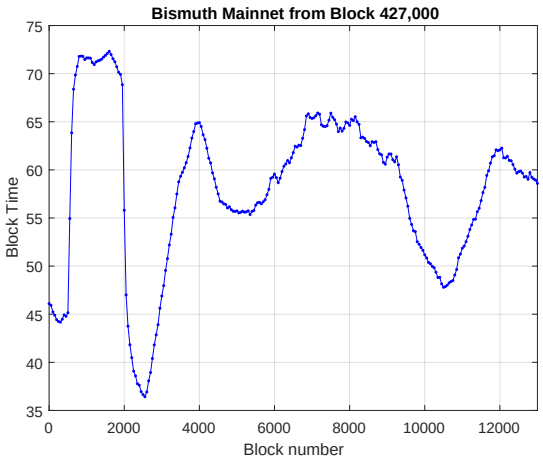

Figure 20: *Block time as a function of block number on mainnet starting from block 427,000.*

The proposed feedback controller for the difficulty adjustment was introduced on the Bismuth mainnet

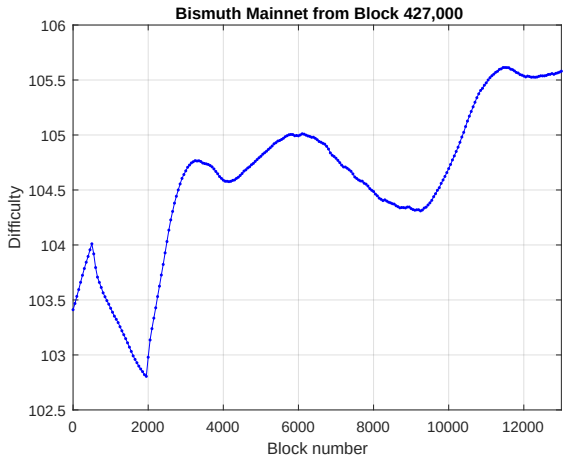

Figure 21: *Difficulty as a function of block number on mainnet starting from block 427,000.*

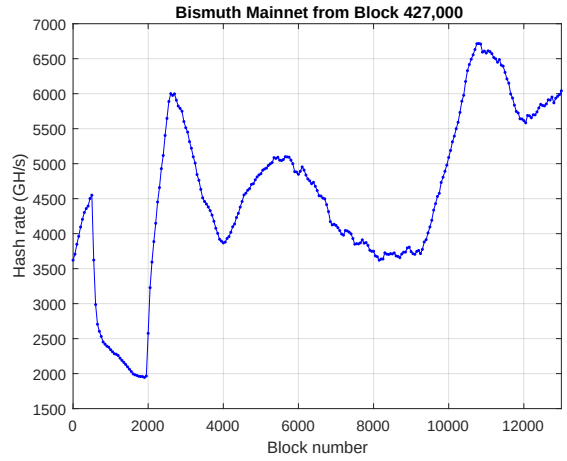

Figure 22: *Hash rate as a function of block number on mainnet starting from block 427,000.*

at block 427,000 on December 15, 2017 at UTC time 22:36:51. Figs. 20, 21 and 22 show the mainnet block time, difficulty and estimated hash rate starting from block 427,000. The initial condition for the block time was slightly above 45 seconds, due to the previous difficulty adjustment algorithm used on mainnet. For the first 500 blocks in the plots the new controller and the previous algorithm were compatible, then at block 500 the chains forked and as a result the hash power was divided between the two chains. Due to the reduced hash rate the block time increased sharply to above 70 seconds while the controller reduced the difficulty below 103, as shown in Figs. 21-20. Then at approximately block 2000 in the plots all the major miners updated to the new version and the hash rate increased sharply as seen in Fig. 22, which as a result caused the blocktime to decrease sharply to 37 seconds. For the remaining blocks 2000-13000 the feedback controller works continuously to keep the blocktime near 60 seconds, despite large variations in the hash rate (disturbance) as miners come and go. The results from mainnet demonstrate that the proposed feedback controller presented in this paper stabilizes the system.

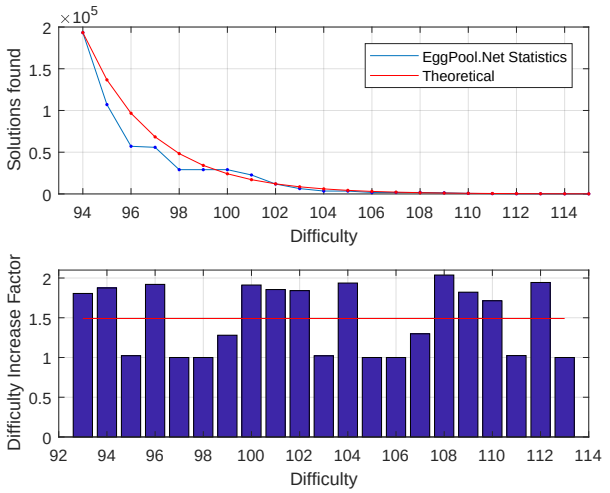

Figure 23: *Top: Analysis of solutions from miners vs. difficulty. The blue curve shows a statistical sample of 556,728 solutions from EggPool.Net, while the red curve is based on eq. (1). Bottom: Real increase in difficulty at different difficulty levels. Figures based on data from EggPool.Net.*

In eq. (1) it can be seen from $2^{\text{floor}(D/2)}$ that the block time should theoretically increase by a factor of $\sqrt{2}$ when D increases by 1 and the hash rate is constant. The developer EggdraSyl at EggPool.Net which is one of the largest mining pools for the Bismuth blockchain, has analyzed the theoretical increase in difficulty vs. the real solutions generated by the miners contributing to the pool. Fig. 23 (top) shows that there is some discrepancy between the modelled difficulty increase and the statistical analysis. While the overall reduction in block generation matches well with $1/\sqrt{2}$, there are variations between the different levels. For example, the reason why $D = 104$ occurs relatively seldom and 105 relatively often, is that 104/8 is a whole integer. The first bit in the next byte is then always a zero introduced by the binary conversion in Table 1. This means that hits on 104 will almost always also give hits on 105. This behavior occurs for other integer values of D/8, for example 96 and 112 ($\pm$ 8 bits from 104). Fig. 23 (bottom) shows the experimentally found difficulty increases at different difficulty levels. As can be seen from this figure, the difficulty increases 96→97, 104→105 and 112→113 are all approximately 1. The average difficulty increase of 1.49 in Fig. 23 matches quite closely the theoretical value of $\sqrt{2}$.

```
# Assume current difficulty D is known
D = diff_block_previous
# Assume current block time is known, calculcated from historic data,
# for example last 1440 blocks
T = block_time
# Calculcate network hash rate
H = pow(2, D / 2.0) / (T * math.ceil(28 - D / 16.0))
# Calculate new difficulty for desired block time of 60 seconds
Td = 60.00
D0 = D
Dnew = (2 / math.log(2)) * math.log(H * Td * math.ceil(28 - D0 / 16.0))

# Feedback Controller
Kd = 10
Dnew = Dnew - Kd*(block_time - block_time_prev)
diff_adjustment = (Dnew - D)/720 #reduce by factor of 720
Dnew_adjusted = D + diff_adjustment
```

Figure 24: *Implementation of controller in the Bismuth blockchain, file node.py as of version 4.2.1.9.*

7. Discussion and Conclusions

In this paper a simulation model for the Bismuth blockchain has been developed and implemented. The model has been validated by a statistical analysis of miners solutions vs. difficulty levels from a sample of more than 500,000 solutions provided by EggPool.Net. Although there are variations in the increase at different levels of difficulty, the overall trend of increase matches the theoretical average of $\sqrt{2}$.

A novel control algorithm for the difficulty adjustment has been developed, analyzed and implemented. The controller estimates the total hash rate H in the network based on the current levels of difficulty D and average block time T. This part of the controller is an approximated model inversion of the system model illustrated in Fig. 2. A discontinuous function (floor()) in eq. (1) is not invertible and hence has not been included in the inversion and system simulation. Instead, this approximation is seen as a model uncertainty and handled by the stability margins of the closed-loop controller. In addition to the model inversion, a derivative term multiplied by the gain K_d is introduced, which significantly reduces oscillations and increases the stability margin of the closed-loop system.

The proposed controller has been tested not only in simulations but also on a testnet using a constant hash rate and on the mainnet of the Bismuth blockchain starting from block 427,000 where there are relatively rapid changes in the overall hash rate. In both implementations the proposed controller has performed as expected and in correspondance with the simulation results.

References

Booth, N. Kyuupichans Difficulty Algorithm (D578) Explained. https://tinyurl.com/y8zxvn7g, visited on Dec. 1, 2017.

Jakobsson, M. and Juels, A. Proofs of Work and Bread Pudding Protocols. *In: Preneel B. (eds) Secure Information Networks. IFIP The International Federation for Information Processing*, 1999. 23:258–272. doi:10.1007/978-0-387-35568-9_18.

Kučera, J. Bismuth source code, release 4.2.1.9. https://github.com/hclivess/Bismuth/releases, commit hash e18a9a8e99c9985f2da637d4e7cb04367a78154a, 2017.

Penard, W. and van Werkhoven, T. On the secure hash algorithm family. https://www.staff.science.uu.nl/~tel00101/liter/Books/CrypCont.pdf, visited on Nov. 19, 2017, 2007.

Rogaway, P. Nonce-Based Symmetric Encryption. *Lecture Notes in Computer Science: Roy B., Meier W. (eds) Fast Software Encryption*, 2004. 3017:348–358. doi:10.1007/978-3-540-25937-4_22.

Sechet, A. Deadalnix's Difficulty Algorithm (D571) Explained. https://tinyurl.com/ycj7j2sg, visited on Dec. 2, 2017.

Smith, S. W. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, San Diego, California, 2nd edition, 1999.

```
import json, base64, hashlib, socks, connections

def file_hash(filename):
    #file_hash
    h = hashlib.sha256()
    with open(filename, 'rb', buffering=0) as f:
        for b in iter(lambda : f.read(128*1024), b''):
            h.update(b)
    return h.hexdigest()
    #file_hash

def blockget(socket, arg1):
    #get block
    connections.send(s, "blockget", 10)
    connections.send(s, arg1, 10)
    block_get = connections.receive(s, 10)
    return block_get
    #get block

s = socks.socksocket()
s.settimeout(10)
s.connect(("127.0.0.1", 5658))
block = blockget(s,406054)
s.close()

jsonData = block[0][11][4:]
jsonToPython = json.loads(jsonData)

with open(jsonToPython['filename'], "wb") as fh:
    fh.write(base64.b64decode(jsonToPython['data'].encode('ascii')))

filehash = file_hash(jsonToPython['filename'])
print("File {} extracted".format(jsonToPython['filename']))
print("File hash matches: {}".format(filehash == jsonToPython['sha256']))
```

Figure 25: *Python3 code for extracting the Simulink model in Fig. 7 from the blockchain, block 406,054, transaction 0. The JSON data for the Simulink model is stored in the OpenField, array index 11.*

A. Simulink Model

The Simulink model in Fig. 7 is stored on the Bismuth blockchain and can be extracted by running the Python3 code listed in Fig. 25. The code in Fig. 25 must be saved and executed in the main directory of a Bismuth local node where the required import file connections.py is also stored. Before the model can be extracted a local node must be started and the blockchain synced. The simulation model is made publicly available in this way in the spirit of open access publication and open source software, to ensure long-term archiving, to encourage others to work on the problem and to provide a simulation testbench for further development and analysis of blockchain feedback controllers. By using the simulation model presented in this paper controller designs can be tested and analyzed in a few seconds compared to typically several days for an implementation on a testnet.