Electric Switch with Ethereum Blockchain Support

Pustišek, Matevž; Bremond, Nicolas; and Kos, Andrej

Abstract: ***This paper presents a prototype implementation of a USB charging device with Ethereum blockchain support for booking and payments. It outlines the design of the system and selection of hardware and software components for the end-to-end solution. Blockchain technologies are relatively new and promising development for decentralized trusted transactions, including micropayments. There are many possible application domains of blockchain technology envisaged; one of them is smart grid. Several initiatives are prototyping solutions in this domain; however, there is still a vast unexplored area related to technological and business aspects of blockchain in energetics. Our prototype provides an end-to-end solution that is comprised of a blockchain enabled end-device, Web applications with blockchain clients for users and administrators, and corresponding smart contracts for the specific transaction logics. The prototype was implemented as a DIN rail compatible device based on RPi and a user application as Javascript code embedded in HTML page that can be opened in Chrome browser with Metamask plugin installed. The system was successfully implemented and tested and it confirmed the viability of the concept. With modest modifications this approach could be extended to electric vehicle chargers, smart grid supply and demand management or other utility support systems.***

Index Terms: ***blockchain, charger, dapp, Ethereum, prototype, smart grid, switch.***

1. *INTRODUCTION*

TRUSTED decentralized applications based on blockchain (BC) technologies are raising immense interest in technical and business communities. However, the actual maturity of BC in terms of technology, business models and applications does not yet match this reputation. According to Gartner's hype cycle for emerging technologies in 2016 [1] blockchain is placed close to the peak of inflated expectations. A way to progress towards productive solutions is developing working prototypes, which apply combinations of technologies to demonstrate viable use-cases that further help reinforcing the role of novel technologies in an even broader scope of ICT applications. Besides, there are only scarce studies with solid scientific backgrounds that research application possibilities of blockchain technologies. This is also one of the motivating factors behind our work.

Energy and smart grid seems a natural application domain for blockchain technologies, so there are already some proof-of-concept developments being presented in this domain. Slock.it is a German company [2] specialized in blockchain applications with real-world objects. One of the potential use cases that they are addressing is energy and smart grid. The Share&Charge [3] is their PoC project in this vertical. Brainbot technologies AG [4] develops and investigates the role of Raiden (see Section 2 for details). It demonstrated in the Raiden Network IoT Demo [5] how an electric switch could be controlled via Raiden as well.

We designed, developed, implemented and tested a smart electric switch with charger, which is controlled through the Ethereum blockchain network [6]. It is comprised of end-user and administrator applications, a smart contract, as well as a BC aware IoT device. It is thus a prototype of an end-to-end IoT solution, based on blockchains — unlike many recent blockchain developments, which frequently focus on creating yet another alternative BC coin along with corresponding smart contracts, to support vaguely defined potential use cases. Not that many initiatives bring BC to the real world—including IoT. The main motivation for this work was gaining experience with blockchain technologies, and related application development and implementation requirements.

In Section 2 we briefly present the role of the Ethereum protocol, which was applied in our prototype. In Section 3 we elaborate the system design and in Section 4 we present the implementation and results. We conclude the paper with a brief outline of possible extensions of the current system.

Manuscript received June 1, 2017.

Author is with the Faculty of Electrical Engineering, University of Ljubljana, Slovenia (e-mail: matevz.pustisek@fe.uni-lj.si).

2. BLOCKCHAINS FOR IOT

Blockchains and distributed ledgers are listed among top strategic technology trends in 2017 [7]. They provide a decentralized framework for trusted transactions. The blockchain technologies are well known fundament of cryptocurrencies (e.g. Bitcoin), but offer many other possible applications areas, too.

The Ethereum protocol [6] and corresponding networks are the basis for trusted, decentralized applications – *Dapps*. Apart from enabling a relevant cryptocurrency – ether [8] – Ethereum protocol is distinguished by a highly generalized programming language. With it one can code a smart contract which is deployed to the Ethereum network. It forms a contract account which is controlled by its contract code. The code is executed every time such an account receives a message/transaction from another account. Executions of smart contracts are validated in the blockchain network. This is a fundament for various IoT related blockchain applications. Particular benefits from BC, relevant to IoT, include machine-to-machine transactions, micropayments, decentralized data feeds, and scalability. Ethereum protocol is deployed in various public Ethereum networks[9]. The two key ones are the *mainnet* and *testnet* (current version is called Ropsten Revival) and both apply the same Ethereum protocols. In *mainnet* the cryptocurrency (i.e. ether) has a real value and can be traded for fiat currencies. The ether in *testnet* has no real value, and the network is meant for testing purposes.

Many established IoT platforms announced or have already implemented support for BC [10], [11]. Different alliances are being built here and different BC protocols can be found in this role. But what they mostly have in common is an API that facilitates use of full scope BC functionality from the Web application development tools. However, while exploring integration of BC into actual IoT architectures, several shortcomings started appearing with implementation of first prototypes. Smart contracts in Ethereum environment operate in a rather isolated space. The Solidity [12] smart contract programming language has e.g. no means to request data form URLs and thus to interface with "real" Internet world — e.g. Web sites and IoT devices — because these external information cannot be trustworthily verified by the contract. This shortcoming can be outdone by oracles [13]. They serve as intermediaries, providing data feeds along with an authenticity proof to the blockchain form/to external software (e.g. Web sites) or hardware entities. Another challenge is transaction validation period. Due to the blockchain nature, this can take from several tens of seconds (in Ethereum about 20s) up to several minutes (in Bitcoin 10min or more). Besides, distributed application developers do not have an influence on these times, which in addition may become even longer due to higher transaction rates in the network. This limits, at least to some extent, the feasibility of scalable instant payments. A solution to this problem is being sought in state channels. This architecture combines off- and on-network transactions to contribute to additional scalability, privacy and reduction of confirmation delays, compared to the current BC architecture. In Ethereum this approach is manifested in the Raiden [14] and in Bitcoin the Lightning network [15].

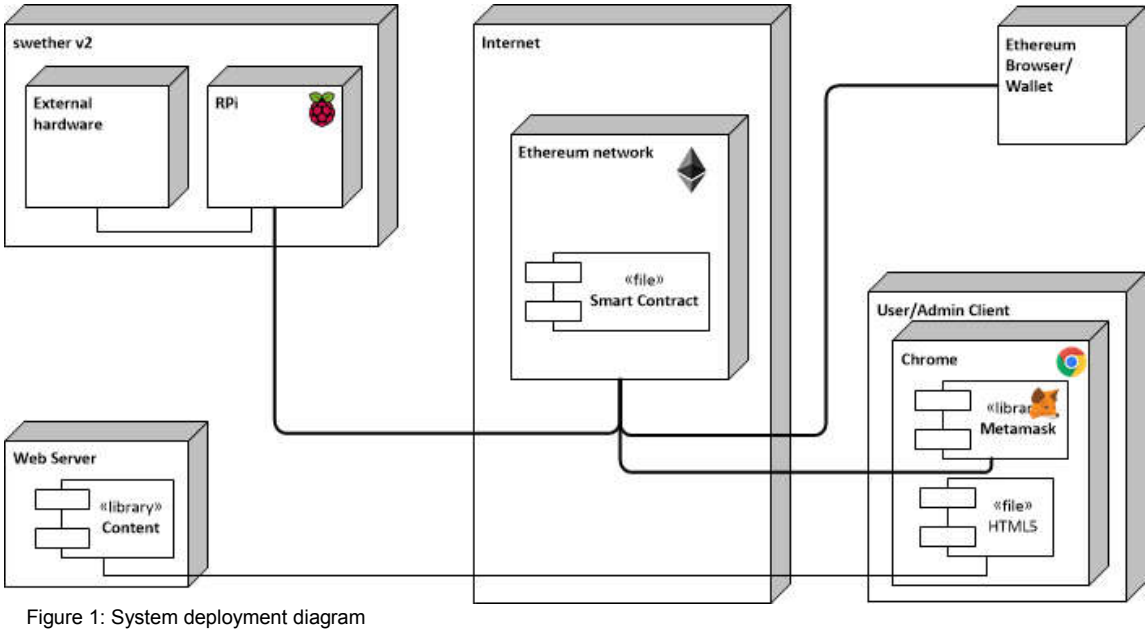

Figure 1: System deployment diagram

3. PROTOTYPE SYSTEM DESIGN

To demonstrate feasibility and to address the full scope of development activities for blockchain, we decided for a system that is comprised of a blockchain enabled IoT device (called Swether – switch with Ethereum support), as well as of customized Web applications for users and administrators to interact with the device via blockchain. The same principle can be easily extended e.g. to 230 V AC chargers or used to control access (and not to charge for it) to devices and services with control of their power supply.

3.1 DApp Architecture and System Model

Overall system architecture is depicted in Figure 1. Swether — switch with Ethereum support — is an IoT device controlled via a DApp and blockchain network. A smart contract implementing the system backend is deployed to the BC network during the system setup by the system owner. Users and administrator access the functionalities implemented in Web applications through Ethereum Web browsers. In our case, the default browsers are Chrome with Metamask plugin and Mist. Web server merely provides the access to user and administrator Web pages. All the actors in the system presented in Section 3.2 need valid Ethereum accounts and minimal sum of ether to compensate for charging costs and transaction fees. The system is currently deployed in Ethereum *testnet* (Ropsten), but can be in the same way deployed in *mainnet*, too.

3.2 System Outline – Actors and Activities

There are four actors foreseen in the system: device user, device administrator/owner, smart Web application, and the Swether device. A **user** can review current availability of free charging slots and book a desired quantity of time for charging. The booked time is instantly paid with Ether. Swether device monitors the Ethereum network and toggles the charging slots according to transaction status. An **administrator** can set the basic parameters of Swether, like the maximum number of charging slots in a device and current price per time unit. Administrator can transfer funds from smart contract address to an arbitrary Ethereum address.

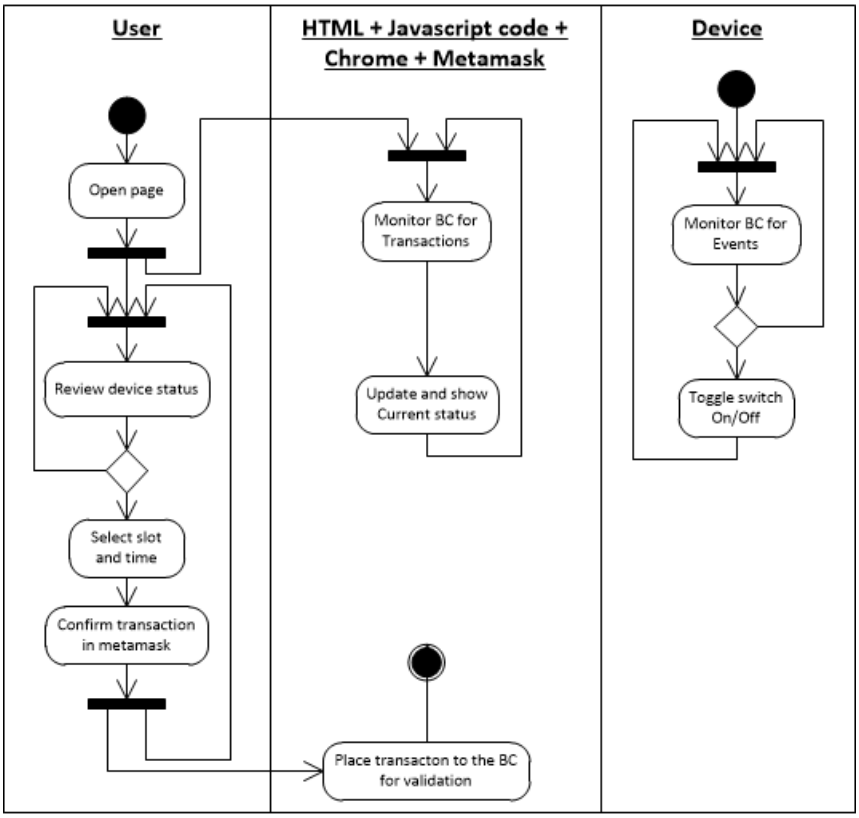

Figure 2: User activity diagram

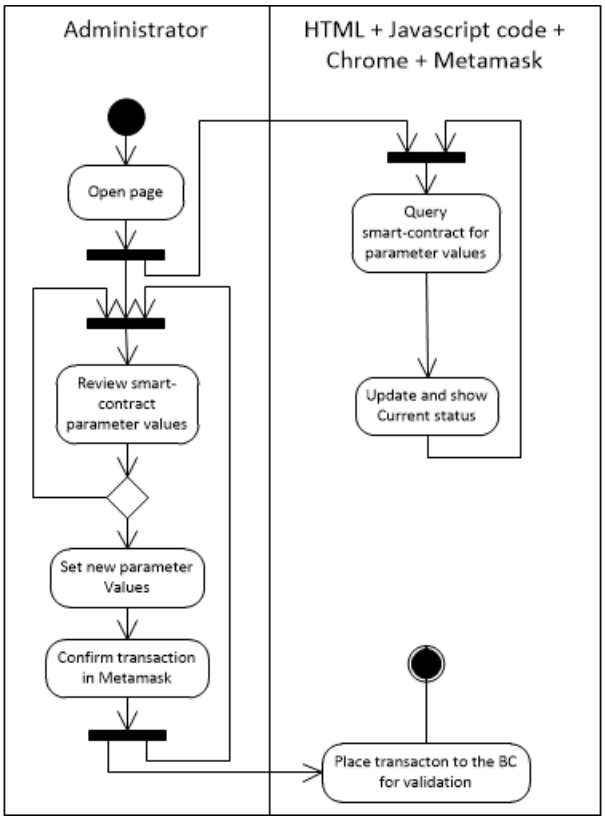

Figure 3: Administrator activity diagram

As depicted in Figure 2, two activities run in parallel. One is related to the operation of the device, which constantly monitors the blockchain for relevant events. An event is created when the transaction for a charging request is validated by the network. If a corresponding event is identified, the device toggles on the selected charging slot based on the parameter values in the transaction. When the charging period terminates, it automatically toggles off the slot.

The second activity involves user and his smart application. User downloads the HTML file and the Javascript logic for communication with blockchain. The page has to be opened in an Ethereum compliant Web browser. At the time of writing we relied on Google Chrome with Metamask plugin [16] as the Ethereum client and Mist. The Javascript application retrieves expected device and charging slot status from the smart contract in blockchain and visualizes it in the page. User can thus review the status and proceeds to the reservation of one of the slots. He selects the desired booking duration or the price in Ether he is willing pay for charging. The application creates the appropriate transaction. Final confirmation is left to the user and then the transaction is placed to the chain for validation. Once validated, the Swether device intercepts corresponding event and reacts on it. There is thus no direct communication between the user application and the device. The only interconnection is via a transaction validated in blockchain.

Administrator on the other hand, has more limited scope of activities. Once the system has been set up, including the deployment of the smart contract to the blockchain (this step is not a part of the runtime and is therefore not depicted in Figure 3), administrator reviews current contract parameter values (e.g. number of charging slots, energy price per minute) and sets new values. The modified values are placed in the smart contract via a BC transaction, too.

Retrieval of funds received by the smart contract is not depicted either. This is an activity which is inherent to smart contract accounts in Ethereum. It enables administrator to collect the revenue, by transferring the funds to another Ethereum account.

4. System Implementation and Results

4.1 Swether

Swether device deployment is depicted in Figure 4. The core of the device is a Raspberry Pi 3 Model B [17], a popular miniature computer and embedded platform, rich in communication capabilities (Ethernet, Wifi, BLE, USB and HDMI). It is able to run various operating systems; among others the Linux flavored

Raspbian [18]. As an embedded device it has 40 GPIO pins to interface external hardware. In our case these included a 5V 4-channel power relay module with optocouplers and LED for status indication.

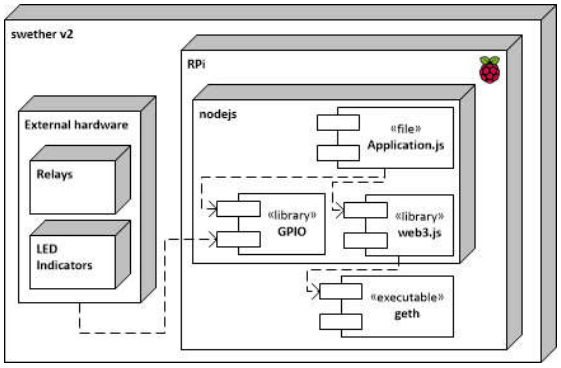

Figure 4: Deployment diagram of Swether v2

The application logic in Swether is implemented in Javascript. This is mostly due to the well documented web3.js API of the *geth* [19] Ethereum client. The *geth* is responsible for running Ethereum protocols and thus the entire communication with the blockchain. Through the web3.js API application can monitor status of the client and monitor or create transaction. As the application requirements for external hardware interfacing via GPIO were not complex, we implemented this part of the application in Javascript, too. In future version of the device we plan to implement hardware related part in Python, due to the better support of additional HW items foreseen in the device and richer programming capabilities.

Figure 5: Swether — the smart charging device

Once configured and deployed, the device requires no user interventions. There are just three LED indicators in the housing to indicate the status of the Ethereum network/client (Figure 5).

Swether device is not represented in the Ethereum network as an active entity and is thus not identified by its own Ethereum address. The *geth* client in the device constantly monitors the transactions in the network can catches the transaction events that were generated by the corresponding smart contract. The appropriate smart contract address is defined at the time of the device deployment.

4.2 End-user Web Application with Ethereum Support

End-user interface and application was implemented as HTML 5 page that has to run in an Ethereum compliant Web browser. This can be done in dedicated Ethereum wallets/browsers like Mist [20]. However, the application of Metamask plugin and Google Chrome browser assures a more transparent user experience. In this case user applies the same Web browser he is already accustomed to. The Metamask [21] is a light Ethereum client exposing the JSON-RPC and standard Ethereum web3 APIs [22]. The web3 library communicates with the Metamask client through JSON-RPC. A HTML page with application logic for Metamask has to be placed and retrieved from a HTTP server (and not from a local file) due to browser security restrictions.

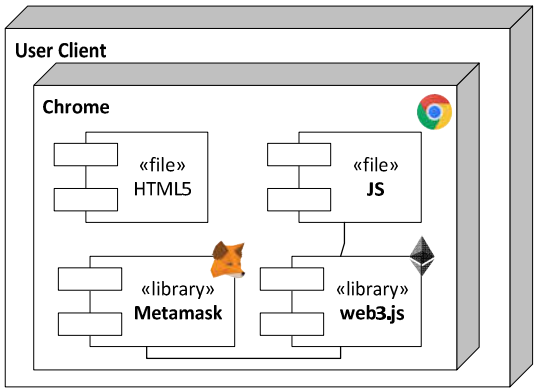

Figure 6: Deployment diagram of the end-user Web application

Upon the launch, the application verifies the Metamask client and Ethereum network status. If the status is OK, user proceeds to the overview of Swether device status, as seen in Figure 7. User can then select one of the unoccupied slots and the desired duration of the reservation or the total price she is willing to pay. The total price or time, which is indicated, is calculated from the current per minute rate that was retrieved from the smart contract. By pressing the “Book” button application code defines the transaction content. It passes it via web3 API to Metamask, which builds the transaction and sends it to the Ethereum network. The transaction is addressed

to the smart contract, where it triggers the function *bookAPlug* with parameters required for a booking. Within the transaction/execution of this function, an event is created. It serves later as an indication to the smart device about a relevant state-changing transaction. Once the transaction is validated by the blockchain, the act of booking is trustworthily recorded.

The key benefit of a custom Web user/admin application is twofold. First, it provides better user experience and customized application appearence. All the functionalities of the smart contract can be used from a general Ethereum compliant wallet, too. But such a use resembles very basic parameter settings. The Web based user interface in our solution can profit from all features of modern Web user interfaces design. Second, the Metamask is easy to be installed and used. It relies on remote geth nodes for better responsiveness (i.e. light-client mode). This again reduces the burden from user and facilitates promp start for using the system.

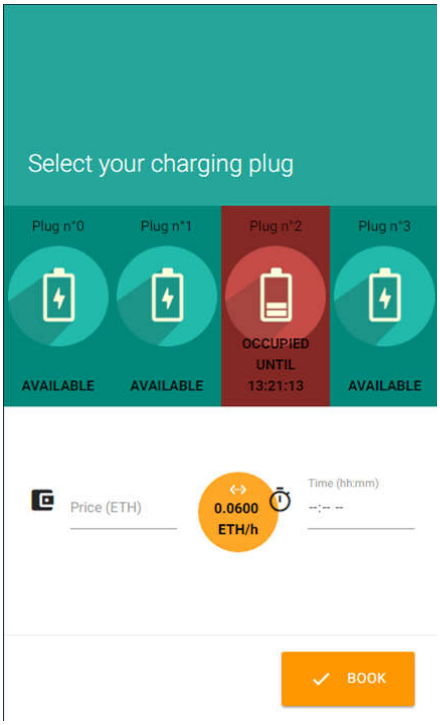

Figure 7: User-interface in Chrome browser

4.3 Administrator Web Application with Ethereum Support

Administrator interface and application was implemented in the same way as the user-interface. The difference is in functionality. On the page the administrator can monitor and change the parameters, which determine device setup and operation. They are listed and described in Section 4.4.

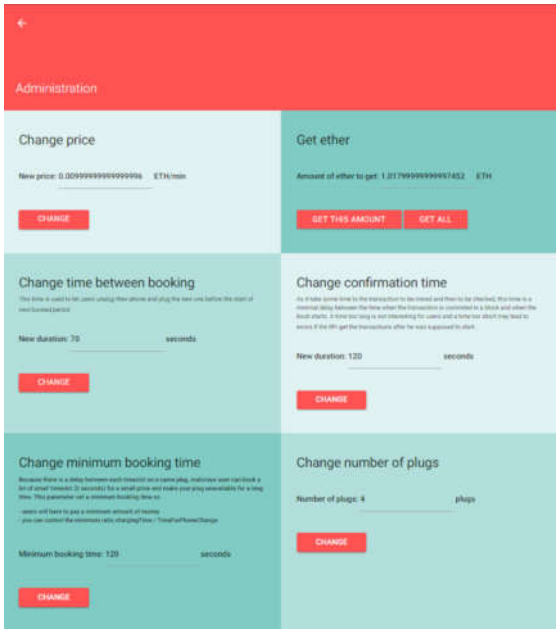

Figure 8: Administrator interface in Chrome browser

The page is publically accessible from a Web server. But only if the page user applies an Ethereum account in Metamask plug-in which is marked in the smart contract as an administrator account, he can set new values for the parameters. The page and the parameters are not hidden, because they can be derived from the Ethereum blockchain anytime. Granting the access to the functions changing the values is assured by a smart contract and not at the Web application level.

4.4 Application Backend — Smart Contract

System operation is supported by a dedicated smart contract, called *plugBooking*. It was implemented in Solidity and deployed to the Ethereum network from the administrator account. The *plugBooking* contract provides two sets of parameters and functions, one supporting the user application and the other the administrator Web interface.

Key parameters that determine the device setup and operation are:

- *plugNumber* – the number of charging plugs/ports that a device has, it must be set accordingly to physical implementation of the device;
- *minimumBookingTime* – minimal duration of plug reservation as a means to prevent malicious reservations with 0 duration and respective DOS of the Swether device;
- *price* – the current per second rate of charging in Wei[1];

[1] Ether (ETH) is the native cryptocurrency in the public Ethereum network and it can be traded for real currencies. $1Wei=10^{-18}$ ETH

- *userSwapTime* – a safety period between two consecutive chargings at the same plug, which allows two users to successfully swap their charged devices;
- *confirmationTime* – an estimation of time needed for the transaction to be considered as valid by Swether. In our case this includes time for 3 block depth for security, too.

The access to the functions for an administrator is limited to special accounts that are listed as admins. The owner account of the contract automatically serves as an administrator's, too. Besides, he can grant access privileges for up to 16 other accounts. The only action he cannot take is destructing the contract. Key administrator functions enable setting new values for the key parameters that determine the device setup and operation (see the list above). Besides, the following function is provided:

- *getEther* – to retrieve funds from the smart contract address to the administrator account address.

In the same contract there is a single function for users and their plug bookings. Access to this smart contract function is not limited and is thus available to any Ethereum network user (willing to book and pay for booking):

- *bookAPlug* – as parameters it receives a selected plug ID, expected duration and a time limit for maximal acceptable starting time (including current time of possible offsets due to transaction processing). There is an optional NCF tag ID parameter foreseen for future extensions of the smart-charging device, where additional user authentication via NFC will be provided.

There is a separate contract implemented for management of the list of *plugBooking* contract administrators.

The Swether device itself does not update status information in the smart contract. It merely applies the settings from the blockchain, to set its status accordingly.

4.5 Application Remarks

After the development, the Swether device was set for operation for a period of several weeks. During this time we experienced several issues in operation, mostly due to the blockchain and Ethereum clients. These included e.g. BC synchronization problems – node was connected to peers, but the chain was not syncing; problems with Ropsten testnet operation (they were solved by the Ethereum developers by forking the BC in testnet), and *geth* stability. The PRi 3 proved to be capable of running light Ethereum client, but we plan to conduct a deeper investigation of the performance of RPi with various client setups and networks. The times for a booking to become valid (transaction time plus time needed for 5 validations) are in our experience around one minute. This is too long for a quasi-real-time operation and this fact has to be considered in future use-case/activity definitions. There is some space to reduce this delay by reconfiguring the clients and setting a private Ethereum network. But even with all the efforts to do that, we do not expect delays to drop significantly without risking the stability of the blockchain.

Decision for user/administrator interfaces to be implemented with Web technologies proved to be the right one. It ensured nicely designed end-user interface with minimum efforts compared to e.g. native mobile app development. We are currently investigating options to develop mobile clients in comparably pragmatic manner.

5. CONCLUSION

The presented version of Swether is a prototype of an Ethereum blockchain controlled IoT device. We implemented the hardware and software for the device along with Ethereum compliant Web application for Swether control.

In the future Swether devices will be extended with three additional hardware components: LCD for status indication, NFC reader for user authentication and power meter to monitor actual energy consumption. The smart contract and device software will be modified as well. The Swether device will become an active Ethereum node capable of e.g. autonomously reporting actual energy consumption to the smart contract. This will enable us novel use cases, including scheduled reservations, refunding pre-booked but unspent energy, etc. We have also foreseeing a version of the system which applies the Raiden protocol to enable near real-time operation and to reduce the transaction validation costs.

ACKNOWLEDGMENT

The authors wish to acknowledge the support of the research program "Algorithms and Optimization Procedures in Telecommunications", financed by the Slovenian Research Agency.

REFERENCES

[1] "Gartner's 2016 Hype Cycle for Emerging Technologies Identifies Three Key Trends That Organizations Must Track to Gain Competitive Advantage," 16-Aug-2016. [Online]. Available: http://www.gartner.com/newsroom/id/3412017. [Accessed: 05-May-2017].

[2] "Slock.it - Solutions," Workshops, Projects and PoCs. [Online]. Available: https://slock.it/solutions.html. [Accessed: 05-May-2017].

[3] “Share&Charge - Charging Station Network - Become part of the Community!” [Online]. Available: https://shareandcharge.com/en/. [Accessed: 05-May-2017].
[4] “Brainbot Technologies AG,” Smart Contract and Blockchain Consulting for Enterprises. [Online]. Available: http://www.brainbot.com/. [Accessed: 05-May-2017].
[5] “Raiden Network IoT Demo,” *Brainbot Technologies*, 20-Dec-2016. [Online]. Available: https://www.youtube.com/watch?v=t6-rf68taTs. [Accessed: 05-May-2017].
[6] V. Trón, “Ethereum Specification,” 23-Jul-2015. [Online]. Available: https://github.com/ethereum/go-ethereum/wiki/Ethereum-Specification. [Accessed: 05-May-2017].
[7] K. Panetta, “Gartner’s Top 10 Strategic Technology Trends for 2017 - Smarter With Gartner,” 18-Oct-2016. [Online]. Available: http://www.gartner.com/smarterwithgartner/gartners-top-10-technology-trends-2017/. [Accessed: 05-May-2017].
[8] “ETH/USDT Market - Poloniex Bitcoin/Cryptocurrency Exchange.” [Online]. Available: https://poloniex.com/exchange#usdt_eth. [Accessed: 05-May-2017].
[9] “What’s the difference between the `testnet` and the production network technically?,” Ethereum Stack Exchange. [Online]. Available: https://ethereum.stackexchange.com/questions/6278/whats-the-difference-between-the-testnet-and-the-production-network-technical. [Accessed: 05-May-2017].
[10] “IBM Watson IoT - Private Blockchain.” [Online]. Available: https://www.ibm.com/internet-of-things/platform/private-blockchain/. [Accessed: 05-May-2017].
[11] “Blockchain as a Service (BaaS),” Microsoft Azure. [Online]. Available: https://azure.microsoft.com/en-us/solutions/blockchain/. [Accessed: 05-May-2017].
[12] “Solidity — Solidity 0.2.0 documentation.” [Online]. Available: http://solidity.readthedocs.io/en/latest/index.html. [Accessed: 08-May-2017].
[13] “Oraclize Documentation,” Overview. [Online]. Available: http://docs.oraclize.it/#overview. [Accessed: 05-May-2017].
[14] “Raiden Network,” High speed asset transfers for Ethereum. [Online]. Available: http://raiden.network/. [Accessed: 05-May-2017].
[15] “Lightning Network,” Scalable, Instant Bitcoin/Blockchain Transactions. [Online]. Available: http://lightning.network/. [Accessed: 05-May-2017].
[16] “MetaMask,” Brings Ethereum to your browser. [Online]. Available: https://metamask.io/. [Accessed: 05-May-2017].
[17] “Raspberry Pi 3 Model B.” [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. [Accessed: 05-May-2017].
[18] “Raspbian.” [Online]. Available: https://www.raspbian.org/. [Accessed: 05-May-2017].
[19] V. Trón, “Geth,” ethereum/go-ethereum Wiki · GitHub. [Online]. Available: https://github.com/ethereum/go-ethereum/wiki/geth. [Accessed: 08-May-2017].
[20] “Ethereum/mist: Mist,” Browse and use Ðapps on the Ethereum network. [Online]. Available: https://github.com/ethereum/mist/#mist-browser. [Accessed: 05-May-2017].
[21] “MetaMask/faq.” [Online]. Available: https://github.com/MetaMask/faq/blob/master/DEVELOPERS.md. [Accessed: 05-May-2017].
[22] “JavaScript API,” ethereum/wiki Wiki · GitHub. [Online]. Available: https://github.com/ethereum/wiki/wiki/JavaScript-API. [Accessed: 05-May-2017].