# Streaming Data Payment Protocol (SDPP) for the Internet of Things

Rahul Radhakrishnan, Bhaskar Krishnamachari
Viterbi School of Engineering
University of Southern California
Los Angeles, CA 90089
Email: rahulrad@usc.edu, bkrishna@usc.edu

***Abstract*—As the deployments of IoT systems grow for a wide range of applications, there are new use-cases emerging where the organizations or individuals that own sensor devices are different from the individuals or organizations that have use for the data from those devices. In such settings it is helpful for the data consumer to be able to effortlessly get data streams in return for monetary payments. The advent of cryptocurrency technologies have made the establishment of bidirectional automatic data micro-payment channels (with data flowing in one direction and micro-payments in the other) feasible. We present an application layer protocol called the streaming data payment protocol (SDPP) which embodies this very idea. The protocol also makes provisions for the data provider to send automated invoices and the data consumer to provide signed receipts for data to be stored on an immutable distributed ledger for auditing and dispute resolution purposes. We present an implementation of SDPP using TCP for data transport and IOTA as both cryptocurrency and a distributed ledger.**

***Index Terms*—IoT, Blockchain, SDPP, TCP, Data Payments, IOTA.**

## I. Introduction

The Internet of Things is enabling a rich set of applications by providing fine-grained real time data streams from a wide array of pervasively deployed sensors and is envisioned to be the key enabling technology for smart cities [1], [2]. The initial set of IoT applications have focused on vertically integrated solutions with end devices, cloud computing, and applications not only developed but also deployed and managed by the same individual or organization. This poses a challenge to scalability and interoperability. In particular, in the context of smart buildings or smart cities, it could be of interest for a third party application developer or an end user to access a stream of data from a particular sensor or set of sensors that is owned and managed by another entity.

Further, that stream may be need to be obtained only for a time-limited period so that it may not be feasible for the prospective consumer of the data to enter into a long-term agreement over the purchase/use of the data stream. As an example, consider a food truck vendor that is visiting a new city for a few days, deciding where to park her truck on different days of the week or even at different times of the day in a downtown area, in order to attract the most customers; there is a different sensor owner or even a collection of different sensor owners that is able to provide real time counts of pedestrians in various city blocks. It would be helpful for the food truck user to be able to order and pay for the pertinent streams of dynamic data that could be used to drive decision making about where to park the truck. It would be most flexible to employ a ”micro-payment” model, where the user could acquire the pertinent local sensor streams as and when needed. This would not only be beneficial to each user of the data from a cost perspective, it would also benefit the sensor device owners by giving them an incentive to share the data from their sensors, and by growing the pool of potential users attracted by the ease of obtaining data. Such micro-payments can also incentivize sensor owners to clean or process their data in ways to make them more valuable to users and also to deploy the right kinds of sensors in places where they are in demand, spurring the creation of a new kind of IoT data economy.

Such pricing-based transactions of sensor data streams are being proposed and starting to be actively explored in both academia and industry. The work by Birmpas *et al.* [3] explored how to price participatory sensing data. The Intelligent IoT Integrator (I3) project from USC has been exploring the creation of a community marketplace for IoT data where buyers and sellers can find each other and transact over real-time data streams [4]. More recently, the IOTA foundation has proposed and demonstrated the concept of a data market based on the DAG-based IOTA cryptocurrency platform in which both data and payments are securely communicated over IOTA's masked authenticated messaging mechanism [5].

We present our design of a streaming data payment protocol (SDPP), which enables a buyer and seller to easily connect and transact with each other using micropayments for streaming IoT data. Our design carefully separates out three key components: the data transport channel (which is operated as a traditional Internet client-server application-layer protocol, atop TCP), a payment channel (implemented using a cryptocurrency protocol), and a records medium (implemented using a distributed ledger technology). In light of the wide range of available options for cryptocurrencies and distributed ledgers (both permissioned and open systems), at the level of the protocol description, we purposely stay technology-agnostic regarding how the payment channel and the records medium are implemented; however, in principle these two could be implemented even on the same system. Our reference implementation of SDPP utilizes the IOTA tangle [6] as both

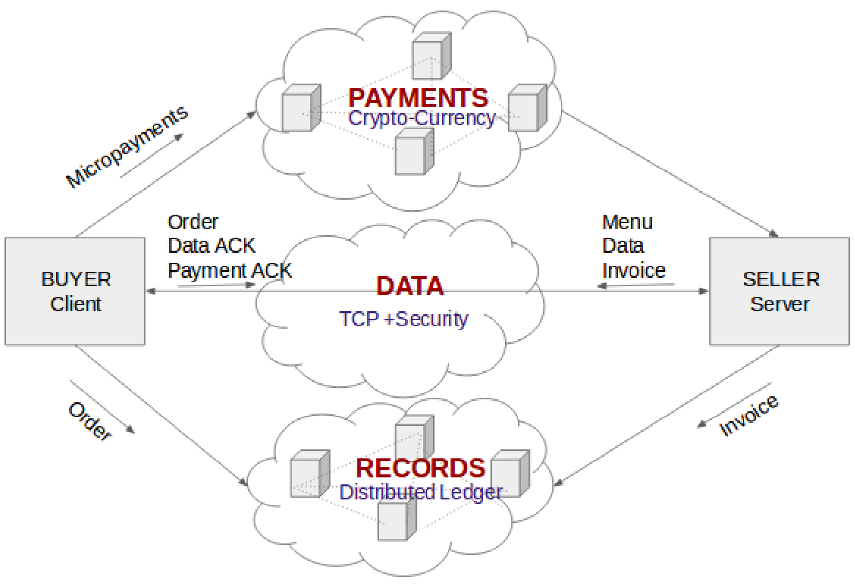


Fig. 1. The Tripartite Architecture of the Streaming Data Payment Protocol

a cryptocurrency and as a distributed ledger.

To our knowledge, SDPP is the first integrated data-payment-record protocol for IoT-type streaming data that leverages the traditional TCP/IP as well as the innovative decentralized blockchain/DAG-based distributed ledger technologies.

## II. Protocol Description

As shown in figure 1, SDPP has three main components to it, which we refer to as the data channel, the payment channel and the records medium. The data channel is implemented as a classic application layer client-server protocol utilizing TCP sockets. We incorporate basic security features at present without specifying a full implementation of TLS to keep it lightweight (but this could be modified in the future). The payment channel may be implemented using a crypto-currency (or any form of electronic payment, for that matter). The records medium is assumed to be implemented using an immutable distributed ledger. The payment and records can be combined and implemented on the same blockchain / DAG-based technology if needed but this is not essential. Given the nascent state of blockchain technology and the ongoing emergence of diverse choices, we do not specify exactly which technology should be used for the payment and records (however, as a proof of concept, we have implemented them both using IOTA).

An overview of how the data channel of the SDPP protocol evolves along with the times when transactions are made to send payments and create/store new records is shown in figure 2. At the data layer, for ease of implementation, and bearing in mind that it's relatively lightweight, we define simple JSON-formatted messages for different types of messages in the protocol. All messages use the general format described in listing 1. It is assumed that the server/seller's public key is known to the client/buyer through an out of band mechanism – this could be a QR code, for example, or from a public website or even the records medium.

```
{
    message_type: ""
    data: ""
    signature: ""
    verification: ""
 }
```

Listing 1: general format

The buyer is assumed to be the client and the seller the server. We will use these terms (buyer/client, seller/server) interchangeably. When the client contacts the server, it may optionally send a simple hello message as shown in listing 2.

```
{
    message_type: "hello"
    data: ""
    signature: ""
    verification: ""
 }
```

Listing 2: hello message from client

The seller responds to the initalization of the connection and the hello message (if used) with a menu, formatted as in the illustrative listing below. Effectively this menu provides the set of sensor topics available with the senors, and the unit price for each topic. The seller also lists possible and default payment options (currencies accepted) as well as the

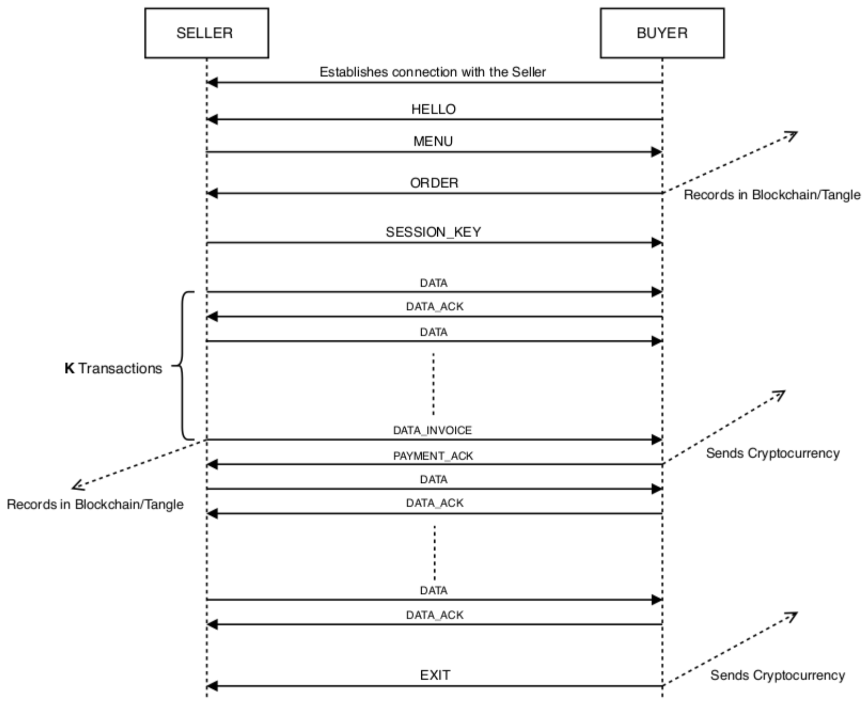


Fig. 2. Message Timeline for Streaming Data Payment Protocol

granularity of payment required (i.e. buyer must pay every $K$ data items). It also indicates if it would like the buyer to send signed acknowledgements, and also sends its own signature (for source authentication and the signed hash of the data for integrity).

The Menu message is responded to with an order message by the buyer. In addition to the order message in the data component, here the buyer will also post the order on the blockchain/tangle which acts as the records component. The order message includes which topic is being purchased, how many items of that topic are being purchased, the invoice address for the buyer, and its public keys to be used for signing and encryption, and finally, a signed hash for message integrity.

Once the order has been placed, the seller sends a symmetric session key encrypted by the buyer's public key, that can be used to efficiently encrypt all data in transit from this point onwards.

The data then starts to flow from the seller to the buyer. The data can be encrypted using the session key, and the signature of the seller applied to a hash of the data could also give message integrity guarantees. In response the buyer will send simple data ack messages (not shown here).

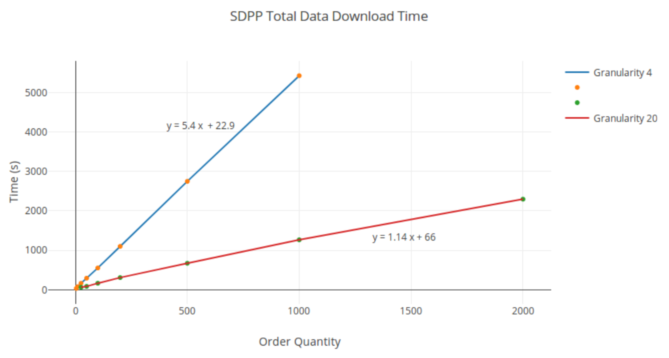


Fig. 3. Total time taken versus size of order

This is followed by a signed ACK from the buyer for each data message to confirm receipt of the data. The seller may resend data if it times out waiting for the ACK.

When the client / buyer wants, it can turn off the service and make a final payment to settle the bill and terminate the connection. The server can then go back to waiting for new connections.

## III. Implementation

We have begun to implement a reference SDPP client and SDPP server, both using Python. For both the payments channel and the records component we decided to use IOTA because of its low transaction fees, friendliness towards embedded implementation, and ease of use including relevant API's. Our SDPP client and server implementations currently still under

```
{
    message_type: "menu"
    data: {
              available_data: {
                         'gas': int
                         'temperature'': int
                    }
Default currency in which the rates are specified: "iota"
List of currencies accepted :  ["iota", "ethereum"]
Payment granularity : int (value of K)
Payment address of the seller : ""
Signature required in every data field or not: (boolean)
               }
    signature: signature of the seller
    verification: ""
 }
```

Listing 3: The Menu message from the Seller

```
{
message_type: "order"
data: {
        dataType: ""
orderQuantity: int
currencyUsed: ""
blockchain/tangle address of the buyer (to send invoices to): ""
Public key of the buyer for signature: ""
Public key of buyer for encryption: ""
           }
signature:  signature of the buyer
verification: transaction hash where the buyer has posted the order details

}
```

Listing 4: The order message from the buyer

```
{
    message_type: "session_key"
    data: "key to be used (encrypted by buyers public key)",
    signature : ""
    verification : ""
}
```

Listing 5: The session key message from the seller

```
{
    message_type: "data"
    data: "encrypted data"
    signature: "signature of seller"
    verification: ""
 }
```

Listing 6: The data being sent from the seller

```
{
    message_type: "data_invoice"
    data: {
            data: "encrypted data"
            invoice: "invoice asking for payment"
               }
    signature:
             verification: "transaction_hash where the invoice is recorded"
}
```

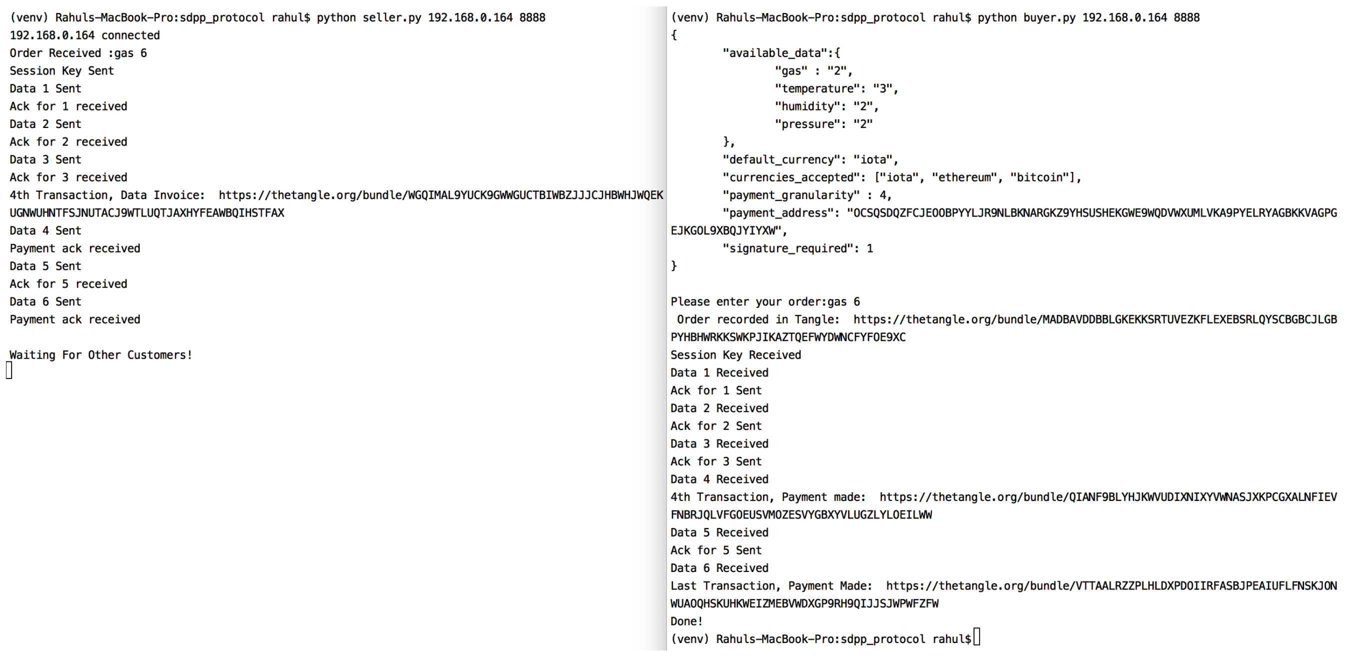
Listing 7: Every K messages, the seller sends a payment invoice, and also attaches it to a distributed ledger

```
{
{
    message_type: "payment_ack"
    data: "ack"
    signature: "signature of the buyer"
    verification: "transaction hash where the payment has been made"
}
```

Listing 8: Every K messages, the buyer sends a payment

```
{
    message_type: "exit"
    data:
    signature: "signature of the buyer"
    verification: "transaction_hash showing balances are paid by the buyer"
}
```

Listing 9: The buyer terminates the connection



```
(venv) Rahuls-MacBook-Pro:sdpp_protocol rahul$ python seller.py 192.168.0.164 8888
192.168.0.164 connected
Order Received :gas 6
Session Key Sent
Data 1 Sent
Ack for 1 received
Data 2 Sent
Ack for 2 received
Data 3 Sent
Ack for 3 received
4th Transaction, Data Invoice:  https://thetangle.org/bundle/WGQIMAL9YUCK9GWWGUCTBIWBZJJJCJHBWHJWQEK
UGNWUHNTFSJNUTACJ9WTLUQTJAXHYFEAWBQIHSTFAX
Data 4 Sent
Payment ack received
Data 5 Sent
Ack for 5 received
Data 6 Sent
Payment ack received

Waiting For Other Customers!
```

```
(venv) Rahuls-MacBook-Pro:sdpp_protocol rahul$ python buyer.py 192.168.0.164 8888
{
        "available_data":{
                "gas" : "2",
                "temperature": "3",
                "humidity": "2",
                "pressure": "2"
        },
        "default_currency": "iota",
        "currencies_accepted": ["iota", "ethereum", "bitcoin"],
        "payment_granularity" : 4,
        "payment_address": "OCSQSDQZFCJEOOBPYYLJR9NLBKNARGKZ9YHSUSHEKGWE9WQDVWXUMLVKA9PYELRYAGBKKVAGPG
EJKGOL9XBQJYIYXW",
        "signature_required": 1
}

Please enter your order:gas 6
 Order recorded in Tangle:  https://thetangle.org/bundle/MADBAVDDBBLGKEKKSRTUVEZKFLEXEBSRLQYSCBGBCJLGB
PYHBHWRKKSWKPJIKAZTQEFWYDWNCFYFOE9XC
Session Key Received
Data 1 Received
Ack for 1 Sent
Data 2 Received
Ack for 2 Sent
Data 3 Received
Ack for 3 Sent
Data 4 Received
4th Transaction, Payment made:  https://thetangle.org/bundle/QIANF9BLYHJKWVUDIXNIXYVWNASJXKPCGXALNFIEV
FNBRJQLVFGOEUSVMOZESVYGBXYVLUGZLYLOEILWW
Data 5 Received
Ack for 5 Sent
Data 6 Received
Last Transaction, Payment Made:  https://thetangle.org/bundle/VTTAALRZZPLHLDXPDOIIRFASBJPEAIUFLFNSKJON
WUAOQHSKUHKWEIZMEBVWDXGP9RH9QIJJSJWPWFZFW
Done!
(venv) Rahuls-MacBook-Pro:sdpp_protocol rahul$
```

Fig. 4. Console output showing inputs and outputs at the buyer and seller ends of Streaming Data Payment Protocol

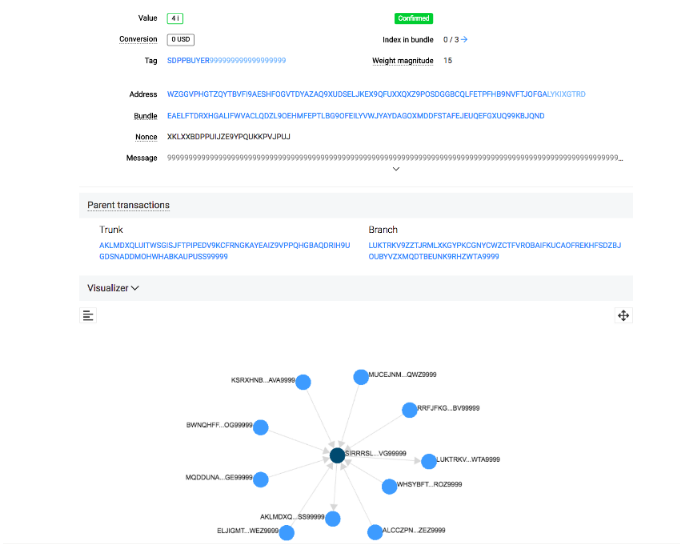

Value 4 i
Confirmed
Conversion 0 USD
Index in bundle 0 / 3
Tag SDPPBUYER9999999999999999
Weight magnitude 15
Address WZGGVPHGTZQYTBVFI9AESHFOGVTDYAZAQ9XUDSELJKEX9QFUXXQXZ9POSDGGBCQLFETPFHB9NVFTJOFGALYKIXGTRD
Bundle EAELFTDRXHGALIFWVACLQDZL9OEHMFEPTLBG9OFEILYVWJYAYDAGOXMDDFSTAFEJEUQEFGXUQ99KBJQND
Nonce XKLXXBDPPUIJZE9YPQUKKPVJPUJ
Message 999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999...
Parent transactions
Trunk
AKLMDXQLUITWSGISJFTPIPEDV9KCFRNGKAYEAIZ9VPPQHGBAQDRIH9U
GDSNADDMOHWHABKAUPUSS99999
Branch
LUKTRKV9ZZTJRMLXKGYPKCGNYCWZCTFVROBAIFKUCAOFREKHFSDZBJ
OUBYVZXMQDTBEUNK9RHZWTA9999
Visualizer

Fig. 5. Web view of transaction on the IOTA Tangle

development (we have a working prototype but are still adding key features), and will be available as open source online at http://github.com/anrgusc/SDPP

Although it is a prototype that we are still working on extending, just as an indication of how lightweight the protocol could be, our current basically functional version of the seller (server) side code is only 203 source lines of code, while the buyer (client) side code is presently only 169 source lines of code. While there are other libraries being utilized and room for further refinement, it underscores that SDPP can in principle be easily deployed on relatively more resource-constrained compute nodes typical of IoT, at least edge devices such as raspberry-pi nodes or mobile devices. If we were to consider really resource-constrained devices such as motes, it may be worthwhile developing an even more lightweight version of this protocol that eschews JSON for bit-codes and TCP for UDP, with the interaction with the blockchain or tangle happening through remote gateway nodes; we leave such a resource-constrained design and implementation to future work.

## IV. Demonstration and Results

In figure 3 we show how the total time for the entire protocol varies as a function of the size of the stream (number of data items ordered). A roughly linear trend is observed, with the slope determined by the granularity parameter. As expected, a higher granularity for payment and acks can significantly reduce the total data download time. The trade-off, however, is that potentially a larger number of data items may be left unpaid for with a higher granularity, in case of early termination without payment.

Figure 4 shows a sample set of console outputs based on the SDPP protocol running, and figure 5 shows a real-time view of one of the transactions as it appears on the IOTA tangle. In our experiments, however, we observed very high variance in confirmation times on the tangle, taking on the order of several minutes or even longer sometimes, suggesting that it is not a sufficiently mature technology just yet (as of April 2018).

## V. Conclusion and Future Work

We have presented SDPP, an application-layer protocol for streaming data with payment processing and auditable records utilizing emerging crypto-currencies and blockchain/tangle-based distributed ledger technologies.

Our reference implementation of SDPP is still work in progress, with ongoing efforts aimed at a first, functioning, release soon while we continue to add features including security, full implementation of the JSON spec for message types described in this paper. We also plan to make a multi-threaded version of the server, and have it evaluated in situ on our university's outdoor IoT testbed with real sensor data. We would also like to have a reference implementation of an application with an SDPP client for Android phones. And we will consider extending the reference implementation to other popular blockchain platforms including Ethereum and Hyperledger fabric (which could be used to store the records).

There are some design and architectural questions we have yet to explore fully - one of these is whether it is beneficial to apply standard transport layer security (TLS) for providing encryption and message integrity functionality (akin to how HTTPS is enabled by TLS added to standard HTTP) instead of our current approach of integrating those capabilities directly into SDPP.

Finally, one comment on SDPP is that it is a plain old-fashioned client-server protocol. A different approach that is gaining in popularity for IoT is a publish-subscribe approach. We are currently working on extending the ideas in SDPP to a publish-subscribe model as well.

## References

[1] H. Schaffers, N. Komninos, M. Pallot, B. Trousse, M. Nilsson, and A. Oliveira, "Smart cities and the future internet: Towards cooperation frameworks for open innovation," in *The future internet assembly*. Springer, 2011, pp. 431–446.

[2] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Sensing as a service model for smart cities supported by internet of things," *Transactions on Emerging Telecommunications Technologies*, vol. 25, no. 1, pp. 81–93, 2014.

[3] G. Birmpas, C. Courcoubetis, I. Giotis, and E. Markakis, "Cost-sharing models in participatory sensing," in *International Symposium on Algorithmic Game Theory*. Springer, 2015, pp. 43–56.

[4] B. Krishnamachari, J. Power, S. H. Kim, and C. Shahabi, "Iot marketplace: A data and api market for iot devices," https://i3.usc.edu/index.php/about/key-documents/, accessed: 2018-04-19.

[5] "Iota data marketplace," https://data.iota.org/, accessed: 2018-04-19.

[6] S. Popov, "The tangle," https://iota.org/IOTA_Whitepaper.pdf, accessed: 2018-04-19.