PayFlow: Micropayments for Bandwidth Reservations in Software Defined Networks

David Chen, Zhiyue Zhang, Ambrish Krishnan, Bhaskar Krishnamachari
Viterbi School of Engineering
University of Southern California
Los Angeles, CA 90089
Email: {chen749, zhiyuezh, ambrishk, bkrishna}@usc.edu

***Abstract*—We present PayFlow, a fine-granularity QoS micropayment system that allows end devices in a software-defined network to make and pre-pay for guaranteed bandwidth reservations for their flows within the network for an arbitrary period of time. PayFlow combines payments using digital currency and storage of transaction records in a distributed ledger with queue-based QoS management using software-defined networks. While the PayFlow architecture is agnostic to the choice of digital currency, ledger technology and SDN platform used, we present a proof of concept implementation of PayFlow using OpenFlow and the IOTA cryptocurrency and distributed ledger, that we evaluate using the Mininet emulator.**

***Index Terms*—Micropayments, QoS, SDN, OpenFlow, Digital Currency, Distributed Ledger, IOTA**

I. Introduction

We address in this paper the question of how users or devices in a network could pay at a fine granularity for quality of service (QoS) guarantees. Specifically, we consider and address how users or devices could pay for bandwidth reservation for their flows through switches in a network for an arbitrary time duration, on the order of seconds. Our proposed system leverages and combines two significant advances of the past decade - software defined networking with centrally controlled open-switches [1] and blockchain, or more broadly, distributed ledger technology for making electronic payments and storing pertinent transaction records in an immutable manner [2]–[4].

Traditionally, such payments have been challenging at a fine-granularity because of the overhead associated with establishing a trusted relationship between network users and administrators. Consequently, in practice, typically network users can only request higher quality service on an aggregate basis over a long period of time after establishing a credible trust-based customer relationship (e.g. paying monthly subscription for a particular tier of service that allows transportation of a specified maximum volume of data at a high rate, with the paid subscription itself subject to prior credit checks into ensure the network provider can be paid in a timely manner.)

We believe there is a benefit to a more agile approach to economic interactions over bandwidth provision that allows participants (network users and providers) without prior trusted relationships to transact with each other at a fine temporal granularity – over short intervals on the order of seconds – and fine device granularity — between end host devices and software-defined network controllers in a completely automated manner (i.e., without requiring human intervention for each transaction). This will not enable more frictionless interactions with personal devices carried by mobile or roaming users, it will also enable more dynamic, decentralized, autonomous interactions between end host devices and the networks they connect to. We believe such autonomous interactions between end devices and networks will be an increasing need for the oncoming age of IoT networks where the ratio of devices to the humans that administer and own them is likely to increase by several orders of magnitude compared to today.

We present PayFlow, a system that allows end hosts in a software-defined network to make and prepay for bandwidth reservations for their flows for a defined period of time of arbitrary length, on the order of seconds or less. We design PayFlow to work with software defined networks, defining a standardized way for hosts to interact with the switch controller to make these reservations with micropayments for relevant transactions that could be made using a cryptocurrency and being recorded on an immutable distributed ledger. The architecture and design of PayFlow is intended to be modular and agnostic to the particular choice of blockchain or distributed ledger protocol used and the particular software defined network system or controller. This modularity and abstraction is particularly important as digital currencies, distributed ledger technologies and software-defined networking systems continue to evolve technically and mature and gain in adoption. However, to make our ideas concrete, we demonstrate it in this work through a reference implementation that uses the IOTA cryptocurrency and distributed ledger technology and the OpenFlow SDN controller.

Our present implementation is limited to a single domain, i.e. a single network provider. As such it is most relevant to settings such as towns, industrial-parks, airports, malls, campuses or cloud networks where a single provider is managing traffic and responsible only for QoS provision for all users within its own domain. We are currently working on generalizing this work to multi-domain systems, with more complex economic interactions crossing multiple trust boundaries, certainly a harder problem but one for which we believe the present work may be a stepping stone or buiding block.

The rest of the paper is organized as follows. In section II

we present and discuss some relevant prior work. In section III we present a high level overview of our proposed system architecture. In section IV, we describe the various messages exchanged between end hosts and the SDN controller in PayFlow. In section V we present our reference implementation of PayFlow using IOTA and OpenFlow SDN. In section VI, we demonstrate the functioning of this implementation using the Mininet emulator. Finally, we present concluding thoughts including our plans and ideas for future work in section VII.

II. Related Work

QoS management in networks has a rich history with a lot of attention paid to the subject particulary in the 1990's as multimedia applications started to emerge as a significant use case for data networks [5]–[8].

In the past decade, a major revolution in networking has been the introduction of programmable software-defined networks, most prominently the OpenFlow architecture which separates the distributed data plane from a more centralized control planes to allow for more flexible software-based management of networks and allowing greater dynamic reconfigurability of network switches [1], [9], [10]. More recent works have therefore tackled QoS management specifically in the context such software-defined networks. Egilmez *et al.* [11] propose OpenQoS an OpenFlow controller design for multimedia delivery with end-to-end Quality of Service (QoS) support that optimizes routes to fulfill the required QoS. Akella and Xiong consider SDN-based bandwidth allocation in a cloud network [12] to meet QoS requirements for priority cloud users using Open vSwitch. However, like most other work on SDN, these papers do not describe an micropayment mechanism to go hand in hand with QoS management.

We build on the prior literature on QoS management in this work; in particular, the bandwidth reservation over a path of routers on a per-flow basis that we adopt could be seen as an extension of integrated services [5]. While integrated services was originally proposed to be used with a specific distributed protocol such as RSVP, in our case we are implementing it for a single-domain network using the centralized control capabilities inherent in the OpenFlow SDN system. Going beyond prior work on QoS management in SDN, moreover, we explicitly address and show, to our knowledge for the first time, how micropayments can be integrated with bandwidth reservation in single-domain networks.

There has been prior work on incentives and payment based routing in the context of mobile ad hoc networks. As far back as 2000, Buttyan and Hubaux [13] proposed the use of a digital currency called "nugglets" to provide incentives for routing in a mobile ad hoc network, and in our group's prior work in 2006 [14] we explored the use of price-based incentives to generate reliable multi-hop routes in lossy wireless networks. However, these early works did not have the benefit of the invention and growing adoption of modern cryptocurrencies which can eliminate the double-spend problem without requiring centralized trusted third parties, a likely reason that these early proposals have not seen adoption in practical systems. PayFlow is designed with modern distributed ledger/blockchain-based cryptocurrencies such as Bitcoin [2], Ethereum [3], IOTA [4] in mind. We borrow some lessons from our recent work on implementing the streaming data payment protocol (SDPP) [15] which is a client-server application-layer protocol that allows clients to pay servers for streaming data in real-time.

III. PayFlow System Architecture

PayFlow is a software defined micropayment for QoS system that allows clients on end-hosts to connect to a corresponding server on an SDN switch controller to find out about bandwidth reservation options along with their price and place an order for a bandwidth reservation for a specified duration on switches that its flows will traverse. Upon payment, the switch controller ensures that the desired reservation is made, and revokes it after the specified duration.

The architecture of PayFlow is shown in figure 1. PayFlow uses a PAYMENTS channel to make digital currency based micropayments and a RECORDS channel to store all relevant transaction records in an immutable ledger. These are similar to the corresponding channels in Streaming Data Payment Protocol (SDPP) [15]. However, unlike that prior work, PayFlow also introduces two additional channels (REQUEST, CONTROL). The REQUEST channel is used by the client on the end host to exhange messages with the server on the SDN controller in order to learn about QoS options and pricing and to place the bandwidth reservation order. The CONTROL channel is essentially using the control channel of the SDN to implement the requested reservation once payment for it has been made. Lastly, the DATA channel in the PayFlow architecture is the data plane of the network.

IV. PayFlow REQUEST Messages

To make a QoS request on the network, the client contacts the local controller through the REQUESTS channel. A typical message exchange can be seen in figure 2.

The REQUESTS channel in PayFlow is implemented using classic client-server software using JSON format. We describe some of the details of the JSON-formatted messages used for the REQUESTS channel here. A typical message format is shown in listing 1.

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

Listing 1. General message format

First the client establishes TCP connection to the server on port 6113 and sends a HELLO message (listing 2). The server then sends a MENU (listing 3) with different QoS options and their price.

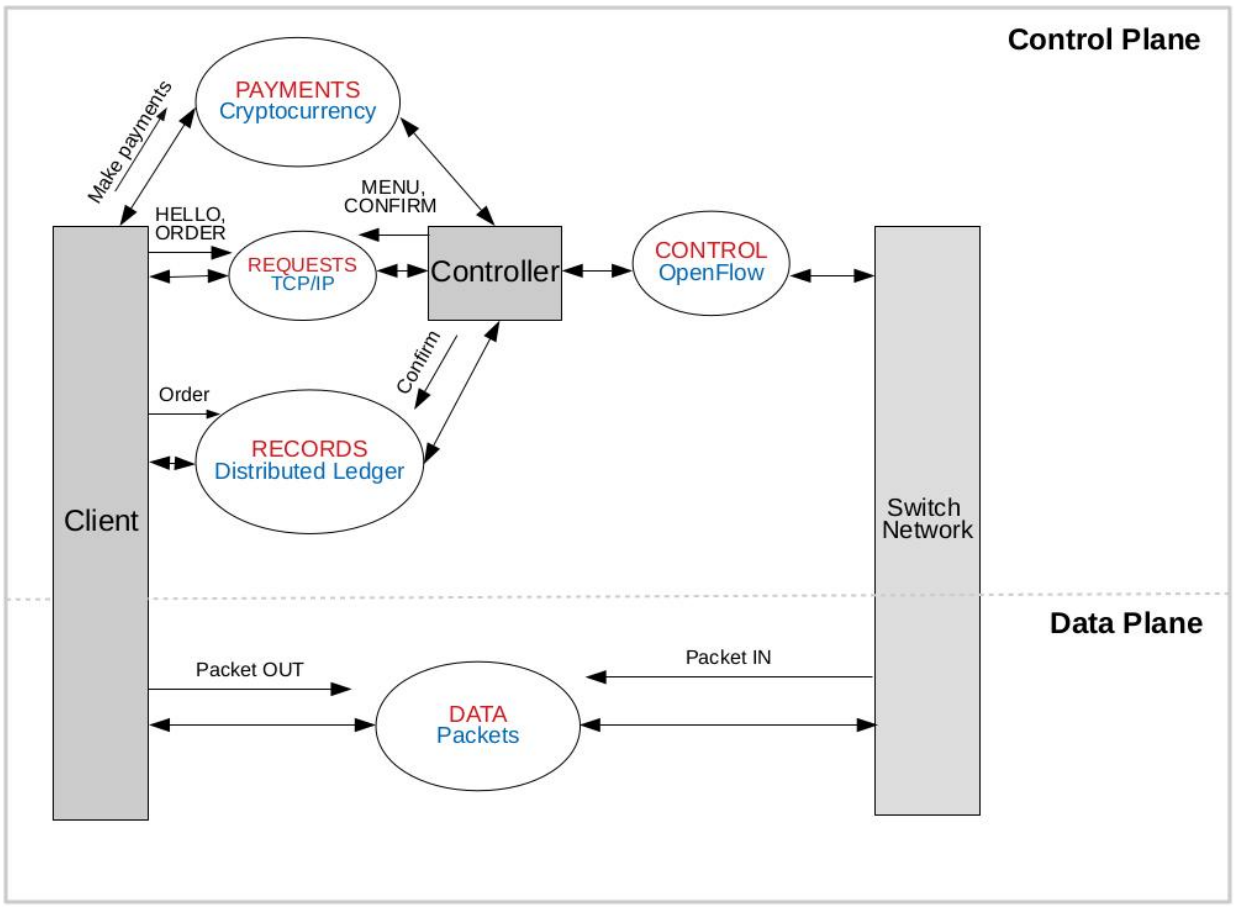

Figure 1. Architecture of PayFlow

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

Listing 2. HELLO message

```
{
    "message_type":"MENU",
    "data": {
            "menu": {
            "level0":"price of level 0",
            "level1":"price of level 1",
            "level2":"price of level 2",
            "level3":"price of level 3"
        },
        "unit":"s",
        "currency":"iota"
    },
    "signature":"signature of seller",
    "verification":""
}
```

Listing 3. Example MENU message (pricing in seconds)

The MENU message consists of two main parts. First, the data consists of the various levels of QoS and their prices in a certain digital currency (IOTA in case of our reference implementation). Second, it contains the wallet address of the controller that the client needs to send the payment to. The MENU message also includes a signed digest.

The client can then choose to purchase a certain level of QoS or decline. If it chooses to purchase, it then it must submit a transaction through the payments channel for the amount specified.

Once the client has made the payment, it has to send an ORDER message (listing 4). Note that IP address 1 and IP address 2 are interchangeable as of now. It is possible that in the future one way QoS (download/upload) can be implemented. The controller can check the record transaction ID to verify the IP address pairing and service level. Then, the controller can check for payments from the client address for verification of payment. In addition, the controller must also confirm by checking the network that the requested QoS can be implemented (an example is given in section 3). After the controller has verified the order and the payment, it sends OPENFLOW ENQUEUE messages to the necessary switches on the network in order enqueue packets between IP address 1 and IP address 2 in a priority queue. After the time specified

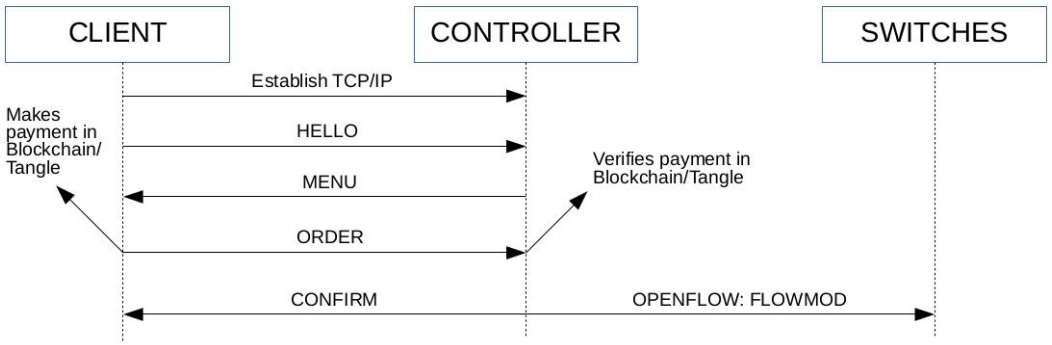

Figure 2. Protocol messages in PayFlow

by the order has elapsed, the controller reverts this action.

```
{
    "message_type":"ORDER",
    "data": {
        "level":"requested level",
        "currency":"currency used",
        "address":"client address",
        "public-key":"client key",
        "ip1":"first IP",
        "ip2":"second IP",
        "time":"duration in units"
    },
    "signature":"signed digest",
    "verification":"transaction ID"
}
```

Listing 4. ORDER message

V. Implementation with IOTA and OpenFlow

For the implementation, we decided to write the controller-side code for PayFlow using the POX framework (version 0.5.0) using OpenFlow 1.0. The documentation and source code can be found at https://github.com/noxrepo/pox and https://www.openvswitch.org/ respectively. The POX modules used include the discovery and host tracker modules. We used both of these modules for topology discovery. For interacting with the IOTA tangle, we used the Pyota package (version 2.0.6), found at https://github.com/iotaledger/iota.lib.py

We are making our implementation of PayFlow using OpenFlow and SDN available publicly as open source, online at https://github.com/anrgusc/PayFlow. The controller-side code for PayFlow is divided into three parts. First, the PriceDynamicRequestsController is the main class that gets instantiated when the controller starts up. For every switch that connects to the controller, this class instiantiates a separate switch object (detailed next). In addition, it also handles events like a service level being changed or a host tracker event. Next, the PriceDynamicPaymentsSwitch has the logic to learn MAC address to port mappings using a standard learning switch logic. It also has the logic to send specific OpenFlow messages such as enqueue, output and flood. For packets with no known flow and no service level associated with it, they are sent onto the interface on the default queue using the MAC-to-port mappings. Lastly, QoSBroker is the class that handles client request. A TCP server is started by the QoSBroker class and it handles sending protocol messages. The basic functionality of the QoSBroker includes sending the MENU when a HELLO message is received and raising an event when a valid ORDER message is received.

For switches we used OpenVSwitch version 2.9.0 with no modification. Queues are created using the ovs-vsctl utility on Linux. When the network starts, a static amount of queues with different maximum bandwidths are created on each interface. As per the OpenVSwitch standard, these queues use Linux's hierarchical token bucket queueing discipline (see https://linux.die.net/man/8/tc-htb).

On the client side, we created a sequential Python program that uses the same Pyota package for Tangle interaction. It sends a HELLO message to the controller, displays the MENU that is sent back and prompts the user for the order. The user can then type in the order, including the two IP addresses and the service level. After obtaining input from the user, the program makes an API call using Pyota to send the transaction and the records message to the Tangle. The records message consists of the two IP addresses of the service, the service level, the duration and the transaction ID of the payment.

VI. Demonstration using Mininet

To demonstrate PayFlow, we used the Mininet emulator (version 2.2.2) on a Xiaomi Notebook Air with an Intel i5 processor. The test topology we used can be shown in Figure 3. To test the system, we first instantiated the toplogy on Mininet and started up the POX controller. Then, a queue setup script is run in order to configure the queues using ovs-vsctl. By default, the controller will route all packets through the default queue on each interface. To visually demonstrate the QoS changes, we used iperf (version 2.0.10). An iperf server is started on hosts three and four on port 4000.

In the first experiment involving a single flow, shown in figure 4, an iperf client is run from host h1, connecting to

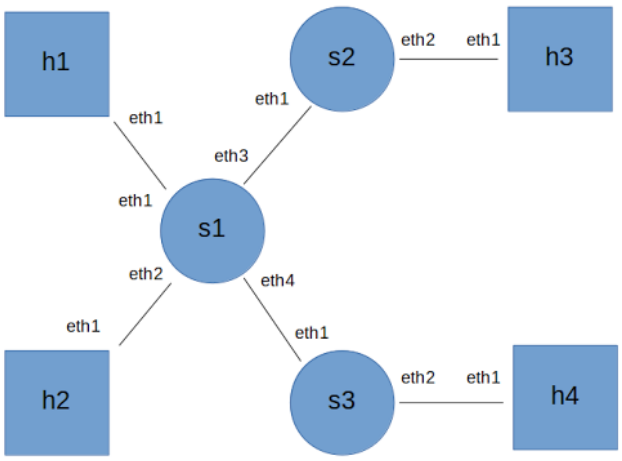

Figure 3. Topology used for mininet evaluation of payflow

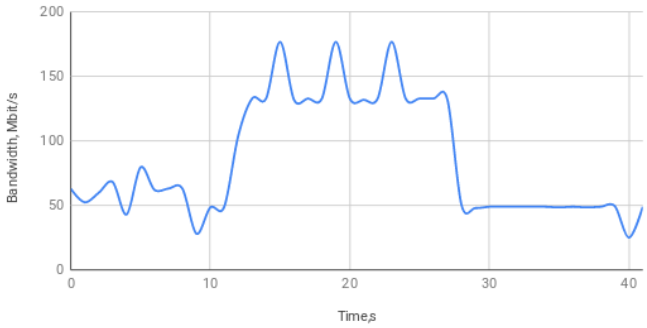

Figure 4. Demonstration of PayFlow with Single Flow

host h3. Measurements were taken every second for a minute. The client program was run during the test, a little after 10 seconds into the test, requesting a level two service between host h2 and host h3 for 15 seconds. As can be seen, during this period of time, the flow receives a higher bandwidth of about 150 Mbps and then reverts back to the default of 50 Mbps.

We also show an example involving two concurrent flows in figure 5. Here, in addition to an iperf client from h1 to h3, an iperf client is also started on h2, connecting to host h4. The same iperf parameters were used. The client program was ran once a little after 20s to request level two service between hosts h1 and h3 for 30 seconds, and again at around 45 seconds, to request level two service between hosts h2 and h4 for 15 seconds. Both flows are seen to receive the higher level of bandwidth during their corresponding reservation period.

Here we have presented a couple of sample scenarios to demonstrate the operation of PayFlow as a proof of concept. We are currently undertaking more systematic evaluations of its performance in terms of metrics such as latency, scalability with respect to various system parameters.

VII. Summary and Future Work

We have presented PayFlow, a system to enable micro-payments for flow bandwidth reservations in software de-

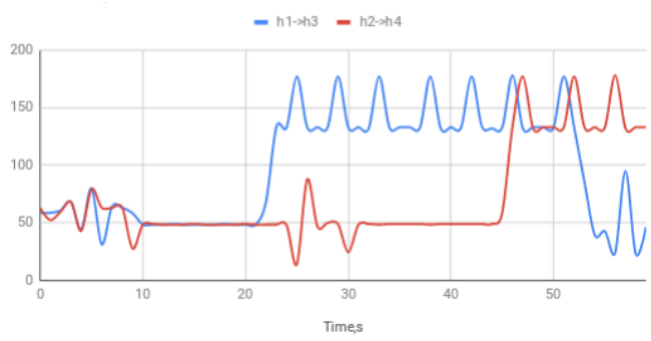

Figure 5. Demonstration of PayFlow with Two Flows

fined networks. PayFlow has been implemented using the IOTA cryptocurrency and distributed ledger and OpenFlow (specifically the POX controller framework in conjunction with OpenVSwitch). We are making our implementation of PayFlow available to the research community as an open source system, online at https://github.com/anrgusc/PayFlow.

In the future we would like to conduct a more thorough evaluation of PayFlow, including on a real experimental testbed. We would also like to work on the harder problem of multiple domains interacting with each other on an economic basis to allow flows to make payments for bandwidth reservations when their routes span different SDN-controlled domains. We believe that a decentralized architecture for such a system could be designed using the present architecture of PayFlow as a building block.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *White Paper*, 2008.

[3] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.

[4] S. Popov, "The tangle," *URL https://iota. org/IOTA_Whitepaper. pdf*, 2017.

[5] R. Braden, D. Clark, and S. Shenker, "Integrated services in the internet architecture: an overview - rfc 1633," IETF, Tech. Rep., 1994.

[6] Z. Wang and J. Crowcroft, "Quality-of-service routing for supporting multimedia applications," *IEEE Journal on selected areas in communications*, vol. 14, no. 7, pp. 1228–1234, 1996.

[7] P. Ferguson and G. Huston, *Quality of service: delivering QoS on the Internet and in corporate networks*. Wiley New York, 1998, vol. 1.

[8] N. Giroux and S. Ganti, *Quality of Service in ATM networks: State-of-the-art traffic management*. Prentice-Hall, Inc., 1998.

[9] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[10] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[11] H. E. Egilmez, S. T. Dane, K. T. Bagci, and A. M. Tekalp, "Openqos: An openflow controller design for multimedia delivery with end-to-end quality of service over software-defined networks," in *Asia-Pacific Signal & Information processing association annual summit and conference (APSIPA ASC)*, 2012.

[12] A. V. Akella and K. Xiong, "Quality of service (qos)-guaranteed network resource allocation via software defined networking (sdn)," in *IEEE 12th International Conference on Dependable, Autonomic and Secure Computing (DASC)*, 2014, pp. 7–13.

[13] L. Buttyán and J.-P. Hubaux, "Enforcing service availability in mobile ad-hoc wans," in *First Annual Workshop on Mobile and Ad Hoc Networking and Computing (MobiHOC)*. IEEE, 2000, pp. 87–96.

[14] H. Liu and B. Krishnamachari, "A price-based reliable routing game in wireless networks," in *Proceedings of ACM workshop on Game theory for communications and networks*, 2006.

[15] B. K. Rahul Radhakrishnan, "Streaming data payment protocol (sdpp) for the internet of things," *Proceedings of BIoT: The 1 st International Workshop on Blockchain for the Internet of Things, held in conjunction with IEEE Blockchain,Halifax, Canada*, 2018.