Decentralized IoT Peer-to-Peer Protocol (DIPP)

A P2P protocol for enabling IoT Device-to-Device (D2D) communication

Sirajus Salekin
Computer Science, Earlham College
Richmond, Indiana 47374
ssalek14@ealham.edu

ABSTRACT

Internet of Things (IoT) devices have gained significant popularity and widespread use over the past few years. The ubiquity of such devices means that the number of connected devices to the Internet is increasing at a rapid rate. Proliferation of such devices also increased the surface for attack. However, research interests in issues that affects users the most such as security, data ownership are not matched with the expansion of these technologies. Focusing on these issues are important from a consumer perspective since there are technical, economical and social implications of using an Internet-enabled device. In addition to security, privacy and ownership challenges, there are rooms for improvement in terms of efficiency when one considers the fact that a good portion of data can be processed locally instead of directing them to the cloud infrastructure. Peer-to-Peer (P2P) models have the potential to address those issues in an IoT ecosystem. Recognizing the above, we are proposing an application layer protocol called the Decentralized IoT Peer-to-Peer Protocol (DIPP). It is decentralized by the virtue of having a P2P model, can enable heterogeneous devices, aimed at better utilization of resources, and attempts to address some privacy and security concerns.

ACM Reference format:
Sirajus Salekin. 2016. Decentralized IoT Peer-to-Peer Protocol (DIPP). In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 7 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Peer-to-peer network is an alternative to the dominant client-server network architecture. According to a client-server model, clients rely on the server for all the contents, and a majority of the services. In a peer-to-peer environment, a relationship of the aforementioned sort does not exist. The peers, instead, act as both clients and servers, providing services and contents to other peers in the network. An obvious upside of peer-to-peer architecture is that it avoids a single point of failure: the server. In the event of server crash, a client-server model is useless since the server is the central entity that is required for the network to function. A peer-to-peer network, on the other hand, would still function if a peer fails, since the remaining peers are capable of providing the network with contents and services.

P2P networks are usually associated with file sharing applications such as Napster, Gnutella, or BitTorrent. However, the applications of P2P protocols are more widespread than they appear to be. P2P networks can be quite useful in device-to-device (D2D) communications. While P2P models are usually more resilient than client-server ones in case of failures and security compromises, absence of a central authority in a P2P network often translates into inefficient search and discovery. Services like those don't scale well in P2P but do so pretty well in client-server. However, P2P models still have large rooms for improvements and significant potential in an increasingly more connected world.

Internet of Things (IoT) are usually small connected devices. Compared to a full-fledged computer, they have a smaller form factor, smaller memory, less processing power and constrained supply of electricity. The range of IoT devices is heterogeneous: from an RFID tag to a Google Home–all theses device are considered to be within the IoT ecosystem. IoT devices are very useful when it comes to collecting data, automating task and environments. While using peer-to-peer on IoT devices can release big waves of innovations, doing so also creates design and implementation challenges that are usually not encountered in client-server models. In this paper, we attempt to tackle on some of those challenges.

2 PROBLEM STATEMENT

The widespread adoption of small devices capable of connecting to the Internet provides consumers with enormous benefits. At the same time, challenges of multiple domains are inherited in the process, and at the intersection, new kinds of problems arise. Specifically, the issues of privacy, security, and data transfer in a network of IoT devices caught our attention. Speaking of security, a vast number of IoT devices are placed around places where sensitive data are produced. For instance, location-awareness of some IoT devices make them quite useful, but the consequences of security compromise of such devices–for example, a wearable tracker–can be very expensive.

Secondly, as we integrate more and more devices into our lives, we are heading towards an economy where data represents and translates into products and profits. This scenario is arguably inevitable as we automate more and more tasks. It is important, though, that we attempt to draw the boundaries at where ownership of a particular entity ends and another starts. The unprecedented level of data collection of large technology firms such as Amazon, Google, Facebook etc., further aggravated the situation.

Conference'17, Washington, DC, USA

DOI: 10.1145/nnnnnnn.nnnnnnn

Issues related to content ownership should be a key focus as consumers introduce more connected devices in their household and workplace, and produce more data for the Internet.

When considering privacy challenges, it is evident that privacy and data ownership are closely interconnected. Ziegeldorf observed in [18] that identification, tracking and profiling are some of the primary threats concerning IoT privacy. They also mentioned that as the IoT ecosystem evolves, privacy issues arise in ways we can not anticipate. IoT ecosystem is not standardized because of the large number of participants. So manufacturers have to focus primarily on their products functioning correctly. Consequently, consumer concerns–especially the ones that have no effect on profits–do not get serious considerations from the device manufacturers or service providers. This void motivated us to propose a platform that enable device-to-device communication and emphasizes primarily on the issues that affect average consumer: privacy, security, data ownership, and efficient resource utilization.

3 PAST RESEARCH

In light of our research interest, we will discuss existing application layer protocols. Notable examples of such protocols include: MQTT, CoAP, HTTP, XMPP and SMQTT. Most of these protocols use either TCP or UDP, or a combination of the two as their transport layer protocols. [1] argues that TCP as a transport layer protocol is inadequate for IoT due to long connection set-up, high overhead for small data exchange, and expensive buffering at the ends. We shall see how the choice of TCP or UDP affected the aforementioned application layer protocols. We will also discuss some data link layer protocols later in this section.

The Constrained Application Protocol (CoAP) appears to be the de-facto choice when it comes to application layer protocols. IETF's Constrained RESTful working group (CORE) specified this protocol considering the low power consumption requirement in order to provide a RESTful interface for HTTP clients and servers. CoAP uses UDP as transport layer protocol and introduces two layers to provide reliability [13]. Although CoAP was well-suited for power-constrained IoT devices, it does not provide security on its own, and neither do its transport layer counterpart, UDP. There have been efforts to replace UDP with a secure protocol such as Datagram Transport Layer Security (DTLS) for CoAP; however, either their compatibility with CoAP is questionable, or they are still under development[6].

Another well-known application layer protocol is MQTT, which was originally designed by IBM in 1999. It has a publisher-subscriber model and the exchanges are facilitated through a broker, i.e., a server. Sensors and lightweight devices typically act as publishers and publish their data. Applications and devices that are subscribed to different topics receive the data published data through the server. MQTT protocol is well-suited for IoT devices, for its low overhead, [6] and economical use of battery. This protocol has shown to be effective at keeping the energy consumption low, despite using TCP as the underlying protocol. [16] In terms of security, MQTT and CoAP are two majorly adopted protocols, and they are compared and experimented frequently on different matrices. They tend to show comparable measurements in latency, packet loss, QoS etc.

The Extensible Messaging and Presence Protocol (XMPP) was once popular but declined because of its incompatibility with newer technologies. However, it is gaining more attention lately because of its applicability to IoT devices. XMPP was originally devised for real-time message exchange. [2] Similar to MQTT, it uses TCP as the Transport layer protocol and provides no QoS on its own other than the ones provided by TCP. One drawback of XMPP is that it transfers data in XML format, which usually requires heavyweight software because of nested structures.

There exists other known application layer protocols, but they are either a specialized version of the aforementioned three, or not as suitable for small constrained devices. For instance, SMQTT (Secure MQTT) is designed to enhance the security of MQTT by adding encryptions. [14]. Another similar protocol is AMQP, which has its use in financial industry.

The protocols discussed above essentially follow client-server architectures, so they rely on the server for successful functioning of the network. As we mentioned earlier, while such model makes control and data aggregation easier, a single point of failure is still present, and these protocols are designed in a way that aggregated data ultimately moves to the cloud.

3.1 Sensor Networks

Internet of Things (IoT) devices became a buzzword in recent years, but the practice of using sensors to automate environments existed before the advent of IoT. Wireless Sensor Networks (WSN) used in those cases, where the sensors would collect data, usually with the help of a sink, and then sent somewhere else to process.

Wireless Sensor Network implementations and technologies have come a long way primarily because of their usefulness. Industrial and home automation are two major areas where sensor networks were used reliably for simple data collection. Attempts to implement such technologies on IoT devices have been important stepping stones in the evolution of IoT. However, challenges in a network of IoT devices are more varied and areas for improvement are much larger than in Wireless Sensor Networks. While aggregating data can be quite convenient in a WSN, filtering, analyzing and processing those data is not. Still, IoT networks will benefit from acquiring a significant portion of WSN technologies, and we will see that happen more and more in future. In fact, the authentication we adopted for DIPP was originally intended for sensor nodes in a WSN.

3.2 Data Link Layer Protocols

The beginning of IoT can be traced to RFID tags. Today, devices such as Google Home or Amazon Alexa is part of the IoT ecosystem. One common characteristics virtually all IoT devices share is constraints on power consumption. This restriction in turn affected how these devices communicate since existing radio technologies such as WiFi or Bluetooth consume considerable amount of energy to run. Realizing these needs, new technology such as ZigBee, Thread, Bluetooth Low Energy (BLE), 6LoWPAN etc were specified and standardized. These standards and protocols are geared towards constrained devices, which was categorized by IETF based on available computing resources[3].

ZigBee is based on MAC and PHY layer standardized by IETF, and they are useful for automating a low powered, resource constrained Wireless Sensor Network. Despite the adoption of ZigBee by a large group of devices, devices with different radio technologies are difficult to connect to a ZigBee network.

Z-Wave is a protocol originally developed by Zensys for the purpose of home automation. The radio packets works primarily at 900 Mhz, and in some cases at 800Mhz. Z-Wave is specifically designed for resource- constrained devices, and the data transfer rate is on the lower end, at 100kbps In addition, a Z-Wave network needs a controller node to initiate, route messages and send commands through other nodes to an end node. Non-controller devices are called slave nodes, which don't have the capacity to initiate messages on their own. Thread is another similar technology, which is agreed upon by a number of manufacturers.

Despite the existence of big alliance, interoperability has not been achieved because large number of IoT manufacturers that creates and follow their own. Standard bodies across the globe have been working on bringing all these devices under a common interface, but progress has been slow because such decisions are invariably entangled with commercial and political interests in addition to being technical challenging.

We will end this section with mentioning a very recent phenomena. It is fog computing, which attempts to address the resource utilization problem we discussed earlier. The idea of fog computing came from cloud computing, in the most literal sense. It proposed that instead of sending all the data to cloud, a major part of which is unnecessary, some data analysis will be done on the edge level. This introduces a layer between the cloud and the local IoT infrastructure. However, fog computing does not solve the problem of security and privacy. Nevertheless, it optimizes data transfer to a degree, and sparked research interest in the field.

4 OUR PROPOSAL

We introduce Decentralized IoT Peer-to-Peer Protocol (DIPP). This protocol can be used for enabling device-to-device communication. Home automation and industrial automation are just two examples out of a large number of potential applications.

4.1 Protocol Overview

DIPP is an application layer protocol for Internet of Things (IoT) devices that lets such devices communicate with each other. It is secure, ensures privacy, retains data ownership, has a peer-to-peer architecture, and aims to utilize local resources more efficiently in a device-to-device (D2D) ecosystem.

An IoT device can connect to a network instantiated by this protocol using WiFi, Bluetooth or Ethernet since DIPP sits on top of transport layer protocols to establish connections and exchange messages. This protocols intends to host a heterogeneous array of radio transmission technology devices. Therefore, any IoT device that are capable of using TCP and UDP sockets is an eligible device.

Secure communication is achieved by encrypting the messages from end to end. Privacy is ensured by blocking devices that are not registered with the network, and every device in the network shares a common key representing the entity that owns the devices. DIPP has a peer-to-peer architecture, which establishes that devices are able to take on the role of service/content provider and switch it to receiver as well. Looking from a resource utilization perspective, a device can hold data of its own, or of other devices in the network, and, in case of a device with small secondary memory, this feature can be quite useful. Additionally, at a given time, devices with low CPU utilization can provide computing resources to devices that don't have much processing power.

4.2 Encoding

DIPP is designed to be a text-based protocol. Opting for textual protocol instead of binary one makes it easier to understand, implement and debug. In case of future extensions, the text-based nature would present minimal roadblock augmenting new structures or modifying the existing ones. The downside of not using a binary structure is insignificant since in the vast majority of the cases, the small IoT devices exchanges data and content that are quite small. Therefore, the efficiency achieved with a binary protocol cannot outweigh the benefits comes with a text-based protocol. We combine ABNF syntax defined in RFC 7405, and use 'utf-8' as external encoding.

4.3 Message Exchange Pattern

DIPP allows a few different interaction models between devices for simplicity and modularization. Here we discussed the models briefly. In the software implementation section, notes and pointers are added to aid in the implementation process.

4.3.1 Request-Response. Request-Response is a messaging pattern very common in protocol that follows client-server model, most notably in HTTP. Client initiates the connection in HTTP, and the server responds. What makes client-server different from P2P model is that the roles are fixed in the former, and flexible in the latter. Therefore, any peer device in DIPP would be able to take on the role of initiator and responder: both client and server interfaces encompassed in one device.

We have two methods for constructing messages according to request-response pattern. REQ, is shorthand for request, and RES, is for response. A message that has REQ method follow the structure below:

$\langle REQ \rangle$	$\langle fieldname \rangle$

This structure essentially communicates to the responder that they want the value(s) for the fieldnames(s) indicated. A list of fieldnames are provided later in the section. Not all these fields are implemented by every device. However, there are some fields that every device is required to provide to enable smooth functioning of the network. Messages that have $\langle RES \rangle$ as their method has the following structure:

$\langle RES \rangle$	$\langle fieldname \rangle : \langle value \rangle$

Request-Response messaging pattern perhaps best illustrated by a sample conversation between the hosts. For the sake of simplicity, we will indicate the connection initiating host with `I` and the responding host with `R`.

```
<host I initiates a connection with host R>
I> REQ deviceType
R> RES deviceType:sensor
I> REQ readingType readingCurrent
R> RES readingType:speed readingCurrent:85
```

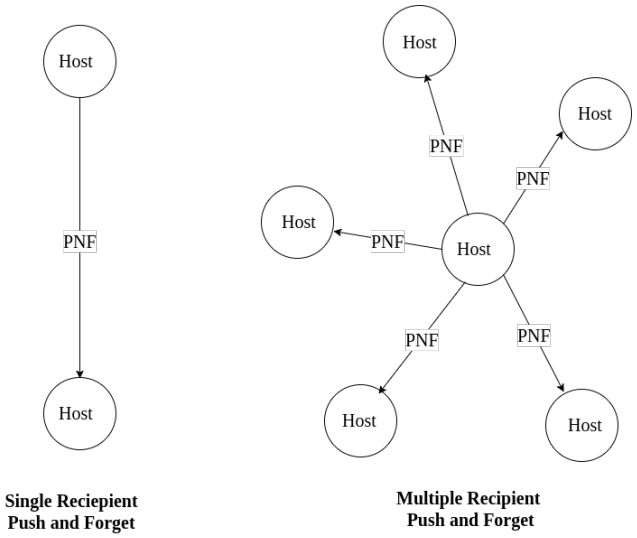

Figure 1: Push and Forget Model

```
I> REQ readingUnit
R> RES readingUnit:mph
R> REQ readingType
I> RES readingType:3001-invalid-request
R> REQ deviceType
I> RES deviceType:actuator
<host I closes the connection with host R>
```

Listing 1: Request-Response Example

In this example, while host I initiated the exchange with R, they both can send REQ and RES messages. Once a REQ message is sent, the receiving host of that message cannot initiate a REQ message to the other host until the receiver responds with a RES message. Additionally, any peer using the DIPP protocol is capable of being server and client, i.e., listen for and request for a connection, respectively.

4.3.2 Push-and-Forget. This model of communication is also known as Fire-and-Forget. Push-and-Forget is self-explanatory: a host pushes data, status, information to another host and forgets about it. It does not care about acknowledgement or any response from the receiving host. This kind of interaction can be useful when dealing with real-time data such as streaming. This is also useful for broadcasting messages. **Figure 1** illustrates sending messages from a sender to a single peer and multiples peers. Regardless of the particular application case, Push-and-Forget (PNF) model is supposed to provide convenience and reduce complexity related to framing. Push-and-forget messages share the same message structure as the messages that have RES in their method field, which has the following structure:

$\langle PNF \rangle$	$\langle fieldname \rangle : \langle value \rangle$

4.3.3 Push-Acknowledge. Push-Acknowledge have the same underlying mechanism as Push-and-Forget, but after sending the push message, a similar type of messages is sent, with the acknowledgement. The push message in push-acknowledge model has the same

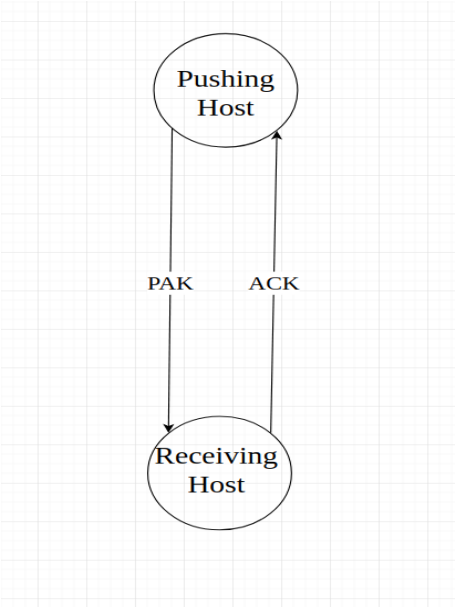

Figure 2: Push and Acknowledge model

structure as the one found in push-and-forget, except for the method line, which contains the string $\langle PAK \rangle$, like the following:

$\langle PAK \rangle$	$\langle fieldname \rangle : \langle value \rangle$

The receiving host sent a message back to the sender, with the following format:

$\langle ACK \rangle$	$\langle fieldname \rangle : \langle ACK/NACK \rangle$

Although push-acknowledge messages are not supposed to be using connection-oriented protocols, receiving acknowledgement after a broadcast Push-Acknowledge (PAK) messages can degrade and congest the channel. Implementing a congestion control mechanism is recommended for this reason.

4.4 Devices

The vast majority of popular application layer protocols that we cited earlier utilizes client-server model, resulting one-directional data flow. In order to realize the full potential of D2D communication, however, bi-directional data flow is necessary. One-directional data flow such as client-server or sensor-aggregator limits the novel ways device-to-device communication can happen. DIPP's three different models attempts to provide that while also considers the different energy restrictions and capabilities of connected devices. IoT devices come with varying degree of computing resources, and we aimed to enable the network to utilize the available resources in a fair and efficient manner. This led us to categorize the devices in the two categories: Nodes and Super Nodes

(1) **Super Node**:A Super Node have the processor and memory available that are the same magnitude of a computer. An example would be an A Raspberry Pi, or an Arduino Yun device. They can act as a server, e.g, authenticating a node a providing it with a network ID. They can store data transmitted other nodes. A Super Node may also have the

capability to process those data with the application logic provided by a developer.

Super Nodes implements support for all three messaging patterns. In addition to authentication, they provide a larger list of neighboring peers. Super Nodes can connect with other Super Nodes too.

(2) **Node**:In contrast to Super Nodes, a Node have limited computing resource, memory and/or storage. They often are quite inexpensive, as a result. Example of a Node can be a sensor, that sends data periodically to a Super Node that stores such data, and do further processing on it. A node has to implement the request-response and push-and-forget models, for minimal capability.

4.5 Operation

Operations performed by an IoT device in the network falls into the three broad categories.

(1) **Registration**: Before being able to connect to a network, a node has to authenticate and obtain a form of identification from a Super Node, with a key that validates the ability to be a part of the network, e.g., which can be set by the entity that owns the device. Registration is triggered by a method called ⟨*REG*⟩ from an unregistered device, and then a series of message exchanges happens. Only upon successful registration to the network, a device can use this protocol to communicate with other devices.
(2) **Data transfer**: After the device is registered, and have a list of nearby devices, it is capable of performing data transfer. It can request for contents and/or services from other devices in the network. The newly added device is supposed to provide similar functionalities to other devices in the network as well.
(3) **De-registration**: Before disconnecting from a network, the standard procedure is to broadcast the intent to de-register so that the data stored in the said device can be relocated to another device. This also allows other nodes in the network to update their list of reachable peers.

Depending on the devices and operations, a devices can be in one of the following states in relation to the network at any point in time.

(1) **Unregistered**: Before registration and after de-registration.
(2) **Active**: Registered and able to exchange messages immediately.
(3) **Unreachable**: Registered but not reachable at the moment, e.g., power being shut down..

4.6 Message Format/Framing

All of the mechanisms are described here uses augmented Backus-Naur Form (BNF) similar standardized in RFC 822[4]. Within the message, fields are separated by the using ⟨*SP*⟩ character

characters. In case of a space being part of some field, they are substituted with a dash character. The message delimiter is ⟨*CLRF*⟩. The rules of escaping flag characters specified in the RFC also applies here.

Method	Payload	Flag

4.7 Message Fields

Due to the heterogeneous nature of IoT devices, it is both unwise and impossible to attempt to specify and implement a comprehensive list of $< fieldname : value >$ pairs. Therefore, we focus on a select few such structures to ensure minimal connectivity between devices within the network. Additionally, a standard implementation must provide a few pairs for accessing sensor and actuators interfaces.

4.7.1 Required Network Services.

fieldname	value(s)
⟨*deviceName*⟩	`manufacturer provided name`
⟨*netDevice*⟩	`FullNode, LimitedNode`
⟨*neighbors*⟩	`neighbor devices addresses`
⟨*location*⟩	`location, refused`
⟨*isAlive*⟩	`True, False`
⟨*isFullNode*⟩	`True, False`
⟨*services*⟩	`non-required services provided`

4.7.2 Required Device Services.

fieldname	value(s)
⟨*deviceName*⟩	`manufacturer provided name`
⟨*deviceTyoe*⟩	`sensor, actuator, controller`
⟨*readingType*⟩	`temperature, humidity, etc.`
⟨*readingUnit*⟩	`celsius, fahrenhite, mph, etc.`
⟨*isFullNode*⟩	`True, False`
⟨*actuatorFunc*⟩	`door-opener, etc`
⟨*actuatorState*⟩	`open,close`

As it illustrated above, very soon the data passed around become dependent on the specific device. Being able to create custom fields and values gives the implementers and application developers the ability to design their application best suited to their needs.

4.8 Status Codes

The status codes are used for reporting errors, provide informations, and is intended to be aides for the users, developers, and implementors of the protocol. Status codes are divided into following three categories. categories. A status code contain a 4-digit numeral for machine parsing, followed by a string of text for the human reader.

(1) **1XXX: Success**
Status codes with the format 1XXX indicates success;
   (a) **1000 Success Registration**
   This status code is sent after a device is able to register to the network.
   (b) **1001 Connected to Host** Whenever a connection is initiated, and it is successful, and the initiator gets this status code is sent as confirmation, and that host can move on carry out other network activities.
(2) **2XXX: Network Error**
The set of codes that start with 2 indicates errors related to network. In order to troubleshoot these issues, one needs to look into their network setup: [needs review]
   (a) **2001 Host unreachable**
   The host can be unreachable for many different reasons; applicable in transport layer error cases

(b) **2002 Address Error** When the IP or the port address is not correct
(c) **2003 Host Not Registered**
(d) **2005 Authentication Failure**

(3) **3XXX: Command Error**
Codes starting with 3 represents the set of situations when errors related to syntaxes or unavailability of services due to a variety of reasons.
(a) **3001 Item Not Found**
(b) **3002 Invalid Request**
(c) **3003 Not Implemented**
(d) **3004 Unrecognized field**

4.9 Authentication and Security

Without a central entity, providing authentication and trust is difficult. There are efforts to provide authentication for a group, which is more suitable for a peer-to-peer protocol. [7] authors claim that the scheme they proposed is lightweight, scalable, and well-suited for IoT devices. However, one drawback of their approach is that there is an assumption of an already established network of devices. Our approach is to ensure the authenticity of a peer before it is able to access data and services from the peers in the network.

Authentication and packet encryption in Wireless Sensor Network (WSN) have been an active field for numerous research. [8] proposed a solution that claims to provide two-way authentication using three-way handshake while being resource efficient at the same time. Strategies described in [17] and [5] argues in favor of using pre-deployed keys to authenticate sensors and encrypt communication. As we stated earlier that data ownership and privacy are among the challenges we aim to solve, DIPP favors authentication protocol proposed by [5]. It aligns with our approach towards retaining device and data ownership. It is simple and resource-efficient enough to be implemented for constrained sensor devices, while provides a good level of security. Additionally, it allows key revocation in case of device compromise.

Notwithstanding the above discussion, we anticipate that the security and/or encryption protocol are not permanent, especially at the early phase of development. Solutions to trust and security challenges need to be continuously evaluated and reexamined, and we don't wish to be an exception to that principle.

4.10 Implementation

An implementation of the protocol is available at [11]; it is written in Python and tested on Arduino Yun devices. Note that this implementation serves as a proof of concept. So a comprehensive implementation and extensive testing is necessary before rolling out a public release.

In this implementation, we used TCP and UDP as Transport layer protocols to provide request-response, and push models, respectively. We looked at few studies that measure the performance of UDP and TCP over protocols such 802.11 (Wifi), BLE, Bluetooth. It's been shown that for an ad-hoc network over wireless medium, although TCP is widely used, not all mobility models and routing protocols provide satisfactory performances over TCP[15].
On the other hand, UDP, when used over a limited bandwidth medium such as 802.11g, can provide a higher bandwidth but loses considerable amount of data. Moreover, in a situation where both UDP and TCP are used, due to the lack flow control, UDP can degrade performance of TCP[12]. This observation proved to be useful when designing the protocol incorporating both datagram and stream sockets at the transport layer.

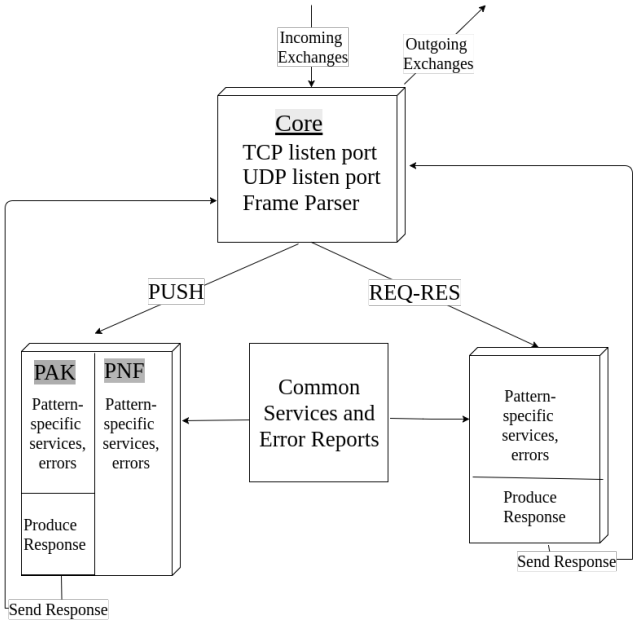

Figure 3: Software Architecture Diagram

Figure 3 shows different modules of the implementations are dependent on each other. This is a high level abstraction of modules but gives a good idea how these modules work together to render the protocol functionalities.

5 ACKNOWLEDGEMENT

I would like to thank my capstone project advisor Professor Charlie Peck for his continuous supervision, guidance, and frequent sanity checks throughout the year. I would also thank Earlham College Computer Science department for providing me with the hardware tools. The specification of this protocol is influenced by several IETF standard track protocols. The tutorial values of RFC 3117[10], and the UNIX philosophy[9] are well-realized in this project, and it proved to be a tremendous opportunity for growth and learning.

6 FUTURE WORK

We were able to present a protocol that provides security, maintains privacy and retains data ownerships of the appropriate entity. However, our implementation is a proof of concept, and we hope to release a complete implementation with rich documentation and technical support. In addition to extending software implementation, we would like to carry extensive testing, perhaps on larger test beds to measure performance and scalability. Finally, as noted in the implementation section, We plan on extending flow and congestion control that covers all three interaction models in the next draft.

7 CONCLUSION

We discussed challenges of integrating everyday objects as first class citizens of the Internet. While doing so provides convenience, extensive research effort need to be undertaken in areas of security, privacy and resource utilization. From our analysis, it turns out that although there are protocols such as MQTT, CoAP, XMPP etc. that are optimized for connection and network performance in the presence of resource constraints, we are yet to encounter protocols that addresses the key issues we identified: eliminating unnecessary data exchanges, utilizing local resources, and addressing consumer issues such as privacy and data ownership.

We proposed a protocol that identifies and attempts address those issues. However, our study is not adequate to tackle the numerous challenges existent in this area; further research into these issues are a necessity.

REFERENCES

[1] Luigi Atzori, Antonio Iera, and Giacomo Morabito. "The Internet of Things: A survey". In: *Computer Networks* 54.15 (2010), pp. 2787–2805. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2010.05.010. URL: http://www.sciencedirect.com/science/article/pii/S1389128610001568.

[2] Sven Bendel et al. "A service infrastructure for the Internet of Things based on XMPP". In: *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013 IEEE International Conference on*. IEEE. 2013, pp. 385–388.

[3] C Bormann, M Ersue, and A Keranen. *RFC 7228: Terminology for Constrained-Node Networks*. Tech. rep. tech. rep., IETF, 2014.

[4] David Crocker. "Standard for the format of ARPA Internet text messages". In: (1982).

[5] Laurent Eschenauer and Virgil D. Gligor. "A Key-management Scheme for Distributed Sensor Networks". In: *Proceedings of the 9th ACM Conference on Computer and Communications Security*. CCS '02. Washington, DC, USA: ACM, 2002, pp. 41–47. ISBN: 1-58113-612-9. DOI: 10.1145/586110.586117. URL: http://doi.acm.org/10.1145/586110.586117.

[6] Vasileios Karagiannis et al. "A survey on application layer protocols for the internet of things". In: *Transaction on IoT and Cloud Computing* 3.1 (2015), pp. 11–17.

[7] Parikshit N Mahalle, Neeli Rashmi Prasad, and Ramjee Prasad. "Threshold cryptography-based group authentication (TCGA) scheme for the internet of things (IoT)". In: *Wireless Communications, Vehicular Technology, Information Theory and Aerospace & Electronic Systems (VITAE), 2014 4th International Conference on*. IEEE. 2014, pp. 1–5.

[8] Martin Noack. "Optimization of two-way authentication protocol in internet of things". PhD thesis.

[9] Eric S Raymond. *The art of Unix programming*. Addison-Wesley Professional, 2003.

[10] Marshall T Rose. "On the design of application protocols". In: (2001).

[11] Sirajus Salekin. *salekinsirajus/DIPP: Making the codebase citable*. Dec. 2017. DOI: 10.5281/zenodo.1098483. URL: https://doi.org/10.5281/zenodo.1098483.

[12] Leandro Melo de Sales, Hyggo O Almeida, and Angelo Perkusich. "On the performance of TCP, UDP and DCCP over 802.11 g networks". In: *Proceedings of the 2008 ACM symposium on Applied computing*. ACM. 2008, pp. 2074–2078.

[13] Zhengguo Sheng et al. "A survey on the ietf protocol suite for the internet of things: Standards, challenges, and opportunities". In: *IEEE Wireless Communications* 20.6 (2013), pp. 91–98.

[14] Meena Singh et al. "Secure mqtt for internet of things (iot)". In: *Communication Systems and Network Technologies (CSNT), 2015 Fifth International Conference on*. IEEE. 2015, pp. 746–751.

[15] Sunil Kumar Singh, Rajesh Duvvuru, and Jyoti Prakash Singh. "Performance impact of TCP and UDP on the Mobility Models and Routing Protocols in MANET". In: *Intelligent Computing, Networking, and Informatics*. Springer, 2014, pp. 895–901.

[16] Dinesh Thangavel et al. "Performance evaluation of MQTT and CoAP via a common middleware". In: *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on*. IEEE. 2014, pp. 1–6.

[17] Sencun Zhu, Sanjeev Setia, and Sushil Jajodia. "LEAP+: Efficient security mechanisms for large-scale distributed sensor networks". In: *ACM Transactions on Sensor Networks (TOSN)* 2.4 (2006), pp. 500–528.

[18] Jan Henrik Ziegeldorf, Oscar Garcia Morchon, and Klaus Wehrle. "Privacy in the Internet of Things: threats and challenges". In: *Security and Communication Networks* 7.12 (2014), pp. 2728–2742.