# 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 towards the cloud infrastructure. Peer-to-Peer (P2P) networks can be an attractive option, in this scenario, that has 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

### 1.1 Peer to Peer

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. and services.

P2P networks are usually associated with primarily file sharing applications such as Napster, Gnutella, or BitTorrent. However, the applications and applicability of P2P protocols and networks are more varied than the apparent narrow focus of applications. In addition to other applications, P2P networks can be quite useful in device-to-device (D2D) communications. While P2P models are usually more resilient than client-server models, absence of a central authority makes often translates into inefficient search and discovery, among other drawbacks. However, P2P models still have large rooms for improvements and significant potential in an increasingly more connected world.

### 1.2 Internet of Things

Internet of Things devices 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. However, they are be very useful when it comes to collecting data, automating specialized task etc. Connecting such devices in a peer-to-peer network creates design and implementation challenges. In this paper, we attempt to tackle on some of those challenges, and our experiments show that it (will be) indeed a reasonable option for connecting IoT devices.

The rest of th documents contains the following sections in order: problem statement, where we detail what motivated us to conduct the project and pinpoint the problems we aimed to solve. In Past Research section, we discuss the existing protocols and research effort, and how they fall short in solving the problems we found out. Research methodology section describes the systematic approach we took to accomplish this project. The next section, Proposal, details the ins and out of the proposed protocol. We end the paper by Future Work and Conclusion sections.

## 2 PROBLEM STATEMENT

The widespread adoption of small devices capable of connecting to the Internet provides consumers with enoromous benefits. At the same time, challenges of multiple domains are inherited in the process, and at the intersection, new kinds of problems arise. In particular, the issues of privacy, security, and data transfer caught our attention. In terms 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,


*Conference'17, Washington, DC, USA*

DOI: 10.1145/nnnnnnn.nnnnnnn

but the consequences of security compromise of such devices–for example, a wearable tracker–can be enormous.

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. As evident in other fields of science and innovation, it is important that we, as practitioners of technology, 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 process.Issues related to content ownership should be a key focus as consumers introduce more connected devices in their household and workplace etc.

When considering privacy challenges, it is evident that privacy and data ownership are closely interconnected. Zigeldorf observed at [**zigeldor**] identification, tracking and profiling are some of the primary threats related to IoT privacy. Additionally, as the IoT ecosystem evolves, privacy issues arise in way that we have not anticipated.

*There are thousands of IoT device manufacturers globally*, and efforts to standardize protocols of different sorts are very challenging. (cite: Thread vs Zigbee article). This resulted in *a large number* of proprietary protocols and frameworks, which, more often than not, focuses on functioning capabilities of their products while consumer concerns are not given serious considerations. This prompted us to propose a platform that enable device-to-device communication and emphasize primarily on the issues that affect average consumer: privacy, security, data ownership, and efficient resource utilization.

## 3 PAST RESEARCH

Communication is one of the building blocks that delivers the IoT functionality[3]. Enabling connectivity to an otherwise 'dumb' device is what drove the evolution of RFID to this huge IoT ecosystem we know today. Research in IoT communication protocols is shaped by factors such as small size, small processing, small memory and limited battery life.

In light of our research interest, we will discuss application/session layer protocols that are available today. Notable examples include: MQTT, CoAP, HTTP, XMPP, SMQTT. Most of these protocols use either TCP or UDP as transport layer protocols. We will also discuss data link layer protocols later in this section. We reviewed novel approaches to handle IoT communications such as fog-computing.

The Constrained Application Protocol (CoAP) appreas to be the de-facto choice when it comes to application layer protocols. IETF's Constrained RESTful working group (CORE) specified this protocols 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 [10]. Although CoAP was well-suited for power-constrained IoT devices, it does not provide built-in security, being on top of UDP. There are efforts to replace UDP with a secure protocol such as Datagram Transport Layer Security (DTLS) for CoAP; however, their compatibility with CoAP is questionable, or they are under development [4].

Another well-known application layer protocol is MQTT, which was originally designed by IBM in 1999. It has a publisher-subscriber model through a broker, i.e., a server. Sensors and lightweight devices are typically act as publishers, and applications, devices that are interested in retrieves those data through the server. MQTT protocol is well-suited for IoT devices, for it's low overhead, [4] and its design to use battery sparingly. This protocol shown to be effective at keeping the energy consumption low, despite using TCP as the underlying protocol. In terms of security, MQTT might require username and password authentication transported over SSL/TSL, which is used in case of HTTPS.

MQTT and CoAP are compared and experimented frequently on different matrices, and the fare closely in latency, packet loss, QoS etc. [13]

The Extensible Messaging and Presence Protocol (XMPP) was once popular but declined because of incompatibility with some 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. [1] 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.

There are 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. [11]. Another similar protocol is AMQP, which has its use in financial industry.

The protocols discussed above follow a client-server architecture, so they rely on the server for successful use of the network. As we mentioned earlier, while such model makes control and data aggregation easier, a single point of failure is present, and these protocols are designed in a way that aggregated data would converge towards using 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, and usually with the help of a sink.

Wireless Sensor Network implementations and technologies have come a long way, primarily because of their applicability. Industrial and home automation, for example, are two major areas where a sensor network can be used reliably, for simple data collection. Attempts to implement such technologies on IoT devices have been an important field of during 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.

Perhaps the most noteworthy aspect of the collection of available IoT devices in the market is their heterogeneity, in terms of functionalities, underlying hardware and software implementations, and connection protocols, among other things. This translates into interoperability between devices on several levels being a major issue in device-to-device communication in the coming days. We discuss some of the major networks protocols, especially on the bottom layers of network models. These protocols and their implementations are shaped by the resource constraints a small, compact devices present.

### 3.2 Data Link Layer Protocols

As IoT evolved from RFID tags to Amazon Alexa or Google Home devices, manufacturers needed to consider the energy constraint. Existing physical communication technologies such as WiFi or Ethernet were, thus, not the most appropriate choice for a good number of IoT applications. Technologies such ZigBee, Thread, Bluetooth Low Energy (BLE) were specified and implemented considering the special requirements of small devices.

Smallest of IoT devices are RFID tags. They are cheap, and contains and the first generation of RFID tags contained only barcodes, so ensuring security or providing ZigBee or Z-Wave are alternatives to WiFi or Bluetooth, since these radio protocols consume a lot of energy.

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 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 pretty low, at 100kbps Apart from that, the 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.

The heterogeneity of communication protocols found at the bottom layers is a problem of interoperability, which cannot be solved solely technical means. There are thousands of manufacturers of IoT devices, and the products either have proprietary communication stack, or follow one of the several alliances like Thread, ZigBee or Z-Wave. Standard bodies such as IEEE-SA have been working on standards for IoT networks and devices, but progress has been slow, partly because such decisions are invariably entangled with commercial and political interests. Focusing on the application layer protocols that can sit on top Transport layer protocols such as TCP, UDP enabled us to abstract away the messy details and let us produce protocols that can be implemented on a wide range of devices.

A very recent phenomena 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 insfrastructure. However, fog computing does not solve the probelm of security and privacy. Nevertheless, it optimizes data transfer to a degree, and sparked research interest in the field.

## 4 OUR PROPOSAL

We introduce a framework that enables a distributed computing environment in an IoT network, with emphasis on network security, data privacy, and ownership to the appropriate entity. Such a system can be used in home automation, industrial automation, and in a host of other cases. We propose to view an IoT network from a distributed computing perspective, treating it a like a system that provides abstract interfaces for the applications to use.

### 4.1 Overview of the framework

DIPP is an application layer protocol for Internet of Things (IoT) devices that lets such devices to communicate with each other. It is secure, provides privacy, retains data ownership, has a peer-to-peer architecture, and aims to utilize local resources more efficiently in a machine-to-machine (M2M) ecosystem.

A device can connect using WiFi, Bluetooth or Ethernet on the Physical layer, and DIPP uses transport layer protocols to establish connections, or send connectionless 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 able connect a network instantiated by this protocol.

Secure communication are achieved through encryption of the messages to be exchanged between devices. Providing adequate privacy to connected devices is a lofty goal, and it could come at the cost of functionality. However, our emphasis on privacy did not let us sacrifice authentication in favor of fast plug-n-play approach. So before a machine can join the network, it needs to be able to provide a form of passcode or fingerprint, which establishes the legitimacy of the devices, and confirm the ownership of the appropriate entity. After a machine authenticates itself, it is able to exchange data with other devices that are part of the network. The network has a peer-to-peer architecture, which establishes that devices are able to serve contents and services to other devices, and be served those functionalities, in the network. 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 represent minimal problem extending new structures or modifying the existing ones. The downside of not using a binary structure is also minimal since in the vast majority of the cases, the

small IoT devices exchanges data and content that are quite small. So the efficiency achieved through a binary protocol is insignificant.

## 4.3 Message Exchange Pattern

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

*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 fiel-names(s) indicated. A list of fieldnames are provided later in the section. An instance of such a message looks like this:

REQ SP "readingType" SP "readingUnit" SP "deviceType"

Not all these fields are implemented by every device. However, there are some fields that every device is required to provide for enabling smooth functioning of network. Messages that have RES as their method has the following structure:

| $\langle REQ \rangle$ | $\langle fieldname \rangle : \langle value \rangle$ |
|---|---|

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
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 server and client, i.e., listen for and request for a connection, respectively.

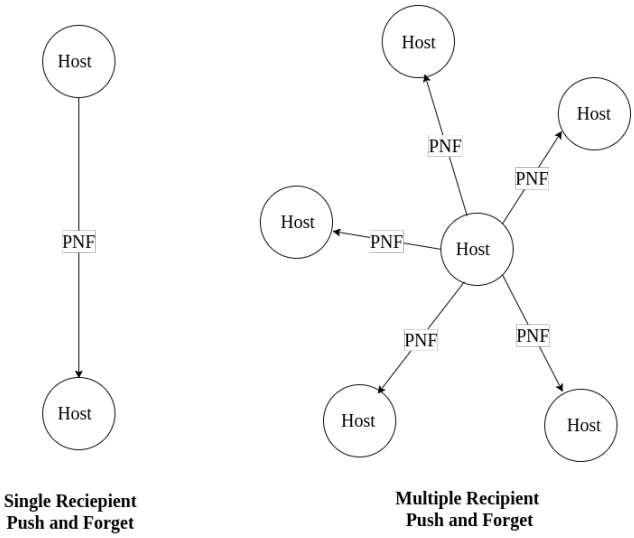


**Figure 1: Push and Forget Model**

*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. The diagram 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. Push-and-forget messages share the same message structure as the messages that have RES in their method field.

*4.3.3 Push-Acknowledge.* Push-Acknowledge should have the same underlying mechanism, but after sending the push message, a similar type of messages is sent but with the acknowledgement. The receiveing host sent a message back to the sender, with the following format:

| $\langle REQ \rangle$ | $\langle fieldname \rangle : \langle ACK/NACK \rangle$ |
|---|---|

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

## 4.4 Participating 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-aggregrator 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 capabilties of connected

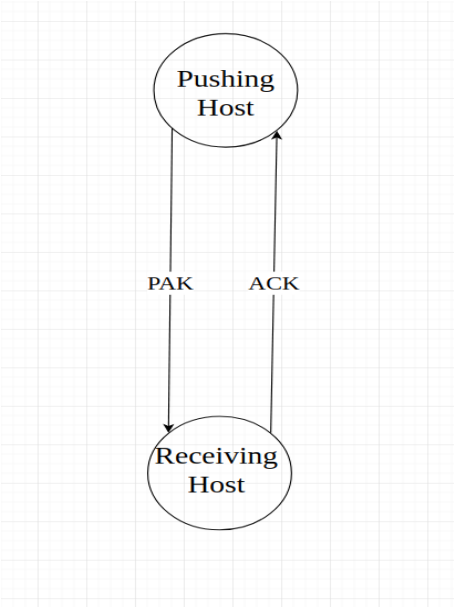


**Figure 2: Push and Acknowledge model**

devices. IoT devices come with varying degree of computing resources, and we aimed to enables the network to utilize the available resources in a fair and efficient manner.

(1) **SuperNode**:A SuperNode 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 SuperNode may also have the capability to process those data with the application logic provided by a developer.

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

(2) **Node**:In contrast to SuperNodes, 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 SuperNode 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 capablity.

## 4.5 Operation

A device that is willing to be a part of the network can do the following sequnce of opertaion:

(1) **Registration**: Before being able to connect to a network, a node has to authenticate and obtain a form of identification from a SuperNode, with a fingerprints or passcode that validates the ability to be a part of the network, e.g., which can be set by the entity that owns the device.

(2) **Data transfer**: After the device recieves a network ID, and have a list of nearby devices, it is capabale 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, including: running a program, storing file(s), providing file(s), providing services and data it produces. We will discuss more on data transfer at a later section.

(3) **De-registration**: Before disconnecting from a network, the standard procedure is to broadcast the intent to deregister, 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.

## 4.6 Message Format/Framing

All of the mechanisms are described here uses augmented Backus-Naur Form (BNF) similar standardized in RFC 822[2]. Within the message, fields are separated by the using $\langle SP \rangle$ character

characters. In case of a space being part of some field, they are subsitituted with a dash character. The message delimiter is $\langle CLRF \rangle$. The rules of escaping flag characters specified in the RFC also applies here.

| Method | Payload | Flag |
|---|---|---|

## 4.7 Semantics Dictionary

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) |
|---|---|
| $\langle deviceName \rangle$ | `manufacturer provided name` |
| $\langle netDevice \rangle$ | `FullNode, LimitedNode` |
| $\langle neighbors \rangle$ | `neighbor devices addresses` |
| $\langle location \rangle$ | `location, refused` |
| $\langle isAlive \rangle$ | `True, False` |
| $\langle isFullNode \rangle$ | `True, False` |
| $\langle services \rangle$ | `non-required services provided` |

### 4.7.2 *Required Device Services.*

| fieldname | value(s) |
|---|---|
| $\langle deviceName \rangle$ | `manufacturer provided name` |
| $\langle deviceTyoe \rangle$ | `sensor, actuator, controller` |
| $\langle readingType \rangle$ | `temperature, humidity, etc.` |
| $\langle readingUnit \rangle$ | `celsius, fahrenhite, mph, etc.` |
| $\langle isFullNode \rangle$ | `True, False` |
| $\langle actuatorFunc \rangle$ | `door-opener, etc` |
| $\langle actuatorState \rangle$ | `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 the following [how many] categories. A status code contain

(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. (The packet might contain necessary information relating registration)
   (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. Whatever the underlying cause is, the destination host cannot be reached at this moment. Issues related to routers, bridges, hubs etc might cause an instance of this.
   (b) **2002 = Address Error** (communicated from TCP) 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**

## 4.9 Authentication

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. [5] 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. We are using SASL as an authentication and security layer.

## 4.10 Implementation

An implementation of the protocol is available at [8]; 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 a necessity before rolling out a public release.

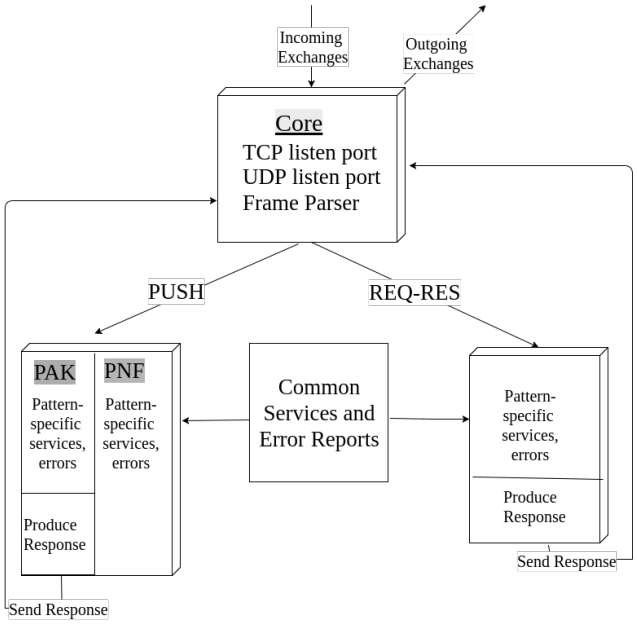


**Figure 3: Software Architecture Diagram**

Push messages are intended to have low-latency and smaller payload. Broadcasting announcements to devices within the network is an example where push messages can be very useful. In our implementation, we used UDP as Transport layer protocol to avail this. In particular, we considered the tradeoff's between relaibility and speed. We looked at few studies that memasure 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[12].
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[9]. This observation proved to be useful when desiging the protcol incorporting both datagram and stream sockets at the transport layer.

The following digram 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 come together as one.

## 5 ACKNOWLEDGEMENT

I would like to thank Professor Charlie Peck for his continuous advising, motivation and frequent sanity checks, which were crucial to my Senior Capstone Project. Additionally, I would 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. We made use of guidances and advices from RFC 3117[7], and the UNIX philosophy[6].

## 6 FUTURE WORK

Although we were able to present a framework that provides security, maintains privacy and retains data ownerships of the appropriate entity, we would like us to be able to implement the software in its complete form so that heterogeneity is achieved. In addition to extending software implementation, we would like to carry extensive testing, perhaps on larger test beds to determine how well does our framework scale. As noted in the implementation section, We plan on extending flow and congestion control that covers all three interaction models in the next draft. Finally, we would like to compare with other protocols or frameworks that uses peer-to-peer architectures for serving IoT devices and strengthen areas where our framework might be make use of improvements.

## 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 undertaken in areas of security, privacy and resource utilization. From our analysis, it turns out that although there are protocols such MQTT, CoAP, XMPP etc. that are optimized for connection and network performance in the presence of resource constraints, we have not encountered protocols that eliminates unnecessary data exchanges, utilizes local resources, and in general emphasize on consumer issues such as privacy and data ownership.

We proposed a protocol that identifies and attempts to at least guide to the right direction. Instead of converging the data produced in the IoT ecosystem to the cloud, we we shifted our focus on making use of local processing capabilities, and provide platform for secure data exchange, transferring ownership to appropriate entities that produced the data in the first place.

## REFERENCES


[1] 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.

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

[3] Ala Al-Fuqaha et al. "Internet of things: A survey on enabling technologies, protocols, and applications". In: *IEEE Communications Surveys & Tutorials* 17.4 (), pp. 2347–2376.

[4] 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.

[5] 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.

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

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

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.