IoT Platform (Device-to-Device Communication)

Akshay Joshi
Electrical and Computer Engineering
University of California, Los Angeles
Los Angeles, CA, USA
akshayjoshi@ucla.edu

ABSTRACT

This project aims to introduce the concept of interconnection of things via the internet commonly known as the Internet of Things (IoT). The work deals with end-to-end transmission and reception of messages along a medium (Wi-Fi in this case). It uses the concept of sockets to use a single IP address and multiple port numbers to enable multiple communication channels on a single IP address (or a single device). It targets system level design and tries to address key aspects that have been taken into consideration while designing the solution. These considerations have been thought over and addressed to their full capacity in order to remove any unwanted complications that may arise during the operation. The report includes demonstration of the project idea along with the results. It also includes the roadblocks faced and the ways chosen to overcome them. Future work that can be done, summary, and insights are also covered.

KEYWORDS

Internet of Things, IoT, socket, SSL, security, authentication, network, transmission, reception, communication

1. Introduction

The Internet of things (IoT) describes the network of physical objects (things) that are embedded with sensors, software, and other technologies for the purpose of connecting and exchanging data with other devices and systems over the Internet. The definition of the Internet of things has evolved due to the convergence of multiple technologies, real-time analytics, machine learning, commodity sensors, and embedded systems. Traditional fields of embedded systems, wireless sensor networks, control systems, automation (including home and building automation), and others all contribute to enabling the Internet of things. In the consumer market, IoT technology is most synonymous with products pertaining to the concept of the "smart home", including devices and appliances (such as lighting fixtures, thermostats, home security systems and cameras, and other home appliances) that support one or more common ecosystems, and can be controlled via devices associated with that ecosystem, such as smartphones and smart speakers. IoT can also be used in healthcare systems.

There are a number of serious concerns about dangers in the growth of IoT, especially in the areas of privacy and security, and consequently industry and governmental moves to address these concerns have begun including the development of international standards. The main concept of a network of smart devices was discussed as early as 1982, with a modified Coca-Cola vending machine at Carnegie Mellon University becoming the first Internet-connected appliance, able to report its inventory and whether newly loaded drinks were cold or not. [1]

2. Background

There have been many previous works done on IoT and its applications. They include open-source projects like DeviceHive, AWS IoT, Google IoT, etc.

2.1 DeviceHive

DeviceHive is an open-source platform which provides instruments for smart devices' communication and management. It consists of the communication layer, control software and multi-platform libraries and clients to bootstrap development of smart energy, home automation, remote sensing, telemetry, remote control and monitoring software and much more.

Speaking more technically, it's a scalable, hardware- and cloud-agnostic microservice-based platform with device-management APIs in different protocols, which allows setting up and monitoring devices' connectivity, control them and analyze their behavior.

The platform covers the whole flow starting from Data Transition, Validation and Collection up to Machine Learning jobs and Artificial Intelligence. It also provides

monitoring tools to start the discovery of devices without being obliged to connect real hardware to the platform at startup. DeviceHive provides REST, WebSocket APIs by default + MQTT API as an additional plugin. All communication is performed via JSON messages. Devices with Python, Node.js or Java support, like Linux boards, Android Things devices, etc. can be easily connected simply by installing DeviceHive client library.

For all RESTful services DeviceHive provides Swagger - API developer tool which allows to test the installation or to explore DeviceHive capabilities. [2]

2.2 AWS IoT

AWS IoT boasts 4 features: broad and deep support, multi-layered security, superior AI integration, proven at scale. AWS has broad and deep IoT services, from the edge to the cloud. AWS IoT brings together data management and rich analytics in easy-to-use services designed for noisy IoT data. AWS IoT offers services for all layers of security, including preventive security mechanisms, like encryption and access control to device data, and a service to continuously monitor and audit configurations. AWS brings AI and IoT together to make devices more intelligent. One can create models in the cloud and deploy them to devices where they run 2x faster compared to other offerings. It is built on a secure and proven cloud infrastructure, and scales to billions of devices and trillions of messages. It integrates with other AWS services to enable building of complete solutions.

3. Implementation

There were three main components involved in this project, viz. sender device, server, receiver device. The prototype of the project supports only two devices at the time. But the entire system can be scaled to include as many devices as needed. The incorporation of server was to facilitate this scalability at a later stage when there would be multiple devices in the environment; and the system would act like a star-based system with the server being the hub, servicing multiple devices at the same time. Figure 1 shows the block diagram of the components and the flow of information.

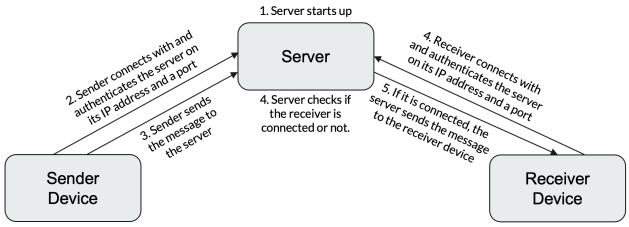

Figure 1: Components and Steps Involved

As shown in the figure, there are three main components involved: namely sender device, server, and receiver device. First step is for the server to start up. Once the server is started, it waits for the sender device to connect. The sender device identifies the server by its IP address and the server assigns a port number for the connection from itself to the sender device. Before transmitting anything, the sender first is obliged to authenticate the server to verify its true identity. Once the sender device authenticates the server, it then sends the message. The server then checks if the receiver device is present or not. Meanwhile, the receiver device identifies the server with its IP address and the server then assigns a port number for the connection from itself to the receiver device. Once this connection is established, the receiver, before receiving anything, authenticates the server using its certificate. After authentication, the server forwards the message to the receiver. At every milestone step, acknowledgements are sent out to the sender device to make the entire procedure as transparent as possible. Negative acknowledgements are also employed.

3.1 Key Considerations

Three key aspects that were taken into consideration were: authentication, acknowledgements, and elimination of dependencies.

3.1.1 Authentication

This authentication procedure uses the concept of certificates to perform the validation. The certificates, also known as SSL certificates, are officially issued to website hosts/servers by the Certificate Authority. For the purpose of this project, self-signed certificates were created using the openssl command. An SSL certificate is a type of digital certificate that provides authentication for a website and enables an encrypted connection. These certificates communicate to the client that the web service host demonstrated ownership of the domain to the certificate authority at the time of certificate issuance. An SSL certificate ensures that the server is who they claim to be

and also indicates secure connections between personal devices and websites. The certificate, in general, consists of various fields, viz. country name, state name, locality name, organization name, organizational unit name, common name, start date of validation, certificate expiry date, and the serial number. [4]

3.1.2 Acknowledgements (ACKs)

Acknowledgements were given thorough importance in order to make the entire operation of transmission and reception as transparent as possible to the sender device. The incorporation of acknowledgements enables the sender device to have up-to-date information of the data sent. The information may include the success/failure of the transmission, server's and receiver's reception of the message, receiver's reception, etc. For example, the server sends an ACK to the sender as soon as it receives the message. And the receive sends an ACK to the server as soon as it receives the message.

It also includes use of negative acknowledgement to let the sender know that the receiver wasn't able to receive the message for some reason.

3.1.3 Elimination of Dependencies

Elimination of dependencies stems from the fact that the sender's operation of sending a message should be independent of receiver device's connectivity with the server. This enables the sender device to send information whenever it has to send and thus takes the receiver's part in the sending procedure out of the picture which is how it should be. The server handles the complications of using a buffer and a timer to check whether the receiver device has connected or not. This leads to pushing all the complexity away from the end devices to the server to make the end devices simple. In this prototype, a timer of 10 seconds was employed by the server during which it waited for the receiver device to connect. If the receiver device got connected, the server sent the message to the receiver and sent an acknowledgement to the sender device. If the timer ran out, the server sent a negative acknowledgement to the sender.

4. Results

The results include the server side's, sender side's and receiver side's terminal windows which show the up-to-date proceedings of the operation.

```
Waiting for sender to connect
('Connected by', ('172.20.10.3', 55080))
Received data from the sender device. Trying to send it to the receiver device..
('Connected by', ('172.20.10.2', 57778))
Message received
One end-to-end transmission and reception successful.
```

Figure 2: Server terminal

```
{'issuer': ((('countryName', u'US'),),
            (('stateOrProvinceName', u'CA'),),
            (('localityName', u'LA'),),
            (('organizationName', u'UC'),),
            (('organizationalUnitName', u'EE'),),
            (('commonName', u'AJ'),)),
 'notAfter': 'Nov 23 04:05:37 2021 GMT',
 'notBefore': u'Nov 23 04:05:37 2020 GMT',
 'serialNumber': u'7DB75C5C61AE661EE80CEF4EC0693524162C1890',
 'subject': ((('countryName', u'US'),),
             (('stateOrProvinceName', u'CA'),),
             (('localityName', u'LA'),),
             (('organizationName', u'UC'),),
             (('organizationalUnitName', u'EE'),),
             (('commonName', u'AJ'),)),
 'version': 3L}
Message has been received by the server. The server is trying to send it to the receiver..
Message has been received by the receiver device.
```

Figure 3: Sender terminal

```
{'issuer': ((('countryName', u'US'),),
            (('stateOrProvinceName', u'CA'),),
            (('localityName', u'LA'),),
            (('organizationName', u'UC'),),
            (('organizationalUnitName', u'EE'),),
            (('commonName', u'AJ'),)),
 'notAfter': 'Nov 23 04:05:37 2021 GMT',
 'notBefore': u'Nov 23 04:05:37 2020 GMT',
 'serialNumber': u'7DB75C5C61AE661EE80CEF4EC0693524162C1890',
 'subject': ((('countryName', u'US'),),
             (('stateOrProvinceName', u'CA'),),
             (('localityName', u'LA'),),
             (('organizationName', u'UC'),),
             (('organizationalUnitName', u'EE'),),
             (('commonName', u'AJ'),)),
 'version': 3L}
('Received data:', "'Lo and Behold!'")
```

Figure 4: Receiver terminal

```
Waiting for sender to connect
('Connected by', ('172.20.10.3', 55163))
Received data from the sender device. Trying to send it to the receiver device..
Waited for 10 seconds. Receiver offline. Retry later.
```

Figure 5: Receiver-offline scenario

Figure 2, figure 3, and figure 4 show the sever terminal, sender device terminal, and receiver device terminal respectively. After starting up, the server waits for the sender device to connect. Once it is connected, it prints the IP address of the sender and the port number assigned to it. The sender then starts the authentication procedure. On the sender side, one can see the certificate details of the server. For project purposes, self-signed (self-created) certificates were used. The sender authenticates it by checking the presence of the certificate in its own set of known certificates. The latency incurred due to this authentication step is approximately equal to 0.65 to 0.70 seconds. After the authentication procedure, it receives the message from the sender and prints out the next task. It also sends an acknowledgement to the sender. The server then waits for the receiver to connect and starts a 10 second timer. Once

the receiver connects to the server (within 10 seconds) and has authenticated the server's identity, the server tries to forward the message to it. After getting acknowledgement from the receiver, the server relays that acknowledgement to the sender to mark the end of one successful end-to-end transmission and reception. If the timer at the server runs out (Figure 5), the server sends a negative acknowledgement to the sender, stating that the operation waited for 10 seconds, but the receiver was offline and asks the sender device to try again later. This inclusion of a timer helps move the complexity from the end devices to the server side in order to keep application devices as simple as possible.

5. Roadblocks

There were quite a few roadblocks encountered during the whole course of the project. Right from not knowing where to start to being confused about how to implement a certain design, it was a huge learning process. The main hurdle was to chalk out the key design considerations. After careful ponderance, it was decided to include the three considerations mentioned in the report above (authentication, acknowledgements, elimination of dependencies). Firstly, about authentication, there was not much on the internet about it. The usage of certificates is quite less explored field and a lot of studying, digging up online, and researching was involved. After the concept of certificates and the procedure to implement authentication was carefully understood, it was decided to create self-signed certificates. Second difficult thing was the elimination of dependencies. It was necessary to move the complexities away from the end devices to keep them simple. Thus, the concept of buffer and a timer was incorporated to enable this.

6. Future Work

This prototype currently supports only one-way authentication, i.e., only the client devices authenticate the server's identity. The feature of server authenticating the client devices can also be included to add another layer of added security/authentication. Second thing that can be added is encryption. The current work only supports certificate validation and not data encryption. Due to this, the unencrypted message can be read by any unintended attacker. The server can also misbehave by reading the message before forwarding. Thus, transmission of sensitive information forms a major risk that is yet to be addressed. Incorporation of public and private keys can be one of the possible solutions to this problem. For example, a sender can use the receiver's public key to encrypt a message. This encrypted message can be decrypted only with the help of the corresponding private key which is known only to the intended receiver. Thirdly, this prototype currently supports only strings of characters. Support can be extended to include various types of files like audio/video files, image files, etc. One more thing that can be done is creating an interface for a more convenient user experience instead of the existing terminal-like interface.

7. Summary and Insights

In this project, a basic IoT platform was created. The platform supported basic network communication via Wi-Fi in which a message (string of characters) was transmitted from one device to the other via a server. Authentication, acknowledgements, and dependencies' elimination were the three design constraints taken into consideration to make it a more complete platform. Authentication was given due importance since security should never be overlooked in any application. Complexity was moved away from the end devices to make them simpler. The server was made smarter instead. The bandwidth required by the network depends on the message to be transmitted. A successful IoT platform has following features among many more: protocol support, scalability, security, data integration and analytics. Out of the above 4, scalability and security have been looked at and explored. One can extend this prototype to include multiple devices and components and scale the system as per need. Also, security forms one of the most important aspect which is often overlooked. This project aimed to elevate the importance of security in a general purpose IoT setting.

REFERENCES

[1] https://en.wikipedia.org/wiki/Internet_of_things

[2] https://docs.devicehive.com/docs/101-overview

[3] https://aws.amazon.com/iot/?nc=sn&loc=0

[4] https://us.norton.com/internetsecurity-how-to-ssl-certificates-what-consumers-need-to-know.html