Självständigt arbete på grundnivå

Independent degree project - first cycle

Dator Teknik
Computer Engineering

Quantitative Comparison of SensibleThings and Dweet.io

Yun Zhao
Abstract

The objective of this paper is to perform a quantitative comparison of Dweet.io and SensibleThings from different aspects. With the fast development of internet of things, the platforms for internet-of-things face bigger challenges. This paper will evaluate both systems in four parts. The first part shows the general comparison of input ways and output functions provided by the platforms. The second part shows the security comparison, which focuses on the protocol types of the packets and the stability during the communication. The third part shows the scalability comparison when the value becomes bigger. The fourth part shows the scalability comparison when speeding up the processes. After the comparisons, I concluded that Dweet.io is more easy to use on devices and supports more programming languages. Dweet.io realizes visualization and it can be shared. Dweet.io is safer and more stable than SensibleThings. SensibleThings provides more openness. SensibleThings has better scalability in handling big values and quick speed.

Keywords: Dweet.io, SensibleThings, internet-of-things
# Table of Contents

Abstract..............................................................................................................iii

Terminology........................................................................................................vi

1 Introduction.................................................................................................1
1.1 Background and problem motivation.....................................................1
1.2 Overall aim.............................................................................................1
1.3 Concrete and verifiable goals.................................................................2
1.4 Scope........................................................................................................2
1.5 Outline....................................................................................................2

2 Theory.............................................................................................................4
2.1 IoT basics................................................................................................4
2.2 Typical IoT system structures..............................................................5
2.2.1 Client-server architecture...............................................................5
2.2.2 Peer-to-peer architecture...............................................................5
2.3 IoT network protocols...........................................................................5
2.3.1 UDP.................................................................................................5
2.3.2 TCP................................................................................................6
2.3.3 TLS.................................................................................................6
2.4 IoT platforms........................................................................................8
2.4.1 SensibleThings................................................................................8
2.4.2 Dweet.io........................................................................................9
2.5 Raspberry PI..........................................................................................9

3 Methodology................................................................................................11

4 Implementation............................................................................................12
4.1 General Comparison.............................................................................12
4.1.1 Input...............................................................................................13
4.1.2 Output.............................................................................................14
4.2 Security Comparison.............................................................................15
4.2.1 Capture packages on Dweet.io.......................................................16
4.2.2 Capture packages on SensibleThings.............................................17
4.3 Bigger value Scalability........................................................................21
4.3.1 Calculate RTTs with a big value on Dweet.io...............................22
4.3.2 Calculate RTTs with a big value on SensibleThings.....................24
4.3.3 Common file functions...................................................................26
4.4 Quicker Speed Scalability.....................................................................26
4.4.1 Calculate RTTs of publishing values at a certain speed................27
4.4.2 Calculate RTTs of getting values at a certain speed.......................29
4.4.3 Output results..................................................................................31

5 Results..........................................................................................................33
5.1 General Comparison.............................................................................33
5.1.1 Input...............................................................................................33
## Terminology

### Acronyms/Abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACK</td>
<td>Acknowledge.</td>
</tr>
<tr>
<td>APP</td>
<td>Application.</td>
</tr>
<tr>
<td>DES</td>
<td>Data Encryption Standard.</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things.</td>
</tr>
<tr>
<td>RUDP</td>
<td>Reliable User Datagram Protocol.</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Socket Layer.</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol.</td>
</tr>
<tr>
<td>TLS</td>
<td>Transport Layer Security.</td>
</tr>
<tr>
<td>UCI</td>
<td>Unique Client Identifier.</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol.</td>
</tr>
<tr>
<td>M2M</td>
<td>Machine to Machine.</td>
</tr>
</tbody>
</table>
1 Introduction

I'm an exchange student majoring in Computer Engineering in Mid Sweden University from DongHua University. This is my bachelor thesis report about IoT (Internet of Things) platforms. The aim of the project is to compare two platforms from different aspects.

1.1 Background and problem motivation

The use of Internet of Things has grown since the concept of a network of smart devices was discussed in 1982.[1] IoT is a network of physical objects, devices, vehicles, buildings and other items which are embedded with electronics, software, sensors and network connectivity which enables these objects to collect and exchange data. The IoT allows objects to be sensed and controlled remotely across existing network infrastructure,[2] creating opportunities for more direct integration of the physical world into computer-based systems, and resulting in improved efficiency, accuracy and economic benefit IoT Experts estimate that the IoT will consist of almost 50 billion objects by 2020.[3] The British Chancellor of the Exchequer George Osborne, posited that the Internet of Things is the next stage of the information revolution and referenced the inter-connectivity of everything from urban transport to medical devices to household appliances.[4] In order to provide IoT services, the platforms that can provide the management and control of networked objects are being developed. Nowadays, there are many IoT platforms on the market. Different IoT platforms have different features. It's essential to explore which platform has better openness, security or scalability and which platform is poor at fault handling or transparency. That's important for users to properly choose the platform.

1.2 Overall aim

The project's overall aim is to compare two different IoT platforms running on limited devices. IoT platforms should be human-friendly because they will be widespread and be used by everyone, including those who know nothing about coding. Although the Internet of Things offers enormous potential, such as expanding information access, enhancing the transparency of the government and empowering citizens, the threats of privacy are immense and its security becomes the biggest concern. IoT platforms are developed so rapidly without proper consideration of security challenges that the information produced by smart devices can be analyzed through network traffic and hackers can remotely control these internet-connected appliances including televisions, kitchen appliances, cameras and thermostats. In addition, IoT platforms face bigger challenges to enable better management of smart cities and systems, which requires large scalability. The problem I will solve in this thesis is to determine which IoT platforms fit best for different scenarios.
1.3 Concrete and verifiable goals

To achieve the above overall aim, a few concrete and verifiable goals have been listed. I have derived these following step-by-step goals:

G1: Learn about the basic concept, the unique addressability of things, the trends and characteristics of IoT.

G2: Learn about the principle and basic architecture of SensibleThings and test its codes on different devices.

G3: Learn about the principle and basic architecture of Dweet.io and test its codes on different devices.

G4: Determine which platform has more ways to publish the data and provide more output functions for users.

G5: Determine which platform is safer and more stable to transfer messages.

G6: Determine which platform is preferred in scenario where a device sets a big-size value.

G7: Determine which platform is preferred in scenario where a device speeds up the transfer.

1.4 Scope

The study has its focus on comparisons of SensibleThings and Dweet.io on Raspberry Pi and the personal computer. I evaluate the performances of two platforms from input ways, output functions, security and stability during the transfer and the scalability. I compare their scalability by increasing the size of the value and speeding up the transfer processes. Except for the public bootstrap of SensibleThings, I compare the local bootstrap of SensibleThings, which runs its own bootstrap on the device. The survey's conclusions should be generally valid for SensibleThings Beta 6. The project doesn't focus on adding nodes to compare the scalability because I don't have enough devices to perform it.

1.5 Outline

The structure of this thesis is organized as follows. Chapter 1 introduces the background and motivation of this research work, both overall goals and concrete. Chapter 2 provides some background theories which are required for the reader's understanding of the report and tools used in the project. Chapter 3 introduces how the methodology is to accomplish the project, which enables readers to interpret my results and repeat my work. Chapter 4 provides detailed process of implementation. Chapter 5 gives the results of the comparisons and
the line charts produced by the data. Chapter 6 is the summary of the whole research work, my own conclusions and the proposals of the future work.
2 Theory

This chapter introduces the theory and related work the paper refers to. First part is about the basic concept of internet of things. The second part introduces the typical system structures of IoT. The third part describes typical network protocols, UDP (User Datagram Protocol), TCP (Transmission Control Protocol) and TLS (Transport Layer Security). The fourth part introduces the IoT platforms compared in this project. The fifth part introduces Raspberry PI.

2.1 IoT basics

The Internet of Things is a system of interrelated computing devices, mechanical and digital machines, objects, animals or people that are provided with unique identifiers and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction.[5] The Internet of Things allows objects to be sensed and controlled remotely across existing network infrastructure.[2] It means that from any time, any place connectivity for anyone, we will now have connectivity for anything. A “thing” can refer to a wide variety of devices and it is defined as inseparable mix of hard ware, software, data and services by the legal scholars. These devices collect and publish the required data with the current technologies and then automatically communicate data with other devices without supervise. The internet of things is infiltrating all of our devices. Figure 1 shows a list of countries with the most IoT devices. Sweden ranks 7th, which means IoT is popular in Sweden.

![Figure 1. Countries with the most IoT devices](image-url)
2.2 Typical IoT system structures

There are two typical IoT system structures, client-server and peer-to-peer. They have different advantages and disadvantages. The choice of these two networks basically determines the relationship each PC or device on the network has with the other in terms of control. [6]

2.2.1 Client-server architecture
The client–server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients.[7] The client program interact with the server program in always-on host computers. Once the server has completed the client's request, the connection is terminated. Usually, both the client and server programs are part of a large program or application, since multiple clients share the same server program and the server manages the shared resources. However, the centralization of service provision and management implied by placing a service at a single address doesn't scale well beyond the capacity of the computer that hosts the service and bandwidth of its network connections.

2.2.2 Peer-to-peer architecture
In this architecture, there is no special machine that provides a service or manages the network resources. Instead, all of the processes involved in a task or activity play similar roles, run the same program and offer the same interfaces to each other. Every node interacts cooperatively as peers. Peer-to-peer architecture has been successfully constructed that enable tens or hundreds of thousands of computers to provide access to data and other resources that they collectively store and manage.[8]

2.3 IoT network protocols

Typical IoT network protocols are organized in layers. The Internet’s transport layer transports application-layer messages between application endpoints. There are two transport protocols in the internet, TCP and UDP. RUDP (Reliable User Datagram Protocol) is an advanced version of UDP. TLS protocol is the successor to the SSL (Secure Socket Layer). SSL enhances TCP with security services.[9]

2.3.1 UDP
UDP is a no-frills, lightweight transport protocol, providing minimal services. It is connectionless. There is no handshaking before the two processes start to communicate. When a process sends a message into a UDP socket, UDP provides no guarantee that the message will ever reach the receiving process.[9] RUDP aims to provide a solution where UDP is too primitive because guaranteed-order packet delivery is desirable. RUDP adds ACK (acknowledge)
of received packets, windowing and flow control, retransmission of lost packets and over buffering to extend UDP to ensure quality.[10]

2.3.2 TCP

TCP is said to be connection-oriented because before one application process can begin to send data to another, the two processes must first “handshake” with each other—that is, they must send some preliminary segments to each other to establish the parameters of the ensuing data transfer.[11] Figure 2 shows the process of TCP handshaking.

![Figure 2. TCP handshaking](image)

2.3.3 TLS

TLS is a protocol that ensures privacy between communicating applications and their users on the Internet. When a server and client communicate, TLS ensures that no third party may eavesdrop or tamper with any message.[11] TLS is composed of two layers: the TLS Record Protocol and the TLS Handshake Protocol. The TLS Record Protocol provides connection security with some encryption method such as the DES (The Data Encryption Standard).[11] The TLS Record Protocol can also be used without encryption. Figure 3 shows a TLS session between a client and web server.
2.4 IoT platforms

IoT platforms provide a common link for IoT devices to connect to the Internet superhighway. There are many IoT platforms which offer open source frameworks in the market now. I focus on two IoT platforms in this project, SensibleThings and Dweet.io.

2.4.1 SensibleThings

The SensibleThings platform enables a large scalable IoT. It uses fully distributed system in a peer-to-peer manner to avoid any single point of failure. The SensibleThings utilizes hash tables to enable logarithmic scaling when the number of entities increases in magnitude. It uses RUDP protocols to minimize the response time. Figure 4 shows the illustrative figure for GET and Figure 5 shows the illustrative figure for SET on the public bootstrap of SensibleThings. The source node registers a sensor first. Then the sink node resolves the UCI (Unique Client Identifier). When the sink node get or set the value, the “getEvent” or “setEvent” functions will be triggered on the source node. When source node sends the value back using “Notify”, the “getResponse” function will be triggered on the sink node.
SensibleThings enables developers to develop new functions for the platform. It provides the local bootstrap. Users can run the local bootstrap on their own device and run the local client to connect to the bootstrap.

### 2.4.2 Dweet.io

Dweet.io is a kind of ridiculously simple messaging (and alerts) for the internet of things. It's fast and free. It is built in client-server architecture. It's M2M (Machine to Machine) for the IoT the way it was meant to be. To publish data from the thing or read the latest value for a thing, simply call a URL. The query parameter can be added as key-value pairs or any valid JSON data. JSON is a lightweight data-interchange format [13] and it is syntax for storing and exchanging data. It is easy for machines to parse and generate. It is also possible to use client libraries on Dweet.io. Things can be locked on Dweet.io if users are willing to pay $1.99 a month. A locked thing provides more functions to users.

### 2.5 Raspberry PI

The Raspberry pi is a series of credit card-sized single-board computers developed in the UK by the Raspberry Pi Foundation.[14] It can be used as an IoT device. It is like a small computer. In order to control the Raspberry Pi, I use an existing application called “Fing” to detect its IP address. I use “Bitvise
SSH Client” to connect to the PI. Set the host as the IP address and the port as 22. When the connection established successfully, input “sudo tightvncserver” in its terminal console to open the vnc server, launch a desktop and set a password for this desktop. I can control the pi by “Vncviewer” with the password. By this way, there is no big difference between Raspberry PI and computer in this project. Figure 6 shows running the JAVA code on the Raspberry PI by “vncviewer”.

![Figure 6. Run JAVA code on the Raspberry PI](image)
3 Methodology

The methodology part is very important part in a project. This chapter will introduce the methodology and concrete methods used to achieve the research work.

To accomplish G1 that has an understanding of IoT, I learn the general concept, history, development and future challenges regards to internet of things, which excites my motivation and guide me to know the current condition of IoT platforms and how to use them.

To accomplish G2&G3 that learn about SensibleThings and Dweet.io, I learn the basic principles, algorithms and protocols at first to know about the mechanism of the IoT platforms. It's essential to identify their weaknesses and strengths. Then I focus on the open source codes, it is helpful to build a basic idea of application development and know how the platforms operate. I'm able to modify the code from the client libraries or from GitHub to create the seniors I need. Since platforms are different from their original version and the world is changing every second, the comparison needs to keep pace with the times. Therefore, the state-of-the-art research papers mustn't be overlooked.

To accomplish G4 that determines which platform has more ways to publish the data and provide more output functions for users, I download all of which the platforms need and run the codes on computers and Raspberry Pi according to step-by-step tutorials listed on the official website. I search the mobile application on the Google Play store and App Store, download them and test them on smart phones.

To accomplish G5 that determines which platform is safer and more stable to transfer messages. I use “Wireshark” to capture the packages through the transfer process and analyze what kinds of protocols are used by the platforms. I will compare the protocol types. I will run the codes on different devices for many times to compare the captured packages to see whether the connection is stable.

To accomplish G6 that determines which platform is preferred in scenario where a device sets a big-size value and P7 that evaluate how the platform perform when a device speeds up the transfer process, I edit the JAVA code downloaded from client libraries of dweet and from official website of SensibleThings to create the scenarios. I compare the round trip time, its standard deviation and failure rate to evaluate different platforms.
4 Implementation

As the Figure 7 shows, this chapter talks about how to compare Dweet.io and SensibleThings in terms of the general comparison, security, bigger value scalability and quicker speed scalability.

![Quantitative Comparisons](image)

**Figure 7. Comparisons of Dweet.io and SensibleThings**

4.1 General Comparison

This part mainly includes two small parts. The first part is from input aspect. Users can use different devices and tools to publish data to IoT platforms. The second part is from output aspect. After publishing data, IoT platforms can provide some functions. Figure 8 shows the general structure of this part.
4.1.1 Input

In order to publish data to IoT platforms, users can use different devices. Both of Dweet.io and SensibleThings claim that they can run on computers, Raspberry Pis and smart phones.

In order to publish a value to Dweet.io, there are many methods. I can call a URL like: `https://dweet.io/dweet/for/my-thing-name?hello=world` in any web browsers and replace “my-thing-name” with a unique name. “hello=world” is the query parameters and the parameters added to request must be a kind of key-value pairs or any valid JSON data. Dweet.io has many client libraries as well. JavaScript, Python, Java, Ruby and C programming languages are available on the devices. I download JAVA codes from the GitHub, establish a project for it and configure the google JSON library. In this project, I run the codes in “Eclipse” on the computer and in “BlueJ” on Raspberry PI to test the JAVA code.

URL method is not available on SensibleThings and Java is the only programming language that programmers can use. I run the source code and then the sink code to use the public bootstrap of SensibleThings. These two projects can be run on the same device or different devices. I modify the UCI in the code to let it be a unique one. The UCI in the sink code should be the same as the source code. SensibleThings provides the local bootstrap for users. I run the local bootstrap code and then the local client code. The UCI in the client code should also be the same as the one in bootstrap code. It is also possible to run these two projects on the same device or different devices.
I can use URL to publish values and get the latest value in the web browsers on the mobile devices. I search “dweet my phone” in the Google Play of Android and APP (application) store of Apple and download the app. As shown in Figure 9, I can publish parameters of my devices and set the publishing frequency and the thing name in this app.

SensibleThings is only available on Android now. I run Android code on “AndroidStudio” application and modify the IP address in the code. I connect my Android phone to the computer to download the apk. But it doesn't work on my phone because of the poor compatibility.

4.1.2 Output

Dweet.io cooperates with freeboard.io. The thing published to Dweet.io can be used as a data source on the freeboard. As shown in Figure 10, I can build real-time, interactive dashboards and visualizations in minutes using the intuitive drag & drop interface on the freeboard.io. I can add any types of widgets as I want.
Dweet.io cooperates with Twitter as well. I can not only copy the URL of the thing to my friends, but also share the data on my Twitter.

Dweet provides locked things for those who are willing to pay $1.99 a month. A locked dweet has 30 day storage and it allows users to protect and reserves the things so that only people with special keys can access it. The name of the locked thing is unique and it cannot be used by anyone else. When something in the data falls outside set of conditions, alerts can notify users by SMS or email. However, I fail to pay for it. The payment page doesn't work properly as shown in Figure 11.

Disappointingly, there are no such ready-made functions on SensibleThings. I can only get the latest value at the terminal consoles now.

4.2 Security Comparison

I modify the JAVA code downloaded from their official websites and use Wireshark to capture the packages during the publishing and getting processes of IoT platforms. For the sake of fairness, I publish a JSON object with “hello=world” property to two platforms. I analyze these packages to see what
kind of protocol they use and how many packages do they need. I run the same code on different devices for several times to see whether the sum of the protocols and the communication process are changed.

### 4.2.1 Capture packages on Dweet.io

As shown in Figure 12, I replace the name of the thing with ”yun-client”, a unique thing name that avoid being replaced by others as far as possible. I delete all the functions about a locked thing in the original code because we only focus on publishing and getting latest value, which can be compared with SensibleThings. I open Wireshark, use wireless network interface to capture from and click the “Start” button. Then, I run the JAVA code. After the return value shown in the terminal console, I click the “Stop” button in the Wireshark and use “ip.addr==dweet.io” to filter the packages.

![Figure 12. Capture packages of Dweet.io](image)

Figure 13 shows the packages of Dweet.io captured by “Wireshark”. When a device publishes the value to Dweet.io, three TCP handshaking packages go first. Then, Dweet.io uses 14 TLS packages to ensure the privacy between device and server. After the TLS handshaking layer, it takes 4 packages to publish the JSON object and 2 packages to get the value. When the program ends, it takes one package to stop the TCP connection.
I run the same code on different devices and capture the packages for several times. The packages are all the same as shown in Figure 13 and there are fixed 24 packages during the communication.

4.2.2 Capture packages on SensibleThings

In order to capture the packages during the communication, I establish “STSource” and “STBootstrap” projects on one device and “STSink” and “STClient” projects on another device in the local area network. It is not possible to capture packages if they are run on the same device or behind the NAT. It is a must to configure the JSON library and SensibleThings library in all projects because we need to publish a JSON object with “hello=world” property.

As shown in Figure 14, I use “yunpublic@miun.se/test” as UCI and set the JSON object with “hello=world” property as the value on the source node in advance to ensure the node will return a required value to the sink node. I run the source code, start “Wireshark” and run the sink code. After the terminal console on the sink node shows the return value, I click “Stop” button and use “UDP” filter to get relevant packages. The processes of registering UCI and resolving UCI are not included. Unexpectedly, the public bootstrap of SensibleThings continuously use TCP protocols instead of UDP protocols claimed on the official website. It is hard to identify these TCP packages. I give
up looking deep into the public bootstrap of SensibleThings and turn to studying the local bootstrap.

Figure 14. Capture packages on public bootstrap of SensibleThings

As shown in Figure 15, I use “yunlocal@miun.se/test” as the UCI and set a JSON object as the value on the bootstrap node in advance. I alter the ip address of bootstrap into the IP address of local bootstrap node on the client node so that it can connect to the local bootstrap. I run the local bootstrap, click “start” in “Wireshark” and run client node. After the terminal console on the client node shows the return value, I click “Stop” button and use “udp.port==9009” to filter the packages.

Figure 15. Capture packages on the local bootstrap of SensibleThings
I open one of the packages to see its binary stream. According to the paper "Reliable UDP and Circular DHT implementation for the MediaSense Open-Source Platform", the bits in the seventh byte in the data layer are flags and I can identify the packets by these flags as shown in Figure 16. However, the flags don't help the analysis so much. The length of ACK packets is 65 or 69. Other packets are data packets.

![Figure 16. The flags in the packet](image)

- Flags (8 bits) - bit field containing flags
  - First (bit 0) - first packet containing application layer data
  - Reset (bit 1) - request to reset sender state
  - ACK (bit 2) - packet contains acknowledge information
  - Data (bit 3) - packet contains application layer data
  - Resend (bit 4) - packet was resent
  - Fragment (bit 5) - packet is fragmented
  - Persist (bit 6) - request an acknowledge packet

Then I right-click on one of the packages, copy its Hex stream bytes and paste them in “HxD” application. There is “1F 8D” in this package and the compressed file often starts with “1F 8D”. I delete the header of the stream and make the file begin with “1F 8D”. Then, I save the file as “pkg.zip” and open the zip file by “WinZip” and open the “pkg” file by notepad. The source IP address, destination IP address and the type of the message is shown in the notepad in Figure 17.
I run the same code on different devices and capture the packages for several times. I find that the sum of the packages changes from 17 to 20. Two devices randomly send ACK packages.

4.3 Bigger value Scalability

This part shows how to evaluate the performance of the IoT platforms when values become bigger. As shown in Figure 18, I calculate the RTTs of publishing a value to different platforms and RTTs of getting the value. To ensure the fairness, the value is a JSON object with SIZE properties. I calculate the RTT and write the effective RTT into the file for SUM times. If it fails, print out the failure. If it succeeds in writing SUM RTTs in the file, file functions will calculate the average RTT and its standard deviation. I will record the results, clear the file and set a bigger SIZE to run the same process again. This part is sub-divided into three small parts. The first part shows how to calculate RTTs of getting and publishing a big value on Dweet.io and how to determine whether the RTT is effective. The second part shows how to implement it on SensibleThings and the third part talks about the common file functions.
4.3.1 **Calculate RTTs with a big value on Dweet.io**

As shown in Figure 19, in order to avoid unnecessary influence, I publish an irrelevant value for Dweet.io before publishing the required value to make sure the connection has no problem and the time of TCP&TLS handshaking is not included. Then I record T1 before I publish the required value with SIZE properties. I record T2 when the device receives the return value. If it returns true, the effective RTT of setting a value with SIZE properties for Dweet.io equals T2 minus T1. If publish function returns false, “fail to publish dweet with SIZE properties” will be printed out at the terminal console. When I set the SIZE to 1300, the terminal console shows the failure message. I change the SIZE to find the boundary of the size and find that I can publish a value with at most 1252 properties to Dweet.io. Since the boundary size is a JSON object with 1253 properties, I set the value size from 1 property to 1250 properties in the comparison of bigger value scalability.
As shown in Figure 20, I publish the value with SIZE properties at first. The TCP&TLS handshaking and publishing time is not included. If it is published successfully, I record T3 before the device sends the request to Dweet.io and record T4 when the device receives the return value. If “get” function returns null or “publish” function returns false, the terminal console will output the failure. If everything goes well, RTT of getting a value with SIZE properties equals T4 minus T3 and the program will execute the common file functions.
4.3.2 Calculate RTTs with a big value on SensibleThings

In this part, the method of calculating RTTs on the local bootstrap and public bootstrap of SensibleThings is the same. But the results are not the same.

As shown in Figure 21, I record T5 on the sink (or client) node before the node publishes the value, where value equals T5. I record T6 when the source (or bootstrap) node receives the value. The RTT of setting a value with SIZE properties for SensibleThings equals T6 minus T5. The resolve time is not included. The “value” sent by sink (or client) node is printed out at the terminal console on the source (or bootstrap) node to see whether it succeeds in setting. If the output value is the same as the published one, the RTT is effective. If not, it's a failure and the results of common file functions are not reliable. It should be noted that two projects must be run on the same device in case of the unsynchronized time.
As shown in Figure 22, I set the value as a JSON object with SIZE properties on the source (or bootstrap) side at first. I record T7 on the sink (or client) node before it sends a request. The source (or bootstrap) node will return the JSON value to the sink (or client) node. I record T8 when the sink (or client) node receives the value. The RTT of getting the value with SIZE properties for SensibleThings equals T8 minus T7. The resolve time is not included. The value sent by source (or bootstrap) node is printed out at the terminal console on the sink (or client) node. If the output is a value with required properties, the RTT is effective. If not, the results of common file functions are not reliable.
4.3.3 **Common file functions**

This part introduces some common functions related to files which are used in the bigger value scalability comparison part. As shown in Figure 23, if the program gets an effective RTT, the program will write the RTT into a file and count for the written RTTs. If there are SUM RTTs in the file, the program will calculate and print out the average RTT and the standard deviation of RTTs. I will record the results in a table.

1. **Void WriteFile(long RTTtemp):** write the effective RTT into the file.

2. **Boolean CountRTTs():** count for how many records in the file. If there are SUM records, return true. If not, return false.

3. **Void Calculate():** if there are SUM records in the file, calculate the average RTT=RTTsum/SUM, standard deviation= Math.sqrt((RTT[i] - averageRTT)^2/SUM)

![Flow chart of common file functions](image)

**Figure 23. Flow chart of common file functions**

4.4 **Quicker Speed Scalability**

This part shows how to evaluate the performance of the IoT platforms when speeding up the transfer. In this evaluation, I set the published and returned value as a timestamp. As shown in Figure 24, I continuously publishing or getting the value at a certain speed. I enable the thread to sleep to control the speed. I use two “ArrayList” containers to record the begin time and the end time. I use one array “RTT” to record RTTs for every set & get. If it fails, the program pluses the “failcount”. “failcount” is used to calculate the failure rate. If it succeeds, print out the RTT[i] and its index. When there are SUM RTTs at
the terminal console or there is no more values return, I press “Enter” to end the listener. The program will calculate the failure rate, timeDrift, averageRTT and its standard deviation and output these results at the terminal console. This part is sub-divided into three small parts. The first part shows how to calculate SET RTTs at a certain speed. The second part shows the implementations on calculating GET RTTs. The third part talks about the output results.

![Flow chart of quick speed comparison](image)

**Figure 24. Flow chart of quick speed comparison**

### 4.4.1 Calculate RTTs of publishing values at a certain speed

I continuously publish the value at a certain speed to different platforms to calculate the SET RTTs for SUM times.

As the Figure 25 shows, I publish an irrelevant value at first to avoid irrelevant factors. Then I record T1 before the device publishes the value, where the value equals T1. When the device receives the ACK packet from Dweet.io, I record T2. I add T1 into “begin” and add T2 into “end”. If Dweet.io returns false, the program plus the “failcount”. Then the thread sleeps. After the thread wakes up, it publishes the timestamp again. After publishing the values for SUM times, the program calculates the output results.
The method of calculating RTTs on the local bootstrap of SensibleThings is the same as the public bootstrap, but the results are different. As the Figure 26 shows, I record T1 on the sink (or client) node before I publish the value1, where value1 equals T1. Then the sink (or client) node sleeps. After the thread wakes up, sink (or client) node publishes the timestamp again. The sink (or client) will publish timestamps for SUM times. When the source (or bootstrap) node receives the published value from sink (or client) node, it will output the index and RTT[i], where RTT[i]=currentTime-value. When there are SUM RTTs shown at the terminal console on the source (or bootstrap) node or no more RTTs printed out, I press “Enter” and the program will calculate the output results. Since the end time and the begin time of the RTT is not on the same device, this experiment must run on the same device.
4.4.2 Calculate RTTs of getting values at a certain speed

I continuously get the value at a certain speed from different platforms to calculate the GET RTTs for SUM times.

As the Figure 27 shows, I publish a timestamp as a value at first to ensure the return values of two platforms are the same. I add T1 into the “begin” and add T2 into the “end”. After the device gets the value, the thread sleeps. When the thread wakes up, it sends the request again. If Dweet.io returns null, the program pluses the failcount. After sending the requests for SUM times, the program calculates the output results.
The method of calculating RTTs on the local bootstrap of SensibleThings is the same as the public bootstrap, but the result is not the same. As the Figure 28 shows, I set the value as a timestamp on the source node at first to ensure the fairness. I record T1 on the sink (or client) node when the request is sent and record T2 when it returns. The program will calculate and output the RTT[i] and its index on the sink (or client) node. The thread starts to sleep after it sends the request. After the thread wakes up, the sink (or client) node will send the request again. When there are SUM RTTs at the terminal console in the sink (or client) node or no more RTT printed out, I press “Enter” and the program will calculate output results. The GET RTT can be run on one or two device. I test both circumstances to see its difference.
Output results

Figure 29 shows the output results of setting value every 506.25ms when SUM=5, requiredSpeed=500ms. The program prints out every RTT and its index, the name of the file, SUM value, failure rate, timeDrift, average RTT and its standard deviation. This part will explicit how to calculate them.

AverageRTT = \frac{\text{RTTsum}}{i}, \text{where “i” means the sum of the effective RTTs.}

Standard deviation = \sqrt{\frac{\sum (\text{RTT}[i] - \text{averageRTT})^2}{i}}, \text{where “i” is the same value in the averageRTT. The standard deviation shows the reliability of the calculated RTT and stability of the connection.}
(3) Time drift=realTimeInterval- requiredSpeed where realTimeInterval = (begin[SUM-1]-begin[0])/SUM. Figure 30 shows the process of getting and setting values on Dweet.io, the real time interval is not the same as the sleep time interval and the required speed. Because the program running on Dweet.io cannot send a packet before the previous packet returns. That's why I calculate the time drift. I not only calculate the timedrift on Dweet.io, but also on the SensibleThings because I want to ensure the time interval is what I want to test. If time drift is bigger than 10ms, I will modify the "sleepTime" in the code to meet the required speed and run the program again. Sometimes it is hard to control the real time interval. I record the time drift to see whether the records are reliable enough. But it doesn't make any sense actually. In addition, I modify the sleep time interval to 0ms to look for the boundary speed of platforms and find that the program cannot publish or get values quicker than 125ms on Dweet.io.

(4) Failure rate= failcount/SUM =(SUM-i)/SUM, where “i” is the same as in the averageRTT. When calculating the set RTTs of SensibleThings, the “set” function doesn't return. I can calculate the “i” on the source (or bootstrap) node and use it to calculate the failure rate.
5 Results

This chapter will show the results of the project, which contains four main parts. The first part is the general comparison in terms of input and output aspects. The second part is the security comparison where we analyze the protocols during the communication. The third part is the scalability comparison when value becomes bigger. The fourth part is the scalability comparison when the speed becomes quicker.

5.1 General Comparison

The general comparison splits into input and output parts.

5.1.1 Input

As shown in Figure 31, Dweet.io can use URL to publish and get values in any web browsers on any devices. Except for the URL method, Dweet.io can support much more programming languages, while SensibleThings only supports Java programming language. If you want to use the IoT on the SensibleThings, you must run at least 2 projects. If you want to develop new functions for SensibleThings, you can use the local bootstrap of SensibleThings, which provides more openness for developers. Users can only publish and get values from Dweet.io. If users want to use the smart mobile devices on the Dweet.io, they can directly download released APP from digital distribution platforms on Apple and Android products. By contrast, SensibleThings is only available on Android now. Other than the poor compatibility, the app can only get the value from the platform.

<table>
<thead>
<tr>
<th>Input</th>
<th>SensibleThings</th>
<th>Dweet.io</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer</td>
<td>JAVA is the only language</td>
<td>provides more programming</td>
</tr>
<tr>
<td>Raspberry Pi</td>
<td>More openness: local bootstrap</td>
<td>languages</td>
</tr>
<tr>
<td>Smart mobile</td>
<td>Hard to run</td>
<td>Existing APPs</td>
</tr>
<tr>
<td>devices</td>
<td></td>
<td>All available through</td>
</tr>
<tr>
<td></td>
<td></td>
<td>URL, no set-up</td>
</tr>
</tbody>
</table>

Figure 31. Input comparison
5.1.2 Output

As shown in Figure 32, SensibleThings has no ready-made functions and users can only get the returned value at the terminal console now. By Dweet.io, users can visualize the data on the freeboard.io and share data with their friends by URL or Twitter. Dweet.io claims that alerts, privacy and storage functions are provided if users are willing to pay for the thing. However, the payment page has some errors. The functions of the locked things are empty promises.

<table>
<thead>
<tr>
<th>Output</th>
<th>SensibleThings</th>
<th>Dweet.io</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visualization</td>
<td>No ready-made functions</td>
<td>Freeboard.io</td>
</tr>
<tr>
<td>Share</td>
<td></td>
<td>URL or Twitter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>——— Payment page error! ———</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If you pay $1.99 a month successfully, notify you by email/ SMS when data falls outside specific condition, only people with special keys can access the unique thing, one month data storage</td>
</tr>
</tbody>
</table>

Figure 32. Output comparison

5.2 Security Comparison

As shown in Figure 33, Dweet.io establishes TCP connection at first and ensures the reliable and safe connections by TLS v1.2. Dweet.io has better stability. It takes fixed 24 packets to publish and get a value. It is not easy to analyze continuous TCP packets when two devices communicate with each other running on the public bootstrap of SensibleThings. RUDP protocol is used on the local bootstrap of SensibleThings. The packets are just compressed and it is easy to decompress them. The connection is unsafe and unstable. The sum of the packets changes from 17 to 20. ACK packets are randomly sent between two devices.
5.3 Bigger Value Scalability Comparison

Figure 34 shows the comparison when the value becomes bigger on two platforms. The following parts will explicit the results and show the line charts of the SET RTT, GET RTT and their standard deviations.

![Table showing security comparison between Dweet.io and SensibleThings](image)

![Table showing scalability comparison between Dweet and SensibleThings](image)
5.3.1 SET RTTs with bigger value

Figure 35 shows that when setting value, the program spends much more time on Dweet.io and the RTTs increase with the size of the values. When publishing a value with about 100 properties for the public bootstrap of SensibleThings, the program spends more time than Dweet.io. If the value becomes bigger than 400 properties, the RTTs of setting values for the public bootstrap of SensibleThings become shorter than 100 million seconds. The RTTs of setting values for the local bootstrap of SensibleThings are much less than the others.

![Figure 35. Line Chart of SET RTTs with bigger value](image)

5.3.2 GET RTTs with bigger value

Figure 36 shows that the program spends much more time on getting a value from the public bootstrap of SensibleThings. When the values are smaller than 400 properties, the Get RTTs are about 500 million seconds. When the values are bigger than 400 properties, the Get RTTs become stable and they are about 350 million seconds. The RTTs of getting values from Dweet.io slightly increase with the size and they are between 100ms and 200ms. The RTTs of getting values from the local bootstrap of SensibleThings are much less than the others.
Figure 36. Line Chart of GET RTTs with bigger value

5.3.3 Standard Deviation with bigger value

From Figure 37, we can see that the standard deviations of getting values from the public bootstrap of SensibleThings is the highest one, which means its RTTs are widely spread and unreliable. The average RTT of setting a value with 100 properties for the public bootstrap of SensibleThings is not reliable. The average RTT of setting a value with 1150 properties to Dweet.io is not reliable. Other standard deviations are mostly under 20 million seconds, which means the RTTs are closely clustered around the mean and reliable.
5.4 Quicker Speed Scalability Comparison

Figure 38 shows the comparison when the speed becomes quicker. The following parts will explicit how to get the results and show the line charts of the SET RTTs, GET RTTs, their standard deviations and the failure rate.

<table>
<thead>
<tr>
<th>SET RTT</th>
<th>Dweet.io</th>
<th>SensibleThings</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>stable shorter</td>
<td>Local Bootstrap</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shortest, ~1ms</td>
</tr>
<tr>
<td>GET RTT</td>
<td>stable shorter</td>
<td>Shortest, ~1ms</td>
</tr>
<tr>
<td>Failure Rate</td>
<td>Increases with quicker speed</td>
<td>No failure, 100% reliable</td>
</tr>
<tr>
<td>Boundary Speed</td>
<td>125ms</td>
<td>0.25ms</td>
</tr>
</tbody>
</table>

Figure 38. Quicker Speed Scalability Comparisons

5.4.1 SET RTTs with quicker speed

Figure 39 shows that when the program speeds up setting values, the public bootstrap of SensibleThings spends much more time. When speed is slower
Quantitative Comparison of SensibleThings and Dweet.io

Yun Zhao

2016-05-18

than 500ms, its RTTs are about 260ms. When speed is quicker than 500ms, its RTTs decrease with the speed. The RTTs of setting values for Dweet.io are from 120ms to 160ms. The RTTs of setting values for the local bootstrap of SensibleThings are much less than the others.

![Figure 39. Line Chart of SET RTTs with quicker speed](image)

From Figure 40, we can see that standard deviations are almost under 20ms on the local bootstrap of SensibleThings. The standard deviations of Dweet are from 20ms to 40ms. When the speed is quicker than 200ms, the average RTTs of setting values for the public bootstrap of SensibleThings is unreliable.

![Figure 40. Line Chart of SET standard deviation with quicker speed](image)

5.4.2 GET RTTs with quicker speed

Figure 41 shows that getting values from the public bootstrap of SensibleThings spends the longest time. It spends more time when two programs of
SensibleThings are running on two devices. Dweet.io spends about 130ms to get the value regardless of the speed. The RTTs of getting values from the local bootstrap of SensibleThings are much less than the others.

From Figure 42, we can see that standard deviations are almost below 40ms on the local bootstrap of SensibleThings and Dweet.io. The standard deviations of Dweet are from 20ms to 40ms. When the speed is 1900ms or 1300ms or 500ms, the average RTTs of getting values for the public bootstrap of SensibleThings on one device are unreliable. When the speed is 1800ms or 150ms, the average GET RTTs of the public bootstrap on two devices are unreliable.
5.4.3 Failure Rate on Dweet.io

Figure 43 shows that the failure rates of setting and getting values on the Dweet.io are the same. When the speed is quicker than 1s, failure rates are not zero any longer and the failure rate increases with the speed.

![Line Chart of Failure Rate on Dweet with quicker speed](image)

Figure 43. Line Chart of Failure Rate on Dweet with quicker speed
6 Conclusions

This chapter talks about the conclusions of the project and some discussion towards the results.

The project is a quantitative comparison of IoT platforms from different aspects. The project's overall aim is to compare the performances of Dweet.io and SensibleThings under different circumstances.

The overall aim and concrete goals have been accomplished well so far. I have achieved the general comparison, security comparison and the scalability comparison of Dweet.io and SensibleThings.

In general comparison, Dweet.io provides more ways to publish data and more output functions for users. If a user wants to use IoT on mobile devices, visualize the published data, share the data with friends or he is ignorant about JAVA programming, Dweet.io is better for him. If a user is familiar with JAVA, SensibleThings provides more openness.

In security comparison, Dweet.io uses TLS based on the TCP and it keeps a stable connection. The public bootstrap of SensibleThings uses TCP. It is not easy to identify the packages. RUDP is used by the local bootstrap of SensibleThings. But it randomly sends ACKs. The packages are easy to decompress. It's unsafe and unstable. Thus, if a user wants to have a safe and stable connection, he should choose Dweet.io.

In bigger value scalability comparison, run a local bootstrap of SensibleThings is the best if a user doesn’t care running two projects. Except for the local bootstrap, if the user wants to publish a value smaller than 1253 properties, Dweet.io is recommended. If the value has more than 1253 properties, the public bootstrap of SensibleThings is better.

In quicker speed scalability comparison, run a local bootstrap of SensibleThings is also the best if a user doesn’t care running two projects. Except for the local bootstrap, if the user wants to publish or get values quicker than 1000ms, Dweet.io is recommended. If the speed is slower than 1000ms, the public bootstrap of SensibleThings is better.

6.1 Ethical considerations

The comparisons on the SensibleThings and Dweet.io can be advices for users to choose the platform. I conduct these comparisons in my home in Sweden and use limited devices. I don't think there are any ethical considerations in my
conduct. However, there are some ethical considerations on the internet of things. It is possible for someone to monitor others' activity through the smart devices. Since internet of things has developed so quickly, security has not always been considered in product design. That will be a big problem. The designer of the IoT platforms should pay attention to the security.

6.2 Future work

When I look back to my whole project, I have ignored two things, one is the network delay and the other one is components quantity scalability. The network delay affects the results of the RTTs. Thus, the future work for me will pay attention to the network delay of the tests, such as when the network delay becomes longer, how big it will affect the performance of the IoT platforms. In addition, how many components can be run at the same time on the IoT platforms? How do the RTTs change with the number of the nodes? In addition, I didn't look deep into the packets captured on the public bootstrap of SensibleThings in this project. In future, I should observe the packets and analyze them.
References


https://www.xkl.com/the-internet-of-a-lot-of-things/
Retrieved 2016-05-17.

Retrieved 2016-05-17.

Retrieved 2016-05-17.

http://www.staidlogic.com/ClientServer.html
Retrieved 2016-05-17.


[10] ietf.org, ”RUDP”,
Retrieved 2016-05-17.

http://searchsecurity.techartarget.com/definition/Transport-Laye
Quantitative Comparison of SensibleThings and Dweet.io

Yun Zhao

2016-05-18

r-Security-TLS
Retrieved 2016-05-17.

[12] kaazing.com, "tls",
Retrieved 2016-05-17.

[13] JSON.org, "JSON",
http://www.JSON.org/
Retrieved 2016-05-17.