computers

Article

Connecting Smart Objects in IoT Architectures by Screen Remote Monitoring and Control

Zebo Yang * and **Tatsuo Nakajima**

Department of Computer Science and Engineering, Waseda University, Tokyo 169-8050, Japan; tatsuo@dcl.cs.waseda.ac.jp

* Correspondence: zebo@dcl.cs.waseda.ac.jp; Tel.: +81-3-5286-3185

Received: 22 August 2018; Accepted: 22 September 2018; Published: 24 September 2018

Abstract: Electronic visual display enabled by touchscreen technologies evolves as one of the universal multimedia output methods and a popular input intermediate with touch–interaction. As a result, we can always gain access of an intelligent machine by obtaining control of its display contents. Since remote screen sharing systems are also increasingly prevalent, we propose a cross-platform middleware infrastructure which supports remote monitoring and control functionalities based on remote streaming for networked intelligent devices such as smart phone, computer and smart watch, etc. and home appliances such as smart refrigerator, smart air-conditioner and smart TV, etc. We aim to connect all these devices with display screens, so as to make possible remote monitoring and controlling a certain device by whichever one (usually the nearest one) of display screens among the network. The system is a distributed network consisting of multiple modular nodes of server and client, and is compatible to prevalent operating systems such as Windows, macOS, Unix-like/Linux and Android, etc.

Keywords: Internet of Things; remote monitoring and control; smart home; cross-platform; middleware; distributed networking

1. Introduction

As innovation accelerates, we have witnessed a rapid progress on digital presentation of multimedia such as images, text, or video, layered on electronic visual display. Functionalities of intelligent devices and home appliances are prominently augmented with the conjunction of electronic display screens. For example, the virtual assistant Alexa of Amazon (Seattle, WA, USA) Echo [1] gets more polymorphous and anthropopathic with the addition of display screen: Amazon Echo Show [2]. LG (Seoul, South Korea) InstaView Refrigerator [3] with a transparent LCD touch-screen brings much more pleasure to the kitchen by supporting the setting of expiration dates, seeing contents inside, uploading photos, leaving notes and more. Electronic visual display has unquestionably turned into one of the most prevailing and accepted ways to interact with the processing systems of intelligent devices. However, at present, screens in our daily lives are mostly isolated from each other. We can only control them directly from the input widgets connected to the device and watch them directly from the output screen. Hanging around these devices and home appliances is laborious and irritating.

This paper aims to create a general middleware architecture to connect smart objects by screen remote monitoring and control (Intelligent devices and home appliances are generally denominated as smart objects in this paper.). Characteristically, the architecture is compatible with the current market of smart objects and extendable for future possibilities of intelligent products. The connection of multiple smart objects over the display screen will, up to a point, save time and efforts, constructing a convenient IoT (Internet of Things) environment indoors and outdoors. To prove that the middleware architecture is feasible and practical, we accomplished a modularized implementation based on the architecture, which supports a remote and permissive view and plays multiple smart objects over display screens. The implementation is composed of two portions: a scaffold framework for developers or manufacturers to integrate upon functionalities into their systems, and a software with a user interface layered on this framework, which supports simple installation and direct execution on different platforms. In other words, developers or manufacturers can choose to develop further over the framework or directly pre-install the software to their systems, and, as a user, one can choose to install it anytime on any compatible systems. Regarding the scaffold framework, it is practical for developers to run the middleware as a server and add streaming services over the system APIs or directly include the full framework into their software. The framework is cross-platform and compatible with most operating systems. Regarding the software implementation, which is based on the framework, users can simply use a web browser to view and play remote content over the web application after installation. The web application and server are simultaneously installed along with the software. The network of display screens resembles a pool of visual smart objects, among which users can arbitrarily control one or multiple devices by any device in the pool, with which many possibilities materialize.

Technically, the implementation is a cross-platform middleware adopting VNC (virtual network computing) [4] as a streaming intermediary, aiming to seamlessly connect all networked smart objects by display screens. The architecture resembles a composition of web servers, managing the VNC server and VNC viewer, and a web application, serving as user-interfaces for both mobile and desktop views. (VNC server is a program running on the source device, aiming to realize remote monitoring and control of display content. Correspondingly, the VNC viewer is a program running on the terminal device that provides connection, display and control abilities to the source device.) Furthermore, the VNC module here is fungible and can be replaced by other screen streaming or remote control methods. The VNC module is merely a decoupled independent extension to the system architecture and we adopted it for the following implementation.

Moreover, for future possibilities, the approach is extendable for projecting multiple screens to a single monitor, a chief behavior of a surveillance application. On the other hand, the middleware can also be integrated with outdoor screens under corresponding permissions, to fit in with the new networked environments: outdoor circumstances are increasingly covered by wireless networks [5] along with the evolution of ICT (Information and Communications Technology). For example, users can use their portable devices such as cell-phones, PDAs (Personal Digital Assistants) or laptops to display the instructional screen inside a department store for a better adoption of guidance. In addition, users are able to stream a wall-sized, high-resolution tiled display to portable devices [6,7]. Furthermore, if the middleware has been integrated with motion and voice commands [8], functionalities like automatically selecting a nearest screen based on the user's motion or volume can be effortlessly implemented. The middleware architecture on the whole may lead to many more possibilities, which become effective and sustaining in our daily lives. Implementations of the spontaneous projecting display save our time and effort from wandering around display screens and using them individually.

In the following section, background and related work of the approach are presented. Thereupon, we propose system architecture design and discuss related technologies. After that, we put forward application scenarios to illuminate and demonstrate the operating mechanism, which is followed by several essential issues and discussions. Lastly, we conclude the paper and prospect future possibilities.

2. Background

Digital multimedia proponents are eagerly seeking solutions that may work within the current regulatory confines of display technologies and other industries. Electronic visual display, as the primary display intermediary for multimedia content, has become extremely popular over the last few decades. It is also a prevalent mode to manipulate smart objects with the advancement of touch screen technologies. Innovative and fresh concepts of screen modalities are also strikingly emerging: the research of Ambient Wall, a smart wall display interface that can be controlled by simple gestures for smart homes [9], and technologies of a Smart Interactive Mirror [10]. However, most visual devices are currently being isolated from each other. People are confined physically by the location of smart objects and have to wander around when dealing with multiple tasks. Smart objects should have been absorbed into a broader form of object connectivity, which is the core concept of the Internet of Things (IoT). Since a remote screen sharing system like a VNC-based system makes it possible to get device control simultaneously when getting display contents, if we implement a network where we can freely stream display contents of whichever devices to any other devices, we are accomplishing the intention of connecting visual intelligent objects by connecting their display screens. The serial automation would bring about tremendous enhancement to our daily lives.

2.1. Related Work

Numerous newly-formed products and research on the Internet of Things continue to emerge in the market, especially in the field of smart homes. In order to acquire a more sufficient understanding in this area, we discuss several existing works related to our approach in the following aspects.

2.1.1. IoT and Smart Home

A convergence of various technologies such as faster and smaller processors, commodity sensors, wireless communication, machine learning, mobile computing and pervasive computing has widened the horizon of Internet of Things. A proportionate increase of IoT devices are manufactured, including those aiming for industry and consumer use. Among consumer usage, smart home products are the one closest to our daily lives. Contemporary smart home products can be generally classified as hubs, gadgets and systems. For example, lately, the popular smart products Amazon Echo and Google Home are those for the smart home hubs. These devices often support a large scale of communication standards, such as Bluetooth, Wi-Fi, NFC, WEBee and even USB ports for downward compatibility, serving as the kernel of the smart home environment. On the other hand, smart home gadgets such as smart light bulbs, smart thermostats, smart refrigerators and smart door locks are progressively developed to supply our daily demands by automating daily housework, either coordinating it or running it independently. In general, usual smart environments are composed of a centralized hub controller and smart objects, merging various functionalities into a unified operation pattern. However, it could be laborious to purchase and maintain an extra device serving as a control center. In addition, the lack of standardization interactions and normalization technologies might lead to unsustainable issues for those automations.

2.1.2. Interaction in Smart Homes

In this section, we discuss several popular ways of smart home interaction. Firstly, voice is one of the prevalent interactive modes among smart devices and appliances in smart homes. For example, say something like "turn on the light in the kitchen" to Alexa of Amazon Echo for turning on the smart bulb Philips (Amsterdam, the Netherlands) Hue [11] installed in the kitchen. Alternatively, portable devices like cell-phones or PDAs with a specialized mobile application provided by the manufacturer can be a data or control center of the smart home. For example, iOS HomeKit [12] application is used to communicate with all connected accessories and to converge them all at once. Additionally, we can manipulate smart objects with a wearable device such as a smart watch or smart bracelet. For example, Apple (Cupertino, CA, USA) Watch [13] can be used to control smart devices and appliances by a HomeKit application on watchOS.

In our approach, we keep the essence and assemble the above interaction modes. Particularly, our approach makes it possible to arbitrarily control any smart objects by any networked screens, extending unidirectional manipulation to a bidirectional one. Furthermore, if the middleware is integrated with a home assistant or the device has a microphone, operations can be automatically completed after receiving certain commands by any means. The home assistant here refers to smart home hubs like Apple HomePod [14] and Google Home [15]. For example, saying "display my phone here" tells the system to stream the smart phone to the most suitable device around you, saving effort and the hesitation when manually electing the from and to display. In this way, people do not have to decide and make the trade-off decision among devices.

2.1.3. VNC and Related Software

Topics of remote monitoring and control (M&C) have been researched for years [6,16–18]. Current M&C systems are multifunctional and omnipotent, but mostly are running locally and support merely one-way streaming. On the contrary, our approach supports remote monitoring and control functions among a pool of multiple categories of display screens. The following paragraphs further interpret specific distinctions.

Firstly, we introduce airplay which is a proprietary protocol suite developed by Apple Inc. for wirelessly mirroring display to certain monitors, currently known as Screen Mirroring [19]. It has numerous derivatives. Mirroring360 is a typical one [20], which is an airplay audio-visual sender, receiver and render middleware for iOS, Android, Chromebook, Windows and macOS. Generally, it is based on RTSP network control protocol [21] and it utilizes UDP (User Datagram Protocol) for audio streaming. In brief, airplay is a remote monitoring protocol without control function while our approach supports them both.

Secondly, multiple research studies concentrate on remote control of home appliances, such as [17,18], etc. They mostly focus on controlling functionalities and no mirroring features. A central application for manipulating home appliances are usually adopted by these implementations. Appliances would be separated and secluded once the central application crashed or non-existent. In our approach, remotely visual controlling and videos streaming are generally supported. In addition, the system is distributed and exists in all devices. Whichever one of the devices log off from the network, the rest of them are continuously functional.

Lastly, remote desktop sharing software based on VNC is widely dispersed, such as Dameware [22], Teamviewer [23] and RealVNC [24], etc. The emergence of Remote Desktop greatly facilitates our demand for using personal computers remotely. The mode of point-to-point remote monitoring and control allows us to watch and control the orientated device on the viewer device. However, it is constrained by scope of application, which is mostly computers and smart phones. In our approach, the networked devices construct a service pool and any two of them in the pool can monitor and control each other optionally, which gives much more spontaneous and permissive experience. The set of networked devices includes computer, portable device, smart wearable, home appliance and any other visually supported devices.

2.2. Comparison with Related Work

Last but not least, multiple remote streaming and control solutions emerged in the past few years, such as smart home hubs, gadgets, screen mirroring and VNC based software. Smart home hubs and gadgets primarily support remote control and screen mirroring products are mainly focused on remote monitoring. The VNC product is the current prevalent way to enable remote monitoring and control but is offering point-to-point streaming only. For example, one can't choose to stream A device to B device from a C device and no device discovery or streaming network support. The ability of arbitrarily doing remote monitoring and control among the networked devices is what we call Arbitrary M&C in this paper. Our approach synthesizes and enhances the functionalities of current remote control solutions, making the connection over the display screen practical and the Arbitrary M&C possible. Primarily, the middleware supports remote monitoring and control, connection and arbitrary control among multiple visual devices and is characterized by its compatibility, modularity and scalability.

In order to construct a visualized impression on the differences between the related works and our approach, we created the comparison table (Table 1). The comparison table indicates that the most obvious difference of our approach compared with other related works is the feature of Arbitrary M&C

and Cross-platform, while Remote Monitoring and Control are fully or partially supported by most related works. In terms of the feature of Arbitrary M&C, we can arbitrarily control one or multiple devices by any device within the same network with our approach. In addition, it is realistic for the approach to be integrated into or directly run on current existing systems because the system is a cross-platform and compatible for these systems. However, the compatibility of input and output methods between source and destination machine is not good enough. For example, when streaming a Windows desktop to a smart phone, it is hard to manipulate a desktop with a touch screen. Regarding the electronic display of the smart home products, like the display screen of a smart refrigerator, the situation could be worse. Nevertheless, this could be optimizable because the streaming module of the system can easily be substituted by other remote control method, for example, voice or gesture control framework, or we can even plug in a mouse or keyboard for a desktop stream. This feature has not yet been developed in the implementation, but it is fully supported by the current infrastructure.

Table 1. Comparison between related works and our approach.

	Hubs [1]	**Gadgets** [2]	**Screen Mirroring** [3]	**VNC Products** [4]	**Our Approach** [5]
Remote Monitoring	No	Depends	Yes	Yes	Yes
Remote Control	Yes	Depends	No	Yes	Yes
Remote Streaming	No	No	No	Yes	Yes
Arbitrary M&C [6]	No	No	No	No	Yes
Extensions/Apps	Yes	Depends	Yes	Depends	Yes
Cross-platform	No	Depends	No	Depends	Yes
Compatibility [7]	Good	Depends	Good	Moderate	Moderate
Scalability [8]	Good	Depends	Depends	Depends	Good

[1] Smart home hubs like Amazon Echo and Google Home. [2] Smart home gadgets like thermostats, door lock, refrigerators. [3] Apple Screen Mirroring and derivatives like Mirroring360. [4] VNC and related software like Dameware, Teamviewer and RealVNC. [5] The system we propose (a software and scaffold framework supporting to be integrated into other platforms). [6] To arbitrarily control any smart objects by any networked screens. [7] The compatibility (between input and output methods) among smart products when doing remote control. [8] The scalability of the system to different environments and future possibilities.

Moreover, the characteristics of independent streaming modules make the system scalable and extendable. We can easily integrate any other input and output methods to the system other than display content. For example, we can stream the audio content from a computer to a smart sound box or from a radio to a smartphone.

3. Design and Implementation

In order to effectuate a spontaneous performance, we ought to support as many operating systems as possible for the end users [25], ranging from system of computer, smart phone, wearable, intelligent household appliance to system of outdoor display like commercial electronic screen advertising. Instead of developing a specific application for every system, we adopt web technology as the main scheme because of its cross-platform peculiarity. Web technology is also withal developer-friendly when constructing an extension platform because developers will share the benefit of implementing one plugin for all operating systems. While cultivating and maintaining an easy to use, high quality system for end users, we are committed to creating a developer-friendly middleware for visual remote monitoring and control. Participation of developers can move business forward.

Basically, we constructed a scaffold framework for developers or manufacturers to integrate the system as a middleware, and also we developed a software program with user interface layered on this framework, which supports simple installation and direct execution on different platforms. In this section, we present how these features come into being: system architecture and implementation.

3.1. System Architecture

Smart home appliances commonly tend to use Unix-like/Linux or Android as the operating system understructure [17,26] and smart devices like smart phone, PDA, smart watch and laptop are generally layered on Windows, macOS, Unix-like/Linux, Android and iOS. Building native applications for every operating system is laborious and unstainable. Instead, we decide to design a web-based skeleton, which is natively cross-platform and also lowers the integration cost [27]. Node.js [28], a JavaScript runtime built on Chrome's V8 JavaScript engine which is Google's open source high-performance JavaScript engine, uses an event-driven, non-blocking I/O model that makes it lightweight and efficient [29]. Moreover, the V8 engine is written in C++, which is able to run standalone or is embedded into any C++ application. Thus, theoretically Node.js can be deployed in any C++ friendly operating systems. Since all operating systems mentioned above support C++, Node.js would be a decent choice for the implementation. Above all things, Node.js is very effective and flexible in developing RESTful (Representational State Transfer) APIs, which compose a great portion of the middleware system. With the asynchronous Node.js server, we adopt MongoDB for local database groundwork [30]. Web server is the intermediate between user interface and streaming server, which in our case is the VNC server. The database is local storage for information like history devices, device status and user preference, etc. User interface is provided by the front-end of the web application.

As shown in Figure 1, we adopt UPnP (Universal Plug and Play) [31] to discover networked devices, specifically SSDP (Simple Service Discovery Protocol) [31] with a USN (Unique Service Name) scheme. The UPnP network develops a de-centralized environment that displaces the way that utilize an extra device, probably hubs like Amazon Echo or Google Home, as the services moderator [32]. The conjunction of distributed middleware allows connected devices to work together as a group that can even survive if some of its members fail. This tolerance of failure is a big advantage of distributed middleware programs, which have a kind of redundancy built in.

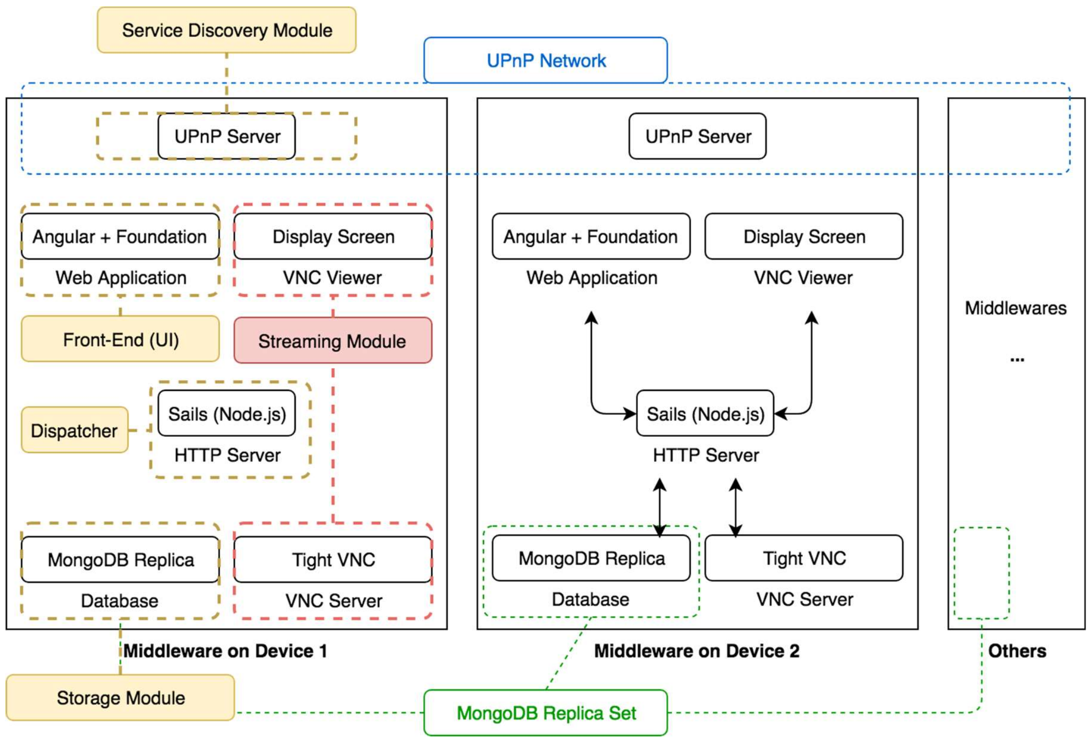
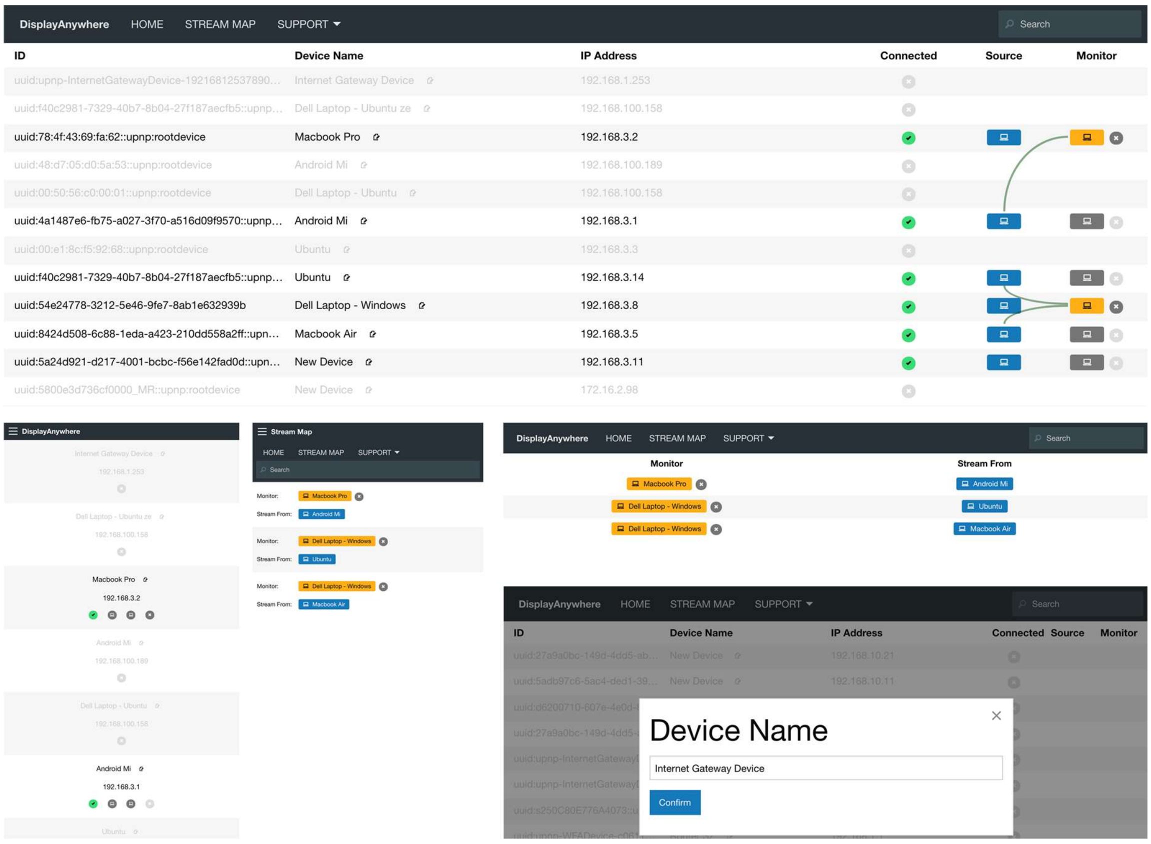


Figure 1. System architecture.

With the event-driven attribute of Node.js, we can handily notify users through the user interface whenever a UPnP client discovers a new device and emits an event named response to the HTTP server. Moreover, as shown in Figure 1, we employ a MongoDB Replica Set to create distributed storage that is redundant and highly available. Replica Set in MongoDB is a group of processes that maintain the same data set among different hosts. Data of device information, device status, user preference and other manifestations are stored by a Replica Set through the middleware on all devices. Every device has a MongoDB instance as a member of the Replica Set and every instance maintains the same data collections automatically. It is a replication and synchronization strategy for local storage of the distributed MongoDB instances, aiming to keep the data consistent and usable anytime anywhere.

3.2. Prototype Implementation

To prove it feasible, we have implemented a prototype on top of the architecture. As shown in Figure 1, the middleware consists of five modules: Service Discovery Module, Front-End Module (User Interface), Dispatcher (Control Center), Storage Module and Streaming Module. We developed the anterior four modules (yellow line boundaries) for the middleware and integrated VNC as the streaming module (red line boundaries), which can also be substituted by any other remote control methods. The Service Discovery Module is on the top of Figure 1, a UPnP server over the USN scheme to automatically detect and discover new and existing devices. To view and play the networked devices, we constructed a web-based application as the Front-End Module (Figure 2), which shows the list of supported devices and specific function entrances of the system. The central part of the middleware is the Dispatcher, which is an HTTP server implemented by Sails [33], a Node.js framework that is Ruby on Rails [34] like, data-driven and service-oriented, layered on Express. The Dispatcher manages the Streaming Module, controlling streaming source and destination execution and affiliated functionalities. The Streaming Module is implemented as a VNC server and viewer in this application. Lastly, the Storage Module consists of a MongoDB locally and constructs a Replica Set among the network. The MongoDB stores information of history devices and the affiliated data while the Replica Set periodically synchronizes the data among the distributed devices.



ID	Device Name	IP Address	Connected	Source	Monitor
uuid:upnp-InternetGatewayDevice-19216812537890…	Internet Gateway Device	192.168.1.253	✗		
uuid:f40c2981-7329-40b7-8b04-27f187aecfb5::upnp…	Dell Laptop - Ubuntu ze	192.168.100.158	✗		
uuid:78:4f:43:69:fa:62::upnp:rootdevice	Macbook Pro	192.168.3.2	✓	🟦	🟧 ✗
uuid:48:d7:05:d0:5a:53::upnp:rootdevice	Android Mi	192.168.100.189	✗		
uuid:00:50:56:c0:00:01::upnp:rootdevice	Dell Laptop - Ubuntu	192.168.100.158	✗		
uuid:4a1487e6-fb75-a027-3f70-a516d09f9570::upnp…	Android Mi	192.168.3.1	✓	🟦	⬛
uuid:00:e1:8c:f5:92:68::upnp:rootdevice	Ubuntu	192.168.3.3	✗		
uuid:f40c2981-7329-40b7-8b04-27f187aecfb5::upnp…	Ubuntu	192.168.3.14	✓	🟦	⬛
uuid:54e24778-3212-5e46-9fe7-8ab1e632939b	Dell Laptop - Windows	192.168.3.8	✓	🟦	🟧 ✗
uuid:8424d508-6c88-1eda-a423-210dd558a2ff::upn…	Macbook Air	192.168.3.5	✓	🟦	⬛
uuid:5a24d921-d217-4001-bcbc-f56e142fad0d::upn…	New Device	192.168.3.11	✓	🟦	⬛
uuid:5800e3d736cf0000_MR::upnp:rootdevice	New Device	172.16.2.98	✗		

DisplayAnywhere

Internet Gateway Device
192.168.1.253

Dell Laptop - Ubuntu ze
192.168.100.158

Macbook Pro
192.168.3.2

Android Mi
192.168.100.189

Dell Laptop - Ubuntu
192.168.100.158

Android Mi
192.168.3.1

Ubuntu

Stream Map
HOME STREAM MAP SUPPORT
Search

Monitor: Macbook Pro
Stream From: Android Mi

Monitor: Dell Laptop - Windows
Stream From: Ubuntu

Monitor: Dell Laptop - Windows
Stream From: Macbook Air

DisplayAnywhere HOME STREAM MAP SUPPORT Search

Monitor	Stream From
Macbook Pro	Android Mi
Dell Laptop - Windows	Ubuntu
Dell Laptop - Windows	Macbook Air

DisplayAnywhere HOME STREAM MAP SUPPORT Search

ID	Device Name	IP Address	Connected	Source	Monitor
uuid:27a9a0bc-149d-4dd5-ab…	New Device	192.168.10.21	✗		
uuid:5adb97c6-5ac4-ded1-39…	New Device	192.168.10.11	✗		
uuid:d6200710-607e-4e0d-					
uuid:27a9a0bc-149d-4dd5-					
uuid:upnp-InternetGateway					
uuid:upnp-InternetGateway					
uuid:s250C80E776A4073::u					
uuid:upnp-WFADevice-c061					

Device Name
Internet Gateway Device
Confirm

Figure 2. User interface of the implementation

The following paragraphs interpret the system modules in detail. For the Service Discovery Module, we implement a UPnP server over a USN scheme to establish an appointed network.

The USN of every device is mixed with *upnp:rootdevice* and a MAC Address (Media Access Control Address). The prototype searches *upnp:rootdevice* for UPnP clients as a background process of the HTTP (HyperText Transfer Protocol) server. Specifically, we use *node-ssdp*, which is an extension package of Node.js from NPM (Node Package Manager) [35], for implementing a UPnP server to broadcast the existence of a certain device. In this way, we are capable of obtaining information of networked devices and display them as a list by user interface of the web application, the Front-End Module. The user interface is implemented by Angular which is an MVC (Model–View–Controller) framework for web development powered by Google, Inc. (Mountain View, CA, USA) [36] and a responsive front-end framework called Foundation [37] powered by ZURB, Inc. (Campbell, CA, USA) [37]. Preferably, both are MIT licensed.

For the Streaming Module, we adopt VNC as the streaming method for remote screen sharing and use Tight VNC [38] for VNC implementation. Server and viewer of Tight VNC are both integrated in the prototype. VNC server listens on port 5921 (port of computer networking), a constant port agreed by all devices. The port number is particularly picked up for avoiding the default port number 5900 of VNC server, in case it is already been occupied by other service. In terms of VNC viewer, the display container, we have several implement options: opening a separated window, displaying directly on web browsers with the web-based VNC viewer noVNC and WebSockify [39]. Provided that the middleware is granted with sufficient low-level permissions by the operating system, feeding full-screen with the streaming graphics can be realized effortlessly [40].

The module of Tight VNC are manipulated by the Dispatcher, a Node.js server in response to the operations of end users. When users select a source and a terminal display, the middleware on the source deivec requests the terminal device by a RESTful API (A RESTful API is an application program interface (API) that uses HTTP requests to GET, PUT, POST and DELETE data.), which in this case is:

- *http://${dest_address}:1337/home/execVncViewer?src_address=${src_address}.*

${dest_address}* would be replaced by an IP address of the terminal device and *${src_address} is the IP address of the source device. Afterwards, the middleware on the terminal device launches a VNC viewer and connects to the VNC server of the source device by port 5921. Ideally, the process of the VNC viewer would become cutting and replacing the monitor's output stream with a display stream of the source device, full-screen on the terminal device. In addition, alternative input interactions such as voice, gesture, motion and other control methods can be easily integrated, merely modifying the module of the Node.js server.

The framework part of the implementation is a module by a module scaffold for developers and manufacturers to layer on and build various applications. The software part of the implementation can be pre-installed in upcoming products or manually installed by users. Either way, the middleware is set up in every smart object, serving as roles of both server and client among the services pool created by the networked middleware.

3.3. Deployment

In our approach, we strictly follow a deployment order for every middleware installation to assure a robust and dependable environment. We created an installer named *install.sh*, a shell script responsible for deploying all middleware requisites beforehand, adhering to rigorous order. The installer can also be implemented as a batch script or Windows Installer, APK (Android application package) [41] and so forth.

Firstly, the installer starts a UPnP server by running *ssdpserver.js*, which is a *node-ssdp* NPM application. Secondly, it searches for devices alternatively by *gssdp-discover* or other UPnP client implementations. In this way, the installer is aware of the amount and identification of the networked devices. Subsequently, it connects to MongoDB Replica Set and joins the Replica Set members after booting its own MongoDB instance. Otherwise, if the Replica Set does not exist, the installer directly starts MongoDB and initializes the Replica Set.

Afterwards, the installer launches the Tight VNC server and starts the Sails server. Lastly, it checks again on the devices list created by the UPnP client. If new devices show up during the deployment processes, they would be grouped up to the system by this step. By far, the server begins to listen to the *response* events triggered by the UPnP client. Whenever there's a new broadcasting device reaching the network, it would be added to the system promptly.

4. System Performance Evaluation

In this section, we present a performance analysis of the middleware system. Since performance of a computer system is multidimensional, we evaluate the system in several dimensions: Latency or Response Time, Bandwidth or Throughput, Usability and Scalability. We have taken three operating systems which are Ubuntu 16.04 (Dell (Round Rock, TX, USA) Inspiron, 2.5 GHz Intel (Santa Clara, CA, USA) Core i5-7200U, Memory 7.7 GB), macOS Sierra 10.12.6 (MacBook Pro, 15-inch, 2016, 2.7 GHz Intel Core i7, Memory 16 GB 2133 MHz LPDDR3) and Android 6.0.1 (Red Mi 3S, 1.4 GHz Octa-core Max, Memory 2 GB) as evaluating platforms and analyzed the performance of the middleware on these platforms. For tracking key events of user actions and system executions, we have utilized the logging feature of Sails, which is based on Winston, a Node.js logger package [42], and allows for custom transports and adapters for log messages. This evaluation aims to analyze the performance of the middleware proposed in this paper only, so the performance of a certain VNC-based library will not be covered in this section. On the other hand, since the power consumption of the middleware system is very small compared to the power consumption of display screens and what our system orients are smart objects with display screens, we would choose to skip power consumption in this section.

4.1. Response Time

Theoretically, response times should be as fast as possible, but how fast is fast enough? User perception of latency could be classified into three limits [43]: (1) about 0.1 s to have the user feel that the system is reacting instantaneously; (2) about 1.0 s to have the user's flow of thought stay uninterrupted; and (3) about 10 s limit to keep the user's attention focused on the context. We installed the middleware into the three evaluating platforms, tested the streaming feature 20 times for each two operating systems mutually and calculated the average response time from user actions to request received, and then to the operations actually happened—for example, 'VNC viewer opened' or 'VNC viewer closed'.

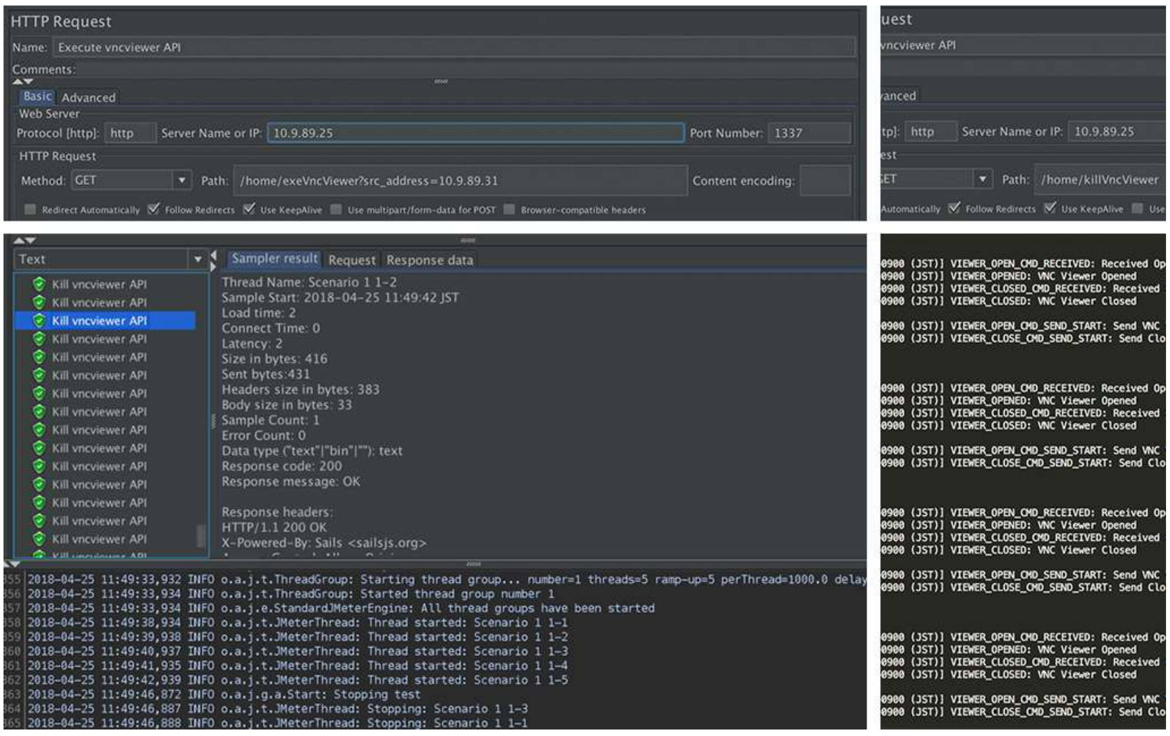
After that, we implement a Node.js script to retrieve the system logs (Figure 3) and calculate the average response times of certain action nodes from the timestamps of the events that the system recorded. The result is given by the table Average Response Time (Table 2). As shown in the table, it takes less than 0.3 s for the middleware to respond to the user actions in most operating systems except when streaming to the Android phone. In other words, the average response times of the middleware satisfy the second user perception limit and are very close to the first limit. The result illustrates that the middleware is able to keep the user's flow of thought stay uninterrupted and could almost have the user feel that system is reacting instantaneously, surely except for the Android phone.

Table 2. Average response time (Unit: millisecond).

Time Points	A1 to A2 [1]	A2 to A3 [1]	A1 to A4 [1]	A4 to A5 [1]
Ubuntu —> Mac	95.1	8.2	83.3	9.3
Mac —> Ubuntu	234.2	9.1	231.1	9.8
Mac —> Android	1294.4	18.2	1308.7	16.5
Android —> Mac	108.2	7.9	94.6	9.7
Ubuntu —> Android	1472.1	17.9	1389.6	16.9
Android —> Ubuntu	268.9	9.3	249.4	10.1

[1] A1: User Action; A2: Request of Opening Viewer Received; A3: Viewer Opened; A4: Request of Closing Viewer Received; A5: Viewer Closed.

With regard to the Android phone, the average response times are slightly greater than the second user perception limit (1 s) and satisfy the third limit. We can conclude from the table that the performance of the middleware mostly depends on the computing power of the operating system of the streaming destination. Therefore, if we choose an Android system with a better configuration, the performance would be significantly improved. With the development of hardware and mobility technologies, an Android system with better configuration should be easily reachable.



```
HTTP Request
Name: Execute vncviewer API
Comments:
Basic  Advanced
Web Server
Protocol [http]: http   Server Name or IP: 10.9.89.25   Port Number: 1337
HTTP Request
Method: GET   Path: /home/exeVncViewer?src_address=10.9.89.31   Content encoding:
☐ Redirect Automatically  ☑ Follow Redirects  ☑ Use KeepAlive  ☐ Use multipart/form-data for POST  ☐ Browser-compatible headers
```

```
Text
Kill vncviewer API
Kill vncviewer API
Kill vncviewer API
Kill vncviewer API
Kill vncviewer API
Kill vncviewer API
Kill vncviewer API
Kill vncviewer API
Kill vncviewer API
Kill vncviewer API
Kill vncviewer API
Kill vncviewer API
Kill vncviewer API
Kill vncviewer API
Kill vncviewer API
```

```
Sampler result  Request  Response data
Thread Name: Scenario 1 1-2
Sample Start: 2018-04-25 11:49:42 JST
Load time: 2
Connect Time: 0
Latency: 2
Size in bytes: 416
Sent bytes:431
Headers size in bytes: 383
Body size in bytes: 33
Sample Count: 1
Error Count: 0
Data type ("text"|"bin"|""): text
Response code: 200
Response message: OK

Response headers:
HTTP/1.1 200 OK
X-Powered-By: Sails <sailsjs.org>
```

```
2018-04-25 11:49:33,932 INFO o.a.j.t.ThreadGroup: Starting thread group... number=1 threads=5 ramp-up=5 perThread=1000.0 delay
2018-04-25 11:49:33,934 INFO o.a.j.t.ThreadGroup: Started thread group number 1
2018-04-25 11:49:33,934 INFO o.a.j.e.StandardJMeterEngine: All thread groups have been started
2018-04-25 11:49:38,934 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-1
2018-04-25 11:49:39,938 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-2
2018-04-25 11:49:40,937 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-3
2018-04-25 11:49:41,935 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-4
2018-04-25 11:49:42,939 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-5
2018-04-25 11:49:46,872 INFO o.a.j.g.a.Start: Stopping test
2018-04-25 11:49:46,887 INFO o.a.j.t.JMeterThread: Stopping: Scenario 1 1-3
2018-04-25 11:49:46,888 INFO o.a.j.t.JMeterThread: Stopping: Scenario 1 1-1
```

```
uest
vncviewer API

anced
tp]: http   Server Name or IP: 10.9.89.25
est
ET   Path: /home/killVncViewer
Automatically ☑ Follow Redirects ☑ Use KeepAlive ☐ Use
```

```
0900 (JST)] VIEWER_OPEN_CMD_RECEIVED: Received Op
0900 (JST)] VIEWER_OPENED: VNC Viewer Opened
0900 (JST)] VIEWER_CLOSED_CMD_RECEIVED: Received
0900 (JST)] VIEWER_CLOSED: VNC Viewer Closed

0900 (JST)] VIEWER_OPEN_CMD_SEND_START: Send VNC
0900 (JST)] VIEWER_CLOSE_CMD_SEND_START: Send Clo


0900 (JST)] VIEWER_OPEN_CMD_RECEIVED: Received Op
0900 (JST)] VIEWER_OPENED: VNC Viewer Opened
0900 (JST)] VIEWER_CLOSED_CMD_RECEIVED: Received
0900 (JST)] VIEWER_CLOSED: VNC Viewer Closed

0900 (JST)] VIEWER_OPEN_CMD_SEND_START: Send VNC
0900 (JST)] VIEWER_CLOSE_CMD_SEND_START: Send Clo


0900 (JST)] VIEWER_OPEN_CMD_RECEIVED: Received Op
0900 (JST)] VIEWER_OPENED: VNC Viewer Opened
0900 (JST)] VIEWER_CLOSED_CMD_RECEIVED: Received
0900 (JST)] VIEWER_CLOSED: VNC Viewer Closed

0900 (JST)] VIEWER_OPEN_CMD_SEND_START: Send VNC
0900 (JST)] VIEWER_CLOSE_CMD_SEND_START: Send Clo


0900 (JST)] VIEWER_OPEN_CMD_RECEIVED: Received Op
0900 (JST)] VIEWER_OPENED: VNC Viewer Opened
0900 (JST)] VIEWER_CLOSED_CMD_RECEIVED: Received
0900 (JST)] VIEWER_CLOSED: VNC Viewer Closed

0900 (JST)] VIEWER_OPEN_CMD_SEND_START: Send VNC
0900 (JST)] VIEWER_CLOSE_CMD_SEND_START: Send Clo
```

Figure 3. Apache jMeter test plan screenshots and a segment of system logs.

4.2. Concurrency Performance

On the other hand, the middleware is distributed and deployed in disparate connected smart devices, so the workload of the serving and computing is separated into multiple processing units among a local area network. Therefore, the bandwidth or server capacity of the system, such as the abilities of parallel processing or concurrent processing, wouldn't be a significant payload or a bottleneck of the system. The household appliances, computers and other smart devices are providing consistent and sufficient computing power. As to prove this assumption, we utilized Apache jMeter (Version 4.0, Apache Software Foundation, Forest Hill, MA, USA) [44] to continuously and concurrently send requests and parameters to the RESTful APIs of the HTTP servers, such as (Those who are commonly requested most):

- *http://${dest_address}:1337/home/pingKillVncViewer*: send the killing VNC viewer command,
- *http://${dest_address}:1337/home/getDevicesFromDB?page=${page_num}*: get device list,
- *http://${dest_address}:1337/home/execVncViewer?src_address=${src_address}*: run VNC viewer, etc.

We built a web test plan with jMeter to continuously send requests to every API and to simulate the concurrency situation, we created 100 threads for each test job of HTTP requests. With this configuration, we were emulating 100 users continuously and simultaneously using all the functions of the system (Figure 3).

While the jMeter test plan was running, we did the 20 times average response time evaluation again and the average response times that we got this time are very close to the result that we concluded above. The outcome shows that the system is able to support hundreds of requests at a time without significantly increasing the average response time as shown in Table 2. The result shows that the implementation is highly sufficient and effective to a smart home environment or even an outdoor networked environment.

4.3. Network Overhead and Payload

To gain a more comprehensive evaluation, we assessed the network performance by calculating and surveying the RESTful APIs in terms of network overhead and payload. We took the API for getting a device list as an example and captured the request and response plain text correspondingly. The tested application was running on the web browser Chrome on macOS Sierra 10.12.6:

- *http://${dest_address}:1337/homegetDevicesFromDB?page=${page_num}*.

Overhead of the HTTP request headers (561 bytes):

```
GET /home/getDevicesFromDB?page=1 HTTP/1.1
Host: localhost:1337
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.106 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9,zh-CN;q=0.8,zh;q=0.7
Cookie: sails.sid=s%3AgMVYt6STGw0fc9CTXwa6qQXADueX2Qai.vBUYOu8na1MfshcioTWmpUS0oAk5s4%2FoaFlgl%2F5qz%2Bs; io=R_6aWzX60d6wpXJ4AAAC
```

Overhead of the HTTP response headers (394 bytes): *HTTP/1.1 200 OK*

```
X-Powered-By: Sails <sailsjs.org>
Access-Control-Allow-Origin:
Access-Control-Allow-Credentials:
Access-Control-Allow-Methods:
Access-Control-Allow-Headers:
Access-Control-Expose-Headers:
Content-Type: application/json; charset=utf-8
Content-Length: 373
Content-Encoding: gzip
Etag: W/"29eb-BrL2sbwO9YyXHUjf0kJmEw"
Date: Sun, 19 Aug 2018 10:25:02 GMT
Connection: keep-alive
```

Payload of the response (751 bytes, which depends on the history devices' amount):

```
[
 {
 "address": "172.16.2.74",
 "family": "IPv4",
 "port": 1900,
 "usn": "uuid:78:4f:43:69:fa:62::upnp:rootdevice",
 "server": "node.js/6.9.5 UPnP/1.1 node-ssdp/3.2.5",
 "name": "Macbook Pro",
 "receive_from": "uuid:78:4f:43:69:fa:62::upnp:rootdevice",
 "createAt": "2018-02-27T09:52:58.362Z",
 "updateAt": "2018-08-17T06:58:48.901Z"
},
{
"address": "192.168.100.158",
"family": "IPv4",
"port": 1900,
"usn": "uuid:00:50:56:c0:00:01::upnp:rootdevice",
"server": "node.js/4.2.6 UPnP/1.1 node-ssdp/3.2.5",
"name": "Dell Laptop - Ubuntu",
"receive_from": "uuid:78:4f:43:69:fa:62::upnp:rootdevice",
"createAt": "2018-03-09T02:51:10.518Z",
"updateAt": "2018-07-09T03:46:20.112Z"
}
]
```

There is very little we can do to reform the data transfer speed because it is usually limited by the network operators. What we can do for optimizing the network performance is minimizing the network overhead and payload. We enabled HTTP compression [45] and removed verbose fields of response content, aiming to condense the data to our actual needs only. We also added some abbreviated API endpoint for the abbreviated payload to optimize the performance of some requests. The key for designing a networking API is about providing only the actual needs of the clients now and in the future. According to the system prerequisite, the network payload data size listed above and the response time we tested before, we consider the network overhead and payload is reasonable and acceptable for building up the system. Moreover, in order to pursue better RESTful performance and reduce the payload size further, we will adopt Lightning Platform REST API [45] characteristics and criteria for the future direction, such as stateless caching behavior, conditional requests and so on.

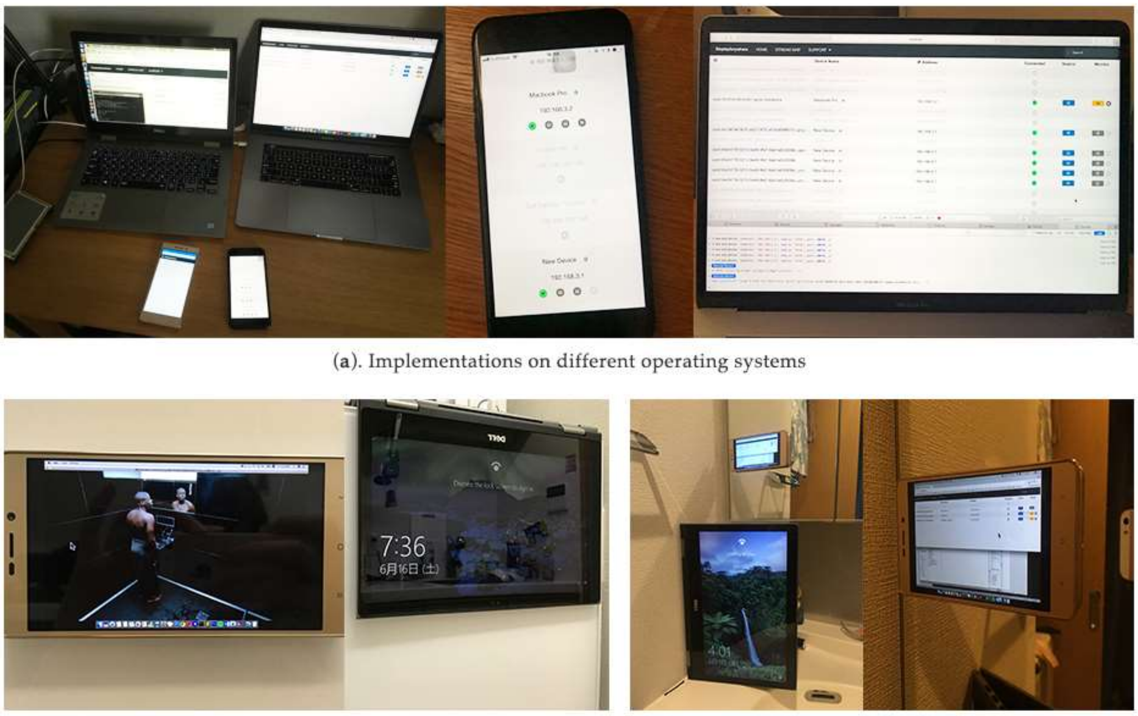
4.4. Scalability

With respect to the system usability and scalability, the characteristics such as modularity, simplicity and single threaded asynchrony of Node.js make the system architecture highly scalable and flexible. With the architecture, we can easily extend the system features by adding new controllers, models, policies or services to Sails. In addition, the architecture let the middleware independent from the streaming modules. We can effortlessly integrate any other input and output methods to the system other than display content. For example, with exactly the same system architecture, we can stream the audio content from a computer to a smart sound box or from a radio to a smartphone. Every time we plan to add a new input or output method to the network, all we have to do is to merely substitute the current VNC part with the new protocols of the certain input or output method.

5. Application Scenarios

We believe that electronic visual display will evolve to target specific needs, whether those of a particular use case, of technical implementation possibilities, or of the regulatory environment. To experiment the mechanism and explore its possibilities, we address several general scenarios concerning user experience, usability, as well as human factors and ergonomics. We have implemented the middleware for multiple operating systems: Windows, macOS, Ubuntu and Android (Figure 4a). End users are able to operate the source and terminal relationship by the web-based user interface (Figure 2). With this set of implementations, we simulated a simple smart screens network environment. We will demonstrate the system in the following paragraphs.

Considering that we can barely find or too costly to get a smart household appliance which supports to install an application from a third-party developer, we attached an Android phone to a refrigerator (Figure 4b) to emulate a smart refrigerator with an electronic visual display. Unix-like/Linux and Android are the most popular embedded systems for constructing a smart household device or appliance, so we believe that attaching an Android device is representative for the demonstration. In addition, we did the same simulation to a "smart hot tub controller". Next, we introduce some demonstrated application scenarios.



(**a**). Implementations on different operating systems

(**b**). "smart refrigerator" situmulated by an Android phone/Tablet

(**c**). A tablet with facial recognition in lavatory

Figure 4. Application scenarios.

Scenario 1: Janine and Zebo were watching a TV show together at night. Janine decided to take a bath but she didn't want to miss the show, so she chose to project the TV show to the hot tub controller that has a touchable display screen. In this way, she enjoyed the show at the same time taking a hot tub bath. At this moment, multiple messages' notifications rang out from Janine phone which is charging in the bedroom. Janine would like to check the messages because they kept coming, so Janine selected the phone as a display source and the hot tub controller as terminal. Janine checked and texted back the messages. Then, Janine wanted to share the messages to Zebo. She selected the television, the one that Zebo was watching, as a display terminal and her phone, nevertheless, as a display source. Zebo read the messages, edited and sent the reply content, which Janine has already typed in, by the remote controller of the television.

Scenario 2: Janine was busy doing her job on the laptop while she had a problem that she couldn't figure out. She wanted to ask Zebo for help so she called out the application and chose to stream her screen to Zebo's laptop with which Zebo was reading articles online. Because Janine is not the owner of Zebo's laptop, Janine's screen content didn't show up directly to Zebo's laptop. Instead, Zebo's laptop received a notification. When Zebo agreed the streaming request, he was able to control Janine's laptop and solved her problem with his own laptop.

Scenario 3: Janine and Zebo planned to cook themselves a simple tea food and accordingly found a "steamed water eggs" recipe by laptop. They don't want to bring the laptop or cell-phone to the kitchen, so they project the display of the laptop to "the screen of the smart refrigerator" and followed the tutorial step by step to finish the cooking. During the cooking, they turned on the music application Spotify in the laptop remotely by "the screen of the smart refrigerator".

Scenario 4: Furthermore, we do believe a smart mirror would be an opportunely decent streaming destination when brushing teeth or dressing up, so we place a Windows PDA with facial recognition function alongside the mirror in lavatory (Figure 4c). Whenever a privileged user faces the mirror, the display unlocks itself automatically. The PDA has an integrated audio input, so we utilize SAPI (Microsoft Speech API) to receive audio commands from users, constructing a spontaneous atmosphere. Specifically, we projected the phone elsewhere to this demonstrating "mirror display" PDA by merely saying "Show my phone here" when facing to the mirror. This hand-free interaction would be greatly effective especially when we are busy smartening ourselves in the lavatory, and we can easily integrate and implement different interaction and functionalities like so in different scenarios with the help of the scaffold infrastructure of the middleware.

6. Initial User Study

To achieve an elementary investigation and evaluate our approach, we designed an experiment that measures users' assessment on different metrics of our system. We implemented questionnaires and interviews after the experiment to conclude generic usability and experience. Our primary objective was to discuss and evaluate the need for people to use display screen arbitrarily and remotely in IoT environments, and determine whether the performance of the middleware is agreeable and enjoyable to users.

6.1. Participants and Procedure

We hired 12 people who are college students or company employees (Age m = 24.90, SD = 2.09, seven males, five females) to work on the experiment. We built up the demonstrated smart home environment mentioned in Section 5 and brought the participant to the room one by one. Basically, we had a "smart refrigerator", a smart TV, smart phones, a hand-made "smart mirror" in the lavatory and a "smart water heating system" in the bathroom, all of which were pre-installed the middleware and supported arbitrarily remote streaming.

Each participant was first introduced to the middleware system installed in the functional devices in the smart home environment and took a profiling questionnaire. Next, we carried out the experience process (Figure 5). Each participant was asked to experience the remote streaming among the smart visual devices in the room. We gave a playbook with the scenario (as described in Section 5) to the participants and they could choose whether to follow the plot or not.

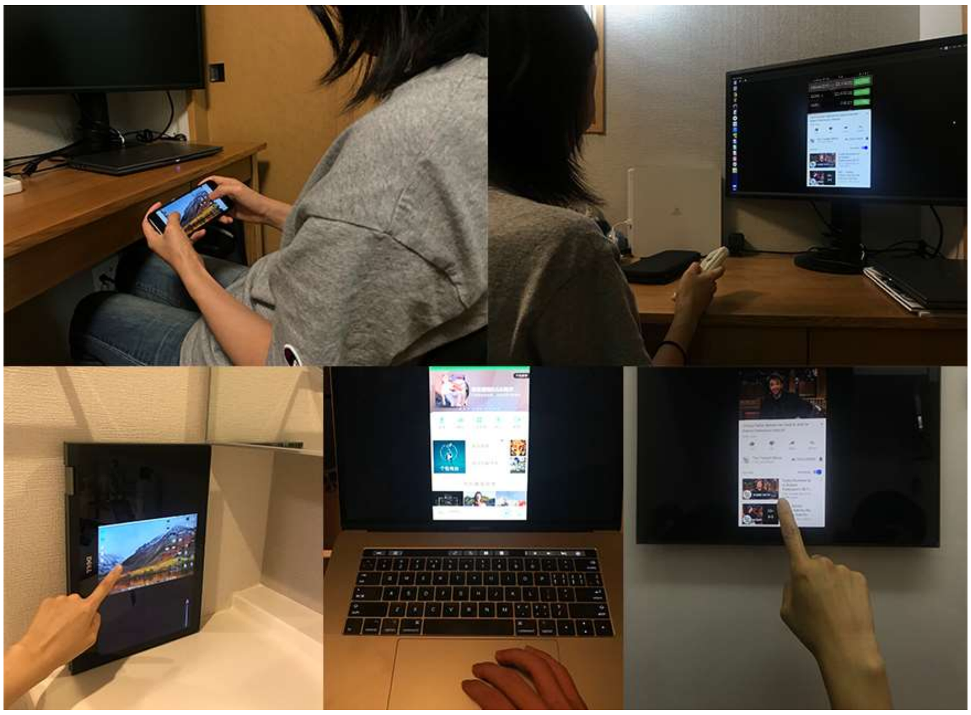


Figure 5. A participant having the experiment.

6.2. Results and Feedback

We used a 5-point Likert scale (1 is Disagree/Bad, 5 is Agree/Good) and designed twelve rating questions. Each participant was asked to filled the questionnaire after the experience process. On average, a single participant spent 12.6 minutes on the experience process and about 15 minutes on the questionnaire and the interview.

Based on the questionnaires (As shown in Table 3) and interviews with the participants, people voted highest acceptability for the concept of connecting smart objects by display screens (Q1), and they gave medium scores for the basic performance of the system (Q3). People generally consider that they are willing to purchase and pay for such products if the user experience were developed better and optimized to production level (Q4). People were generally skeptical about the current demonstrated system (Q2) because some of the smart products were stimulated and the compatibility of different operating system has not yet been dealt with, yet some thought it could be acceptable. Basically when the system and the smart home environment is mature enough, it is optimistic to bring some changes to the market. First and foremost, the compatibility among different display screens should be concerned.

Table 3. User average satisfaction [1].

	Q1 [2]	Q2 [2]	Q3 [2]	Q4 [2]	General
Rate	4.19	3.51	3.92	3.83	3.86

[1] This questionnaire used a 5-point Likert scale (1 is Disagree/Bad, 5 is Agree/Good). [2] Q1: How do you like the concept of connecting smart objects by display screens? [2] Q2: The system worked as what you expected. [2] Q3: The system ran smoothly and effortlessly. [2] Q4: How are you willing to pay for such products layered on the system?

Our secondary evaluation gauged the users' preference of streaming to/from (source/destination) devices through the system. Since there are too many pairs among different streaming devices, we pick up the top eight streaming to/from pairs and conclude them as Table 4. As shown in Table 4, participants voted highest acceptability for the streaming from TV to Phone, Desktop to Phone and Phone to Tub Controller, and they gave medium scores for the streaming from Desktop to Tub Controller, Phone to Fridge, Phone to TV and Desktop to Fridge. Besides the devices mentioned above but included in the experiment, people generally felt moderate about streaming among these devices, yet some thought it could be potential.

Table 4. Average satisfaction among streaming devices [1].

Streaming Platforms	↓ Q1 [2]	Q2 [2]	General
TV to Phone	4.12	3.50	3.810
Desktop to Phone	4.02	3.87	3.945
Phone to Tub Controller	3.98	3.51	3.745
Desktop to Tub Controller	3.86	3.53	3.695
Phone to TV	3.83	3.59	3.710
Phone to Fridge	3.79	3.49	3.640
Desktop to Fridge	3.52	3.43	3.475
Phone to Desktop	3.35	3.62	3.485

[1] This questionnaire used a 5-point Likert scale (1 is Disagree/Bad, 5 is Agree/Good). [2] Q1: Streaming between these devices is useful or needed to you (Ordered by this column). [2] Q2: The remote streaming worked as what you expected.

In conclusion, we speculate that connecting smart objects by display screen has incremental values to smart home products. In addition, the implementation of our approach is generally accepted by users because it is feasible for the products of current market, by serving as a middleware or an independent application. We believe the acceptability of such products is dependent on the user experience, system performance and how trustworthy the system is, which demands reliable and fail-proof future implementations. Basically, people will generally accept and be willing to pay for the free-stream functionalities when the process is effortless and pleasurable.

7. Current Limitations and Challenges

The search for electronic displays that are reliable for daily demands and acceptable to regulators led developers to display-based systems. As screen materials becoming efficient and cost-effective, modalities of electronic displays diversify regularly, pushing visual experience of every kind more exquisite and intriguing. To gain a macroscopic view on the impact and possibilities of electronic displays on IoT, in this section, we discuss some generalized solutions and hypotheses relevant to the approach.

7.1. System Compatibility and Flexibility

The evolvement of visual display technologies makes multiple display modalities possible, such as display screens of magic mirrors, smart glasses, wearable devices, and VR (Virtual Reality) or HLP (Holographic Laser Projection) devices [46]. Following the rapid progress of visual display technologies, there will be a great amount of interaction and input modalities emerge correspondingly.

7.1.1. Limitations

By the diversity of HCI (Human–Computer Interaction) and the emergence of new interaction modalities, the experience of our approach is and will be constrained by input modalities when streaming one display to another. For example, when a user chooses to project the smart phone to a smart watch, he/she might not be able to fluently control the smart phone by a watch because the watch screen is not designed for such operation. It is too small for showing the whole screen of the smart phone, so he/she has to keep scrolling the display in order to finish a certain task. Furthermore, if the watch screen is not a touch screen, it would be frustrated to scroll and interact with the cell-phone with gear or other physical buttons that the watch provides.

Moreover, this limitation will be enlarged when new interaction modalities appear. For example, the popularization of AR and VR displays brings a harder dilemma to our approach because normal video formats are generally not compatible with VR displays and the convert between them is ordinarily difficult. When we find a way to perfectly convert the video formats, the input methods' inconsistency is still a problem. It is unpredictable, making it user-friendly to control a desktop, a mobile phone by a VR controller.

7.1.2. Future Challenges

Nevertheless, with the cross-platform and modular architecture, we believe that, at some point, our approach can be adaptive and compatible to the challenges of continuously emerging carriers and agencies. For example, specialized extensions can be integrated for compatibility of distinct purposes. The system architecture makes the middleware independent from the module of media content streaming (specifically, VNC viewer and server, as shown in Figure 1). The VNC module is actually a specialized extension for display streaming that can be effortlessly extended or replaced by other plugins for explicit purposes.

On the other hand, assume that we have a normal 2D video in hand and we want to stream and watch it on a VR headset with a 3D effect. Before we can properly watch it, we need to convert it to 3D SBS MP4 or 3D SBS MKV format that is generally supported by VR headsets. In order to realize this function, we build a general video format converter and integrate it into the middleware as an extension. The format converting extension will automatically convert normal 2D videos to VR videos right after the action of selecting a VR HMD (Head-Mounted Displays) as a destination display screen.

In this way, people are able to seamlessly view and play any types of display content over any kinds of display devices.

Furthermore, in order to provide more flexibility to the advanced users like developers, a software development kit should be built for triggers that respond to the presence of actions like selecting a destination display or booting up a device. With this development kit, advanced users are able to exploit any functions they need and customize the interaction and automation within the environment provided by the middleware.

7.2. Generalization for Multiple Modalities

There are many different input and output methods that exist in the world, such as physical buttons, touch screens, LED indicator light, neon lamp, hologram, VR, etc. It is foreseeable that it won't be only one universal input or output modality. Diversity of input and output modality is also needed to satisfy different demands of personalized individuals. In this section, we discuss how generalization for multiple modalities affects our approach.

7.2.1. Limitations

Even though visual displays are prevalent among intelligent machines and devices, there might still be exclusive machines or devices who don't need to have a display screen. For example, electronic display, hologram or any visual mode is suitable for multifunctional and virtual contents, but, for specialized machines or appliances such as air conditioners, microwaves and ventilators, physical controllers or simple displays like pilot lamps will be more appropriate and cost-effective. Apparently, those who don't have a display screen are not supported by our implementation and consequently are secluded from the network, without assistance from extra applications. The connection among smart objects composed by our approach is limited by display screens. Being limited by display screens when connecting smart objects is occasionally not sufficient for users. Therefore, the concept of connecting smart objects over display screen is only for the visual display devices and is a lack of operation consistence among those who don't have a display screen. In order to include these devices in the network, further extensions or new streaming modules are needed to compensate the deficiency. However, even though these devices are successfully integrated to the system, the compatibility of input and output inconsistence that we mentioned in Section 7.1 is yet another issue.

7.2.2. Future Challenges

At the same time, other than extending the middleware by adding extensions to enhance the display streaming features, the extensions module, specifically the display streaming module, can be entirely replaced by other input or output methods such as text, audio, vision, etc. For example, we can integrate voice recognition, facial recognition, body motion or hand gesture recognition to the middleware as input modalities and acoustics, vibration, lights (smart bolts for example) as output modalities. In this way, we can include those machines and devices that haven't got a display screen into our network.

The extensible architecture diversifies interaction of smart home and smart city environments over the modular middleware, pursuing a world in which we can flexibly manipulate whichever smart objects by whichever input devices. Specifically, a user might communicate with the smart objects by voice or gesture, and simultaneously receive or perceive the feedback by the most convenient and appropriate way based on current situations. Anyone of the smart objects that is reachable to the user can become a control center of all other devices, no matter what type of and what device it is.

Smart home as a demonstrated scenario: a user can speak to a refrigerator to lower the volume of the television when he/she was cooking in kitchen, reply a phone message by keyboard of the computer when he/she was playing computer games and manage the temperature of a thermostat by the remote controller of the television when he/she was sitting on the couch watching television.

7.3. Distributed Data for Personality-Based Recommendation

The network of devices created by the middleware is a self-organizing decentralized system, among which local communications establish order and coordination automatically without a central influence. With the decentralized network, the data set including user preference, device information and configuration stays with every device. The network which naturally has failure redundancy built in allows connected machines to work together as a group that can even survive if some of its members fail. When a networked device is compromised and logged off from the network, the data set is still intact and stored in the device, waiting and standing by for the next connection. Several challenges could be targeted based on the distributed data set. For example, it is possible to set up preferable configuration for users automatically when joining a new network. In addition, personal data can be employed as big data sources for data mining to predict the user preference and present appropriate personality-based recommendation for users.

Assume that when a user is cooking in the kitchen, he/she prefers to heighten the brightness of the screen of a smart refrigerator that is showing the display content of his/her smart phone because he/she wants to follow the recipe application on the phone. Based on the fact of multiple similar operations, let us assume that he/she visit a friend's house and join the phone to the new network. The middleware reads the personal data and automatically heightens the brightness of the refrigerator screen according to his/her daily habits, when he/she is planning to cook at the friend's house and chooses to project the phone to the refrigerator. Moreover, when the user connects his/her laptop to the network of a department store, the system would notify and remind the user that a new movie is now playing in the theater, based on the personal data that he/she has watched several trailers recently.

However, personal data in respect of user privacy and sensitive information should be tackled to refrain from leaking to public or provisional network owned by other users. In particular, the personal data which is dispersedly stored in the networked devices should not be promptly accessed by the newly joined devices. There should be two data collections classifying owner and guest permissions for owner and guest devices. The challenge and change of device authorization should be handled beforehand.

8. Conclusions

This paper presents a cross-platform solution and a universal middleware that particularly supports VNC-based remote monitoring and control for visual smart objects, aiming to resolve the inconvenience caused by isolated smart objects and display screens. Hanging around smart products and home appliances is laborious and irritating. We designed a general architecture and accomplished creating a feasible implementation based on the architecture. For developers or manufacturers, the scaffold framework is beneficial for propagating to the developer community and to the smart devices' markets. On the other hand, for the end users, the software, which can be downloaded from the Internet, supports simple installation and direct execution on different platforms. To verify the system, we implemented performance evaluation, from which we prove that the system is feasible and reliable for smart home environments, and user study, from which we conclude that people will generally accept and be willing to pay for the free-stream functionalities when the process is effortless and pleasurable.

This paper also foresees a future smart indoor/outdoor environment and demonstrates the ingenuity and contribution of the middleware. While connecting display screens is commonplace and collective, our middleware piles out the constraints of "one display per device" and "one operating system per device". We can use the most convenient screen to manage the most needed but unreachable devices. In the future, streaming strategies should be adopted to place reasonable and optional streaming quality to different devices, especially for lightweight hardware and monochromatic display. Moreover, we conclude limitations and challenges from the aspects of system compatibility, flexibility and generalization for multiple modalities, which shows complexity about user-friendly and comprehensible attributes.

Author Contributions: Conceptualization, Z.Y. and T.N.; Data Curation, Z.Y.; Formal Analysis, Z.Y.; Investigation, Z.Y. and T.N.; Methodology, Z.Y. and T.N.; Project Administration, Z.Y. and T.N.; Resources, Z.Y. and T.N.; Software, Z.Y.; Supervision, T.N.; Validation, T.N.; Visualization, Z.Y.; Writing—Original Draft, Z.Y.; Writing—Review and Editing, Z.Y. and T.N.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Amazon Echo. Available online: https://www.amazon.com/Amazon-Echo-Bluetooth-Speaker-with-Alexa-Black/dp/B00X4WHP5E (accessed on 31 May 2018).
2. Amazon Echo Show. Available online: https://www.amazon.com/Amazon-Echo-Show-Alexa-Enabled-Black/dp/B01J24C0TI (accessed on 31 May 2018).
3. LG InstaView ThinQ™ Black Stainless Steel Refrigerator. Available online: http://www.lg.com/us/refrigerators/lg-LNXS30996D-door-in-door (accessed on 31 May 2018).
4. Richardson, T.; Stafford-Fraser, Q.; Wood, K.R.; Hopper, A. Virtual Network Computing. *Internet Comput.* **1998**, *2*, 33–38. [CrossRef]
5. Remmert, H.P. Method and Apparatus for Wireless Outdoor Environment Communications Networks. U.S. Patent US6735450B1, 14 August 2004.
6. Stødle, D.; Bjørndalen, J.M.; Anshus, O.J. De-centralizing the VNC Model for Improved Performance on Wall-Sized, High-Resolution Tiled Displays. In Proceedings of the International Conference on Norsk informatikkonferanse, Toulouse, France, 5–7 July 2011.
7. Liu, Y.; Anshus, O.J. Improving the performance of VNC for high-resolution display walls. In Proceedings of the International Symposium on Collaborative Technologies and Systems, Baltimore, MD, USA, 18–22 May 2009; pp. 376–383.
8. Jacobsen, J.J.; Parkinson, C.; Pombo, S.A. Remote Control of Host Application Using Motion and Voice Commands. U.S. Patent US9235262B2, 12 January 2016.
9. Kim, H.-J.; Jeong, K.-H.; Kim, S.-K.; Han, T.-D. Ambient Wall: Smart Wall Display interface which can be controlled by simple gesture for smart home. In Proceedings of the SIGGRAPH Asia 2011 Sketches (SA '11), Hong Kong, China, 12–15 December 2011.
10. Rahman, A.S.M.M.; Tran, T.T.; Hossain, S.A.; Saddik, A.L. Augmented Rendering of Makeup Features in a Smart Interactive Mirror System for Decision Support in Cosmetic Products Selection. In Proceedings of the 14th International Symposium on Distributed Simulation and Real Time Applications, Fairfax, VA, USA, 17–20 October 2010.
11. Philips Hue. Available online: https://www2.meethue.com/ (accessed on 31 May 2018).
12. Apple HomeKit. Available online: https://developer.apple.com/homekit/ (accessed on 31 May 2018).
13. Apple Watch. Available online: https://www.apple.com/watch/ (accessed on 31 May 2018).
14. Apple HomePod. Available online: https://www.apple.com/homepod/ (accessed on 31 May 2018).
15. Google Home. Available online: https://store.google.com/product/google_home (accessed on 31 May 2018).
16. Nakajima, T.; Kobayashi, N.; Tokunaga, E. Middleware Supporting Various Input/Output Devices for Networked Audio and Visual Home Appliances. In Proceedings of the International Symposium on Ubiquitious Computing Systems (UCS 2004), Tokyo, Japan, 8–9 November 2004; pp. 157–173.
17. Mowad, M.; Fathy, A.; Hafez, A. Smart Home Automated Control System Using Android Application and Microcontroller. *Int. J. Sci. Eng. Res.* **2014**, *5*, 935–939.
18. Tanaka, H.; Suzuki, H.; Watanabe, A.; Naito, K. Evaluation of a secure end-to-end remote control system for smart home appliances. In Proceedings of the 2018 IEEE International Conference on Consumer Electronics (ICCE 2018), Las Vegas, NV, USA, 12–14 January 2018; pp. 1–2.
19. Apple Airplay. Available online: https://support.apple.com/en-us/HT204289 (accessed on 31 May 2018).
20. Splashtop Mirroring360. Available online: https://www.mirroring360.com/ (accessed on 31 May 2018).
21. Schulzrinne, H.; Rao, A.; Lanphier, R. *Real Time Streaming Protocol 2.0*; Internet Engineering Task Force: Fremont, CA, USA, 2016.
22. SolarWinds Dameware. Available online: https://www.dameware.com/ (accessed on 31 May 2018).

23. TeamViewer. Available online: https://www.teamviewer.com/ (accessed on 31 May 2018).
24. RealVNC. Available online: https://www.realvnc.com/ (accessed on 31 May 2018).
25. Bishop, J.; Horspool, N. Cross-Platform Development: Software that Lasts. *Computer* **2006**, *39*, 26–35. [CrossRef]
26. Bing, K.; Fu, L.; Zhuo, Y.; Yanlei, L. Design of an Internet of Things-based smart home system. In Proceedings of the 2nd International Conference on Intelligent Control and Information Processing, Harbin, China, 25–28 July 2011.
27. Raj, C.P.R.; Tolety, S.B. A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. In Proceedings of the Annual IEEE India Conference (INDICON), Kochi, India, 7–9 December 2012.
28. Joyent Node.js. Available online: https://nodejs.org/ (accessed on 31 May 2018).
29. Bangare, S.L.; Gupta, S.; Dalal, M.; Inamdar, A. Using Node.js to Build High Speed and Scalable Backend Database Server. *Int. J. Res. Adv. Technol.* **2016**, 61–64.
30. Maatouki, A.; Szuba, M.; Meyer, J.; Streit, A. A Horizontally-Scalable Multiprocessing Platform Based on Node.js. In Proceedings of the 13th International Symposium on Parallel and Distributed Processing with Applications, Helsinki, Finland, 20–22 August 2015.
31. UPnP Device Architecture 2.0. Available online: http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v2.0.pdf (accessed on 31 May 2018).
32. Song, H.; Kim, D.; Lee, K.; Sung, J. UPnP-based sensor network management architecture. In Proceedings of the Second International Conference on Mobile Computing and Ubiquitous Networking (ICMU 2005), Osaka, Japan, 13–15 April 2005.
33. Sails. Available online: https://sailsjs.com/ (accessed on 31 May 2018).
34. Rails, Ruby on Rails. Available online: https://rubyonrails.org/ (accessed on 31 May 2018).
35. NPM, Node Package Manager. Available online: https://www.npmjs.com/ (accessed on 31 May 2018).
36. Google Angular. Available online: https://angular.io/ (accessed on 31 May 2018).
37. Foundation by ZURB, Inc. Available online: https://foundation.zurb.com/ (accessed on 31 May 2018).
38. Tight VNC. Available online: https://www.tightvnc.com/ (accessed on 31 May 2018).
39. Shen, Z.; Guo, L.; Hou, B.; Quan, H.; Zhou, S. Key technique research on desktop virtualization in cloud environment. *Int. J. Model. Simul. Sci. Comput.* **2017**, *8*, 1750045. [CrossRef]
40. Markel, S.O. Displaying Full Screen Streaming Media Advertising. U.S. Patent US7661117B2, 9 February 2010.
41. Google Android Application Package. Available online: https://developer.android.com/google/play/expansion-files/ (accessed on 31 May 2018).
42. Winston, C.R. A Logging Library With Support for Multiple Transports. Available online: https://github.com/winstonjs/winston/ (accessed on 31 May 2018).
43. Nielsen, J. *Usability Engineering*; Morgan Kaufmann: San Francisco, CA, USA, 1993.
44. Apache JMeter. Available online: http://jmeter.apache.org/download_jmeter.cgi/ (accessed on 31 May 2018).
45. Salesforce.com, Inc. REST API Developer Guide. Available online: https://developer.salesforce.com/docs/atlas.en-us.api_rest.meta/api_rest/intro_what_is_rest_api.htm (accessed on 31 August 2018).
46. Wagner, D.; Pintaric, T.; Ledermann, F.; Schmalstieg, D. Towards Massively Multi-user Augmented Reality on Handheld Devices. In Proceedings of the International Conference on Pervasive Computing (Pervasive 2005), Boppard, Germany, 6–8 April 2005; pp. 208–219.

© 2018 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).