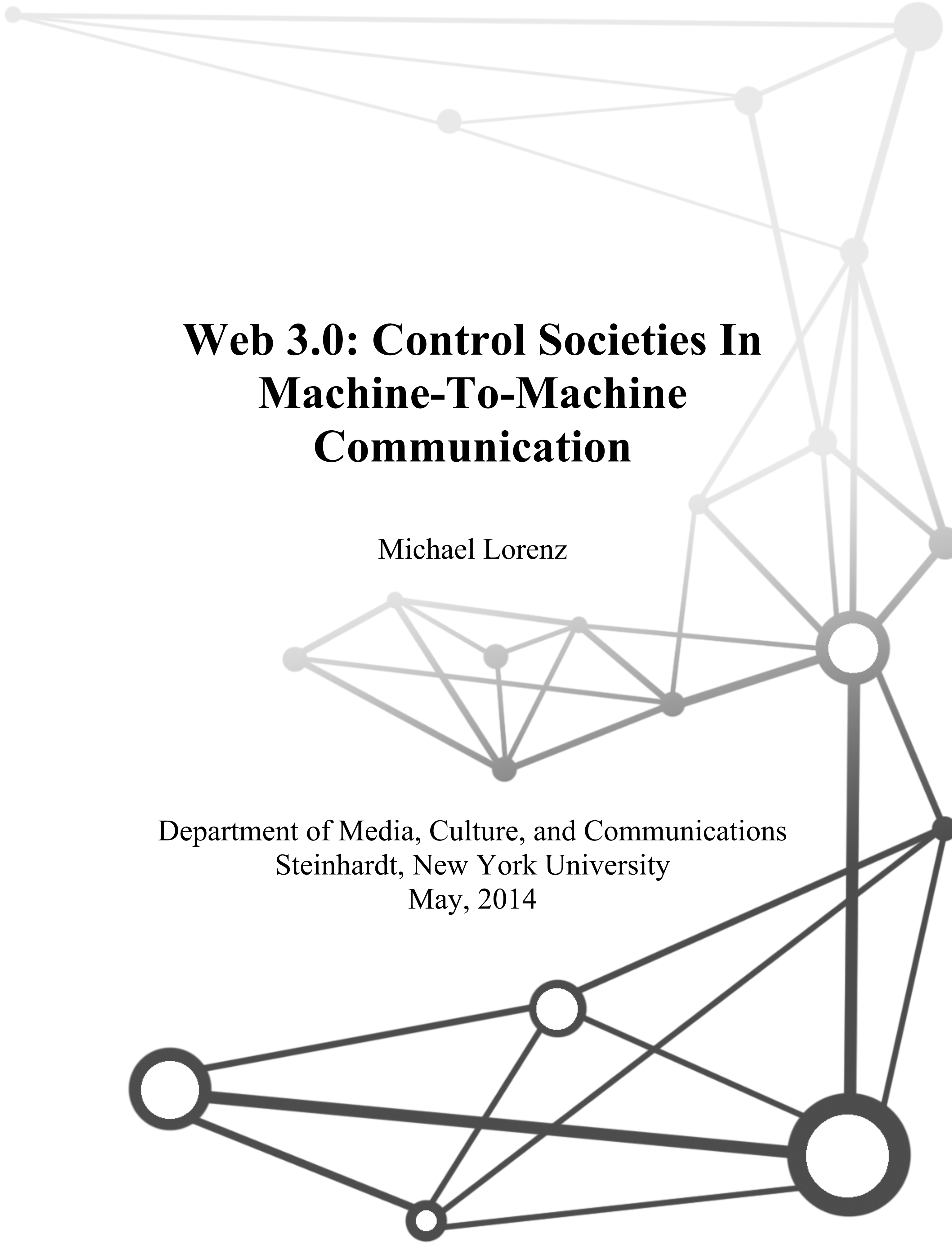


Web 3.0: Control Societies In Machine-To-Machine Communication

Michael Lorenz

Department of Media, Culture, and Communications
Steinhardt, New York University

May, 2014

Table of Contents

1. List of Acronyms and Terms

AJAX	Asynchronous Javascript and XML
C	General purpose programming language
DOM	Document Object Model
FLOSS	Free Libre Open-Source Software
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
IP	Internet Protocol
Java	General purpose programming language
JavaScript	General purpose programming language used in browsers
JSON	Javascript Object Notation
JSONP	Javascript Object Notation with Padding
jQuery	Javascript library used for manipulating the DOM
M2M	Machine-To-Machine
NFC	Near Field Communication
REST	Representational State Transfer
RFID	Radio Frequency Identification
SOAP	Simple Object Access Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Location
XML	Extended Markup Language

2. Introduction

```
public class HelloWorld {
     public static void main(String [] args) {
          System.out.println("Hello, World!");
     }
}
```

If an object were to communicate to another object in some sort of meaningful way, what exactly would it say? Would it stream out a seemingly endless series of 0's and 1's, or would it simply say the classic programming exercise of "`Hello, World!`" Now this may seem like an absolutely ridiculous idea, why would an object communicate to another object and what sort of meaningful thing could a lifeless object ever say. However, this happens every day. Computers are in a constant state of discussion with each other and are engaging in "meaningful conversation". From this viewpoint then, if we gave an object the ability to communicate like a computer communicates how exactly would it express its message? We are beginning to see this more and more with "smart" objects like phones, watches, and TV's and the fundamental concept of this interaction is called Machine-To-Machine communication.

It's often too easy for the general public, and for some scholarship, to view the subject of networks and Internet media from a teleological perspective, in so far that the there is always a general trend or final status that is to be reached. When in fact, this viewpoint is not only reductive, but it is also detrimental to one's understanding of Internet media as a whole. By viewing Internet media from this perspective, we not only blind ourselves to the tremendous amount of distal factors that go into the development of new media but to the failed projects, businesses, or technologies that existed along the way. However, there has been a general trend among recent scholarship to view digital media from a philosophical perspective, that is grounded in sciences, and from a technological perspective so that we can not only understand how a technology operates but what were the governing principles and distal ideas that went into its creation.

My research attempts to break down and survey this dichotomy, the convergence between academia and the technological, with special attention paid to the way in which the Internet, the World Wide Web, and protocols are discussed. In order to adequately discern what affordances were favorable, and by proxy will be favorable in the next iteration of the Web, I break down this teleological understand of Web history and examine critically the affordances of two technologies in a key transitional period between Web 1.0 and 2.0. Through this close analysis between two technologies we will be able to discern why, out of the many possibilities and iterations of the Web that could have been, did the transition from 1.0 to 2.0 happen to better serve user generated content and what might we learn from this to actualize a specific version of Web 3.0 utilizing Machine-To-Machine communication.

3. Existing Scholarship

It is important to look at these subjects and case studies from a scholarly and technological perspective because as of now the Internet and the Web are over 40 and 20 years old respectably and in order for it to progress forwards it is necessary to understand and historicize it. The Internet and the Web has gone through many phases, each of which that necessitate analysis, and soon it will enter into a new age that will need to build on these analyses in order to speculate what technological advancements need to happen.

3.1 Early Internet And Web 1.0

It is important to note that there is a significant distinction between the Internet and the Web and that over the years the two have been used interchangeably and have become synonymous with each other. That is not the case. The Internet is a system, or network of networks, that is utilized by computers and devices to communicate with each other on this interconnected networks (Inter-Net). Whereas the World Wide Web is a resource, or application, which utilizes the Internet and it is a series of connected hypermedia linked by hyperlinks. When talking about the early history of the Internet and the Web we have a tendency to look at it from an antiquated perspective and from the mindset that they are both tools to serve the general population. When in reality the histories of these two technologies are grounded in the military and university research.

John Ryan in his book A History of the Internet and the Digital Future does an excellent job of surveying the topology, the history of the Internet, and how we have come to see it today. He posits that there are three characteristics that govern the Internet and the Web throughout all its iteration: "the Internet is a centrifugal force, user-driven and open" (Ryan, 3). To Ryan, this idea of openness and a user-driven Internet means that there needs to be a certain degree of democratization of information and that the mass public will want to use the Internet to access this information. In addition, the Internet as a centrifugal force means that it needs to be able to disseminate information/data regardless of the status of the network. This is explained in the context of geopolitical struggles and rising tensions between the US and the USSR. The US needed a way to communicate and retain control without the need for central points, that way if there were ever a nuclear strike messages would still be able to flow freely and it would not have an effect on control.

The idea of maintaining control is of special importance to Alexander Galloway in his book Protocol: How Control Exists After Decentralization, where he examines from a philosophical perspective, with a heavy basis in Deleuze, Foucault, and Marx how the concept of control societies are integral to the development protocols and networks. Galloway examines the historical context of control from Foucault's idea of sovereign societies as well as disciplinary societies. Galloway, and many others, see this as the predecessor to Deleuze's idea of control societies. Whereas sovereign and disciplinary control manifest themselves in the physical, societies of control manifest themselves within information technology.

Galloway presents the three dominant models for network theory. The first is a centralized network where information flows from one focal point (a hub) outwards to many nodes. Information is spread unilaterally throughout a centralized network. The second is a decentralized network, which is a multiplication of the centralized network, where there are many central focal points that have dependent nodes from them. The final is the distributed network, or the rhizomatic network as he, Deleuze, and Guattari, refer to it. Where there are no hubs and only nodes and the network exists in a constant state of milieu and growth. This is the model network that composes the Internet. Each node is autonomous and the network itself is neither linear nor hierarchical. One might argue that by having each node be autonomous it's indicative of a network without any sense of control. However, Galloway argues that it is in fact the opposite. Protocol is what governs and controls a distributed network. By having a protocol that governs the network every node is then self-policing and has a set actions of things they can and cannot do. This simplifies the complexity of a distributed network and allows for the ability to utilize it to its fullest.

Galloway then presents the forms that this control manifests itself. In doing so he quotes Tim Berners-Lee, the creator of the World Wide Web. According to Berners-Lee "The job of computers and networks, is to get out of the way, to not be seen… the technology should be transparent, so we interact with it intuitively" (Galloway, *Protocol*, 65). This is fitting because this is the foundational characteristic of the World Wide Web. The Web is transparent and the creation of the hyperlink as the "killer app" of Web 1.0 brought the intuitiveness and transparency to the Internet. It facilitated the model that Web 1.0 was for interacting with on a surface level and that its function was as a library for reading. So then, if the hyperlink was the "killer app" of Web 1.0 that grounded the Web in Ryan's three characteristics of centrifugal, user-drive, and openness then what exactly was the "killer app" of Web 2.0.

3.2 Web 2.0 and Representational State Transfer (REST)

Roy Fielding begins to tackle this very idea of the next "killer app" in his doctorate thesis <u>Architectural Styles and the Design of Network-based Software Architectures</u> where he positions multiple web architectures against each other as a way to strategize the efficacy of each so that we could fully utilize the Web, moving it away from a mere library of documents to a Web that was grounded in user generated content. He presents his own architecture "Representational State Transfer" as the best way to organize data and elements within the Web. REST describes a client to server architecture relationship where the interactions are stateless. According to REST everything sent between the client and the server is a resource and a resource can be anything but is most generally a web page, data, text, images, etc (these are most commonly referred to as static or variables). When a client requests a resource from a server, the server then sends the client a representation of that data. For Fielding, the goal of presenting REST was to suggest a solution to the problem of how to advance the web, while maintaining the same functionality it currently has, and to allow for future technologies to grow. Fielding states the defining characteristics that REST emphasizes in his introduction.

> REST emphasizes scalability of component interactions, generality of interfaces, independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems. (Fielding, xvii)

These characteristics when applied to the Hypertext Transfer Protocol (HTTP) govern how data is accessed on the Web. It presents how the common actions of GET, PUT, POST, and DELETE of HTTP should function and how data should transition between the client and the server using those actions. The act of simplifying the way data is transferred between the client and the server ushered in the possibility for Web 2.0 and user-generated content. For Fielding, these actions should occur the same way regardless of the interface (i.e. mobile or desktop).

The issue of the interface and the generality of interfaces are of special importance to Alexander Galloway again, in his book The Interface Effect. He begins his analysis by looking at Lev Manovich’s The Language of New Media as a foundation for which the majority of modern digital media scholarship has been based off of. Manovich states that new media objects have five basic characteristics. They can be numerically represented, they are modular, they can be automated, they are variable and can change, and they are transcoded. Galloway keeps this in mind when he begins to analyze the interface and its importance. Fielding would say that the interface is a physical object, it exists and that we interact with it on a daily, sometimes hourly, basis. Whereas Galloway refutes this and suggests that “The interface is a medium that does not mediate. It is unworkable” (Galloway, *Interface Effect*, 52). For Galloway, there is an important distinction between interface and intraface. Intraface is the word used to describe this imaginary dialogue between the workable and the unworkable, for him it is a zone of indecision and it is a part of the interface. Not only is it a zone of indecision it is a zone of intra-action. From this he posits that the interface, although unworkable, is always the effect in which this battle between the workable and unworkable plays out. There is a common assumption that the interface is a window or a door for us to use and for us to view the world, but instead it is actually an effect between the edges of what we see and the center of our attention when looking at objects through these interfaces. This approach to the interface and to generality of interfaces has become more relevant as Web 2.0 winds down. We are beginning to see ourselves shy away from this idea of “the social Web” and more on the “Internet of Things” or the “ubiquitous Web” where we are in constant connectivity and the line between what is and is not an interface becomes harder and harder to discern. The interface, and more importantly how we not only interact but intra-act with it as Karen Barad would say, is no longer a physical object but a concept by which we design products and protocols.

3.3 Web 3.0 and Future Technology

We live in a digital age that is in constant flux and on the verge of a new epoch. Many cite that Web 3.0 may take many phases, either the “Semantic Web”, the “Internet of Things", or constant and pervasive connectivity to name a few. The scholarship behind these prevailing possibilities is often looked at by the mass public as purely science fiction or that they are delusions of grandeur, however it is necessary to look at these speculative scholarships and speculative media because they offer a blue print through which we can model actual change. No example is more perfect than the Internet of Things (IoT).

Julian Bleeker exams the key concepts and fundamental assumptions for the IoT in his essay "A Manifesto for Networked Objects – Cohabiting with Pigeons, Arphids and Aibos in the Internet of Things". For Bleeker, the IoT

> Is more than a world of RFID tags and networked sensors. Once 'Things' are connected to the Internet, they can only but become enrolled as active, worldly participants by knitting together, facilitating and contributing to networks of social exchange and discourse, and rearranging the rules of occupancy and patterns of mobility within the physical word. (Bleeker, 2)

What this means from a basic level is that once everyday objects become communicable within the network and with themselves we are not only giving the network tremendous amounts of vital data, but we are giving these objects a sense of agency. These objects would no longer be referred to as objects, but rather as either "blogjects" or "spimes" (a combination of space and time). By giving these objects agency and the ability to communicate we are giving them the ability to "blog" in a sense. The "blogject's" life would be indexable and communicable with not only us in a meaningful way but with other objects. And in order for this to occur, we have to have a strong understanding of our past when it comes to technological advances and the defining scholarship that has governed it.

4. Comparative Analysis

As we can see from the summation of existing scholarship above that the history of the Web is not so black and white. The Web was based and founded solely on social interaction. The Web was founded in academia, with the purpose of sharing scholarly documents and with hopes that the mass public would eventually use it. The shifts in Web technology that we see every decade is not so much a shift in technological advancements that then play out in the world, but rather they are shifts in cultural trends. When we use the Web long enough, we begin to find and adapt certain features for social functions. The transition from Web 1.0 to 2.0 was facilitated by technological advancement, but first and foremost these drastic changes in epochs arise out a social and cultural needs. However, with this assumption comes a classical debate of "which came first" the technological advancements or the societal needs. For the purposes of this analysis the answer is that the technological advancements help facilitate new forms of social communications. However, this isn't to say that social functions and needs do not play a role in the development of protocols; these social needs and technological advancements operate in a symbiotic relationship. We will see this when I examine the shift between Web 1.0 and 2.0, and it is from this examination that I will be able to demonstrate the technological advancements that need to be met in order to serve the social needs and function of the future Web.

There needs to be a certain sense of isolation and periodization when discussing the Web or technological advancements. If we look at the entire scope of change it is impossible to discern any key points of transition, and if we use a myopic approach we lose all context. Alexander Galloway brings this up when he states

> Periodization is an initial technique that opens the path and allows us to gain access to history and historical differences … [and] that protocol is a system of management historically posterior to decentralization. (Galloway, *Protocol*, 21)

Through periodization Galloway was able to argue that protocol, as a system of control, precedes the decentralization of the Internet. Using Galloway's analysis and approach as the central model for my argument we will be able to see how through periodization inklings of technological change that need to happen for the Web to advance will begin to show. But in order to do this we need to ask ourselves what the current constraints on our system.

4.1 Historical Analysis

We are currently in a major transitional period in Web history. We are consuming bandwidth and data at alarming rates and are in need of a network architecture that is both smart, efficient, and secure with how it deals with data. The severity of issues we are facing is similar to the technological problems that existed right before the shift to Web 2.0. Before the eventual switch to Web 2.0 the client-server relationship was riddled with overly complex protocols that caused burdens on servers and slower performance. The Web was defined by static, library-like, Web pages with very little user interaction. The functionality of Web 1.0 rarely expanded upon the treasure trove of Geocities sites, personal homepages, and scientific institutions.

The reason why it never expanded upon this was because there wasn't an easy way for users to interact with web pages. However, in 2000 Roy Fielding introduced a design specification for a web architecture that would help alleviate all these problems. He called it REST (REpresentational State Transfer) and it relied on a simplified client-server relationship where users would be able to, by utilizing the functionality of HTTP, gather and edit data on servers easily. REST was a simpler solution to SOAP (Simple Object Access Protocol), which was released in 1998.

SOAP and REST provided similar functionality, however ultimately REST won out as becoming the preferred standard for web development. By looking at this conflict between competing architectures/protocols we are able to glean a possible strategy for how the architecture and infrastructure of the Web will have to change to actualize a desired version of the Web. We have to remind ourselves yet again that it is far too easy to just assume that there was a unified path that Web technologies followed, and that rather the history of the Web operates in a rhizomatic structure with any number of possibilities. It is necessary to analyze the different strata of these technologies, because these objects do not exist in a void. They are an assemblage of electrical impulses (computer readable binary), technological code (high-level programming languages), human use (the interface and interactivity of a technology), and abstract concepts that work together to actualize many different possibilities for the Web. These objects not only exist and interact within a rhizomatic structure of the Internet, as popularized by Delueze and Gutarri and then later picked up by Alexander Galloway, but they embody the rhizomatic structure with their utility and meaning.

4.1.1 SOAP

Developed by primarily Don Box, Bob Atkinson, and Mohsen Al Ghosein, SOAP was created as a “lightweight protocol for exchange of information in a decentralized, distributed environment” (Atkinson, Box, Ghosein, W3). It is a communication protocol encoded in XML (Extended Markup Language) tree that was designed to facilitate communication and the transfer of data between peers in a decentralized network. The three main components of SOAP are

> The SOAP envelope construct defines an overall framework for expressing what is in a message; who should deal with it, and whether it is optional or mandatory. The SOAP encoding rules defines a serialization mechanism that can be used to exchange instances of application-defined datatypes. [And] the SOAP RPC representation defines a convention that can be used to represent remove procedure calls and responses. (Atkinson, Box, Ghosein, W3)

The envelope for SOAP is a way to not only encapsulate a message or a response, but as a way to prime a server that receives this envelope for what type of information it should be looking for and how to decode what's inside the tree. Inside the envelope is where there is a definition, or a reference to a definition within the SOAP specification, to what the syntactical structure of the XML will look like. Among other things, this primes the server or recipient for what type of language and syntax is going to be used in the XML tree and how it is going to be organized. Finally the SOAP RPC (Remote Procedure Call) using the predefined syntactical structure is what tells the server what information to grab, delete, create, etc. The server then can send a response back, in the same fashion using a new SOAP envelope, body, and RPC with the information needed.

Even though the technology itself is well documented and free to implement, its history and motivation for development is based in corporate interests. The documentation for SOAP was developed in two days by Atkinson, Goshein, and Box, who were all Microsoft employees, and for Microsoft in 1998. Shortly there after the three were then able to implement SOAP successfully and began testing it (Ferguson, Developers Journal). Because Microsoft developed it, they held proprietary concerns paramount over building a protocol that could be easily implemented and modular. This is evident in the basic structure of a SOAP response/request.

The basic structure for a request/response requires a lot of priming for the server to understand what information is needed. Rather than just asking the server for information located at a specific URI, a SOAP request needs to be wrapped in an XML document that specifies all of the permissions for the data, what the syntax of the request/response is going to be in, and the data itself. Once the envelope is constructed a SOAP request has to be primed with an HTTP request. An example of this would look like

```
POST /StockQuote HTTP/1.1
HOST: www.stockquoteserver.com
Content-Type: text/xml; charset=″utf-8″
Content-Length: nnnn
SOAPAction: “http://example.org/2001/06/quotes″

<env:Envelope
  xmlns:env=″http://www.w3.org/2001/06/soap-envelope″>
  <env:Body>
    <m:GetLastTradePrice
     env:encodingStyle=http://www.w3.org/2001/06/soap
     encoding″xmlns:m=″http://example.org/2001/06/quot
     es″>
      <symbold>DIS</symbol>
    </m:GetLastTradePrice>
  <env:Body>
</env:Envelope> (Atkinson, Box, Ghosein, W3)
```

This is a sample request that utilizes the functionality of an HTTP request to grab the most recent stock trade price. It defines what type of request is going to happen, the host server, any content, the action, and then the physical XML SOAP message. We can see this by looking at the markup. The first section is an HTTP request, using an existing protocol to direct the message where to go. The XML tree is dived into the three key sections. First is the envelope, which has encoded in it the type of structure and syntax this XML tree is going to use. Then defines the body, which is only meant to encapsulate the message. The message then defines what type of information it is looking for and where to access it. Most of the work for this type of message is done in abiding by the strict protocol syntax of HTTP and SOAP. Even though a computer or a server can process this message relatively fast it is only when thousands of these responses get sent out that it begins to slow the network down. After each request the server needs to parse the XML tree, discern what information is needed, grab and gather the data, then reconstruct a response in another SOAP envelope to the right URI. This type of response, while helping to facilitate client server interactivity, is very rigid in how it operates and requires tremendous amounts of unnecessary text.

4.1.2 REST

This is where the simplicity of REST comes into play. Developed by Roy Fielding for his dissertation at University California Irvine, REST provides a simple, scalable, approach to web architecture and to representing or accessing data. By having any resource that needs to be accessed uniquely identifiable by a URI and have that URI be a representation of the type of data it allows for a very simple call for information based on that URI regardless of what language used. Even though REST is not a communication protocol like SOAP is, they both serve similar purposes for the modern Web. Unlike SOAP, REST does not have a defined syntax or based upon any specific language (like XML), rather it is just a model for how data should be represented and how data should be accessed to best utilize the web.

Fielding took tremendous time considering all possible ways to implement the specifications for REST, and the possible constraints that a model like this might undertake. This is exemplified in Fielding’s dissertation by having the first half of his dissertation being merely about possible constraints and what a web architecture model means both technically and abstractly. Once he thoroughly explores all possible options and how to efficiently implement a model into the network that is backwards compatible and capable of scalability he then begins to talk about how REST, as a model would operate in this paradigm.

REST relies on a separation between the client and the server, stateless communication, cached results, uniformity, layered systems, and most importantly code-on-demand (Fielding, 78-85). What Fielding means when he defines this is that by abstracting both the client and the server from each other each can be developed independently from each other, regardless of type of database or type of interface. By keeping communication stateless, meaning that no amount of information is stored between sessions on the client side, we allow for a fresh slate every time the client requests information. However, that being said there does need to be a way for servers to cache information in case a client makes the same request for information multiple times. This allows for less of a burden on network functionality. As for uniformity, Fielding states that, “By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified and the visibility of interactions is improved”(Fielding, 81). Layered systems allow for the ability to have only certain parts of a code base public and capable of interaction. It allows for a certain level of abstraction that fosters privacy and security for resources. Finally and recently becoming most important is code-on-demand. By having the ability for a RESTful interface separate from the browser, either through a Java applet, downloadable application, etc, fosters a uniform usability regardless of medium or interface. Now that we have examined what constraints dictate REST we can begin to examine how it differs from SOAP and how exactly does a RESTful request operate.

REST is meant to operate within hypermedia and according to Fielding this means that,

> REST ignores the details of component implementation and protocol syntax in order to focus on the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements. (Fielding, 86)

Unlike SOAP, REST is not a protocol. It is simply just a model and architecture to represent data in your network. It relies of grabbing data through HTTP (which is a protocol), and meaningfully identifiable URIs. Because it's just an abstract model for representing data, REST is easily scalable and malleable to any situation or language (provided you are accessing the right URIs).

A typical REST call, since it relies on HTTP, incorporates the basic semantic elements of an HTTP request (GET, PUT, POST, DELET). An example of what this would look like in JavaScript if a user clicked a button that fired off a request for data would look like

```
$.ajax({
     url: 'http://path/to/my/awesome/data',
     type: 'GET',
     data: 'Id=1&Username=MikeLorenz',
     dataType: 'json',
     success: 'function(response) {alert('Here is your
          awesome data' + response);}
});
```

This code snippet relies only JavaScript/jQuery/AJAX, and the only thing about this that is RESTful is the way in which we are accessing data from the URI. Because REST is just a model for representation we can use any language we want to access the URI. For this specific request we are combining jQuery (a popular JavaScript library used to manipulate elements on a webpage) with AJAX (Asynchronous JavaScript and XML) to make a request for information. This is represented by the text $.ajax({});. Within the AJAX call we are able to utilize standards and parameters predefined in AJAX to compose a RESTful request. The URL parameter defines the REST URL we want to access. The type parameter defines what type of actions we will be taking with this information (i.e. GET, PUT, POST, DELETE). We are relying on GET from HTTP to execute a RESTful request. The data parameter is then appended to the URL parameter so that when it is sent to the server it knows exactly what ID to look at and what the username is. The dataType parameter then specifies to the server what format we want our response to be. Finally, the request is then registered by the server, which is already built to all of the necessary constraints detailed by Fielding above. The data is then returned in the response variable within the success function. The format in which the data is returned (JSON, JSONP, HTML, or plain text) is entirely left up to the software engineer designing the database architecture or the engineering creating the request.

4.2 SOAP vs. REST

Now that we analyzed the bare bones technical aspects of each technology we can compare the affordances, both physical and social, that each technology has in order to shine light on the issues we face currently. In late 90's and the turn of the century we were in a time of technological crisis, but a boom in technological advancements. The dot-com bubble brought technological minds together with corporate interests. Early Web and Web 1.0 was rooted in academia, and Web 2.0 as we can see from above came from this power struggle between corporate structures and the FLOSS movement. This physical and theoretical struggle is perfectly summarized by Eric Raymond's software models in The Cathedral and the Bazaar. The cathedral is highly structured, controlled, and monitored. Whereas the bazaar comes forth organically, is built to adapt and grow with the change in environment. In the transition between 1.0 and 2.0 SOAP is the cathedral structure that is highly regimented and controlled with its predefined structure and reliance on specific protocols and languages. It is for corporate interest first and foremost, and then about developing for the community at large. REST, on the other hand is the bazaar. Developed to serve the public at large and in conjunction with the HTTP specification, REST and Roy Fielding represent everything that the bazaar stands for.

The analogy between the bazaar and the cathedral allows us to adequately understand why software engineers chose REST over SOAP from a philosophical standpoint. REST can be expanded upon, changed, and manipulated to serve whatever function the software engineer needs it for. Whereas SOAP requires that you remain within the proper syntax, structure, and that you have the server capacity to parse and construct XML at large volumes. For this particular instance, because of the simplicity of REST over SOAP software engineers gravitated towards the approach that allowed for faster, lighter, and at the time efficient communication. It is important to note that while it would seem logical to always choose the most efficient technology to serve the greater public that is not always the case. However, in this instance because the demand for a faster, more reliable, and a scalable approach to the client server relationship REST was popularized as the default way to build web applications. It bridged the gab between static, almost library-like, functionality of Web 1.0 to a web that serves a social function. It brought us to Web 2.0.

4.3 Theoretical Analysis

If, technologically speaking, the transition from Web 1.0 to 2.0 is defined by the ability for faster, uniform, data transfers then what does this transition and the technologies that defined it signify philosophically. The ability to share and create content further democratized the Web. It also allowed for users to have access to types of content they never had before, and it ushered in a new level of control society as popularized by Deleuze in his essay Postscript on the Societies of Control. It is widely accepted that the Internet as a rhizomatic structure grows forth organically, has many layers, and never ends (it is in a constant state of milieu). However, if we understand control societies in this way we are able to discern the levels of control and how to access them. By this I mean that access within a control society does not give a user or an agent access to everything. Access can mean fundamentally different concepts for different aspects of a control society. Societies of control are governed by access or passwords and while REST and SOAP have access implemented into the very core of their structure the way in which access is represented changes. Access does not mean the same thing to REST as it does for SOAP, each fundamentally approach access from two completely different ideological standpoints.

SOAP, because it is layered on top of an XML tree has an extra level of abstraction, and with that an extra level of access that needs to be broken through. Now this isn't to say that abstraction of data or abstraction in general is negative, however the way that it is implemented in SOAP further separates the data from the carrier. In order to decode, a computer needs to not only understand the SOAP protocol but XML as well and then it is finally able to understand what specifically the request is looking for. The computer first needs to access its own understanding of what SOAP is, then XML, and then try and access a logical order for the information that is needed from the SOAP request. Finally, once it has the URI with the destination for the data it needs, it needs to access that section of the database. At every step of the decoding process there is a series of parsing and accessing structures to properly understand the message. For SOAP, control and access are about decoding and parsing a request.

Yet REST on the other hand doesn't rely on any existing programming language and just works through an HTTP request. Because of this flexibility it lowers the level of abstraction and the degree of access necessary to use it. REST can be implemented in many different languages because it is just a model of representation. It can take the form of an AJAX call or, just an HTTP request. For REST, control and access as it pertains to language and syntax are negligible. Control for REST is much more about what the user has access to what specific amount of data and what types of actions that user can have on that data set. REST is just a model for requesting data and in order to access it a computer just has to be sent an HTTP request with the URI.

REST provides us the ability to interact on the Web much faster, but it has many constraints and limitations. The transition from 1.0 to 2.0 much like the web itself was produced through academia, the open source movement, and a reliance on modularity rather than corporate interests. We can see this when comparing REST and SOAP. It is from this focus on corporate interest as opposed to developing for the mass public that ultimately led to SOAP failing to catch on. However, now the transition that we have to go through is no longer marked by the needs of academia. This is because the metric with which we were using to gauge has shifted. Suddenly, protocols and architectures that were developed by the academy, that were once seen as stance against proprietary languages, are no longer products of freethinking and free structured entities. While REST remains a stable force in maintaining uniform usability across different Web applications, we are still held back by the way in which we communicate. As Foucault and later Deleuze and Guattari conceptualize, the academy was originally an entity of a discipline society however with the rise of control societies it maintained discipline and control by using access. The academy however is now as controlling as corporate structures. If this is the case then transition must come from both individuals and from those with a vested interest in the future of the Web.

5. Current Problems

REST provided an easier technological model for users to create content, however with this ease in creation comes the problem of making sense of all this information. The current obstacles we are facing in terms of technological advancements are how to deal with massive amounts of data, efficiently, securely, and to utilize it in a meaningful manner that will facilitate everyday life. This doesn't necessarily have to be a new architecture like REST, but rather it could be an extension upon existing technologies like the implementation of IPv6. Even though these obstacles are technical at face value, they have a much deeper social implication that needs to be fulfilled. This can be seen when looking at the constraints and limitations of REST.

One of the major limitations that REST has is that by default security and privacy (as it pertains to sensitive data) are not built into the model. This makes sense because REST is not a protocol centered on privacy or security, but rather it is a model centered on efficiency and the representation of data. Security then is left up to the software engineer and how they want to implement alternative methods for obfuscating sensitive data through REST transmissions. However, one of the major issues with obfuscation is that there is never a foolproof way to obfuscate data. There will always be a method to remove salt encryptions or any other form of obfuscating data. Even though this transparency of data is a limitation, it is what REST was designed to do. It facilitated the client-to-server relationship by representing data easily and making it easily accessible through a URI.

Another limitation is that it can, depending on how it's implemented, expose your database structure to anyone that has the proper URI. For example, if I have user data stored at http://myapplication.mywebsite/profiles/userid/123 then I am exposing to anyone with this URI that my database is structured such that there is a parent resource called profiles, that userid is located within, and then the user's specific id. This is called strong coupling and a malicious attack would easily be able to use this information to expose more sensitive information about the structure of an application. This type of implementation though can easily be avoided or prevented by removing the ability to send requests from non-registered computers, implementing a proper URI structure, and by correctly restricting access to your database. While strong coupling has its advantages, because it allows for an easier understanding of an application structure using easily identifiable nouns as resource names, a solution to this would be to randomize the resource name. The very same URI, using a loose coupling approach, would look like http://myapplication.mywebsite/profiles/cb77380-7425-11df-93f2-0800 where the numbers on the end is an obfuscated representation of the user id number. However, as mentioned earlier obfuscation should not be trusted entirely because there is always a way to decrypt an input. Even though the above is an example of loose coupling, it requires that both parties (the supplier of the URI, and the person or application interacting with that URI) have a mutual understanding of what the encrypted user id means. It is important to note that yet again this is not so much a limitation in the design of REST, but rather an example of a use that could be problematic.

A social limitation of Web 2.0 is that, while it does facilitate collaborative creation of content, it is still “buggy” and not predicative of what type of content a user wants to see. We are using the Web more so than ever, and the types of keywords we are using aren’t entirely expressive of the type of content that we are looking for. This is evident when a user will get more results when searching “limitations of REST” rather than “What are the limitations of REST in Web 2.0”. This is because even though the algorithms used by Google and Microsoft are advanced, they are not advanced enough to recognize syntactically full sentences. HTML semantic elements are starting to combat this problem, however much more work needs to be done before they are correctly implemented.

In addition Web 2.0 is limited in the sense of how information is distributed between devices. For example, if a user wants to send a message to someone else that is in the same room as them the message gets sent to the nearest cell tower, received, interpreted, and sent out to the corresponding person. There is a tremendous amount of time wasted when the message is sent to the cell tower and back, it should just be easier to send a message straight to the person next to you. This is often referred to as Machine-to-Machine (M2M) communication. M2M, and the idea that sending and receiving data should be smart, is integral to how Web 3.0 operates.

As a whole the system that we have now, regardless of whether or not REST is used as a model, isn't prepared for or isn't utilizing the amount of data we have and the data we are collecting to the fullest. So then how exactly would a system like this operate? What technological advancements are necessary? What social factors will need to be addressed? And finally what sort of effects could this bring about?

6. Machine-To-Machine Communication

Web 3.0 has many names and ultimately will serve many functions. It has been called the Internet of Things (IoT), the Semantic Web, or the ubiquitous Web. Regardless of what it is called or what function it serves all popular definitions or speculative models have big data being used efficiently to facilitate everyday life. As we have seen in the transition between Web 1.0 and 2.0 with the battle between REST and SOAP that efficiency, simplicity, and modularity should be the pillars of Web 3.0 technologies. This is echoed in Jean-François Blanchette's essay A Material History of Bits when he discusses the importance on modularity and layering. Blanchette states that,

> The sending of a simple email over the Internet requires the correct functioning of thousands upon thousands of heterogeneous material and logical components, connected together in a network of staggering complexity. Such a system must be able to accommodate, among other things, growth in size and traffic, technical evolution and decay, diversity of implementations, integration of new services to answer unanticipated needs, emergent behaviors, etc (Blanchette, 8).

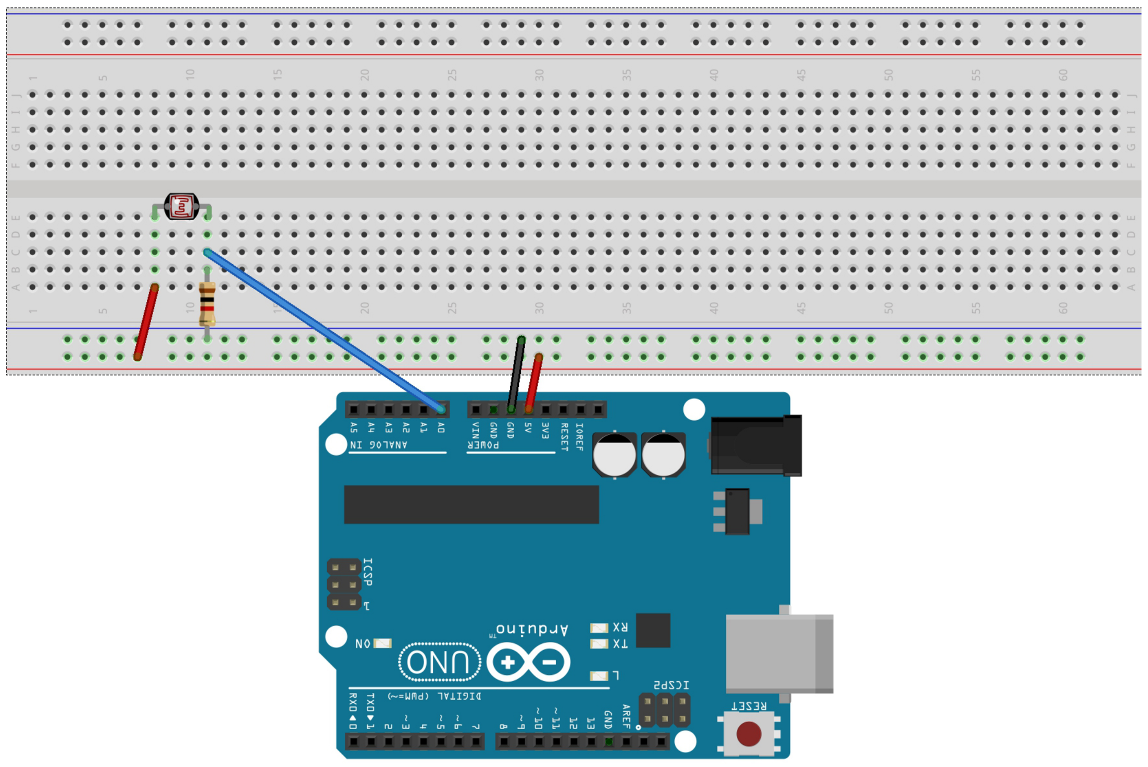
The network we are operating in currently involves navigating between many different layers, so then why should we not build a protocol that decreases the distance between these modular systems and if designed properly could be capable of subverting them.

As discussed earlier, one possible implementation of Web 3.0 is the Internet of Things. The IoT is characterized by having every device or object indexable, capable of historically recording data for the entirety of its lifespan, and be able to communicate this information in a meaningful manner not only to us but also to other devices. If this were to become actualized there would need to be a system implemented that allows for a significant increase in uniquely identifiable addresses. A necessary advancement for this to happen is for the implementation of IPv6. In the current iteration of IP (IPv4) we are drastically running out of uniquely identifiable addresses. To solve this IPv6 uses 128-bit addresses as opposed to the IPv4, which uses 32-bit addresses. An IPv6 address would look like 2001:0DB8:AC10:FE01:50DA:00F1:4391:FA29 as opposed to an IPv4 address which would look like 216.165.95.70. By expanding the bit value IPv6 is able to accommodate 2^{128} unique IP addresses. These addresses are composed of eight sections, each section containing a four digit hexadecimal number, separated by colons. By implementing IPv6 we allow ourselves, for now, an almost infinite number of unique addresses and the significance of this is that these new addresses could then be given out to “blogjects” across an IoT. IPv6 is the first step to actualizing a possible iteration of Web 3.0.

Extending to IPv6 allows for each object to be indexable, but it does not necessarily help these objects communicate with each other efficiently, simply, and modularly. For this, we could use RFID (Radio-Frequency Identification) chips or NFC (Near Field Communication) chips with extremely small flash storage. RFID has come a long way from being primarily used to prevent theft in stores by placing a little magnetic strip inside an item. By using RFID or NFC, in conjunction with IPv6, as a means to build this modular, efficient, and simple idea for the Web we give objects agency to make their own decisions about what should and should not be communicated over a network. This allows for objects the ability to communicate with not only each other but with us. It also allows, through the use of tiny ICs (Integrated Circuits), the massive amount of continuous data that these objects are collecting to distribute the information in a meaningful way.

6.1 Technological Analysis

Yet what exactly would this look like from a bare bones technical standpoint? Does it look anything like REST or SOAP? In fact it is a completely different approach to communication that neither REST nor SOAP propose. What made REST so popular was that there was a certain level of abstraction between the client and the server. With M2M on the other hand, the client (in this case any sort of object or machine) becomes the server, and is capable of recording, communicating, and interacting with other clients. For example, what this might look like in Arduino (an open source language, which is based in C, used for interacting with the Arduino microcontroller) and Processing (an open source platform, which is based in Java, released by MIT) for a microcontroller to be waiting until a message is sent to it and then interpreting that data in a meaningful way would look like this.



Arduino layout with photoresistor attached to pin A0 and breadboard

Arduino Code

```
const int photoPin = A0;
boolean light = false;

void setup(){
    Serial.begin(9600);
}

void loop(){
    int photoVal = analogRead(photoPin);
    photoVal = map(photoVal, 0, 1023, 127, 0);
    if(photoVal < 90) {
        light =  true;
    } else {
        light = false;
    }
    Serial.write(light);
    delay(20);
}
```

Processing Code

```
import processing.serial.*;
Serial port;

void setup(){
    port = new Serial(this,
          "/dev/tty.usbmodemfd121",9600);
    port.write(65);
}

void draw(){
    while(port.available() > 0){
        int input = port.read();
     try{
         if(input == 1){
             println("You're home! Let's open up your
               browser!");
             open("/Applications/Google Chrome.app");
         } else {
             println("Oh no! You're leaving me. Okay,
               I'll close Chrome for
                   you.");
             Runtime rt = Runtime.getRuntime();
             rt.exec("pkil –HUP Google Chrome");
         }
     } catch (Exception e){
         println(e);
     }
    }
}
```

What this program does, once it is properly installed into an Arduino microcontroller and computer, is wait until you turn on your light then it sends a message to a computer saying that a light has been turned on. The computer then interprets that message and opens up the web browser Google Chrome. If your light is turned off, in the instance that you leave your room or house, it will close your browser for you. How it accomplishes this though needs to be explained a little bit further in order to adequately understand what it exactly means to give object agency and the ability to make decisions without being told to.

We begin by defining what pin on the Arduino will we be looking for a value and assigning it a variable name of photoPin, since the pin will only be used by a photoresistor. Then we define a boolean variable light and initialize it to false, so that when the program runs, it initially thinks the lights are off. After that we define a function that is only going to be run once called setup, and its only purpose is to start the serial port at a specific interval. This essentially primes the Arduino to allow it to send messages to another machine or object. Then we define the key part of program, the loop function. The loop function's key purpose, in the context of M2M and the IoT is to allow for continuous data collection. It is run extremely fast and once it finishes running, it runs again allowing for it to constantly output data. Inside the loop function we are reading in the value from the pin and mapping it to a smaller number. Then we make a key decision that this entire program is based off of. What exactly constitutes a light being on or off for a computer? For this code a light on is any value from the pin that is less than 90, anything greater than that we can assume the light is off. We then send out this message that the light is either on or off to any device connected to the Arduino. Yet the explanation above only cover one half of the metaphorical "handshake" that one machine has to make to another machine for them to both send messages to each other.

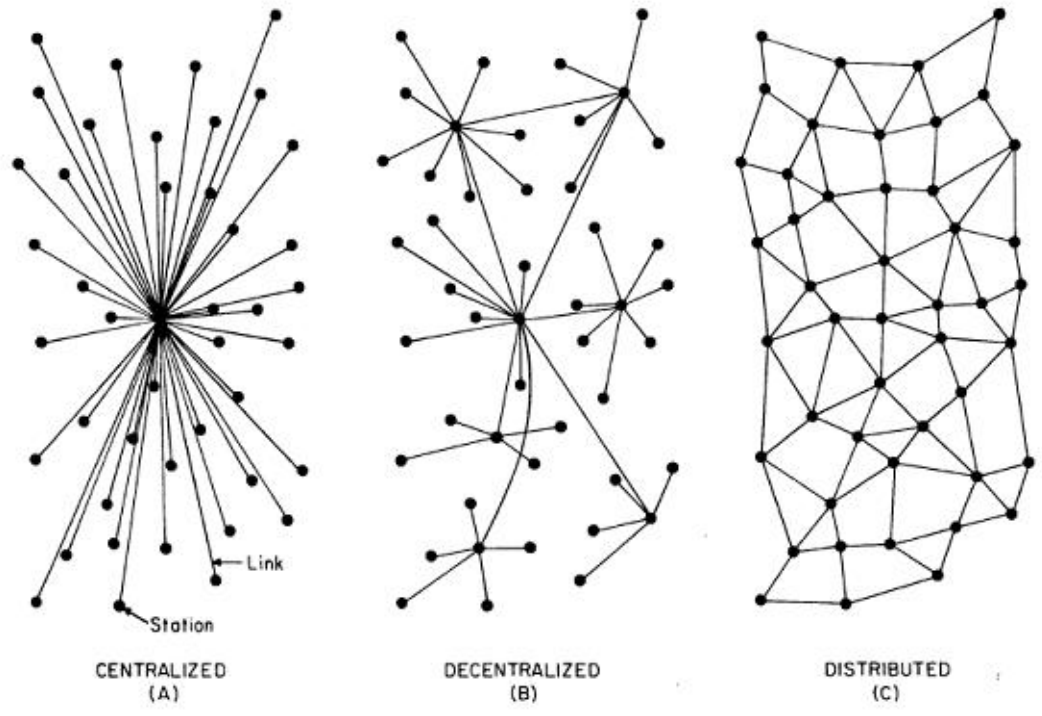
The other half of the code above is able to interpret what sort of message is being sent to the Arduino and make a meaningful decision based upon that message. Very similar to the Arduino code, the Processing code has two key functions that serve similar purposes except that this code is run on the computer that is attached to the Arduino. We begin in the setup function and define what port should we be listening for a message. Then in the draw function, similar to the loop function, which is run extremely fast multiple times per second we listen to that port for a message. Essentially, while the port is available we read an integer from the port. Because of the basics of coding and information theory we can assign the value that was sent to the computer (either 'true' or 'false') is an integer of either 1 or 0. Then we check if the value sent to the computer is either 1 or 0. If it is 1 then we know that the Arduino is telling the computer that a light is on, then we run a command to open Google Chrome. If not, then we run a command to 'kill', essentially close, Google Chrome.

Although this is a very primitive example of M2M it lays down the basics of what exactly from a technological perspective needs to occur so that one machine can send a message to another machine, without using the Internet. In this case though the two machines are wired together and are communicating through serial data. In a much higher-level product this type of communication would occur through RFID, NFC, or any other form of wireless communication. These two objects are operating in a simple, efficient, and if there were more than two objects communicating at once in a modular way. By giving each object the ability to record and send out data we embody a certain sense of agency in these objects. These objects are able to make decisions independent of us.

6.2 Theoretical Analysis

However, now that we have examined what M2M looks like primitively from technological perspective we must also examine what this means from a theoretical context and what guiding principles should play key roles in its development, as it pertains to control. For this we need to understand how data is transported across a network. Alexander Galloway discusses this when he analyzes the affordances and limitations of a centralized, decentralized, and distributed network. He states that

> The shift includes a movement away from central bureaucracies and vertical hierarchies toward a broad network of autonomous social actors… Distributed networks have no central hubs and no radial nodes. Instead each entity in the distributed network is an autonomous agent. (Galloway, 33)



Centralized, Decentralized and Distributed Systems (Paul Baran, 1964)

By having each node in the network be autonomous and capable of communication we allow for communication to occur much faster, for it to grow organically, and for the network to still be able to run if one node is removed. If we think of the distributed network in relation to how content is access we will see that the majority of how content is accessed is through making a request to one single location or server. This is very similar to the problem mentioned earlier where a message sent, regardless of how close the recipient is will always go through a central node. This defeats the purpose of a distributed network. Then how exactly would we solve this problem?

If we think of distributing content much like we understand a distributed network then we begin to grasp at the idea of M2M. David Lake, Ammar Rayes, and Monique Morrow in their article Internet of Things explain this by stating,

> Context that is distributed as opposed to centralized is a core architectural component of the IoT for three mains reasons: Data Collection … Network Resource Preservations … [and] Closed-Loop Functioning. (Lake, Morrow, Rayes, Internet of Things)

By allowing for M2M communication between machines in close proximity not only do we fully utilize the functionality of a distributed network but we drastically improve the speed of communication between two or more devices because they no longer have to reach some third-party node at some indeterminate location. We are beginning to see inklings of this now with concept designs for self-driving cars, but if we extrapolate this to an Internet of Things size we can truly begin to see how big data should be dealt with.

Rather than having data being collected in one single point, like a server farm, objects within the IoT don't need to communicate to a central unit, and are able to record their entire history in a small lightweight storage. Most importantly though, objects wouldn't need to bog down the network with unnecessary data communication, thereby efficiently dealing with big data by dispersing the computational power needed to an infinite number of devices, similar to multiprocessing. Instead data is sent directly to the object in the perimeter.

From a theoretical perspective M2M communication follows the same logic that allowed REST to be seen as the better model over SOAP. Compared to traditional communication that is centralized both operate within the definition of a control society but to both control means something completely different. For traditional communication that is decentralized but still relies on key nodes control is about having access to that central node. For distributed communication control is governed by the objects themselves. M2M communication subverts this traditional role of an overarching control society and has access be governed by the objects that want to communicate with each other. If REST democratized content creation on the Web by allowing users the ability to interact with the Web then M2M democratizes the Web for objects, regardless of human agency.

In the Arduino and Processing example above we can now see what exactly control means to each object. Control for the Arduino is not about access to a WiFi password, or access to a database. In fact, if we were to use the model we currently use for how requests are made on the Internet, the Arduino would be connected to the Internet and have the value for whether or not the light is on or off stored at some server in some indeterminate location. If the light were turned on, the Arduino would have to access the Internet, send a request to the server to update the information, and then notify the computer that some undisclosed information was updated. Instead operating in a M2M paradigm control for the Arduino and computer are about accessing each other.

Yet how do we deal with a system that is always present and always collecting data about us? It is far too easy to say that we should just obfuscate data, implement higher and harder security measures. As I've mentioned above, obfuscation is not a clear-cut solution to protecting data. Obfuscation can always be reversed and data is never always protected. Even if data is sent directly using M2M it isn't secure at all. The signal can always be interrupted or monitored by something in the vicinity.

One solution is to have strict authorization protocols between devices so that only authorized devices can communication with other devices. But even with this approach authorization can be faked. The better, and albeit harder, solution is to construct a new meaning and model for private information. If private information can always be stolen, deobfuscated, or spoofed then perhaps we have to construct a new form of private information that serves the same function as the model we have now but does not have any potentially harmful private information. For example, instead of investing energy in finding new ways to salt social security numbers we would have an alternative numbering system that serves the same function as social security numbers but if stolen cannot be used for malicious activities.

7. Conclusions and Future Work

We have seen by looking at the history of the Web from its early ages in 1.0 to the transitional period of 2.0 that efficiency has been held paramount above all else. By looking at this period, and more specifically the battle between REST and SOAP, from both a technological and theoretical perspective we can see that how engineers reinvented the idea of data transmission and representation allowed for the full utilization of the HTTP specifications, thus bringing in Web 2.0. Yet when looking forward to the idea of Web 3.0 we should keep in mind the battles that were fought before and why certain values were picked over others. If we want to actualize this idea of the IoT we need to not only address the social demands of ubiquitous use of information, but the technological advancements necessary. We would need to define some sort of specification or paradigm, that is mutually agreed upon, for how two objects should communicate to each other in a distributed way independent of the Internet. By doing so we give objects agency and take back control from key nodes and corporations and embody each object with control over itself. These social and technological demands need to be met with the implementation of IPv6 and the standardization of M2M communication.

By delving into the history of key Web technologies we were able to unearth a possible roadmap for one specific iteration of Web 3.0. My research is by no means encompasses the full scale of what would need to occur for Web 3.0. It is my hope though that through this research others will begin to question what exactly would need to be built or thought out to actualize the Semantic Web, the Ubiquitous Web, or any other form of the next Web. Perhaps looking at REST and SOAP would not be beneficial for unearthing the technology of the Semantic Web because REST and SOAP are models and protocols respectively, while the Semantic Web could be an entirely new programming language. Regardless, M2M is a necessary step for embodying objects with agency so that one day they too might be able to tell us "`Hello, world`".

Works Referenced

Bendrath, Ralf, and Milton Mueller. "The End of the Net as we Know it? Deep Packet Inspection and Internet Governance." New Media & Society 13.7 (2011): 1142.

Bhardwaj,Puneet, and Vandana Dubey, and Shipra Sharma. "Comparative Analysis of Reactive and Proactive Protocol of Mobile Ad-Hoc Network ." International Journal on Computer Science and Engineering 4.7 (2012): 1281.

Blanchette, Jean-François. "A Material History of Bits." Journal of the American Society for Information Science and Technology 62.6 (2011): 1042. . 04/15/14 <http://onlinelibrary.wiley.com/doi/10.1002/asi.21542/abstract>.

Bleecker, Julian. "A Manifesto for Networked Objects — Cohabiting with Pigeons, Arphids and Aibos in the Internet of Things." Near Future Labratory (2006):

10/1/13. . Near Future Labratory <http://nearfuturelaboratory.com/2006/02/26/a-manifesto-for-networked-objects/>.

Braman, Sandra. "Privacy by Design: Networked Computing, 1969−1979." New Media & Society 14.5 (2012): 798.

Deleuze, Gilles. A Thousand Plateaus Capitalism and Schizophrenia. Minneapolis : University of Minnesota Press, 1987.

Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, Dave Winer. "Simple Object Access Protocol (SOAP) 1.1." 2000. <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>.

Ferguson, Derek. "Exclusive .NET Developer's Journal "Indigo" Interview with Microsoft's Don Box." 2004. <http://dotnet.sys-con.com/node/45908>.

Fielding, Roy. Architectural Styles and the Design of Network-Based Software Architectures. Doctorate of Philosophy University of California, Irvine, 2000.

Galloway, Alexander R. The Interface Effect. Cambridge, UK ; Malden, MA : Polity Press, 2012.

Galloway, Alexander R. Protocol How Control Exists After Decentralization. Cambridge, Mass.: MIT Press.

Lake, David, and Ammar Rayes, and Monique Morrow. "The Internet of Things." The Internet Protocol Journal 15.3 (2012).

Latzer, Michael. "Information and Communication Technology Innovations: Radical and disruptive?" New Media & Society 11.4 (2009): 599.

Postel, Jonathan. RFC 793: Transmission Control Protocol. RFC 793 - Transmission Control Protocol, 1981.

Raymond, Eric S. The Cathedral and the Bazaar : Musings on Linux and Open Source by an Accidental Revolutionary. Beijing ; Cambridge, Mass.: O'Reilly.

Smith, Greg. "Web 3.0: 'Vague, but Exciting'." MediaWeek 19.24 (2009): 19-.

Strom, David. "The Demise of Web 2.0." The Internet Protocol Journal 15.3 (2012).

Tian, Ye, Kai Xu, and Nirwan Ansari. "TCP in Wireless Environments: Problems and Solutions." IEEE Communications Magazine 43.3 (2005): S27-32.