Web-Based Data Visualization with 3D Portrayals for Communications Applications

Nikita Igorevich Sharakhov

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science
In
Computer Engineering

Carl Dietrich
Nicholas Polys
Vuk Marojevic

06/19/2014
Blacksburg, VA

Keywords: Dynamic Spectrum Access, Software Defined Radio Testbed, Computer Graphics, Web Applications, WebGL, HTML5, WebSockets, Geospatial, Visualization, 3D, Broadband

Copyright Nikita Igorevich Sharakhov
Web-Based Data Visualization with 3D Portrayals for Communications Applications

Nikita Igorevich Sharakhov

ABSTRACT

The modern web has evolved into a highly capable software platform, which enables near-native performance, while offering installation-free cross-platform applications with a uniform user base and rapid update deployment. SVG, WebGL, and HTML5 Canvas, along with various higher-level JavaScript frameworks allow web applications to drive both 2D and 3D visualization. These technologies allow developing novel visualization applications, which can be applied in the communication domain to geospatially map service quality, and to provide tools research and education in wireless communication. We present two such web applications GeoSpy and CORNET3D.

GeoSpy provides 2D and 3D visualization of geospatial data on the web. The application is primarily focused on leveraging 3D portrayals to increase the number of broadband Quality of Service (QoS) metrics, which can be attached to a single point on a map. Additionally, GeoSpy has proven to be a flexible visualization platform by giving the user a high level of customization over HTTP API data. This allows GeoSpy to venture beyond broadband mapping, and provide 3D portrayals of any well-formatted geospatial JSON API.

Research of Software Defined Radio (SDR) and Dynamic Spectrum Access (DSA) can be used to significantly improve the wireless QoS. CORNET3D provides a 3D view of the Virginia Tech CORNET SDR testbed with information on which nodes and radios are operational. The application can also display 2D and 3D plots of the spectrum, which is sensed by the radios in real time. The data is sent to the client over a WebSocket connection to enable low latency, compared to conventional HTTP. CORNET3D can teach students about strategies for optimal use of spectrum resources through a game—by providing them with real-time scoring based on their choices for radio transmission parameters. CORNET3D has demonstrated that the not only can web applications provide rich portrayals of real-time sensor data, but can also serve as a 3D “serious game” platform.
To my parents, my grandparents and my capybara.
# Table of Contents

1. Introduction ......................................................................................................................... 1

2. Background .......................................................................................................................... 6

   2.1 Mapping Broadband Coverage ......................................................................................... 6

   2.2 Spectral Resources and Dynamic Spectrum Access ......................................................... 7

   2.3 Wireless Communication Systems .................................................................................... 10

   2.4 Multiple Access in Wireless Communication Systems .................................................... 13

      2.4.1 FDMA .......................................................................................................................... 14

      2.4.2 TDMA ........................................................................................................................ 16

      2.4.3 CDMA ....................................................................................................................... 18
5.7 Related Work ................................................................................................. 76

5.7.1 Wireless Testbeds ......................................................................................... 76

5.7.2 Existing Tools for Spectrum Visualization .................................................. 77

5.7.3 3D Visualization on the Web ....................................................................... 80

5.7.4 Gamification .................................................................................................. 81

5.8 CORNET3D Design ....................................................................................... 82

5.8.1 CORNET Infrastructure ............................................................................... 83

5.8.2 HTTP Data Service Implementation ............................................................ 84

5.8.3 Spectrum Sensing ........................................................................................ 85

5.8.4 Client-Side Implementation ......................................................................... 88
5.8.5 GUI Design ............................................................................................................. 88

5.8.6 WebGL and X3DOM ............................................................................................. 89

5.9 Data Visualization .................................................................................................. 90

5.9.1 Testbed Access–Node Plan ................................................................................ 91

5.9.2 Spectrum Visualization ....................................................................................... 91

5.10 Usability Engineering ............................................................................................ 93

5.10.1 Leveraging the Familiar ..................................................................................... 95

5.10.2 Mobile-Device Interaction ............................................................................... 95

5.11 Spectral Games .................................................................................................... 96

5.12 Future Work ........................................................................................................... 99
5.13 Maintaining CORNET3D

5.13.1 Low Layer

5.13.2 Node Status

5.13.3 Spectrum sensing

5.13.4 CRTS

5.13.5 Middle Layer

5.13.6 High Layer

5.14 References

6 Conclusion
List of Figures

Figure 2–1 Frequency allocation of radio spectrum in the United States................................. 8


Figure 2–3 In FDMA different channels are assigned different frequency bands, thus allowing multiple access [Rappaport, T.S. Wireless Communications: Principles and Practice. Upper Saddle River, NJ: Prentice Hall, 2002. Print, Used under fair use, 2014.]................................. 15

Figure 2–4 TDMA timeslots are allocated to users and cyclically repeated [Rappaport, T.S. Wireless Communications: Principles and Practice. Upper Saddle River, NJ: Prentice Hall, 2002. Print, Used under fair use, 2014.].............................................................. 17

Figure 2–6 CDMA, where each channel is assigned a unique PN code, orthogonal to PN codes used by others [Rappaport, T.S. Wireless Communications: Principles and Practice. Upper Saddle River, NJ: Prentice Hall, 2002. Print, Used under fair use, 2014.] .......................................................... 20

Figure 2–7 TDMA Frame allows allocating different numbers of time slots to different users [Rappaport, T.S. Wireless Communications: Principles and Practice. Upper Saddle River, NJ: Prentice Hall, 2002. Print, Used under fair use, 2014.] ............................................................................... 22

Figure 2–8 OSI Model [16]. ........................................................................................................................................... 24

Figure 2–9 A typical 2D portrayal of radio spectrum frequency at one time instance (queried by uhd_fft) ............................................................................................................................... 26

Figure 2–10 GNU Radio Companion flow graph, which was developed for spectrum sensing in CORNET3D ........................................................................................................................................ 28

Figure 3–1 GeoSpy running in Mozilla Firefox with all of the 3D visualization options shown ........................................................................................................................................... 39

Figure 3–2 API visualization options ........................................................................................................................................ 43
Figure 3–3 API Performance.................................................................................................................................................. 45

Figure 3–4 Terrain. ................................................................................................................................................................. 49

Figure 3–5 2D visualization......................................................................................................................................................... 51

Figure 3–6 Broadband on the Road. ........................................................................................................................................... 53

Figure 3–7 DataHawk for Android............................................................................................................................................. 57

Figure 3–8 DataHawk Test Results .......................................................................................................................................... 58

Figure 5–1 CORNET3D Software Architecture....................................................................................................................... 77

Figure 5–2 A typical 2D portrayal of radio spectrum frequency at one time instance (queried by uhd_fft)........................................................................................................................................................................ 79

Figure 5–3 The main view shows spatially-located sources of real time data: clickable radio nodes. Red: offline, olive: online without radio, green: full functionality. Using different shapes on different floors makes distinguishing between floors easier................................................................. 83
Figure 5–4 Real-time spectrum portrayals in CORNET3D. Top: traditional 2D line plot, middle: 3D waterfall, bottom: 2D waterfall

Figure 5–5 Comparison between WebSockets and RESTful HTTP data services [Arun Gupta. 2014. REST vs WebSocket Comparison and Benchmarks. Online. http://blog.arungupta.me/2014/02/rest-vs-websocket-comparison-benchmarks/, Used under fair use, 2014.]

Figure 5–6 Spectrum access game, purple indicated the user-selected frequency band

Figure 5–7 Radio performance metrics interface
Attribution

I would like to thank several colleagues for their help with this project.

Nicholas Polys is an Affiliate Research Professor in the Department of Computer Science and the Director of Visual Computing with the Virginia Tech Advanced Research Computing Group (ARC). Peter Sforza is a Research Scientist and the Director of the Center for Geospatial Information Technology at Virginia Tech. Nicholas Polys and Peter Sforza provided feedback and ideas for the development of the software, discussed in chapter 3. Peter Sforza provided funding. They are also coauthors on the corresponding paper.

Ferdinando Romano is a graduate student in the Electrical and Computer Engineering Department (ECE) at Virginia Tech. Carl Dietrich is a Research Associate Professor in the ECE Department. Vuk Marojevic is a Research Associate in ECE. All three are members of the CORNET (Cognitive Radio Network Testbed) research group. In chapter 4 Ferdinando Romano developed the Cognitive Radio Test System, with which CORNET3D interfaces. Nicholas Polys provided funding. Carl Dietrich, Vuk Marojevic, Peter Sforza, and Ferdinando Romano have all contributed ideas and feedback for the project and are all co-authors on the corresponding paper.
1 Introduction

Digital communication plays a crucial role in driving the information economy and visualization is a key tool for understanding how communication systems operate and finding ways to improve them. The three main challenges facing such visualization solutions: data collection, accessibility, and usability.

It is now possible to outsource data collection to large groups of citizen-scientists, because sensor-equipped mobile devices, such as smartphones and tablets, have become
pervasive. Moreover, modern software-defined radios support robust software interfaces which allow delivering real-time data over the web.

Displaying data in a web application makes it accessible on any browser-enabled platform. Modern browsers support sophisticated presentation techniques, such as 3D rendering with WebGL. However, web applications still support fewer 3D formats than their desktop counterparts, which is an obstacle for rendering existing 3D models on the web. The X3DOM library can provide WebGL rendering for these models because many of them are either in X3D format, or there are existing pipelines for converting them to X3D.

Both data collection and accessibility raise security concerns, which need to be addressed with proper implementation of secure software applications. Security concerns are also critical when dealing with Software Defined Radios because an attacker could be get unauthorized access the radios end even make unauthorized transmissions. Unauthorized transmissions could violate FCC spectrum licensing and jam legitimate traffic.

The problem of usability rises from the high-dimensional nature of network quality data. For example, a single data sample can contain values for upload and download speed, delay, jitter, and location. In the area of Cognitive Radio, the volume and dimensionality of the data presents significant challenges to accessibility and usability: how does one represent spectrum availability for a human analyst? When dealing with numerous such data samples, it becomes challenging to
display so many values at once in a user-friendly way. This problem of high-dimensionality can be approached with 3D visualization, which can display more useful information at once than 2D approaches. The spatial context, which such data often carries, can be addressed by superimposing the visualization onto maps, 3D terrain, and 3D building models.

We developed several applications to allow visualizing data, which pertains to digital communications—GeoSpy, DataHawk, and CORNET3D. GeoSpy is a web platform for API-based geospatial data visualization. GeoSpy primarily focuses on broadband service quality data, but has enough flexibility to be agnostic of the underlying type of data. Thus, GeoSpy can visualize data from arbitrary geospatial APIs, which meet the required format. DataHawk is an Android data collection application—which allows citizen-scientists to report network service quality data from their phones or tablets. Once uploaded, the data becomes available for visualization via an HTTP API. CORNET3D is a platform for visualizing software-defined radio data, such as 3D spatial node location data as well as spectral data, which also spans across three dimensions—time, frequency, and amplitude.

Both GeoSpy and CORNET3D are web applications which can run on almost any platform with a browser, which significantly improves accessibility. However, the applications handle data collection differently. CORNET3D interfaces directly with a software radio testbed over a WebSocket interface. GeoSpy relies on available data services from the FCC and the National Broadband Map. It can also query data collected by custom-built tools, such as DataHawk.
Both applications support various alternative portrayals of data in two and three dimensions. 3D portrayals are intended to improve usability when data is inherently three-dimensional, such as the spectrum, or when numerous values are needed to be packed into a single coordinate, such as with georeferenced broadband service quality.

These data portrayals are meant to allow researchers and engineers to improve wireless communications, by geospatially mapping broadband service quality using GeoSpy and by visualizing dynamic spectrum access using CORNET3D. Additionally, CORNET3D can be used to teach students about spectrum access through spectral games.

Improving wireless communication is a critical task. Demand for communication services is projected to increase due to the increasing number of connected devices. Several trends drive this paradigm. Wireless sensor networks require wireless communication for monitoring various phenomena, such as weather or pollution. Wearable computing devices also require wireless communication capabilities. The President’s Council of Advisors on Science and Technology (PCAST) estimates 5 Billion mobile wireless devices in 2012, 50 Billion in 2020. All of these wireless communication devices need to access the spectrum, which is a precious natural resource. Thus, the demand for spectrum is projected to increase.

Fortunately, spectral scarcity is in large part artificial. It is caused by outdated licensing approaches, where much of spectrum is allocated to individual services and not actually used
much of the time. Thus, it would be a better approach for devices to dynamically share spectrum according to demand. TV Whitespace, which is the unused TV channel frequencies, is particularly suitable for this. In fact, PCAST estimates 1 GHz of shared spectrum to produce over $1 trillion in value.
2 Background

Our society relies heavily on digital communication. Several approaches can be taken to improve wireless and wired communication coverage and quality by both developing new technologies and better utilizing existing ones.

2.1 Mapping Broadband Coverage

Many US broadband customers experience speeds that are slower than claimed by their ISP. In fact, only 30% receive them as advertised [18]. This situation creates a need for a geospatial broadband mapping tool that is accessible to consumers and service providers. Bernardi et al. developed a web application for superimposing broadband data onto a map using GeoExt and OpenLayers [19]. The Virginia Broadband Map currently deploys a similar solution using ArcGIS Flex [20]. However, these approaches are based on two-dimensional maps and are limited in the amount of information they can display at once.
2.2 Spectral Resources and Dynamic Spectrum Access

The modern information economy is largely driven by wireless communications, which rely on a limited natural resource—the radio spectrum. Various parts for the spectrum are strictly allocated to different entities by the FCC, as shows in Figure 2–1. These frequency band owners are banned from transmitting in frequencies for which they do not have a license, because doing so would create interference with other transmissions. This creates spectrum scarcity—better quality of service in wireless communications has to be achieved through more efficient use of the licensed frequency bands.
Figure 2–1 Frequency allocation of radio spectrum in the United States

Cognitive radio (CR) was introduced fifteen years ago as an application of software-defined radio (SDR) [7] and can combat the spectrum scarcity problem through dynamic spectrum access (DSA) [4] [22]. Dynamic spectrum access allows for a more efficient use of a precious natural resource—the electromagnetic spectrum [4]. Since spectrum is not permanently used by licensed or primary users, cognitive radios can make opportunistic use of unused portions of the spectrum in the time and frequency dimensions. Commercial
applications exist, but are limited to a few bands and limited bandwidths (e.g. TV white spaces) and often use a reservation-based database approach.

There are also proposals to allocate spectrum in the 3.5 GHZ band, between 3550-3650 MHz, for small cell use [26]. The FCC first made unlicensed spectrum available in 1985. In the 1990s the FCC allowed unlicensed use of spectrum in the following ISM (industrial, scientific, and medical) bands: 902 to 928 MHz, 2.400 to 2.4835 GHz, 5.725 to 5.875 GHz [27]. These bands are intended for scientific, medical or industrial use [28]. Interestingly microwave ovens and cordless phones fall into ISM bands as well, operating at 2.4 GHz. 802.11 devices also operate in ISM bands. Still, the FCC requires devices to broadcast below a certain power levels, in order to operate in these bands.

Despite 15 years of intensive research on cognitive radio around the world, the big impact that it promises for wireless communications is still not tangible. Several entities need to play together to make this actually happen. Changes can be observed, though, with regulation bodies opening frequency bands for opportunistic use, standardization bodies considering cognitive radio features for new communication systems, and cognitive radio research results making it out of the labs and into the real world. The success of CR and DSA is also a matter of education and belief in this technology, in general. In order to create more awareness and demonstrate the wide applicability and potential of CR and DSA, our research develops tools that allow education and research on dynamic spectrum sharing. We therefore develop a 3D
visualization platform for visualizing spectral resources and the implications of proper and improper management in real time. We demonstrate the suitability of our tools, applying them to Virginia Tech Cognitive Radio Network (CORNET) Testbed and discuss the feasibility of creating game-like applications.

2.3 Wireless Communication Systems

The design of digital communication systems, such as the one in Figure 2–2 is heavily influenced by the properties of the physical channel [25]. The figure displays a transmitter and a receiver communicating over a physical channel. The source output can be either analog, or digital (with a fixed number of possible output characters).
In digital communication systems the source encoder converts the information to be transmitted into a sequence of binary digits, thereby compressing it. The goal at this point is to represent the data with as few digits as possible with no redundancy.
The channel encoder adds redundancy to the message in a controlled manner in order to overcome the negative effects of noise and interference from the physical channel. This improves the fidelity of the received signal and improves the reliability of the transmissions. There are various encoding schemes. The most trivial one simply involves repeating each digit a number of times. Each message can be split up into a number of k-bit sequences. More sophisticated schemes involve mapping each k-bit sequence onto an n-bit sequence. The n-bit sequences are known as code words. Thus n/k measures the amount of redundancy. The ratio n/k is known as the code rate:

\[
\text{amount of redundancy} = \frac{n}{k}
\]

\[
\text{code rate} = \frac{k}{n}
\]

The digital modulator sits at the interface between the channel and the sender. It maps binary strings onto waveforms. Binary modulation involves using only two waveforms—one for bit value 1 and one for bit value 0. Using more waveforms is also possible. Then the number of waveforms is \(M=2^b\), where \(b\) is the number of bits in the sequence, which is mapped to waveforms.

\[
M=2^b
\]
These waveforms are then transmitted and the physical channel will corrupt the signal in some manner.

In the receiver workflow, the demodulator estimates what the symbols should be based on the received channel-corrupted waveforms. The result is passed to the decoder, which attempts to reconstruct the original message from it. Thus the amount of errors in the decoded message acts as a performance indicator of the communication system. This amount of error is a function of channel characteristics, as well as codes and waveforms used.

2.4 Multiple Access in Wireless Communication Systems

Multiple Access techniques can allow multiple users to utilize a finite amount of radio spectrum [30]. Moreover, this can allow one user to both send and receive simultaneously. This is known as duplexing, and it has been achieved in wired communication systems. Thus, making it possible in wireless communication systems would be desired as well. The three most common multiple access techniques are FDMA (frequency division multiple access), TDMA (time division multiple access), and CDMA (code division multiple access). A less common technique is SDMA (space division multiple access).
2.4.1 FDMA

FDMA allows multiple access by assigning different frequency bands to different channels, as shown below. Thus, an FDMA channel can only carry one phone circuit. Unused channels in FDMA cannot be used by others and thus are wasted. FDMA has less communication and computational overhead because the different users do not need to synchronize like in TDMA. FDMA channels are typically narrow, around 30 kHz. Thus, the number of users is limited to the number of channels.
Figure 2–3 In FDMA different channels are assigned different frequency bands, thus allowing multiple access [Rappaport, T.S. Wireless Communications: Principles and Practice. Upper Saddle River, NJ: Prentice Hall, 2002. Print, Used under fair use, 2014.]
2.4.2 TDMA

In TDMA time is divided into time slots and each user can either transmit or receive in them [30]. Thus TDMA transmissions occur in bursts, rather than continuously. This requires the users to synchronize with each other and thus adds more overhead. The time slot, which the user occupied is repeated cyclically. Thus, the number of users is limited to the number of timeslots. The different users use the same carrier frequency.
The advantage of TDMA is that senders and receivers can be turned off when not in use, thus reducing power consumption.

2.4.3 CDMA

More complicated communication flow graphs can involve adding a pseudorandom pattern generator [25]. This technique is used in CDMA (Code Division Multiple Access). This makes
the transmission appear to be noise and makes it difficult to decipher by unintended recipients. This can also prevent jamming by mimicking the signal. The intended sender and receiver have their pseudorandom generators synchronized. Thus, in CDMA systems each channel is assigned a unique pseudorandom noise (PN) code, which is orthogonal to codes used by others, as shown below [30].
Figure 2–6 CDMA, where each channel is assigned a unique PN code, orthogonal to PN codes used by others [Rappaport, T.S. Wireless Communications: Principles and Practice. Upper Saddle River, NJ: Prentice Hall, 2002. Print, Used under fair use, 2014.]
The pseudorandom signal is also known as the spreading signal because it has a very large bandwidth. As a result all users in a CDMA system can transmit simultaneously on the same frequency.

2.4.4 Comparing CDMA with TDMA & FDMA

CDMA offers an advantage over TDMA and FDMA—namely a soft capacity limit [30]. A CDMA system is not limited to a number of users. Rather, as the number of users increases the noise floor rises linearly and the performance of the CDMA system gradually degrades. Channel data rates are also very high in CDMA systems [30]. FDMA has a drawback: unused channels in FDMA cannot be used by others and thus are wasted. FDMA has less communication and computational overhead because the different users do not need to synchronize like in TDMA. Because TDMA transmissions occur in bursts, the advantage of TDMA is that senders and receivers can be turned off when not in use, thus reducing power consumption.
One TDMA Frame

Figure 2–7 TDMA Frame allows allocating different numbers of time slots to different users

TDMA allows providing bandwidth on demand, because different numbers of timeslots can be allocated to different users per each TDMA frame [25], as shown in figure Figure 2–7. The figure also shows the communication overhead, such as the preamble. These timeslots can be reassigned in subsequent frames.
2.5 Spectrum Access Parameters

Software defined radios can be configured with various parameters to access the spectrum. In layered network protocol stacks, such as the OSI (Open Source Interconnection) model, these parameters are often found at the physical and the data link layers.
These parameters can be payload size, gain, delay between transmissions, inner and outer forward error correction schemes, the modulation scheme, and the type of cyclic redundancy check. There is no best modulation scheme and scheme selection has a significant effect on performance, depending on characteristics of the transmission channel [15]. A modulation
scheme should be selected based on QoS requirements. A modulation scheme is the manner in which the combination of characteristics of a sinusoidal carrier are adjusted. These characteristics are frequency, amplitude, and phase. Forward error correction schemes involve adding redundancy to the transmission in order to recover from errors over a noisy channel.

### 2.6 Performance Metrics

Several metrics could be used to gauge performance of a software-defined radio [23]. These metrics involve checking whether the header and the payload of a packet is valid. It is also possible to measure the error vector magnitude (EVM) or the received signal strength indication (RSSI). EVM is a measure of modulation quality. RSSI is the measure of the amount of power in the received signal. Moreover, it is possible to measure the packet error rate, the payload byte errors, and the payload bit errors. The Cognitive Radio Test System (CRTS) system was developed at Virginia Tech to provide various performance metrics for cognitive radio [24].

### 2.7 Software Radio Testbeds

Wireless testbeds play a major role in wireless communication research and education. Other universities and state or private institutions have developed different kinds of testbeds for R&D as well as education [10]. CORNET is used in undergraduate and graduate level courses
taught at Virginia Tech and in many internal and external research projects [21]. It is fully SDR capable, using Ettus Research USRP2 frontends and powerful servers for implementing any waveform in software. It features 48 SDR nodes located on a four story building at Virginia Tech main campus. CORNET is remotely accessible for free and can accommodate several users simultaneously [13]. The tools developed here can be ported to other testbeds due to the modular architecture of CORNET3D, Figure 1.

![Figure 1](image_url)

**Figure 2–9** A typical 2D portrayal of radio spectrum frequency at one time instance (queried by uhd\_fft)
2.8 Tools for SDR Research

A common tool for spectrum visualization is uhd_fft, which is bundled with GNU radio, Figure 3. GNU Radio is an open source SDR software development toolkit that can run on desktop platforms, such as Windows, Mac OS X, and Linux. uhd_fft can display line charts to visualize the frequency, as shown in Figure 3. Moreover, GNU Radio [17] Companion allows creating custom algorithms for SDR, using block diagrams, as shown in Figure 4.
Figure 2–10 GNU Radio Companion flow graph, which was developed for spectrum sensing in CORNET3D

2.9 References


3 GeoSpy: a Web3D Platform for Geospatial Visualization

Nikita Sharakhov  Nicholas Polys  Peter Sforza
Virginia Tech  Virginia Tech  Virginia Tech
Blacksburg, VA 24061-0002  Blacksburg, VA 24061-0002  Blacksburg, VA 24061-0002

3.1 Abstract

Broadband networks are key utilities in driving the information economy. In the United States multiple publicly available web APIs provide broadband service quality data by geographic location. However, views into this data have usually been two-dimensional and limited in the amount of information that they can portray at once. In this paper, we detail a location-aware Web3D application for visualization of high-dimensional georeferenced data, such as broadband quality information, on desktop and mobile devices. The application presents a wide variety of two- and three-dimensional visualization methods and uses HTML5 for novel user interaction via access to onboard location and orientation sensors on equipped devices. Considering that the application uses geospatial web APIs as a data
source, we describe its applicability in domains beyond broadband. GeoSpy is hosted at http://filebox.vt.edu/users/sharni/gs

Original publication

Categories and Subject Descriptors

I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual Reality; I.3.6 [Methodology and Techniques]: Standards—Languages; H.5.1 [Information Systems]: Information Interfaces and Presentation—Multimedia information system

Keywords

3.2 Introduction

Many US broadband customers experience speeds that are slower than claimed by their ISP. In fact, only 30% receive them as advertised [3]. This situation creates a need for a geospatial broadband mapping tool that is accessible to consumers and service providers. Bernardi et al. developed a web application for superimposing broadband data onto a map using GeoExt and OpenLayers [2]. The Virginia Broadband Map currently deploys a similar solution using ArcGIS Flex [12]. However, these approaches are based on two-dimensional maps and are limited in the amount of information they can display at once.

As a solution, we developed a web application—GeoSpy— that visualizes data in 2D and 3D across a map. This application builds on our previous work [9]. Because the application is engineered to receive the data from geospatial web APIs, the visualization domain can be easily changed to something other than broadband, by
changing which API is used. Using the rich sensor and data capabilities of HTML5, such as geo-location services, access to on-board orientation sensors, and dynamic web queries, we provide a novel interface to view and interact with this high-dimensional geo-referenced data. The application is compatible with both mobile and desktop browsers, and on mobile devices we create an embodied user interface, by allowing the user to navigate using the on-board gyroscope.

3.3 Related Work

3D visualization of geo-referenced data has a long history and continues to evolve. In this paper, we focus on those that employ open standards and open source examples. For example, the 3D Portayal Interoperability Experiment, run recently by the Open Geospatial Consortium, demonstrated several client and server technologies interoperating to serve X3D, X3DOM and Collada/KML through its Web3D Service [8]. This cooperative effort brings together several international consortia and is evolving into a full-fledged standardization effort.

Delivering 3D content to end-users without requiring proprietary technology or plugins led Tilden et al. to choose a WebGL framework, X3DOM, for its openness and its ability to display 3D content in a web browser without a plugin [11]. Indeed, the
'mashability' of X3DOM is very flexible—X3DOM seamlessly integrates into a webpage by extending the existing DOM tree [1]. It uses tags to define a 3D scene in the same manner as HTML defines a webpage, and can therefore be manipulated via standard JavaScript and CSS operations. This makes it easier to adopt for web developers and suitable for web mashups, and in particular for integrating data from various geospatial, map, and mobile sensor APIs. Tilden et al. developed an application where the user could interact with the X3DOM 3D scene through the controls on an HTML page.

X3DOM can be contrasted with other WebGL frameworks, such as SpiderGL and XML3D. Both XML3D and X3DOM aim to be web developer friendly by extending the DOM and supporting DOM scripting [1], [10]. SpiderGL, on the other hand, is not as web developer friendly, because it does not extend the DOM, and according to Behr et al., it requires the web developer to understand new APIs and graphics concepts [1]. Unlike other WebGL frameworks, X3DOM is based on X3D. Because we come from an X3D background, this factor is an advantage for us.
Figure 3–1 GeoSpy running in Mozilla Firefox with all of the 3D visualization options shown

3.4 User Interaction

An advantage of web applications is their ability to work across various platforms without requiring users to install them or their updates. We are targeting desktop operating systems, such as Windows, OS X, and Linux, as well as mobile, such as Android and iOS. We have tested the application on Windows and Android. This
project attempts to bridge the gap between desktop and mobile platforms. Although the application can run in both desktop and mobile browsers, there are tradeoffs. On the desktop, we have better rendering performance, but we lose some inputs that come from sensors on the mobile device, such as GPS and gyroscope. The two main components of our GUI are the map window and the 3D scene window, as can be seen in Figure 1. The user can navigate around the virtual world by manipulating the 3D scene as well as around the real world by changing the location and map scale. This creates an augmented virtuality environment [5].

3.5 Geolocation

The location and map scale are stored as global JavaScript variables and represent internal state of the application. A map is embedded into the application via the Google Maps JavaScript API, with the current location shown by a marker. The API also allows the application to respond when the user interacts with the map. The scale can be changed by zooming in and out on the map. The location can be set by dragging the map around or via the HTML5 Geolocation API. The API is actually agnostic to the source of the location information; however, the accuracy value gives a good idea of how the location was determined, which is API-reported to be between 20 and 200 meters for internet-based geolocation and around 5 meters for
devices with GPS. The location can also be typed in as latitude and longitude or geocoded using the built-in geocoder. When either location or scale is changed, the 3D visualization can be updated.

### 3.6 Data Services

Numerous web APIs provide georeferenced data. Most of them take a geospatial coordinate as input and return data for that coordinate in JSON, JSONP, or XML format. We developed GeoSpy to visualize this data in 2D and 3D. Before sending requests to a geospatial API, GeoSpy calculates a grid of coordinates based on two internal states of the application: geographic location and map scale. The grid is centered on the location and the spacing between grid coordinates is determined by the scale. Then the application sends a request to an API to get data for each of the grid coordinates in a JSONP format.

We chose JSONP because it is a way to circumvent the same origin policy that browsers have and to allow crossdomain communication. The application uses the lightweight jQuery library which does the necessary computation to receive multiple JSONP responses asynchronously. This is faster than sending the next response after
the previous one has been received. However, the JSONP method cannot handle HTTP status codes, and can only handle errors by timing out requests.

By default the application uses the Federal Communications Commission (FCC) Consumer Broadband Test API, which returns 10 broadband data values for the corresponding coordinate [4]. These values are maximum and average upload and download speeds for wired and wireless broadband, as well as the number of wired and wireless tests executed to measure the speeds. Thus we can visualize upload and download speed data across a map to provide end users a view of broadband service quality in a various geographic areas. The user can also select other broadband APIs to visualize, such as the Wireless Broadband, Wireline Broadband, or Community Anchor Points API, [6]. In domains beyond broadband, the user can select the Census or the Demographics API [6]. We also give the user the option to input an API request string, as show in Figure 2, to use an API that we have not included with the application.
Figure 3–2 API visualization options
It is important to note that the application also deals with APIs of a different structure. For example, the Google Elevation API allows inputting multiple coordinates in one request, which gives a performance boost over inputting one coordinate per request. Also, the OpenSignal NetworkRank API returns average broadband data values for a specified area rather than for individual coordinates [7]. We have also developed an API that returns the same data as the FCC API but allows multiple coordinates as input. The application also utilizes APIs that return map overlays as semitransparent images.

### 3.6.1 Geospatial API Performance

We tested the performance of the FCC API to see how fast it can deliver broadband data to our application. A test bed application was developed, which recorded the time needed to complete sets of requests of sizes from 10 to 500 coordinates. Each set was repeated 10 times and the results were averaged. Figure 3 shows a linear relationship between the number of requests and the time needed to receive the data. It also shows that as the number of requests gets high the lag will hurt the user experience.
This problem led us to develop an API, using PHP and PostgreSQL, that would deliver the same data, but do so in one call for a group of coordinates rather than for single one. This API also supports the trivial case where the group only contains one coordinate. Additionally, our API allows the user to specify which fields to return, in case some
are not needed, to further speed up the performance by reducing the amount of data transmitted. We performed two additional tests with this API: one where we grouped all of the coordinates together and one where we submitted them one by one. Figure 3 shows the difference between the three tests. Our API performed around 10 times faster on average when the coordinates were grouped. This shows that if geospatial API developers intend their APIs to be used to deliver data for large sets of coordinates, then it is critical for performance to allow processing these coordinates in a single API call. Allowing the user to select which fields to deliver would give an additional performance boost. Most of the geospatial APIs that we encountered on the web lack these features.

3.7 Information-Rich Environment

3.7.1 3D Visualization

Once the API data is received, we build our visual representation with X3DOM. Because X3DOM simply extends the existing DOM, the visualization is done via standard JavaScript DOM manipulations, such as adding X3DOM elements to the DOM and changing element attributes. 3D visualization allows packing 10 values into a single geographic coordinate without compromising usability. Users can control a number of visualization parameters through familiar HTML and jQuery UI elements on the page,
resulting in an integrated information-rich visualization environment. 3D visualization is done using two methods, shown in Figure 1: 3D markers and an elevation grid, which can be textured with a heat map as an additional visualization option. The markers and elevation grid points are placed above a texture of a map according to their geographic coordinates. The map texture is delivered via the Google Static Maps API as an image, based on user-defined location and scale values. Data values can be visualized using the heights and colors of a cylinder and a rectangular prism, which make up a 3D marker. The cylinders and prisms also have collars on them, which allow for additional data visualization through their heights and colors. There is also an option to scale the marker heights to the square root of their value. Additionally we provided a slider for linearly scaling the markers and the elevation grid. Another slider can be used to adjust the transparency of the elevation grid between invisible and fully opaque. For coordinates where the application does not receive any data from an API the corresponding marker or grid point is colored black.

By default, the cylinder height shows the maximum wireless speed and the height of the rectangular prism shows the maximum wired speed from the FCC API. The color of each shape represents the number of wireless or wired tests performed, which shows the degree of confidence in the measurement. The collar heights denote the average speeds. The user can also slide a semi-transparent measurement plane up and down, showing the exact speed value above the slider itself. Because wired speeds are often
much greater than wireless speeds, the option to scale markers to the square root of their values can be useful.

The application provides another georeferenced visualization method—a 2D graph that cuts across the map and plots georeferenced speed data versus the line of its intersection, as seen in Figure 1. It can also be rotated to visualize data for different transections. The graph is made by using an HTML5 Canvas element as a texture on an X3DOM object. The plotting is done using the RGraph library.
Figure 3–4 Terrain.
Satellite imagery is provided by Google [Google Maps Image API. Online. https://developers.google.com/maps/documentation/staticmaps/, Used under fair use, 2014.]

Additionally the data from the Google Elevation API can be loaded to display a terrain mesh. This mesh can be textured with either a map or satellite imagery (for realism) from the Google Maps Static API, as seen in Figure 4.
3.7.2 2D Visualization

GeoSpy supports 2D visualization via map overlays, as seen in Figure 5. The simplest type is an image overlay, which places a semi-transparent image over the map view.
GeoSpy gives the user a number of broadband-related image overlays to choose from. The application adjusts this image by sending new requests to the image API, as the user pans and zooms the map.
2D visualization is also accomplished by placing circular markers on the map. These markers visualize the data by having their color mapped to a field in the JSON object, which the API returns. It is currently set to the wireless average download speed from the FCC API. The markers are colored in the typical heat map fashion: by varying the hue from blue (minimum) to green to red (maximum). The marker will be colored black if no data is available. Hovering the mouse over a marker will display a tooltip, showing the speed value of the marker. The Google Maps JavaScript API also supports declaring the shapes of markers in SVG, which can give us more visualization options in the future.

The user can place these markers by clicking at a point on the map. Alternatively, they can be placed along user-defined paths on the map, as seen in Figure 5. The density of the markers is also user-defined. These paths can be drawn using the Google Drawing Library. Alternatively, they can be generated as navigation routes along two or more destinations using the Google Directions JavaScript Service. As the user types these destinations, the built-in geocoder will suggest autocompletion options for US locations. Once rendered, the path can also be dragged around and modified. This feature can be used for estimating the mobile broadband speed and coverage for a particular trip before getting on the road. Moreover a separate application, Broadband on the Road, was created to focus on this feature, as shown in Figure 6.
Unlike other APIs, the Open Signal Network Rank API returns average data values by taking an area as input. To visualize data from this API the user must draw circles on the map, thus specifying the area to input to the API. Once the application receives the data, it finds the network with the highest RSSI, prints the network name and RSSI value in the circle, and colors the circle according to how high this value is. Circles
can also be selected by clicking to show more data for more networks, which is displayed in a table under the OPEN SIGNAL tab on the left side, along with the location and radius of the selected circle, as seen in Figure 5.

### 3.7.3 Custom Geospatial API Visualization

GeoSpy allows the user to select which geospatial API to use as a data source. The user can choose from several built-in APIs or can type in a URL for another geospatial API, as long as it takes latitude and longitude as input. Next, the application calls this API once and displays all of the fields in the returned JSON object. The application lets the user bind the fields one by one to any of the aforementioned 10 3D visualization indicators, such as elevation grid point height, cylinder color, etc, as seen in Figure 2. The user can also enter a value by which to scale the JSON fields. The application will show any already existing binding, which will be the default bindings if the application was just started. GeoSpy also keeps track of which of these indicators are already taken and lets the user unbind them if necessary.

As an extra step the user can type custom mathematical expressions involving the JSON fields. This expression can be assigned to an indicator, just like a regular JSON
field. The application checks that the expression can be evaluated before it is bound. Once all of the necessary assignments have been made the application can start visualizing data for this API across a grid of geographic coordinates, which are computed in the same manner as before. The custom expressions, if present, will be evaluated when each of the API calls returns.

To illustrate the usefulness of user defined expressions, let us consider a simple example with the built-in demographics API. The API delivers the percentage of population with income below 25 thousand dollars, the percentage with income between 25 and 50, and a some other fields. If the user wanted the application to calculate and visualize the percentage of population with income below 50, he could input a custom expression that adds the two aforementioned fields, as show in Figure 2. For instance, this could be useful for studying the relationship between income and broadband speeds. Custom expressions can also be used to access data that is not explicitly included in the JSON tree, such as the length (number of fields) of the object. For instance, the expression data.Results.wirelessServices.length will give the number of wireless providers. We are using the JavaScript eval() method to evaluate the expressions. This method has raised some security concerns in the JavaScript community. Upon investigation, we discovered that a client-side eval() does not create any additional security vulnerabilities because the user can already type in and evaluate JavaScript code using a browser console.
3.8 Crowdsourcing Data Collection

Modern sensor-equipped mobile devices, such as smartphones and tablets, have become pervasive, making it possible to crowdsource data collection tasks to large groups of citizen scientists. We have developed DataHawk—an Android data collection application, based on the CalSPEED source code, which allows citizen-scientists to report network service quality data from their phones or tablets. DataHawk is shown in Figure 7. DataHawk connects to a server, which is running iPerf, which allows measuring four quality of service metrics: upload speed, download speed, latency, and jitter. Subsequently, DataHawk taps into the onboard GPS on smart devices and adds a georeference (latitude and longitude) to the data. The georeferenced measurements are sent to a database, and made accessible via an HTTP data service. Thus, the data can later be visualized in a web application by placing the data points onto a map, based on their coordinate-references.
Figure 3–7 DataHawk for Android
Figure 3–8 DataHawk Test Results
3.9 Conclusion

Through GeoSpy, we have demonstrated an integrated, information rich 3D geospatial visualization over the web. Using the native browser environment of HTML5 and X3DOM, we can meet our requirements to deploy this application to desktop or mobile clients. We have identified the impacts of data request latency as a key concern for the performance of the application. We have shown the need for geospatial API developers to allow the APIs to return data for multiple coordinates in a single call and to allow the user to select which data fields to deliver.

3.10 Future Work

Based on these initial results, we will continue the development and evaluation of GeoSpy. Having developed both 2D and 3D visualization capabilities, we plan on comparing them for effectiveness across various domains by building use cases and assessing the end-user usability. Additionally we are interested in further integration of geospatial data (e.g.: tower locations and properties, 3D terrain), augmented reality technology, and the use of onboard sensors for novel 3D user interfaces. Specifically, we plan to develop functionality for virtual broadband tower citing because this procedure would be much less costly than physical citing. Virtual citing would allow stakeholders
to view how the towers mix with 3D terrain, buildings, and vegetation, which would be loaded into the application. The Google Elevation API is currently used as a proof-of-concept. However, it lacks accuracy and data about trees and buildings. Due to the integration of different types of data, future work must include a strategy for web standards development and integration. As the 3D rendering capacity of mobile web-enabled hardware continues to improve, we expect to see more Web3D applications and frameworks providing access to data for informed citizens and consumers.
3.11 References


4 Securing Data on the Web

4.1 Introduction

Data-driven web applications rely on the security of the underlying data. A visualization approach will not be effective if the integrity or the availability of the data, which drives it, is compromised. Thus, securing the databases, which make the data available, is paramount.

Several recent technological advances have caused databases to become richer and also caused more users to interact with them [2]. Additionally, more people are now on the internet due to several factors, including the increased number of mobile internet-connected devices, and the wider availability of internet connectivity, particularly broadband. Moreover, hacking has become an enterprise [2]. There is even a market for selling vulnerabilities in popular software, including web browsers.

4.2 SQL Injection Attacks

A common attack on a database is a SQL injection attack. A SQL injection attack (SQLIA) is a type of a code injection, where an attacker inserts (injects) malicious SQL code into a query, which is sent to the database, thus changing the original intent of the query [5]. Such queries are
able to bypass authentication and authorization controls can even execute commands at the operating system level [6]. This is often done by entering malicious input into forms, which are exposed on the web and are used to send data to a database. Input validation can protect against SQL injections [6]. There are several types of SQL injection attacks, which will be discussed in subsequent sections.

4.2.1 First Order SQL Injection Attacks

There are several ways in which the malicious code can actually be injected. Code can be injected into user input, which is typically sent to the database via HTTP GET and POST requests [5]. Cookies are used by web applications to store state information on the client-side. Thus, if this information is used to construct SQL queries by the web-application, a malicious client can tamper with cookies and inject malicious SQL commands into them. Some web applications store information from HTTP headers in a database, with the intent of monitoring usage statistics and browsing trends. Therefore, an attacker can insert code into these headers.

4.2.2 Second Order SQL Injection Attacks

Second order SQLIAs start by inserting values into a database, which are not necessarily malicious by themselves [5]. However, these values are later used when a different query is sent,
such that this combination produces a malicious effect. Because there are multiple stages, second order SQLIAs can be more difficult to detect and prevent than first order ones.

4.2.3 Attacker Intent

There are various reasons for performing a SQLIA. Some attacks simply aim to gather information, with which to launch subsequent attacks. For example they aim to identify, which application parameters and user input fields are vulnerable to SQL injections. Other attacks, known as database finger printing attacks, aim to determine the type and version of the database, which can aid the attacker in launching future attacks. Additionally these attacks can aim to determine the database schema, such as table and column names or column data types. This can aid the attacker in correctly extracting data form the database.

The most common type of SQLIAs aims to extract data from the database. Also attacks can aim to modify data. An attacker can store malicious commands in a database and make them available for users to execute. Some attacks aim to lock or drop database tables which results in denial of service. Other attacks aim to bypass authentication and gain the privileges of another user, or to escalate the privileges which the attacker already has. Finally, some attacks aim to prevent detection and auditing.
4.3 Database Security Mechanisms

4.3.1 Access Control

Access control is a mechanism for ensuring database confidentiality and integrity [4]. Authentication is one component of access control, which verifies that an individual (or an entity) is whom he claims to be [7]. Additionally, authentication also allows creating an audit trail—a list of changes made by specific users [8]. The presence this record could discourage users from maliciously tampering with data. Authorization is another component of access control, which applies a set of rules determine what type of actions a user can perform on certain data fields in the database [8].

4.3.2 Privilege and Data Separation

Authorization can be given through permissions. Typically a database will have an owner or a superuser [6]. This user can grant privileges to other users by specifying which queries they are allowed to run. Each user should not have any more privileges than what is necessary to do the intended tasks [9]. This is known as the least privilege principle. Given the fact that access credentials can be compromised, reducing the level of permissions, which a user is granted can improve database security [6]. Additionally, the passwords for different users of the database should be hashed rather than stored as plain text [10]. This system of permissions raises
implication for designing applications, which access the database—allowing applications to connect to a database as a superuser is least secure. It is also a good practice to only give the root user access to the table in the database, which contains user accounts [10].

Often applications are designed with every module having complete access to the database [9]. This introduces a major security vulnerability: a flaw in one module can expose data, which the module was not originally intended to access. Privilege separation dictates that privileges be separated between application parts, with each part receiving just enough privileges to carry out the intended functions and no more. Privilege separation can applied to a database by splitting the application, which connect to it, into modules and applying the least privilege principle to each module: code sections should only have access to the data, which they need. This principle is known as data separation. It protects against bugs in SQL queries, as well as SQL injection attacks by limiting the privileges of access credentials, which could be compromised. This principle also facilitates code review by making access privileges of code sections explicit to software developers.

Using these principles, Felt et al. implemented data separation in their system, named Diesel [9]. Diesel intercepts database queries and applies module-level restrictions to them. Felt et al. have retrofitted Diesel onto website content management systems: Drupal and WordPress. Diesel is intended to limit the privileges of plugins on these platforms, because plugins generally have a less stringent security review process than the core platform. Felt et al. recommend that the
platforms require plugins to declare their required database privileges in a policy file. This security measure should reduce the impact of a security vulnerability present in a plugin. Additionally, the plugins often only need to access the tables, which they create. Diesel is a light-weight feature: implementing it for Drupal or WordPress required less than 50 lines of code.

The Brilliant Gallery plugin for Drupal has a flaw [9, 11]. Combined with a lack of data separation this flaw grows into a SQL injection vulnerability, which allows an attacker to retrieve an administrative password. Since the plug in does not need to have administrative privileges, it can benefit from data separation. Thus, Felt et al. have refactored the plugin to use Diesel. The refactored plugin only has access to the table, which it creates. Therefore, the ramifications of the same SQL injection vulnerability are greatly reduced because the attacker can only capture the credentials to access the table, which was created by the plugin. The authors have also made a similar implementation for a WordPress plugin.

4.3.3 Business Logic in the Database

Permissions of an application can be limited by moving the business logic from the application to the database by using views, triggers, and rules. A view is simply a virtual table, which results from a database query. For example, permissions can be given on a view rather than on an entire database or an entire table, which limits the amount of data which be viewed and modified in the
event, the application is compromised. A trigger is piece of procedural code, which is executed in response to certain events in a database. Utilizing triggers allows computation to occur in the database, rather than in the application.

**4.4 Design Tradeoffs**

Several tradeoffs appear in designing secure databases. For example, a tradeoff exists between security and usability: database encryption limits the types of queries, which a user can run.

There will always be a tradeoff between security of an application and the ease of its implementation. Even the database security measures which are relatively simple to implement, will still take time to program, which some developers are simply not willing to spend, thus creating security vulnerabilities. A good example of this is the Microsoft Windows platform in the 1990s, which tried to attract application developers with its lenient security policies [26]. This made software development easier on the Windows platform and attracted many third-party application developers. The abundance of software packages for the Windows platform, greatly increased its market share. Although this assured the commercial success of Windows, it also made the platform vulnerable from a security standpoint. The same principle applies to database security. Thus, database security measures must not only be effective but also easy to implement. Diesel presents a good compromise between security and implementation overhead,
by only asking Drupal or WordPress plugin developers to declare the privileges, which their software needs in policy file. This procedure is quick to implement and protects administrative database privileges from being compromised due to bugs in implementation of plugins, which do not need the administrative privileges to begin with.

Implementation issues aside, there are additional tradeoffs between security and efficiency—encryption and decryption procedures add computational overhead, thus reducing the performance of an application, which may have a negative effect on end-user satisfaction.

There are also tradeoffs between security and monetary cost. The cost tradeoff also factors in the implementation and performance tradeoffs. Costs increase from a need to hire more developers to implement the security mechanisms. Costs can also increase from the need to purchase more computing resources to compensate for the cryptographic overhead. Nevertheless, costs can be reduced by utilizing cloud computing solutions instead of purchasing in-house hardware. However, these outsourced services add their own cloud-computing-specific security vulnerabilities, which were discussed in this paper.
5 Visualizing Real-Time Radio Spectrum Access with CORNET3D

Nikita Sharakhov*, Vuk Marojevic†, Ferdinando Romano‡, Nicholas Polys§, Carl Dietrich¶

Virginia Tech

5.1 Abstract

Modern web technology enables the 3D portrayal of real-time data. WebSocket connections provide data over the web without the time-consuming overhead of HTTP requests. The server-side “push” paradigm is particularly useful for creating novel tools such as CORNET3D, where real-time 3D visualization is required. CORNET3D is an innovative Web3D interface to a research and education test bed for Dynamic Spectrum Access (DSA). Our system can drive several 2D and 3D portrayals of spectral data and radio performance metrics from a live, online system. CORNET3D can further integrate the data portrayals into a multi-user “serious game” to teach students about strategies for the optimal use of spectrum resources by providing them with real-time scoring based on their choices of radio transmission parameters. This paper describes the web service architecture and Webd3D front end for our DSA testbed, detailing new methods for spectrum visualization and the applications they enable.
5.2 Original Publication


5.3 Categories and Subject Descriptors

I.3.6 [Computer Graphics]: Methodology and Techniques—Ergonomics
I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual reality
I.3.8 [Computer Graphics]: Applications

5.4 Keywords


5.5 Contact

*e-mail: sharni@vt.edu
Recent developments in web technology, such as WebGL and WebSockets, allow low-latency 3D portrayals of real-time data natively in web browsers. Moreover, 3D views can serve as a natural interface to spatially located data sources. These features are highly suitable for cognitive radio research and education, where currently 3D portrayals and web exposure are lacking. In particular, the radios themselves play the role of the spatially located data sources because they are typically installed in various locations inside buildings or outdoors and provide real time data, such as spectral power distribution or transmission performance metrics.
Cognitive Radio (CR) was introduced 15 years ago as an application of Software-Defined Radio (SDR) [Mitola and Maguire 1999] and can combat the spectrum scarcity problem through Dynamic Spectrum Access (DSA) [Haykin 2005]. Dynamic spectrum access allows for a more efficient use of a precious natural resource— the electromagnetic spectrum [Haykin 2005]. Since spectrum is not permanently used by licensed or primary users, cognitive radios can make opportunistic use of unused portions of the spectrum in the time and frequency dimensions. Commercial applications exist, but are limited to a few bands and limited bandwidths (e.g. TV white spaces) and often use a reservation-based database approach.

Despite 15 years of intensive research on cognitive radio around the world, the big impact that it promises for wireless communications is still not tangible. Several entities need to play together to make this actually happen. Changes can be observed, though, with regulatory bodies opening frequency bands for opportunistic use, standardization bodies considering cognitive radio features for new communication systems and cognitive radio research results making it out of the labs and into the real world. The success of CR and DSA is also a matter of education and belief in this technology, in general. In order to create more awareness and demonstrate the wide applicability and potential of CR and DSA, our research develops tools that enable education and research on dynamic spectrum sharing. We have developed a 3D visualization platform for visualizing spectral resources and the results of proper and improper spectrum management in
real time. We demonstrate the suitability of our tools, applying them to Virginia Tech Cognitive Radio Network (CORNET) Testbed and discuss the potential of creating game-like educational applications.

5.7 Related Work

5.7.1 Wireless Testbeds

Wireless testbeds play a major role in wireless communication research and education. Other universities and state or private institutions have developed different kinds of testbeds for R&D as well as education [The Networking and Information Technology Research and Development (NITRD) Program 2014]. CORNET is used in undergraduate and graduate level Electrical and Computer Engineering curricula taught at Virginia Tech and in many internal and external research projects. It is fully SDR capable, using Ettus Research USRP2 front ends and powerful servers for implementing any waveform in software. It features 48 SDR nodes located on a four-story building at Virginia Tech main campus. CORNET is remotely accessible for free and can accommodate several users simultaneously [Virginia Tech 2014]. The tools developed here can be ported to other testbeds due to the modular architecture of CORNET3D, Figure 1.
5.7.2 Existing Tools for Spectrum Visualization

A common tool for visualizing spectrum in 2D line charts is uhd fft, which is bundled with GNU radio, Figure 2. GNU Radio is an open source SDR software development toolkit that can run on desktop platforms, such as Windows, Mac OS X, and Linux. CORNET 3D offers several advantages over uhd fft in terms of how the data is portrayed. Both applications offer 2D line charts, but CORNET 3D can also simultaneously show 2D and 3D spectral waterfall plots, which
visualize past and present data on the same screen. Additionally, CORNET3D benefits from being able to execute inside of a web browser.

SDRs have a limit on the width of the spectrum that they can sense. Unlike uhd fft, CORNET3D allows to exceed that limit by both retuning the receiver and aggregating data from multiple receivers. Retuning the receiver involves scanning a band and then retuning to scan an adjacent frequency band. Afterwards the bands are concatenated to cover a wider spectrum. This creates a coverage-accuracy tradeoff, because the scans for the different bands will be made at slightly different times. Spectral data can also be integrated from several transmitters, which cover adjacent frequency bands. This again increases coverage, but this time at the expense of spatial accuracy. The two methods can also be combined.
Figure 5-2 A typical 2D portrayal of radio spectrum frequency at one time instance (queried by uhd_fft)
5.7.3 3D Visualization on the Web

CORNET3D delivers both 3D spatial node location data, and spectrum measurements at all nodes, which span across three dimensions—frequency, time, and amplitude. This led us to examine various approaches of 3D visualization on the web. Delivering 3D content to end-users without requiring proprietary technology or plugins led Tilden et al. to choose a WebGL framework, X3DOM, because of its openness and its ability to display 3D content in a web browser without a plugin [2011], [Behr et al. 2009]. Indeed, the “mashability” of X3DOM is very flexible—X3DOM seamlessly integrates into a webpage by extending the existing DOM tree [Behr et al. 2011]. It uses tags to define a 3D scene in the same manner as HTML defines a webpage and can therefore be manipulated via standard JavaScript and CSS operations. Tilden et al. developed an application where the user could interact with the X3DOM 3D scene through the controls on an HTML page.

X3DOM can be contrasted with other WebGL frameworks, such as SpiderGL and XML3D [Di Benedetto et al. 2010]. Both XML3D and X3DOM aim to be web developer friendly by extending the DOM and supporting DOM scripting [Behr et al. 2011], [Sons et al. 2010]. SpiderGL, on the other hand, is not as web developer friendly, because it does not extend the DOM, and according to Behr et al., it requires the web developer to understand new APIs and graphics concepts [2011]. Unlike other WebGL frameworks, X3DOM is based on the standard
X3D (Extensible 3D) scenegraph, which bridges the gap between web developers and 3D application developers with a declarative content model.

### 5.7.4 Gamification

Gamification (not to be confused with game theory) generally refers to adding game elements to non-game applications to increase user engagement. A good example of this is the site Stackoverflow, which awards users various badges to motivate them to contribute to the discussions on the site. Gamification has been successfully used in several research projects. Generally gamified academic applications aim to achieve one or both of the following goals: education (of the players) and data analytics (players analyze data) [2013]. Vara et al. developed MECCO to educate players about an eco-friendly lifestyle [2011]. Ahn et al. developed a game where players label images, thus producing useful data [2004]. Khatib et al. created Foldit, where players create accurate protein structure models [2011]. Seung et al. developed EyeWire, where players identify neurons in a large collection of images [2013]. For motivation, the EyeWire players are rewarded with points and rankings.

The majority of people in the world rely on wireless communication in some form. However, they often lack the understanding of basic concepts of how wireless communication function. Therefore, we considered the gamification of the CORNET testbed to motivate people to learn the principles of spectrum access.
5.8 CORNET3D Design

Several factors led us to develop CORNET3D as a web application. Let us first consider the myth that web applications are severely limited in comparison to their desktop counterparts. Several factors actually allow web applications to match and even surpass the value proposition of native software platforms. First, end-users do not need to install web applications or worry about managing updates. Second, web applications are highly cross-platform and run on a multitude of desktop and mobile platforms. Third, recent advances in the capabilities of web-browser APIs have allowed web applications to rival traditional desktop application in functionality. Fourth, the emergence of cloud computing allows web applications to be inexpensively deployed and scaled. By deploying CORNET3D as a native web application we avoid the complications of platform-specific compilation and installation. We believe that this will improve the accessibility and adoption of SDR, CR, and DSA research, development and education.

CORNET3D has a modular architecture with clear separation across the HTTP interface of cognitive radio data services on the backend and data visualization of the frontend, as shown in Figure
1. This allows developers to connect the CORNET3D frontend to HTTP and WebSocket data services. Thus, our visualization platform can be easily ported to other backends, systems and testbed architectures.

![Image of CORNET testbed](image)

*Figure 5–3 The main view shows spatially-located sources of real time data: clickable radio nodes. Red: offline, olive: online without radio, green: full functionality. Using different shapes on different floors makes distinguishing between floors easier.*

### 5.8.1 CORNET Infrastructure

The CORNET testbed consist of a set of second-generation USRP (Universal Software Radio Peripheral) devices. Each USRP2 is a software-defined radio (SDR) front end, and can be
controlled through software APIs, such as GNU Radio and liquid-dsp. Twelve of these devices are installed on each of the four floors of Kelly Hall, a building on the Virginia Tech main campus in Blacksburg, VA. Each USRP2 is connected to an Ubuntu Linux machine, which we refer to as a computing node. An SDR needs a computing and radio node to implement the signal processing and transmit and receive signals over the air. The computing nodes can be accessed from the internet via SSH, allowing the users to control the radios. The Ubuntu machines have wired connectivity with a web server, which allows delivering data on the web from these software defined radios over HTTP in JSON format, among others. The web server runs commands to gather data from the radios over SSH in Node.js scripts, which are exposed to the web and can be used as a data service by web applications. The CORNET testbed can be used for a variety of SDR and cognitive radio research, such as developing strategies to optimally utilize the limited spectral resources for sending packets from one radio to another.

### 5.8.2 HTTP Data Service Implementation

We developed several web based data services for CORNET3D in the form of HTTP APIs and WebSocket streams. One of them delivers data on the status of the nodes. The HTTP API provides information of whether the node can be accessed over SSH. If so, the API also indicates whether the radio transmitter is operational.
Using the GNU Radio Companion SDR application development tool, we developed a spectrum sensing Python script that outputs the amount of energy at different frequencies by performing a Fast Fourier Transform (FFT) in each band. The Node.js web server which executes the spectrum sensing script over SSH and reads the results. The Socket.io library was used to allow a client can establish a WebSocket connection with the server. Once the connection is established the server can push the real-time spectral data to the client as soon as it becomes available. Over the WebSocket connection, the client can adjust the spectrum sensing parameters, such as center frequency, bandwidth, and the number of scans per minute.

Our first implementation of spectrum sensing used HTTP polling, where the client sent a request for each time-slice of spectral data to the server. This allows receiving one time-slice of spectrum only every few seconds because of the overhead associated with sending a request each time, to which the server has to respond. This lag is not caused by the size of data which is usually on the order of a few hundred floating point values, and a few thousand at the most. Rather delays appear because request-response HTTP is not a suitable protocol for streaming real-time data.

In contrast, WebSockets do not have the HTTP overhead. They allow the server to push new spectral data to the client without the client having to request each piece. Thus, many time-slices
can be received in one second, which is significantly faster than HTTP. In fact, the upper limit of the data rate is beyond what is needed for human users.

**Figure 5-4 Real-time spectrum portrayals in CORNET3D.** Top: traditional 2D line plot, middle: 3D waterfall, bottom: 2D waterfall

WebSockets eliminate unnecessary network traffic, which is inherent in HTTP connections. HTTP is particularly unsuitable for real-time data and forces developers to resort to hacks, such
as polling. Polling involves sending HTTP requests at certain intervals to query data from the
server. Figure 5–5 shows how WebSockets outperform HTTP in terms of speed.

Figure 5–5 Comparison between WebSockets and RESTful HTTP data services [Arun Gupta. 2014. REST vs WebSocket Comparison and Benchmarks. Online. http://blog.arungupta.me/2014/02/rest-vs-websocket-comparison-benchmarks/, Used under fair use, 2014.]
5.8.4 Client-Side Implementation

Several web technologies were integrated in the CORNET3D client-side implementation. In order to supply the application with data, jQuery was used to make asynchronous HTTP GET requests to the CORNET web server, which are sometimes referred to as AJAX requests. Asynchronous data delivery is needed to ensure that the GUI is not locked up while the data is being retrieved. Additionally the client-side portion of Socket.io was used to manage WebSocket connections.

5.8.5 GUI Design

The GUI was defined using HTML and CSS, as well as more novel features, such as CSS3 transparency. jQuery UI is used to make animated, draggable, and resizable UI windows appear on the HTML page. jQuery UI was also used for button animation. Although CORNET3D resides on the web, it does not have the appearance of a typical webpage. CSS was used to make the application fill the entire browser window without a scroll bar. Additionally, the HTML5 Fullscreen API was used to allow the application to fill up the entire screen, removing browser toolbars from view. Additional control buttons, which appear at the bottom of the screen, offer a similar user experience as the taskbar buttons on most desktop operating systems. The windows can be dragged, resized, and closed just like in desktop applications. These features make CORNET3D similar in user experience to desktop applications, which most users are
accustomed to. This familiarity should have a positive effect on the user experience and the user performance.

5.8.6 WebGL and X3DOM

The 3D content of CORNET3D was coded using X3D, which is a 3D markup language that describes 3D scenes in a fashion similar to how HTML describes multimedia documents. X3D offers a higher level of abstraction than WebGL code, which simplifies development, while offering enough functionality to make it suitable for describing the 3D geometry of CORNET3D. X3D development is easier for individuals with a background in declarative standard scenegraphs, such as VRML (Virtual Reality Modeling Language), which X3D is based on. X3D is also likely to remain supported in the future because it is an ISO standard. Furthermore, X3D scenes are platform independent and can be rendered outside of web applications.

X3DOM, JavaScript library, was used to translate X3D into WebGL, which modern browsers support. X3DOM offers an advantage over other WebGL libraries to developers who have a background in X3D or VRML. X3DOM leverages the similarities between X3D and HTML by allowing DOM operations and CSS styling to be performed on X3D objects. Selecting 3D objects with document.getElementById(), changing their attributes, or adding new 3D objects using DOM manipulation is possible with X3DOM. jQuery manipulation of X3D elements is also supported. These methods were used in CORNET3D to make the 3D scene dynamic and
interactive. Thus, X3DOM creates a seamless experience for developers, by making it appear as if the browser natively supports X3D elements in the DOM tree. Thus, developers with an HTML background should find working with X3DOM easier than with certain other WebGL frameworks.

### 5.9 Data Visualization

The web application visualizes various types of data: the Kelley Hall floor plan with the location of radio nodes within the building and their status, active radios, address and port number of nodes, (Figure 3), as well as the measured energy in different frequency bands. Each node consists of a radio, which is plugged into a computing node (a Linux machine). The nodes are visualized as spheres and cylinders. These shapes are commonly used in scientific and engineering literature to denote nodes on a network, which should make their look familiar to most users. The application uses a tooltip to display the address and port information. The tooltip functionality has been packaged as a separate JavaScript library to facilitate code maintainability and reuse. The nodes are colored green to indicate that the USRP and the computing node are on. Red indicates that the computing node is off–cannot be reached over SSH. Olive indicates that the computing node is on, but the USRP is not.
5.9.1 Testbed Access–Node Plan

The floors are visualized by rendering the floor plans as textures. This creates the spatial context of the radio nodes. The floors were made semi-transparent to reduce occlusions. However, a user can hover over a floor to make it opaque for better visibility or click on a floor to hide all other floors. This design decision uses objects which already exist in the scene for user interaction, rather than adding dedicated buttons. As a result the GUI appears clean and uncluttered. The 3D view can be rotated, zoomed, and scaled through various modes of 3D interaction, which are supported by X3DOM, ranging from standard click-and-drag operations to “game mode”, where user navigation is similar to that of first-person games.

5.9.2 Spectrum Visualization

Figure 4 shows three portrayals of the actual cellular traffic in the 751 MHz 4G LTE band, along with the controls for spectrum sensing. Figure 5 shows one node sensing a signal which is emitted by another node at 460.2 MHz. The spectrum sensing settings controls to specify the precision, the data rate, the bandwidth, and the center frequency. Thus it is possible to zoom the data in and out for finer or coarser granularity of the spectrum view.
The application provides three distinct portrayals of the same spectral data: a 3D plot, a heatmap, and a traditional 2D line plot. Spectrum visualization can be started for a particular node by simply clicking on it. In case spectral data is not available (for instance, due to software updates being in process) a message appears on the top of the application. Once a node is clicked a 3D spectrum plot appears instead of the floor plan view. Clicking on the 3D spectrum plot takes the user back to the floor plan view.

The first method of portraying spectrum is a 2D line plot of energy against frequency. It only displays a single temporal snapshot, without displaying the time-variant properties of the spectrum. The plot was created using the RGraph library, which abstracts HTML5 Canvas.

The height and color of the 3D plot represent the amount of energy at a particular frequency. The 3D figure consists of several temporal snapshots of the spectrum, which are arranged consecutively with newer ones in the back. In each snapshot the horizontal dimension is the frequency. The 3D plot utilizes the waterfall representation to visualize the temporal behavior of the spectrum. A new temporal snapshot appears on the back side of the 3D plot and pushes all of the other snapshots to the front side. The oldest snapshot is removed, keeping the total number of snapshots in the 3D plot constant. The ElevationGrid X3D geometry was used to create the waterfall.
The heatmap visualization is essentially a 2D projection of the 3D spectrum plot, which maps the energy to a color value. Thus, the heatmap visualization also follows the waterfall representation, with the newest temporal snapshots appearing on the top and pushing the other snapshots to the bottom of the screen. A color scale is present to denote how the colors are mapped. The heatmap was created using HTML5 canvas, SVG, and the D3.js library, which abstracts SVG.

Having multiple portrayals of the same spectrum data allows the user to choose those that he finds the most effective. This also enables research into which portrayals of spectrum are the most effective for different user populations and tasks.

5.10 Usability Engineering

The GUI was designed using several principles, based on human information processing to allow pre-attentive perception and facilitate user performance. The GUI design utilizes the idea of affordances [Mcgrenere 2000] [Gaver 1991]. An affordance is a feature of a physical or a virtual object, which communicates its use. When a user hovers over a 3D object, the application highlights it and changes the mouse cursor to a finger-pointer. This communicates that the user can afford clicking on the object. Buttons already offer a strong affordance due to their appearance, and therefore using a finger-pointer cursor is not necessary, when hovering
over them. Given the fact that some users may not be accustomed to the novelty of this web application, affordances can steer their actions in the right direction. It is also vital to direct the user’s attention to the reference material, when they launch the application. Therefore, the GUI utilizes a singleton—a flashing help button. A singleton is a sensory cue, which is unique throughout the application [Wickens et al. 2012].

The application uses redundant coding by mapping the signal power to both color and height in 3D spectrum visualization. The redundancy makes the 3D plot easier to read [Wickens et al. 2012]. The application uses the rainbow color map because it is the dominant color scheme in engineering heat maps. In fact, during 2001 through 2005 between 45 % and 71 % of non-medical IEEE Visualization proceedings publications used the rainbow color map [Borland and Taylor 2007]. This familiar color scheme would fit well into the mental model of most users, improving user performance [Wickens et al. 2012]. The rainbow color map has been criticized for having less perceptual resolution than diverging color maps [Moreland 2009]. However, in CORNET3D it is more important to provide a clear distinction between radio signal and the noise floor. The rainbow heatmap does this well with red representing signal and blue representing the noise floor.

For the same purpose of clarity, the application uses different shapes to symbolize radio nodes on adjacent floors to make it easier for the user to distinguish which nodes belong to which floor. The application also dims the background, while the user is resizing GUI windows, to make
these windows stand out. Icons were used throughout the application to label buttons because humans can process icons much faster than text. The icons were such that they allow the user to infer their function from their appearance.

5.10.1 Leveraging the Familiar

CORNET3D aims to use elements in the GUI, which the users are already familiar with, such as the look and feel of a desktop application, widely-used node symbols, and some commonly-used icons. The red and green node color scheme leverages the existing human associations with these colors, such as those of traffic lights and other devices. In CORNET3D the user interacts with the 3D scene is a manner similar to a first-person game. These features help the user to formulate a correct mental model of the application and make correct predictions about untested situations [Wickens et al. 2012].

5.10.2 Mobile-Device Interaction

Mobile devices allow novel ways for interacting with web applications. HTML5 presents JavaScript APIs for accessing accelerometer and gyroscope data. CORNET3D uses the gyroscope data to allow the user to rotate the 3D scene by tilting the mobile device, such as an
Android tablet. Accelerometer gestures are also used: the user can shake the device to turn the orientation sensing on and off.

5.11 Spectral Games

As wireless communications systems become software-defined and the technology matures for implementing cognitive radios, researchers need to develop suitable approaches. This essentially means being able to allocate spectral resources according to the actual needs, fairly and transparently, to the end user. A game-like approach to spectrum management would increase awareness and encourage the use of cognitive radio. Gamifying the application communicates to the user that software-defined and cognitive radio is an enjoyable field of R&D, thus increasing user engagement and the actively involved user base. This aids the mission of STEM (science, technology, engineering, and mathematics) outreach, facilitates education in the field of software-defined and cognitive radio, and facilitates the recruitment of new research talent.
The modular architecture of CORNET3D, which is shown in Figure 3, allows the frontend to work with both real and simulated transmission controllers through the Cognitive Radio Test System (CRTS) backend. CRTS provides dynamic feedback on radio performance as transmission parameters are adjusted, either by computer or human operators. The front-end communicates with CRTS over WebSockets.
Figure 5 shows the user selecting a frequency band to transmit on by dragging a selection box with the mouse cursor. The application also gives the user real-time visual feedback by highlighting the selected frequencies with magenta color in the real time spectrum visualization. The user can choose additional transmission parameters using sliders and drop-down menus. These parameters are payload size, gain, delay between transmissions, inner and outer forward error correction schemes, the modulation scheme, and the type of cyclic redundancy check. The icons above the slider change in size as the slider is moved to provide visual feedback.

The users are provided the CRTS feedback metrics, which gauge the results of their selected transmission parameters (Figure 6). The metrics are also condensed into a score, shown in the top-left of Figure 5. Various scoring methods are possible. The current implementation adds one point for each valid payload and subtracts one point for each invalid one. For novice users, a score would be easier to comprehend than various technical metrics. As a result, the user is faced with an optimization problem—choosing a set of optimal transmission parameters to achieve a desired score.

CRTS also supports cognitive engines, which can be used to compete with, advise, or potentially learn from a human operator. In a way, cognitive engines take the role of non-player characters (NPCs), which are encountered in typical computer games. The pace of the game can be scaled to allow human operators of various proficiencies and cognitive engines to participate on an equal basis. Difficulty can be increased by using more sophisticated cognitive radio algorithms.
Moreover, because there are 48 nodes in CORNET3D, the backend infrastructure allows multiple users, with each user using two nodes—a sender and a receiver. Borrowing video game terminology, this means that both single and multiplayer modes are possible. As shown in Figure 5, with a constant signal at 460.2 MHz, the game can be made more difficult with such techniques as jamming and general interference.

![Figure 5–7 Radio performance metrics interface](image)

### 5.12 Future Work

Numerous game scenarios can be added to the CORNET3D spectral games. Both human and computer players can form teams. Teams could allow for relays, where a radio retransmits a
message from a radio on the same team, so that the transmission can travel farther. Repeating a signal at a different frequency, in particular, may be suitable strategy for low power transmission systems and significant channel variation within the testbed facility. The latter is particularly suitable for outdoor radio testbeds, such as O-CORNET.

It is also possible to add features, which can simulate certain real world scenarios. The application can simulate limited battery life, which depletes slowly on standby, and fast when radios make a transmission. Battery depletion will vary with transmission power levels, creating a tradeoff, which the players should optimally solve. Solar panel battery recharging can also be simulated.

Various economic implications can be implemented. Players can start out with a certain amount of virtual credits, and they can be rewarded with more virtual credits (rather than points) for successful transmissions, as if they were hired by a client to make the transmissions. There can be virtual auctions, where a player with the lowest price gets the contract to make a transmission. The virtual credits can be spent at the equipment store on virtual upgrades, such as better antennas, improved battery life, better solar panels, or more computing power. Custom software modules for radios could also be purchased, creating an app store. These modules could improve data transmission through forward error correction (FEC), upgraded cognitive engines, advanced modulation schemes, and other algorithms for software radios.
Incentives, such as virtual usage or licensing fees for certain spectrum bands could also be implemented. These fees may vary based on the demand/criticalness of the resource, just like stocks or any market value. The player would need to trade radio versus computing resources, and probably application resources (available waveforms for the given hardware) as well. Because the amount of credits will be limited, deciding how to spend them will be critical for success in the game. The amount of credits, which a player accumulates, can be used as one of the performance metrics. The top scores of all players could be compiled to create a leaderboard.

To simplify software radio operation human players can choose to let an algorithm do some of their tasks for them. The game could run on other platforms from mobile to immersive. It could utilize accelerometer and gyroscope sensors for interaction on mobile devices. For immersion, the X3D game could be run in the VisCube. The VisCube is a cave automatic virtual environment (CAVE) at Virginia Tech.

Another viewpoint for setting up the game is encouraging the development of algorithms or schedulers that make best use of the available radio resources in a fair and efficient way. This would foster the idea of spectrum being a shared resource. Rather than competing for resources and giving credit to the user that completes the transmission first, users may assign priorities to their transmissions and voluntarily give precedence to another radio that claims having a higher priority. Selfish versus cooperative behavior can then be tested and the results used for deriving appropriate strategies and training cognitive engines.
5.13 Maintaining CORNET3D

CORNET3D has a layered architecture, which consists of 3 layers, as shown in Figure 5–1.

The layers communicate with each other over network interfaces. The lowest layer executed on individual nodes and does not use web technologies.

5.13.1 Low Layer

The low layer consists of several modules, which are executed as SSH commands.

5.13.2 Node Status

The Node Status is the uhd_find_devices command.

5.13.3 Spectrum sensing

The Spectrum Sensing is a python file, which was generated from a GNU Radio Companion flowgraph. Some modifications were added to it later.
5.13.4 CRTS

CRTS was written by Ferdinando Romano.

5.13.5 Middle Layer

Middle layer is a Node.js server. It consists of two JavaScript files. One calls uhd_find_devices, and the other calls the other modules. The middle layer communicates with the low layer over SSH. To do spectrum sensing the middle layer passes the spectrum sensing parameters as command-line arguments.

5.13.6 High Layer

The high layer only communicates with the middle layer and not with the low layer. It receives node status information via a jQuery JSONP request. Other data is received using Socket.io, which is a high-level WebSocket library for Node.js.
5.14 References


6 Conclusion

GeoSpy and CORNET3D have shown that the web platform has a lot of power for creating visual tools for research and education. Moreover, several factors actually allow web applications to match and even surpass the value proposition of native software platforms. First, end-users do not need to install web applications or worry about managing updates. Second, web applications are highly cross-platform and run on a multitude of desktop and mobile platforms. Third, recent advances in the capabilities of web-browser APIs have allowed web-applications to rival traditional desktop application in functionality. Fourth, the emergence of cloud computing allows web applications to be inexpensively deployed and scaled. Finally, web applications can be accessed anywhere from a device with only a browser and an internet connection. CORNET3D has demonstrated that real time control of remote special-purpose hardware, such as a software-defined ratio, is practical from the web platform. Also, we have shown that it is possible to package dynamic spectrum access as an educational game.

A geospatial API takes latitude and longitude as input parameters and returns data for the corresponding location. The GeoSpy project showed that geospatial HTTP APIs should allow querying data for multiple coordinates in a single HTTP request to boost speeds by approximately 10 times. Unfortunately, none of the third-party geospatial HTTP APIs support this and instead allow only one coordinate per request. Our tests have shown a linear relationship between the number of coordinates and the time needed to receive the data for all of
Moreover, the tests have shown that querying for one coordinate per request is approximately 10 times slower than querying for all coordinates in a single request.

WebSockets have presented a clear advantage over HTTP polling for pushing real-time data from the sensor through the server to the client. Our first implementation of spectrum sensing used HTTP polling, where the client sent a request for each time-slice of spectral data to the server. This allows receiving one time-slice of spectrum every only few seconds because of the overhead associated with sending a request each time, to which the server has to respond.

This lag is not caused by the size of data which is usually on the order of a few hundred floating point values, and a few thousand at the most. Rather delays appear because request-response HTTP is not a suitable protocol for streaming real-time data.

WebSockets do not have the HTTP overhead. They allow the server to push new spectral data to the client without the client having to request each piece. Thus, many time-slices can be received in one second, which is significantly faster than HTTP. In fact, the upper limit of the data the rate is beyond what is needed for human users.