NETWORK CHANNEL VISUALIZING SIMULATOR: A REAL-TIME, 3D, INTERACTIVE NETWORK SIMULATION PLATFORM

A Thesis

Presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Sean Forsberg

June 2012

© 2012

Sean Forsberg

ALL RIGHTS RESERVED

COMMITTEE MEMBERSHIP

TITLE:	Network Channel Visualizing Simulator: A Real-time, 3D, Interactive Network Simu- lation Platform
AUTHOR:	Sean Forsberg
DATE SUBMITTED:	June 2012
COMMITTEE CHAIR:	John Bellardo, Ph.D.
COMMITTEE MEMBER:	Hugh Smith, Ph.D.
COMMITTEE MEMBER:	Chris Turner, Ph.D.

Abstract

Network Channel Visualizing Simulator: A Real-time, 3D, Interactive Network Simulation Platform

Sean Forsberg

With a focus of always being connected, it's become typical for laptops and mobile devices to include multiple wireless network devices. Though the additional network devices have created mobility and versatility of how a user is connected, it is common for only one to be active at any given time. While likely that new mesh protocols will help maximize connectivity and power consumption by utilizing lower-power multi-hop techniques, it is still difficult to visualize these protocols due to the complexity created by each node's simple choices. Further challenges are presented by the variety of network devices which share frequency ranges with different output power, sensitivities, and antenna radiation patterns. Due to the complexity of these configurations and environments, it becomes clear that reproducible simulations are required.

While several network simulators have been thoroughly tested over their many years of use, they often lack realistic handling of key factors that affect wireless networks. A few examples include cross-channel interference, propagation delays, interference caused by nodes beyond communication range, channel switching delays, and non-uniform radiation patterns. Another key limitation of these past tools is their limited methods for clearly displaying characteristics of multichannel communication. Furthermore, these past utilities lack the graphical and interactive functions which promote the discovery of edge cases through the use of human intuition and pattern recognition.

Even with their other limitations, many of these simulators are also extend-

able with new components and simulation abilities. As a result, a large set of protocols and other useful discoveries have been developed. While the concepts are well tested and verified, a new challenge is found when moving code from prototype to production due to code portability problems. Due to the sophistication of these creations, even small changes in code during a protocols release can have dramatic effects on its functionality. Both to encourage quicker development cycles and maintain code validation, it would be advantageous to provide simulation interfaces which directly match that of production systems.

To overcome the various challenges presented and encourage the use of innate human abilities, this paper presents a novel simulation framework, Network Channel Visualizing Simulator (NCVS), with a real-time, interactive, 3D environment with clear representation and simulation of multi-channel RF communication through multiple network device types.

Contents

Li	List of Tables		$\mathbf{i}\mathbf{x}$	
1	Intr	oducti	ion	1
	1.1	Softwa	are Testing	2
	1.2	1.2 Project Motivation		
		1.2.1	Diverse, Multi-Interface MANET (Mobile Ad-hoc Network)	5
		1.2.2	Automatic Distributed Network Configuration	6
	1.3	Netwo	rk Channel Visualizing Simulator (NCVS)	7
2	Rel	ated V	Vork	9
	2.1	Netwo	rk Simulation	9
		2.1.1	NS-2	10
		2.1.2	NS-3	11
		2.1.3	OPNET (Optimized Network Engineering Tools)	12
		2.1.4	OMNeT++ (Objective Modular Network Testbed in C++)	14
		2.1.5	PlanetLab	15
		2.1.6	Others	15
2.2 Similar Immersive Simulations		r Immersive Simulations	17	
		2.2.1	Mechanical Design	17
		2.2.2	Architecture and Civil Planning	18
	2.3	2.3 Commonality with Ray Tracing		18
		2.3.1	Wireless Link Calculation	19
		2.3.2	Traveling Wave Effects	19
3	Imp	lemen	tation	20

	3.1	Goals		20
		3.1.1	Interactive, 3D Environment	21
		3.1.2	Shared Mediums	22
		3.1.3	Extensibility and Portability	23
	3.2	Code	$Design \ldots 2$	24
		3.2.1	Object Oriented	24
		3.2.2	Publisher-Subscriber	24
		3.2.3	Partition between Simulation and Graphics	25
	3.3	Simula	ation Class Hierarchy	25
		3.3.1	Overview	25
		3.3.2	Class Details	27
	3.4	Graph	ic Class Hierarchy	10
		3.4.1	Overview	10
		3.4.2	Class Details	11
	3.5	Mediu	m Channel-Link Creation	52
		3.5.1	Wired	53
		3.5.2	RF	53
	3.6	Data '	$Transmission \dots \dots \dots \dots \dots \dots \dots \dots \dots $	59
		3.6.1	Timing	59
		3.6.2	Collisions $\ldots \ldots \ldots$	30
		3.6.3	Implementation Limitations	52
	3.7	Chann		52
	3.8	Specia	al Scheduling Considerations $\ldots \ldots $	33
	3.9	Frame	$e \text{ Data Handling} \ldots $	34
4	Vali	dation	6	55
	4.1	Physic	cal Layer (OSI-Layer 1) Isolation	36
		4.1.1	Summary	36
		4.1.2	Procedure	37
		4.1.3	Results	37
	4.2	CSMA	A/CD (OSI-Layer 2) Isolation	38
		4.2.1	Summary	38
			·	

		4.2.2	Procedure	69		
		4.2.3	Results	69		
	4.3 Experiment Discussion			71		
		4.3.1	Click Timing	71		
		4.3.2	Simulator Runtime	72		
5	Fut	ure Wo	ork	73		
	5.1	Optim	izations	73		
		5.1.1	Multi-Threaded	74		
		5.1.2	RF Link-State Calculations	77		
		5.1.3	Perspective Based Rendering	79		
		5.1.4	2D & 3D Click Detection	81		
	5.2	User E	$\mathbf{Experience} \ . \ . \ . \ . \ . \ . \ . \ . \ . \ $	81		
		5.2.1	Node Sub-Component Placement	82		
		5.2.2	Menu Elements & Support	83		
		5.2.3	Dynamic Click Configuration Creation	84		
		5.2.4	Routing Topology Highlighting	86		
	5.3	Simula	ation Expansion	87		
		5.3.1	RF Simulation Expansion	88		
		5.3.2	Link Error Models	96		
		5.3.3	Wi-Fi Scanning	98		
		5.3.4	Power Consumption Modeling	99		
		5.3.5	Random Generation Manager	101		
		5.3.6	Special Simulation Modes	102		
	5.4	Additi	ional Modules	104		
		5.4.1	Simulated Devices	105		
		5.4.2	Protocols	106		
6	Con	clusio	n	108		
Bi	Bibliography 110					

List of Tables

3.1	Broadcast Event Enumerations	39
3.2	Rendered Object Classes	49
3.3	Rough Link Calculations	57
3.4	Rough Link Calculations (Optimizied)	58
41	L2 Experiments - PCAP summary (NS-3)	70
1.1		10
4.2	L2 Experiments - PCAP summary (NCVS)	70

Chapter 1

Introduction

The core of any new invention is the need to test and verify its functionality and performance. For small physical devices, this can be done by mechanically pushing the device to its extremes. As these tests are being performed, our innate comprehension of the physical world allows us to grasp the operations being performed and the resulting stresses they cause. Combined with the human ability of pattern recognition, the designer is able to make correlations, iteratively improve the design, and end up with a well performing product. Such abilities of pattern recognition and intuition of the physical world derives while years of experience in interaction from new-born through adulthood. While education increases skills and provide explanation of why things work the way they do, they only add to the inherent understanding created during cognitive development.

This task may be more difficult for larger physical objects, say a bridge, but is still feasible. Each independent component can be tested separately, a reducedscale model can be stressed, and a computer simulation can be used to provide system analysis of several factors at once. While simulations of physical objects can often achieve good results, they can't account for everything. This is partially due to the number variables and the time it would take to calculate every case. Instead, designs will focus on areas they will be weak. While unique combinations of variables may have unexpected results, the human's innate comprehension of the world contributes to their overall success. If nothing else, this comprehension allows the observers to notice peculiar events, even without understanding why it looks incorrect. Another solution in physical design is the use of a significant safety factor, especially when human safety is involved. Unfortunately this worldly intuition and ability to add a "fudge factor" often doesn't translate to software.

1.1 Software Testing

Like complex physical structures, software packages are often the composition of multiple smaller components. Computer systems, and the software they run, are extremely complex and developed by large teams of engineers. Advanced features and user-friendly interfaces require utilize the computer power of consumer systems. However in the case of networks, complexity is often not created by monolithic components like other software systems. Instead, the complex interactions between extremely simple tasks allow them to function at gigahertz link speeds.

These challenges are probably best related to the study of insect logic. A great deal of performance is achieved through a small set of extremely simple tasks, performance at an extraordinary magnitude. While each instruction may be easy to understand, the connection between those tasks and the results when performed millions or billions of times are often not. In order to attempt to grasp the nuances and edge cases, scientists utilize the same functional testing mentioned above. They study each instruction individually, build robotic devices to mimic the tasks in small scale, and use computer simulations to try and catch special cases which only happen at full scale. While such tests prove useful for insect study, there are a number of challenges bringing to the similarly challenged study of network protocols.

Code can be reviewed, small-scale unit tests run, and large-scale simulations run but confidence in the results often lack due to the difficulty in catching unexpected behavior. Additionally, some network protocol components cannot be easily split up. A great example of being unable to easily divide a protocol is found with TCP. As discovered by researchers developing the Datagram Congestion Control Protocol (DCCP) [29], the congestion control techniques of TCP fall apart like a house-of-cards when transmission reliability is removed. While this could have been reasoned out, it's not uncommon for these cases to creep up when unexpected. Additionally, there are numerous cases when the complexities of the edge cases make it extremely unlikely for the designer to plan for them. As a result, these unforeseen cases are very difficult to develop experiments which will expose them.

Another from with current network simulation tools is the lack of code portability. In this case, the code tested is different than the production code. Although this still allows for the concepts behind to be validated, the production implementation is not. As such, the production code is vulnerable to human error during the code port.

While physical simulation tools allow the user to interactively add forces, current network simulation tools lack this ability to interact and, therefore, the ability to maximize human intuition. Once a simulation is run, additional tools are used to reformat the output in order to mold it into a human useable format. After tweaking a variable, the simulation is again run, data reformatted, and analyzed. While this procedure has the flavor of iterating over a physical design, the code and system complexity is often not supported by the same intuition embedded in humans like that with physical objects. At the core of this fault is the lack of ability to easily poke, prod, and manipulate the system.

Another significant difficulty system design is the lack of ability to truly grasp the scope of complex systems. While most high-school students could recite the speed-of-light, they would be lacking when it came to explaining the magnitude of a 10⁸ operation. This is a great example of being able to state something without truly grasping the nuisances. This is often the case when analyzing simple-withmagnitude operations where intuition is greatly reduced. With the exception of code that can be logically proven, the large scope of a software system limits the ability for verification. This presents a perfect example of how tools for development need to move forward with the development of new technologies.

1.2 **Project Motivation**

While there are a number of network simulators out there that can provide end-to-end verification, they often lack portability to production and user interfaces which encourage intuitive discoveries. As research pushes forward, and the complexity of inter-dependence of variables increase, the need to rethink the user experience of tools should be reconsidered. This sub-section will present several examples of on-going (and future) research focused on network mobility and connectivity which acted as the driving force for this paper. While these topics influence each other, the following definitions will be used:

- Mobility: The ability to always be connected to the Internet while actively moving.
- Connectivity is the ability to connect with other neighboring devices

1.2.1 Diverse, Multi-Interface MANET (Mobile Ad-hoc Network)

With more devices including multiple network devices and a greater focus put on always being connected, several network research opportunities present themselves. As usual there are challenges pared with the opportunities [42][28][21]. The first item is the simple need for a method to visualize how each interface interacts with like devices. For example, how does Node A with a long-range radio, Wi-Fi, and Bluetooth interact with another similar device, Node B? This is enough of a challenge with just two nodes when considering the various factors such as orientation (both node and antenna), antennas, interference, transmit power, and the network devices themselves. Expanding this thought experiment quickly becomes unmanageable due to the inter-relations. Additionally, like DCCP [29], separating the components results in an incomplete analysis due to cross-dependence created by a shared medium.

There are also several less common projects could benefit from the ability to visually see the results of multi-interface characteristics. Beyond the usual mobile phones, laptops, and other portable electronics, there are Underwater Autonomous Vehicles (UAV), advanced sensor networks [48][27] (such as those used by utility companies), and the increased use of smart appliances (such as TVs and stereos.) As the use of portable electronics has increased, so has the desire to always be connected to both the Internet as well as to each other. Protocols such as AirPlay [3], DLNA [5], and Wi-Fi Direct [16] create links with neighboring devices for shared media transport and use. With this increased connectivity comes greater power consumption and RF interference caused by the network devices. These two goals, mobility and connectivity, end up sharing many of the same end needs, require the same tools for development, and could benefit from each other if they were able to coordinate their efforts.

Even devices which dynamically adjust their transmission power to minimize its RF footprint [33][23][25][51][50], they do not benefit from nearby devices perhaps with greater transmit power (thus range) and reduced power constraints. While research continues on multi-hop mesh networks [45][33][47][35][19][32][34], most devices today still resort to infrastructure modes where connections are made from device to a central device. Furthermore, much of the mesh network research is focused on a single device protocol instead of bridging network devices. While mobility and connectivity could both be increased with diverse, multi-interface mesh networking, further inter-dependencies are created by crossdevice interference. In order to maximize these opportunities, in addition to other protocol opportunities, devices will need to act as a cohesive unit.

1.2.2 Automatic Distributed Network Configuration

With more information available, a network trunk can achieve many optimizations that several smaller links with the same total bandwidth could not (i.e. compression, coding, error correction, and so on.) Instead of using this type of cohesive system, many networks instead fight for the medium and thus not only lose the opportunity for optimization but a stricken with higher overhead due to collisions.

A perfect example of such losses is found in dense urban areas where multiple housing units each have their own wireless routers. Even when the frequencies of the 802.11 networks are well configured, the small set of orthogonal channels can result in high Radio Frequency (RF) contention. So while signal strength may be high, actual bandwidth is dramatically reduced. Additionally, RF devices create interference at much greater distances than connections can be established.

In such a system, maximizing link stability and minimizing overhead will be important to success. While some testing can be done on independent units of the system, wide-spread use will require verification of the system as a whole. As with any cutting-edge research, a number of expected and unexpected edge cases are sure to develop[26]. It quickly becomes clear that network simulators were lacking an environment which utilized human intuition and pattern recognition.

1.3 Network Channel Visualizing Simulator (NCVS)

Current network simulators are able to provide testing environments for a wide-range of protocols. While they have been thoroughly used in research, they present a number of challenges which hamper development of new experimental protocols. In order to overcome these deficiencies in current network simulation and development tools, NCVS is presented as an extendable, real-time simulation engine with an immersive GUI to encourage intuitive discoveries in complex network environments. This includes a 3D interface focused on clearly indicating the various network devices and their channel dependencies. With the addition of the Click Modular Router [30] and other code interfaces which recreate production environments, NCVS provides the ability to quickly move from prototype to production with minimal (if any) code changes. The result is "sandbox" simulation environment that can be used to discover, test, and verify advanced network protocols.

The rest of this paper will be presented in the following: Section 2 will provide a survey of the network simulation tools currently available (and how they compare with NCV.) Section 3 will provide an overview of the design of NCV. Section 4 will compare and validate components of NCVS with another mainstream tool. Finally, section 5 will summarize additional functions that would be helpful to add to NCV.

Chapter 2

Related Work

The concepts behind NCVS are obviously not completely new. Instead, its goal is to combine the successful features of past simulators, propose solutions to unsuccessful traits, and provide the immersive environment that other engineering fields have used for years to networks. To facilitate the recreation of networks simulation environments, the tools used to simulate environments in other engineering fields will be reviewed. It's also important to note the many similar performance issues, and solutions, that ray tracers have with wireless network simulation.

2.1 Network Simulation

In the field of network simulation, a vast number of utilities have been developed for every special case under the sun[44][48][31]. Many of these do not provide any unique infrastructure and simply create a very specific use-case. To avoid the need to recreate simulators for every case, it becomes clear there is a need for an easy-to-use, flexible simulation engine that can accomplish specialized tasks. Furthermore, several of the tools were created to support direct integration with a specific final product. While several of these simulation tools are not elaborated on below due to the extremely small niches, they do demonstrate the need for a simulation tool to bridge prototype-to-production.

Though a large quantity of simulators is available, some including useful and unique functions, the common four referenced time and time again are NS-2 [7], NS-3 [8], OPNET [10], and OMNet++ [9]. Although most of the other simulation tools are use-case specific or a duplicate of these main tools, a select few have emerged with a unique features. Perhaps the most specialized semi-simulation and emulation engine, PlanetLab [12], is worth special note due to the ability of testing protocols on world-scale networks with actual workloads and error conditions. To round out the survey, a small set of other specialized tools with unique features will be briefly touched on.

2.1.1 NS-2

Obviously anyone in the field of networks has heard of, and probably used, NS-2 which is sponsored by DARPA and NSF. Among other reasons, it's long history, large base of contributed protocols and devices, and huge user-base all help create confidence in its simulation core. Furthermore as an open-source platform with a versatile, ISO & OSI focused class structure, new functions are added on a regular basis. Using C++ for its low-level component code and OTcl, an object-oriented version of Tcl script language, for its simulation setup and control, NS-2 attempts to provide an optimized interface for development.

With on-going development keeping NS-2 current since the early 90's, a number of models tracking communication channels, battery support, and protocol stacks are included. Various forms of wireless protocols are implemented including IEEE 802.11 and satellite communications. Additionally error models, such as link interruption and node modification, are editable via the OTcl scenario scripts. NS-2 also supports both non-real-time and real-time simulation modes. The real-time mode supports interactions with real network connections can be used in an emulation mode. NS-2 also has extensions which enable some parallel and distributed execution options.

While fairly simple, NS-2 does have extensions which provide graphical interfaces for scenario generation. With several limitations on node state and accuracy, the use of NAM provides some visual support for post-simulation analysis of events. Other analytics are performed manually by reviewing captured pcap files and custom configured event trace output. While some extensions have been created to assist with this process, they are all post-processors with limited information.

Although NS-2 has provided support in a long history of developments, there are several limitations [20]. In addition to a heavy learning curve for use, there are several annoyances such as user defined objects requiring full NS-2 recompile. [31] also references the high resource usage of the simulator.

2.1.2 NS-3

In an effort to build on the successes, and correct failures, of NS-2, a new generation of the simulator has been created from the ground up. While not compatible with NS-2, NS-3 provides more flexibility in its design while being friendlier to users. Examples of such improvements include using Python for scripting instead of the less common OTcl language.

Other changes include a focus on the realism, code portability, better interaction with the real-world, and custom tracing extensions. In the first case, a focus on realism is targeted at making simulations tie closer to that expected in deployment of protocols. Portability focuses on reducing the need to rewrite code and instead make it possible to incorporate open-source networking developments. Using virtual system interface, the simulation is able to connect to real networks and perform emulation functionality. Lastly, the NS-3 designers wished to avoid full simulator recompiling by providing an API to support custom tracing in user developed code.

Of course a challenge faced by a new simulator, as NS-3 is, the simulator's core functionality lacks credibility due to the lack of a long history of use. Like new modules for NS-2, this doesn't mean that the code is not valid, just unverified. In order to gain the reputation that NS-2 has, NS-3 will need more users to test, create, and contribute new code. By doing this, the building blocks used to create the new modules gain verification and NS-3 gains its credibility. To help this task, several specialists have taken on various components of NS-3 and are providing support in order to develop user confidence

2.1.3 OPNET (Optimized Network Engineering Tools)

A slightly different product, OPNET provides a suite of commercial products to support network development, setup, and maintenance. While the latter two provide interesting opportunities for tool and protocol verification via simulated versus production comparison, the network development is provided in their OPNET Modeler package. However due to the multiple products, OPNET Technologies, Inc has a large industry following with a significant market share. Now focusing on OPNET Modeler, it follows the traditional object-oriented, hierarchical model structure. Along the lines of NS-3, OPNET focuses on realistic scenarios for testing protocols. Additionally, it provides support for defining custom packets as to support its internal tracing and statistic creation. Inter-module communication is handled through a common messaging exchange system.

As a result of being a commercial product, a significant effort was put into optimization and accuracy. In addition to being fast, the company backs the credibility of the large set of included components. The OPNET Modeler also boasts parallel and distributed simulation abilities. Another key factor is solid RF modeling with customizable factors such as propagation delay, interference, and various transmission and receiver characteristics. Furthermore, interconnection with wired networks and node mobility including handover support is provided.

OPNET Modeler also provides some graphical interfaces for building components from smaller building blocks (similar to that of the Click Modular Router.) Once the components are created, other graphical interfaces help create the topologies, run the simulation, and provide analysis and debugging output.

In addition to being able to select various tracing output option, the analysis functions are mathematical filters to statistically reduce or combine data. [31] also references performances statistics which are collected a runtime. The OPNET suite also supports accessing real network infrastructure and can act as a sink for NetFlow data.

2.1.4 OMNeT++ (Objective Modular Network Testbed in C++)

With growing support, OMNeT++ provides an interesting set of abilities. Created by Andras Varga from Technical University of Budapest, the design appears to be a hybrid of several simulators. Like the other simulators presented, OMNeT++ uses C++ in addition to a topology descriptive language, NED. Furthermore, a modular hierarchy is created with Click-like ports for message passing. Another unique design element is that the simulation kernel compiles into the users application. As such, it can run completely independent of the simulator itself.

OMNeT++ provides a wide array of modules and specialized child-projects, such as Castalia for wireless sensor networks (WSN). Due to the special cases which emerge in low-power devices, Castalia uses the versatility of OMNeT++ to handle issues like clock-drift, properties of low-power network devices, and power consumption tracking. In addition to a graphical interface, a commandline interface (CLI) is available for batch processing. OMNeT++ also includes parallel processing via MPI and PVM3 APIs.

A number of graphical interface are provided to edit the network, handle simulation execution, and process the output. As OMNeT++ uses its own descriptive language, a graphical edited was provided to ease development. For simulate tasks, OMNeT++ is the first to provide interactive, real-time support. While executing a simulation, the user can pause and update various parameters on-the-fly. Lastly, the output processing seems to provide verbose set of graphs and plots to analyze event timing. Although we have not had a chance to use the utility, the videos presented seemed to show good reporting. However, [31] suggested that OMNeT++ incorporated poor reporting and required the user to implement the output.

2.1.5 PlanetLab

While not a simulator, PlanetLab provides large-scale, world-wide network and system visualization for "planetary-scale" [12] experiments. As of April 29, 2012, 1125 nodes at 542 world-wide sites [12]. In addition to provide realistic link characteristics and failures due to being a real network, artificial events can be triggered.

PlanetLab also is continually updating their own platform through folding development achievements back into the system. An example of such an extension is VINI, Virtual Network Infrastructure, which provides additional controls over network conditions [18].

2.1.6 Others

In addition to the mainstream simulators included above, special environments and situations have spawned the creating of several smaller projects with unique features with mentioning.

General

With no real special category, these simulators simply have interesting features not referenced in previous ones. **JSim** As the 'J' suggestions, JSim [6] is Java based with an additional scripting language, Jacl, which is the Java version of Tcl. While only supporting the 802.11 MAC layer, 3 propagation models were implemented including Free Space, Two-ray Ground, and Irregular Terrain modules.

QualNet As the commercial derivative of GloMoSim [2] and PARSEC [17], QualNet [13] appears to be focused on parallel execution for Battlefield network simulation. In addition to referencing several common wireless device and propagation library modules, their site references integration with actual client applications and live hardware and software components. They also reference a 2D and 3D interface for network topology creation. However with them mentioning International Traffic in Arms Regulations (ITAR) restrictions, it is likely that they have done simulation work for the United States Military and that there would be other significant code restrictions.

Sensor Networks

Sensor networks have a number of special requirements including custom radios with non-OSI-layer protocols and other unique characteristics. Additionally power consumption and timing can be extremely important due to the lack of external power sources.

TOSSIM The most unique feature for TOSSIM[38] is the bit granularity support on its network protocols. It also uses a C dialect called nesC for use with TinyOS so resulting code is extremely small. However,[48] references problems with the fine-grained timing and interrupt handling of the compiled code.

UWSIM As a simulator for underwater sensor networks, UWSIM [15] includes support for acoustic modems. Taking into consideration factors such as salinity, temp, and depth allow the sound wave propagation delays to be correctly modeled[48].

2.2 Similar Immersive Simulations

As has already been mentioned, NCVS has the goal of bringing an immersive environment to network simulation. While referencing the possible usefulness and intuitive nature of a 3D environment, no solid examples have been provided. Here we present a few engineering fields which utilize 3D Computer Aided Design (CAD) for various purposes.

2.2.1 Mechanical Design

Depending on the complexity and materials included in an object, physical prototyping can be both financial and time expensive. While 2D CAD had been available for some time, it was difficult to visualize layers of complex parts. The introduction of 3D CAD dramatically changed the game. By allowing the designer to easily rotate the object and view it from various, like he or she would with a physical prototype, tolerance and layout flaws could be caught without expense. Software packages, like SpaceClaim [14], enable rapid prototyping by simply clicking and dragging objects around.

Taking mechanical prototyping to the next level, some 3D CAD application allow stress analysis, fully functional movement, and other "like real" actions. Packages, like SolidWorks [1], allow complex devices to be created, tested, and broken without ever creating a physical object. While some analysis may take an extended period of time to complete, it is significantly less than that to actually create the object.

2.2.2 Architecture and Civil Planning

While this applies to mechanical 3D CAD situations as well, architects need to be able to convey a clear image of a design to clients. Whether this is a bridge, building, city layout, or so on, an image is with a thousand words. AutoCAD Civil 3D [4] is an example of 3D simulation being used for civil works. In this case, not only does it provide a visual reference for how a city functions together, but it also provides structural analysis features. Unlike designing a widget which could be physically prototyped, a bridge or building at most can be created in a scaled version.

2.3 Commonality with Ray Tracing

While developing a network simulator, there are a number of issues which arise when scaling tasks to large numbers of components. Rather than recreating the wheel, many solutions have already been created to handle identical problems in ray tracing. While some solutions are mathematical magic, some involve caching, partitioning, and even just indexing trickery. In any case, it's important to keep this unrelated work with related problems in mind when seeking solutions. Here we present two such cases, wireless link calculation and traveling wave effects. Since the solutions will be referenced in later sections, only the high level concepts will be presented here.

2.3.1 Wireless Link Calculation

As wireless connections have a number of radio transmission effects which determine if two nodes can contact each other, several math intensive checks have to be performed. Since these checks will need to be performed on every RF network device on every Node, code complexity is $O\{N^N\}$ (where N is the number of network devices.) While in small scale, this is not a problem. However, when expanding the number of nodes with multiple wireless network devices, the complexity quickly growths.

In ray tracing, bounding volumes structures to subdivide the environment and maximum ranges are often used to simplify the complexity of such issues. With a bit of memory, per-computation, and maintenance overhead a significant reduce in run-time computational load is achieved.

2.3.2 Traveling Wave Effects

Another key item in ray tracing is the ability of light to bounce off and bend around (reflection and diffraction respectively.) While one solution is to simply ignore such attributes and assume line-of-sight only, this results in an unrealistic result. Instead, approximations, bounding volumes, and maximum ranges can again be used to dramatically reduce the computational requirements.

Chapter 3

Implementation

Since NCVS is meant to be a ground-up rethink on previous simulation engines, it's important to establish the clear goals and reasoning. Several factors were taken into consideration when developing the API in order for it to be extensible as new devices, technologies, and portable elements are designed. This section will focus on conveying the various interfaces and current implementation used throughout the tool.

3.1 Goals

As with any project, it's important to set specific goals in place to ensure the task desired is achieved. This is perhaps even more important for NCVS since it acts as an API for future development. It is also important to clearly state the goals which make NCVS unique among network simulators. Perhaps the most significant is the focus on interactivity and 3D rendering for channel separation. With a focus on multi-interface, multi-channel research, there is also an importance for handling of shared mediums such as Radio Frequency (RF) communication. A component of providing an API is providing for future development of the tool with extensibility. As this tool will be used for testing and verification, component interfaces should closely match those of real-world application in order to minimize prototype-to-production code changes.

3.1.1 Interactive, 3D Environment

At the core of any engineer comes questions like, "what if we do this?," or, "I wonder how this affects that," when looking at something new. NCVS not only provides a means for quickly answering these questions but in fact encourages them. Probably the most influential design goal of NCVS is the interactive, 3D component. Where many previous simulators focused on number crunching prior established experiments as quickly as possible then using post-analysis, NCVS focuses on immersing the developer into the network world. As a result, a NCVS acts as a network development sandbox. Instead of post-analyzing data, the user can fly around this world, look at a variety of current state information, and make changes to the current setup to further expose interesting components as they emerge. While this may not assist all stages or types of protocol development, it does add significant benefit to the motivating use cases previously mentioned.

While the 3D component may seem like a gimmick that doesn't add to functionality, the vertical axis allows individual mediums and channels to be separated. Thus instead of simply showing links between nodes, these links are clearly visible on a per-channel of each interface of each node. NCVS is also not limited to 2D position grids and includes facing directions. While a simulator with theoretical isotropic (equal gain in all directions) antennas may not benefit from such considerations, as will be discussed later in the section, NCVS can properly model the variable signal lobes.

Beyond assisting developers with the creation of new, unique research, this immersive environment also dramatically assists with developer-to-developer communication. With the complex environments in which this simulator will find itself, discussion of the experiment being conducted can be difficult and prone to miscommunication. Using NCVS, the communication can use humans strong visual learning abilities rather than require each person to visualize the situation in their mind.

While development may have been focused on development, there are significant auxiliary effects of this immersive environment worth mentioning. A wise professor once mentioned the value of the PR component of such functionality is also not to be under-estimated. In addition to assist with developer-to-developer communication, NCVS provides a straightforward channel for developer-to-client communication. Especially since clients often don't have the technical backing or familiarity with the projects details, they lack the background required to mentally visualize the situations. By developing in NCVS, the developers will find themselves having already created significant portions of presentations.

3.1.2 Shared Mediums

While encouraging curiosity and enabling marketing is important, NCVS is a simulator at its core. Again referencing the influencing use-cases of NCVS, cross-channel modeling was a key goal. The RF medium is the most significant example of devices having channel-based communication which interferes with other channels. However, several other mediums and protocols create the same environment. An example is Ethernet Virtual Local Area Network (VLAN)s. While each VLAN acts as an independent broadcast domain, the wire can only transfer one "channel" at a time, thus a shared medium.

While wire-based shared mediums are fairly straightforward to handle, the multitude of variables affecting RF present several challenges. Even though there are many challenges to modeling RF, its performance directly ties to the validity of any testing of Dynamic Frequency Selection (DFS), Cross-Channel Interference (CCI), and Packet Success Rate (PSR.) At minimum, the links should take in consideration the transmit power, the antenna gains at the given direction, free path signal attenuation, the receivers sensitivity, and background noise.

While moving at the speed-of-light, it may still take a packets bit several microseconds to arrive at their destination. As such, all links need to consider propagation delays and the transmission time of packets in order to correctly handle the collision domains of mediums. Although 802.11 devices often assume a 1 microsecond delay maximum (related to the specified ~300 meter range), long-range radios such as those used in mobile phone networks may take over 50.

3.1.3 Extensibility and Portability

Simply as a young simulator, its extremely important that NCVS be flexible, easy to use and easy to extend. Even with the significant functionality already included in NCVS, special use-cases are guaranteed to change the goal priorities listed here. Furthermore, only subsets of devices, protocols, and managers have been developed thus far.

In addition to extensibility, it's also important to minimize the changes required to move a prototype to production. Without this consideration, any validation and testing of a protocol can become useless due to the sensitivity of many protocols to the smallest of changes. As such, code interfaces should closely (or exactly) match those used in production. Again referencing flexibility, NCVS should be able to support the inclusion of common APIs such as Netfilters and Interface APIs of Linux.

3.2 Code Design

With the general goals in mind, we'll facilitate future discussions with an overview of how the code is organized. While some of these details may not play much importance on current functionality discussed in this paper, this simulator is focused on future development. Additionally this content should help clarify how the interfaces discussed in the next sub-section are used.

3.2.1 Object Oriented

In order to meet the goal of extensibility, it makes sense to use object-oriented architecture. While there is some performance loss, this is offset by the easy extension of interfaces and modification of already implemented classes.

3.2.2 Publisher-Subscriber

Along the lines of many APIs these days, NCVS uses a publisher-subscriber model for many of its operators. Examples of this include network devices subscribing to a node, objects subscribing to a scheduler for timed events, and global notification of system events such as object state changes. This provides an easy to extend model for future development and encourages the use of Managers to handle common tasks (instead of duplicating code through the project.)

3.2.3 Partition between Simulation and Graphics

Another important separation to maintain is the separation between simulation and visualization. While the visualization classes will need information from the simulation, this can be achieved through notifications of updates in the publisher-subscriber model. By maintaining this separation, not only could a head-less (or GUI-less) version of NCVS be created for high-speed output (like other simulators) but it also will ease future development of a multi-threaded (or multi-system) version of NCVS.

3.3 Simulation Class Hierarchy

This sub-section will outline the various interfaces of the simulation side of NCVS. Utilizing the publisher-subscriber design, a functional hierarchy of devices and managers emerge. While devices often subscribe to a higher device, managers are utilized to coordinate tasks and manage common information. By extending these interfaces (and base classes), new functionality can quickly be added with minimal code.

3.3.1 Overview

Prior to reviewing the various interfaces, below provides a normal outline of the hierarchy that has developed to emulate a physical network. The simulation classes are further separated into two categories, components and managers. We'll start by describing the components since it will make the overview of the managers either.

Components

Components are classes designed to implement a specific instance of a function, or partition, of the network system. While many components emulate physical devices such as a node or network device, other components implement protocols and simulate the mediums and links between devices. Below is the typical hierarchy of these components.

- Nodes
 - Protocols
 - * Medium Access Control (MAC)
 - * Routing
 - \cdot Netfilters
 - · Click Modular Router
 - NetDevices
 - * NetDevice Feature
 - \cdot Antenna
 - * Channels
- Mediums
 - Channel Links
- Animation

Managers

In order to create, maintain, and interconnect the various component instances, managers are implemented with global singletons and provide the core of NCVS. Not only do managers reduce the duplication of code, they also provide opportunity for optimization due to their global knowledge of the system. In the simulation side of the code, 4 managers are used regularly.

- SimManager: Master class which acts as a hub of all simulation component instances.
- Scheduler: Handles all time-based events for all simulation objects.
- EventManager: Acts as a public hub for action triggered events which are broadcast system-wide to subscribers.
- **RF_MediumManager:** A central medium instance which handles crossmedium RF link and interference modeling based on frequencies.

3.3.2 Class Details

With the general components and managers groups described, this sub-section will outline the various objects specific tasks as well as describe what has been implemented.

Nodes

Central to the simulation is an object containing the various NetDevices, protocols, and provide position information. Based on this simple definition, a wide range of devices emerge. One subset of Nodes includes network infrastructure such as switches, access points, and routers. Another includes desktop computers, servers, and laptop. Mobile device examples include phones, tablets, and media players. Although less commonly considered as such, TVs, embedded
wireless sensors, and other appliances are emerging in research of how wireless communication can be maximized. While not implemented, yet, RF noisy devices such as microwaves could also be considered a Node. While a noise-maker wouldn't likely have a functional network device, the infrastructure for network simulation could be used to replicate issues found in the world.

What's the common ground on these objects? The key components to a node are to provide position and orientation information as well as ability to contain NetDevices and protocols.

At the time of writing this paper, 3 general classes of objects were included in development, General Node, Switch, and Access Point. Currently there are no special simulation functions of one over another since each requires it's NetDevices and protocols to be manually assigned. However, the infrastructure was added with future development in mind. While general devices create flexibility for the time being, future works using NCVS will likely use several devices with specific models and configurations. As such, the infrastructure in place would allow the developer to create specific classes with preconfigured sets of NetDevices with specific Tx power, Antennas, Rx Sensitivity, and other settings by simply instantiating the class.

To facilitate cleaner experiment code, a NodeFactory was implemented which hides the large number of header files as well as to act as a creation and destruction manager.

Protocols

This group of components handles all communication decisions and payload creation. While NetDevices include the MAC-layer (OSI Layer-2) protocols, IP (OSI Layer-3) and above are included in the Nodes.

Medium Access Control As previously mentioned, MAC layers are often included in the implementation very close to the NetDevices (if not on-board.) Even in the case of SoftMAC devices, Operating Systems include the MAC layer handling in the drivers for the devices. This close connection if often required by the MAC layer protocols as they require low-level information about the physical links of the device.

In the current implementation of NCVS, 3 MAC-layers are at least partially implemented. Two of these implementations include the Carrier Sense Multiple Access (CSMA) derivatives used in 802.11 and Ethernet. Specifically these are CSMA/CA (Collision Avoidance) and CSMA/CD (Collision Detection) respectively. The third is a pass-through from Routing directly to Physical layers closely resembling ALOHA.

These 3 protocols were chosen for a variety of reasons including the commonality of Ethernet and 802.11 devices as well as ensuring the special features they require of the physical layer were implemented.

CSMA/CA Since 802.11 devices are unable to transmit and sense collisions at the same time, CSMA/CA instead tries to avoid the problem. This is done by listening for noise prior to sends, backoff randomization when invalid frames are detected, and other optional algorithmic solutions such as RTS/CTS (Ready To Send and Clear To Send.) As a protocol heavy in scheduling, it was actually the first implemented.

CSMA/CD While 802.11 devices were unable to detect collisions, Ethernet devices can. By detecting that a frame has collided mid-transmission, transmissions can be terminated without having to wait for a timeout to occur thus the interference period is greatly reduced. In order to support this mid-transmission termination of frames, special handing of transmission timing handling was required.

Prior to the implementation of this protocol, receiving nodes were tied up until the original transmission period (based on the original bytes expected to be sent) had terminated. In the re-writing of this timing handling, the implementation was moved from partially in Nodes and ChannelLinks to a state-machine within the ChannelLinks. This final design will be discussed in its later sub-section.

ALOHA Similar As was mentioned in the summary, ALOHA is basically the lack of MAC-layer thus frames are simply dropped onto the physical links whenever available. Instead of sensing for frame reception like the CSMA versions previously mentioned, ALOHA will send the next frame as soon as it's not transmitting a previous one. While not exactly ALOHA, NetDevices derived from the NetDevice base-class without a MAC protocol added simply bypass it. As such, it will report the interface is ready as long as it is not current transmitting data.

Routing The Routing subset of protocols handle the frame creation and handling above the L2 layer. Keeping in mind the long-term goal of portability, a rough design of a Netfilter-like API was implemented. Due to the complexity of NetFilters and the lack of time for development, the API varies from Linux Netfilters (thus cannot be directly transferred over.) However, within a semi-NetFilter class the Click Modular Router is found. Click Module Router

A very flexible and scripted software router is created with a graph-based architecture. This platform provides extremely simply Elements which are plumbed together to form extremely complex functions. In addition to providing an extremely wide variety of Routing functions, Click provides tracing, queuing, data injection, and much more. Click also provides WiFi elements via MadWifi elements [36] [37]. Using the nsclick API created for ns-2 [43] and ns-3 [49], simulation of complex tasks can easily be created.

Another key feature, and reason for including Click, is the portability of the scripts. It is possible to take the near-exact same script from NCVS to a Linux computer and have it actively perform the same. However, more than just Linux computers, OpenWRT, an open-source Linux distro for embedded routers and access points, also supports Click. Needless to say, but we will, this provides an extremely wide-range of validated, instant deployment of a developed protocol.

NetDevices

As a network simulation engine, it only makes sense to have network devices! The term NetDevice was chosen for the sake of differentiating C++ interfaces from network interfaces. In NCVS, NetDevices acts as a container for physical connections, channels, and other properties.

NetFeatures While some functions are likely common across NetDevices such as physical links, an unknown number of special properties are guaranteed to emerge. To enable quick expansion of NCVS without giant interfaces of unused functions to handle all possible properties, NetFeatures are introduced. In essence, NetFeatures are a contract that an interface is implemented for a given NetDevice. NetDevices provide an enumerated list of these contracts. Based on its supported features, a NetDevice class can utilize polymorphism and be dynamically cast for managers (and other classes) to access these unique functions

RF In the case of RF NetDevices, properties include Antennas (type, gain, and orientation on device), transmission power, receiver sensitivity, and so forth.

Channels

Channel classes act as an interface from NetDevices and their backing mediums. While the concept of channels is stable, the configurations and special information needed may change for mediums. Although both Ethernet and 802.11 mediums currently utilize the same generic Channel class, there may be a later requirement to create a unique class for each in order to facilitate optimizations. However, Mediums can correctly identify the Channels via a ResourceID as a reference in a medium lookup

In NCVS, Channels are also used to simplify non-common NetDevice tasks. An example of this is used in the transmission of an 802.11 frame. In reality, a NetDevice would adjust its frequency to a channel value and simply broadcast its transmission. To encourage future portability options, NCVS uses Channel classes to achieve a similar task. NetDevice select a Channel and simply call a send function. The Channel class then handles the actual data transmission through all the established ChannelLinks thus simulating the broadcast task.

ChannelLinks

This class acts as the connection from NetDevice-to-NetDevice in the simulation. Created by Mediums, ChannelLinks simulate interference, propagation delays, transmission times, and provide information about utilization to a specific neighboring device (where as NetDevices only know what they have sent all or received from any other NetDevice.)

Mediums

In order to hide the details for special mediums (especially shared-mediums like RF), the Medium class acts as a mini-Manager for its subset of NetDevices. Often there will be a direct relation between NetDevices and Mediums since Mediums are in charge of creating Channels (the NetDevice-to-Medium interface) based on how the protocol divides the connection options.

While non-shared mediums with Channels that don't interference with each other can take care of all their own ChannelLink creation (i.e. Ethernet), others may pass along data to a common Manager. As will be discussed later in this section, this is exactly what is done for all RF mediums currently implemented.

SimManager

With an overview of the various components used in NCVS, we will now look at the glue which ties them together into an advanced simulation engine. As the core this engine is the SimManager which acts as a registrar for all components. In addition to providing a means of lookup and memory management, SimManager also handles broadcasting notifications of component creation and destruction through the EventManager. This provides publishing classes a simple method monitor when their various subscribed components are being removed (and thus ensure safe memory access.) The ObjectManager (described in further detail in graphics class section) uses these notifications to create and destroy rendered objects.

Scheduler

Of course in an extreme time-dependent system such as networks, NCVS must provide an easy-to-use API which can optimize simulation performance. The implementation of the Schedule too in consider a number of factors in order to make future development easier and more versatile as well as provide a central means for optimization.

To start, the Scheduler needed to be easy for the developer to use. This functionality is incorporated into the Scheduler with the help of subscribing interface called ScheduledObjs. While a simple scheduler could just iterate over all subscribed objects at every time slice, this would be detrimental to performance. Due to the large number of subscribing objects and the relatively large gaps between tasks, the overhead for wasted virtual function calls and time comparisons quickly add up. Instead, an object implementing this interface can subscribe to the Scheduler in order to have its callback triggered based on an expiring timer. Furthermore, a number of subscription methods are provided to handle common functions such as reoccurring events, events in a period of time from current, and events at a specific time.

Each subscription is uniquely identified by the pointer to the subscribing object as well as an identifying number and time value. The inclusion of this event id simplifies the developers task by allowing quick lookups of what should be processed at based on the firing of an event. Also, this allows their state machines to not have to track whether they've already subscribed at a time or not. They can simply call to the Scheduler and allow it to do the work thus avoiding a significant duplication of code. While this identifier has a large number of possible uses, several classes in NCVS use it to tie to specific tasks or managed objects to be updated.

As was previously mentioned, many choices were for future optimization as well as future object development. While significant improvement in performance in achieved by skipping large time-blocks without events, a unified Scheduler also provides an opportunity for future multi-threading support by its innate workermaster queuing. As this is not currently implemented, it will be further discussed in the Future Works section.

Another key benefit to a unified scheduler is the ability to handle simulation time independent of the actual systems timer. This is important for a future headless-mode of NCVS (discussed in Future Works) and allows variable speed simulation in interactive-mode. This makes it extremely easy for NCVS users to focus on events at important time sequences. Included with current implementation is even the ability to step at microsecond and event granularity. This is a very important feature in the graphical interface which currently allows a user to see bits traverse link timing state machines thus watch collisions and other edge cases occur.

This ability to handle events independent of the system timing also allows the current single-threaded implementation to maintain a minimum frame rate without missing events. To achieve this, the scheduler monitors how much time it has spent in the current event handling loop and will return when simulation meets the expected delta in real-time or a maximum runtime has expired. Again with developers in mind, the Scheduler also needed to be able to handle its functions being called at any time during execution, including the iteration over the scheduled event collection. While this is a minute detail, it is yet another example of how NCVS decisions and design at focused on the easy of development without extensive understanding of the core simulator itself.

Animation While there is not a specific class called Animation, the topic is included here as yet another example of the fore-thought in the Scheduler making the creation of custom movement, data injection, and other time-based events. At the time of writing this paper, a Node Movement and Node Data Injection class was created in exactly this way. While some code is duplicated, each of these classes was implemented in no time. Future developers will likely create base classes which provide the infrastructure for common tasks like adding nodes to a class implementing the ScheduledObj interface.

Node Movement This simple class provides a means of randomizing node movement based on a set of variables.

- Decision Time (Min/Max): Time until a new set of random values is created
- Velocity (Min/Max): Speed at which the nodes will move
- Position Bounds (Min/Max): 3D points which define a box the nodes are restricted to.

Based on random values based on this set of conditions, a movement vector is created for each node which will be used to update the Node location for a period of time. Once that period has expired, a new vector and duration is created randomly.

Node Data Injection While having a similar purpose of the movement model, the data injection model acts like an application layer pushing data down into the L3 routing through a TAP interface. Data is then routed according to the L3 protocol assigned to the node.

- Decision Time (Min/Max): Time until a new set of random values is created
- Data Rate (Min/Max): Speed (in bps) to send data

While still choosing a random duration for update the other value, this class simply chooses a data rate with a set of bounds. Depending on the interval this recurring schedule object is set to, a packet of the necessary size is sent. Should the speed be small enough that a valid packet cannot be sent, a counter is used to keep track of how much data is queued to be sent.

Again, this is a simple, and easily created, class which provides a wide variety of testing possibilities.

EventManager

While the Scheduler takes care of time-based event handling and directly dependent classes take care of their children, there is often a need for global notification of changes. In addition to the previously mentioned class creation and destruction example for the SimManager, there are also events such as Node movement, Channel state changes, NetDevice parameter changes, and several others. Rather than each object having to maintain a list of dependents, it can leave it to the EventManager.

Any class wishing to receive these events simply implements the EventListener class and subscribes to the desired enumerated types. Although not currently implemented, it may be helpful (and an optimization) to include an option for source filters to these subscriptions. Similar to some optimizations added to the Scheduler, this would reduce the number of calls to EventListener classes and their checking if it's data they care about.

When events are broadcast, the type, source, and a variable pointer parameter are included. Based on this information, subscribed objects can further localize the event to a specific component of interest. For example, RF Mediums need to know when a node with a supported NetDevice is moved. By comparing the source field to an internal list of objects, the medium can update links as needed.

Although a manageable issue, the use of an enumerated type means that a list change may require a complete recompiling of NCVS to ensure enumerations are updated. As NCVS is currently small and doesn't use dynamically linked modules, this is not a problem. However it may be worth providing a means for string or GUID typing. This would allow the dynamic addition of new triggering events from future created modules.

Event Types At the time of writing this paper, the following are the enumerated types of broadcast events.

Name	Source	
NODE_ADD	SimManager inclusion of a new node	
NODE_REMOVE	SimManager removal of an existing node	
NODE_MOVE	Propagated from Node (or Antenna) events	
NETDEVICE_ADD	SimManager inclusion of a new NetDevice	
NETDEVICE_REMOVE	SimManager removal of an existing NetDevice	
NETDEVICE_MOVE	Physical change of location of parent Node	
NETDEVICE_PARAM_CHG	RF Transmitter or Receiver Level Changes	
CHANNEL_ADD	SimManager inclusion of a new Channel	
CHANNEL_REMOVE	SimManager removal of an existing Channel	
CHANNEL_STATE_CHG	A channel reporting it's state having changed	
LINK_ADD	SimManager inclusion of a new Link	
LINK_REMOVE	SimManager removal of an existing Link	
MEDIUM_ADD	SimManager inclusion of a new Medium	
MEDIUM_REMOVE	SimManager removal of an existing Medium	
SCHEDULER_PAUSE		
SCHEDULER_SPEED_CHG		

Table 3.1: Broadcast Event Enumerations

RF_MediumManager

Since frequencies used by various RF-based mediums will overlap, there is a need for a manager-of-managers to coordinate the creation interference links of non-compatible NetDevices. The RF_MediumManager further extends the task of a normal Medium by allowing mediums to register their channel frequency ranges and Channel instances. Based on this information, the RF_MediumManager can determine which channels can establish active links with each other and which will only interfere with each other. Additionally, a significant duplication of code is eliminated as all RF Link Creation (described below) can be handled by this one class.

When iterating through the set of channels within a collision domain, only those with matching source Mediums, Resource IDs, and within signal range are able to create active links. All other links within interference range are considered to be interfering with each other and frames transmitted will corrupt and collide with others in the domain.

3.4 Graphic Class Hierarchy

On the other side of the coin is the graphics system. As a core function of NCVS, this system of managers and objects provides the easy-to-extend visual and IO tasks. This section is primarily going to focus on the infrastructure managers of the graphics system rather than the current objects created. This is due to the number of visual objects created, the objects are the renderings of the simulation components, and finally complexity and importance of the manager tasks.

3.4.1 Overview

Like the simulation objects, we first overview the general tasks of the various graphics objects.

- **GLScheduler:** Scheduling for non-simulation timing which is handled at the graphical frame rate
- **IOManager:** A handler for all user input. Class also includes several interfaces to optimize the handling of various input types.
 - ClickableListener: Interface for clickable, 3D objects
 - ClickableMenuHandler: Interface for menu handling of clickable
 object dynamic menu

- KeyboardListener: Interface for keyboard events
- MouseListener: Interface for mouse click events
- WindowEventListener: Interface for handling window resizing events
- Renderer: Managing class for all GL and GLUT graphical output
 - Camera: Handles perspective changes for Renderer based on user input
 - RendObjs: Interface for 3D Rendered Objects
 - * Rendered Helpers: A set of classes added to reduce code duplication and make it easier to implement the various RendObj functions
 - MenuPanel: A base-class handling the common tree-hierarchy handling tasks for 2D menu boxes
 - * MenuPanelListener: Interface for clickable, 2D menus
 - * **MenuManager:** A special case of MenuPanel which assists when the first level of MenuPanel placement
- ObjectManager: Ties the SimManager to the graphical system

3.4.2 Class Details

With a basic outline of how the graphical system is divided, we will now dive into specific design decisions and implementation of the various components.

GLScheduler

Like the simulation Scheduler, the GLScheduler handles time-based events to subscribed ScheduleObjs interfaces. However they differ in that GLScheduler is based on GL frame updates and does not currently handle specific timing. Instead all subscribed classes are called with each interaction.

The differentiation between the simulation and graphical Scheduler is simply due to the different timing needs of the two components. While the simulation engine must trigger exact events, graphics updates are only to update visual elements are only going to be rendered once per frame anyway. One may ask why not just perform these scheduling tasks in the later described Renderer draw functions. The answer is simple, pre-planning for multi-threaded rendering where one thread will stage information for rendering. Since the entire application is only single-threaded, this feature is obviously not implemented yet.

Either way, a number of objects subscribe and unsubscribe from the GLScheduler based on their need to make visual changes. A few examples which use this functionality are Nodes when being moved, packet traces and they propagate, and packet collision & drop graphics.

IOManager

Of course for an interactive utility to be interactive, it requires a method for components receive input. Due to the varying types of input, and different needs from said input, several interfaces are supported by the IOManager. Should an object have a need, it can implement the various interfaces and subscribe accordingly. The next subsections are going to describe the provided functionality as well as any special handling the IOManager provides. Although not controlled by the IOManager itself, it's important to note that several functions are different when mouse-warping is enabled. As these functions are used by the Camera for camera "flight", IOManager simply ignores several events while the Camera has taken control. This includes 2D menu clicking, 3D clickable object check, and a few others minor cases.

ClickableListener A significant component of NCVS is the ability to click on objects, receiving information, and make changes as needed. To provide easy implementation for future developers, the SelectableObj interface was created to achieve the Object specific components and IOManager then handles the global, multi-selection details as well as menu creation. When mouse-warping is disabled, the mouse is activated and the following clickable logic is followed. Additionally, MenuPanels have priority over 3D objects and thus the current mouse position is checked with the MenuManager prior to 3D objects.

To start, clickable objects need to assist the IOManager and let it know if a point in 3D is touching said object. As any different shapes, sizes, and orientations are possible, IOManager simply passes it a point and expects a Boolean indicator if it's touching. To further assist developers avoid duplication of code with common shapes, RendObjHelpers include touch checking.

After an object is tested for its touching status, and it is touching, it will receive a notification of the button which triggered the event for special processing. Furthermore, its selected state is set and IOManager add it to an internal list of selected objects. This set of currently selected objects is then used by the dynamically created menu functions. Replicating the common interface of using the CTRL key to multi-select, IOManager takes care of all this logic to once again make easy extensibility a focus. Once IOManager has made the determination of which objects are selected, it will update the dynamic right-click menu based on an enumerated set of supported functions. By finding the intersection of all selected objects, a common set of functions are established. This method for creating the menu options allows common tasks like "Delete Object" or "Un-select All" possible across devices. This also provides flexibility across components of the same general type (such as NetDevices.) While many NetDevices may have common tasks, an easy example of differences is RF versus Ethernet. In the RF classes, there is a toggle for the rendering of the RF cloud where Ethernet obviously would not. Lastly, when a menu command is triggered, a subscribed ClickableMenuHandler will process all currently selected objects.

ClickableMenuHandler With a wide range of possible functions performed by the dynamic menu, a division was needed to avoid code and dependence bloat of IOManager. As such, an interface called ClickMenuHandler was added. Once a class implements the interface, it subscribes to the IOManager as the handler for a given task value. Since managers in the graphics domain often already have all the required information, they act as prime candidates for menu handling. Once a menu event is triggered, IOManager attempts to find a handler and calls it with the list of objects to perform the event on. So that a manager can process multiple events types, the menu event value is also included in the menu handler call.

KeyboardListener As a very common input method, IOManager supports the keyboard using the publisher-subscriber architecture. Classes wishing to receive notification of keyboard events can implement the KeyboardListener interface. As

keys are pressed (and released), an update is triggered to all subscribed objects.

In addition to the use of the interface, classes can also access a global mapping of keys to their states. Since the KeyboardListener only reports the key which triggered the event, it may be helpful to detect combinations (beyond modifier keys like Shift, CTRL, and ALT since these are included in the per-event notification.)

MouseListener Like the keyboard, the mouse is an important, and common, input method. The MouseListener interface provides two different callbacks, one for clicks and another for scroll-wheel movement. While this provides another ClikeableListener-like interface, MouseListener is not specific to just the object being touched.

Like the keyboard, the current mouse coordinates and warp state are stored in a global location so that objects can read them as needed.

WindowEventListener Simply, some classes (such as the Camera and Menu-Manager) require notification when the screen shape has changed. When such an event occurs, all subscribers are given the new pixel width and height of the screen.

Renderer

At the core of the graphic output is the Renderer class. As the name would suggest, it is in charge of frame rendering and acts as the publisher for all RenderedObjs. By providing a wrapper for OpenGL calls, the Renderer provides versatility for future development should there be a desire to extend utility to other platforms.

With extensibility in mind, the Renderer also provides a per-window subscription system so that sub-windows, and future multi-window support, can render different content. This allows different views, such as the top-down "radar" view, to show only the information which relays the desired information. Currently 3 subscription groups are used to support the main screen, radar view, and menus.

While the Renderer provides the core functionality, it also was designed to allow different Camera with different views and controls. Once the environment is setup for rendering, all subscribed objects are then iterated through and their draw functions called. Finally, menus are rendered as to overlay the renderings below.

Camera

As OpenGL provides an open framework for perspective and a user will want to be able to move what they are looking at, the Camera class was created. While there is currently only one camera implemented, the thought was to allow for different perspectives and, more importantly, different controls. The current Camera uses the typical WASD controls for movement with mouse (in warp mode) and arrow-keys providing look-around functionality. It may be desirable to add a future camera which would orbit around a point (such as a Node) instead of freefly. By providing camera changing support, each camera could remain simpler while providing multiple modes.

Rendered Objects

This section will be focused on the design decisions of RenderedObjs, how they tie into the system, how future development has been made easier, and a brief overview of components that have been implemented.

Rendering Helpers Again to make development easier, especially with Clickable objects, a number of rendering helpers were created for common shapes. While some of these shapes are simple, and already provided in the GLUT API, these helpers provide additional isTouching logic. Additionally GLUT and GLU do not always provide intuitive commands nor do their coordinate systems always tie to that of NCVS.

While the specific calls vary, in general there are two calls required by helpers, draw and eventClickedCheck. In the case of the draw, the center position and rotation has already been established by the caller. As such, only the size details are usually required. However, in the eventClickedCheck case, the position, rotation, size details, point to check, and tolerance all have to be passed in. These classes do follow a design, however, the varying parameters make it less conducive to the use of an interface. If future development greatly extends the use of these classes, it should be fairly easy to modify their design to accommodate an interface design.

At the time of writing this paper, 6 general helpers were provided:

- Cone
- Cube
- Cylinder

- Sphere
- Prism
- Isosurface

Since the first 5 are fairly intuitive we won't elaborate on them further. However, the isosurface helper is a powerful subsystem which is heavy used for the non-uniform shaped RF clouds of NCVS.

Isosurface Especially due to the non-uniform shapes of items like RF clouds, a helper was needed that could quickly and easily create triangle models based on a value field. Using Marching Cubes [39], this helper creates a surface along the partitioning cells on either side of a value. Whenever a cell has one or more corners along a partition, the said corners are sliced off creating triangles. To further smooth the surface of a low cell-count value field, the coordinates of the triangles are based on interpolating a midpoint between the partitioning values.

While this may seem a computationally intensive taste, several optimizations are added. To start, look-up tables provide code optimization for the vertex to corner removal algorithms. As such, the creating is simplified to cell vertex testing, a look-up for which corners were removed, and then a final look-up for which points triangles should be formed from. Using strategic array assignment, all these look-ups are simple static array indexes.

Further optimization is provided through the helpers caching of values. This allows the triangles calculated at different values to be reused until the value field is updated. In the case of RF clouds, the field is filled and several different values related to signal strength are computed. The resulting surfaces are then reused until factors like transmission power, background noise, or receiver sensitivity are changed.

Class	Purpose
World	Ground grid for node creation and RGB/XYZ scale
RO_Node	Rendering of the various node types which varying shapes
NetDevice	General NetDevice Rendering
NetDeviceRF	Extension of NetDevice to include RF Rendering
Antenna	Isotropic surface general handling for RF Cloud
Channel	Channel objects included state information
ChanLink	Rendering of links (and util.) between NetDevice Channels
Medium	Allow entire mediums to have changes made
PacketLink	Frame bits as they traverse the Link TX State Machine
PacketTrack	Traverse the links to illustrate data size being transferred
PacketCollision	Visual indicator for collisions which occur at a NetDevice
PacketDrop	Visual indicator for frames dropped at a NetDevice

Table 3.2: Rendered Object Classes

Examples

Menus

Menus provide an important interface for IO functions. In addition to providing information in all types of forms, they also allow the user to send commands to the simulator. While the decision was made to create a custom Menu system for the sake of library independence, quick prototyping, and overall versatility, several commonly found architectures were used as a template.

MenuPanel At the core of the NCVS menu system is the base class, Menu-Panel. This class provides the general purpose rendering with ties to its parent and children MenuPanels. A recursive architecture makes it extremely easy to create specialized panels by combining smaller sub-components. Due to the flexibility of this panel-based system, it makes sense that it is used by many (if not most) GUI development (examples include Android, iOS, WPF, and Swing.)

Using the reference to child and parents, components can be dynamically sized (and resized) based on static edge assignments. For example, you have a MenuPanel containing two child MenuPanels which should each take have the size. Both will have three edges attached to the parents edges (top, bottom, and left or right depending on sub-panel.) Next, the left and right menu have their common edge attached. Now if the sub-panels wish to get a static size, they will trigger a resize in the parent. Likewise, should the parent resize, it can intelligently adjust the sub-panels accordingly.

Another benefit of this hierarchy based panel design is the ability to quickly determine which panel is being interacted with. Since children always exist within its parent's region, a quick recursive call back is broken once the mouse's coordinates are outside of a clickable region.

In order to provide easy means for layout and dynamic resizing of elements, child panel edges can be assigned to the parents edges and to each other. With this information, objects will automatically adjust their placement in the rendering window as it is resized.

While layout is important, users need to be able to interact with the menu functions. For simple panels, this can be performed in the click checking of the current panel. This logic fails though when dealing with a hierarchy structure where a panel is a sub-element of a more complex panel. To handle such a case, the MenuPanelListener interface was added to enable panels to create custom events and publish those to subscribers.

Another consideration is how menus will often tie to the Rendered Objects of

simulation components. Although the MenuPanel base class does not implement this behavior by default, the ClickableListener interface used by several Rendered Objects provides an easy hook-point. Using that interface's isSelected functions, IOManager takes care of selections via MenuPanel. Panels can then provide direct access to the Rendered Objects and underlying simulation components. Again, this design maintains the separation between simulation and graphical environments.

MenuPanelListener Like most of the event interfaces of NCVS, the implementation is easy for classes that wish to support it but the benefits are huge. In the case of the MenuPanelListener, a single function is required. When subscribed to a publishing MenuPanel, the interface will receive an update when the last MenuPanel in its tree is clicked. This notification contains the source MenuPanel, an event id, and the button state. To provide flexibility, MenuPanels already contain the logic to add subscribers, set unique event ids, and for the event triggering itself.

MenuManager

With a background of menus covered, its import to note the specialized root of all nodes, the MenuManager. While a majority of the implementation is a direct extension of the MenuPanel base class, several differences exist.

- Clickable Region is the entire screen and is updated via the IOManager WindowEventListener interface
- No background is drawn, since it's the full screen

- Clickable Region checking forwards calls to children instead of using its own clickable region (since it's the full screen)
- A singleton is used to provide a global instance

ObjectManager

Briefly mentioned in the EventManager section, the ObjectManager handles the creation and destruction of graphical objects based on event updates broadcast from the SimManager.

To facilitate clean-up and other look-ups, the ObjectManager also maintains a mapping between Simulation Component classes and the Graphical Elements. These mappings are helpful for any objects listening to EventManager broadcasts about state changes since the source will be the simulation class, and not the visual item. Furthermore, several menus will provide system-wide adjustments such as disabling all RF clouds for a Channel, Medium, or Node.

It is also the task of the ObjectManager to handling the global layering of the mediums and channels. Whenever a new medium is added or removed, the ObjectManager will use its look-ups and update all elements vertical components to maintain a clean GUI.

3.5 Medium Channel-Link Creation

Since link handling is a key requirement of a simulator, and a significant feature of NCVS, this sub-section will dive into the various considerations used in the NCVS implementation. While the following sections are titled wired and RF since they are the common cases implemented in NCVS, several of the concepts tie to general handling of non-shared and shared mediums. However, RF links presents extra challenges due to them not being based on a physical link but instead of varying scale of connectivity, interference across device types, and complexity of wave phenomenon.

3.5.1 Wired

While wired links are not truly as simple as active or inactive, it is adequate to make this assumption for most simulations. Among other reasons, this makes link state calculations easy to determine. As NCVS is currently not handling waveform simulation, it uses this assumption for its Ethernet medium link creation. So long as the maximum link length is not exceeded and there's no wire-splicing, this seems to be a valid and reasonable assumption.

3.5.2 RF

As previously mentioned, RF is a completely different situation from wired links. In addition to distances being flexible, interference handling is a requirement for any kind of accurate collision domain handling. Instead of links simply being active or not, RF factors create directional link states and interference cases. In the latter, signal is still received at the destination, however it is not strong enough to provide reliable data.

Factors

While non-shared links are fairly straightforward, the shared-medium of RF devices create a number of variables (and therefore challenges) related to link state. This section is going to discuss at a high-level the factors which NCVS currently takes into consideration when updating a link state.

Transmission Power The transmission power of a radio is the number one controllable aspect, other than positioning, of RF links. While the power of various devices are limited based on regulation, power constraints, and equipment qualities, it is often not the optimum solution to set the transmit power at its maximum level either. In addition to often using more power to achieve the higher output, the interference range is also increased.

Antenna Gain As an under-simulated component of many simulators, the Antenna Gain model is key feature of NCVS for actual, non-ideal situation usecase testing. While ideal isotropic with perfect spherical antennas don't exist [41], it's understandable that many tools make this assumption due to the required infrastructure needed to use non-spherical radiation patterns. This one feature was the driving cause for a number of NCVS modifications. A short list includes:

- Node facing
- Antenna facing (relative to the Node)
- Antenna interface
- Antenna Rendering Object
- Isotropic Rendering Helper

- Quaternion implementation (for node-to-node direction)[22]
- And lots of supporting code

Although these inclusions added a significant time investment to the NCVS implementation, it adds a desirable feature which enables special edges cases to appear. A simple, but elegant, example is the antennas for mobile phones. Since there are not perfect isotropic antennas, the Wi-Fi, Bluetooth, and Mobile Network reception will differ the phone is held upright (as is done when talking on the phone) versus laying down (like is done during browsing.) As mobile devices were to be a central player in the two use-cases which drove NCVS development, removing this realistic component would likely result in a significant difference from simulation versus production.

Attenuation As radio waves propagate, their signal strength decreases due to various factors such as dispersion. Taking in consideration the transmission frequency and distance between nodes, NCVS currently uses a Free Space Attenuation model [41].

$$Attenuation = -(32.4 + (20.0 * \log(freq_{MHz})) + (20.0 * \log(dist_{Km})))$$
(3.1)

Receiver Sensitivity Based on the electronic characteristics of the receiver, a minimum Signal-To-Noise Ratio (SNR) is required between the incoming signal and interfering background noise. Frames which arrive between these thresholds will likely not be successful and will actually act as background noise. For this reason, NCVS treats receiving frames between these thresholds as a collision is another packet is received.

Background Noise A variety of sources can add to the background interference in any location. A few common examples are environment, microwaves, lighting, and other wireless devices.

While the current implementation of NCVS uses a static background noise value, several hooks have been included to allow dynamic and position specific noise modeling. These details will be further discussed in the Future Works section.

Calculations

Whenever a new Channel from an RF Medium is added or when the Event-Manager broadcasts a position or state update, the RF_MediumManager recalculates the link state of the affected channels. Using the dB values of the previously discussed factors, the directional link states are updated to either active, interface, or no link based on the following formula [41].

 $RxPower = TxPower_{dB} + Antenna_{Tx(dBi)} + Attenuation + Antenna_{Rx(dBi)}$ (3.2)

- If Rx Power >= Rx Sensitivity, then active link
- If Rx Power >= Background (but < Rx Sensitivity), then interference link
- Else, no link

Scope Based on the description above, a channel change results in a large number of channel-to-channel based comparisons due to needing to check every channel in the collision domain (caused by overlapping frequencies.) In fact, the

original calculation of all links is $O\{N^M\}$ operation where N is the number of channels and M is the subset within the collision domain.

To provide a rough estimate of the worst-case number of link calculations required, the use of an 802.11 US-region radio with 20 MHz bands and 1 MHz Bluetooth radio will be analyzed [24]. Because the 802.11 channels overlap with neighboring channels, a range of complexity is included due to the middle channels will cause more link calculations than outside channels which have fewer neighbors. Furthermore, ranges are much wider with Bluetooth radios included since each Bluetooth channel is non-overlapping and much smaller than 802.11 channels.

Configuration	Per-Channel	Per-NetDevice	Initial
10 Nodes with 2.4GHz 802.11	$45-81 \ (=9^{5-9})$	495-891	4950-8910
radios			
10 Nodes with 2.4GHz 802.11	9-281	90-1681	1800-33620
& Bluetooth radios			

Table 3.3: Rough Link Calculations

In the first case, the best case scenario is an edge 802.11 channel change with the worst case being a center channel. In the second case, the best case is when an edge Bluetooth channel changes and worst again being center 802.11 channel changing. Also worth note, these examples actually cause a much greater impact the simple comparison with the optimized version below. This is due to the fact that multiple nodes moving within the same time-value will trigger multiple re-calculations of link states.

Optimizations Throughout development, several optimizations have been added to NCVS in order to reduce the processing time required for link state updates.

While not all the optimizations completely eliminate a comparison, many reduce the frequency and enable the comparison to exist early. Although not a complete list, the major contributors include tracking the Channels previous state information, updating once per time value, avoiding active link calculation when across mediums, and proactive maintaining a list of channels within the collision domain.

Configuration	Per-Channel	Per-NetDevice	Initial
10 Nodes with 2.4GHz 802.11	$0-9 (=9^{0-1})$	>0 - <99	>0 - <990
radios			
10 Nodes with 2.4GHz 802.11	0-19	>0 - <209	>0 - <4180
& Bluetooth radios			

Table 3.4: Rough Link Calculations (Optmizied)

Note: These examples are actually on the high side due to the intelligent, per-time comparison logic which avoids the duplication of comparisons (much the same as how combinations result in a lower number than permutations.)

As the simulation size grow, even these reduced comparisons will still cause a significant impact. To compensate, a few unimplemented optimizations will be discussed in future works.

Directionality Caused by varying Tx Power, Rx Sensitivity, and varying background noise, it's possible to have an active link in one direction and in interference (or even non-link) in the other.

3.6 Data Transmission

Another significant portion of network simulation is how data transmission is handled. Using the links created in the previous sub-section, bits are transmitted from one node to another via a compatible channel. However in addition to handling successful transmissions, cross-channel and out-of-range RF interference creates the need to handle invalid data cases.

The following sub-sections will discuss how NCVS handles the timing, collisions, and special cases that are required for various protocols.

3.6.1 Timing

In the event handling for data transmission, there are two timing components to take into consideration, propagation delay and transmission time. The different combinations of these two elements provide the core for collision detection for both the transmitting and receiving node(s).

Propagation Delay

The time it takes a bit to move along the link at the speed-of-light from transmitter to receiver. At microsecond resolution, this delay accounts for a delay of 1 microsecond per 333 meters of separation. During this period at the beginning and end of packet transmission, the transmitter and receiver will have differing views of the line state (as will be discussed in more detail in the next section.)

Transmission Time

While propagation delay ties to node separation, transmission time related link speed and frame length. During the period of time, simulated bits are being actively transmitted along links.

3.6.2 Collisions

While collisions are reasonably easy to deal with in physical links, the primary use-cases for NCVS require valid simulation of collisions on RF links in order to provide valid results for a shared-medium.

In order to alleviate high CPU impact of handling link-speed transmissions, state machines on either side of the links keep track of the frame state on the wire. The next two sub-sections will discuss the relevance on the transmitter and receiver's perspectives.

Transmitter

While full-duplex connections are not concerned with receiving data while transmitting, half-duplex links must keep track of transmission collisions. In such cases, the line must be monitored for the transmission time. If an incoming frame notification is received during this period, the NetDevice class will mark it as a collision and handle the final receive frame as if it were corrupt. The reason behind this is the inability for many half-duplex devices, especially RF, to receive while transmitting. Since this is not always the case, such as an Ethernet NetDevice which supports full-duplex running in a half-duplex mode, the current implementation can be adjusted on a per-NetDevice level. Worth note, the current implementation has the ability to cancel a queue transmission. This feature, discussed in an upcoming section, was specifically implemented to support transmission collision handling for NetDevices, such as Ethernet with CSMA/CD, which can detect an incoming frame while transmitting.

Receiver

After the link state machine's propagation delay timer has expired, it will notify the receiver that a frame in starting. At the end of the transmission timer, the link forwards the actual frame information. If the link is an interference link, the frame information is simply NULL. Should the receiver be notified of another frame starting between another start and completion call, it is deemed to be a collision.

Partial Transmissions

As there are cases where a NetDevice may wish to cancel a transmission in progress, such as CSMA/CD on half-duplex Ethernet devices, it made sense to add the functionality to current implementation.

Since a state machine is used to more efficiently handle frame transmission schedule, NetDevices forward the request to cancel a send to all links. The link in turn will check their transmission state machine and make adjustments to the current transmission (if one exists.) The link state machine can determine if a transmission is in progress if less time than the transmission time has elapsed on the last frame sent. Luckily this can be confined to just the last packet since each link if unique between NetDevice Channels and each NetDevice can only be transmitting a packet at a time.

To modify the frame, the data itself is cleared, to simulate an invalid frame, and the transmission time to update to the time that has elapsed. The partial frame will then traverse the link like interference. By handling partial transmissions in the way, the receiver can correctly handle the effects of the partial frame on its collision domain based on a correct period of time.

3.6.3 Implementation Limitations

Due to the current design of frames being handled in the links, there is a short period of time in which a receiver could miss partial frames. There are two cases where this can happen, creation of a link while a NetDevice is "currently" transmitting a frame and destruction of a link while there are still packets in its transmission state machine. While the maximum time is less than the propagation delay plus transmission time, it is something requiring attention in future development. Another detail in this edge case is that frames that fall into the category are partial frames (thus would be handled as interference) or interference frames.

3.7 Channel Switching

A number of devices, especially RF-based, take a short time to change from one frequency to another [33][23]. While this period can range from 10's of microseconds to 10's of milliseconds, NetDevices will not receive valid data during this period. While many devices tend to stay at a given channel for an extended period of time, procedures like Wi-Fi scanning and Bluetooth frequency hopping needs to take in consideration this delay. Furthermore, the auto-configuration use-case which drove part of NCVS development included regular scanning and schedule-based multi-channel communication. Without considering the delay between channels, prototype performance would greatly differ from production.

In current implementation in the RF NetDevice base class, a channel change triggers the original channel to be disabled and a timer to be scheduled for the switch-delay time. When the Scheduler notifies the NetDevice of the timers expiration, the desired channel is activated and links established. To handle the characteristics of various NetDevices, the switch-delay is a configurable value.

While this implementation does handle the switch delay, it does suffer from the link creation issue previously mentioned in the Data Transmission section.

3.8 Special Scheduling Considerations

As with any event-based scheduling system, there are a number of challenges having to do with concurrency. While a multi-threaded system may allow the OS scheduling to handle the issue, a simulation engine should desire a predetermined outcome. Although the discreet time-step helps with some of these issues, the handling of dependent events with the same time value still exists.

An example of such a challenge is in packet collision handling. In this situation, a frame received between another frame start and completion is defined as a collision. Thus two events, frame 2 start and frame 1 completion, received in the same time value has very different results depending on the order processed. If processed frame 2 start before frame 1 completion, a collision is record. However, if frame 1 completion is processed before frame 2 start, both succeed.
To solve this issue, a priority scheduler could be used where completion events have a greater priority than start events. However, at the time writing this paper, NCVS does not include such a system. Instead, the protocols utilize the fact that the Scheduler ensures that any event scheduled added to the end of the time value's set. If they detect that they have a dependent operation, then they are able to re-queue their scheduling which ensures the other operation is performed. While this is functional, later development will likely want to establish the clear priority based system previously described.

3.9 Frame Data Handling

While NCVS will likely be run on fairly powerful systems, it's always important to keep memory access in mind. Especially true with repetitious items like the data handling for NetDevice frame transmission, small optimizations can have a large impact on the system. Due to the multi-stage scheduling of a frame being transmitted, not only is memory allocation a user of resource, but the tracking for clean-up can be a challenge.

NCVS takes care of this issue by using a shallow-copying template object for frames. Then each stage can shallow copy the object if it needs it longer than the function call. When it is done with the object, it simply releases its reference. These two actions cause a reference count to increase and decrease which results in completely deletion of the underlying data when the reference count zeros.

Chapter 4

Validation

As with all testing and validation tools, they themselves must be validated prior to their use. Each tool may be created with good purpose in mind, however without sufficient validation the lack of confidence in their output often results in the projects dying. In an effort to show the stability and validity of NCVS, this section will attempt to compare it with NS-3 in order to isolate their differences.

While the versatility and flexibility of NS-3 and NCVS are key concepts in their development, these same concepts create an immense challenge when trying to run similar tests for benchmarking. Not only are there variations in the output, there's a large variety of propagation, RF, MAC, and other component flavors. As a result of the large pool of configurations possible, the task of devising the experiment is more difficult than the analysis itself.

4.1 Physical Layer (OSI-Layer 1) Isolation

In the OSI model, the physical layer relates to the bits being transmitted "on the wire." This includes the various transmission characteristics as well as modulation and other medium related connection details.

4.1.1 Summary

This test should compare how the transmission timing is handled in the two utilities and what factors influence such timing. The goal will be to validate the medium and link handling similarities between NCVS and NS-3.

Propagation Delay

As was previously discussed in the NCVS description, propagation delay is the time from what a packet first starts transmitting to when it's received at the other side. This subset of the test will analyze how this time varies over different distances between two nodes to confirm the assumptions made in NCVS were also made in NS-3.

Transmission Time

This next subset of tests looks at how NS-3 handles the period of time from when a data frame starts leaving to when it finishes sending the transmitter. Reviewing the output pcap files of both the transmitter and receiver at different propagation delays will isolate the transmission time.

4.1.2 Procedure

To isolate the L1 characteristics, two nodes will be set at a fixed distance with only one node constantly transmitting frames. NCVS has the node distance set at a distance that would cause a 1 microsecond propagation delay (350 meters.) As NS-3 doesn't take into consider node placement, the propagation delay setting is manually set to 1 microsecond to match NCVS. In both utilities, the link speed is set to 1 Mbps with 256 byte frames injected. Since only one node is transmitting, collisions should not occur and it should be possible to determine the total transmission timing based on a timestamp comparison of packet captures from the transmitting and receiving nodes.

Since both NS-3 and NCVS utilize the nsclick [43] API, the data injection and packet captures will be a function of a common Click script. Each utility will use the simplest configuration possible for an Ethernet link between two nodes. While both packets from both simulators will move their respective MAC layer protocols, the lack of collisions should cause any timing in such MAC layers to remain the same due to no need for random backoff.

4.1.3 Results

As no random backoff is triggered in this experiment, it is expected that all runs of the same experiment in a given simulator will be identical. Furthermore, the total transmission time should only include the propagation and transmission time. This is seen in NCVS with each packet completing 1954 microseconds after transmission starts. This works out as expected with 1 microsecond propagation delay and 1953 microsecond transmission time. The same identical behavior is also seen in NS-3, however, with a 1984 microsecond delay. The difference is that NS-3 correctly added the 4 byte Frame Checksum (FCS) where NCVS is not. When accounting for this 4 bytes difference, NS-3 indicates an identical total transmission time of 1954 microseconds. The 30 microsecond is consistent with the transmission timing of 4 bytes (32 bits) at 1 Mbps link speed.

While the results are common for the 1 microsecond propagation delay, NCVS times are based on position rather than a static variable like NS-3. Thus, as nodes spacing is decreased and increased, the timing automatically adjusts as expected based on the distance divided by the propagation speed, currently assumed to be the speed of light.

4.2 CSMA/CD (OSI-Layer 2) Isolation

The MAC layer of the OSI model includes a number of tasks including sharing the mediums, detecting (and possibly correcting) errors, and assisting with flow control. While CSMA include different sub-types, Ethernet networks use the Collision Detection scheme to actively sense collisions and terminate packet transmission if they would collide.

4.2.1 Summary

In the isolation of layer 2 differences, the results are specific to the MAC classes used. Of course this presents a challenge due to the partial implementation of the CSMA/CD specification in NCVS. Either way, the factors which should be evaluated for CSMA variants include the partial packet transmission and collision backoff algorithms.

Partial Packet Transmission

Since Ethernet, using CSMA/CD, can detect when an incoming frame would collide, it will actively stop transmitting the frame and reattempt when the line is again sensed to be free. As such, the entire packet is not transmitted allowing the collision window to shrink and allowing quicker recovery.

Collision Backoff

In CSMA-based MAC layers, a statistically random backoff model is used to allow the shared access to the medium. Due to the randomization used, special considerations will need to be used in the analysis of the backoff timing analysis.

4.2.2 Procedure

Like in the L1 isolation, two nodes will be used at a fixed distanced with a fixed 1Mbps link. However, this experiment will have transmissions from both nodes and link at half-duplex to create a collision domain. As this will cause collisions, both partial transmission and random backoff algorithms will be triggered. Due to this randomization, multiple runs will be executed and averages analyzed.

4.2.3 Results

While deterministic results are often desired, it was expected that the random backoff algorithms would introduce some variance in the results. After running 5 identical runs in both NCVS and NS-3, only NCVS exhibited this expected variance. This would suggest that CSMA class of NS-3 either does not use an exponential random backoff, or, it has a pre-set seed for consistency. Also worth note is that although the source and destinations of NCVS frames may vary, the total number of frames remained the same across all runs. As discussed in the Future Work section, NCVS will later need to have a random generation class created to enable both deterministic and random modes.

Run	Total	Sent	Recv	Time First Node 1 Recv
All	1463	781	682	0:0.026509

Table 4.1: L2 Experiments - PCAP summary (NS-3)

Run	Total	Sent	Recv	Time First Node 1 Recv
1	1563	600	936	0:0.515903
2	1563	883	653	0:0.314641
3	1563	658	878	0:0.062575
4	1563	1224	312	0:1.526121
5	1563	662	874	0:1.043483
Avg	1563	805	731	0:0.692545

 Table 4.2: L2 Experiments - PCAP summary (NCVS)

In both simulators, we see a fair sharing of the medium between the two nodes. However, NS-3 traces show a better, interlaced distribution of the transmitting node where as NCVS traces showed a greater clumping of each nodes frame transmissions.

While the CSMA/CD MAC layer of NCVS still needs some refining, it is important to also note that NS-3 uses a generic CSMA model without collision detection and partial transmission handling. As such, frames will appear to create a collision domain for a longer period of time in NS-3 than NCVS.

4.3 Experiment Discussion

While running the validation experiments above, a number of discoveries emerged. This subsection will discuss of a couple of such as a bug discovered in the nsclick API and the consequences on experiment runtime.

4.3.1 Click Timing

In the main edition of the Click Modular Router, a large number of elements already exist for data injection including pcap reading and generic rate or time based packet creation. While setting up these experiments, it was discovered there is a bug causing events not to update the event scheduling used by the nsclick API. While the ICMPPingSource element would correct send pings every 1 second, elements such as the FromDump and RatedSource would not send any packets after the first. When looking into the issue, it was discovered that some objects use a Task class to interact with the Click timers. Elements using the Task class didn't correctly trigger the "update at xx:xx" call which Click makes to the simulator event system through the nsclick API. As such, the event based simulators would not make the call to update the Click router as needed.

To overcome this issue, either Click configurations could use elements which did not use the Task class, or, they could include an element which could force the router to be updated via the API. In the experiments above, the Timer-Source element was scheduled to trigger router updates every microsecond. With this addition, elements using the Task class would correctly process with the router update as if they had correctly scheduled it themselves. However, such a workaround exposed efficiency issues in NS-3

4.3.2 Simulator Runtime

In order to minimize the changes made to NS-3, NCVS, or Click while validating NCVS, the above workaround of including a non-Task containing element to trigger Click updates was used. While this caused the Click scripts to function as expected, NS-3 runtimes were dramatically affected. NCVS ran the simple experiments in near real-time, however, NS-3 runs took 700x (for single transmitter) and 1200x (for both transmitting) longer to process the same scripts.

While normal use of NS-3 may not include updates every microsecond, such a performance difference is worth noting. Furthermore, the workaround was required to make the nsclick API, created specifically for NS-2 and NS-3, to correctly function with all elements. Although this issue would be resolved by update the nsclick API within Click, it does still illustrate a scheduling inefficiency within NS-3 itself which was avoided in the implementation of NCVS.

Chapter 5

Future Work

While the design of NCVS has already proven to be a versatile, it is a unique and young utility. Without the benefit of a long history and on-going user contribution, NCVS lacks optimizations, tested user experiences, more advanced simulation functions, and large pool of protocol and device classes found with other simulators. This section is going to outline a number of these items which are known opportunities that would add to the functionality, and therefore benefit, of NCVS.

5.1 Optimizations

As the long-term goal for NCVS would be large scale simulations, a significant number of optimizations need to be included to reduce per-component costs. In addition to reduce the component costs, there are a number of opportunities to maximize the multi-core, multithreading systems that are common today.

5.1.1 Multi-Threaded

While systems of the past were single core systems which had to deal with regular context switching issues, today's systems have focused on parallelism. As a result, serialized applications often barely touch the performance capabilities of their platforms. Although the application performance is greatly limited by serial coding techniques, a great deal of simplicity and reduction in edge cases is achieved.

As the same time, bad parallel code architecture can result in worse performance than a serialized application. Common causes to such failure are over locking (sometimes to the extreme of causing serialization), I/O thrashing, and under-utilizing the time scheduled. The section will discuss some of these concerns with concurrency and what will be required for future implementation.

Priority & Dependence Handling

As one of the primary goals of NCVS is ease of extension, a simple and clear system needs to be established for the partial ordering of events. In some cases, these events will function completely independent of each other thus can be concurrent. In other cases, dependence will exist and tasks will need to be serialized. While the microsecond granularity provides a time-based partial ordering, the sub-granularity ordering can greatly affect the predetermined outcome which is a strongly desired property of simulations.

In essence, this is the same scheduling issues previously discussed with the example regarding transmission completion and new transmission happening in the same time slot. While this particular case is currently handled by rescheduling the conflicting item to the end of the time value, thus allowing the dependent event to be handled, there are a number of significant assumptions and issues. The biggest failure of this method is requirement to the developer to understand how the Scheduler functions instead of providing a clear API which establishes order. This problem of requiring experience with the simulation kernel implementation is part of the difficulty with NS-2 which should not repeat with NCVS. Another set of issues issue are the lack of ability to handle inter-component time issues and does it provide a significant opportunity for maximizing concurrency by detecting serial chains.

Locking

Obviously there is a heavy interaction between components in NCVS. While this interaction is required for the simulation, it provides challenges on the simulator. Namely, component properties which may be accessed by multiple other components need to be protected from corruption and invalidation. Furthermore, if multiple objects edit another, there needs to be a clear understanding of what the final output will be. These are not new problems, however, they are still a challenge. A few methods commonly used would seem to apply well to NCVS, Worker-Master Queuing and Multi-Buffer Switching.

Worker-Master Queuing Luckily the choice to use the publisher-subscriber architecture in NCVS provides a good interface point to insert a message system which can facilitate a Worker-Master threading model. In this model, a Master (in this case a combination of EventManager and Scheduler) can orchestrate the tasking of Workers (threads) with independent tasks. With the extra information available, the Master could be facilitated with the ability to determine a component level partial ordering thus limit locking to component level. While there is obviously additional analysis needed, there's a fairly good chance for sub-component partial ordering while still maintaining ease of effort on the developer.

Multi-Buffer Switching For some objects, only a very rough, and relatively infrequent, granularity of locking is required. The most obvious example of this is the graphics-to-simulation interactions. While the simulation subsystem is scheduled at microsecond granularity, the graphics subsystem is scheduled in the 1's to 10's of milliseconds. Due to the 3 orders-of-magnitude difference of graphics pulling information, it would desirable to not have heavy, per-function locking implemented if avoidable.

Another issue with the graphics-to-simulation interacts is the need for graphics to retrieve information from every component in the system. Furthermore, the information retrieved should be valid, at the time of capture, across the entire system. Not only does this quire per-function locking, but requires the system to provide a means of providing a snapshot. This could be achieved by preventing the Master from allocating tasks to Workers during the render state, however, the render time is significant and would results in wasted thread cycles.

To avoid the requirements for heavy locking and to provide a snapshot, the use of a barrier and multi-buffer system would make sense. In such a system, the simulation would maintain two buffers, an active one and an inactive one. At some granularity, based on time or an event, it would pause operations and copy the active buffer to the inactive. After this short copy, it would resume with services. While this still does result in a short pause, the copying operation will be much quicker than the rendering. Then a thread-safe accessor to the inactive buffer can be used by the graphics subsystem to retrieve the system snapshot. Of less importance, two thread-safe accessor methods include the use a single mutex for the graphic system to copy the data or an additional buffer with an atomic swap to toggle between two buffers.

In order for such a system to function most efficiently, each component would provide a data context bundle for all its state shared state information. Based on this, static class accessors would decrypt the information for the graphics subsystem while still maintaining component separation. While it would take a significant amount of time for initial implementation, later development should be able to use templates for much quicker creation. Furthermore, entire classes could be used as the data bundles with copy constructors, however, special care would have to be taken for pointers.

Apart from optimizing simulation performance, this modification would allow for modularity of components across systems. Likely the initial use would be the separation of rendering and simulation to separate computers. If these two components were separated, it would even be easy to support multiple users accessing the same simulation. However with the long term desire to expand NCVS to larger and more complex simulations, such logic would help support multi-system simulation.

5.1.2 **RF Link-State Calculations**

As was discussed in Medium Channel-Link Creation of the Implementation section, the non-optimized $O\{N^M\}$ operation was a problem even at small scale. However, even with the optimizations already included, the scale of these calculations will likely set the limit for simulation size. While not yet implemented, here presents two optimizations which should drastically help with performance.

Maximum RF Range

Using a hook already included in the Antenna model, maximum gain, it should be possible to calculate a maximum encompassing sphere for the devices transmit power and a minimum background noise. Like the early comparison for inactive channels, this maximum radius can be compared to the distance between two nodes and early return should the distance be greater than the radius. Such an optimization is especially helpful in less dense, but high channel count, simulations. In addition to avoiding the orientation calculations, this excludes the need to call for a specific reading from the Antenna model, power additions, and various comparisons performed on every check. Furthermore, the maximum radius can be stored and updated only when the Antenna changes, transmission power, or minimal receiver power change.

The one drawback to currently using a single maximum gain value is with highly directional antennas since their encompassing spheres will be rather large. When this same issue is dealt with in graphics, an alternative bound box can be used instead of a sphere. However, spheres are often preferred due to the simple radial check instead of requiring several comparisons.

Independent Multithreading Opportunity

This is a perfect example of how multithreading could be used for large numbers of independent operations. As all calculations are independent of each other, various threads could concurrently process them. Using a barrier at the end of calculation, a multi-threaded version of this comparison could be dropped into the current implementation without other modifications.

Additionally, the repetitious nature of the calculations lends themselves nicely

to Graphics Processing Unit (GPU) processing. GPUs are able to process huge amounts of data when it comes to highly concurrent, non-branch processing such as this. With it becoming more common for systems to include basic GPU units and the horizon showing signs of hybrid-CPU/GPU units, these calculations may become a much less significant of an issue.

5.1.3 Perspective Based Rendering

Due to the large complexity of the simulation system, a number of graphical optimizations are yet to be implemented in NCVS. Currently all objects are rendered at their set quality factors without consideration of where the camera is. By performing a few checks prior to adding content to the graphics pipeline, a reasonable improvement will be achieved through. Furthermore, this will help with scaling issues that will occur, especially as the number of link and packet objects increase.

View Frustum Culling

In order to render objects from their 3D-world to 2D-screen locations, the projection matrix created by the Camera class is used. From this matrix, a 6-side prism (with the sides being top, bottom, left, right, near, and far) is created. When an object is outside of this prism, the object is not rendered. While this is our goal as well, we can use a bounding volume, like those discussed in maximum RF range optimization, to simplify the task. Thus instead of having to check every triangle that makes up an object with the prism, VFC simply checks a sphere (or box) with the prism.

A few items to note include that it is possible for every object to still be in the

view frustum (prism) and the separation of graphics scheduling versus rendering. While the first item will be discussed in the next subsection, the first ties back to the discussion about the GLScheduler. Here's a case where the forward thinking to separate graphic scheduling from rendering becomes more important since objects will (likely) still need to be updated whether rendered or not.

Variable Quality Rendering

While VFC was able to optimize object in-view checking, it's important to consider how distant objects are rendered. Depending on its original size and distance, it's possible to have a complex object appear as a single pixel on the screen. Although we're lucky that the graphics pipeline takes care of such issues, it performs the operations on a triangle-by-triangle basis. As such, it would be desirable to have less complex objects the further away the camera is from the object. While this minor per-object reduction may affect the CPU load much, the data no longer has to be transmitted to the graphics pipeline nor does the pipeline have to process it. Also, this reduction is resolution would be done after VFC and only on objects which can be simplified.

Currently the most complexity compared to size object is the node spheres. Due to their small size, it's possibly that the 100+ triangles which make up the sphere render to a small set of pixels. It's very possible that future objects will be more complex. Perhaps the user would like to see a person to illustrate a phone, or, perhaps the user creates a new way of perceiving data which has a higher complexity.

Although Variable Quality Rendering is not used on the objects themselves, the camera-to-object distance calculation is already used on Channels and NetDevices to pop-up the name tags when near. As such, little work should be needed to extend it to the objects themselves.

5.1.4 2D & 3D Click Detection

In order to provide the I/O support for mouse events, every click triggers the MenuManager recursively checks its 2D menus and the IOManager iterates though and 3D objects. While the 2D menus' bounding box space makes this determination quickly and another optimization ignores 3D objects if a menu is found, there are still a number of cycles taken in these tasks. Especially as the number of Clickable 3D objects increases, this task could result in a noticeable pause. Luckily the 3D object intersection testing presents another case while independent tasks enable multitasking with little effort. Again, this is a small optimization but so would be the effort required once some multithreading support is added.

5.2 User Experience

With the NCVS focus on user interaction, user experience improvements are just as important as the simulation details. While there are likely several modifications to GUI that will eventually be needed, there are three important cases. First, current implementation does not handle multiple NetDevices with the same medium on the same node well. Along the same lines, it would be insightful to break the Medium, NetDevice, and Channel vertical offset logic into a separate class. Second, there are a number of graphical building blocks needed to support additional functionality. Third, the Click Modular Router should be modifiable, both automated and manually, via the GUI.

5.2.1 Node Sub-Component Placement

A significant issue when handling any GUI is how to display the information in an intuitive, clear form. Unfortunately the unexpected edge-cases often complicate original concepts and thus have to be reviewed from time to time. While the next two sub-sections relate to each other, the first is how resolve the current problem with multiple like-Medium NetDevices whereas the second is a more general concept of flexible, on-the-fly placement adjustment.

Multiple, Same-Medium NetDevices

Currently NetDevices and their Channels are organized based on the supporting Medium. The thought behind this was to create a clear tier effect for similar devices. Unfortunately when multiple NetDevices, backed by the same Medium, are attached to the same Node, they end up stacking up and being unclickable via the object. While many systems dont have this problem, servers and network infrastructure often require multiple NICs of the same type. They can still be accessed via the Node menus, however, a clear indication of multi-devices should be present. While the next sub-section will eventually include such logic, current implementation could move the NetDevice cylinder from the Node line to horizon circle of orbiting devices. As more like-devices are added, their orbit could grow in radius or multiple orbits. This would allow each to be individually clickable and still maintain the vertical laying.

Component Placement Class

While handling horizontal placement to avoid overlap is an issue in and of itself, interesting characteristics would likely emerge by adjusting the vertical placement model of sub-components. The current method of stacking the objects related to a Medium creates a distinction between mediums and devices. However, it would also be interesting to be able to stack channels based on their shared-medium characteristics. Directly referencing RF channels, this would provide a clearer picture interfering channels.

The first thought that comes to mind is horizontally offsetting NetDevices into an orbit pattern, like mentioned above. Then channels could be vertically placed, and sized, relative to their RF frequencies. This would also support the issue above by just including multiple NetDevices from the same Medium as another device in the orbit(s).

5.2.2 Menu Elements & Support

While Future Work sections usually focus on technical components, a graphic environment increases the importance of the user experience. Often in the creation of that GUI experience is the creation of various smaller items which are thoroughly reused. Though not as significant of a need, the following visual elements will provide support for on-going development and thus are included.

Animation Interactions

Another common function that current does not have a GUI is Node Animation. As previously discussed, there is currently no interface simulator for Node Animation, instead, the current animation simple extends the ScheduledObj class for scheduling. Since there will likely be more than just Data Injection and Movement Node Animations, an interface (and supporting GUI element) need to be a added. While each Animation class would require customization of the GUI element, the base functionality for adding and removing nodes will (likely) be consistent.

TextBox and Console Elements

In support of several user-defined options, the infrastructure for typed input needs to be added to IOManager and menu elements. While this is similar to the KeyboardListener already in place, handling of a future ConsoleListener would need to disable regular keyboard events so typed input wouldn't trigger other events. Of less immediate need would then be validation rules (likely a RegEx string) to ensure valid text was entered.

Examples of use:

- NetDevice MAC Addressing
- NetDevice IP/Subnet Entry
- Click Console
- Annotations

5.2.3 Dynamic Click Configuration Creation

Using the flexible Click Modular Router creates a versatile, scripting interface for creating advanced, portable protocols from simple blocks. Since it doesn't require re-compiling, and allows on-the-fly changes, a GUI interface to each node's scripts would make sense. Unfortunately at the time of writing this, such support is not provided. Instead, an external text editor is used and simulation re-started from scratch. Such an interface could be broken down into two parts, Automated Modification, based on a template for various NetDevices, and manual editing from within NCVS.

Automated Modification for NetDevice Addition/Removal

Click provides a number of useful constructs to be able to quickly create functional Routers. While you can create new elements in C, the scripting language also provides support for in-script block creation. When well developed, these blocks can act as a complete device which simply needs instantiation. Since all the instantiated blocks are tested when the script runs, it requires the node to have a valid NetDevice relating to the string provided. This would be fine for production, however, the NCVS prototyping, interactive interface allows the addition and removal of NetDevices on-the-fly.

In order to support the NetDevice modification done in the GUI, the user needs NCVS to dynamically update the Click scripts with the NetDevice instantiations of blocks. However to best achieve this is still unknown since there can be different script blocks for different types of NetDevice. A solution may be to provide a script block name along with the NetDevice instantiation. Alternatively an additional menu items could be populated with block choices (based on the current click router used) when the device in not referenced in the current script and a remove reference option it does have a reference. Either way, this will require both simulation and graphical interface changes.

In-Utility Scripting

While automatic modification is helpful in many cases, it is also reasonable that the user will want to be able to adjust blocks in script form. While this provides new functionality, it also would act as a stop-gap for the Automated Modification issue by allowing manual addition and remove.

The benefit on this feature is still to be seen. Especially without a script parser to ensure correct functionality, a minor error may result in the simulation closes. In the long-term, such a parser would be added so that a graphical view of a script could also be seen. This would also provide a helpful interface for interacting with Click handlers, the programmatic access to the elements.

In any case, such scripts would need to be saved in order to be loaded by Click and to allow the user to have a copy of what was run.

5.2.4 Routing Topology Highlighting

An entire subset of functionality is current missing from NCVS to support interactive route look-up. While packets are correctly routed via Click, the hooks to access its routing table outside of packet routing is not current used. Once this has been corrected, the user should be able to enable several route highlighting tasks. While there are others, two include Source-to-Destination Node Route and Source Reachable Nodes.

Source-Destination Route

In order to receive a clear visual indication of how a packet would travel if transmitted, it would be convenient to have link highlighting based on a source and destination address. In addition to seeing how a packet would travel at the current moment, leaving the highlighting active during motion models would clearly show when packets would be dropped due to the lack of a route.

Source Reachable Nodes

The development of multi-hop networks enables a number of significant network optimizations at the cost of complexity. A challenge obviously faced will be the developers need to see how available routes are updated as a node is added and removed. An equivalent feature to this reachable node feature would be source-destination routes to every possible address. While implementing the reachable nodes feature in this method would achieve the task, there would also be a high system load required to perform the individual checks. Instead, it would be better to access all Click routing tables and create a master mapping.

5.3 Simulation Expansion

Although optimizations will be an on-going task and the user experience is an important feature of NCVS, the core simulation accuracy and flexibility is of high priority.

This includes continued development of more complex RF characteristics and modeling. Using the extra information provided by the addition modeling, addition error modeling can be incorporated to data transmissions. It's also important to include easy methods and examples for common tasks like frequency scanning. Additionally, the power restrictions which limit connectivity and mobility opportunities should also be included. Based on the abilities already present in NCVS, other modes of use for the simulation engine and graphical interface should be considered.

5.3.1 RF Simulation Expansion

As has been discussed multiple times already, without a physical wire isolating communication, the general RF medium provides a number of issues. While assumptions and approximations are frequently used to avoid heavy computation, other factors are simply ignored due to a lack of understanding. Some omissions result in a little variation between prototype simulation and production release thus are reasonable to prioritize lower. Other details play a significant factor in wireless network performance.

This section is going to overview a number of these details and their importance on NCVS. These areas will include non-network objects, electromagnetic properties, carrier waves, and forward error correction.

Non-Network Object

Although a network simulator is focused on the relation between devices, the dependence on RF devices to their environment requires suggests that items outside of the network domain should be considered.

Moving forward, we'll separate the non-network objects into two sub-categories, obstacles, as physical objects which inhibit RF, and emitters, as focused background noise makers.

Obstacles As will be discussed in an upcoming section, multiple wave characteristics are affected by objects they come into contact with. Especially true as wave frequencies increase, objects reduce or completely block the propagation of the signal. To avoid complication, many simulations completely ignore common objects such as walls, terrain, and even the people using the devices. However, a number of protocols used for mesh networking and Dynamic Frequency Selection are significant affected by such omissions

In order to minimize the difference between simulation and production, consideration should be made for obstacle affects on the signals involved in protocols and NetDevices functionality. While low frequency radios are able to handle objects with better success, waves transmitted by higher frequency radios will often decay on impact.

Fresnel Zones A common approximation for signal changes due to obstacles is the use of ellipsoids of various volumes, defined by equations, from source to destination. A relation exists been the calculation of the largest ellipsoid, or Fresnel Zone, which is unimpeded by any obstacle and the resulting signal.

Extending the bounding volume infrastructure already to be added for various other optimizations, Fresnel Zones (combined with later described penetration) handling would provide a much more accurate simulation of real environments.

Background Noise As with physical objects causing issues with RF signals, some common non-network devices emit interfering electromagnetic radiation. Since RF acts as a shared-medium, such interference should be taken into consideration for transmission handling as well as protocol development. Especially in the case of DFS and automated network configuration protocols, they will want to discovery channels with nearby interfering components they are unable to adjust.

Examples:

- Microwaves
- Cordless Phones
- Fluorescent Lights
- Electrical Wiring
- Game Controllers

Background Noise Handling To aid with future development, NCVS already includes, and uses, a hook for retrieving background interference at a point. While current implementation simple returns a constant background value, this could easily be extended to handle more complicated field simulation based on the emitters above.

Furthermore, the emitters are often stationary thus the simulation processing costs would be minimal. Simply updating a RF field table, similar to what is used by the RF cloud isosurfaces, whenever the emitters make their rare changes would provide the needed information.

Electromagnetic Properties

As RF communication is merely a subset if the Electromagnetic Spectrum, common EM wave characteristics plays a direct role in link quality. Many of these characteristics directly relate to the frequency of the wave in question [11] [46].

Factors such as the penetration and reflection also depend on the various materials interact with. In addition to reflections changing the line-of-sight path of a transmission, diffraction can cause waves to appear to bend around the edges of objects. Further modification to the waves amplitude and frequency can be caused by interference with other waves and the movement of the source or destination radios.

While many of these characteristics include heavy math, and are of varying impact on most transmissions, the NCVS goal to support a wide range of devices requires their consideration. This is especially true when taking in consideration physical obstructions as discussed above.

The various factors will be discussed in an order similar to their importance to the simulation and well as ease of implementation. Worth note is that future NCVS handling of a number of wave characteristics will likely match those of graphical ray-tracing techniques.

Polarization Like the waves of a beach, the movement and force of energy has a direction. As such, perspective of sensing the wave can reduce the it's signal down to a theoretical non-existence. For example, if you could only sense a wave exuding energy along one axis and you turned 90 degrees, no force would be sensed. However, this extreme is unlikely in terrestrial environments due to the effects of penetration, reflection, and scattering. With this said, it wouldn't be overly complicated to add polarization details to the Antenna model and ChannelLink calculations.

A few properties would be required to account for both linear and circular polarizations as well as the calculations for handling the sensed signal reduction based on the varied orientations. Such calculations would also need to support the effects of the material for penetration and angle of reflection. **Penetration** The ability of the wave to penetrate an object depends on a number of factors including wave frequency, material to penetrate, and the thickness of the object. Furthermore, the thickness is the distance traveled through the material when traversing the link path from source to destination. Thus diagonal transmission through a regular wall will result in a distance greater than its width.

Using standard ray-tracing techniques, finding the objects which a link will pass through is fairly straightforward. As the total loss should be proportional to the product of the above factors the following equation develops:

$$loss = k * d * m * f \tag{5.1}$$

k	proportionality constant
d	distance through material
m	material constant
f	frequency

Reflections While the impact of a wave on an obstruction will result in some signal loss, a portion is reflected outward. This means that it is possible for signals to still reach an obstructed line-of-sight destination.

In the simplest form, the signal is reflect back along a matching angle of incidence on the opposite of the normal with the surface. Although not fully accurate, this assumption makes is computationally reasonable to find reflective links. Using the same maximum gain optimization, a maximum distance traveled can be established to limit the scope of obstacle checking. Next, an ellipsoid is formed with the nodes and the focus points. Once a subset of objects is found, their surfaces would be checked to see if any match the requirements of the law of reflections whose ray is emitted from the source and arrives at the destination. Finally, the resulting signal loss would be calculated to determine if the link would act as an active, interference, or none link. Similar to the equation used for penetration, the following would be used:

$$loss = k * angle * r * f \tag{5.2}$$

k	proportionality constant
angle	angle of incidence
r	material reflective constant
f	frequency

While this solution provides single-bounce links, high-power and low-frequency waves may be able to maintain enough signal-strength through multiple bounces. Should such functionality be required, there are ray-tracing algorithms to handle such cases at the cost of computational load.

Although discussed more in the constructive & destructive interference section, this also means that the same transmission could arrive at the destination at multiple time steps due to different length links caused by reflection, diffraction, and line-of-sight.

Diffraction With the wave characteristic of bending around the edge of obstacles, diffraction provides yet another opportunity for active and interference links.

Simplifying this case, the source could find all object edges within the maximum gain range. Upon calculating the signal at the edge, the edge would then act as an emitter and repeat the algorithm. Should a destination be found, its link state would be calculated using the re-emission signal strength. **Doppler Shift** Due to movement causing the distance between two Nodes to change, the frequency of the transmissions will change. In the case the Nodes approach each other, the signal will become greater. As the opposite, Nodes moving away from each other will result in a Doppler shift to a lower frequency.

As a result of such frequency changes, it is possible that a Channel's collision domain will change. However assuming that nodes are not moving at high speeds, the resulting change is fairly minor. Such effects will also likely disappear in due to common RF radio hardware abilities described in the next section.

Constructive & Destructive Interference Already lightly mentioned in the reflection section, there are cases in which multiple links could be established between a source and destination. As different path lengths will result in varying propagation delays, the summation of arriving waves may cause increase or decrease their amplitude. Furthermore, the offset can cause bit changes in the resulting data.

As this is a common issue with RF radios, they include filters to account for frequency shifts and reflection handling. Due to the high frequency of the waves arriving and the sub-microsecond timing required to correctly simulate such effects, it is unlikely that NCVS will be able to handle this task in the foreseeable future. Should the previous characteristics be implemented, an assumption that the hardware will adjust to the strongest signal from a given source is fairly reasonable and manageable.

Carriers

Carrier waves are the foundation to most RF communication. Through the modulation techniques used, various bandwidths, error correction, and noise issues emerge. As a number of modulations exist with different bandwidth abilities, negotiations between physical devices establish the properties on a particular link based on device compatibility and environment influence.

While many simulation engines ignore carriers in exchange for shorter simulation run-time, NCVS is focused on providing real-world emulation of all properties which impact multi-channel communications. As such, we make note of two features, physical negotiation and the resulting data rates.

Physical Negotiation With the additional information provided by the increase in RF simulation abilities, it should be possible to create models which represent the physical negotiation performed by their devices. While such negotiations are short, they do require use of the medium. As such they add to background noise and interference with other devices.

Data Rate As a result of these device negotiations, the bandwidth available on the links is changed. Furthermore, different modulations will be sensitive to various forms of errors. Taking these issues into link data rates (and error modeling) will result in a more complete representation of the production world.

Forward Error Correction

In order to limit the amount of transmission errors in noisy channels, FEC uses various coding techniques to include error detection and recovery data. Through these methods, a higher bandwidth usage is traded for the ability to avoid retransmissions. In addition to avoiding additional usage of the shared-medium, such techniques allow one-to-many transmissions to be handled without a need for a reverse path.

As the previous physical RF properties are added NCVS, the ability to correctly simulation these error handing methods will be important to consider medium overhead and receiver correctable errors.

5.3.2 Link Error Models

Although NCVS current handles cross-channel interference, it's done so using an all or nothing error model. To better test protocols, especially MAC layer, it would conducive to include variable levels and forms of error models.

Considerations

Before discussion the types of errors, we'll overview some of the considerations that an error model may wish to include.

Static Error Model In development, it's often useful to create errors of various types on purpose at specific times. A static error model would allow a developer to easily device drops, flips, and partial sends to test various parts of a protocol.

RF Factors As the previous mentioned RF factors discussed, reflections, SNR, frequency shifts, and other wave-based effects result in different errors. When tracking of these issues is added, this information could be used to create realistic errors.

Distance (including Wired) While several of the RF factors often don't affect wired connections, the distance traveled (especially if near noise sources like electrical lines) can result in interesting error cases.

Types of Errors

The following will overview some common errors which occur as well as situation that can cause them.

Known Dropped Frame Caused by a low SNR (high background interference and low signal strength), frequency shifting, or other NetDevice issues, a packet arrive at a destination but not be able to be recorded. However, the device detects that a frame was being sent. Currently this is how all errors in NCVS are handled.

Unknown Dropped Frame Ultra low SNR results in a packet appearing as just a higher level of background noise. Not only is the frame not received, but the NetDevice can't use it for carrier sensing, timing, or shift detection.

Bit Flipping Caused by an error on either side, a partial interference of another frame, or other error source, an invalid frame is successfully received but Frame Check Sequence (FCS) should fail in the MAC layer

Partial Frame While a partial frame can appear as a bit flip case, it is unique for NetDevices and protocols which use set timing or multi-stage messages which provide expected length information. While the total message should still fail a FCS test, the error may be noted at an earlier stage.

5.3.3 Wi-Fi Scanning

In order for wireless testing to correct simulate real-world situations, the interrupts in connections, delay in connections, and overhead in connection maintenance need to be considered. A significant part of this comes down to the scanning and synchronization requirements of each protocol. While Click provides beacon handling, NCVS needs to trigger client systems to scan their available channels from time-to-time and provide an API for non-Click protocols to interact. More important for the sake of simulation is also the ability to detect, record, and report such actions.

Discovery

The first task which must be conducted by a Wi-Fi (as well as other) Net-Devices is to scan the local environment. During this action, the NetDevice will change channels (with its delay), listen for beacons or other frame types for a period of time, and response as required. If this is triggered while a NetDevice is actively linked, that connection is temporarily lost until the scan cycle is complete and synchronization is reestablished.

Connection

When making a connection, a NetDevice has a hand-shaking process which takes care. In addition to creating frames which could cause interference or be interfered with, another delay is added. In the motivating use-cases for Intelligent Network Configuration, such delays can be important to simulate so that the protocol can detect them.

5.3.4 Power Consumption Modeling

One of features missing was exposed when comparing NCVS with other simulation utilities, power consumption modeling and tracking. Since NCVS is focused on maintaining connectivity and mobility, it needs to consider their arch-enemy, power restricts[51][27][40]. In a perfect world, batteries would last forever and be able to put out an unlimited about in instantaneous power. Obviously this is not the case.

Beyond the consumption of the NetDevices, various other factors play their part such CPU scaling, displays, and other loading sub-components. This section is going to lay the ground work for the inclusion of a power aware API for NCVS components.

Nodes

As was previous mentioned, most Nodes will have a finite power source from a battery and a number of loading sub-components. To support such components, Nodes can have their publisher-subscriber interfaces extended to account for batteries and power loads.

Battery Obviously not all Nodes are battery powered, however, a significant number of those that will be simulated with NCVS will be. Batteries have a set of qualities including:

- Capacity (Max, Current)
- Current (Max Draw, Max Charge, Current)
In order to correctly simulate their constraints, these values need to be collected from all attached sub-components.

Power Load Within a Node, there are several items beyond the NetDevices (discussed next) which draw their power. The following provides a few common categories which should be simulated. Note that several are not directly related to networking itself, however, they provide backing for realistic use-cases. Since little extra computation would be required to support them, there are included here.

CPU Load Protocols can be demanding on a CPU, especially in low-power, embedded systems. The power consumption of a processor is often related to its load (via speed scaling.)

Display In mobile phones, often the most significant power consumer is the display. Again, this is definitely not related to networks directly, however, a developer may wish to simulate user activities like browsing the Internet and checking their email.

Other Components Should there be other significant power loaders, they can extend the general base class.

Chargers Again simulating use cases, a charger is an anti-load item which provides power to the system. A little extra logic would have to be added provide realistic charging, however, protocols may act differently when power is freely available versus mobile.

NetDevices

Luckily in the case of NetDevices, their power consumption is often directly related to their activities. While there may be other states, providing the following 3 cases in the NetDevice base class will make general development easy.

Off A device can be considered off when no channel is active and no channel switch is in progress. While in this state, little or no power would be drawn.

Active Rx In order to receive data, the devices have to be powered on and will often be powering a signal amplifier. While some devices use the same power when transmitting as receiving, most save a significant amount. Furthermore, some schedule between Rx and off based on Access Point beacon synchronization.

A device is in this state when a channel is active but no content is being transmitted.

Active Tx As the high power consumer, a transmitting device will have to power an amplifier as well as other signal general hardware.

5.3.5 Random Generation Manager

Simulation expansions such as multithreading, recording, and the prior requirement for deterministic outcome require an easy-to-use, component specific order and time-based random number generation. The addition of being component specific will allow concurrent tasks to maintain their values no matter of the request order.

5.3.6 Special Simulation Modes

In order to maximum the use of NCVS, it makes sense to provide the functionality of simulation and graphics in varying forms. While current implementation only provides a real-time, interactive mode with partial recording, the architectures used throughout NCVS provide clear interface points which would allow a division of functionality.

To provide an overview of these functions, the rest of this section will be split into 3 parts, recording, playback, and a hybrid through event branching.

Recording

While current implementation has the ability to output pcap files at various stages, Node positions, NetDevice properties changes, and other dynamic factors are not saved. In addition to providing a method for playback to analyze other components of a network after the fact, such recording would allow a non-realtime, headless simulation to process the data at high-speed. These two methods for simulation processing, interactive and headless, will provide different abilities useful to varying development.

Interactive Through the use of the current mode available in NCVS, it lacks the ability to record the users actions. While the deterministic factors would allow for the regeneration of the same simulation to allow the user to look at another component later, it would be useful to minimum re-simulation, provide archiving of output, and allow non-interactive graphical output.

Beyond recording the simulation tasks, there may be benefit to recording camera and other graphical interaction events. These abilities would allow the playback of the interactive recording from both a new users perspective as well as the exact same perspective as the original recording.

To provide flexibility to both these cases, the recording should extend the publisher-subscriber model used elsewhere and allow user components to broadcast updates which could later be imported and replayed.

Headless A GUI-less method for simulation also provides the useful feature of quick run-time. Combined with the already discussed abilities for recording, headless simulation ability would promote future development in clustersupported simulation. While such logic could be useful in real-time, interactive modes, a number of latency challenges would likely emerge. Furthermore, smaller data sets may not have enough computational requirements, or concurrency, to offset the overhead related to a distributed system.

Simulation-less Playback

Without the weight of simulation, graphical representation should be quick and allow for forward and backward playback. However, without the backing of simulation, such playback would be non-interactive in the sense of network changes. The camera would still have full movement, visual components could be added and removed, and such output would provide great video-like presentation materials.

Branched Interactivity

A mix between recording and playback with the ability to move forward and backward in time, make changes, and have all paths be saved. While such tasks could be completed in independent simulations runs, a change may be desired late into an extended case. Furthermore, the duplicate-time data can be stored once thus saving total trace size.

To provide such ability, the trace files would need to include enough data to recreate the system at some granularity of time. This includes the state of all components, managers, and pseudo-random generation. Luckily these tasks are already partially achieved through the infrastructure supporting Multi-Buffer locking and random generation.

While this would likely be a sub-option of simulation recording, trace files which support the requirements for branched expansion could allow quick modification. Along the original goals of providing the ability to handle the "what if we do this" type of questions, being able to fast-forward a per-executed simulation to the point of interest, made a change, and instantly view the output provides an interesting opportunity.

5.4 Additional Modules

As the primary goal of NCVS is research in network interactions and development for new protocols, it's important to implement currently available technologies for comparison. Once in place, the interactive environment provided by NCVS will allow experimentation on current items with new eyes. Beyond current protocols, it's also important to expand the available simulated NetDevices and how they interact with each other. Additional research should also be performed on Antenna configurations with the visual feedback provided by NCVS.

5.4.1 Simulated Devices

While theoretical devices provide a general understanding of systems, it's the interactions between similar-but-different devices which can illustrate special cases. An additional variety of situations is achieved when non-common, but interfering, devices function in the same environment. When it comes to NCVS, the main hardware contributors to network variety are the NetDevices themselves and various Antenna models that connect to.

NetDevices

While the 802.11 series of devices are often backward compatible, variations in their timing, speeds, and error correction all provide interesting situations. In addition to creating various models of COTS 802.11 and Bluetooth devices, long range radios provide a number of relevant and useful information. A few common types of these devices include:

- Freewave Radios
- Mobile Phone Networks (TDMA, CDMA, GSM)
- WiMax

In addition to these devices, other sophisticated, high-bandwidth devices are actively being research. Examples using the 60 GHz spectrum provide interesting, short-range, high-bandwidth opportunities. Active is also being conducted in Near Field Communication devices.

By implementing a large base of these devices and prototyping various combinations, it's hopeful that developers will find new methods to achieve greater levels of connectivity and mobility.

Antennas

Along with devices, the Antennas they use dramatically affect the final results. By modeling various antennas, it should be possible to close the divide between prototype and production. This involves creating models of various types (such as dipole, dish, and arrays) and then testing them with the newly implemented NetDevices.

5.4.2 Protocols

As the primary motivation toward the creation of NCVS, protocol development is extremely important. While the use-cases were focused on network optimizations through intelligent configuration, the newly implemented NetDevices will need to have their MAC layers in place. Only after this will the research toward the various categories of network configuration is able to optimize the mediums.

MAC Layers

In the initial implementation of NCVS, a revision of CSMA/CA and CS-MA/CD has been partially implemented. After this implementation has been further developed to handle more of the various sub-features included, focus can be put on other the protocols for other devices. Some like protocols like 802.15.1 are specific to their device type (in this case Bluetooth.) In other situations, general classes can be created to explore scheduled and coded protocols (such as TDMA and CDMA.)

Network Configuration

Only after all the other tasks have been completely will development toward connectivity and mobility maximizing network configuration protocols achieve their full potential. While the following two cases relate to each, the development of either can be explored independently.

Dynamic Frequency Selection Although site surveys can be enlightening, phones, laptops, and interference source constant change the RF medium's characteristics. With multi-channel devices that function in a shared-medium with other compatible and interfering devices, it is desirable to dynamically adjust the frequency to achieve the least interference and the greatly link stability. As NCVS has a unique set of features to enable sandbox testing of situations and ability to reproduce them, new insight is undoubtedly on the horizon.

Intelligent Network Configuration Whether including DFS or not, INC provides a unique set of distributed system coordination. Perhaps the most obvious of challenges is how to share information while modifying the communication paths. Additional opportunities such as leader election, multi-interface coordination, and link statistic gather all benefit from a graphical interface.

Chapter 6

Conclusion

This paper has explored the simulation-based systems used throughout network development. Included in this coverage was an overview of how current development tools do not take advantage of human intuition due to their lack in clear visualization and immersive interactivity. Using traits of previous simulators and tools developed for other engineering fields, the Network Channel Visualizing Simulator, NCVS, was proposed and implemented.

Through the design a set of specific goals were established and later achieved through the implementation. Most important, the use of an interactive, 3D environment focused of channels (often RF-based) helps to facilitate the human ability for pattern recognition and intuition. Furthermore, the simulator made it a priority to support interference and timing related to shared-medium network links. RF support also included realistic antenna radiation patterns in order to provide use-case simulation of real-world, orientation aware, 3D movement. A great amount of effort was also put into creating an API which would be easy to use, and extend, without having to be a NCVS expert. Lastly, the use of the Click Modular Router [REF] provided the means of moving from prototype to production with only minor script modification.

While NCVS still requires additional work prior to being a mainstream simulator, this paper successfully showed a very similar output to that of NS-3. Though this validation was limited to a CSMA/CD protocol, it provided an example of how future development could be compared with NS-2 and NS-3 using common Click scripts. While left to future works, the inclusion of a Randomization Manager and refinement of the CSMA/CD MAC-layer should allow for deterministic output of the L2 layer and discrepancies to be overcome.

Along the implementation path, numerous hooks were added to enable the Future Work discussed to be efficiently added. Due to complexity of NCVS, especially in respect to cross-channel interference handling not provided by other tools, the current implementation does have a limitation on the number of nodes (and NetDevices) it can handle in real-time. To overcome this limitation, a number of optimizations have been proposed which would allow much larger simulations while still supporting the complexity of a simulation engine in real-time. Using the innovative abilities of NCVS, new protocol research and development can benefit from simple to implement, more complete, and easier to communicate simulation of network environments.

Bibliography

- [1] 3D CAD design software SolidWorks. http://www.solidworks.com/.
- [2] About GloMoSim. http://pcl.cs.ucla.edu/projects/glomosim/.
- [3] AirPlay. http://www.apple.com/itunes/airplay/.
- [4] AutoCAD civil 3D civil engineering design software autodesk. http://usa.autodesk.com/civil-3d/.
- [5] DLNA. http://www.dlna.org/.
- [6] JSim home page. http://physiome.org/jsim/.
- [7] NS-2. http://nsnam.isi.edu/nsnam/index.php/Main_Page.
- [8] ns-3. http://www.nsnam.org/.
- [9] OMNeT++ network simulation framework. http://www.omnetpp.org/.
- [10] OPNET technologies, inc. http://www.opnet.com/.
- [11] PDF a practical introduction to radio physics.
- [12] PlanetLab | an open platform for developing, deploying, and accessing planetary-scale services. http://www.planet-lab.org/.

- [13] QualNet SCALABLE network technologies. http://www.scalablenetworks.com/content/products/qualnet.
- [14] SpaceClaim|3D direct modeling software for engineering, manufacturing and CAE. http://www.spaceclaim.com/en/.
- [15] UWSim | the UnderWater simulator. http://www.irs.uji.es/uwsim/.
- [16] Wi-Fi direct. http://www.wi-fi.org/discover-and-learn/wi-fi-direct.
- [17] Bagrodia, Meyer, Takai, Chen, Zeng, Martin, and Song. parsec.pdf.
- [18] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford. In VINI veritas. <u>ACM SIGCOMM Computer Communication Review</u>, 36(4):3, Aug. 2006.
- [19] J. Broch, D. A. Maltz, D. B. Johnson, Y. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. page 8597. ACM Press, 1998.
- [20] R. Calvo and J. Campo. Adding multiple interface support in NS-2.
- [21] S. Corson and J. Macker. <u>Mobile Ad hoc Networking (MANET): Routing</u> <u>Protocol Performance Issues and Evaluation Considerations.</u>
- [22] J. Diebel. <u>Representing Attitude: Euler Angles, Unit Quaternions, and</u> Rotation Vectors. 2006.
- [23] B. Eslamnour and S. Jagannathan. <u>A Hybrid Multi-Channel Multi-Interface</u> Routing Protocol For Wireless Ad-Hoc Networks.
- [24] HP. Wi-Fi and Bluetooth Interference Issues.

- [25] X. Huang, S. Feng, and H. Zhuang. Jointly optimal congestion control, channel allocation and power control in multi-channel wireless multihop networks. Computer Communications, 34(15):18481857, Sept. 2011.
- [26] S. Ju and J. Evans. <u>Cognitive Multipath Multi-Channel Routing Protocol</u> for Mobile Ad-Hoc Networks.
- [27] Kasirajan, Xu, Zawodniok, and Jagannathan. <u>Demonstration of a</u> <u>Multi-Interface Multi-Channel Routing Protocol (MMCR) for WSNs using</u> Missouri S&T Motes.
- [28] A. Khetrapal. <u>Routing techniques for Mobile Ad Hoc Networks Classification</u> and Qualitative/Quantitative Analysis.
- [29] E. Kohler, M. Handley, and S. Floyd. Designing DCCP. page 27. ACM Press, 2006.
- [30] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. <u>ACM Transactions on Computer Systems</u>, 18(3):263–297, Aug. 2000.
- [31] M. Koksal. A survey of network simulators supporting wireless networks.
- [32] P. Kyasanur and N. Vaidya. <u>Routing in Multi-Channel Multi-Interface Ad</u> Hoc Wireless Networks.
- [33] P. Kyasanur and N. Vaidya. Routing and interface assignment in multichannel multi-interface wireless networks. In <u>Wireless Communications and</u> <u>Networking Conference, 2005 IEEE</u>, volume 4, pages 2051 – 2056 Vol. 4, Mar. 2005.

- [34] P. Kyasanur and N. H. Vaidya. Routing and link-layer protocols for multichannel multi-interface ad hoc wireless networks. <u>ACM SIGMOBILE Mobile</u> Computing and Communications Review, 10(1):3143, Jan. 2006.
- [35] L. Lamont, M. Wang, L. Villasenor, T. Randhawa, and S. Hardy. Integrating WLANs & MANETs to the IPv6 based internet. volume 2, page 10901095. IEEE.
- [36] B. Letor, De Cleyn. The madwifi wireless extension for nsclick.
- [37] N. Letor, P. De Cleyn, and C. Blondia. Enabling cross layer design: adding the MadWifi extensions to nsclick. In <u>Proceedings of the 2nd international</u> <u>conference on Performance evaluation methodologies and tools</u>, ValueTools '07, page 19:119:10, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).
- [38] P. Levis and N. Lee. Tossim: A simulator for tinyos networks.
- [39] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. pages 163–169. ACM Press, 1987.
- [40] Lou, Liu, and Zhang. <u>Performance Optimization using Multipath Routing</u> in Mobile Ad Hoc and Wireless Sensor Networks.
- [41] B. W. C. Ltd. PDF radio signal propagation.
- [42] S. Mueller, R. P. Tsang, and D. Ghosal. Multipath routing in mobile ad hoc networks: Issues and challenges. In M. C. Calzarossa and E. Gelenbe, editors, <u>Performance Tools and Applications to Networked Systems</u>, volume 2965, page 209234. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

- [43] M. Neufeld, A. Jain, and D. Grunwald. Nsclick: bridging network simulation and deployment. page 74. ACM Press, 2002.
- [44] J. Pan. A survey of network simulation tools: Current status and future development. http://www.cse.wustl.edu/~jain/cse567-08/ftp/simtools/index.html.
- [45] G. Parissidis, V. Lenders, M. May, and B. Plattner. Multi-path routing protocols in wireless mobile ad hoc networks: A quantitative comparison. In Y. Koucheryavy, J. Harju, and V. Iversen, editors, <u>Next</u> <u>Generation Teletraffic and Wired/Wireless Advanced Networking</u>, volume 4003 of <u>Lecture Notes in Computer Science</u>, pages 313–326. Springer Berlin / Heidelberg, 2006. 10.1007/11759355_30.
- [46] I. Rosu. PDF basics of radio wave propagation.
- [47] E. Royer and C. Toh. A review of current routing protocols for ad hoc mobile wireless networks. IEEE Personal Communications, 6(2):4655, Apr. 1999.
- [48] Sundani, Li, Devabhaktuni, Alam, and Bhattacharya. Wireless sensor network simulators: A survey and comparisons.
- [49] P. L. Suresh and R. Merz. ns-3-click: click modular router integration for ns-3. In <u>Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques</u>, SIMUTools '11, page 423430, ICST, Brussels, Belgium, Belgium, 2011. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).
- [50] T. Tsai, S. Tsai, and T. Liu. Cross-Layer design for multi-power, multiinterface routing in wireless mesh networks. In Advances in Mesh Networks,

2009. MESH 2009. Second International Conference on, pages 109–114, June 2009.

[51] B. Yan and H. Gharavi. Multi-Path Multi-Channel routing protocol. page 2731. IEEE.