



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

**REPEATABLE UNIT TESTING OF DISTRIBUTED
INTERACTIVE SIMULATION (DIS) PROTOCOL
BEHAVIOR STREAMS USING WEB STANDARDS**

by

Tobias Brennenstuhl

June 2020

Thesis Advisor:
Second Reader:

Don Brutzman
Terry D. Norbraten

Research for this thesis was performed at the MOVES Institute.

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE June 2020		3. REPORT TYPE AND DATES COVERED Master's thesis
4. TITLE AND SUBTITLE REPEATABLE UNIT TESTING OF DISTRIBUTED INTERACTIVE SIMULATION (DIS) PROTOCOL BEHAVIOR STREAMS USING WEB STANDARDS				5. FUNDING NUMBERS
6. AUTHOR(S) Tobias Brennenstuhl				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000				8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A				10. SPONSORING / MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.				12b. DISTRIBUTION CODE A
13. ABSTRACT (maximum 200 words) <p>The IEEE Distributed Interactive Simulation (DIS) protocol is used for high-fidelity real-time information sharing among simulations and trainers across the entire international Modeling and Simulation (M&S) community. If archivally saved and replayed, DIS streams have the potential to become a valuable source of Big Data. The availability of archived prerecorded behavior streams for replay, adaptation, and analysis can benefit an immense variety of application areas. The computer science principle "a stream is a stream" indicates that data in motion is equivalent to data at rest. This characteristic can enable powerful capabilities for DIS.</p> <p>This thesis presents prototypes to demonstrate how various forms of repeatability are key to gaining improved benefits from DIS stream analysis. Unit testing of DIS behavior streams allows confirmation of both repeatability and correctness when testing all manner of applications, exercises, simulations, and training sessions. A related use case is automated after-action review (AAR) from recorded DIS streams. This thesis also shows how a DIS stream is converted into autogenerated code that can animate an X3D Graphics model. Many obstacles were overcome during this work, and so various best practices are provided. Of note is that unit testing might even become a contract requirement for incrementally developing and stably maintaining Live Virtual Constructive (LVC) code bases. This progress provides many opportunities for future work.</p>				
14. SUBJECT TERMS distributed interactive simulation, modeling and simulation, behavior streams, unit testing, X3D				15. NUMBER OF PAGES 171
				16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified		18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified		19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified 20. LIMITATION OF ABSTRACT UU

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

**REPEATABLE UNIT TESTING OF DISTRIBUTED INTERACTIVE
SIMULATION (DIS) PROTOCOL BEHAVIOR STREAMS USING WEB
STANDARDS**

Tobias Brennenstuhl
Lieutenant Colonel, German Army
EE, University of the German Armed Forces, 2006

Submitted in partial fulfillment of the
requirements for the degree of

**MASTER OF SCIENCE IN MODELING, VIRTUAL ENVIRONMENTS, AND
SIMULATION**

from the

**NAVAL POSTGRADUATE SCHOOL
June 2020**

Approved by: Don Brutzman
Advisor

Terry D. Norbraten
Second Reader

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The IEEE Distributed Interactive Simulation (DIS) protocol is used for high-fidelity real-time information sharing among simulations and trainers across the entire international Modeling and Simulation (M&S) community. If archivally saved and replayed, DIS streams have the potential to become a valuable source of Big Data. The availability of archived prerecorded behavior streams for replay, adaptation, and analysis can benefit an immense variety of application areas. The computer science principle “a stream is a stream” indicates that data in motion is equivalent to data at rest. This characteristic can enable powerful capabilities for DIS.

This thesis presents prototypes to demonstrate how various forms of repeatability are key to gaining improved benefits from DIS stream analysis. Unit testing of DIS behavior streams allows confirmation of both repeatability and correctness when testing all manner of applications, exercises, simulations, and training sessions. A related use case is automated after-action review (AAR) from recorded DIS streams. This thesis also shows how a DIS stream is converted into autogenerated code that can animate an X3D Graphics model. Many obstacles were overcome during this work, and so various best practices are provided. Of note is that unit testing might even become a contract requirement for incrementally developing and stably maintaining Live Virtual Constructive (LVC) code bases. This progress provides many opportunities for future work.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	OVERVIEW.....	1
B.	CURRENT CAPABILITIES AND LIMITATIONS.....	2
C.	PROBLEM STATEMENT	3
D.	SPECIAL DEFINITIONS.....	4
1.	“Repeatable”	4
2.	“Compression”	5
E.	MOTIVATING USE CASES.....	5
1.	Software Assessment.....	5
2.	Software Testing.....	5
3.	Simulation Analysis	6
F.	BIG DATA	6
G.	ORGANIZATION OF THIS THESIS.....	8
II.	RELATED WORK.....	9
A.	WEB STANDARDS.....	9
B.	3D MODELING	9
1.	Software for X3D	10
2.	Comparison of X3D Players.....	12
3.	X3D TimeSensor Node	27
4.	X3D PositionInterpolator Node	28
5.	X3D OrientationInterpolator Node.....	28
C.	NETWORKING AND DISTRIBUTED VIRTUAL ENVIRONMENTS	29
D.	COMMAND AND CONTROL SYSTEM TO SIMULATION SYSTEM INTEROPERATION (C2SIM)	30
E.	PDU ENCODINGS	31
F.	PROPRIETARY ENCODINGS AND STANDARDIZATION CONSIDERATIONS	32
III.	SHARED BEHAVIORS IN 3D VIRTUAL ENVIRONMENT (VE) SIMULATIONS	33
A.	INTRODUCTION.....	33
B.	DISTRIBUTED INTERACTIVE SIMULATION (DIS)	33
C.	AVAILABLE SOFTWARE.....	34
1.	OpenDIS7.....	34
2.	Redsim DIS PDU Recorder.....	35

3.	MAK Data Logger	36
4.	Pitch Cross Domain Security (CDS) Gateway	39
D.	BASIC ARCHITECTURAL COMPONENTS OF DIS.....	39
1.	Entity State PDU (ESPDU) Timestamp Considerations	40
2.	ESPDU Location	40
3.	Location with Respect to the World.....	40
4.	Velocity.....	40
5.	Orientation.....	41
6.	Dead Reckoning (DR)	41
7.	World Coordinate System.....	42
8.	Entity Coordinate System	43
E.	TYPES OF PDUS	44
F.	SPIDERS3D VIRTUAL ENVIRONMENT (VE)	46
G.	SUMMARY	47
IV.	IMPLEMENTATION AND DEMONSTRATION	49
A.	INTRODUCTION.....	49
B.	CAPTURING DIS PACKETS.....	49
C.	RECORDING DIS PACKETS	55
1.	Saving Recorded PDUs to Base64 Encoding	56
2.	Saving Recorded PDUs to a Parsable Plain-Text File	56
D.	PLAYING BACK DIS PACKETS.....	57
E.	RECORDING DATA FROM SIMULATIONS.....	58
F.	PLAYING BACK RECORDED DIS DATA TO SIMULATIONS	62
G.	TRANSFORM A PDU STREAM INTO X3D POSITIONINTERPOLATOR.....	63
H.	CREATE A MATCHING ORIENTATIONINTERPOLATOR.....	71
I.	X3D IMPLEMENTATIONS OF PDUS	73
J.	AUTOMATED ANALYSIS OF PDU STREAMS FOR AFTER- ACTION REVIEW (AAR).....	74
K.	SUMMARY	75
V.	TESTING AND TROUBLESHOOTING.....	77
A.	INTRODUCTION.....	77
B.	CONNECTING TO DIFFERENT TYPES OF NETWORKS	77
C.	TEST AND EVALUATION OF CONNECTIONS	77
D.	UNIT TESTING PRINCIPLES AND BEST PRACTICES	81
E.	QUALITY ASSURANCE (QA) OF PLAYED BACK STREAMS.....	82
F.	UNIT TESTING OF SOFTWARE PACKAGE INTEGRITY	83

G.	TEST FOR NUMBERS OF PACKETS RECEIVED	85
H.	COMPARATOR FOR DIS STREAMS	86
I.	VERIFICATION, VALIDATION AND ACCREDITATION (VV&A) FOR LVC APPLICATION AND SCENARIOS	89
J.	TROUBLESHOOTING	90
1.	Wireshark	90
2.	Test Setup with OpenDis7Examples	91
3.	Firewall Settings.....	91
4.	AllPduRoundTripTest.java	91
5.	CPU Speed	92
6.	Test Dead Reckoning (DR).....	93
K.	FUNDAMENTAL IMPORTANCE OF REPEATABLE MEASUREMENTS	96
L.	SUMMARY	97
VI.	CONCLUSIONS AND RECOMMENDATIONS.....	99
A.	CONCLUSIONS	99
B.	RECOMMENDATIONS FOR FUTURE WORK.....	100
APPENDIX A. LIST OF ALL PDUS.....		101
APPENDIX B. FIREWALL CONFIGURATION.....		111
APPENDIX C. SOURCE CODE.....		119
A.	INTRODUCTION.....	119
B.	PLAYERPLAINTEXT.JAVA	119
C.	RECORDERPLAINTEXT.JAVA.....	126
D.	SLIDINGWINDOW.JAVA.....	130
E.	COORDINATES.JAVA	133
F.	CREATEX3DINTERPOLATORS.JAVA.....	135
G.	EXAMPLE OF AUTOGENERATED X3D NODES	141
LIST OF REFERENCES		143
INITIAL DISTRIBUTION LIST		149

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

Figure 1.	Entity in VR Forces application following a straight track	7
Figure 2.	Entity in VR Forces application following a many-sided polygon.....	7
Figure 3.	QR code for Hello Germany Model displayed by X_ITE	13
Figure 4.	FreeWRL default view of Hello Germany scene.....	13
Figure 5.	H3DViewer default view of Hello Germany scene	14
Figure 6.	Instant Reality default view of Hello Germany scene	14
Figure 7.	Octaga Player default view of Hello Germany scene	15
Figure 8.	View3DScene default view of Hello Germany scene.....	15
Figure 9.	Xj3D default view of Hello Germany scene.....	16
Figure 10.	Plane crossection view of the X3D model of a ring extrusion	17
Figure 11.	View3DScene Diagnostic Testing for Extrusions	18
Figure 12.	Intersecting inner and outer ring of the model leads to incorrectly formed geometry and erreneous results	18
Figure 13.	Fixed model with open ring as a single convex strip rotated around a circle.....	19
Figure 14.	QR code for Olympic Rings URL displayed by X_ITE	19
Figure 15.	FreeWRL default view of the Olympic Rings scene is correct	20
Figure 16.	H3D default view of the Olympic Rings scene is erroneous	20
Figure 17.	Instant Reality default view of the Olympic Rings scene is correct	21
Figure 18.	Octagon Player default view of the Olympic Rings scene is correct.....	21
Figure 19.	View3DScene default view of the Olympic Rings scene is correct	22
Figure 20.	Xj3D default view of the Olympic Rings scene	22
Figure 21.	FreeWRL default view of the Lighthouse scene is correct.....	24
Figure 22.	H3D default view of the Lighthouse scene without beam cone	24

Figure 23.	Instant Reality default view of the Lighthouse scene without beam cone	25
Figure 24.	Octaga Player default view of the Lighthouse scene is correct	25
Figure 25.	View3DScene default view of the Lighthouse scene without beam cone	26
Figure 26.	Xj3D default view of the Lighthouse scene.....	26
Figure 27.	Code excerpt for a X3D TimeSensor shows XML encoding of keyfields in this node	28
<i>Figure 28.</i>	Code excerpt for a X3D PositionInterpolator shows XML encoding of keyfields in this node.....	28
<i>Figure 29.</i>	Code excerpt for a X3D OrientationInterpolator shows XML encoding of keyfields in this node	29
Figure 30.	Option panel of Redsim’s DIS PDU Recorder	35
Figure 31.	MAK Data Logger main window displaying bars and annotations.....	36
Figure 32.	PDU Kind filter selection window in MAK Data Logger	37
Figure 33.	Plain Text output of a DIS PDU recorded by MAK Data Logger.....	38
Figure 34.	The “world coordinate system” shows how a geocentric coordinate system is arranged. Source: IEEE (2012).	43
Figure 35.	Entity coordinate system. Source: IEEE (2012).	44
Figure 36.	Dialog to select network interface	50
Figure 37.	View of Wireshark in capturing mode.....	51
Figure 38.	View of text field for filter expression.....	51
Figure 39.	View of an applied filter for a single IP address.....	52
Figure 40.	Details of a single DIS PDU packet parsed by Wireshark.....	53
Figure 41.	All 72 IEEE DIS PDU types captured with Wireshark	54
Figure 42.	Recording and play back of PDU streams can occur using different encodings. Future encoding pairs are straightforward and testable.....	55

Figure 43.	PDU types 5 – 12 from AllPduSender.java in BASE64 encoding text characters offering some compression but not plain-text readability	56
Figure 44.	PDU types 5 – 12 from AllPduSender.java in plain text makes the values of each received PDU human readable.....	57
Figure 45.	Wireshark snapshot of PDU types 5 – 12 from Base64 encoded and unencoded log file.....	58
Figure 46.	VR Forces Application Simulation Connection Configuration	59
Figure 47.	PduListenerSaver.java is ready to record PDUs	60
Figure 48.	VR Forces window with status of DIS entities	60
Figure 49.	VR Forces screen with one track and one hostile entity	62
Figure 50.	Algorithm for creating X3D tracks through PDU filtering.....	64
Figure 51.	Code excerpt for a X3D TimeSensor shows XML encoding of keyfields in this node	65
<i>Figure 52.</i>	Code excerpt for a X3D PositionInterpolator shows XML encoding of keyfields in this node.....	65
Figure 53.	UML-diagram for Coordinates.java.....	68
Figure 54.	Flowchart of sliding window algorithm to compute area A	69
Figure 55.	Heron’s Formula for measuring collinearities. Adapted from AmBrSoft (2014).	70
Figure 56.	DIS track of an aircraft with 12 PDUs.....	71
Figure 57.	Compressed DIS track of the aircraft from with 4 PDUs	71
Figure 58.	A aircraft flying a circle.....	72
Figure 59.	Replay of an aircraft using a solitary PositionInterpolator	72
Figure 60.	Replay of an aircraft using a PositionInterpolator and OrientationInterpolator	73
Figure 61.	Windows Command Prompt command for collecting test statistics	78
Figure 62.	Sample log file for 10 packets over local WiFi	79
Figure 63.	Windows Command Prompt for infinite packet sending.....	80

Figure 64.	Sample log file for an infinite connection test	81
Figure 65.	MAK Logger Unit Test Steps: Capture stream, play, store, replay, then perform difference (diff) comparison	82
Figure 66.	UML Diagram for AllPduRoundTripTest.java Comparison. Source: Schutt (2019).....	84
Figure 67.	Status message after a successful run of AllPduRoundTripTest.java.....	84
Figure 68.	Status message after an unsuccessful run of AllPduRoundTripTest.java	84
Figure 69.	Two recorders connected to one simulation to check for differences in the recorder output	86
Figure 70.	Flow chart for proposed DisStreamComparator.java unit testing utility class	88
Figure 71.	Flowchart for using VV&A procedures to prove that code for behavior stream handling is free of errors	90
Figure 72.	One entity with a track of six waypoints containing multiple turns set up in VR Forces in order to illustrate dead reckoning effects	93
Figure 73.	Images of DIS run, DIS replay, and DIS replay with DR side by side.....	94
Figure 74.	DIS original run of the scenario in Figure 72	95
Figure 75.	DIS replay of the scenario Figure 72	95
Figure 76.	DIS replay of the scenario in Figure 72 with network connection interrupted twice	95
Figure 77.	Network Icon in Windows Taskbar	111
Figure 78.	Network Settings Dialog.....	111
Figure 79.	Network Status window	112
Figure 80.	Firewall & Network protection window	112
Figure 81.	Windows 10 Defender Firewall window	113
Figure 82.	Windows Defender Firewall Inbound Rules view.....	114
Figure 83.	Windows Defender Firewall New Inbound Rule Wizard Type Selection.....	114

Figure 84.	Windows Defender Firewall New Inbound Rule Wizard Port Selection.....	115
Figure 85.	Windows Defender Firewall New Inbound Rule Wizard Allow/Block Dialog	115
Figure 86.	Windows Defender Firewall New Inbound Rule Wizard Network Selection.....	116
Figure 87.	Windows Defender Firewall New Inbound Rule Wizard Name Dialog.....	117
Figure 88.	Windows Defender Firewall Inbound Rules view with new DIS Port 3000 rule	117

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1.	Comparison of player performance for Hello Germany	16
Table 2.	Comparison of player performance for Olympic Rings	23
Table 3.	Comparison of player performance for Lighthouse with beam cone	27
Table 4.	Dead Reckoning formulas. Source: (IEEE, 2012)	42
Table 5.	Average roundtrip times for tested network connections	80
Table 6.	Number of PDUs recorded within three minutes for different CPU types	92

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AAR	After Action Review
AR	Augmented Reality
C2	Command and Control
C2SIM	Command and Control System to Simulation System Interoperation
CAS	Close Air Support
C-BML	Coalition Battle Management Language
C-DIS	Compressed DIS
CDS	Cross Domain Security
COTS	Commercial Off The Shelf
CSV	Comma Separated Values
DARPA	Defense Advanced Research Projects Agency
DIS	Distributed Interactive Simulation
ESPDU	Entity State Protocol Data Unit
GOTS	Government Off The Shelf
HLA	High-Level Architecture
HTML	Hyper Text Markup Language
i18n	Internationalization
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
l10n	Localization
LAN	Local Area Network
LVC	Live, Virtual and Constructive
M&S	Modeling and Simulation
MAGTF	Marine Air-Ground Task Force
MOTS	Military Off The Shelf
MSDL	Military Scenario Definition Language
MSG	Modeling and Simulation Group
MTWS	MAGTF Tactical Warfare Simulation
NPS	Naval Postgraduate School
PDU	Protocol Data Units
QA	Quality Assurance
RNG	Random Number Generator

RPR	Real-time Platform Reference
SAVAGE	Scenario Authoring and Visualization for Advanced Graphical Environments
TENA	Test and Training Enabling Architecture
VE	Virtual Environment
VLAN	Virtual Local Area Network
VPN	Virtual Private Network
VR	Virtual Reality
VV&A	Verification, Validation and Accreditation
WAN	Wide Area Network
X3D	Extensible 3D Graphics International Standard
X3DJSAIL	X3D Java Scene Access Interface Library
X3DPSAIL	X3D Python Scene Access Interface Library
XML	Extensible Markup Language

I. INTRODUCTION

DIS-emitting simulations and trainers can be found all over the DOD. Using them to drive co-located simulations and trainers is one possible use case. But recording DIS streams opens many other use cases that can be applied to DIS streams. Converting DIS streams that drive entities within simulations that can communicate via DIS into autogenerated code that can drive other software products is a focus of this research. A DIS stream is converted into autogenerated code to drive an X3D model using a PositionInterpolator, an OrientationInterpolator, and a TimeSensor Node.

To achieve autogenerating it is necessary to change code that is already available within the OpenDIS7 distribution. Because OpenDIS7 is publicly available it must not be corrupted during the process of code development. One way to achieve this is to use the concept of unit testing, which will be explained and applied to code written for this thesis.

Helpful troubleshooting for ensuring interoperability between simulation and using Java classes from the OpenDIS7 distribution is provided to protect the user of this thesis's code from frustration.

Within this thesis, “simulation” and “simulation application” are used synonymously.

A. OVERVIEW

The IEEE Distributed Interactive Simulation (DIS) protocol is used for high-fidelity real-time information sharing among simulations and trainers across the entire international Modeling and Simulation (M&S) community. If archivally saved and replayed, DIS streams have the potential to become a valuable source of Big Data. The availability of archived prerecorded behavior streams for replay, adaptation and analysis can benefit an immense variety of application areas. The computer science principle “a stream is a stream” indicates that data in motion is equivalent to data at rest. This characteristic can enable powerful capabilities for DIS.

This thesis presents prototypes to demonstrate how various forms of repeatability are key to gaining improved benefits from DIS stream analysis. Unit testing of DIS behavior streams allows confirmation of both repeatability and correctness when testing all manner of applications, exercises, simulations, and training sessions. A related use case is automated after-action review (AAR) from recorded DIS streams. This thesis also shows how a DIS stream is converted into autogenerated code that can animate an X3D Graphics model. Many obstacles were overcome during this work, and so various best practices are provided. Of note is that unit testing might even become a contract requirement for incrementally developing and stably maintaining Live Virtual Constructive (LVC) code bases. This progress provides many opportunities for future work.

B. CURRENT CAPABILITIES AND LIMITATIONS

Typically every significant military off-the-shelf (MOTS) simulation communicates via one of the standards for simulation interoperability: Distributed Interactive Simulation (DIS), Test and Training Enabling Architecture (TENA) or High-Level Architecture (HLA). A common characteristic for each is that small data packets are sent over the network to share the current position and status of simulation entities.

Commercial off-the-shelf (COTS) software is available for purchase or rental under licensing terms. Government off-the-shelf (GOTS) and MOTS software can be open source and restricted government produced.

Most of the MOTS simulation can speak more than one standard. Nearly all of them lack the possibility to record and playback more than their own PDUs. When running more than one simulation together on one network and recording PDU streams of the whole simulation a third-party product is necessary (but often not available). In the next chapter, Related Work, two potential COTS products are introduced. Further information or licensing considerations is available at the NPS Scenario Authoring and Visualization for Advanced Graphical Environments (SAVAGE) Research Group Developers Guide website, in sections “Licensing” and “Free as in Freedom” (NPS Savage Research Group, n.d.-a).

None of the aforementioned products can directly convert a PDU stream to make it playable over the internet using Web standards like X3D. This thesis demonstrates that such capabilities are easily achieved.

C. PROBLEM STATEMENT

Modeling and Simulation (M&S) is not relevant to active warfighting unless interoperability can be achieved between Live, Virtual and Constructive (LVC) simulations, robot telemetry, and command and control (C2). Therefore, M&S becomes more relevant when more participants and observers are reached and attracted by the outcome of simulations or the use of models. These domains have remained “stovepiped” and disconnected for decades. Common grand opportunities for cross-connected interoperability at syntactic semantic and functional levels are described in detail by Dr. Curtis Blais in his NPS dissertation (Blais, 2018).

Utilizing open-source Web technology provides valuable resources that can help numerous systems regardless of classification of the information streams that they handle. Such techniques are also useful when it comes to publishing outcomes to a broad range of participants and observers. A replay of a simulation run can be distributed to many more computers that are unable to run the simulation software. All that is needed is a converter that ingests a PDU stream and converts it into a format that is displayable using open-source Web technology using a Web browser on any personal computer or mobile device. Once a stream is recorded and stored to a file, it can be analyzed and adapted for simulations of any kind.

Handling 3D models and streams needs to become as easy as opening a Web browser of choice and searching for keywords. Installing specialized software to display 3D models is not attractive to the user and raises the bar for enterprise systems to a point, where the users have no access new techniques because adaption is too complex. This problem state has repeated for many years (indeed decades) due to a general failure to adopt repeatable open standards. The combination of repeatable behavior streaming in combination with searchable authenticating data opens the doors for both LVC-C2 relevance and Big Data techniques for modern data science.

D. SPECIAL DEFINITIONS

For this thesis, two words must be defined: repeatable and compression.

1. “Repeatable”

In regards to simulation, the word “repeatable” can be defined in three different ways. They are important distinctions that are explored in detail in this thesis.

a. Replayable

The simplest definition of “repeatable” is that a simulation that was run once is recorded and can be identically replayed again and again. In this context “repeatable” is hereby applied to the characteristic that a simulation can be repeated identically and testably for as many times as desired.

b. Simulation Seed Consistency

The second meaning of “repeatable” is the repetition of the simulation itself using a starting seed for pseudo-random number generators (e.g., for a given seed, the fight between two entities always ends the same way), although probabilities and random numbers are used. Varying the seed allows diverse scenario validations to be repeated with varying results. Keeping the random-number-generator (RNG) seed construct provides identical inputs for identical output results.

c. Simulation Seed Variability

The third definition of “repeatable” is the run of a simulation with different seeds to get stochastically consistent outcomes. New entities are injected into the simulation using another simulation, so the entities show up at the same place and time in every simulation run. The outcome for each simulation run differs because the random number generators are started with different seeds. In this example “repeatable” applies to the repeated injection of entities for every simulation run. Such controlled variation is valuable not only for analytic assessment but also machine-learning training and software/system/personnel evaluation.

2. “Compression”

For this thesis, “compression” describes the fact that a certain amount of data that is recorded, stored or transmitted, is modified or distilled to reduce the amount of data that has to be handled, stored or transmitted. Such compressions can be lossy but are preferably lossless for best repeatability and interoperability. Compaction of data can also lead to reduced computation loads and improved communications throughput.

E. MOTIVATING USE CASES

PDU streams that are captured in the MOVES LVC Lab can be replayed to the original application that sent them out or in any other application that can communicate via DIS. There is one major issue with playing it back verbatim to an application. The older the PDU stream gets, the trickier it becomes to handle the difference in time. A PDU packet must have a valid timestamp to be accepted by the DIS application. Another issue is the life cycle of DIS applications. After storing a DIS stream for some years the application may be discontinued. An additional issue is evolution of the standard. A DIS stream valid under a certain version of the standard may not be understandable on a system running a different version of the standard—new versions are not guaranteed to be backward-compatible.

1. Software Assessment

One use case for this thesis is to make the DIS stream playable using open-source standards and software. X3D, for example, can move 3D models in an arbitrarily complex 3D world and deliver it in a Web browser. The journey from recording a DIS stream, compressing it and turning it into a PositionInterpolator/OrientationInterpolator pair with TimeSensor clock is one part of this thesis. A block diagram of this journey is sketched out in Figure 50.

2. Software Testing

On the way from a DIS stream to the aforementioned X3D code, many lines of code were produced. When integrating code into an existing software distribution it is important to maintain the core functionality of the existing code, such as OpenDIS7 Code Base or

X3D Examples Archive. Therefore, it is essential to repeatedly test functionality. Repeatedly testing blocks of code (known as “unit testing”) is achieved by having code that can be run automatically during a build process that typically stops the process immediately once a test fails. Unit testing is the only way to ensure that new code does not damage old code. For DIS, unit testing is described in detail in this thesis.

3. Simulation Analysis

Another use case is the analysis of a recorded stream to test it for a specific constellation of entities. E.g., a pilot flying in a simulator that is connected to a LVC network utilizing PDUs to send out the position of the aircraft of choice, can be evaluated in an AAR by a simple analysis of the stream. The stream contains all information about the flight, target, enemies killed, and attempts of enemies to attack the pilot. No COTS software is needed to evaluate the pilot’s performance. This use case supports training reinforcement, qualification testing, after-action analysis, and synthetic exercise experimentation.

F. BIG DATA

While the outcome of a simulation run is important, the storage needed is relatively small. But when it comes to recording simulation streams, the amount of data being stored is increasing.

An example illustrates this situation. Within a run of a simulation many PDUs are sent over the network. An entity that follows a straight track simulated in VR Forces produces about 174 Entity State PDUs (ESPDUs) per minute (Figure 1), whereas an entity following a circle-shaped route produces 290 ESPDUs per minute (Figure 2).

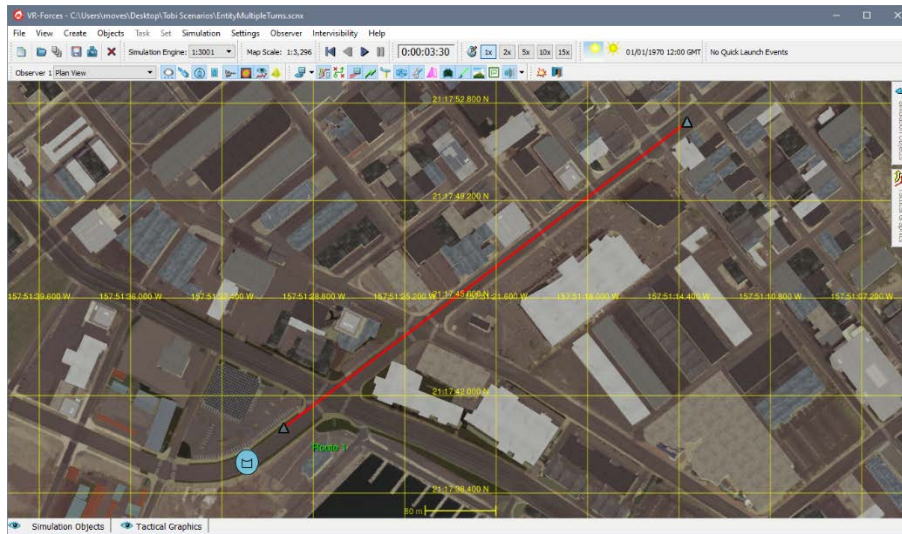


Figure 1. Entity in VR Forces application following a straight track

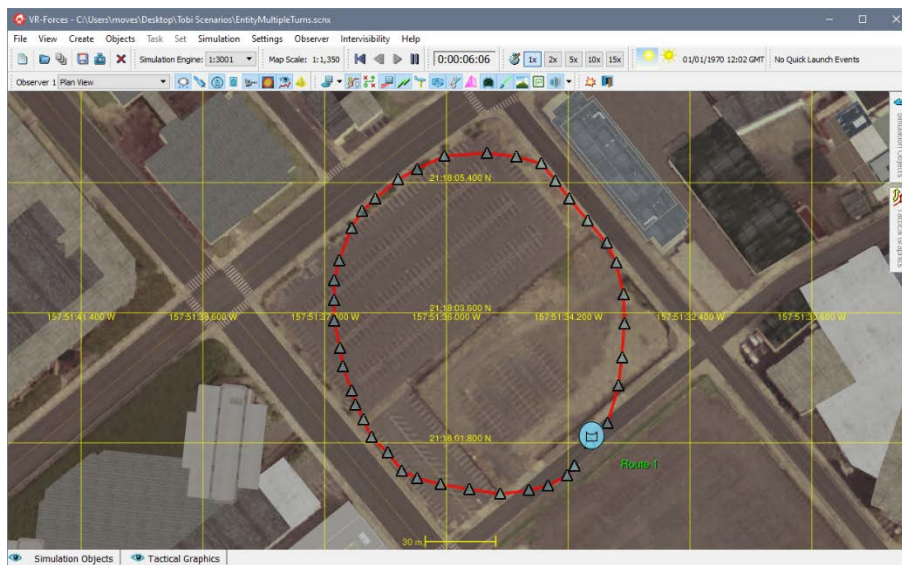


Figure 2. Entity in VR Forces application following a many-sided polygon

Simulations with only one entity are relatively rare. A normal simulation can have up to 1000 entities that generate PDUs. Not all entities move all the time. Therefore, approximately 100 PDUs per entity per minute is a reasonable estimate. A thousand entities times 100 PDUs per minute averages 6,000,000 PDUs per hour, or over 1500 PDUs per second.

These numbers are assumptions. A live run of the default HawaiiGround scenario in VR Forces simulation 52 with entities led to 4000 PDUs per minute which equals about 76.9 PDUs per minute per entity. Therefore, 100 PDUs per entity per minute is a good number to estimate the PDU load that must be handled reliably by network and storage.

Big Data characteristics are volume, variety, veracity, and velocity. These are known as the four V's of Big Data that create value that can be explored. Creating recorded LVC behavior streams together with searchable comprehensive metadata makes this entire field suitable for data science.

G. ORGANIZATION OF THIS THESIS

This thesis starts with background information on Web standards that are used in this research. We review the kind of products currently available on the market and the functionalities they lack. A short introduction to networking and distributed virtual environments (VE) is followed by information about PDU encodings.

Chapter III provides insights on shared behavior in 3D worlds focusing on the basics of DIS with its different types of PDUs and a short note on SPIDERS3D Virtual Environment.

Chapter IV describes the implementation of capturing, recording, and storing DIS packets with its different encodings. A demonstration is provided on how a PDU stream from an arbitrary simulation is converted into autogenerated code that is playable by every open-source software that can play X3D animations.

Chapter V provides insights into procedures for troubleshooting the process of developing new code and implementing it into a simulation. Verification, validation and accreditation (VV&A) is also addressed, because the worst error that can occur is the one that goes unnoticed.

Chapter VI summarizes the main points of this thesis and recommends future work.

II. RELATED WORK

A. WEB STANDARDS

To get a reliable solution that lasts multiple decades, it is fundamentally important to use open-source standards that are well-known and broadly distributed over the internet and the community of interest the solution is intended for. From a long-term perspective, such strategies enable all manner of repeatability.

X3D is a royalty-free open standard defined in XML to represent 3D objects arranged in scenes (Web3D, n.d.-a). XML has the big advantage in that it can be tested against a schema and multiple quality assurance (QA) tools (Web3D, n.d.-c). XML is the internet standard for writing well-defined documents. It can be used for cross-platform, inter-application data transfer.

Hyper Text Markup Language (HTML) is the primary publishing language of the World Wide Web (WWW or Web). The current stable version of the standard, HTML5.2, can transport all varieties of content, including videos, to the user. According to www.w3.org (W3, n.d.-a)

HTML is the World Wide Web's core markup language. Originally, HTML was primarily designed as a language for semantically describing scientific documents. Its general design, however, has enabled it to be adapted, over the subsequent years, to describe a number of other types of documents and even applications. (W3, n.d.-b)

To give the user an easy way to display X3D models, X3D source or links can be embedded in HTML. Currently there are two players available to embed X3D directly within HTML: X3DOM (X3dom, n.d.) and X_ITE (Create3000, n.d.-a). Formalization of multiple techniques to integrate X3D with HTML files is in progress as part of the Web3D Consortium X3D version 4 standardization efforts (Web3D, n.d.-e, p. 3).

B. 3D MODELING

Three-dimensional modeling for distributed simulation can be accomplished using X3D. X3D is an open-source, royalty-free, international standard for 3D graphics used on

the Web. X3D is designed to be interoperable with most 3D modeling tools by offering a compatible path to web publication. X3D also provides import and export publishing compatibility for many formats (Brutzman & Daly, 2007). X3D is scalable from simple 3D models up to large-scale multi-model environments (Brutzman & Daly, 2007).

1. Software for X3D

Four types of software are used for X3D: authoring tools, players, converters, and libraries.

a. X3D Authoring Tools

To create X3D models and scenes, authoring tools must be used. Tools can either come as a rich client like X3D-Edit (NPS Savage Research Group, n.d.-d), Blender (Blender Foundation, n.d.), Titania X3D Editor (Create3000, n.d.-b), Meshlab (Cignoni et al., 2008), or can be installed as a plugin in NetBeans, for example (Web3D, n.d.-f). Native XML authoring tools can also be used, such of XMLSpy by Altova (Altova, n.d.).

The simplest authoring tool is a text editor. It has no amenities, but can produce a valid X3D file, if the author is confident in their coding skills.

b. X3D Players

Since the X3D standard is an open-source standard, it can be found in numerous different players. The following widely-used players can be downloaded and used to display X3D models:

- FreeWRL is an Open Source, cross platform VRML2 and X3D compliant browser (SourceForge, n.d.-a).
- H3DViewer is a standalone X3D browser for Windows (GitHub, n.d.-b).
- Instant Reality is a framework that provides a comprehensive set of features to support Virtual Reality (VR) and Augmented Reality (AR). The framework is designed to support various industry standards, like VRML and X3D (Instant Reality, n.d.).

- Octaga Player is a X3D player by Octaga Visual Solution AS which is free for non-commercial use (Visco, n.d.).
- Castle Game Engine View3DScene is a viewer for many 3D model formats like X3D, VRML, glTF, Collada, 3DS and many more (Kamburelis, n.d.).
- Xj3D is a Java project of the Web3D Consortium focused on creating a toolkit for VRML97 and X3D content (SourceForge, n.d.-b).
- X3DOM is an open-source framework and runtime to publish 3D graphics on the Web (X3dom, n.d.).
- X_ITE is a 3D JavaScript library that uses WebGL for 3D rendering (*Getting Started » X_ITE X3D Browser » CREATE3000*, n.d.).

Each player can display X3D models, but the number of functions implemented differs.

c. X3D Converters

The following X3D converters simplify conversions between different data formats.

- VRMLout for AutoCad and VRML Translator for Inventor (CadStudio, n.d.).
- X3D In/Out Toolkit(Sons, 2013/2019).

d. X3D Libraries

The following X3D Libraries can be used as a basement upon which to build other X3D software.

- X3D Java Scene Access Interface Library (X3DJSAIL) (Web3D, n.d.-g).

- X3D Python Scene Access Interface Library (X3DPSAIL) (Carlsonsolutiondesign, 2019/2019).
- X3DOntology for Semantic Web (Web3D, n.d.-b).
- X3DJSONLD is a local node.js webserver for serving files from localhost (Carlson, 2015/2020).

2. Comparison of X3D Players

To compare the different standards, displayed outputs in three different models with varying difficulty were used: Hello Germany, Olympic Rings, and Lighthouse. Another important point is to test for international text with special characters as X3D fully supports Unicode.

For each of the three scenes, six screenshots and a table are provided. Swirl did not display one scene properly, so it was removed from the comparison in order to not distract the reader with all black screenshots. Xj3D Plugin and Xj3D Player both use the same source code, so Xj3D Plugin is used for screenshots.

a. Hello Germany Scene

The first scene set up for testing the players incorporates three blocks with three different colors and two different text shapes. The intent of one of the text shapes is to test the player's ability to support localization (l10n) and internationalization (i18n) (W3, n.d.-c).

To see the scene displayed by X_ITE, scan the QR code in Figure 3 or visit http://x3dgraphics.com/examples/X3dForAdvancedModeling/HelloWorldScenes/HelloGermanyX_ITE.html.



Figure 3. QR code for Hello Germany Model displayed by X_ITE

(1) FreeWRL

FreeWRL displays the scene with special characters but the flag is too close to the upper text field. The text is too large (Figure 4).



Figure 4. FreeWRL default view of Hello Germany scene

(2) H3DViewer

H3DViewer correctly displays the scene with special characters. The flag is positioned correctly (Figure 5).



Figure 5. H3DViewer default view of Hello Germany scene

(3) Instant Reality

Instant Reality correctly displays the scene with special characters. The flag is positioned correctly (Figure 6).



Figure 6. Instant Reality default view of Hello Germany scene

(4) Octaga Player

Octaga Player correctly displays the scene with special characters. The flag is positioned correctly (Figure 7).



Figure 7. Octaga Player default view of Hello Germany scene

(5) Castle Game Engine View3DScene

View3DScene correctly displays the scene without special characters (the issue was reported and will be fixed in a later version). The flag is too close to the upper text shape (Figure 8). Note: The default bounding box was disabled.



Figure 8. View3DScene default view of Hello Germany scene

(6) Xj3D

Xj3D displays the scene with special characters correctly. The flag overlaps the top text shape (Figure 9).



Figure 9. Xj3D default view of Hello Germany scene

Table 1. Comparison of player performance for Hello Germany

Name of Player	Vertical Alignment	Special characters	Resolution and size of text
FreeWRL	Incorrect	Yes	Too large text, but resolution is high.
H3DViewer	Correct	Yes	Correctly sized text and high resolution.
Instant Reality	Correct	Yes	Correctly sized text and high resolution.
Octaga Player	Correct	Yes	Correctly sized text and high resolution.
View3DScene	Correct	No	Too large text and low resolution.
Xj3D	Correct	Yes	Too small text but high resolution.

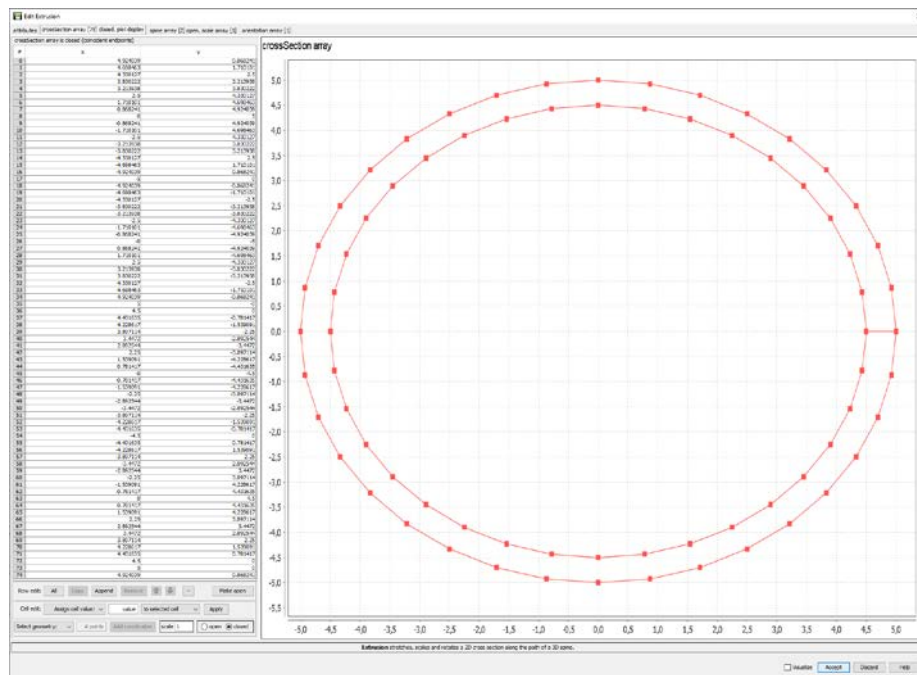
(7) Summary Assessment

For the first scene with three blocks and two text shapes, five out of seven players provide perfect spacing. One player lacks special characters. Details are listed in Table 1. The simple presentation of interoperable geometry, text display and internationalization (I18n) of characters in a 3D virtual environment is a widespread capability available for further use.

b. *Olympic Rings X3D Model*

The second X3D model set up for testing players is made of extrusion nodes. DEF and USE are used for defining one ring and reusing it with different colors. The rings are slightly rotated by a few degrees in the x-y-plane to make the intersections more effectively visible.

The extrusion is set up as a ring that is elevated on the x-axis by a spine of 1 meter (Figure 10).



Note connecting segment for this closed strip on right-hand side.

Figure 10. Plane crosssection view of the X3D model of a ring extrusion

The first iteration of the model had an issue with intersecting inner and outer circles. The computed circle points were created using an Excel spreadsheet. The problem with the original computations was that the spreadsheet calculated the points counterclockwise separately for both rings, leading to self-intersecting geometry. This error leads to the tessellation issue shown in Figure 12: The last point of the inner ring connects to the first point of the outer ring.

Although this is not a valid X3D model because it has a self-intersecting extrusion, interestingly Octaga Player and Instant Reality handled it.

After opening an issue on the Github page for View3DScene (GitHub, n.d.-a), Michalis Kamburelis pointed out the issue with the model (Web3D Consortium, 2020b). After fixing the model (Figure 13) four of the six tested players displayed the model perfectly. Visual debugging techniques by Dr. Kamburelis (Figure 11) were instrumental in diagnosing this obscure error. Figure 11 shows the erroneous geometry characteristics, excerpted using the Castle Game Engine View3DScene diagnostic testing.

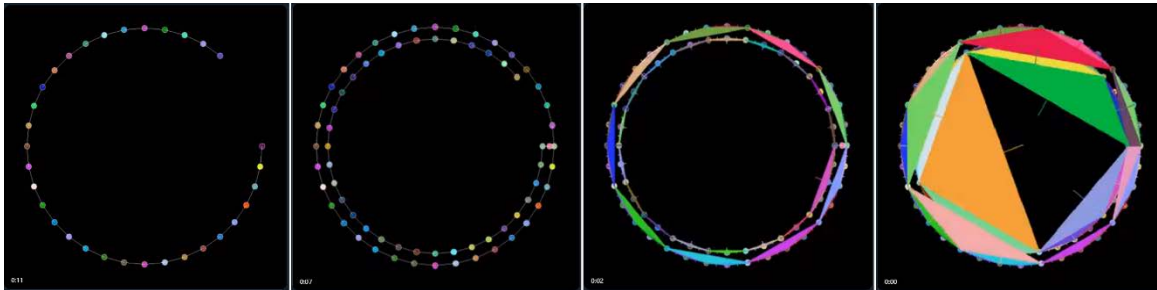


Figure 11. View3DScene Diagnostic Testing for Extrusions

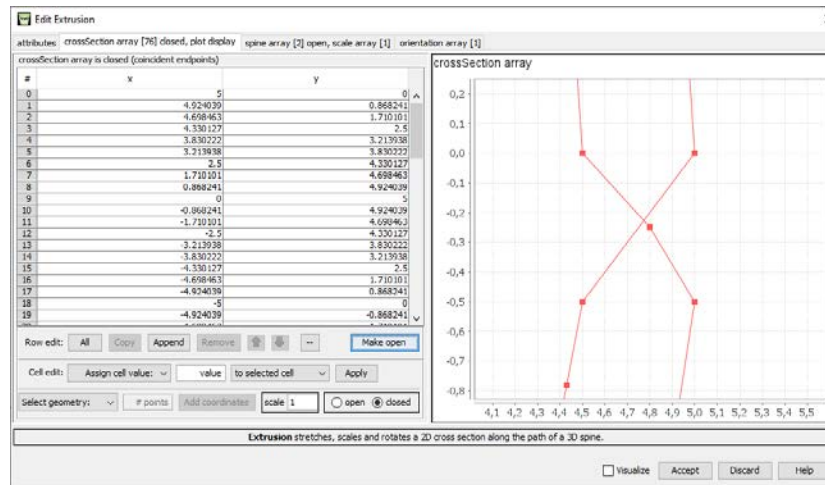


Figure 12. Intersecting inner and outer ring of the model leads to incorrectly formed geometry and erroneous results

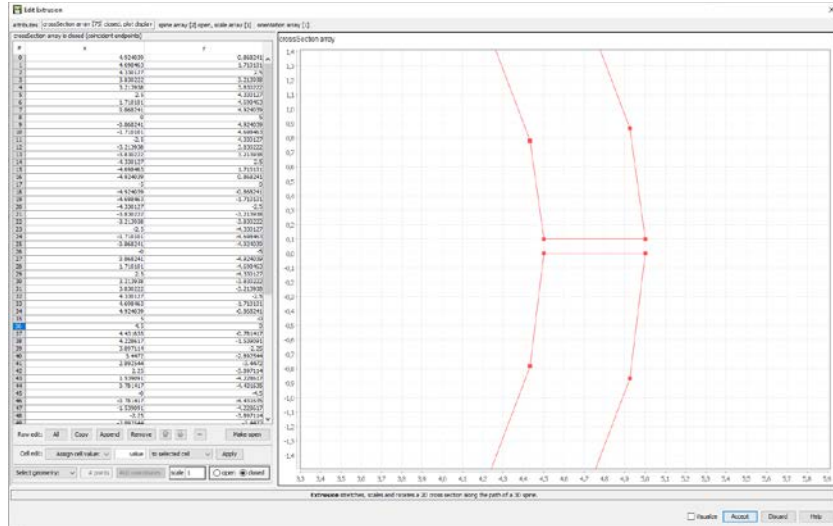


Figure 13. Fixed model with open ring as a single convex strip rotated around a circle

To see the scene displayed by X_ITE scan the QR code in Figure 14 or visit http://x3dgraphics.com/examples/X3dForAdvancedModeling/GeometricShapes/OlympicRingsX_ITE.html.



Figure 14. QR code for Olympic Rings URL displayed by X_ITE

(1) FreeWRL

FreeWRL displays the scene with the rings in a perfect shape. The intersections are handled perfectly. The lighting of the scene is correct (Figure 15).

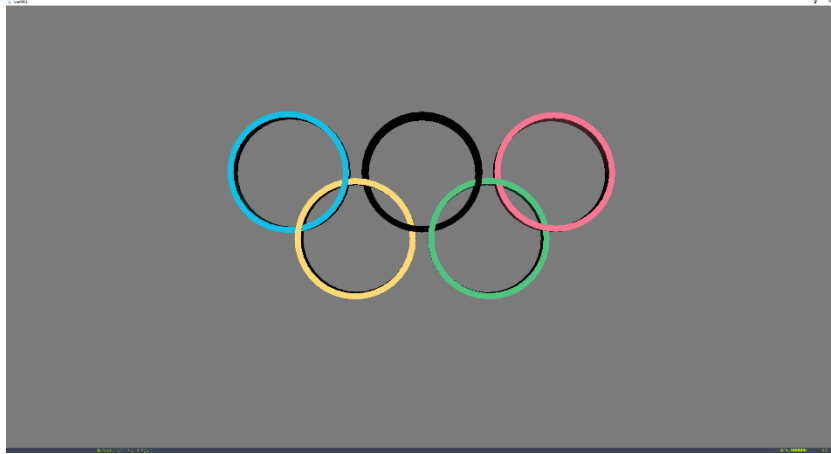


Figure 15. FreeWRL default view of the Olympic Rings scene is correct

(2) H3DViewer

H3DViewer has difficulties viewing convex extrusions. The intersections are noticeable. The lighting of the scene is correct (Figure 16).

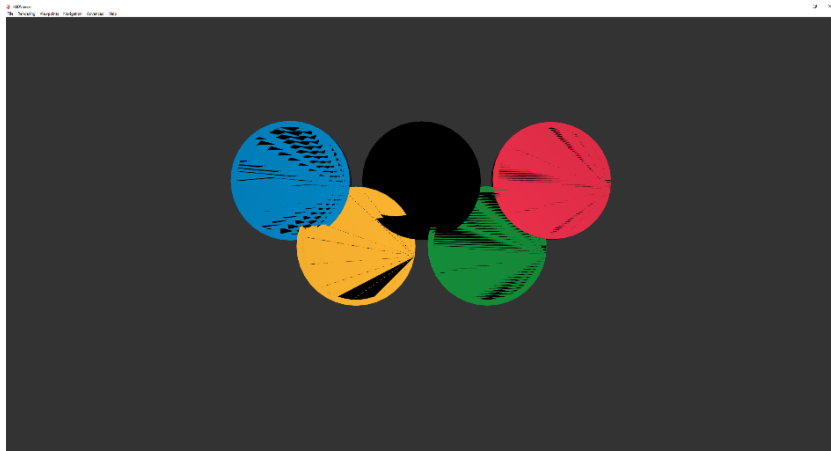


Figure 16. H3D default view of the Olympic Rings scene is erroneous

(3) Instant Reality

Instant Reality handles extrusions correctly. Intersections and light are correct (Figure 17).

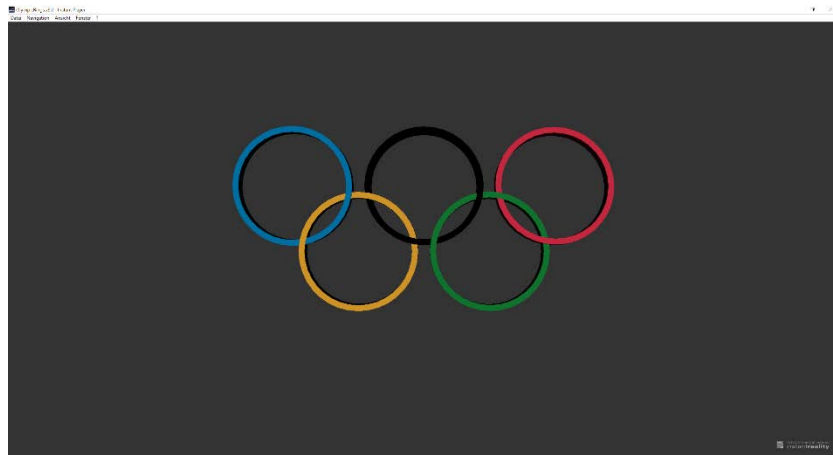


Figure 17. Instant Reality default view of the Olympic Rings scene is correct

(4) Octaga Player

Octaga Player handles extrusions, intersections, and light correctly (Figure 18).

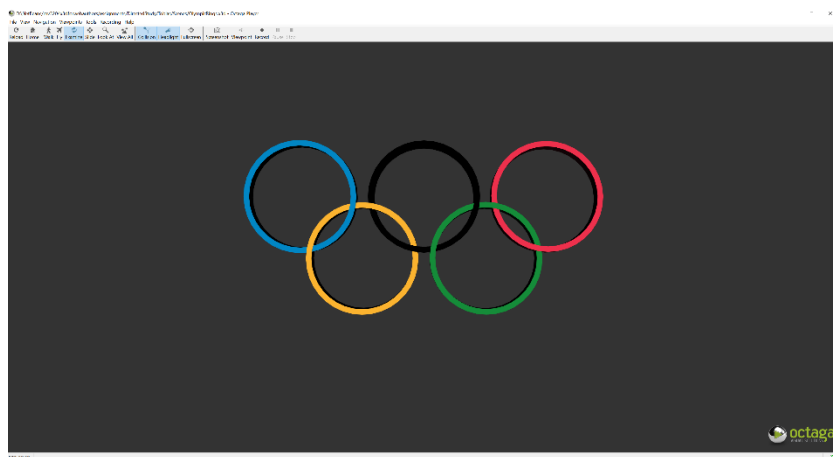


Figure 18. Octagon Player default view of the Olympic Rings scene is correct

(5) View3DScene

View3DScene handles extrusions, intersections, and light correctly (Figure 19).

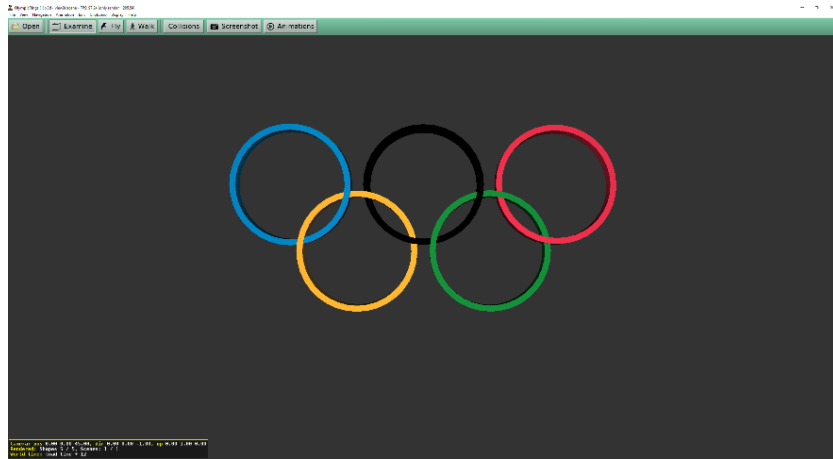


Figure 19. View3DScene default view of the Olympic Rings scene is correct

(6) Xj3D

Xj3D 2.2 handles extrusions, intersections, and light correctly (Figure 20). This open-source codebase is actively maintained and improved by NPS for Web 3D (NPS Savage Research Group, n.d.-b).

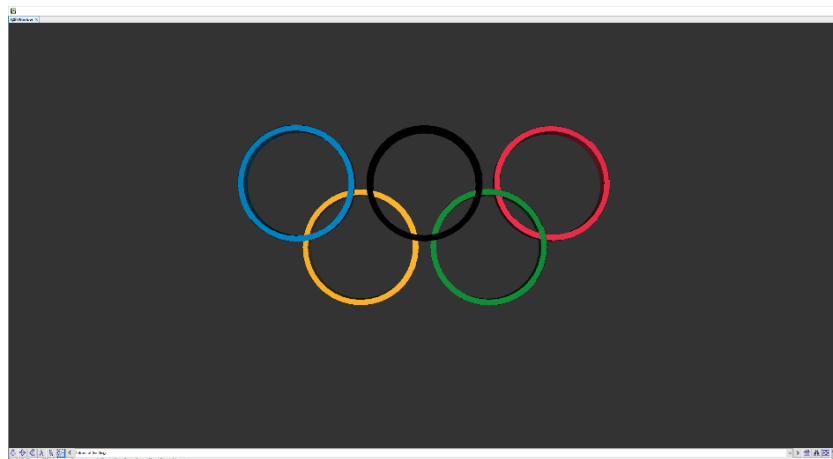


Figure 20. Xj3D default view of the Olympic Rings scene

Table 2. Comparison of player performance for Olympic Rings

Name of Player	Extrusion	Intersections	Light
FreeWRL	Excellent	Excellent	Excellent
H3DViewer	Incorrect	Incorrect	Excellent
Instant Reality	Excellent	Excellent	Excellent
Octaga Player	Excellent	Excellent	Excellent
View3Dscene	Excellent	Excellent	Excellent
Xj3D Plugin	Excellent	Excellent	Excellent

(7) Summary Assessment

For the second scene with extrusions, DEF, USE and intersecting objects, Octaga Player, Instant Reality, View3Dscene and Xj3D Plugin perform best. Thus, more complex X3D geometry computed at real time is an available capability for shared Web-compatible virtual environments.

c. Lighthouse Scene

The third X3D scene for testing players is a complicated setup with routes between TimeSensors, OrientationInterpolators and TouchSensors nodes. A beam cone is loaded from the Savage X3D Example Archive as prototype and imported into the scene. The beam cone is geometry, semitransparent with lines and a PointLight node. The lighting of the scene is separate. Detailed descriptions and specification links for these critical animation nodes are available in the X3D Tooltips (Web3D, n.d.-d).

(1) FreeWRL

FreeWRL displays the scene with the imported prototype. Animation and lighting with action buttons work (Figure 21).

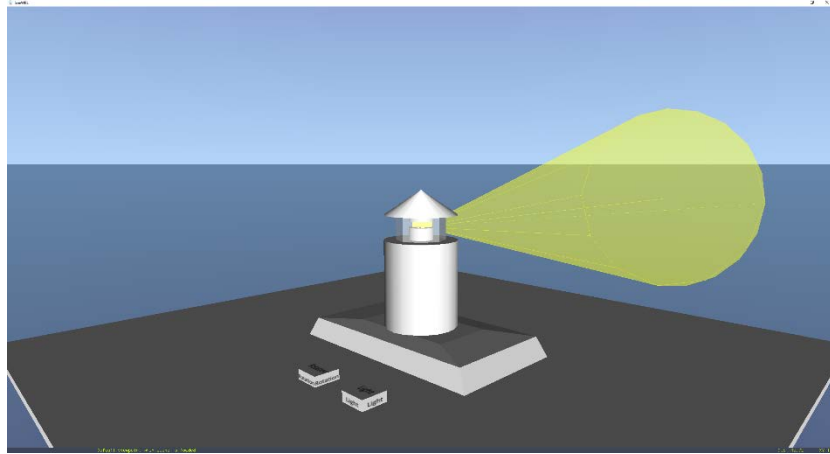


Figure 21. FreeWRL default view of the Lighthouse scene is correct

(2) H3DViewer

H3DViewer displays the scene without the imported prototype (Figure 22). Animation and lighting with action buttons do not work despite model testing using X3D validator (NPS Savage Research Group, n.d.-c).

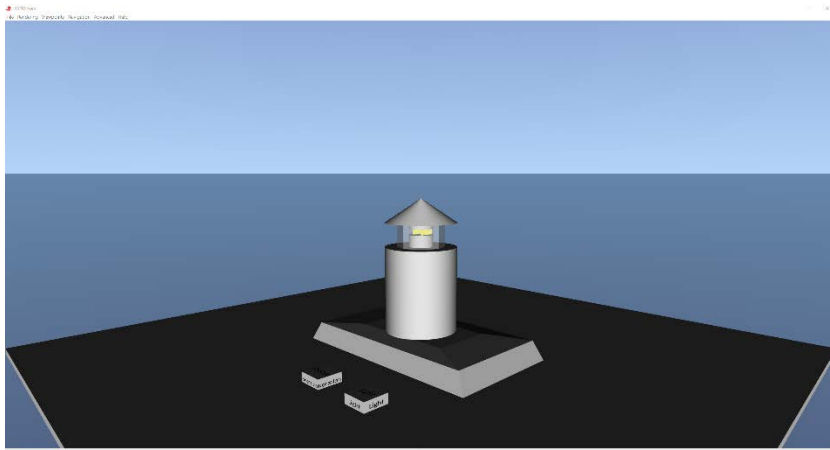


Figure 22. H3D default view of the Lighthouse scene without beam cone

(3) Instant Reality

Instant Reality displays the scene without the imported prototype. Animation and lighting with action buttons work (Figure 23).

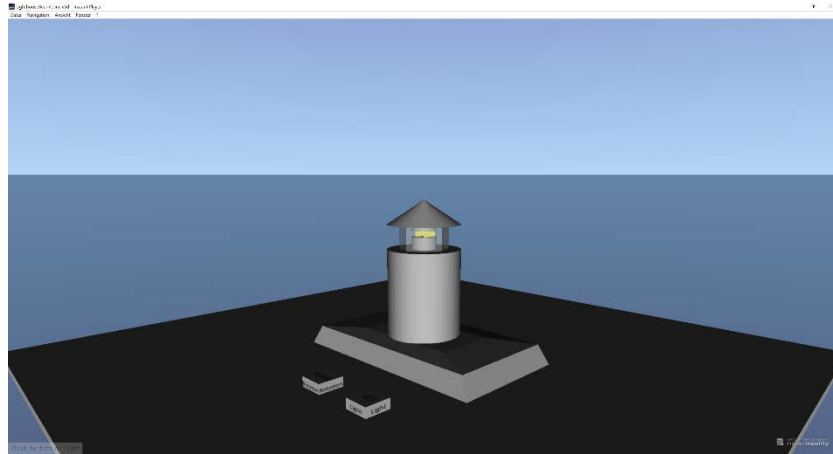


Figure 23. Instant Reality default view of the Lighthouse scene without beam cone

(4) Octaga Player

Octaga Player displays the scene with the imported prototype. Animation and lighting with action buttons work (Figure 24).

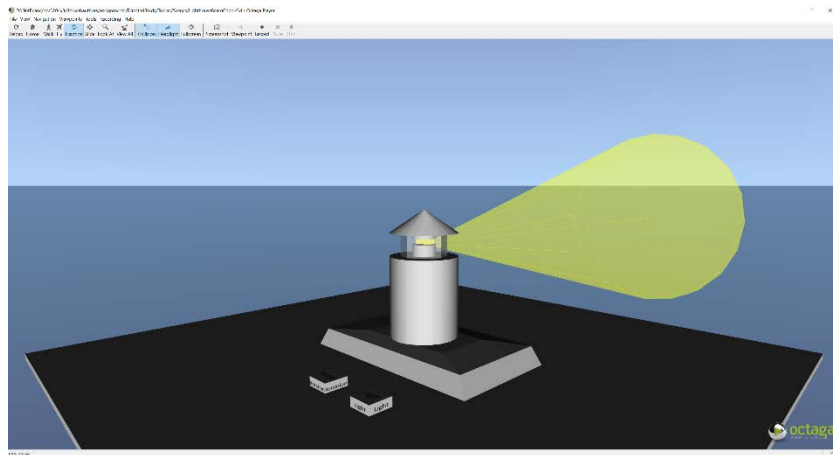


Figure 24. Octaga Player default view of the Lighthouse scene is correct

(5) View3DScene

View3DScene displays the scene without the imported prototype. A message on the console states that imports are not support as of the time of the test. Animation and lighting with action buttons work (Figure 25).

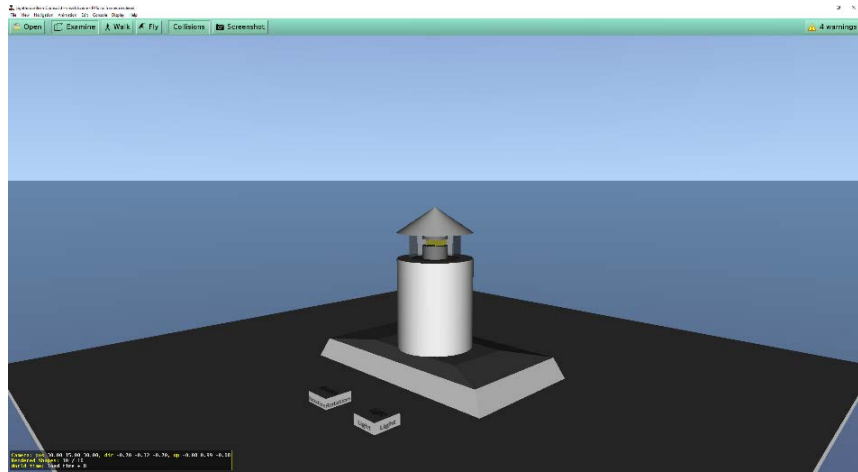


Figure 25. View3DScene default view of the Lighthouse scene without beam cone

(6) Xj3D

The Xj3D displays the scene with the imported prototype. Rotation does not work, but lighting with action buttons works (Figure 26).

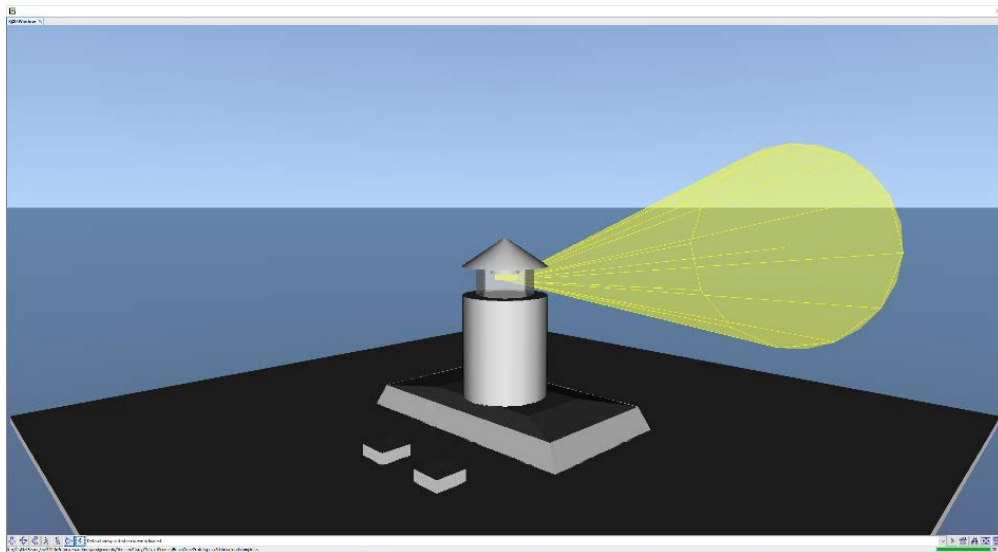


Figure 26. Xj3D default view of the Lighthouse scene

Table 3. Comparison of player performance for Lighthouse with beam cone

Name of Player	Import Prototype	Rotation, Light	Notes
freeWRL	Correct	Yes, Yes	
H3DViewer	Incorrect	No, No	Light is displayed within the Lighthouse.
Instant Reality	Incorrect	Yes, Yes	Rotation is working without cone being displayed.
Octaga Player	Correct	Yes, Yes	
View3DScene	Incorrect	Yes, Yes	Rotation is working without cone being displayed. Light is displayed within the Lighthouse.
Xj3D Plugin	Correct	No, Yes	Light is displayed within the Lighthouse.

(7) Summary Assessment

For the lighthouse scene with rotation, touch sensors and an imported prototype freeWRL and Octaga Player perform best. Once again, these tests provide high confidence that frequently used animation and interaction techniques can be widely available to support Web-standard compatible VEs. Animation chains using X3D TimeSensor, PositionInterpolator, and OrientationInterpolator nodes are vitally important for DIS behavior stream visualization.

3. X3D TimeSensor Node

A X3D TimeSensor node is used as a simulation clock to provide an animation time that can be used as an input to interpolators. The flag `Loop = "true"` starts the TimeSensor with the scene and keeps it running repeatedly after it reaches the end. The TimeSensor must have a name defined with a DEF tag to make it addressable to the ROUTE connection to corresponding interpolator nodes. A cycle interval of 1 means that one complete cycle lasts 1 second (Figure 27).

```
<TimeSensor DEF = "BoxTimeInterval" cycleInterval='1'
loop='true' />
```

Figure 27. Code excerpt for a X3D TimeSensor shows XML encoding of keyfields in this node

4. X3D PositionInterpolator Node

A PositionInterpolator node provides interpolated values between two coordinates to smooth animations. X3D uses the two fixed coordinates to approximate the position between those two points over time of an entity animated by the PositionInterpolator.

For every value in “key” there must be a tuple of floats in “keyValue” representing the x, y, and z coordinates. A code sample for a PositionInterpolator can be found in Figure 28.

```
<PositionInterpolator DEF='Entity' key='0.0 0.25 0.5 0.75
1.0' keyValue='0 0 0 0 1 0 0 2 0 0 1 0 0 0 0 />
```

Figure 28. Code excerpt for a X3D PositionInterpolator shows XML encoding of keyfields in this node

5. X3D OrientationInterpolator Node

An OrientationInterpolator node provides interpolated values between two fixed points to make animations smoother. A rotation around the y-axis has a start rotation value of 0 and an end rotation value of 6.283185 (2 Pi). The OrientationInterpolator provides values between both values stretched over the cycle interval of the TimeSensor (Figure 27). Intermediate values are included to avoid direction ambiguity. Therefore, one complete rotation around the y-axis takes 1 second. A code sample for an OrientationInterpolator can be found in Figure 29.

```
<OrientationInterpolator DEF='SpinEntity' key='0.0 0.25 0.5  
0.75 1.0' keyValue='0 1 0 0 0 1 0 1.57 0 1 0 3.14 0 1 0 4.71  
0 1 0 6.28/>>
```

Figure 29. Code excerpt for a X3D OrientationInterpolator shows XML encoding of keyfields in this node

C. NETWORKING AND DISTRIBUTED VIRTUAL ENVIRONMENTS

The introduction of this thesis describes how different MOTS simulations work together using a network within a lab. In the MOVES LVC lab, for example, the Marine Air-Ground Task Force (MAGTF) Tactical Warfare Simulation (MTWS) and MAK Inc.'s VR Forces can be set up to use a network to share ESPDUs. Every simulation runs its own entities populating their state to the network so that all other simulation applications connected to the network can populate them within their simulation space. Every entity has one simulation that oversees its state. It must maintain the location, current heading, current speed, and dead reckoning parameters. With an ESPDU being sent out over the network, one simulation informs all other participating simulations about location, current heading, current speed, and dead reckoning parameters, so that they can use them in their own simulation space and display them to the user.

For military applications, live, virtual, and constructive (LVC) entities are used that can share one simulation space. For example, live tanks with soldiers drive around the Ft. Irwin National Training Center, supported by virtual CAS aircrafts controlled by an aircraft simulator at any base in the U.S., being threatened by constructive hostile artillery operated by civilian contractors sitting at computers in a control room at the simulation center. To get an even bigger picture in this simulation, constructive entities are added to the simulation space to reach a higher level of real-world fidelity.

To connect LVC applications bridges and gateways are used. Bridges and Gateways are critical to most distributed Live, Virtual, and Constructive (LVC) simulation environments. The role of bridges and gateways is often misunderstood by distributed simulation planners. A good source to read more on gateways and bridges is the tutorial

“Gateways in Distributed Simulation” given at Simulation Innovation Workshop in February 2020. Inquiries can be made at the SISO university webpage (SISO, n.d.-b).

D. COMMAND AND CONTROL SYSTEM TO SIMULATION SYSTEM INTEROPERATION (C2SIM)

C2SIM is an international standard published by the Simulation Interoperability Standards Organization (SISO) for information interchange across C2 systems, simulation systems, and robotic and autonomous systems (RAS). These interconnections are required to support live/virtual/constructive training. Additional related requirements are exercise replay, mission planning, mission rehearsal, and mission re-creation (Dr. J. M. Pullen et al., 2019). The Military Scenario Definition Language (MSDL) and the Coalition Battle Management Language (C-BML) are prior standards being replaced by C2SIM.

The NATO Modelling and Simulation Group (MSG) 145 is working on “Operationalization Of Standardized C2-Simulation Interoperability” (NATO S&T Organization, 2018). To get a good understanding of C2SIM it is useful to read the C2SIM vision by NATO SMG 145:

We are working toward a day when the members of a coalition interconnect their networks, command, and control (C2) systems, and simulations simply by turning them on and authenticating, in a standards-based environment. A C2SIM Coalition is a system of systems (M. Pullen et al., 2018).

There are strong interrelationships between NATO and SISO. On the one hand the NATO MSG 145 depends on SISO for open industry-based standards, but on the other hand, SISO depends on NATO technical activities to field and validate C2SIM technology. This balance of imperatives produces excellent results.

C2SIM was designed for complex environments. According to (Brook, 2015, p. 1), a complex environment includes:

- more than one nation participating
- single or multiple domains
- multiple service branches

- multiple systems and networks
- distributed environments (multi-site operation) (Brook, 2015).

C2SIM does not specify a transport mechanism. Rather, it is intended for use in any distributed framework, such as that provided by HLA, DIS, or TENA for interconnecting simulation systems.

E. PDU ENCODINGS

The current DIS 7 version supports recorded DIS streams stored with BASE64 encoding (Josefsson, 2006, p. 64). For future versions of DIS, some more file formats are useful. When conducting on this research, one plain text file format with comma separated values (CSV) was used (see Chapter IV.C.2). Other file formats can be added to DIS7 by adapting the existing source code.

Possible future file formats include:

- Plain Text: PDU Type, Values (both CSV and TSV with comments appended),
- Binary Stream (matching the stream on the wire),
- JSON, minified JSON, EXI4JSON (numeric arrays and objects),
- XML fully documented PDU, with and without default values (using schema validation),
- EXI (Debich, 2015; Hill, 2015; Snyder, 2010; Williams, 2009)
- MAK Data Logger format,
- MAK Data Logger text format (pretty print), and
- Compressed DIS (C-DIS) (draft format for limited bandwidth situations) (SISO, n.d.-a)

Each type of encoding has its own advantages and disadvantages. A file that is compressed to save storage is often not human-readable. A file encoded with the MAK Data Logger Text Format (Figure 33) is relatively user friendly, but consumes large amounts of storage space. This encoding does have another disadvantage: it is not parsable. Therefore, once stored there is no way to read it back in to send to another simulation. For all other encodings, a matching player to parse files to send them back to a network is achievable.

F. PROPRIETARY ENCODINGS AND STANDARDIZATION CONSIDERATIONS

Of note, several commercially available tools (MAK data logger, Red Sims DIS PDU Recorder, Pitch, etc.) provide logging capabilities but record them using proprietary formats. This situation indicates that common shared logging capabilities can be achieved easily, and that converters are possible. Since logging formats are themselves archival for the session, lack of interoperability holds little value from a data-science or simulation perspective.

Since IEEE DIS specification defines bit formats explicitly, such logs can be considered authoritative. Since the existence of XML, EXI and JSON schemas can strictly define alternative encodings with exact syntactic equivalences, it follows that XML, EXI and JSON logs can be considered functionally equivalent to IEEE-compliant DIS binary and plain-text CSV/TSV stream recordings.

III. SHARED BEHAVIORS IN 3D VIRTUAL ENVIRONMENT (VE) SIMULATIONS

A. INTRODUCTION

Chapter III covers what is important to know when it comes to simulating using DIS. First, a small history on DIS is given. Second, recording of DIS streams is explained and software applications for that purpose are reviewed. Some main components of DIS are highlighted, then a list of all functional DIS areas and all types of PDUs is provided. An overview is also provided SPIDERS3D Virtual Environment.

B. DISTRIBUTED INTERACTIVE SIMULATION (DIS)

To achieve interoperability for the scenario described, DIS is one of the standards that can be used. DIS is a standard for running real-time simulations across multiple networks and computers hosting simulations. The first standard for DIS was defined in IEEE 1278–1993–Standard for Distributed Interactive Simulation–Application Protocols [1]. DIS is defined in IEEE Standard 1278.

The “SISO Reference for Guide: DIS Plain and Simple” (still in draft) (SISO, 2007) contains a good chapter covering the history of DIS:

The standard was developed over a series of “DIS Workshops” at the Interactive Networked Simulation for Training symposium, held by the University of Central Florida’s Institute for Simulation and Training (IST). The standard itself is very closely patterned after the original SIMNET distributed interactive simulation protocol, developed by Bolt Beranek and Newman for Defense Advanced Research Project Agency in the early through late 1980s. BBN introduced the critical concept of dead reckoning to efficiently transmit the state of battle field entities, as well implementing DARPA’s vision of simulations involving inexpensive general purpose computers (vs. 6DOF motion platforms and/or supercomputers), hundreds of online players (not just the ‘onesies and twosies’ which had been done before), wherein the realism and training value came not from high-fidelity simulation of vehicle dynamics but by the real time play with lots of intelligent allies and lots of intelligent opponents.

In the early 1990s, IST was contracted by the United States Defense Advanced Research Project Agency to undertake research in support of the U.S. Army Simulator Network (SimNet) program. Funding and research

interest for DIS standards development decreased following the proposal and promulgation of its successor, the High Level Architecture (HLA, initially entitled DIS++), in 1996. HLA was produced by the merger of the DIS protocol with the Aggregate Level Simulation Protocol (ALSP) designed by Mitre (SISO, 2007).

DIS usage is widely spread across Department of Defense, NATO, and allied nations. It is used for real-time simulation and occasionally for connecting virtual worlds to C2 systems. For DIS there are many distributions available for use. The one used for this thesis is Open-DIS Version 7 mainly developed by the MOVES Institute at the Naval Postgraduate School (NPS). OpenDIS is royalty-free and available to the public under <https://github.com/open-dis>.

The new OpenDIS7 is currently under development. It provides full coverage of the IEEE DIS version 7 specification. Support for full capabilities using Java is expected to be announced this year. Similar to the prior version 4 release of the OpenDIS library, auto generation of code pattern, from Java to other programming languages (Python, C#, XML, JSON, etc.) is expected.

C. AVAILABLE SOFTWARE

Because all entity states are sent to the network the simulations are connected to, it is obvious that it must be possible to record and play them back. For this thesis, a study of related work was conducted to find software that can record and playback PDUs. One open-source framework and two available COTS software products were found that can achieve the goal of recording and playback DIS streams.

1. OpenDIS7

The cheapest way, financially, to record and playback DIS streams is to utilize some classes of the OpenDIS7 distribution. OpenDIS7 provides many classes including examples that can be used to record and play back DIS streams including every available PDU type defined in the DIS standard (72 total). This approach is used and explained throughout this thesis and therefore not explained in detail at this point.

The older OpenDIS4 version provides classes that can also record DIS streams, although more programming work is necessary to achieve all the same capabilities as in OpenDIS7. Work on these projects is steadily progressing.

2. Redsim DIS PDU Recorder

Redsim's DIS PDU Recorder can record and playback and comes with a dual LAN option. It supports DIS streams sent via multicast and broadcast. The software can filter, capture or playback for certain family-related PDUs (RedSim, n.d.).

Redsim's DIS PDU Recorder lacks the capability to compress DIS streams or convert them into another format. Figure 30 shows a screenshot with some options.



Figure 30. Option panel of Redsim's DIS PDU Recorder

More information on Redsim's DIS PDU Recorder can be found at <http://www.redsim.com/products/dis-pdu-recorder.html>.

3. MAK Data Logger

MAK Data Logger was available for review in an already installed version on a computer in the MOVES LVC lab. MAK Data Logger can record and playback DIS PDUs. Playback is available in slow motion and fast forward. The main window of the MAK Data Logger displays bars correlating to the number of PDUs received per time unit (Figure 31). Similar to video streams, during recording and playback, annotations can be added to mark certain points on the timeline. Two type of annotation are available: time points and time frames. Annotations are important metadata that can help navigate within a stream of many PDUs. These can also be used to step through the important points of interest while giving an after-action briefing.

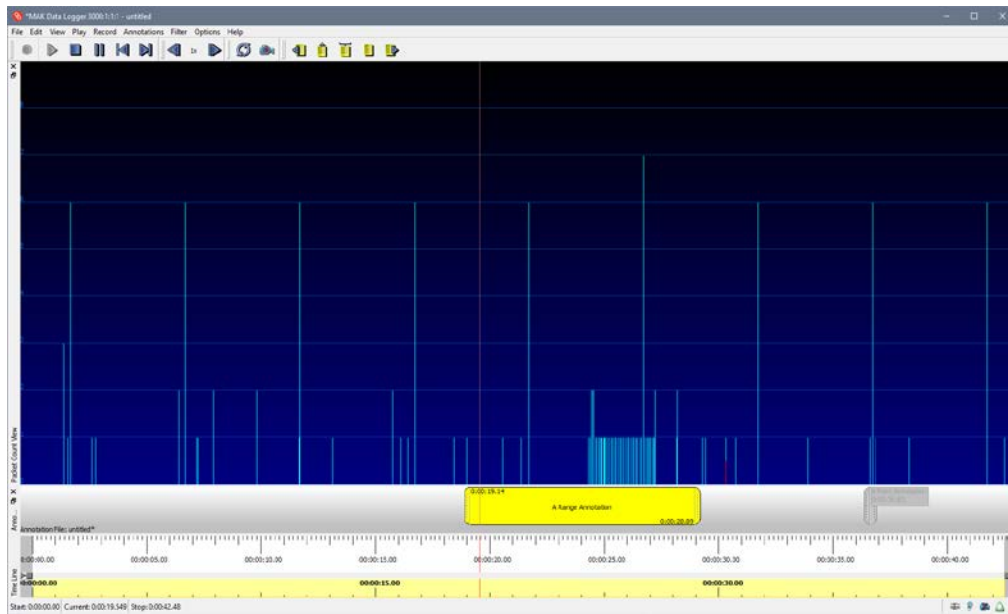


Figure 31. MAK Data Logger main window displaying bars and annotations

Another feature is the ability to filter for certain PDUs. This can be done with an extra dialog wherein PDU families can be checked or unchecked (Figure 32).

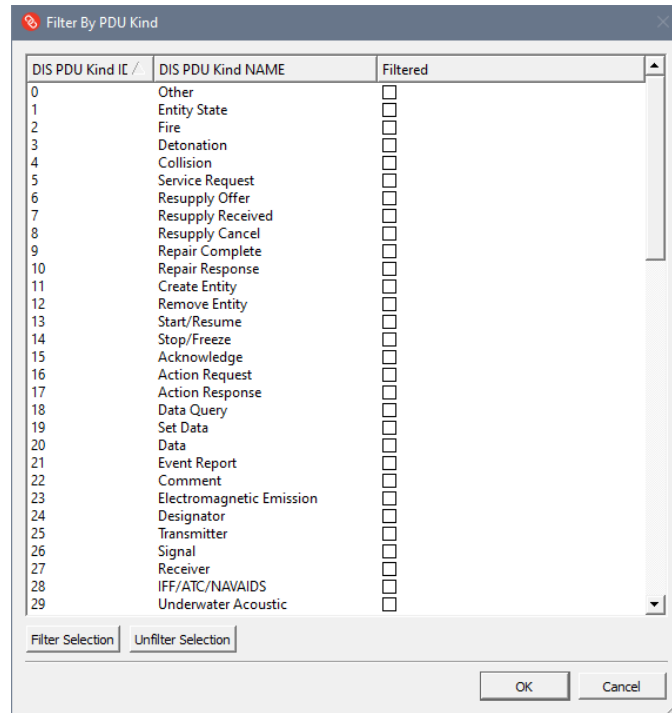


Figure 32. PDU Kind filter selection window in MAK Data Logger

The recordings can be saved to make a proprietary *.lgr format or exported to a human readable plain-text file (Figure 33).

```

*****
*
**** Packet #6      Size=864      time=0:00:01.3619, 20:57:36.1909 Mon Apr 13, 2020
#### WARNING: size mismatch: packet(864), DIS header(144) ####
***** Type=PDU #4 *****
PduKind:           EntityStatePduKind   (1)
Version:           7
Exercise:          1
ProtocolFamily:    FamilyEntityInteraction (1)
TimeStamp:         3456.19
TimeStampType:     Relative
receivedTime:      0
Size:              144
PDU Status (bits) : 00001000
Transferred Entity Indicator : PduStatusTransferredEntityIndicatorNoDifference
(0)
LVC Indicator :    PduStatusLVCIndicatorNoStatement (0)
Coupled Extension Indicator : PduStatusCoupledExtensionIndicatorCoupled (1)
-----
EntityId:          1:3001:12
EntityType:        1:1:222:6:1:0:0
ForceID:           ForceFriendly (1)
Location:          [ x= -5.50673e+06 y= -2.24105e+06 z= 2.30185e+06 ]
Velocity:          [ x= -1.55846 y= 8.67897 z= 4.68981 ]
Acceleration:      [ x= -0.265273 y= 1.47729 z= 0.798275 ]
Orientation:       [ Psi= 1.74847 Theta= -0.488808 Phi= 1.99472 ]
AngularVel:        [ x= 0 y= 0 z= 0 ]
DrAlgorithm:       DrDrmRvw (4)
NumArtParams:      0
NumAttachedParts:  0
Guise:             1:1:222:6:1:0:0
Appearance:        4194304
PaintScheme:       0
Immobilized:       FALSE
FirePowerKill:     FALSE
DamageState:       DamageNone (0)
EngineSmoke:       FALSE
SmokePlume:        FALSE
TrailState:        TrailingEffectsNone (0)
HatchState:        HatchNA (0)
LightState:        LightsNone (0)
Flames:            FALSE
Frozen:            FALSE
PowerPlant:        TRUE
FinalPdu:          FALSE
LauncherRaised:    FALSE
CamouflageType:    DesertCamouflage (0)
Concealed:         FALSE
Tent:              FALSE
Ramp:              FALSE
Marking:           UtilVeh 1
CharSet:           1
Capabilities:       0

```

Figure 33. Plain Text output of a DIS PDU recorded by MAK Data Logger

The MAK Data Logger stores PDUs into an internal database keeping track of the order they arrived by using the PDU timestamp. They can be saved in a proprietary format. To replay a stream, the PDUs are pulled from the database and resent over the network as PDUs. Slow motion and fast forward are available. The amount of traffic on the network for every replay is the same as in the original simulation run. No conversion to X3D interpolators or compression to reduce the file size is available. Since no text-based structured encoding (such as CSV or TSV) is provided, further conversion is not currently practical.

More information on the MAK Data Logger can be found under <https://www.mak.com/products/link/mak-data-logger>.

4. Pitch Cross Domain Security (CDS) Gateway

Pitch Cross Domain Security (CDS) Gateway is a proxy application that can be used together with different security devices to match requirements and policies. It also includes tools to monitor and log data (Pitch Technologies, n.d.). The product description page and directed inquiries did not reveal whether all of the DIS-based data in the Real-time Platform Reference (RPR) FOM (SISO, 2015) log could be bridged to DIS or not.

D. BASIC ARCHITECTURAL COMPONENTS OF DIS

Each entity is run by a simulation application. One simulation application can run several entities. The simulation application is responsible for translating user interactions into messages sent to other simulation applications.

The messages sent over the network by a simulation application are ground truth for the entities controlled by the simulation application that sent the message. Receiving simulation applications are responsible for deciding whether a reported entity is visible by their own controlled entities or not.

The main components of the Entity State PDU used in this research are the timestamp and the location arguments. A PDU contains many more arguments used to display entities with high fidelity. A list of all information contained in an Entity State PDU can be found in (IEEE, 2012).

1. Entity State PDU (ESPDU) Timestamp Considerations

It is important to verify the correct format of an ESPDU. There is one definition in (IEEE, 2012, p. 323).

The definition in (IEEE, 2012, p. 323): “The scale of the time value contained in the most significant 31 bits of the timestamp shall be determined by letting zero represent the start of the hour and letting $2^{31}-1$ represent one time unit before the start of the next hour. The next hour then starts back at zero. This results in each time unit representing exactly $3600/(2^{31})$ s (approximately $1.67638063 \mu\text{s}$).”

Given that the time is reset to zero after one hour, every simulation on the network must be aware of this to interpret PDUs in the correct order.

Another definition for a time scale for ESPDUs sometimes used for simulations with a lower resolution is the Unix time system. It uses a signed 32-bit integer to give the number of seconds that have elapsed since 1 January 1970.

Using a timestamp scale, other than the one described first, causes chaos in an environment that has more than two simulations communicating.

2. ESPDU Location

ESPDUs contain four different kinds of information describing the location of an entity with respect to the world, velocity, orientation, and dead reckoning parameters.

3. Location with Respect to the World

The location with respect to the world is always specified as the origin of the entity coordinate system. Each entity is displayed as a 3D model. The origin of the 3D model matches the location of the entity with respect to the world coordinate system.

4. Velocity

Linear and angular velocity of an entity is important when it is in motion through three dimensions. In addition to velocity, acceleration can be helpful for dead reckoning. The velocity is always given with respect to the world coordinate system. Relative effects like wind do not apply and are added later.

5. Orientation

The orientation of an entity is described by three Euler angles. The three angles are used to transform from the world coordinate system into the entity's local coordinate system. The order of rotation about coordinate axes is z, y, x.

6. Dead Reckoning (DR)

Within the DIS specification a process called dead reckoning is described in detail to show how velocity and acceleration values are used to project existing position and orientation progress prior to the next ESPDU arriving. The process is designed to reduce the amount of PDUs sent over the network.

Normally a simulation keeps track of an entity with its model designed for that specific entity. In addition, the simulation uses a dead reckoning algorithm to estimate the entity's position, since the last update was received. If the difference between the actual position of the entity and the estimate using the dead reckoning process reaches a certain threshold an update PDU is sent (IEEE, 2012, p. 70). If there is no difference between the actual and the estimated position, then a PDU must be sent at least every 5 seconds (the "keep alive" interval), to prevent the entity from being inactive. All available dead reckoning algorithms are displayed in Table 4.

Table 4. Dead Reckoning formulas. Source: (IEEE, 2012)

Field	Model	Formula	Examples
1	STATIC	N/A	Static entities
2	DRM (FPW)	$P = P_0 + V_0 \Delta t$	Constant velocity (or low acceleration) linear motion
3	DRM (RPW)	1) $P = P_0 + V_0 \Delta t$ 2) $[R]_{w \rightarrow b} = [DR] [R_0]_{w \rightarrow b}$	Similar to DRM 2 but where orientation is required (e.g., visual simulation)
4	DRM (RVW)	1) $P = P_0 + V_0 \Delta t + \frac{1}{2} A_0 \Delta t^2$ 2) $[R]_{w \rightarrow b} = [DR] [R_0]_{w \rightarrow b}$	Similar to DRM 5 but where orientation is required (e.g., visual simulation)
5	DRM (FVW)	1) $P = P_0 + V_0 \Delta t + \frac{1}{2} A_0 \Delta t^2$	High speed (e.g., missile) or maneuvering at any speed
6	DRM (FPB)	1) $P = P_0 + [R_0]_{w \rightarrow b}^{-1} ([R1] V_b)$	Similar to DRM 2 but when body-centered calculation is preferred
7	DRM (RPB)	1) $P = P_0 + [R_0]_{w \rightarrow b}^{-1} ([R1] V_b)$ 2) $[R]_{w \rightarrow b} = [DR] [R_0]_{w \rightarrow b}$	Similar to DRM 3 but when body-centered calculation is preferred
8	DRM (RVB)	1) $P = P_0 + [R_0]_{w \rightarrow b}^{-1} ([R1] V_b + [R2] A_b)$ 2) $[R]_{w \rightarrow b} = [DR] [R_0]_{w \rightarrow b}$	Similar to DRM 4 but when body-centered calculation is preferred
9	DRM (FVB)	1) $P = P_0 + [R_0]_{w \rightarrow b}^{-1} ([R1] V_b + [R2] A_b)$	Similar to DRM 5 but when body-centered calculation is preferred

7. World Coordinate System

Locations in a simulation are expressed by a right-handed, geocentric system. It is called the world coordinate system. The origin of the world coordinate system shown in Figure 34 is located at the center of the sphere. The axes of the world coordinate system are labeled X, Y, and Z. One length unit in the simulated world corresponds to one meter in the real world.

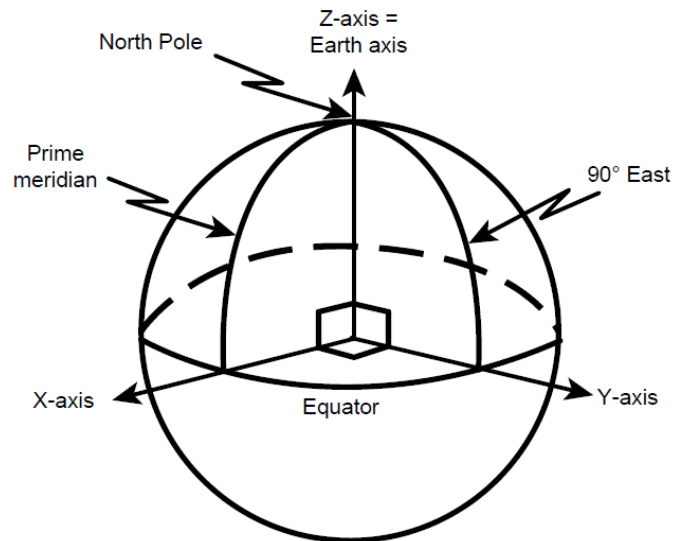


Figure 34. The “world coordinate system” shows how a geocentric coordinate system is arranged. Source: IEEE (2012).

8. Entity Coordinate System

Every entity in a simulation has an entity coordinate system. It is also a right-handed system. The center of the entities bounding volume shown in Figure 35 is the origin of the entity’s coordinate system. The origin of the entity coordinate system represents the entity’s location within the world coordinate system (IEEE, 2012).

In comparison with the world coordinate system, an entity’s coordinate system is relative to the given rigid body and axes are labeled with the lower case letters x, y, and z.

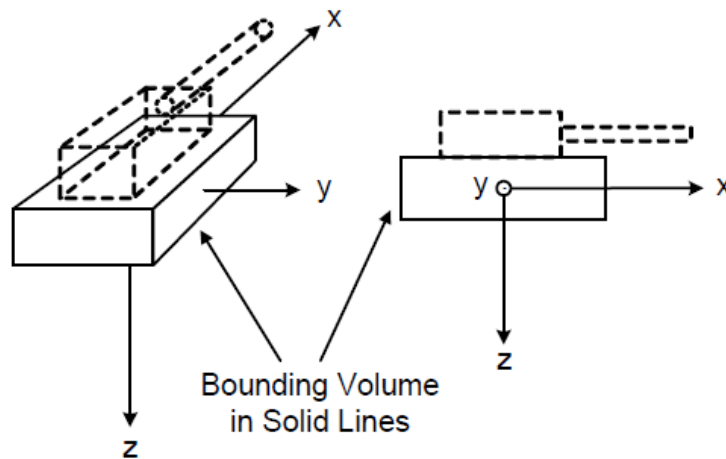


Figure 35. Entity coordinate system. Source: IEEE (2012).

E. TYPES OF PDUS

There are 72 types of PDUs in 13 functional areas. The 13 functional areas are (IEEE, 2012):

- Entity Information/Interaction

All PDUs that provide basic entity and entity collision information are listed under the functional area of Entity Information/Interaction.

- Warfare

All PDUs that provide basic warfare information (e.g., firing or launch of weapons, detonation of ammunition) are contained in the functional area of Warfare.

- Logistics

PDUs that model means of logistics like repair and resupply logistics are modelled within the functional area of Logistics.

- Radio Communications

All PDUs modeling transmitting or receiving interactions between entities are modelled within the functional area of Radio Communications.

- Simulation Management

PDUs in the functional area of Simulation Management are used to manage an exercise and facilitate the operation of the network the exercise is running on.

- Distributed Emission Regeneration

All PDUs emitting waves or beams (e.g., radar, laser, sonar) are summarized under the functional area of Distributed Emission Regeneration.

- Entity Management

The functional area of Entity Management provides tools to support larger DIS exercises where it is helpful to aggregate or group entities and keep track of their aggregation or grouping status. Additionally, protocols for transferring the ownership of an entity between two simulations are included.

- Minefield

The simulation of minefields and single mines is contained within the functional area of Minefields.

- Synthetic Environment

All nonentities like weather, diurnal effects, and natural and human-made disturbances are handled within the functional area of Synthetic Environment.

- Simulation Management with Reliability

Same functional area as Simulation Management but with built-in reliability so critical management tasks are completed even if individual PDUs are dropped.

- Information Operations

All operations that interrupt the enemy's decision-making process are handled within the functional area of Information Operation (e.g., electronic warfare, computer network operations).

- Live Entity (LE)

Live entities that are reporting their position or heading via DIS uses the functional area of Live Entity Information/Interaction to account for limited bandwidth by using smaller footprints.

- Non-Real-Time Protocol

The functional area of Non-Real-Time Protocol is used to run an exercise slower or faster than real-time.

A list with all 72 different PDUs can be found in Appendix A.

F. SPIDERS3D VIRTUAL ENVIRONMENT (VE)

SPIDERS3D Virtual Environment is an online collaboration platform using geographical data enriched by X3D models that are driven by DIS streams. SPIDERS3D Virtual Environment can interface with a Web browser accessing a designated SPIDERS3D Virtual Environment server. The official page for SPIDERS3D Virtual Environment contains a useful summary:

SPIDERS3D is a N4 facilities logistics platform that delivers interactive, geospatially accurate, real-world 3D Virtual Environments to improve systems engineering and advance planning communications to identify risks for introducing new weapon platforms into the shore environment.

Naval Enterprise engineers and planners use the SPIDERS3D tool to support major defense acquisition programs by visualizing platform-to-shore

interface design concepts and by identifying potential incompatibilities and risks early in the design process. Consequently, this enables earlier consideration of program alternatives for installations worldwide.

One critical aspect of SPIDERS3D is that it allows multiple users across the DOD enterprise to conduct real-time 3D collaboration over the Web on their desktop. The platform currently contains more than 100 virtual real-world installations and bases and locations, and over 300 different 3D models of weapon platforms, shore support equipment, shore infrastructure and facilities.

The Web-based Extensible 3D (X3D) technology implemented in SPIDERS3D provides a framework to increase 3D data interoperability. Extensible 3D is a royalty-free, open-standard file format and run-time architecture to represent and communicate 3D scenes and objects using extensible markup language. This International Standard (IS) also enables repurposing of other 3D data, and is complementary with emerging digital thread technologies, such as 3D scanning and 3D printing.

The strategic need for increased agility and creativity in our technical coordination activities has never been more important in this time of COVID-19 forced isolation. SPIDERS3D provides a ready and scalable capability to sustain remote connectedness, and accelerate technical collaboration across DOD, Allies and partner nations via the Web (Brutzman, n.d.).

Future work by NPS has been proposed, to continue porting OpenDIS7 library capabilities to JavaScript and JSON so that direct DIS streams connected from a gateway bridge emitting DIS streams originating from heterogenous LVC behavior sources. The age of open data standards (such as DIS) and open-source code makes such an activity fundamentally valuable with long-term stability and repeatability expected as a logical outcome of these reliable design principles.

G. SUMMARY

Chapter III focused on what is important to know when it comes to DIS simulation. After a short overview of DIS history, the chapter covered recording of DIS-streams and COTS products available on the market. Some main components of DIS were highlighted to get the user comfortable with the topic of DIS. A list of all 13 functional areas with its 72 PDUs was provided although the next chapters mainly focus on ESPDUs (PDU Type 1), as well as SPIDERS3D, its use of X3D, and its planned connections to DIS.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. IMPLEMENTATION AND DEMONSTRATION

A. INTRODUCTION

Chapter IV describes all the tools needed to make a DIS stream driving a X3D model. This is needed to promote DIS behavior streams as first-class media types in the future. We start with capturing DIS packets using Wireshark to show how to make simulation data visible. Recording and playing DIS streams to and from files in different formats is described to offer a manual to follow when this area of knowledge is new to the reader.

Transforming a PDU stream into autogenerated code that can be used to drive a X3D model, the key part of this thesis, is described in IV.G and IV.H.

B. CAPTURING DIS PACKETS

For analysis it is often useful to capture DIS packets as they are sent over ethernet. This can be achieved using software that captures all network traffic on a specific network device. Useful software for this purpose is Wireshark (WireShark, n.d.).

One possible use case for capturing DIS packets might be the following: DIS packets are broadcast from a local machine running a Coalition Battle Management Language (CBML) application (IP: 192.168.188.87) to the broadcast address of the local network (IP: 192.168.188.255).

After installing Wireshark to a machine on the local network, Wireshark must be started. The startup screen of Wireshark shows a selection dialog (Figure 36) for the target network interface to capture packets from. For most of our LVC environments it is the ethernet interface.

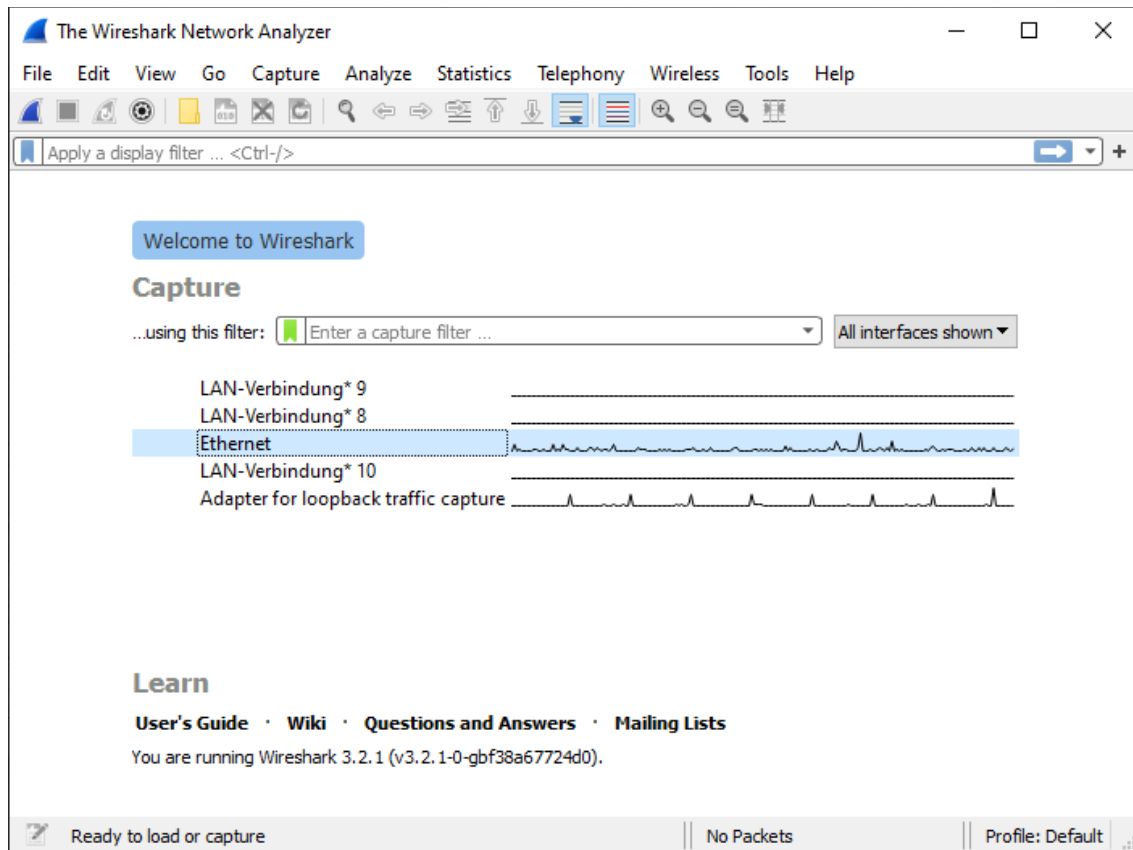


Figure 36. Dialog to select network interface

To begin Wireshark in capture mode, a user double-clicks on one of the displayed network interfaces (Figure 37).

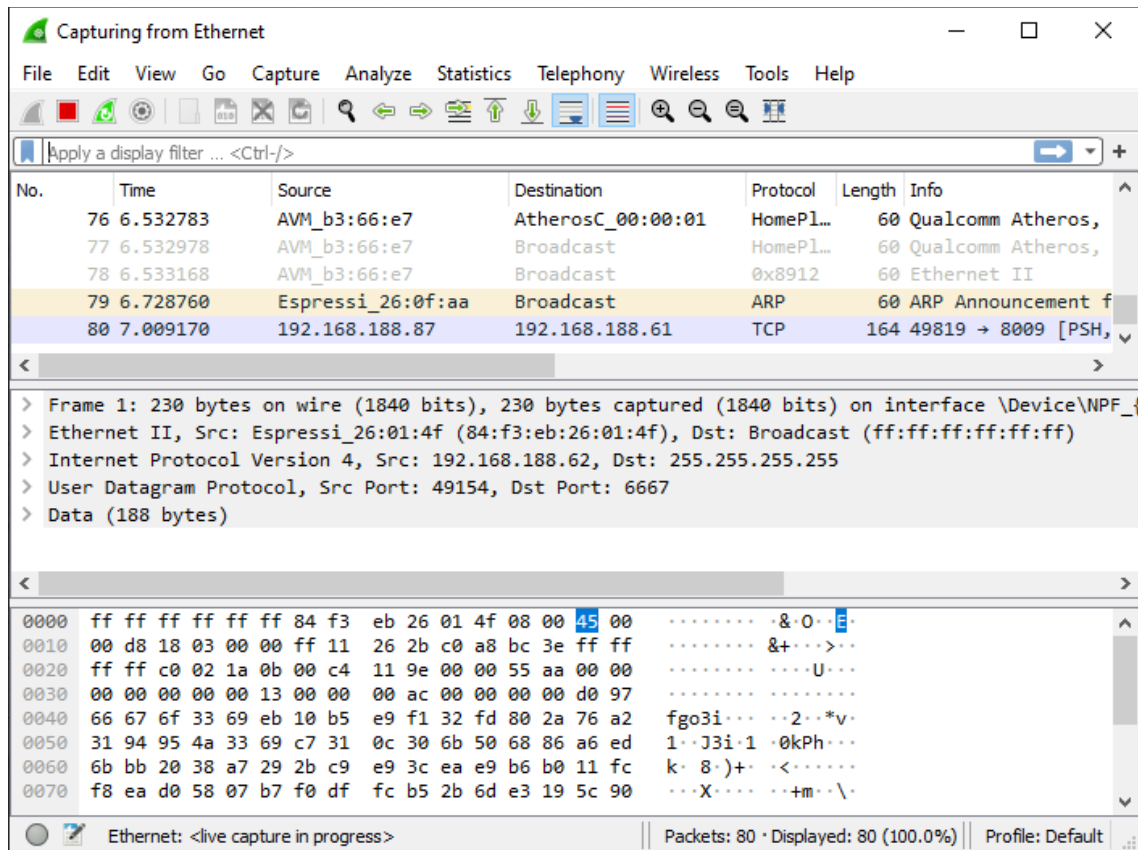


Figure 37. View of Wireshark in capturing mode

Often large amounts of network traffic are present. To focus on a specific address or packet type it is useful to apply a filter by selecting the text field (highlighted in Figure 38).

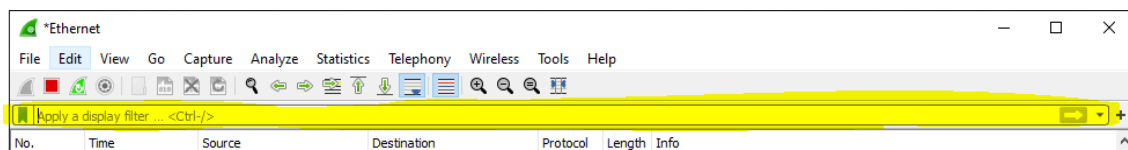


Figure 38. View of text field for filter expression

The syntax to apply a filter for a specific network address is `ip.addr == <IP addr>`. For this example, `ip.addr == 192.168.188.255` is entered, which is the multicast address being used for the DIS simulation in progress. After applying the filter

by pressing {Enter}, Wireshark displays all packets with 192.168.188.255 either as source or destination address (Figure 39).

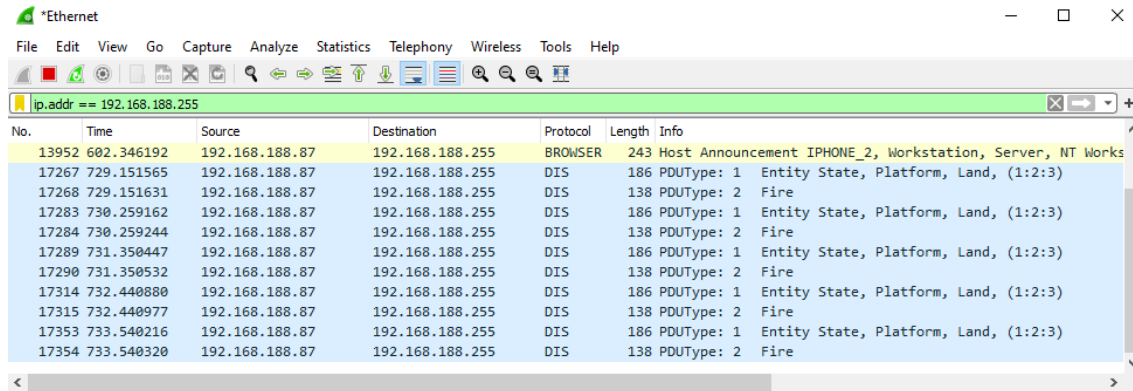


Figure 39. View of an applied filter for a single IP address

To apply an additional filter for a specific protocol, the name of the protocol must be typed in the text field shown in Figure 38. To filter DIS packets, it is enough to type in “dis” and {Enter}. All protocols must be typed in lower case. Thus the example combined filter is `ip.addr == 192.168.188.255 && dis`.

Since Wireshark parses all DIS PDUs, information within a DIS packet gets displayed in a human-readable form. To analyze a single packet’s information, it must be selected to show the details view. When all nodes are expanded, packet information is displayed as shown in Figure 40.

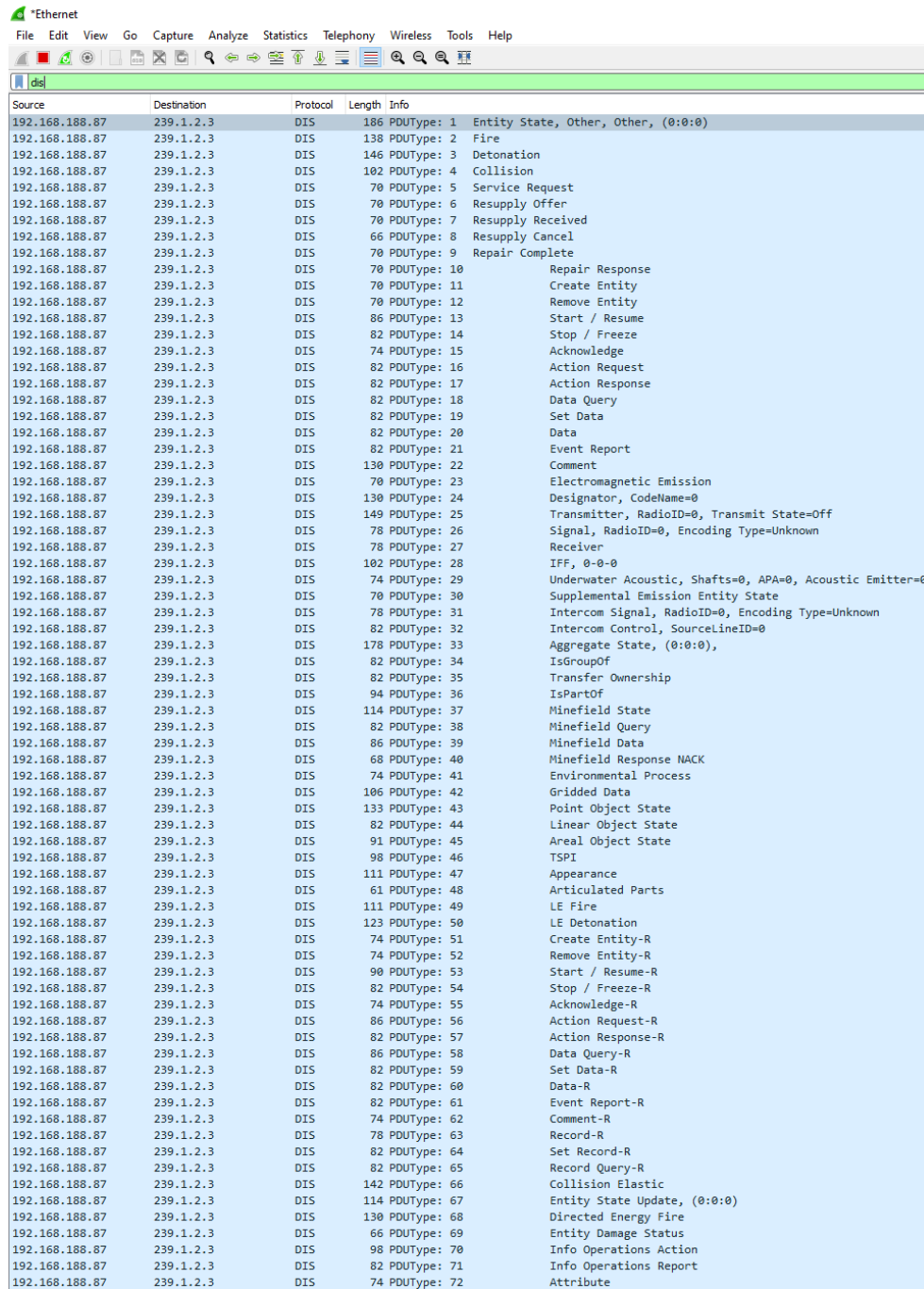
```

> Frame 17267: 186 bytes on wire (1488 bits), 186 bytes captured (1488 bits) on interface
> Ethernet II, Src: Micro-St_2c:0d:15 (00:d8:61:2c:0d:15), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
> Internet Protocol Version 4, Src: 192.168.188.87, Dst: 192.168.188.255
> User Datagram Protocol, Src Port: 3000, Dst Port: 3000
▼ Distributed Interactive Simulation
  > Header
  ▼ Entity State PDU
    ▼ Entity ID
      Entity ID Site: 1
      Entity ID Application: 2
      Entity ID Entity: 3
      Force ID: 0
      Number of Articulation Parameters: 0
    ▼ Entity Type, (1:1:225:1:1:3:0)
      Kind: Platform (1)
      Domain: Land (1)
      Country: United States (225)
      Category / Land: Tank (1)
      Subcategory: 1
      Specific: 3
      Extra: 0
    ▼ Alternative Entity Type, (0:0:0:0:0:0:0)
      Kind: Other (0)
      Domain: Other (0)
      Country: Other (0)
      Category: 0
      Subcategory: 0
      Specific: 0
      Extra: 0
    ▼ Entity Linear Velocity
      X: 0
      Y: 0
      Z: 0
    ▼ Entity Location
      X: -2707488,36777687
      Y: -4353666,73524438
      Z: 3781450,32027544
    ▼ Entity Orientation
      Psi: 0
      Theta: 0
      Phi: 0
      Appearance: 0x00000000
    ▼ Dead Reckoning Parameters
      Dead Reckoning Algorithm: Other (0)
      Dead Reckoning Other Parameters: 00000000000000000000000000000000
    ▼ Entity Linear Acceleration
      Entity Linear Acceleration X: 0
      Entity Linear Acceleration Y: 0
      Entity Linear Acceleration Z: 0
    ▼ Entity Angular Velocity
      Entity Angular Velocity X: 0
      Entity Angular Velocity Y: 0
      Entity Angular Velocity Z: 0
    ▼ Entity Marking
      Entity Character Set: Unused (0)
      Capabilities: 0

```

Figure 40. Details of a single DIS PDU packet parsed by Wireshark

Using AllPduSender.java to send all 72 DIS PDUs to multicast address 239.1.2.3 leads to a capture of all 72 PDUs in Wireshark (Figure 41). Screen capture information can also be copied and saved as a plain-text data file for further processing and analysis.



Source	Destination	Protocol	Length	Info
192.168.188.87	239.1.2.3	DIS	186	PDUType: 1 Entity State, Other, Other, (0:0:0)
192.168.188.87	239.1.2.3	DIS	138	PDUType: 2 Fire
192.168.188.87	239.1.2.3	DIS	146	PDUType: 3 Detonation
192.168.188.87	239.1.2.3	DIS	102	PDUType: 4 Collision
192.168.188.87	239.1.2.3	DIS	70	PDUType: 5 Service Request
192.168.188.87	239.1.2.3	DIS	70	PDUType: 6 Resupply Offer
192.168.188.87	239.1.2.3	DIS	70	PDUType: 7 Resupply Received
192.168.188.87	239.1.2.3	DIS	66	PDUType: 8 Resupply Cancel
192.168.188.87	239.1.2.3	DIS	70	PDUType: 9 Repair Complete
192.168.188.87	239.1.2.3	DIS	70	PDUType: 10 Repair Response
192.168.188.87	239.1.2.3	DIS	70	PDUType: 11 Create Entity
192.168.188.87	239.1.2.3	DIS	70	PDUType: 12 Remove Entity
192.168.188.87	239.1.2.3	DIS	86	PDUType: 13 Start / Resume
192.168.188.87	239.1.2.3	DIS	82	PDUType: 14 Stop / Freeze
192.168.188.87	239.1.2.3	DIS	74	PDUType: 15 Acknowledge
192.168.188.87	239.1.2.3	DIS	82	PDUType: 16 Action Request
192.168.188.87	239.1.2.3	DIS	82	PDUType: 17 Action Response
192.168.188.87	239.1.2.3	DIS	82	PDUType: 18 Data Query
192.168.188.87	239.1.2.3	DIS	82	PDUType: 19 Set Data
192.168.188.87	239.1.2.3	DIS	82	PDUType: 20 Data
192.168.188.87	239.1.2.3	DIS	82	PDUType: 21 Event Report
192.168.188.87	239.1.2.3	DIS	130	PDUType: 22 Comment
192.168.188.87	239.1.2.3	DIS	70	PDUType: 23 Electromagnetic Emission
192.168.188.87	239.1.2.3	DIS	130	PDUType: 24 Designator, CodeName=0
192.168.188.87	239.1.2.3	DIS	149	PDUType: 25 Transmitter, RadioID=0, Transmit State=Off
192.168.188.87	239.1.2.3	DIS	78	PDUType: 26 Signal, RadioID=0, Encoding Type=Unknown
192.168.188.87	239.1.2.3	DIS	78	PDUType: 27 Receiver
192.168.188.87	239.1.2.3	DIS	102	PDUType: 28 IFF, 0-0-0
192.168.188.87	239.1.2.3	DIS	74	PDUType: 29 Underwater Acoustic, Shafts=0, APA=0, Acoustic Emitter=0
192.168.188.87	239.1.2.3	DIS	70	PDUType: 30 Supplemental Emission Entity State
192.168.188.87	239.1.2.3	DIS	78	PDUType: 31 Intercom Signal, RadioID=0, Encoding Type=Unknown
192.168.188.87	239.1.2.3	DIS	82	PDUType: 32 Intercom Control, SourceLineID=0
192.168.188.87	239.1.2.3	DIS	178	PDUType: 33 Aggregate State, (0:0:0),
192.168.188.87	239.1.2.3	DIS	82	PDUType: 34 IsGroupOf
192.168.188.87	239.1.2.3	DIS	82	PDUType: 35 Transfer Ownership
192.168.188.87	239.1.2.3	DIS	94	PDUType: 36 IsPartOf
192.168.188.87	239.1.2.3	DIS	114	PDUType: 37 Minefield State
192.168.188.87	239.1.2.3	DIS	82	PDUType: 38 Minefield Query
192.168.188.87	239.1.2.3	DIS	86	PDUType: 39 Minefield Data
192.168.188.87	239.1.2.3	DIS	68	PDUType: 40 Minefield Response NACK
192.168.188.87	239.1.2.3	DIS	74	PDUType: 41 Environmental Process
192.168.188.87	239.1.2.3	DIS	106	PDUType: 42 Gridded Data
192.168.188.87	239.1.2.3	DIS	133	PDUType: 43 Point Object State
192.168.188.87	239.1.2.3	DIS	82	PDUType: 44 Linear Object State
192.168.188.87	239.1.2.3	DIS	91	PDUType: 45 Areal Object State
192.168.188.87	239.1.2.3	DIS	98	PDUType: 46 TSPI
192.168.188.87	239.1.2.3	DIS	111	PDUType: 47 Appearance
192.168.188.87	239.1.2.3	DIS	61	PDUType: 48 Articulated Parts
192.168.188.87	239.1.2.3	DIS	111	PDUType: 49 LE Fire
192.168.188.87	239.1.2.3	DIS	123	PDUType: 50 LE Detonation
192.168.188.87	239.1.2.3	DIS	74	PDUType: 51 Create Entity-R
192.168.188.87	239.1.2.3	DIS	74	PDUType: 52 Remove Entity-R
192.168.188.87	239.1.2.3	DIS	90	PDUType: 53 Start / Resume-R
192.168.188.87	239.1.2.3	DIS	82	PDUType: 54 Stop / Freeze-R
192.168.188.87	239.1.2.3	DIS	74	PDUType: 55 Acknowledge-R
192.168.188.87	239.1.2.3	DIS	86	PDUType: 56 Action Request-R
192.168.188.87	239.1.2.3	DIS	82	PDUType: 57 Action Response-R
192.168.188.87	239.1.2.3	DIS	86	PDUType: 58 Data Query-R
192.168.188.87	239.1.2.3	DIS	82	PDUType: 59 Set Data-R
192.168.188.87	239.1.2.3	DIS	82	PDUType: 60 Data-R
192.168.188.87	239.1.2.3	DIS	82	PDUType: 61 Event Report-R
192.168.188.87	239.1.2.3	DIS	74	PDUType: 62 Comment-R
192.168.188.87	239.1.2.3	DIS	78	PDUType: 63 Record-R
192.168.188.87	239.1.2.3	DIS	82	PDUType: 64 Set Record-R
192.168.188.87	239.1.2.3	DIS	82	PDUType: 65 Record Query-R
192.168.188.87	239.1.2.3	DIS	142	PDUType: 66 Collision Elastic
192.168.188.87	239.1.2.3	DIS	114	PDUType: 67 Entity State Update, (0:0:0)
192.168.188.87	239.1.2.3	DIS	130	PDUType: 68 Directed Energy Fire
192.168.188.87	239.1.2.3	DIS	66	PDUType: 69 Entity Damage Status
192.168.188.87	239.1.2.3	DIS	98	PDUType: 70 Info Operations Action
192.168.188.87	239.1.2.3	DIS	82	PDUType: 71 Info Operations Report
192.168.188.87	239.1.2.3	DIS	74	PDUType: 72 Attribute

Figure 41. All 72 IEEE DIS PDU types captured with Wireshark

C. RECORDING DIS PACKETS

DIS Packets can be recorded and saved with different encodings without loss of relevant information. The intended use of the files (replay convenience, analytic tool, file sharing, network bandwidth, etc.) drives the decision on which encoding is to be used. Figure 42 illustrates the process from a PDU stream being recorded, saved to a file, read from a file, populated to PDUs, and sent out, utilizing any kind of network to finally become a PDU stream again. The architecture of this approach is modular, so many new encodings can be added just by adding a Recorder/Player software pair of classes that handle the specific encoding.

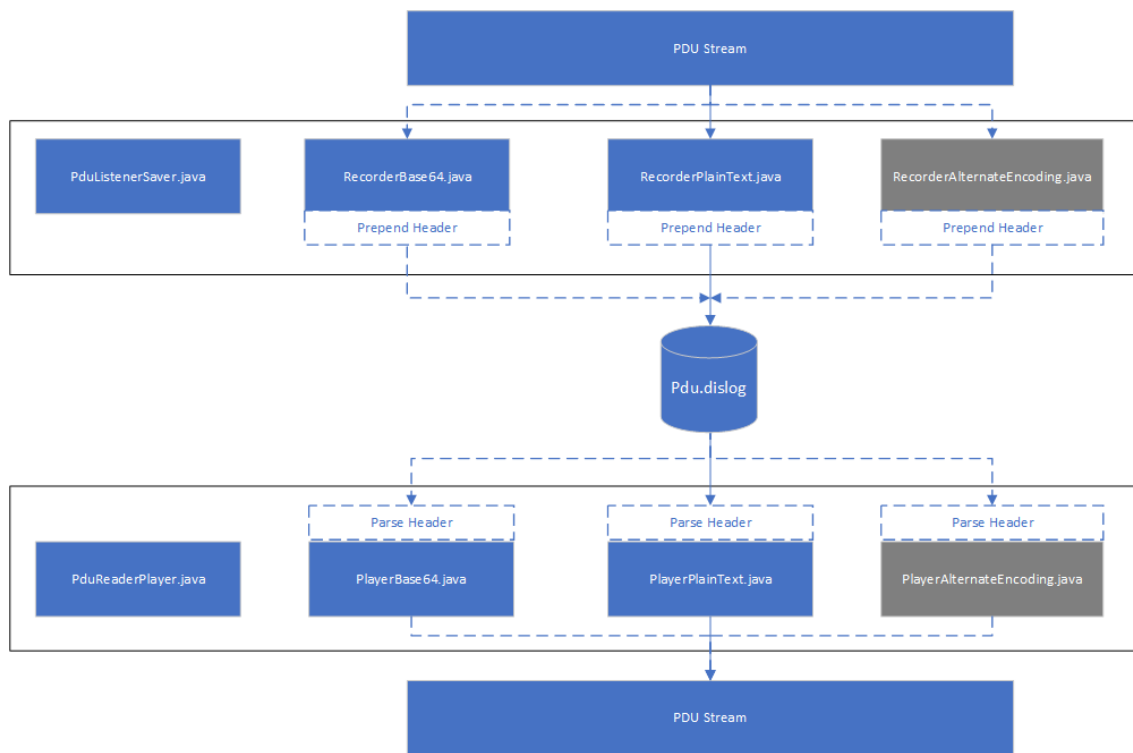


Figure 42. Recording and play back of PDU streams can occur using different encodings. Future encoding pairs are straightforward and testable.

1. Saving Recorded PDUs to Base64 Encoding

The DIS 7 distribution contains classes for recording and playback of DIS packets. They are named Recorder.java and Player.java. Base64 is the name for a common text-based encoding that uses 64 distinct alphanumeric characters as a means of representing 6 bits of information ($2^6 = 64$) in an 7- or 8-bit ASCII character (Josefsson, 2006). This technique is commonly used for binary attachments in text-based email messages by the Multipurpose Internet Mail Extensions (MIME) Standard (Borenstein & Freed, n.d.). In the DIS 7 distribution, two examples are provided showing how to record and save DIS packets into a file. PduListenerSaver.java inherits from instantiates a recorder from Recorder.java and saves all PDUs being received into a file encoded with BASE64 (Figure 43).

```
AAAAABnRSPQ=,BwAFAwAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA==
AAAAAB/m8kg=,BwAGAwAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA==
AAAAACXtWVw=,BwAHAwAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA==
AAAAACvzmWA=,BwAIAwAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAADH7KVQ=,BwAJAwAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA==
AAAAADgffug=,BwAKAwAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA==
AAAAAD5FCqQ=,BwALBQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA==
AAAAAERbrfg=,BwAMBQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA==
```

Figure 43. PDU types 5 – 12 from AllPduSender.java in BASE64 encoding text characters offering some compression but not plain-text readability

2. Saving Recorded PDUs to a Parsable Plain-Text File

For debugging purposes, the BASE64 format is not ideal since it is compressed and not human readable. For the intent of this thesis, it is important that a user be able to inspect recorded files by hand and search for content. Therefore, modified Player.java and Recorder.java classes were developed for “plaintext” readable usage (such as Big Data analysis). The original Base64 programs written by Mike Bailey have been renamed to RecorderBase64.java and PlayerBase64.java. The newly developed human readable plain-text classes are named RecorderPlainText.java and PlayerPlainText.java. The file

PduListenerSaver.java was also modified to either let the programmer choose the RecorderBase64.java or RecorderPlainText.java class as designed for processing a given PDU log file.

As displayed in Figure 44, it is now possible to inspect PDU packets when they are recorded in plain text. PDU Header and PDU Content are clearly separated from each other: [PDU Header], [PDU Content]. Interestingly an existing Java class was utilized to accomplish this input/output format:

[0,0,0,0,25,127,120,8],[7,0,5,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
[0,0,0,0,31,-121,-102,20],[7,0,6,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
[0,0,0,0,37,-99,101,0],[7,0,7,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
[0,0,0,0,43,-90,-122,-24],[7,0,8,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
[0,0,0,0,49,-83,-29,-80],[7,0,9,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
[0,0,0,0,55,-78,-37,-8],[7,0,10,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
[0,0,0,0,61,-34,-4,120],[7,0,11,5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

Figure 44. PDU types 5 – 12 from AllPduSender.java in plain text makes the values of each received PDU human readable

D. PLAYING BACK DIS PACKETS

PduReaderPlayer.java instantiates a player from player.java opens a file containing a PDU stream, and then sends all packets back to a specific broadcast address. Like PduListenerSaver.java, the PduReaderPlayer.java class was modified to instantiate either the PlayerBase64.java file encoding by Mike Bailey or the PlayerPlainText.java file encoding written for this thesis.

In order to independently test recorded files, Wireshark again proves useful. Wireshark can decode received PDUs and make them easy to compare by displaying them in a table. In Figure 45 Wireshark is used to record the playback of the PDUs from Figure 43 and Figure 44. It is obvious that the first eight lines are identical to the last eight lines. A comparison by hand would have taken much longer (if not forever) and is not practical.

Source	Destination	Protocol	Length	Info
192.168.188.87	239.1.2.3	DIS	70	PDUType: 5 Service Request
192.168.188.87	239.1.2.3	DIS	70	PDUType: 6 Resupply Offer
192.168.188.87	239.1.2.3	DIS	70	PDUType: 7 Resupply Received
192.168.188.87	239.1.2.3	DIS	66	PDUType: 8 Resupply Cancel
192.168.188.87	239.1.2.3	DIS	70	PDUType: 9 Repair Complete
192.168.188.87	239.1.2.3	DIS	70	PDUType: 10 Repair Response
192.168.188.87	239.1.2.3	DIS	70	PDUType: 11 Create Entity
192.168.188.87	239.1.2.3	DIS	70	PDUType: 12 Remove Entity
192.168.188.87	239.1.2.3	DIS	70	PDUType: 5 Service Request
192.168.188.87	239.1.2.3	DIS	70	PDUType: 6 Resupply Offer
192.168.188.87	239.1.2.3	DIS	70	PDUType: 7 Resupply Received
192.168.188.87	239.1.2.3	DIS	66	PDUType: 8 Resupply Cancel
192.168.188.87	239.1.2.3	DIS	70	PDUType: 9 Repair Complete
192.168.188.87	239.1.2.3	DIS	70	PDUType: 10 Repair Response
192.168.188.87	239.1.2.3	DIS	70	PDUType: 11 Create Entity
192.168.188.87	239.1.2.3	DIS	70	PDUType: 12 Remove Entity

Figure 45. Wireshark snapshot of PDU types 5 – 12 from Base64 encoded and unencoded log file

E. RECORDING DATA FROM SIMULATIONS

Recording and playing back synthetically created PDUs is good for testing purposes but not useful when it comes to working with simulations, which may generate immense numbers of packets.

DIS simulation data used for this thesis was generated using MAK VR Forces. Therefore, it is important to know how to record data from simulations. When VR Forces starts, a screen for the configuration of the simulation connection is displayed. In this screen the DIS version, the port and the multicast address can be entered to ensure connectivity to corresponding Recorder.java (Figure 46).

Before recording PDUs it is important to check lines 21 and 22 in the corresponding Recorder.java:

```
21    private final static String MCAST_ADDR = "239.1.2.3";
```



```
22     private final static int DIS_PORT = 3000;
```

The multicast address and port number from Figure 46 goes to the appropriate constants. Improved handling of these parameters is expected as part of upcoming OpenDIS7 library improvements.

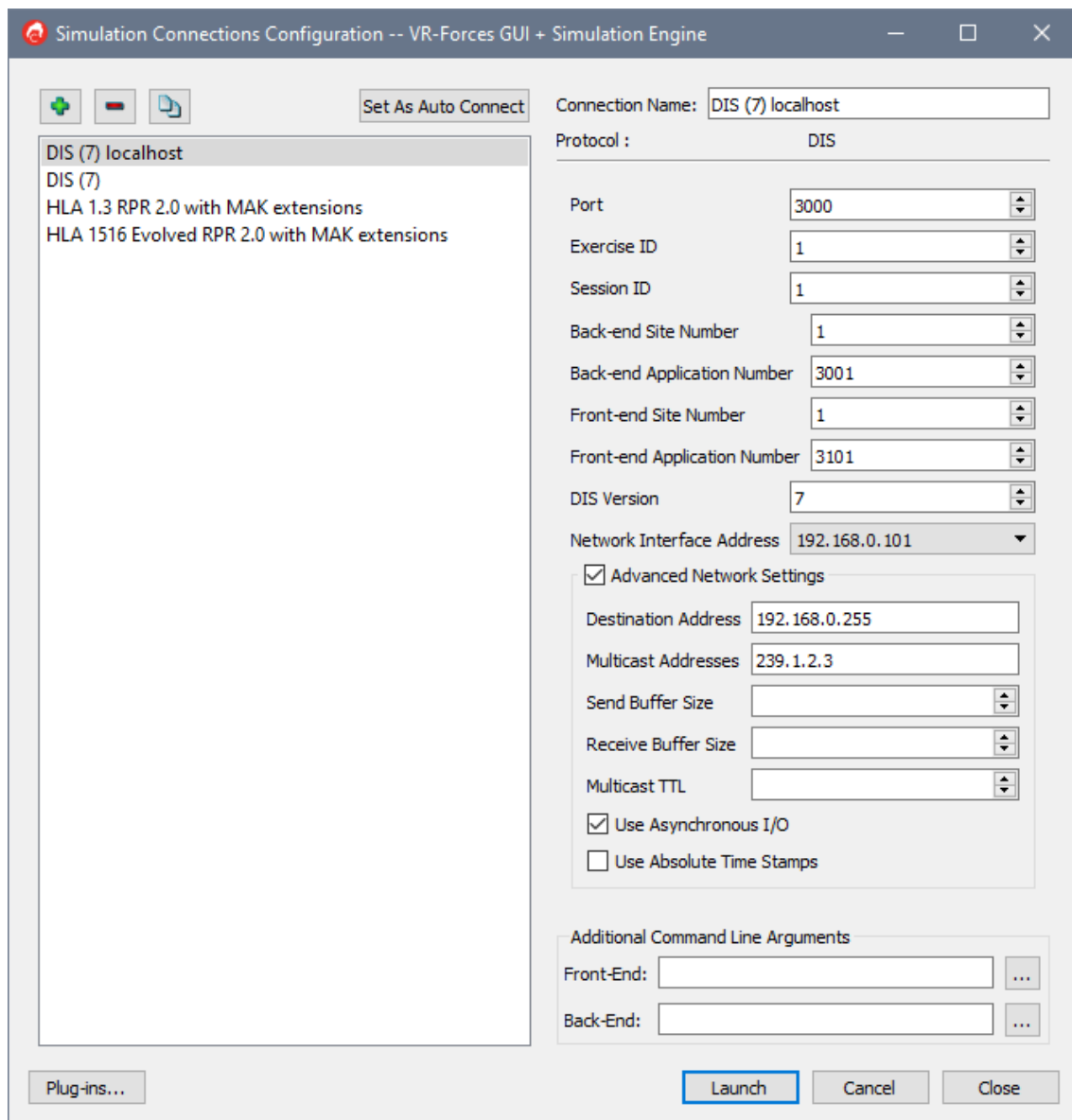


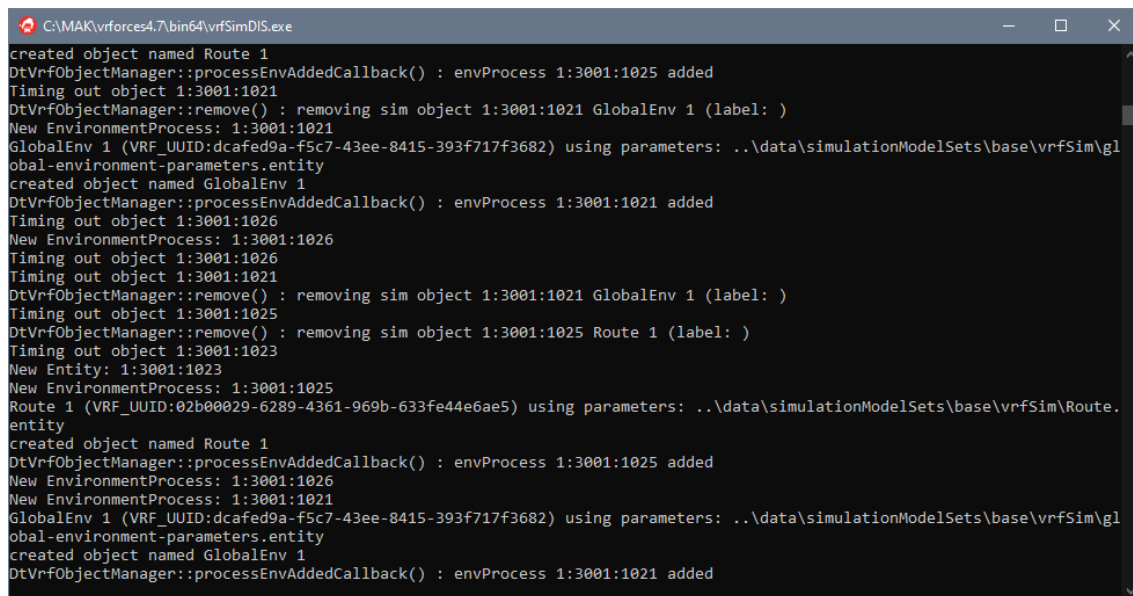
Figure 46. VR Forces Application Simulation Connection Configuration

Once the simulation is running and a scenario ready to start the main class `PduListenerSaver.java` can be started. When the `PduListenerSaver.java` is ready to record, the command console looks like Figure 47.

```
run-single:
DisExamplesOpenDis7.PduListenerSaver started...
Beginning pdu save to directory pduLog
* recorder.startResume(), state=RUNNING, recording in progress...
Warning: you must quit when complete, otherwise recorded PDUs are lost!
Type p/enter to pause, r/enter to resume, q/enter to stop recording, save and quit
```

Figure 47. `PduListenerSaver.java` is ready to record PDUs

VR Forces has a corresponding background window called `vrfSimDIS.exe` (Figure 48) which displays DIS activity. PDUs are decoded and displayed in a way that is easy to understand and helpful when it comes to playing back recorded streams to VR Forces.



```
C:\MAK\vrforces4.7\bin64\vrfSimDIS.exe
created object named Route 1
DtVrfObjectManager::processEnvAddedCallback() : envProcess 1:3001:1025 added
Timing out object 1:3001:1021
DtVrfObjectManager::remove() : removing sim object 1:3001:1021 GlobalEnv 1 (label: )
New EnvironmentProcess: 1:3001:1021
GlobalEnv 1 (VRF_UUID:dcafed9a-f5c7-43ee-8415-393f717f3682) using parameters: ..\data\simulationModelSets\base\vrfSim\gl
obal-environment-parameters.entity
created object named GlobalEnv 1
DtVrfObjectManager::processEnvAddedCallback() : envProcess 1:3001:1021 added
Timing out object 1:3001:1026
New EnvironmentProcess: 1:3001:1026
Timing out object 1:3001:1026
Timing out object 1:3001:1021
DtVrfObjectManager::remove() : removing sim object 1:3001:1021 GlobalEnv 1 (label: )
Timing out object 1:3001:1025
DtVrfObjectManager::remove() : removing sim object 1:3001:1025 Route 1 (label: )
Timing out object 1:3001:1023
New Entity: 1:3001:1023
New EnvironmentProcess: 1:3001:1025
Route 1 (VRF_UUID:02b0029-6289-4361-969b-633fe44e6ae5) using parameters: ..\data\simulationModelSets\base\vrfSim\Route.
entity
created object named Route 1
DtVrfObjectManager::processEnvAddedCallback() : envProcess 1:3001:1025 added
New EnvironmentProcess: 1:3001:1026
New EnvironmentProcess: 1:3001:1021
GlobalEnv 1 (VRF_UUID:dcafed9a-f5c7-43ee-8415-393f717f3682) using parameters: ..\data\simulationModelSets\base\vrfSim\gl
obal-environment-parameters.entity
created object named GlobalEnv 1
DtVrfObjectManager::processEnvAddedCallback() : envProcess 1:3001:1021 added
```

Figure 48. VR Forces window with status of DIS entities

In Figure 49 a basic scenario with one hostile entity and one track setup in VR Forces is displayed. After starting the scenario, the entity follows the track and stop at the end of the track.

Recording the PDUs sent over the network leads to a Pdusave.dislog file stored in the pdulog subfolder of the currently used Recorder.java. To see whether DIS files are travelling over a specific network and reaching the computer with the recorder running, Wireshark with a properly set DIS filter is useful but not necessary.

In the current implementation it is important to stop the Recorder.java by pressing {q} on the keyboard. If the recoding process is not shut down properly, the Pdusave.dislog is not closed and therefore, there is no end comment marker written to it. Without an end comment marker the file is not playable by the Player.java. If this happens an end comment marker can be copied from another saved logfile and appended to the file that was not closed properly.

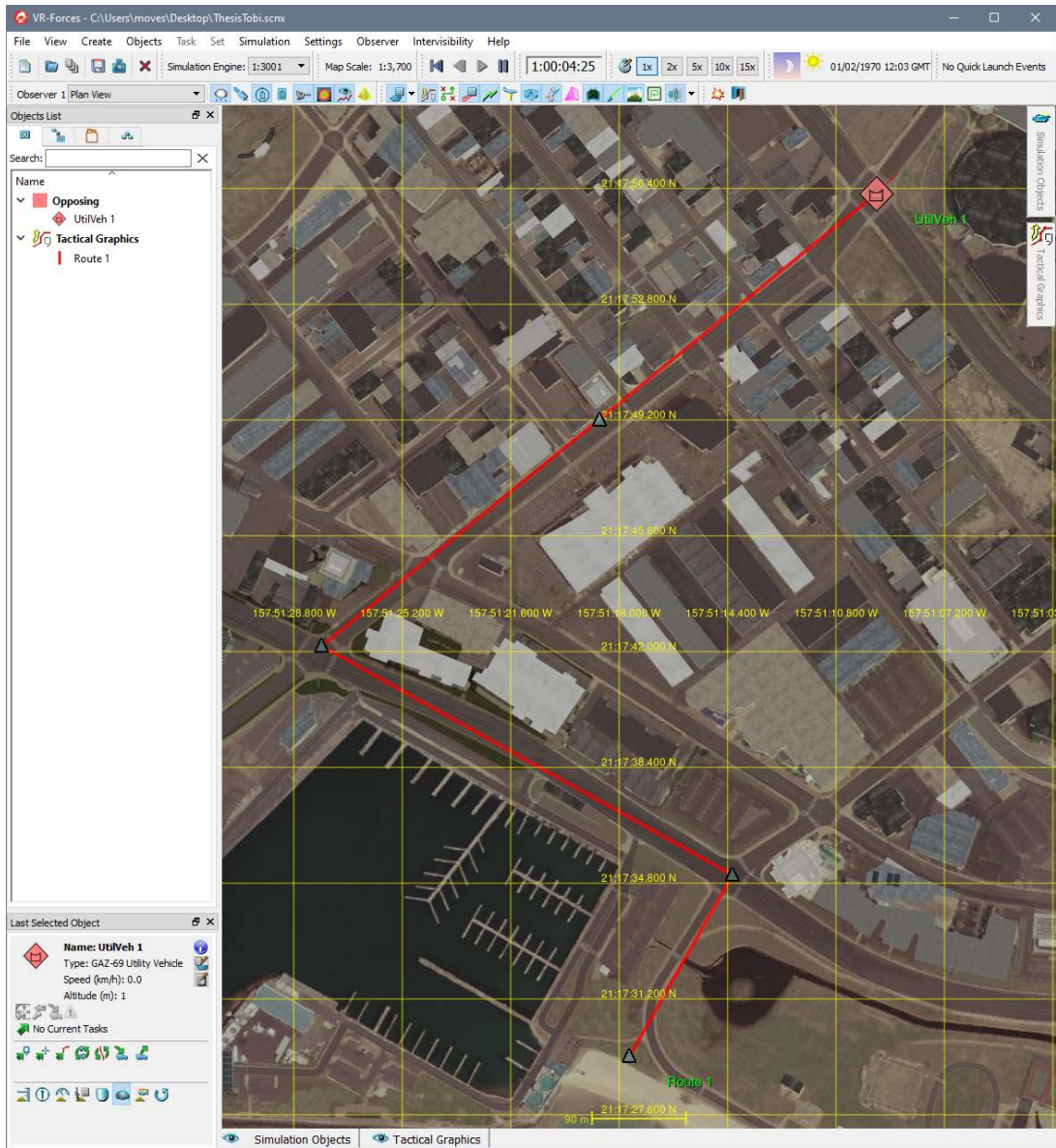


Figure 49. VR Forces screen with one track and one hostile entity

F. PLAYING BACK RECORDED DIS DATA TO SIMULATIONS

Data recorded in the previous steps can be played back to a simulation using `PduReaderPlayer.java`. The easiest setup to test this is to playback the entities to the simulation they came from. Therefore, the scenario on VR Forces must be cleaned from entities and tracks. Before starting the playback, it is important to set VR Forces into run mode by pressing the Play button. Once the simulation is in run mode it is moving its entities

and listening for DIS packets. As there are no entities there is nothing to move until the playback is started.

To start the playback run `PduReaderPlayer.java`. It playbacks all *.dislog files in the `pdulog` subfolder. If only one file shall be played back it is important to move all other files to another folder or delete them.

The PDUs being played back over the ethernet shall lead to some output in the DIS window of VR Forces and populate our entity we set up in our example scenario moving the track we created.

If PDUs are being sent the entity is moving according to the location and heading sent in the PDUs. For testing purposes and to see the dead reckoning at work, the network cable can be disconnected at any time. The entity continues moving with latest speed and heading send by an PDU. If no update is sent for more than five seconds (the specified “keep alive” period), the entity disappears from the map.

If the network cable is disconnected right before a turn, the entity keeps moving straight for a maximum of 5 seconds, missing the turn. A PDU is unable to transmit future data, for example a left turn in 3 seconds. Therefore, the entity continues moving until either an update is received, or the entity disappears from the map. If the cable is plugged back in and the network connection is restored before the 5 seconds time frame ends, the entity is merged back to the heading and speed send by the latest PDU. According to default dead reckoning parameters it is not doing a magic move to the new position.

G. TRANSFORM A PDU STREAM INTO X3D POSITIONINTERPOLATOR

In the MOVES LVC lab there are many simulations sending PDUs to inform other simulations about the status of entities. One of the simulations is VR Forces. VR Forces is used for many tests because it is easy to have an entity set up, moving a defined and repeatable route of travel. PDUs can be recorded and saved to a file. Another use of a recorded stream is to extract entities and convert them into X3D entities. X3D entities have a higher fidelity to the real world when they move within the scene driven by a `PositionInterpolator` accompanied by a matching `OrientationInterpolator`, driven together by

a shared TimeSensor node. The algorithm for the conversion from the stream of a DIS application into an animated X3D entity driven by autogenerated code is shown in Figure 50.

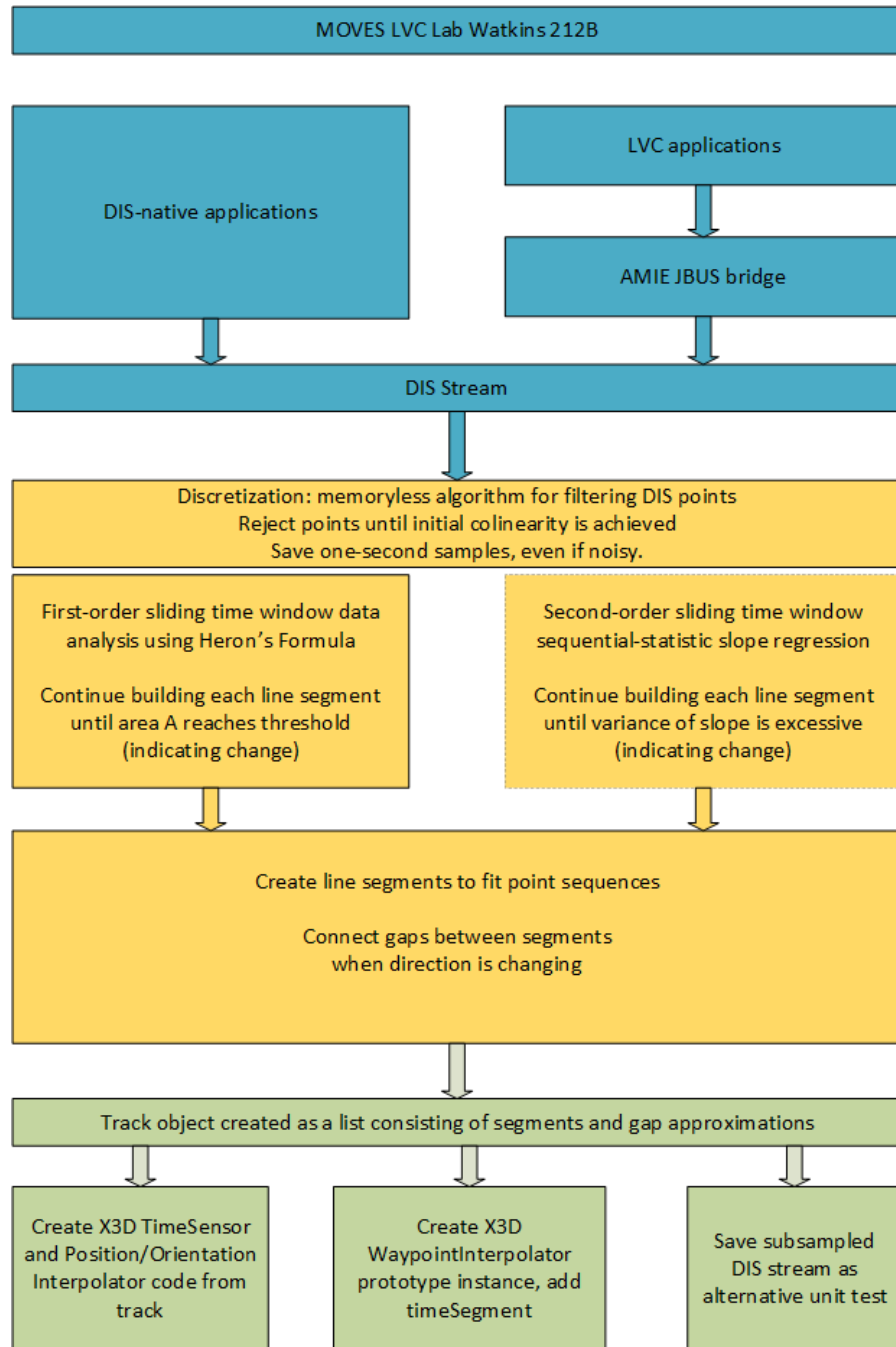


Figure 50. Algorithm for creating X3D tracks through PDU filtering

A recorded DIS stream can be used to drive a TimeSensor (Figure 51) and a PositionInterpolator (Figure 52) in X3D.

```
<TimeSensor DEF = "BoxTimeInterval" cycleInterval='1'
loop='true' />
```

Figure 51. Code excerpt for a X3D TimeSensor shows XML encoding of keyfields in this node

```
<PositionInterpolator DEF='Entity' key='0.0 0.25 0.5 0.75
1.0' keyValue='0 0 0 0 1 0 0 2 0 0 1 0 0 0 0 />
```

Figure 52. Code excerpt for a X3D PositionInterpolator shows XML encoding of keyfields in this node

To achieve this a DIS stream can be recorded using the class PduListenerSaver.java from OpenDis7Examples. This class saves a file with the extension *.dislog. The logfile must be read by the PduReaderPlayer.java originally written by Mike Bailey and modified for this thesis. The class PduReaderPlayer.java reads in the logfile, create ESPDUs and unmarshalls the byte stream from the file to populate the created ESPDUs. The timestamp and location from each ESPDU are read and stored into a hash map. A hash map is chosen because it uses the same connection between a key and a key value like the PositionInterpolator in X3D uses (Figure 52).

The values from the ESPDUs can be obtained with getTimeStamp() and getEntityLocation(). By adding, for example, getX() to getEntityLocation a single variable can be accessed. To start populating the hash map the timestamp of the first ESPDU must be stored to get relative times between ESPDUs. There is no use for absolute simulation times for playbacks. The value of the first timestamp is subtracted from the first and all subsequent ESPDUs. Therefore, the first value is 0 and all other keys have ascending values greater than 0. Timestamps are stored as keys. Locations are stored as belonging keyValues.

Once all values are stored in a hash map, the duration of the entire DIS stream must be derived and put into the TimeSensor attribute cycle Interval. To be in line with the standard for X3D PositionInterpolators, times in the key attribute field must be between zero and one. To normalize times in the hash map all values of the hash map must be divided by value of the latest timestamp. After doing this calculation the first timestamp in the hash map shall still be zero but the last timestamp shall be one.

Keys and keyValues in the hash map are now set up to be written into a TimeSensor and a PositionInterpolator.

TimeSensor and PositionInterpolator are both put together in the same manner. Both have an opening XML tag, one or more attributes, and values for the attributes. XML tags and attributes are hard-coded strings. Values for the attributes are pulled from a hash map, converted from double to strings with three digits, and appended to strings for either the TimeSensor or PositionInterpolator.

After all values are stored in two strings, the XML closing tag is added to both to make them valid XML expressions. They are printed to the console to make them available to be copied into a X3D file.

a. Storing PDUs

Once configured, VR Forces send updates using the IP addresses provided. The PDUListenerSaver.java class from the DIS7 examples records the DIS stream to a plain-text file on another computer. PduListenerSaver.java originally uses the Saver.java by Mike Bailey to store the stream in a BASE64 encoded file. For the purpose of this thesis, the Recorder.java file was modified to get an unencoded, more human-readable file. The source code for the RecorderPlainText Java program can be found in Appendix C.

b. Processing Stored PDUs

With the PDU stream in a text file the processing of the file is straightforward. The file is read in by a Java class called PduReaderPlayer.java originally written by Mike Bailey and modified for this thesis.

The class reads the file line-by-line as strings and populates the strings to PDUs. Once a PDU is populated to an ESPDU using the `es pdu.unmarshal` method, timestamp, coordinates and orientation are accessible the following commands:

- `Espdu.getTimestamp()` provides a float representing the ticks in time elapsed since the top of the current hour,
- `Espdu.getEntityLocation().getX()` provides a float representing the entities X coordinate within the world coordinate system,
- `Espdu.getEntityLocation().getY()` provides a float representing the entities Y coordinate within the world coordinate system,
- `Espdu.getEntityLocation().getZ()` provides a float representing the entities Z coordinate within the world coordinate system,
- `Espdu.getEntityOrientation().getPhi()` provides a float representing the entities Phi angle within the world coordinate system,
- `Espdu.getEntityOrientation().getPsi()` provides a float representing the entities Psi angle within the world coordinate system, and
- `Espdu.getEntityOrientation().getTheta()` provides a float representing the entities Theta angle within the world coordinate system.

There is more information available in the Javadoc for extracting information from PDUs that can be used in future work.

All timestamps, coordinates and angles are stored into a Java `LinkedHashMap` with a double integer as key and a class “Coordinates” as key value. The class “Coordinates” (Figure 53) was written for this thesis and maps x, y, z, phi, psi, and theta into one class, so that it can be handled by the `LinkedHashMap`.

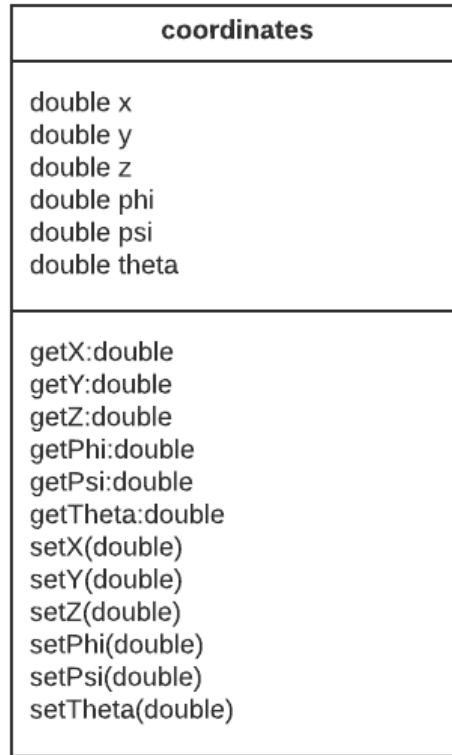


Figure 53. UML-diagram for Coordinates.java

In a simulation an entity moves towards a goal in a generally straight direction. Although heading and speed does not change, the simulation sends out updates as PDUs. For replay purposes, the updates have to conform to the DIS standard of transmitting an update at least every 5 seconds (the “keep alive” or “heartbeat” interval).

To avoid an entity from moving in a jerky manner, it is important to send out updates, once the heading of the entity changes. To achieve this a stream of PDUs must be postprocessed.

c. Physics-based filtering and compression

If we assume that an entity is unable to change direction and speed arbitrarily, we can use a first-order approximation between two waypoints. If a sequence of waypoints describe a straight line, only the first and the last points are needed. Filtering and deleting collinear points between two waypoints compresses the size of the stream.

There is no need to test for certain packets that describe a state of the entity after something important happened in the simulation. E.g., if there is a large explosion that affects many entities, the stream contains ground truth whether the entity is affected or not. A compression of a stream in the sense described in this thesis is always retrospective. If the entity changes heading and speed because of an explosion, the simulation sends out ESPDUs so that the algorithm applied can filter them in the way described before.

d. Distill concise first-order linear interpolators from streams

For postprocessing a sliding window mentioned in (Arasu & Manku, 2004) and shown as a flowchart in Figure 54 is used. Points from the stream are added to a sliding window. Once three points are within the sliding window they can be checked for collinearity. When they are collinear the first and the last points are needed to get a high-fidelity playback. In addition to that the size of the sliding window is tracked, as shown in Figure 54. Once the difference between the first and the last packets in the sliding window exceeds 4 seconds, the first and the last points are added to the track to comply with the DIS “heartbeat” requirement. The process described can be used for creating X3D PositionInterpolators and compressing DIS streams directly. An example of an autogenerated PositionInterpolator can be found in the Appendix C.

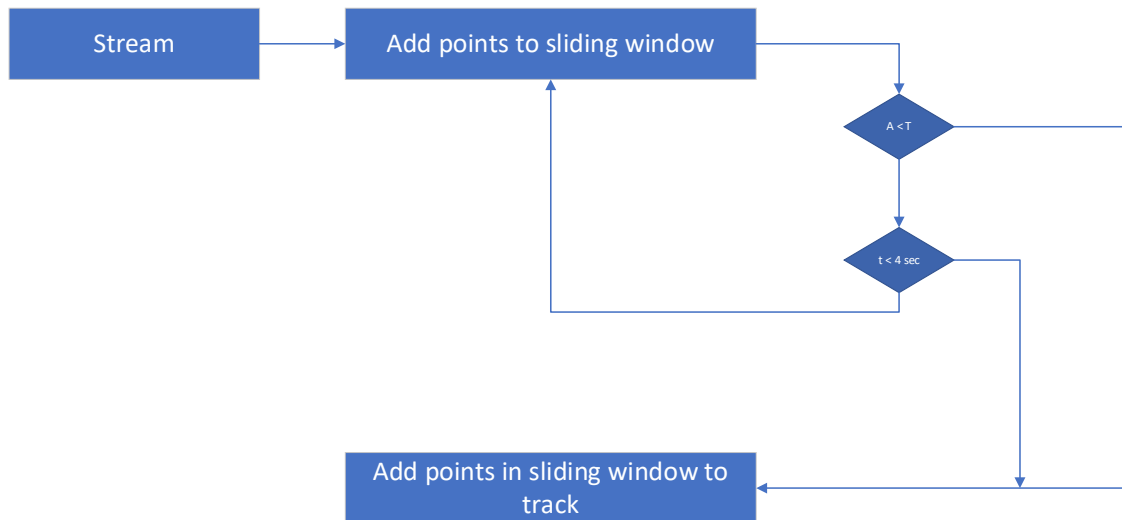


Figure 54. Flowchart of sliding window algorithm to compute area A

Testing for collinearity is performed with Heron's Formula (Figure 55) (Kendig, 2000). A triangle with the area A is constructed between the first two and the last points within the sliding window. Variables a , b , and c for Heron's Formula can be calculated by subtracting the coordinates of two of the three points mentioned before.

Given three points, $P_1(x_1, y_1, z_1)$, $P_2(x_2, y_2, z_2)$ and $P_3(x_3, y_3, z_3)$, the variables a , b , and c can be calculated by the following calculations (AmBrSoft, 2014):

$$a = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

$$b = \sqrt{(x_3 - x_1)^2 + (y_3 - y_1)^2 + (z_3 - z_1)^2}$$

$$c = \sqrt{(x_3 - x_2)^2 + (y_3 - y_2)^2 + (z_3 - z_2)^2}$$

When the area A of the triangle is zero, the three points can be considered as collinear.

$$A = \sqrt{s(s - a) \cdot (s - b) \cdot (s - c)}$$

$$s = \frac{(a + b + c)}{2}$$

Figure 55. Heron's Formula for measuring collinearities. Adapted from AmBrSoft (2014).

To account for outliers a threshold is used. Once the threshold is met, the first and the last points are stored to a new map containing all points necessary to achieve a high-fidelity playback of the original recorded track. The threshold can be adjusted for a smoother playback.

An example of a model animation driven by a PDU stream is provided in Figure 56 and Figure 57. The first figure provides a track of an aircraft flying from one point to another with a slight change of track in the middle of the flight. Twelve PDUs with position and heading updates are sent to inform other simulations about the actual position and heading of the aircraft.

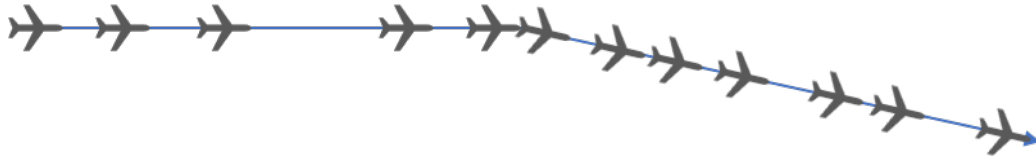


Figure 56. DIS track of an aircraft with 12 PDUs

After applying the sliding window algorithm, four PDUs are left that are needed to represent the track. Between the updates the simulated aircraft keeps flying with speed and orientation of the last provided update or in the case of X3D an approximated direct route between two known positions.

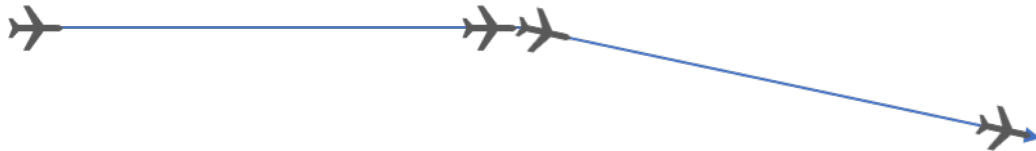


Figure 57. Compressed DIS track of the aircraft from with 4 PDUs

H. CREATE A MATCHING ORIENTATIONINTERPOLATOR

In X3D, entity states can be modified using PositionInterpolator and OrientationInterpolator pairs. The PositionInterpolator alters the entity position. To picture a model of an aircraft without an OrientationInterpolator, imagine a circling aircraft (Figure 58). The aircraft's state is determined by a simulation which transmits PDUs. The remote simulation creates ESPDUs to publish the heading and speed of the aircraft. The interpolators effectively create a distilled "recording" of the entity track that is easily saved within an X3D model, no longer requiring network connectivity or DIS protocol parsing capabilities.

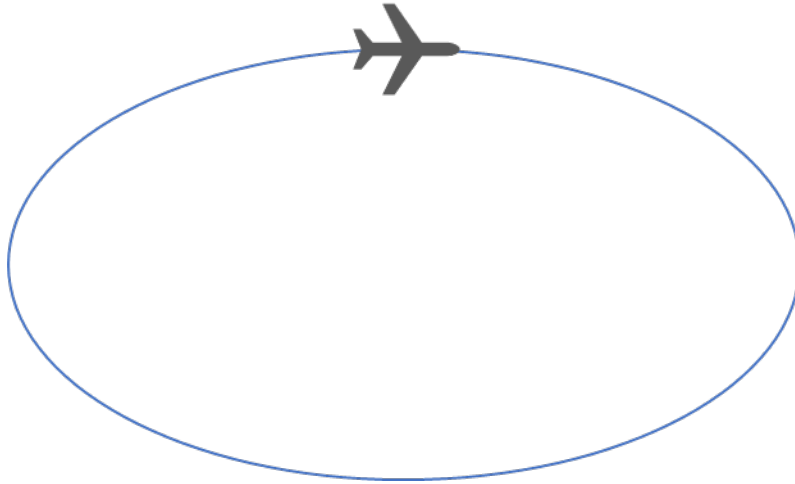


Figure 58. A aircraft flying a circle

Recording ESPDUs to drive the movement of the aircraft in Figure 56 using a PositionInterpolator would result in the replay displayed in Figure 59.

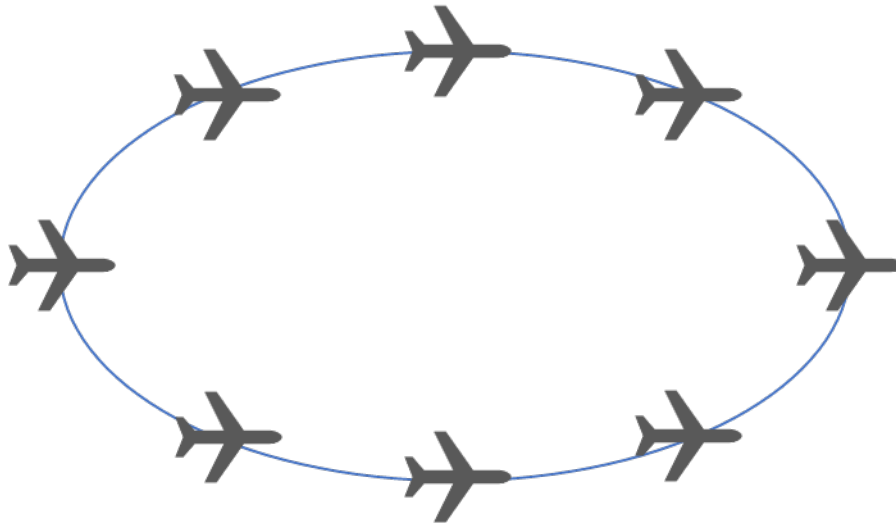


Figure 59. Replay of an aircraft using a solitary PositionInterpolator

The PositionInterpolator alters the position of the entity representing the aircraft but not the heading. A movement like this (aircraft flying backwards) is of course incorrect and also distracting for the user.

Changing the heading of the aircraft results in a rotation of the aircraft. To achieve the rotation in X3D an OrientationInterpolator must be utilized. Using a PositionInterpolator together with an OrientationInterpolator results in replay that is closer to reality and better accepted by the viewer (Figure 60).

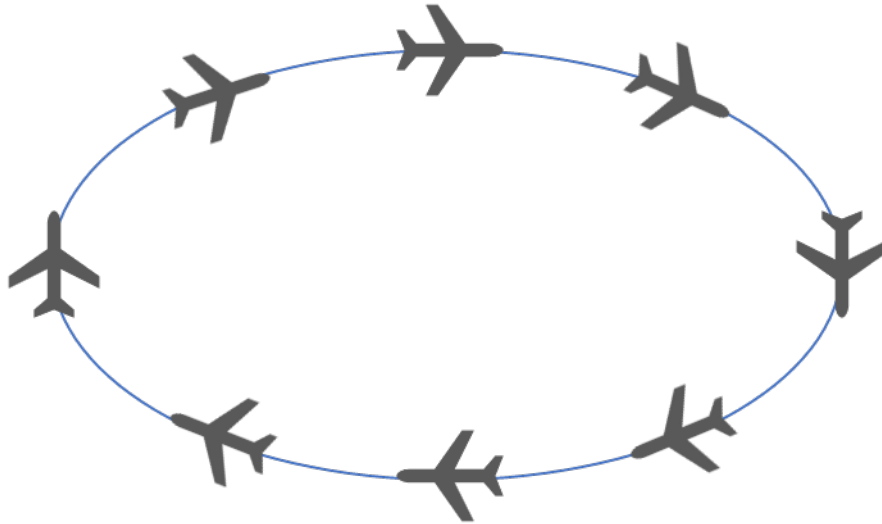


Figure 60. Replay of an aircraft using a PositionInterpolator and OrientationInterpolator

An example of an autogenerated OrientationInterpolator can be found in Appendix C.

I. X3D IMPLEMENTATIONS OF PDUS

All PDUs transmitted by a simulation represent defined entities. For ESPDUs those can be trucks, tanks, cars, bicycles, aircrafts, helicopters, ships, and many more. Simulations like VR Forces provide a 3D model database for drawing the entity at its position in the world coordinate system. Having many different models provides a more diversified picture than having a generic model represent all land-based entities.

When it comes to converting PDU streams to X3D position and OrientationInterpolators, it is also important to represent the entity in X3D in a way that is acceptable to the user. For this thesis, a red rectangular block was initially used to show

changes of speed, orientation, and position. A future topic could be the automation to extract the type of vehicle from an ESPDU, match it with a model from the X3D model exchange, generate code to apply the X3D prototype to the scene, and wiring it up with ROUTE commands using autogenerated code like the code that is used to generate position and orientation interpolators.

Other PDUs like fire or explosions can be handled in a similar way. The X3D Distributed Interactive Simulation Architecture Component defines native mappings for ESPDU, Fire, Detonation, Collision, Radio, Signal and Transmitter PDUs. The addition of comment PDUs is suggested next. For confirming players, X3D can replay recorded DIS streams in a 3D world that is freely explorable by a viewer using arbitrary viewpoints.

J. AUTOMATED ANALYSIS OF PDU STREAMS FOR AFTER-ACTION REVIEW (AAR)

After-action review (AAR) is a common briefing activity for analysts and military applications. To perform after-action analysis of PDU streams, Java-based code can be used to measure anything one can imagine when it comes to LVC. Regularizing DIS to align with best practices for Big Data has great potential value.

For example, the DIS stream of a trainer is recorded during a participant training mission having the goal to reach a certain target, destroy the object, and return to base. The entire mission is recorded as a PDU stream. From that recorded stream, many questions can be answered just by comparing PDUs:

- Was the target reached (distance to target)?
- Was the participant hit by another entity (query of collisions, fires, explosions)?
- Did the participant fire on the target?
- Did the participant hit the target?
- Did the participant return to base without getting destroyed?

- What were corresponding metrics for other participants?
- What was the probability distribution function for other entities?
- How do metric results and distributions compare to other recorded exercises and incidents?

All those questions need nothing more than a query of the PDU stream. It can all be reduced to checking the minimum distance of entities or checking for a collision of an entity with any type of ammunition. AAR analysis code for achieving this can be used (and reused) universally so the current problem with custom written analysis for custom simulations would be overcome. This is an important area for continued development.

K. SUMMARY

This chapter provides an introduction on all tools needed to get a DIS stream driving a X3D model. DIS packets were displayed using Wireshark to make the data that is vital to distributed simulations visible. Recording and playing DIS streams to and from files in different formats was described to get everything set up that is needed to transform a PDU stream into autogenerated Position and OrientationInterpolator. All techniques described in this chapter are key to have DIS runs repeatably available on servers accessible via Web browsers. Representations of PDUs in X3D and automated analysis of PDU streams were introduced to the reader to suggest important areas for future work.

THIS PAGE INTENTIONALLY LEFT BLANK

V. TESTING AND TROUBLESHOOTING

A. INTRODUCTION

This chapter focuses on testing and evaluating connections, PDU streams and entities. It continues to introduce the reader to troubleshooting techniques including the use of Wireshark, OpenDIS7 examples, and some other Java classes to explore the “art of the possible” and path forward for continued progress.

B. CONNECTING TO DIFFERENT TYPES OF NETWORKS

Real-world simulations connect over multiple types of networks and interfaces. To connect for example a live unit at the National Training Center with the LVC lab at NPS several LANs, VLANs, WANs and VPNs must be utilized. All kinds of networks must be configured to send UDP packets when broadcast and multicast are used. In the case of utilizing unicast for distribution, sending and receiving must be configured.

Multicast and broadcast are used when it is important to send PDUs to multiple receivers simultaneously, where senders, receivers and simulations are directly connected to one or more networks. A PDU bridge can connect these networks so that they appear like a single network environment for participants. When sending PDUs across multiple networks where no simulation is running, it is important to use a VPN for classified or private direct unicast connection for unclassified data.

C. TEST AND EVALUATION OF CONNECTIONS

Network connections are the key asset when it comes to distributed interactive simulations. Without network connections simulations cannot interconnect. More important than the presence of a connection is its quality. For most DIS applications bandwidth is no longer an issue because almost all network connections today provide at least 10 Mbit/s. The main issue for DIS is the delay of packets caused by a network. In this context, the network includes all cables, active, and passive components. When LAN connections are used a latency up to 5 ms is normal. For WAN connections between different bases 10–100 ms are acceptable. If VPN and satellite connections are included

latency times can go up to 1000 ms. A latency time of 1000 ms leads to a difference in timestamps of one second between two simulations. For a simulation, on the aggregate level, this is not an issue. For a simulation on the entity level, when two virtual entities (operated by a human) fight each other, 1000 ms in latency time can cause an unsatisfactory user experience.

For this thesis, multiple different network connections were tested:

- Same network connected via LAN (Ethernet),
- Same network connected via WiFi,
- Domestic connection using VPN over WiFi,
- Transatlantic connection using VPN over LAN (Ethernet),
- Transatlantic connection using VPN over WiFi.

All connections were tested using a ThinkPad T470 with Intel Core i7 and 32 GB RAM. For the WiFi tests the network cable was disconnected. For the LAN tests the WiFi was turned off. VPN connections were established using Shrewsoft VPNClient [16] and terminated on a AVM FritzBox 7490 (AVM International, n.d.).

To obtain test statistics, the Windows command prompt together with the freely available standardized network tool *ping.exe* was used. Ping can send a defined number of packets keeping track of the time for one roundtrip. An alternative tool for multiple operating systems is netcat.

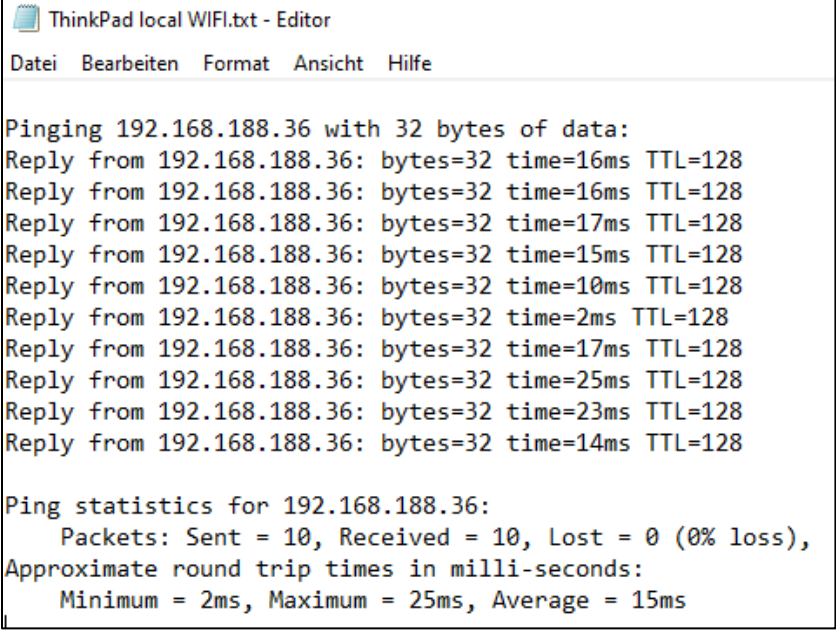
The complete command used for this test can be found in Figure 61.

```
ping 192.168.188.36 -n 100 >>C:\log.txt
```

Figure 61. Windows Command Prompt command for collecting test statistics

The IP address belongs to a VR Forces workstation connected via ethernet to a gigabit switch. The option indicated by “-n” is the switch for setting the number of packets being sent followed by the number. The double right arrow “>>” redirects output from the

output window into the named text file. The execution of this command takes about 100 seconds because one packet is sent per second. An example output for a test with 10 packets can be seen in Figure 62. After receiving 10 packets, statistics on the quality of the network connection are provided.



```
ThinkPad local WIFI.txt - Editor
Datei  Bearbeiten  Format  Ansicht  Hilfe

Pinging 192.168.188.36 with 32 bytes of data:
Reply from 192.168.188.36: bytes=32 time=16ms TTL=128
Reply from 192.168.188.36: bytes=32 time=16ms TTL=128
Reply from 192.168.188.36: bytes=32 time=17ms TTL=128
Reply from 192.168.188.36: bytes=32 time=15ms TTL=128
Reply from 192.168.188.36: bytes=32 time=10ms TTL=128
Reply from 192.168.188.36: bytes=32 time=2ms TTL=128
Reply from 192.168.188.36: bytes=32 time=17ms TTL=128
Reply from 192.168.188.36: bytes=32 time=25ms TTL=128
Reply from 192.168.188.36: bytes=32 time=23ms TTL=128
Reply from 192.168.188.36: bytes=32 time=14ms TTL=128

Ping statistics for 192.168.188.36:
    Packets: Sent = 10, Received = 10, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 2ms, Maximum = 25ms, Average = 15ms
```

Figure 62. Sample log file for 10 packets over local WiFi

The connection in Figure 62 is a WiFi connection on the local network resulting in an average of 15 ms and 0% loss (which is excellent and expected for an uncontested home network). See Table 5 for all tested connections.

Table 5. Average roundtrip times for tested network connections

Connection	Average round trip time (latency)
Same network via LAN	0 ms
Same network via WiFi	15 ms
Domestic WAN using VPN over WIFI	27 ms
Transatlantic WAN using VPN over LAN	173 ms
Transatlantic WAN using VPN over WiFi	251 ms

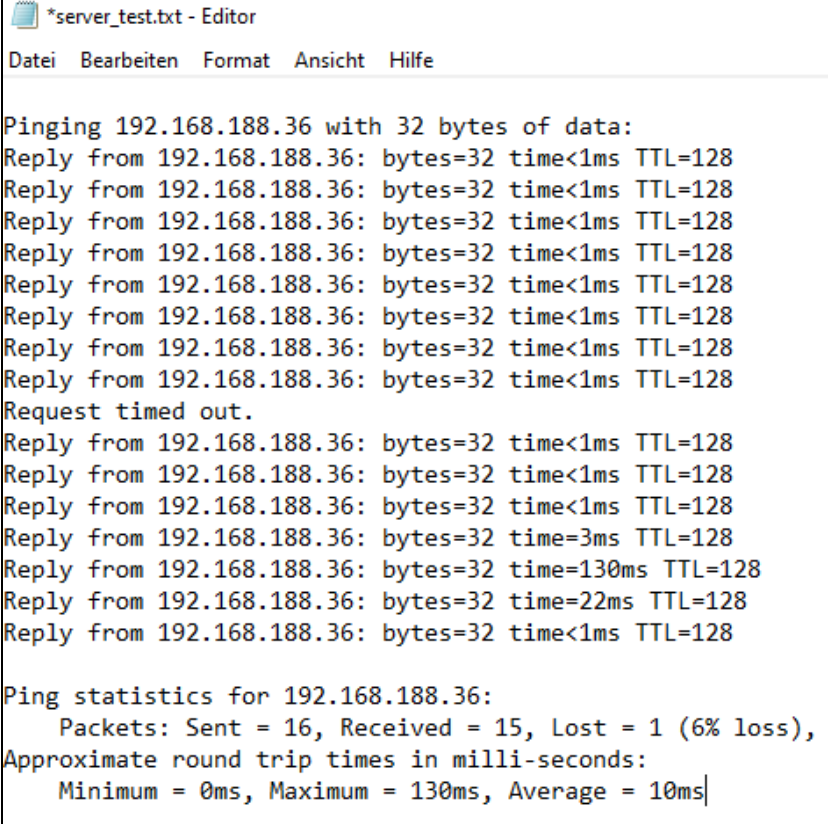
Sometimes network connections are not consistently reliable but when they are tested with the described procedure, they appear reliable. To test a network connection over a longer period, ping can operate in an infinite mode. With the help of the following procedure it is possible to distinguish between a broken network connection and a server that is sometimes not responding. From the simulations point of view, both scenarios look identical.

To set this test up the command in Figure 63 can be used. It sends packets until {Ctrl}-C is pressed. Then it writes the summary statistics and closes the file.

```
ping 192.168.188.36 -t >>C:\log.txt
```

Figure 63. Windows Command Prompt for infinite packet sending

For demonstration purposes, the log file in Figure 64 was created by disconnecting the network cable for about a second. This resulted in one request time-out and a packet loss of about 6%. When this test runs for a longer time with more than 100,000 packets one lost packet is not an issue. This log file is best analyzed by considering the average roundtrip time and packet loss. If there are no lost packets, then the possibility exists that there is a server issue, perhaps resulting from low server performance due to high CPU usage.



```
*server_test.txt - Editor
Datei Bearbeiten Format Ansicht Hilfe

Pinging 192.168.188.36 with 32 bytes of data:
Reply from 192.168.188.36: bytes=32 time<1ms TTL=128
Reply from 192.168.188.36: bytes=32 time<1ms TTL=128
Reply from 192.168.188.36: bytes=32 time<1ms TTL=128
Reply from 192.168.188.36: bytes=32 time<1ms TTL=128
Reply from 192.168.188.36: bytes=32 time<1ms TTL=128
Reply from 192.168.188.36: bytes=32 time<1ms TTL=128
Reply from 192.168.188.36: bytes=32 time<1ms TTL=128
Reply from 192.168.188.36: bytes=32 time<1ms TTL=128
Request timed out.
Reply from 192.168.188.36: bytes=32 time<1ms TTL=128
Reply from 192.168.188.36: bytes=32 time<1ms TTL=128
Reply from 192.168.188.36: bytes=32 time<1ms TTL=128
Reply from 192.168.188.36: bytes=32 time=3ms TTL=128
Reply from 192.168.188.36: bytes=32 time=130ms TTL=128
Reply from 192.168.188.36: bytes=32 time=22ms TTL=128
Reply from 192.168.188.36: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.188.36:
    Packets: Sent = 16, Received = 15, Lost = 1 (6% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 130ms, Average = 10ms|
```

Figure 64. Sample log file for an infinite connection test

D. UNIT TESTING PRINCIPLES AND BEST PRACTICES

There are two different approaches for testing the integrity of a recording tool. The first is described in Figure 65. A stream of defined DIS PDUs is sent to a DIS logger to get the first fingerprint of the stream. A perfect approach for sending out all 72 PDUs in a repeatable way is the Java class AllPduSender.java from the DIS 7 examples. From the logger the stream is resent and recorded with the preferred DIS recording tool. That tool saves the stream in whatever format is chosen. The tool should now replay the stream. The stream is sent to the DIS logger used for the first reference copy.

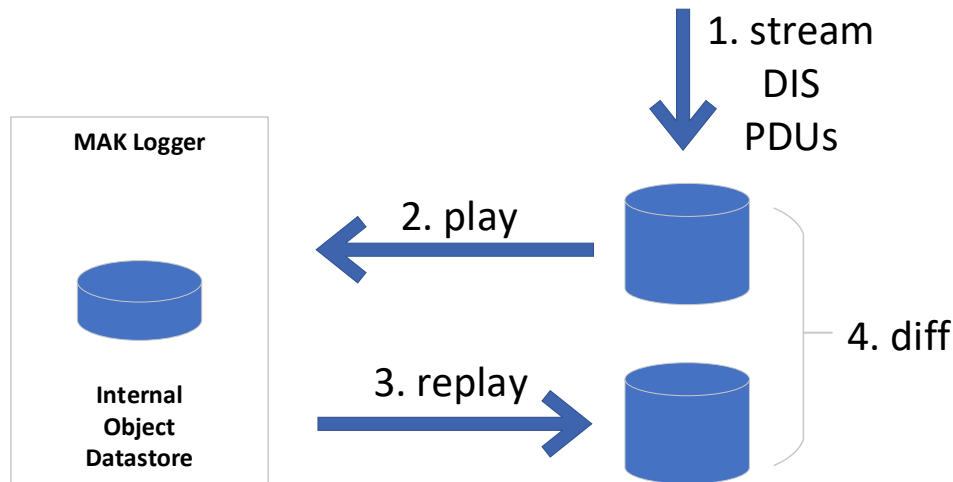


Figure 65. MAK Logger Unit Test Steps: Capture stream, play, store, replay, then perform difference (diff) comparison

After the second reference copy is taken, the two reference copies can be compared. If there is no difference in the fingerprints, the tested DIS recording tool is validated in accordance with IEEE Std 1278.2–2015 (IEEE, 2012).

E. QUALITY ASSURANCE (QA) OF PLAYED BACK STREAMS

To test the integrity of DIS streams being recorded and played back it is important that the packets being recorded match the packets being played back. A first step to test for this is to record all 72 PDUs sent by AllPduSender.java, record them with Wireshark, play them back, and record the packets again. A difference between the Wireshark output files would indicate an error in the recorder.java or player.java.

It is important to not just encode and decode the streams and play them back. If there are just strings copied around there is no way to get a handle to the coordinates. Therefore, the player must decode the stream and parse the strings into an integer array so that coordinates can be extracted.

F. UNIT TESTING OF SOFTWARE PACKAGE INTEGRITY

One objective of unit testing is to isolate a section of code, run it repeatedly, and verify its correctness every time it runs. The code isolation is usually performed by the developer during the development process because the code is best known by its author.

Working on a software packet that is in use in all kinds of civilian and military simulations is dangerous. If something breaks, the software users face errors not knowing where they originate. One important way to prevent this is to implement unit testing. Unit testing is the use of code snippets to test the integrity of code before and after changing it.

Unit testing for sending and receiving PDUs, for example, is realized by sending all different 72 PDUs over the network and to a hash map. The transmitted PDUs are captured by a receiver and put into another hash map. In a first step, the two hash maps are tested for same size of key set. If this holds true, both keysets are used to compare the key value of each in key in both sets. If this holds true, the assumption is made that sent and received PDUs are the same. For the DIS distribution the described behavior is implemented in `edu.nps.moves.dis7.AllPduRoundTripTest.java` (Figure 66). If `AllPduRoundTripTest.java` is run, it shows a message with the number of packets sent and received. If the number of received packets equals the number of sent packets, `AllPduRoundTripTest.java` displays the test result “true” (Figure 67).

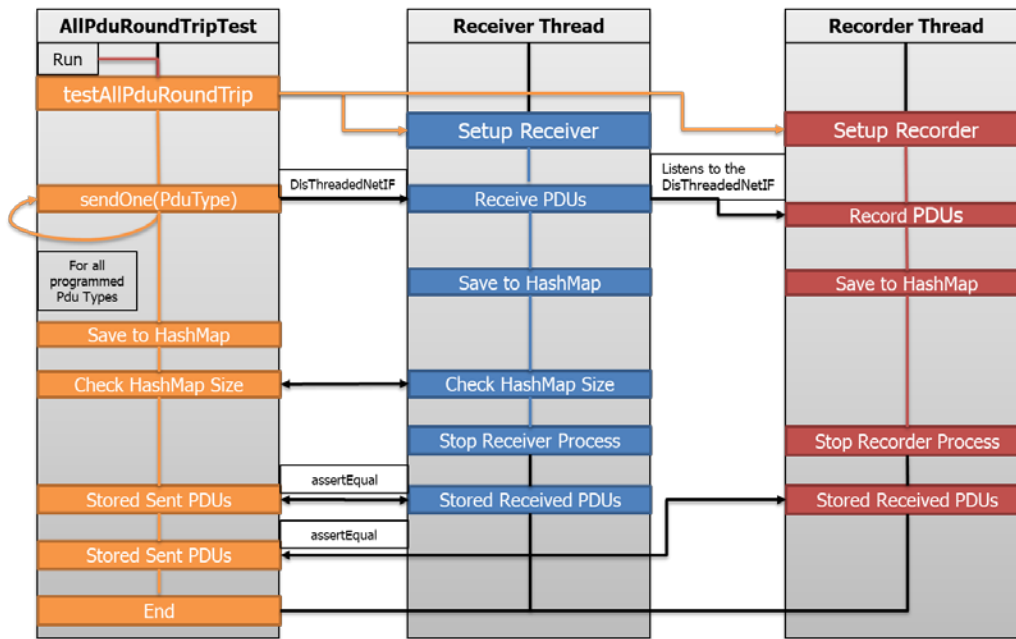


Figure 66. UML Diagram for AllPduRoundTripTest.java Comparison.
Source: Schutt (2019).

If the number of received packets does not equal the number of sent packets, the result is “false” (Figure 68).

```
Recorder log file closed
pduReceivedMap.size()=72, pduSentMap.size()=72, match=true
```

Figure 67. Status message after a successful run of
AllPduRoundTripTest.java

```
Recorder log file closed
pduReceivedMap.size()=67, pduSentMap.size()=72, match=false
No pdus, or not all sent pdus, received ==> expected: <true> but was: <false>
```

Figure 68. Status message after an unsuccessful run of
AllPduRoundTripTest.java

Hints for finding errors when running AllPduRoundTripTest.java can be found in Chapter V.

To automate this test, AllPduRoundTripTest.java can be run automatically as part of every software build that is run. Failing AllPduRoundTripTest.java fails the build, signaling that something is broken within the package.

The described procedure is not a bulletproof way of testing the integrity of PDUs, but it flags a large class of unacceptable errors that otherwise make it to the end-user distribution without any failure notification.

G. TEST FOR NUMBERS OF PACKETS RECEIVED

A simple test for the fidelity of a receiver/recorder is to count the number of valid PDUs received within a given time window. For this test we need a simulation and two receivers on one network.

The two recorders are set to listening mode before the simulation is started. In this case, started means that the simulation software is not running. This can be verified by checking the packet counters in the two simulations. This is important because VR Forces (for example) regularly sends out DIS packets to test the network. Starting the recorders after the simulation can lead to a difference in counted packets.

Both recorders capture PDU packets from a small scenario. The recorders are not stopped before the simulation software is properly shut down to ensure that all packets are captured. After the simulation is stopped, the two recorders are also shut down. If both recorders come from the same software distribution/developer, the output files can be compared directly by finding the difference in the files. If not, an inspection or count by hand has to be done to see how many packets were received (Figure 69).

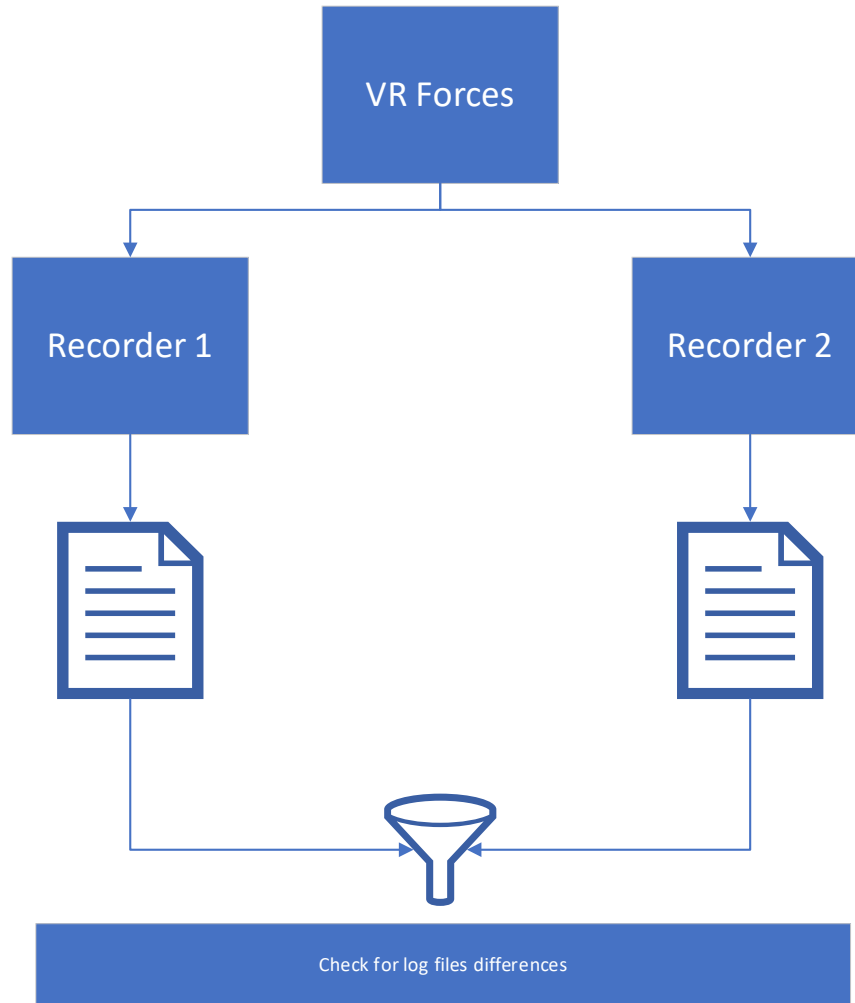


Figure 69. Two recorders connected to one simulation to check for differences in the recorder output

H. COMPARATOR FOR DIS STREAMS

The sketch for a new Java class called `DisStreamComparator.java` is described in this paragraph and a flow chart is provided in Figure 70. The idea is to use a separate Java class to read in two different streams. The streams do not require the same format or encoding. The class compares both inputs and reports some statistics and values that provide evidence on whether the inputs are the same. If not the same, then the provided statistics and values help users decide why and how much they differ. Eventually this work is expected to implement the Java interfaces “comparable” and “comparator” so that the full Java functionality can be gained seamlessly (Naftalin & Wadler, 2007).

Although streams can theoretically be infinite, this work treats them as finite because tools use parts with a defined beginning and end. A continuous ongoing stream might be saved in sections without loss of data.

The class `DisStreamComparator.java` has two arguments as inputs. The two arguments are DIS streams in whatever encoding or file format they are available. Both files will be decoded to obtain the raw PDUs. The raw PDUs are stored in buffers to make the comparison and collection of comparison results easier.

After both streams are decoded and stored into buffers, the buffer size must be compared. If the size is equal more tests will be run. If the size is not equal a test for checking whether one stream is part of the other must be run. One way to achieve this is to test whether three subsequent PDUs of the smaller buffer are within the other buffer. If this holds true, all PDUs from the smaller buffer must be tested. If this also holds through, the smaller stream is a subset of the larger stream.

To get a fingerprint of PDU streams it is helpful to evaluate the PDU distribution and display it as block diagram. Eyeballing two block diagrams provides a fast way to see where one PDU stream differs from another.

If they are equal and other tests can be run it is useful to start with a test for equal timestamps, which is useful if both streams were recorded on the same machine. A method iterates over the first stream, extracts the timestamp from the first PDU, and tests the first PDU in the second stream for an equal timestamp. If this holds true for all timestamps, this test can be repeated for PDU types and PDU content.

PDU streams captured on different machines can have different timestamps if they do not have access to NTP. Therefore, the test for timestamps must be modified. The modification might test for the difference in timestamps and not for absolute timestamps. A run-time test is likely advisable to test whether it is faster to test all arguments of a PDU at once or in subsequent test segments.

In general, collections of behavior-based streams for analysis of simulation data for track, motion and interaction do not appear to be a common practice or expected capability.

Thus addition and deployment of the techniques shown in this thesis, as common tools of analysis, has broad potential value for simulation practitioners, analysts, and end users alike.

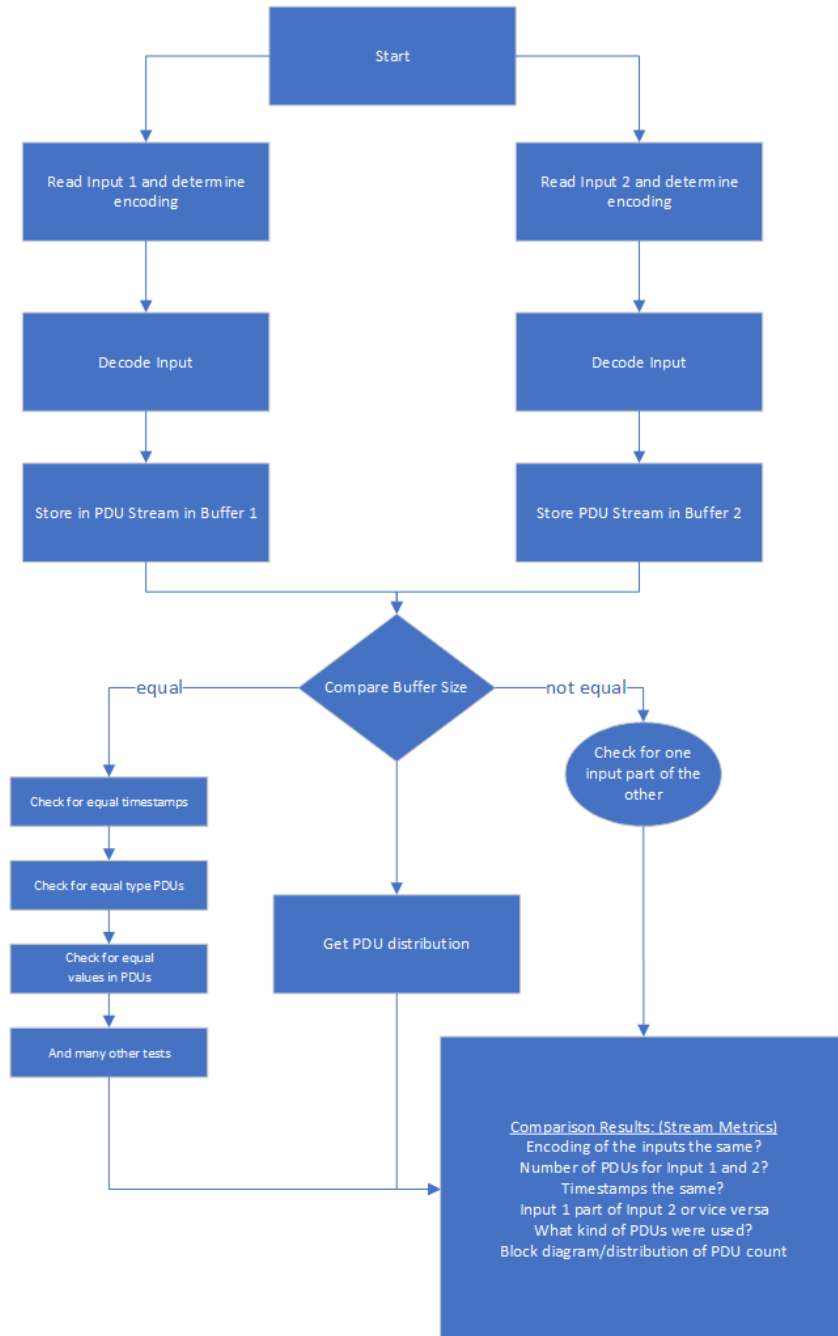


Figure 70. Flow chart for proposed DisStreamComparator.java unit testing utility class

I. VERIFICATION, VALIDATION AND ACCREDITATION (VV&A) FOR LVC APPLICATION AND SCENARIOS

The worst error that can happen to a system is the one that goes unnoticed. Therefore, it is critically important to install a mechanism that repeatably brings up any issues within the development process of code that is used for LVC. For simulations, unit testing of behavior streams holds important value and indeed might be specified as a future contract requirement for maintaining and developing an LVC code base.

Figure 71 is derived from Figure 42 by adding two boxes for Unit Testing and a reference pdu.dislog and the labels Verification, Validation and Accreditation to it. It shows how recording and playing back a PDU stream can be used to test a code base for newly introduced errors. When this is run automatically after each code change, an error in code is automatically discovered.

Before handing out code to a contractor a reference PDU stream must be recorded and saved to a reference pdu.dislog file. Following a software change, if recorded and played back PDU streams are equal and the recorded pdu.dislog matches the reference pdu.dislog, then there is enough evidence that no errors were introduced to the code handling the PDUs.

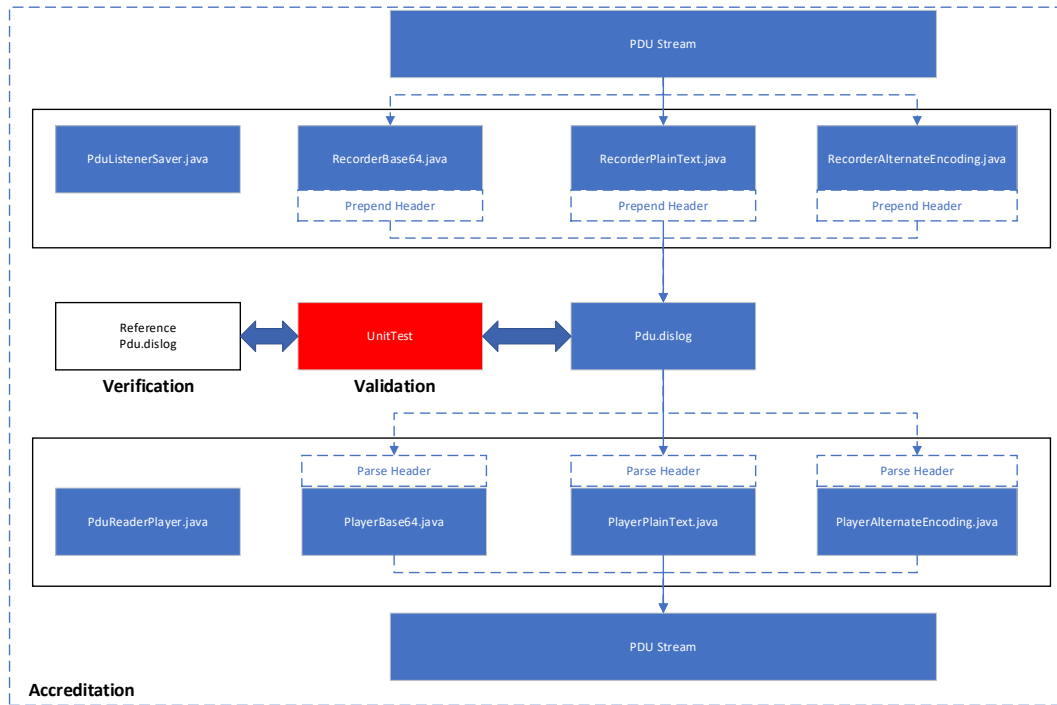


Figure 71. Flowchart for using VV&A procedures to prove that code for behavior stream handling is free of errors

DIS stream recording for unit testing of software stability, application interoperability, and LVC replay capabilities provide powerful capabilities with great potential benefit. It is important to continue pursuit of these potential industry-wide benefits through further reviews in the SISO VV&A forum.

J. TROUBLESHOOTING

1. Wireshark

If one of the unit tests in Chapter V fails, then it is best to have a list of possible solutions handy. The best tool to start with is Wireshark. Wireshark monitors network traffic and can show whether PDUs are sent over the network. If they are, Wireshark can display source, destination, and content. Refer to Chapter IV and the online documentation to see how Wireshark can be utilized to capture PDUs.

2. Test Setup with OpenDis7Examples

If Wireshark does not capture any DIS packets it is helpful to check the recorder.java, PDUListenerSaver.java or PduReaderPlayer.java for IP address and port. The file player.java does not have to be checked. It uses the configuration from the recorder.java.

To test a network, it is best to see whether it can handle multicast distributions. The easiest testcase is to setup a computer with NetBeans and a clone of the MV3500 files from <https://gitlab.nps.edu/Savage/NetworkedGraphicsMV3500>. In the folder OpenDis7Examples in the MV3500 Examples project are two files that are useful to achieve a simple test setup: EspduReceiver.java and EspduSender.java. The class EspduReceiver.java must be run first. After the receiver has prompted to the output window that is started, EspduSender.java can be run. EspduSender.java immediately starts to send 4 ESPDUs. Those four must now be printed to the output window of EspduReceiver.java.

If this test is successful, EspduSender.java and EspduReceiver.java can be run on different machines.

3. Firewall Settings

When ESPDUs are being sent from one computer to another there are often configurations necessary that enables the machines to receive messages from each other. For computers running Microsoft Windows 10 (Version 1909 or older), it is important to add an incoming rule to the systems firewall allowing PDUs on Port 3000 (default) to pass. It is likely that newer versions of Windows 10 must be configured in the same way. In Appendix B, a manual for configuring the Windows 10 Firewall is provided.

4. AllPduRoundTripTest.java

Although AllPduRoundTripTest.java is written for the purpose of automatically testing whether code was changed leading to inconsistent PDUs, the class can be used to check whether there are issues with multi-threading on the test computer. When conducting this research, a correlation was found between number of CPU cores and missing PDUs after one run of AllPduRoundTripTest.java.

CPUs with 8-cores/16-threads, like AMD Ryzen 7, do not require sleep commands within the sendOne method. Slower dual-core notebook CPUs like Core i7 (6th generation) need a sleep time of 250 ms to capture 72 out of 72 packets.

When the sleep time is not added the CPU cannot capture all packets and the test fails. Numbers of 52–68 out of 72 were recognized when running tests for this thesis.

A CPU with at least 4-cores/8-threads is highly recommended for working with multi-threading using OpenDIS 7.

5. CPU Speed

For this research many runs with PDUs being recorded were made. A table shows how many PDUs were recorded within a defined time on a scenario that comes built into VR Forces. The scenario used for testing the CPU speed is HawaiiGround. VR Forces workstation and computers recording PDUs are on the same network. All tablets, laptops, or desktops run Windows 10 Pro. All computers are equipped with solid-state drives. The Java code runs with the NetBeans console switched off to avoid irrelevant input/output delays.. Table 6 shows the number of recorded PDUs within a run of the first three minutes of the HawaiiGround default scenario. From the slowest to the fastest tested CPU, a difference of about factor 30 is noticeable.

Table 6. Number of PDUs recorded within three minutes for different CPU types

Computer	CPU	RAM	#PDUs recorded
Surface Pro 6 Tablet	Intel Core i5-8250U	8 GB	394
Surface Pro X Tablet	Microsoft SQ1	16 GB	580
ThinkPad T470 Laptop	Intel Core i7-7500u	32 GB	695
Customized Desktop	AMD Ryzen 3 3200G	64 GB	9,015
Customized Desktop	AMD Ryzen 7 2700X	32 GB	12,763

Following extensive debugging and diagnostics, the dramatic difference in performance is likely due to poor threading implementation. Significant improvements

may be possible for all systems following a future performance review and enhancement of the OpenDIS7 library.

To work with PDUs it is preferable to have a powerful computer with a desktop CPU that can handle the threading efficiently. Unfortunately, no Intel desktop CPUs were available to measure their performance. The only available desktop CPU available is built into the VR Forces workstation. That CPU was already at 100% CPU load running VR Forces. Running NetBeans with a Java class that uses many threads would have been an unfair comparison. It might be the high CPU utilization by VR Forces that enabled recording 710 PDUs within three minutes.

6. Test Dead Reckoning (DR)

Sometimes it can be useful to test whether the dead reckoning and entity timeout is working properly. The easiest way to set this up is through an entity performing multiple small turns of about 30° (Figure 72) and recording this as a stream using the technique described in Chapter IV. The entity must then be deleted from the simulation, so that it does not interfere with the entity being played back from the stream. To test the playback the stream must be played back to the simulation. The entity is following the originally planned and recorded route, doing multiple small turns.



To test the dead reckoning parameters the stream must be played back to the simulation, but right before a small turn the network connection must be interrupted. Now the entity follows the last known heading with the last known speed until either the network connection works again within five seconds or the entity disappears because no ESPDU is received for more than five seconds. When an update is received within five seconds the entity jumps back to the position reported by the ESPDU received after the connection was reestablished.

To give an example, the setup shown in Figure 72 is run. The PDU stream of the runs is recorded on a second computer. A screen video capture of the run is recorded (Figure 73, left image). After the DIS run, the entity and the track are deleted. The simulation time is reset to the beginning, waiting for the stream to be played back. The first playback has no issues with the network connection so the track from the playback equals the track of the original simulation run (Figure 73, middle image). The last run played back has in interrupted network connection starting at waypoint two. Therefore, the simulation is starting to make use of the DR algorithm and approximating the entity's location based on the last known speed and heading (Figure 73, right image). The position at waypoint three is now different from the DIS run. After the network connection is reestablished the entity instantly jumps back to the position that was published in the first ESPDU received.



From left to right: DIS run at Waypoint 3, DIS replay at Waypoint 3, DIS replay at Waypoint 3 with applied dead reckoning after losing network connection at Waypoint 2.

Figure 73. Images of DIS run, DIS replay, and DIS replay with DR side by side

To get an better understanding of dead reckoning, the three MP4 videos the screenshots were taken from, are available for playback in Figure 74, Figure 75, and Figure 76.



Figure 74. DIS original run of the scenario in Figure 72



Figure 75. DIS replay of the scenario Figure 72



Figure 76. DIS replay of the scenario in Figure 72 with network connection interrupted twice

If the entity behaves like the one in the videos, the dead reckoning is working.

K. FUNDAMENTAL IMPORTANCE OF REPEATABLE MEASUREMENTS

According to R. W. Hamming, from his book *The Art of Doing Science and Engineering*, “You get what you measure” (Hamming, 2003, Chapter 29), the decision to measure and choice of metrics is fundamentally important for understanding and analyzing a problem. Indeed, ongoing measurements have a profound influence on the shared understanding of a problem. Meanwhile missing, incorrect or ill-conceived measurements can have a confusing or confounding impact on the understanding of the problem. When this principle is applied to the domain being studied in this thesis, it is important to know what is broadcasted in order to have the ability to compare results for checking the capture of all PDUs that were sent. Simply put, not knowing the ground truth leads to incorrect measurements. Modern principles of unit testing for quality assurance (QA) and software cybersecurity require such testing before release. Interoperability among numerous DIS applications typically appears to be excellent, apparently conducted through extensive live exchange of streams between systems. Given the critical nature of LVC and given that subtle data errors can easily remain undetected during informal observation, it is likely that current practice can be improved even further through performance of DIS interoperability unit testing.

The DIS standard was first established by IEEE in 1995. After 25 years it remains surprising that no public evidence of unit testing for LVC application streams can be found. This is a major impediment to repeatable functionality, interoperability between software products, and even the secure distribution of updated applications. Furthermore, the stakes for success and failure could not be greater because many LVC applications are used to access, certify, and qualify the safety and mission-critical nature of human-directed systems providing the capacity for lethal force. No other major categories of software lack such maturity. Further, it is questionable whether the purchase of such software is allowed by either contracting requirements or security certification frameworks. Since DIS is an interoperability standard mapping to all manner of different LVC protocols, this gap exists across an entire range of LVC applications. Therefore, the addition of repeatable measurements and well-defined metrics is fundamentally important for the future of military modeling and simulation.

L. SUMMARY

This chapter underlined the importance of the principle pursued by testing, because the worst error is the unnoticed error. It is important to know that errors can occur at any point in the life cycle of code. The reader learned how to test and evaluate connections with an introduction to unit testing principles and their application. The chapter also examines how to troubleshoot common problems that can occur working with DIS-driven applications. Knowing how to repeatably measure effectively, and why ground truth is important, is a fundamental problem for networked simulation.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

Unit testing of behavior streams is important, ideal, and if integrated within the development they can be suitable as a contract requirement for maintaining and developing LVC code bases.

Working with DIS streams and handling PDUs requires performant computers to ensure that all PDUs are captured. Once a PDU is recorded and saved to a file there are arbitrary many-use cases that can utilize the stream that rests in a file. Some of the use cases were described, e.g., replaying a stream to a simulation, converting a stream to a X3D entity with position and OrientationInterpolators or using a stream for an automated AAR that can be run on any COTS simulation or trainer able to connect via DIS to a network.

Troubleshooting is an important part of daily business when it comes to working with PDUs. There are many factors that influence the number of PDUs that are being sent or recorded.

Writing PDU handling code in Java makes it available to a broad community. DIS7 still supports Java 8 to let all users make it compatible to MOTS, GOTS and COTS simulations.

Evaluation of behavior-based LVC streams can have a major impact on training, evaluation, exercise analysis and interoperability between Command and Control (C2) and M&S.

Knowing how to measure is important for every field in science. It is even more important in a field where LVC applications are used to access, certify, and qualify the safety and mission-critical nature of human-directed systems with the capacity for lethal force. More work is necessary in this area of LVC simulation.

B. RECOMMENDATIONS FOR FUTURE WORK

Extending the proof of concept of converting an entity's movement recorded in a DIS stream into a X3D entity driven by a position and OrientationInterpolator to multiple entities each driven by their own interpolator can be a useful first step.

A large second step would be finding a useful representation within the X3D model exchange, auto-generating code to handle prototype imports for all 72 PDUs and animating them, so that a high-fidelity 3D environment would be accessible using open-source web standards.

APPENDIX A. LIST OF ALL PDUS

DIS is a standard for running real-time simulations across multiple networks and computers hosting simulations. The first standard for DIS was defined in IEEE 1278–1993–Standard for Distributed Interactive Simulation–Application Protocols [1]. DIS is defined in IEEE Standard 1278. The 72 different types of PDUs are:

1. ENTITY_STATE (Entity Information/Interaction)

An Entity State PDU is issued whenever an entity moves or rotates within the world coordinate system.

2. FIRE (Warfare)

The Fire PDU is used to communicate information about firing a weapon or expendable.

3. DETONATION (Warfare)

A Detonation PDU is issued when a fired weapon causes a detonation or for example, a landmine explodes.

4. COLLISION (Entity Information/Interaction)

Collisions between entities are communicated by issuing a Collision PDU.

5. SERVICE_REQUEST (Logistics)

Requests for logistics support are communicated by issuing a Service Request PDU.

6. RESUPPLY_OFFER (Logistics)

Offerings of supplies are communicated by issuing a Resupply Offer PDU.

7. RESUPPLY_RECEIVED (Logistics)

Receipt of supplies are communicated by issuing a Resupply Received PDU

8. RESUPPLY_CANCEL (Logistics)

Resupply Cancel PDUs are used to communicate the canceling of a resupply service provided through logistics support.

9. REPAIR_COMPLETE (Logistics)

Repair Complete PDUs are used by the repairing entity to communicate the repair that has been performed for the entity that requested repair service.

10. REPAIR_RESPONSE (Logistics)

An entity receiving a Repair Complete PDU from its repairing entity, must acknowledge the receipt of the repair by issuing a Repair Response PDU.

11. CREATE_ENTITY (Simulation Management)

The creation of a new entity is communicated using a Create Entity PDU.

12. REMOVE_ENTITY (Simulation Management)

The removal of an entity from an exercise is communicated with a Remove Entity PDU.

13. START_RESUME (Simulation Management)

The Start/Resume of an entity/exercise is communicated using a Start/Resume PDU.

14. STOP_FREEZE (Simulation Management)

The stopping or freezing of an entity/exercise is communicated using a Stop/Freeze PDU.

15. ACKNOWLEDGE (Simulation Management)

The acknowledgment of the receipt of a Start/Resume PDU, Stop/Freeze PDU, Create Entity PDU, or Remove Entity PDU is communicated by issuing an Acknowledge PDU.

16. ACTION_REQUEST (Simulation Management)

A request from a Simulation Manager to a managed entity to perform a specified action is communicated using an Action Request PDU.

17. ACTION_RESPONSE (Simulation Management)

When an entity receives an Action Request PDU, that entity acknowledges the receipt of the Action Request PDU with an Action Response PDU.

18. DATA_QUERY (Simulation Management)

A request for data from an entity is communicated by issuing a Data Query PDU.

19. SET_DATA (Simulation Management)

Initializing or changing internal state information is communicated using a Set Data PDU.

20. DATA (Simulation Management)

Information issued in response to a Data Query PDU or Set Data PDU is communicated using a Data PDU.

21. EVENT_REPORT (Simulation Management)

A managed entity is reporting the occurrence of a significant event to the Simulation Manager using an Event Report PDU.

22. COMMENT (Simulation Management)

Arbitrary messages can be entered into the data stream by using a Comment PDU. All simulation data that cannot be transported by any other PDU can be encapsulated into a Comment PDU.

23. ELECTROMAGNETIC_EMISSION (Distributed Emission Regeneration)

The Electromagnetic Emission PDU is used to communicate active electromagnetic emissions, including radar and radar related electronic warfare.

24. DESIGNATOR (Distributed Emission Regeneration)

The Designator PDU is used to communicate information for designation functions to support a laser-guided weapon engagement.

25. TRANSMITTER (Radio Communications)

The Transmitter PDU is used to communicate the state of a particular radio transmitter or simple intercom.

26. SIGNAL (Radio Communications)

The Signal PDU is used to convey the audio or digital data carried by the simulated radio or intercom transmission.

27. RECEIVER (Radio Communications)

The Receiver PDU is used to communicate the state of a particular radio receiver.

28. IDENTIFICATION_FRIEND_OR_FOE (Distributed Emission Regeneration)

Information about military and civilian interrogators, transponders, and other specific electronic systems is communicated using the IFF PDU.

29. UNDERWATER_ACOUSTIC (Distributed Emission Regeneration)

Information about acoustic emissions is communicated using an Underwater Acoustic PDU.

30. SUPPLEMENTAL_EMISSION_ENTITY_STATE (Distributed Emission Regeneration)

Certain supplemental information on an entity's physical state and emissions is communicated using the Supplemental Emission Entity State PDU.

31. INTERCOM_SIGNAL (Radio Communications)

The actual transmission of intercom voice or data is communicated by using an Intercom Signal PDU.

32. INTERCOM_CONTROL (Radio Communications)

Detailed information about the state of an intercom device and the actions it is requesting of another intercom device or the response to a requested action is communicated by an Intercom Control PDU.

33. AGGREGATE_STATE (Entity Management)

Detailed information about aggregating entities and communicating information about the aggregated entities is communicated using the Aggregate State PDU.

34. ISGROUPOF (Entity Management)

Information about a particular group of entities is communicated by issuing an IsGroupOf PDU.

35. TRANSFER_OWNERSHIP (Entity Management)

Information initiating the dynamic allocation and control of simulation entities between two simulation applications is initiated by a Transfer Ownership PDU.

36. ISPARTOF (Entity Management)

The joining of two or more simulation entities forms a single entity with constituent parts that is communicated by issuance and acknowledgment of the IsPartOf PDU.

37. MINEFIELD_STATE (Minefield)

The Minefield State PDU provides information about the complete minefield.

38. MINEFIELD_QUERY (Minefield)

The Minefield Query PDU provides the means by which a simulation queries a minefield simulation for information on the individual mines contained in a minefield when operating in Minefield Query Response Protocol mode.

39. MINEFIELD_DATA (Minefield)

Information about the location and status of a collection of mines in a minefield is communicated using the Minefield Data PDU on an individual mine basis.

40. MINEFIELD_RESPONSE_NACK (Minefield)

The Minefield Response NACK PDU contains information about the requesting entity and the PDUs that were not received in response to a query.

41. ENVIRONMENTAL_PROCESS (Synthetic Environment)

The Environmental Process PDU communicates information about the environment, including simple environment variables, small-scale environmental updates, and embedded processes.

42. GRIDDED_DATA (Synthetic Environment)

The Gridded Data PDU transmits information about large-scale or high-fidelity spatially and temporally varying ambient fields and about environmental features and processes.

43. POINT_OBJECT_STATE (Synthetic Environment)

The Point Object State PDU communicates the addition/modification of a synthetic environment object that is geometrically anchored to the terrain with a single point.

44. LINEAR_OBJECT_STATE (Synthetic Environment)

The Linear Object State PDU communicates the addition/modification of a synthetic environment object that is geometrically anchored to the terrain with one point and has a segment size and orientation.

45. AREAL_OBJECT_STATE (Synthetic Environment)

The Areal Object State PDU communicates the addition/modification of a synthetic environment object that is geometrically anchored to the terrain with a set of at least three points that come to a closure.

46. TIME_SPACE_POSITION_INFORMATION (Live Entity)

The Time Space Position Information PDU communicates information about the live entity's state vector. This PDU includes state information that is necessary for the receiving simulation applications to represent the issuing live entity's location and movement in its own simulation.

47. APPEARANCE (Live Entity)

The Appearance PDU communicates information about the appearance of a live entity. This includes state information that is necessary for the receiving simulation applications to represent the issuing live entity's appearance in the simulation application's own simulation.

48. ARTICULATED_PARTS (Live Entity)

The Articulated Parts PDU communicates information about an entity's articulated and attached parts.

49. LIVE_ENTITY_FIRE (Live Entity)

A Live Entity Fire PDU represents weapons fire in a DIS exercise involving live entities.

50. LIVE_ENTITY_DETONATION (Live Entity)

The Live Entity Detonation PDU communicates information associated with the impact or detonation of a munition.

51. CREATE_ENTITY_RELIABLE (Simulation Management with Reliability)

The Create Entity Reliable PDU has the same function as the Create Entity PDU (see 11) but with the addition of reliability service levels.

52. REMOVE_ENTITY_RELIABLE (Simulation Management with Reliability)

The Remove Entity Reliable PDU has the same function as the Remove Entity PDU (see 12) but with the addition of reliability service levels.

53. START_RESUME_RELIABLE (Simulation Management with Reliability)

The Start Resume Reliable PDU has the same function as the Start Resume PDU (see 13) but with the addition of reliability service levels.

54. STOP_FREEZE_RELIABLE (Simulation Management with Reliability)

The Stop Freeze Reliable PDU has the same function as the Stop Freeze PDU (see 14) but with the addition of reliability service levels.

55. ACKNOWLEDGE_RELIABLE (Simulation Management with Reliability)

The Acknowledge Reliable PDU has the same function as the Acknowledge PDU (see 15) but with the addition of reliability service levels.

56. ACTION_REQUEST_RELIABLE (Simulation Management with Reliability)

The Action Request Reliable PDU has the same function as the Action Request PDU (see 16) but with the addition of reliability service levels.

57. ACTION_RESPONSE_RELIABLE (Simulation Management with Reliability)

The Action Response Reliable PDU has the same function as the Action Response PDU (see 17) but with the addition of reliability service levels.

58. DATA_QUERY_RELIABLE (Simulation Management with Reliability)

The Data Query Reliable PDU has the same function as the Data Query PDU (see 18) but with the addition of reliability service levels.

59. SET_DATA_RELIABLE (Simulation Management with Reliability)

The Set Data Reliable PDU has the same function as the Set Data PDU (see 19) but with the addition of reliability service levels.

60. DATA_RELIABLE (Simulation Management with Reliability)

The Data Reliable PDU has the same function as the Data PDU (see 20) but with the addition of reliability service levels.

61. EVENT_REPORT_RELIABLE (Simulation Management with Reliability)

The Event Report Reliable PDU has the same function as the Event Report PDU (see 21) but with the addition of reliability service levels.

62. COMMENT_RELIABLE (Simulation Management with Reliability)

The Comment Reliable PDU has the same function as the Comment PDU (see 22) but with the addition of reliability service levels.

63. RECORD_RELIABLE (Simulation Management with Reliability)

The Record Reliable PDU is used to respond to a Record Query Reliable PDU or a Set Record Reliable PDU.

64. SET_RECORD_RELIABLE (Simulation Management with Reliability)

The Set Record Reliable PDU is used to set or change certain parameter values.

65. RECORD_QUERY_RELIABLE (Simulation Management with Reliability)

The Record Query Reliable PDU is used to communicate a request for data in record format.

66. COLLISION_ELASTIC (Entity Information/Interaction)

Information about elastic collisions in a DIS exercise is communicated using a Collision-Elastic PDU.

67. ENTITY_STATE_UPDATE (Entity Information/Interaction)

Nonstatic information about a particular entity is communicated by issuing an Entity State Update PDU.

68. DIRECTED_ENERGY_FIRE (Warfare)

The firing of a directed energy weapon is communicated by issuing a Directed Energy Fire PDU.

69. ENTITY_DAMAGE_STATUS (Warfare)

The Entity Damage Status PDU is used to communicate detailed damage information sustained by an entity regardless of the source of the damage.

70. INFORMATION_OPERATIONS_ACTION (Information Operations)

Actions initiated by an IO simulation to support interactions with other IO simulations is communicated using the IO Action PDU.

71. INFORMATION_OPERATIONS_REPORT (Information Operations)

The information operations status of an entity is conveyed using the IO Report PDU.

72. ATTRIBUTE (Entity Information/Interaction)

An Attribute PDU is used to extend another PDU or communicate attributes that are not associated with a specific PDU type.

(IEEE, 2012)

Every PDU is used for a special purpose. The semantics of each message and interaction are documented in detail in IEEE Std 1278.1–2012 p. 31 - 34.

APPENDIX B. FIREWALL CONFIGURATION

A common problem blocking new users is a default firewall connection on individual computers. This appendix shows an example adjustment that allows proper network execution of software.

To get to the Windows Firewall Configuration right click the network icon on the Windows taskbar (Figure 77).

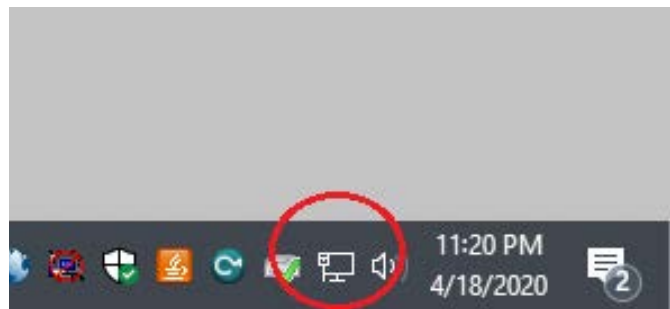


Figure 77. Network Icon in Windows Taskbar

It shows a dialog where “Open Network & Internet settings” must be selected (Figure 78).

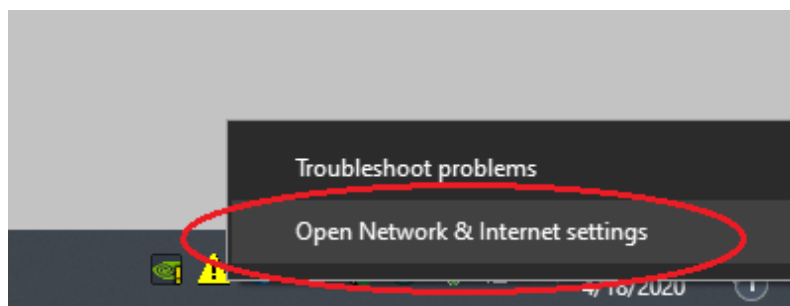


Figure 78. Network Settings Dialog

Selecting “Open Network & Internet settings” displays a window with the network status and a link to the Windows Firewall settings (Figure 79).

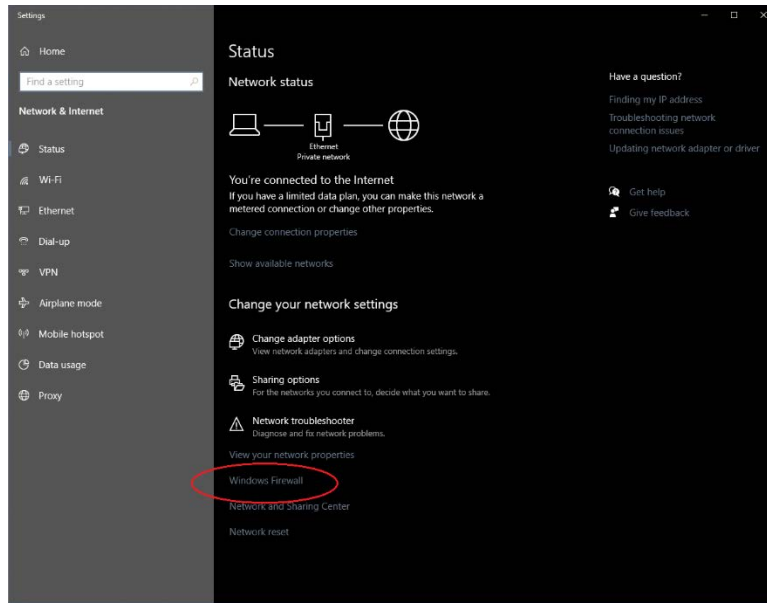


Figure 79. Network Status window

Selecting “Windows Firewall” displays a window with the status for “Firewall & network protection” (Figure 80).

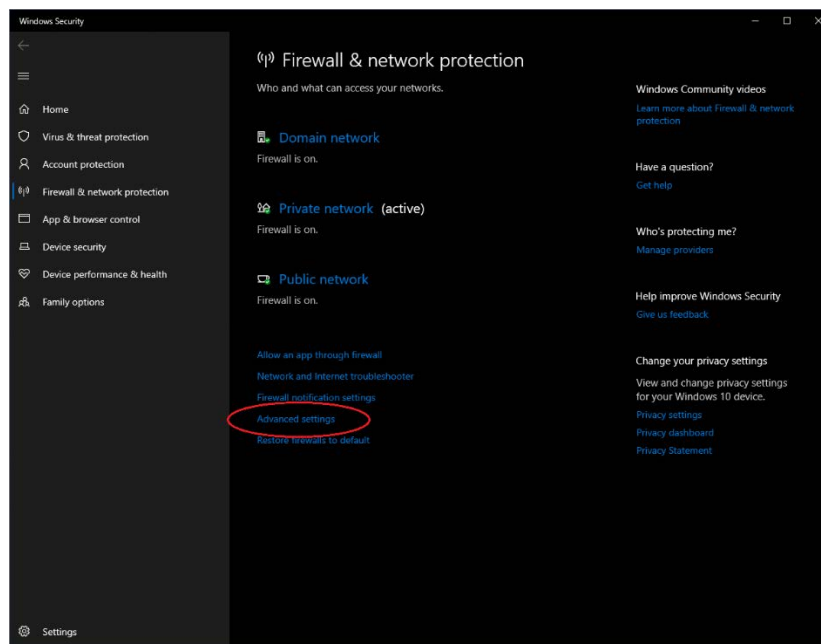


Figure 80. Firewall & Network protection window

After selecting “Advanced settings” the final dialog of changes that must be made is displayed (Figure 81).

To access the inbound rules the menu item “Inbound rules” must be chosen (Figure 81). In Figure 82 all inbound rules are displayed.

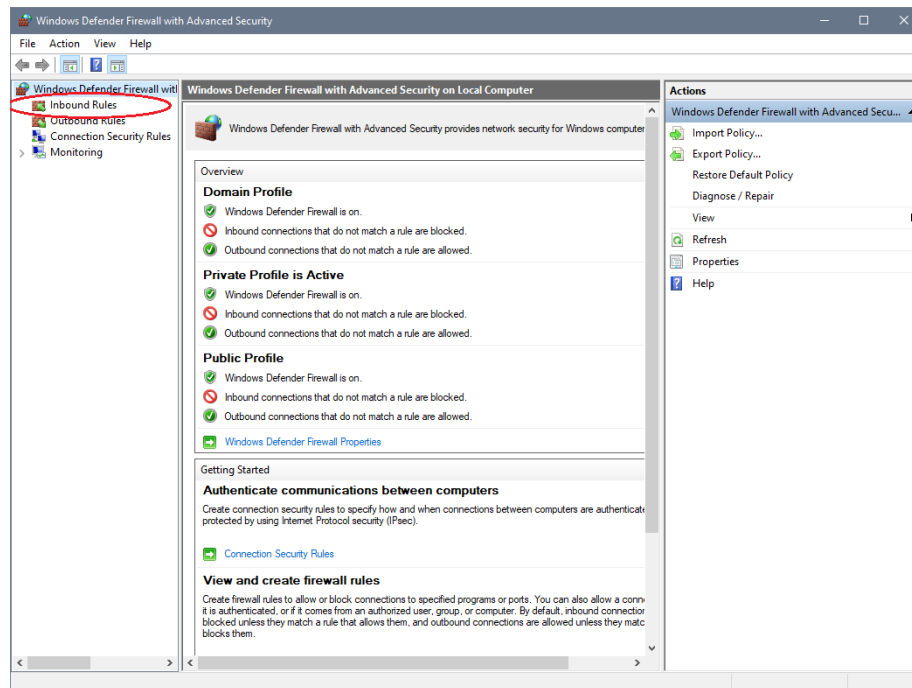


Figure 81. Windows 10 Defender Firewall window

In the Actions box on the right side of Figure 82, the action “New Rule...” must be selected to bring up the “New Inbound Rule Wizard” (Figure 83).

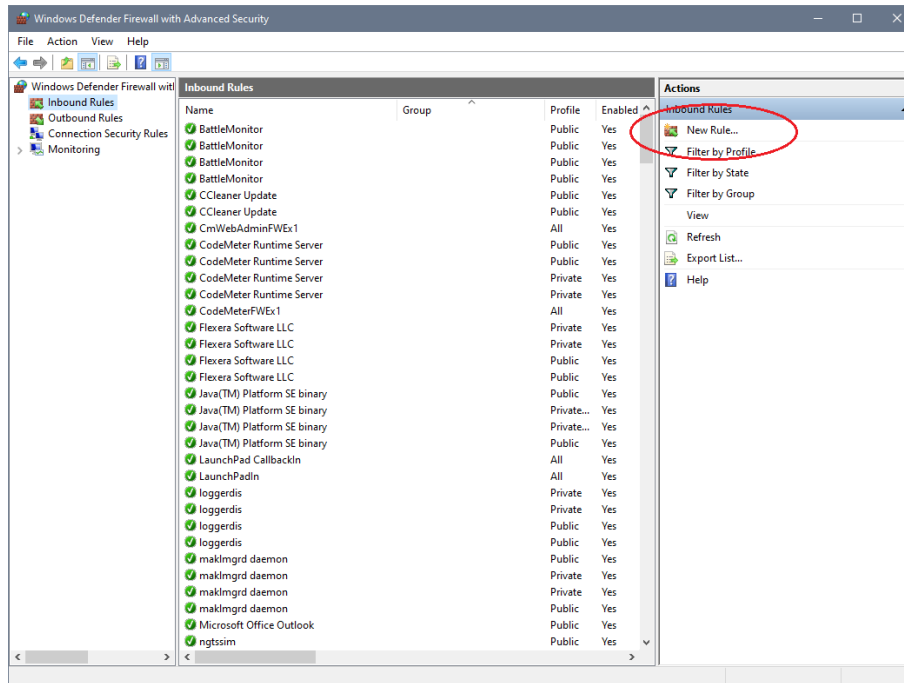


Figure 82. Windows Defender Firewall Inbound Rules view

A single Port 3000 must be opened, so the radio button for “Port” must be selected (Figure 83).

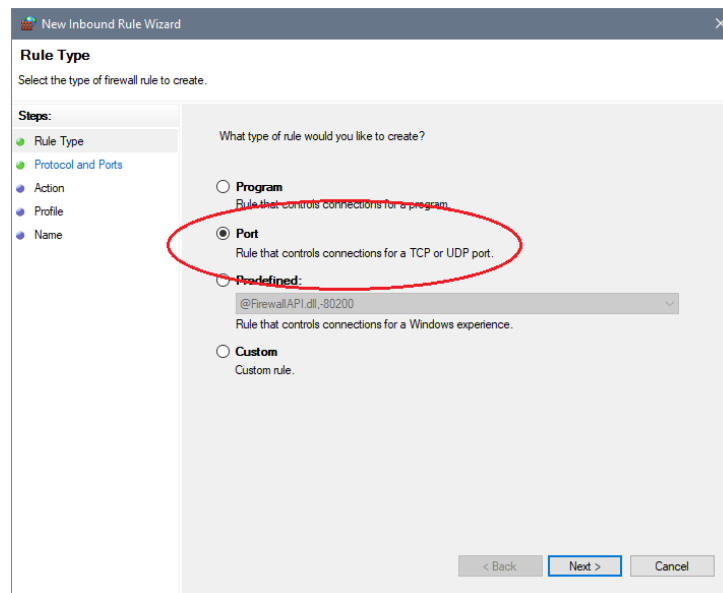


Figure 83. Windows Defender Firewall New Inbound Rule Wizard Type Selection

In Figure 84 UDP must be selected and the port number 3000 must be added to the text field.

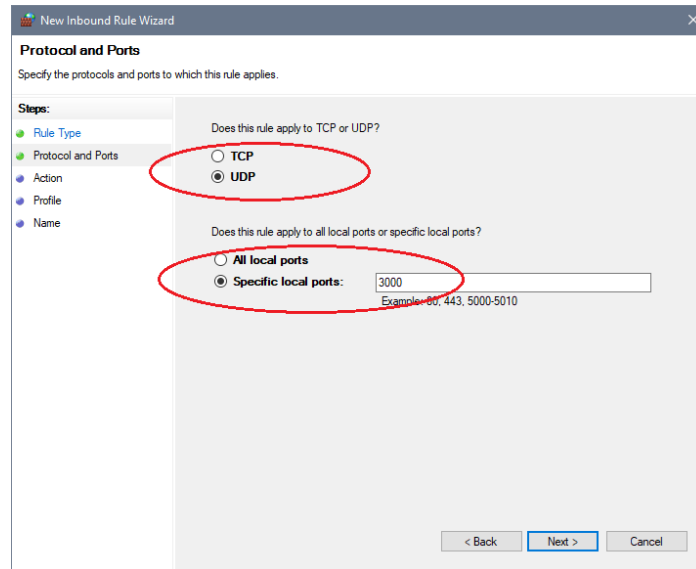


Figure 84. Windows Defender Firewall New Inbound Rule Wizard Port Selection

In Figure 85 the default must not be changed.

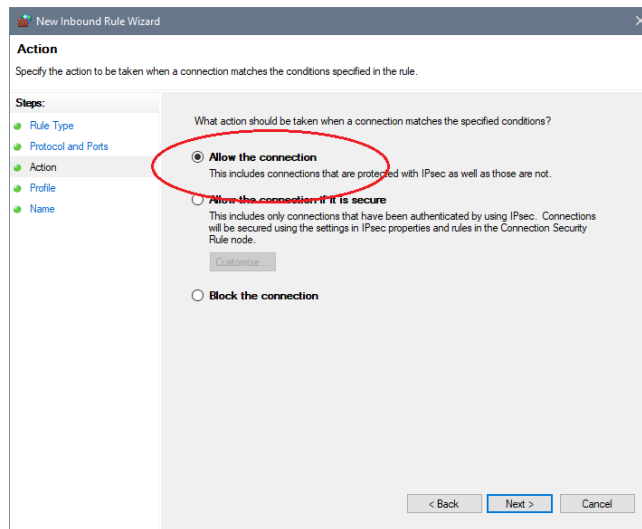


Figure 85. Windows Defender Firewall New Inbound Rule Wizard Allow/Block Dialog

In Figure 86 the rule can be specified for different type of networks. The default can be left unchanged.

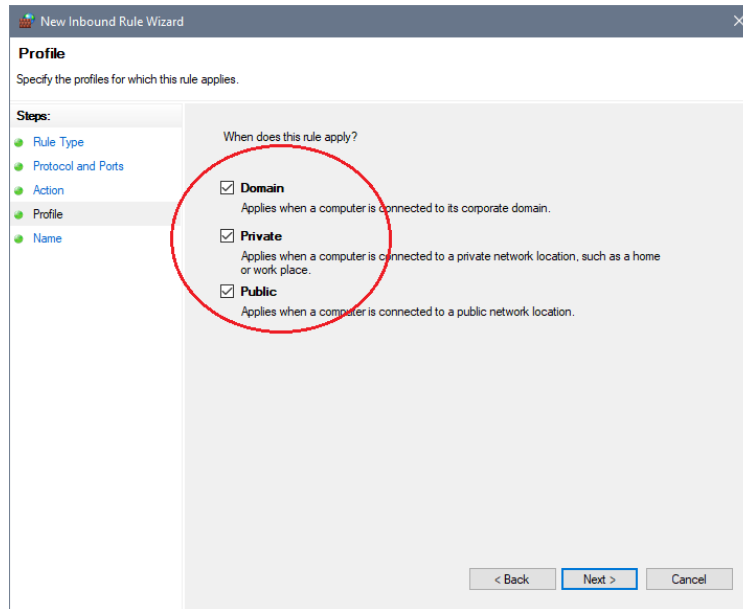


Figure 86. Windows Defender Firewall New Inbound Rule Wizard Network Selection

In the last window of the New Inbound Rule Wizard (Figure 87) a name for the new rule must be provided. It is a best practice to use a descriptive name. After closing the wizard by clicking “Finish” the new rule is displayed within the Windows Defender Firewall Inbound Rules view (Figure 88).

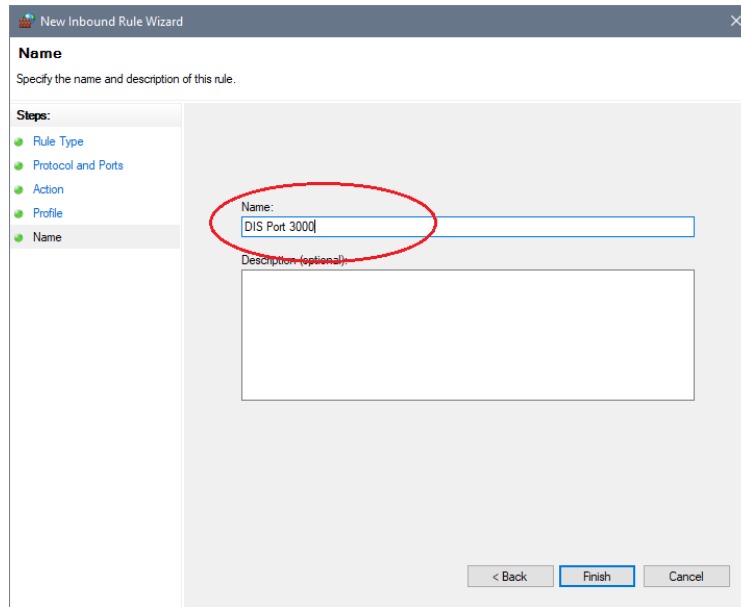


Figure 87. Windows Defender Firewall New Inbound Rule Wizard Name Dialog

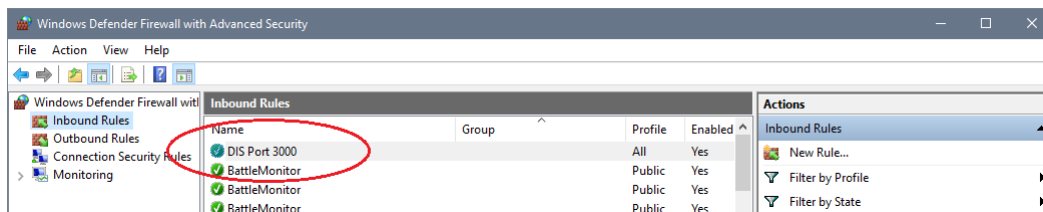


Figure 88. Windows Defender Firewall Inbound Rules view with new DIS Port 3000 rule

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. SOURCE CODE

A. INTRODUCTION

All source code written for this thesis can be found under <https://gitlab.nps.edu/Savage/SavageTheses/-/tree/master/BrennenstuhlTobias>.

B. PLAYERPLAINTEXT.JAVA

```
package PduStreamTools;

/**
 * Copyright (c) 2008-2019, MOVES Institute, Naval Postgraduate School.
 * All
 * rights reserved. This work is licensed under the BSD open source
 * license,
 * available at https://www.movesinstitute.org/licenses/bsd.html
 *
 */
import static PduStreamTools.RecorderBase64.COMMENT_MARKER;
import static PduStreamTools.RecorderBase64.START_COMMENT_MARKER;
import static PduStreamTools.RecorderBase64.STOP_COMMENT_MARKER;
import com.google.common.primitives.Longs;

import java.io.*;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.nio.ByteBuffer;
import java.nio.IntBuffer;
import java.nio.file.Path;
import java.util.Arrays;
import java.util.regex.Pattern;

/**
 * PlayerPlainText.java created on Mar 2, 2020 MOVES Institute Naval
 * Postgraduate
 * School, Monterey, CA, USA www.nps.edu
 *
 * @author Mike Bailey, jmbailey@nps.edu
 * @author Tobias Brennenstuhl, tobias.brennenstuhl.gy@nps.edu
 */
public class PlayerPlainText {

    private Path disLogDirectory;
    private String ip;
    private int port;
    private Thread thrd;

    public PlayerPlainText(String ip, int port, Path disLogDirectory)
        throws IOException {
```

```

        this.disLogDirectory = disLogDirectory;
        this.ip = ip;
        this.port = port;

        thrd = new Thread(() -> begin());
        thrd.setPriority(Thread.NORM_PRIORITY);
        thrd.setName("PlayerThread");
        thrd.setDaemon(true);
        thrd.start();
    }

    private Integer scenarioPduCount = null;
    private boolean showPduCountsOneTime = false;
    private int pduCount = 0;
    private DatagramSocket dsock;
    private BufferedReader brdr;
    private Long startNanoTime = null;
    private boolean paused = false;

    @SuppressWarnings("StatementWithEmptyBody")
    public void begin() {
        try {
            System.out.println("Replaying DIS logs.");
            InetAddress addr = InetAddress.getByName(ip);

            FilenameFilter filter = (dir, name) ->
name.endsWith(RecorderBase64.DISLOG_FILE_TAIL) && !name.startsWith(".");

            File [] fs = disLogDirectory.toFile().listFiles(filter);
            if (fs == null) {
                fs = new File [0];
            }

            Arrays.sort(fs, (f1, f2) -> {
                return f1.getName().compareTo(f2.getName());
            });

            dsock = new DatagramSocket();

            // ----- Begin Variables for Position
            Interpolator
                CreateX3dInterpolators x3dInterpolators = new
CreateX3dInterpolators();
            CreateX3dLineSet x3dLineSet = new CreateX3dLineSet();
            byte [] globalByteBufferForX3dInterPolators = null;

            // ----- End Variables for Position
            Interpolator
                for (File f : fs) {
                    System.out.println("Replaying " + f.getAbsolutePath());
                    brdr = new BufferedReader(new FileReader(f), 1024 * 200);
                }
            // 200kb buffer

            String line = brdr.readLine();
            while (line != null && !Thread.interrupted()) {

```

```

while (paused) {
    sleep(5001); // half sec
}
if (line.length() <= 0); // blank lines ok
else if (line.startsWith(COMMENT_MARKER)) {
    if (handleComment(line, f)) {
        // here if we got an end comment
        break; // out of read loop
    }
} else {

//Pattern splitting needed for playback of
unencoded streams

String REGEX = "\\],\\[\";
Pattern pattern = Pattern.compile(REGEX);
String [] sa;
sa = pattern.split(line);
//Add the "]" to the end of sa [0]. It was taken
off by the split

sa [0] = sa [0].concat("]");
//Add the "]" to the end of sa [0]. It was taken
off by the split

sa [1] = "[".concat(sa [1]);
if (sa.length != 2) {
    System.err.println("Error: parsing error.
Line follows.");

    System.err.println(line);
    byebye();
}

if (startNanoTime == null) {
    startNanoTime = System.nanoTime();
}

byte [] pduTimeBytes = null;

String [] splitString;

//Split first String into multiple Strings
cotaining integers

REGEX = ",";
pattern = Pattern.compile(REGEX);

sa [0] = sa [0].substring(1, sa [0].length() - 1);

splitString = pattern.split(sa [0]);

//Define an array to store the in values from the
string and initalize it to a value drifferent from NULL
int [] arr = new int [splitString.length];

String tempString = "";

//Test
for (int x = 0; x < splitString.length; x++) {

```

```

        tempString = splitString [x].trim();

        int tempInt = Integer.parseInt(tempString);
        arr [x] = tempInt;

    }

    // Credit:  https://stackoverflow.com/questions/
1086054/how-to-convert-int-to-byte
        ByteBuffer          byteBuffer1          =
ByteBuffer.allocate(arr.length * 4);
        IntBuffer intBuffer = byteBuffer1.asIntBuffer();
        intBuffer.put(arr);

        pduTimeBytes = byteBuffer1.array();
        long          pduTimeInterval          =
Longs.fromByteArray(pduTimeBytes);

        // This is a relative number in nanoseconds of the
time of packet reception minus first packet reception for scenario.
        long          targetSimTime          =      startNanoTime      +
pduTimeInterval; // when we should send the packet
        long now = System.nanoTime();
        long          sleepTime          =      targetSimTime      -      now;
//System.nanoTime(); // the difference between then and now

        if (sleepTime > 20000000) { // 20 ms //
            System.out.println("sim interval = " +
pduTimeInterval + ", sleeping for " + sleepTime / 1000000L + " ms");
            sleep(sleepTime / 1000000L, (int) (sleepTime
% 1000000L));
        }

        //Handle the second String
        //Split second String into multiple Strings
cotaining integers

        REGEX = ",";
        pattern = Pattern.compile(REGEX);

        sa [1] = sa [1].substring(1, sa [1].length() - 1);

        splitString = pattern.split(sa [1]);

        //Define an array to store the in values from the
string and initalize it to a value drifferent from NULL
        arr = new int [splitString.length];

        //trim spaces, if any
        tempString = "";

        //Test
        for (int x = 0; x < splitString.length; x++) {

            tempString = splitString [x].trim();

```



```

        int tempInt = Integer.parseInt(tempString);
        arr [x] = tempInt;

        //System.out.println(tempInt);
    }

    // Credit:  https://stackoverflow.com/questions/
1086054/how-to-convert-int-to-byte
    ByteBuffer                byteBuffer2                =
ByteBuffer.allocate(arr.length * 4);
    intBuffer = byteBuffer2.asIntBuffer();
    intBuffer.put(arr);

    byte [] buffer = byteBuffer2.array();

    //When the byteBuffer stores the array of Integers
into the byte array it store a 7 as 0 0 0 7.
    //Therefore a shortBuffer is created where only
every forth value is stored.
    //it must be done with modulo instead of testing
for "0" because a "0" could be there as value and not as padding
    byte []    bufferShort    =    new    byte
[byteBuffer2.array().length / 4];
    int bufferShortCounter = 0;

    for (int i = 1; i < byteBuffer2.array().length;
i++) {

        if (((i + 1) % 4) == 0) {

            bufferShort [bufferShortCounter] = buffer
[i];

            bufferShortCounter++;

        }

    }

    DatagramPacket    packet    =    new
DatagramPacket(bufferShort, bufferShort.length, addr, port);

    globalByteBufferForX3dInterPolators = new byte
[byteBuffer2.array().length / 4];
    globalByteBufferForX3dInterPolators    =
bufferShort.clone();

    dsock.send(packet);

x3dInterpolators.addPointsToMap(globalByteBufferForX3dInterPolators);

x3dLineSet.addPointsToMap(globalByteBufferForX3dInterPolators);
    pduCount++;
    if (scenarioPduCount != null) {

```

```

        scenarioPduCount++;
    }

    if (showPduCountsOneTime || pduCount % 5 == 0) {
        showCounts();
    }
}

line = brdr.readLine();
}
brdr.close();

x3dInterpolators.makeX3dInterpolator();
x3dLineSet.makeX3dLineSet();

}
} catch (Exception ex) {
    System.err.println("Exception reading/writing pdus: " +
ex.getClass().getSimpleName() + ": " + ex.getMessage());
    thrd = null;
    closer();
}
}

private void sleep(long ms) {
    try {
        Thread.sleep(ms);
    } catch (InterruptedException ex) {
    }
}

private void showCounts() {
    if (scenarioPduCount != null) {
        System.out.print(pduCount + " " + ++scenarioPduCount + "\r");
    } else {
        System.out.print(pduCount + "\r");
    }
    showPduCountsOneTime = false;
}

private void goodbye() throws IOException {
    System.out.println("Replay stopped.");
    closer();
    throw new IOException("Dis Replayer parsing error");
}

private void closer() {
    try {
        if (dsock != null) {
            dsock.close();
            dsock = null;
        }
        if (brdr != null) {
            brdr.close();
            brdr = null;
        }
    }
}

```

```

    }
    } catch (Exception ioex) {
        System.err.println("IOException closing reader in Player");
    }
}

private boolean handleComment(String s, File f) //true if we're done
{
    boolean ret = false;
    if (s.startsWith(START_COMMENT_MARKER)) {
        //System.out.println();
        s = s.substring(START_COMMENT_MARKER.length());
        System.out.println(s + " ");
        showPduCountsOneTime = true; // get the first one in there
    } else if (s.startsWith(STOP_COMMENT_MARKER)) {
        showCounts();
        System.out.println();
        System.out.println("End of replay from " + f.getName());

        scenarioPduCount = 0;
        startNanoTime = null;
        ret = true;
    }
    return ret;
}

public void startResume() {
    paused = false;
}

public void stopPause() {
    paused = false;
}

public void end() {
    thrd.interrupt();
    closer();
}

public static void main(String [] args) {
    try {
        //new Player("230.0.0.0," 3000, new
        File("./pdulog").toPath()).startResume();
        new PlayerPlainText("239.1.2.3," 3000, new File("/Users/mike/
        NetbeansProjects/open-dis7-java/examples/pdulog").toPath());
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

private static void sleep(long ms, int ns) {
    // @formatter:off
    try {

```

```

        Thread.sleep(ms, ns);
    } catch (InterruptedException ex) {
        System.out.println("InterruptedException");
    }
    // @formatter:on
}
}

```

C. RECORDERPLAINTEXT.JAVA

```

package PduStreamTools;

import com.google.common.primitives.Longs;

import edu.nps.moves.dis7.Pdu;
import edu.nps.moves.dis7.enumerations.DISPDUType;
import edu.nps.moves.dis7.util.DisNetworking;
import edu.nps.moves.dis7.util.DisNetworking.BuffAndLength;
import edu.nps.moves.dis7.util.PduFactory;
import edu.nps.moves.dis7.util.playerrecorder.PduReceiver;
import org.apache.commons.io.FilenameUtils;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.nio.file.Path;
import java.util.Arrays;

/**
 * RecorderPlainText.java created on Mar 2, 2020 MOVES Institute Naval
 * Postgraduate
 * School, Monterey, CA, USA www.nps.edu
 *
 * @author Mike Bailey, jmbailey@nps.edu
 * @author Tobias Brennenstuhl, tobias.brennenstuhl.gy@nps.edu
 */

public class RecorderPlainText implements PduReceiver
{
    static String DEFAULT_OUTDIR = "./pdulog";
    static String DEFAULT_FILEPREFIX = "Pdusave";
    static String DEFAULT_MCAST = "239.1.2.3";
    static int    DEFAULT_PORT = 3000;

    static String DISLOG_FILE_TAIL = ".dislog";

    public static String COMMENT_MARKER = "!";
    static String START_COMMENT_MARKER = COMMENT_MARKER + "Begin" +
COMMENT_MARKER;
    static String STOP_COMMENT_MARKER = COMMENT_MARKER + "End" +
COMMENT_MARKER;

    private BufferedWriter bwr;
    private File logFile;

```

```

private DisNetworking disnet;
private Thread receiverThrd;

public RecorderPlainText() throws IOException
{
    this(DEFAULT_OUTDIR,DEFAULT_MCAST,DEFAULT_PORT);
}

public RecorderPlainText(String outputDir, String mcastaddr, int port)
throws IOException
{
    logFile          =          makeFile(new          File(outputDir).toPath(),
DEFAULT_FILEPREFIX+DISLOG_FILE_TAIL );
    bwr = new BufferedWriter(new FileWriter(logFile));

    disnet = new DisNetworking(port, mcastaddr);
    // Start a thread to receive and record pdus

    receiverThrd = new Thread(()->{
        int count = 1;
        while(!Thread.interrupted()){
            try {
                BuffAndLength blen = disnet.receiveRawPdu();
                //System.out.println(""+ count++ + " Got pdu from
DisNetworking");
                receivePdu(blen.buff,blen.length);
            }
            catch(IOException ex) {
                // this is the normal exception if you do disnet.stop()
System.err.println("Exception receiving Pdu:
"+ex.getMessage());
            }
        }
    });
    receiverThrd.setPriority(Thread.NORM_PRIORITY);
    receiverThrd.setDaemon(true);
    receiverThrd.start();
}

public void startResume()
{
    dosave = true;
}

public void stopPause()
{
    dosave = false;
}

public String getOutputFile()
{
    if(logFile != null)
        return logFile.getAbsolutePath();
    return null;
}

```

```

public void end()
{
    disnet.stop();
    receiverThrd.interrupt();

    try {
        writeFooter();
        bwr.flush();
        bwr.close();
        System.out.println();
        System.out.println("Recorder log file closed");
    }
    catch (IOException ex) {
        System.out.println("IOException closing file: " + ex.getMessage());
    }
}

Long startNanoTime = null;
StringBuilder sb = new StringBuilder();
// Base64.Encoder encdr = Base64.getEncoder();
int pduCount = 0;
boolean headerWritten = false;
boolean dosave = true;

@Override
public void receivePdu(byte [] buff, int len)
{
    if(!dosave)
        return;

    long packetRcvNanoTime = System.nanoTime();
    if (startNanoTime == null)
        startNanoTime = packetRcvNanoTime;

    byte [] timeAr = Longs.toByteArray(packetRcvNanoTime - startNanoTime);
    //System.out.println("wrote      time      "+(packetRcvNanoTime -
startNanoTime));

    sb.setLength(0);
    //sb.append(encdr.encodeToString(timeAr));
    sb.append(Arrays.toString(timeAr));
    sb.append(',');
    byte [] buff sized = Arrays.copyOf(buff, len);
    //sb.append(encdr.encodeToString(buff sized));
    sb.append(Arrays.toString(buff sized));
    try {
        if (!headerWritten) {
            writeHeader();
            headerWritten = true;
        }
        //Added a REGEX to strip all spaces from the string before writing
to file
        bwr.write(sb.toString().replaceAll("\\s",""));
        bwr.newLine();

```

```

    }
    catch (IOException ex) {
        System.err.println("Fatal exception writing DIS log file in
Recorder.start()");
        throw new RuntimeException(ex);
    }
    System.out.print(++pduCount + "\r");

    //bwr.flush();
    sb.setLength(0);
}

public String getLogFile()
{
    return logFile.getAbsolutePath();
}

private void writeHeader() throws IOException
{
    String template = "Beginning of DIS capture file, %s. [PDU Header],[PDU
Stream]";
    String startComment = String.format(template, logFile.getName());
    bwr.write(START_COMMENT_MARKER + startComment);
    bwr.newLine();
}

private void writeFooter() throws IOException
{
    String template = "End of DIS capture file, %s.";
    String endComment = String.format(template, logFile.getName());
    bwr.write(STOP_COMMENT_MARKER + endComment);
    bwr.newLine();
}

private File makeFile(Path outputDir, String filename) throws
IOException
{
    String bname = FilenameUtils.getBaseName(filename);
    String ext = FilenameUtils.getExtension(filename);

    Integer count = null;
    File f;
    boolean fileExists;
    outputDir.toFile().mkdirs();
    do {
        String fn = bname + (count == null ? "" : count) + "." + ext;
        f = new File(outputDir.toFile(), fn);
        fileExists = f.exists();
        if (count == null)
            count = 1;
        else
            count++;
    } while (fileExists);
    if (!f.createNewFile()) {

```

```

        System.out.println("Cannot create dis log file at " +
f.getAbsolutePath());
        throw new RuntimeException("File creation error");
    }
    return f;
}

/* Example test usage */
public static void main(String [] args)
{
    PduFactory factory = new PduFactory(); //default appid, country, etc.
    DisNetworking disnet = new DisNetworking(); // default ip and port

    Path path = new File("./pdulog").toPath();
    String filename = "Pdusave";

    RecorderPlainText recorder;
    try{recorder = new RecorderPlainText();} catch(IOException ex) {
        System.err.println("Exception creating recorder:
"+ex.getLocalizedMessage());
        return;
    }

    DISPDUType all [] = DISPDUType.values();
    Arrays.stream(all).forEach(typ-> {
        if(typ != DISPDUType.OTHER) {
            try {
                Pdu pdu = factory.createPdu(typ);
                disnet.sendPdu(pdu);
                sleep(100);
            }
            catch(Exception ex) {
                System.err.println("Exception sending Pdu:
"+ex.getLocalizedMessage());
            }
        }
    });
    sleep(2000);

    recorder.end();
}

private static void sleep(long ms)
{
    try{Thread.sleep(ms);}catch(InterruptedException ex) {}
}
}

```

D. SLIDINGWINDOW.JAVA

```

package PduStreamTools;

import static java.lang.Math.pow;
import static java.lang.Math.sqrt;
import java.util.ArrayList;

```



```

import java.util.LinkedHashMap;
import java.util.List;
import java.util.Set;
import java.util.TreeMap;

/**
 *
 * @author Tobias Brennenstuhl @ NPS
 */
public class SlidingWindowCompression {

    private LinkedHashMap<Double, Coordinates> localMap;

    public SlidingWindowCompression(LinkedHashMap<Double, Coordinates>
localHashMap) {

        this.localMap = new LinkedHashMap<>();
        Set<Double> keys = localHashMap.keySet();
        for (Double k : keys) {
            localMap.put(k, localHashMap.get(k));
        }
    }

    public TreeMap<Double, Coordinates> doSlidingWindow() {

        System.out.println("DISTools.Regression.doRegression()");
        //Check whether points could be deleted to compress the stream
        //https://www.crashkurs-statistik.de/einfache-lineare-
regression/
        TreeMap<Double, Coordinates> streamMap = new TreeMap<>();
        //Copy LinkedHashMap into TreeMap to be able to pull the first
element.
        streamMap.putAll(localMap);
        TreeMap<Double, Coordinates> returnMap = new TreeMap<>();
        //TreeMap of slidingWindows will store all of the points that are
currently processed
        //use .pullFirstEntry() to get rid of the points at the beginning.
        TreeMap<Double, Coordinates> slidingWindow = new TreeMap<>();

        while (streamMap.size() > 0) {
            slidingWindow.put(streamMap.firstEntry().getKey(),
streamMap.get(streamMap.firstEntry().getKey()));
            streamMap.pollFirstEntry();

            //Calculate the mean and SD
            Set<Double> slidingWindowKeys = slidingWindow.keySet();

            if (slidingWindow.size() >= 3) {

                List<Double> tList = new ArrayList<>();
                List<Double> xList = new ArrayList<>();
                List<Double> yList = new ArrayList<>();
                List<Double> zList = new ArrayList<>();
                List<Double> phiList = new ArrayList<>();
                List<Double> psiList = new ArrayList<>();
            }
        }
    }
}

```

```

List<Double> thetaList = new ArrayList<>();

Double [] k = new Double [slidingWindowKeys.size()];
slidingWindowKeys.toArray(k);

for (int i = 0; i < slidingWindow.size(); i++) {

    tList.add(i, k [i]);

    phiList.add(i, slidingWindow.get(k [i]).getPhi());
    psiList.add(i, slidingWindow.get(k [i]).getPsi());
    thetaList.add(i, slidingWindow.get(k
[i]).getTheta());

    xList.add(i, slidingWindow.get(k [i]).getX());
    yList.add(i, slidingWindow.get(k [i]).getY());
    zList.add(i, slidingWindow.get(k [i]).getZ());

}

//Calculate Area of Triangle
//Credit:      http://www.ambrsoft.com/TrigoCalc/Line3D/
LineColinear.htm
    for (int i = 0; i < slidingWindow.size(); i++) {

        double a = sqrt(pow(xList.get(1) - xList.get(0), 2) +
pow(yList.get(1) - yList.get(0), 2) + pow(zList.get(1) - zList.get(0),
2));

        double b = sqrt(pow(xList.get(i) - xList.get(0), 2) +
pow(yList.get(i) - yList.get(0), 2) + pow(zList.get(i) - zList.get(0),
2));

        double c = sqrt(pow(xList.get(i) - xList.get(1), 2) +
pow(yList.get(i) - yList.get(1), 2) + pow(zList.get(i) - zList.get(1),
2));

        double s = (a + b + c) / 2;
        double areaA = sqrt(s * (s - a) * (s - b) * (s - c));

        if ((areaA >= 0.1) || (tList.get(i) - tList.get(0) >=
4.0)) {

            //grab the first and the last point from the
sliding window and push it to the returnMap
            Coordinates firstPoint = new Coordinates();
            firstPoint.setX(xList.get(0));
            firstPoint.setY(yList.get(0));
            firstPoint.setZ(zList.get(0));
            firstPoint.setPhi(phiList.get(0));
            firstPoint.setPsi(psiList.get(0));
            firstPoint.setTheta(thetaList.get(0));

            Coordinates lastPoint = new
Coordinates(xList.get(i), yList.get(i), zList.get(i), phiList.get(i),
psiList.get(i), thetaList.get(i));

            returnMap.put(tList.get(0), firstPoint);
            returnMap.put(tList.get(i), lastPoint);

```

```

        slidingWindow.clear();

        tList.clear();
        xList.clear();
        yList.clear();
        zList.clear();
        phiList.clear();
        psiList.clear();
        thetaList.clear();

        break;
    }

    if ((areaA <= 0.1) && (tList.get(i) - tList.get(0) <=
4.0) && streamMap.size() == 0) {

        //System.out.println("StreamMap empty. All points
left will be added. Break");
        //grab the first and the last point from the siding
window and push it to the returnMap
        for (int j = 0; j < slidingWindow.size(); j++) {
            Coordinates leftPoints = new
Coordinates(xList.get(j), yList.get(j), zList.get(j), phiList.get(j),
psiList.get(j), thetaList.get(j));
            returnMap.put(tList.get(j), leftPoints);
        }

        break;
    }
    //System.out.println("Area of Triangle: " + areaA);
}

}

}

return returnMap;

}

;

}

```

E. COORDINATES.JAVA

```

package PduStreamTools;

/**
 * This class is a holder for coordinates and angles of ESPDUs to store
them in
 * HashMaps
 *
 * @author Tobias Brennenstuhl @ NPS
 */
public class Coordinates {

```

```

private double x;
private double y;
private double z;
private double phi;
private double psi;
private double theta;

    public Coordinates(double x, double y, double z, double phi, double
psi, double theta) {
        this.setX(x);
        this.setY(y);
        this.setZ(z);
        this.setPhi(phi);
        this.setPsi(psi);
        this.setTheta(theta);
    }

    public Coordinates() {
        this.setX(0.0);
        this.setY(0.0);
        this.setZ(0.0);
        this.setPhi(0.0);
        this.setPsi(0.0);
        this.setTheta(0.0);
    }

    public double getPhi() {
        return phi;
    }

    public void setPhi(double phi) {
        this.phi = phi;
    }

    public double getPsi() {
        return psi;
    }

    public void setPsi(double psi) {
        this.psi = psi;
    }

    public double getTheta() {
        return theta;
    }

    public void setTheta(double theta) {
        this.theta = theta;
    }

    public double getX() {
        return x;
    }

```

```

    public void setX(double x) {
        this.x = x;
    }

    public double getY() {
        return y;
    }

    public void setY(double y) {
        this.y = y;
    }

    public double getZ() {
        return z;
    }

    public void setZ(double z) {
        this.z = z;
    }
}

```

F. CREATEX3DINTERPOLATORS.JAVA

```

package PduStreamTools;

import edu.nps.moves.dis7.EntityStatePdu;
import edu.nps.moves.dis7.Pdu;
import edu.nps.moves.dis7.enumerations.DISPDUType;
import edu.nps.moves.dis7.util.PduFactory;
import java.nio.ByteBuffer;
import java.text.NumberFormat;
import java.util.LinkedHashMap;
import java.util.Locale;
import java.util.Set;
import java.util.TreeMap;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author Tobias Brennenstuhl @ NPS
 */
public class CreateX3dInterpolators {

    private byte [] bufferShort;

    // ----- Begin Variables for Position Interpolator
    private Boolean firstTimeStamp = true;
    private int firstLocalTimeStamp = 0;

    private double firstLocalX = 0;
    private double firstLocalY = 0;
    private double firstLocalZ = 0;

```

```

private double firstLocalPhi = 0;
private double firstLocalPsi = 0;
private double firstLocalTheta = 0;

private   LinkedHashMap<Double,   Coordinates>   testMap   =   new
LinkedHashMap<>();

//Setting up a NumberFormatter for limiting the decimal count to 3
private   NumberFormat   coordinateNumberFormat   =
NumberFormat.getInstance(new Locale("en," "US"));

// ----- End Variables for Position Interpolator
public CreateX3dInterpolators() {

    //3 significant digits equals milimeter position accuracy and
    0.001 radians = 0.0572963266634555 degrees
    coordinateNumberFormat.setMaximumFractionDigits(3);

}

public void addPointsToMap(byte [] localBufferShort) {

    this.bufferShort = localBufferShort.clone();

    if (bufferShort [2] == 1) {

        //PDU Factory
        PduFactory pduFactory = new PduFactory();
        Pdu localPdu = null;

        localPdu = pduFactory.createPdu(bufferShort);

        // TODO figure out how to do this! makeEntityStatePDU
        EntityStatePdu localEspdu = pduFactory.makeEntityStatePdu();
        //Put all the data we need into the localEspdu
        ByteBuffer espduBuffer = ByteBuffer.wrap(bufferShort);
        try {
            localEspdu.unmarshal(espduBuffer);
        } catch (Exception ex) {

Logger.getLogger(CreateX3dInterpolators.class.getName()).log(Level.SEVERE,
null, ex);
        }

        double localTimeStamp = 0;
        double localX = 0;
        double localY = 0;
        double localZ = 0;

        double localPhi = 0;
        double localPsi = 0;
        double localTheta = 0;

        //Store the first timestamp to subtract it from all others
        //Same with X,Y,Z to create a local coordiante system

```

```

        if (firstTimeStamp) {

            firstLocalTimeStamp = localPdu.getTimestamp();
            localTimeStamp = localPdu.getTimestamp();
            firstLocalX = localEspdu.getEntityLocation().getX();
            firstLocalY = localEspdu.getEntityLocation().getZ();
            firstLocalZ = -1 * localEspdu.getEntityLocation().getY();

            firstTimeStamp = false;
        }

        localTimeStamp = localPdu.getTimestamp();
        localX = localEspdu.getEntityLocation().getX();
        localY = localEspdu.getEntityLocation().getZ();
        localZ = -1 * localEspdu.getEntityLocation().getY();
        localPhi = localEspdu.getEntityOrientation().getPhi();
        localPsi = localEspdu.getEntityOrientation().getPsi();
        localTheta = localEspdu.getEntityOrientation().getTheta();

        localTimeStamp = localTimeStamp - firstLocalTimeStamp;
        localX = localX - firstLocalX;
        localY = localY - firstLocalY;
        localZ = localZ - firstLocalZ;

        //Divide TimeStamp by 1,300,000 to get something close to a
second per Unit.
        //According to the DIS standard one tick is 3600/(2^31) seconds
~ 1.6764 µs
        //1,100,000 was derived from a stream that is 83 seconds long.
The number was adjusted to get a timesensor with 83 seconds
        //ToDo find the real conversion between TimeStampDelta and
seconds
        localTimeStamp = localTimeStamp / 1100000;

        //Only add to stream if it is an ESPDU
        //ToDo: Add support for multiple Entities
        if ((localPdu.getPduType() != null) && (localPdu.getPduType()
== DISPDUTYPE.ENTITY_STATE)) {

            testMap.put((double) localTimeStamp, new
Coordinates(localX, localY, localZ, localPhi, localPsi, localTheta));

        }

    }

    public void makeX3dInterpolator() {

        //Compression of the testMap.
        //Remove all collinear points.
        SlidingWindowCompression slidingWindowCompression = new
SlidingWindowCompression(testMap);

        TreeMap<Double, Coordinates> returnMap = new TreeMap<>();

```

```

        //To turn of the compression just comment the next line out and
the very next in.
        returnMap = slidingWindowCompression.doSlidingWindow();
        //returnMap.putAll(testMap);

        //Writing all values from the KeyMap to a proper Position
Interpolator String
        System.out.println("Writing Position and Rotation Interpolator");
        Set<Double> keys = returnMap.keySet();
        //Set<Double> keys =
tempKeyKeyValueSetPositionInterPolator.keySet();
        String positionKey = "key = `";
        String positionKeyValue = "keyValue = `";
        String positionInterpolatorToCopy = "<PositionInterpolator
DEF='EntityPosition' ";

        String orientationKeyX = "key = `";
        String orientationKeyValueX = "keyValue = `";
        String orientationInterpolatorToCopyX = "<OrientationInterpolator
DEF='EntityOrientationX' ";

        String orientationKeyY = "key = `";
        String orientationKeyValueY = "keyValue = `";
        String orientationInterpolatorToCopyY = "<OrientationInterpolator
DEF='EntityOrientationY' ";

        String orientationKeyZ = "key = `";
        String orientationKeyValueZ = "keyValue = `";
        String orientationInterpolatorToCopyZ = "<OrientationInterpolator
DEF='EntityOrientationZ' ";

        //Find highest time to do the normalization
        double lastTimeStamp = 0;

        for (Double k : keys) {

            if (k > lastTimeStamp) {

                lastTimeStamp = k;

            }

        }

        //Normalize all times in the set
        var keyKeyValueSetPositionInterpolator = new
LinkedHashMap<Double, String>();

        var keyKeyValueSetOrientationInterpolatorX = new
LinkedHashMap<Double, String>();
        var keyKeyValueSetOrientationInterpolatorY = new
LinkedHashMap<Double, String>();
        var keyKeyValueSetOrientationInterpolatorZ = new
LinkedHashMap<Double, String>();

```



```

for (Double k : keys) {

    String localCoordinateString;
    String localOrientationStringX;
    String localOrientationStringY;
    String localOrientationStringZ;

    double tempX = returnMap.get(k).getX();
    double tempY = returnMap.get(k).getY();
    double tempZ = returnMap.get(k).getZ();

    double tempPhi = returnMap.get(k).getPhi() / 6.28;
    double tempPsi = returnMap.get(k).getPsi() / 6.28;
    double tempTheta = returnMap.get(k).getTheta() / 6.28;

    localCoordinateString      =      "      "      +
coordinateNumberFormat.format(tempX)      +      "      "      +
coordinateNumberFormat.format(tempY)      +      "      "      +
coordinateNumberFormat.format(tempZ);
    localOrientationStringX    =      "      1      0      0      "      +
coordinateNumberFormat.format(tempPhi);
    localOrientationStringY    =      "      0      1      0      "      +
coordinateNumberFormat.format(tempTheta);
    localOrientationStringZ    =      "      0      0      1      "      +
coordinateNumberFormat.format(tempPsi);

    keyKeyValueSetPositionInterpolator.put(k / lastTimeStamp,
localCoordinateString);
    keyKeyValueSetOrientationInterpolatorX.put(k / lastTimeStamp,
localOrientationStringX);
    keyKeyValueSetOrientationInterpolatorY.put(k / lastTimeStamp,
localOrientationStringY);
    keyKeyValueSetOrientationInterpolatorZ.put(k / lastTimeStamp,
localOrientationStringZ);

}

keys = keyKeyValueSetPositionInterpolator.keySet();

//Setting up the timeSensor
//Only one timeSensor for both interpolators is needed
String timeSensor      =      "<TimeSensor      DEF='PduStreamClock'
cycleInterval='";

timeSensor += lastTimeStamp;

timeSensor += "` loop = 'true'/>";

//Printing the timeSensor to the console
System.out.println(timeSensor);

//Setting up PositionInterpolator and OrientationInterpolator
for (Double k : keys) {
    //System.out.println("Time: " + k + " Position (x,y,z) " +
keyKeyValueSetPositionInterpolator.get(k));

```

```

        //PositionInterpolator
        positionKey += coordinateNumberFormat.format(k) + " ";
        positionKeyValue += keyKeyValueSetPositionInterpolator.get(k)
+ " ";

        //OrientationInterpolator for X (phi)
        orientationKeyX += coordinateNumberFormat.format(k) + " ";
        orientationKeyValueX                                     +=
keyKeyValueSetOrientationInterpolatorX.get(k) + " ";

        //OrientationInterpolator for Y (theta)
        orientationKeyY += coordinateNumberFormat.format(k) + " ";
        orientationKeyValueY                                     +=
keyKeyValueSetOrientationInterpolatorY.get(k) + " ";

        //OrientationInterpolator for Z (psi)
        orientationKeyZ += coordinateNumberFormat.format(k) + " ";
        orientationKeyValueZ                                     +=
keyKeyValueSetOrientationInterpolatorZ.get(k) + " ";

    }
    positionKey += "\n";
    positionKeyValue += "\n";

    orientationKeyX += "\n";
    orientationKeyValueX += "\n";

    orientationKeyY += "\n";
    orientationKeyValueY += "\n";

    orientationKeyZ += "\n";
    orientationKeyValueZ += "\n";

    //PositionInterpolator
    positionInterpolatorToCopy += positionKey + "\n";
    positionInterpolatorToCopy += positionKeyValue;
    positionInterpolatorToCopy += ">";

    //PositionInterpolator for X
    orientationInterpolatorToCopyX += orientationKeyX + "\n";
    orientationInterpolatorToCopyX += orientationKeyValueX;
    orientationInterpolatorToCopyX += ">";

    //PositionInterpolator for Y
    orientationInterpolatorToCopyY += orientationKeyY + "\n";
    orientationInterpolatorToCopyY += orientationKeyValueY;
    orientationInterpolatorToCopyY += ">";

    //PositionInterpolator for Z
    orientationInterpolatorToCopyZ += orientationKeyY + "\n";
    orientationInterpolatorToCopyZ += orientationKeyValueZ;
    orientationInterpolatorToCopyZ += ">";

    //Printing PositionInterpolator to the console

```

```

System.out.println(positionInterpolatorToCopy);

//First Rotation must be around z axis by psi
//Printing OrientationInterpolator for X to the console
System.out.println(orientationInterpolatorToCopyZ);

//Second Rotation must be around resulting y (y') axis by theta
//Printing OrientationInterpolator for Y to the console
System.out.println(orientationInterpolatorToCopyY);

//last rotation must be around resulting x (x') axis by phi
//Printing OrientationInterpolator for Z to the console
System.out.println(orientationInterpolatorToCopyX);
}
}

```

G. EXAMPLE OF AUTOGENERATED X3D NODES

```

<TimeSensor DEF='PduStreamClock' cycleInterval='83.02751636363637' loop
= 'true'/>
<PositionInterpolator DEF='EntityPosition' key = '0 0.049 0.05 0.059 0.06
0.098 0.098 0.157 0.157 0.214 0.23 0.291 0.292 0.307 0.308 0.315 0.315
0.369 0.403 0.461 0.487 0.554 0.564 0.582 0.583 0.623 0.623 0.678 0.678
0.752 0.77 0.829 0.829 0.888 0.888 0.949 0.949 0.993 1 '
keyValue = ' 0 0 0 -1.569 -2.435 -1.337 -1.663 -2.653 -1.342 -2.354 -
5.179 -0.429 -2.374 -5.43 -0.218 -1.825 -19.38 15.562 -1.824 -19.447
15.633 -1.63 -50.664 48.405 -1.631 -50.734 48.476 -1.971 -75.294 73.045
-1.994 -76.957 74.71 -2.087 -83.614 81.369 -2.086 -83.725 81.486 -
1.339 -84.888 84.524 -1.224 -84.866 84.785 0.009 -83.851 86.765 0.06
-83.785 86.823 17.157 -62.331 106.653 30.427 -46.434 122.826 52.939 -
20.165 150.983 58.56 -13.605 158.013 63.708 -7.598 164.452 64.514 -
6.658 165.46 66.272 -3.281 166.287 66.351 -3.009 166.201 66.45 12.101
150.812 66.45 12.17 150.742 66.77 41.41 121.281 66.773 41.483 121.211
67.942 76.175 88.195 68.008 78.127 86.338 68.228 84.669 80.112 68.23
84.74 80.042 56.369 72.819 63.221 56.32 72.76 63.162 32.784 44.414
34.636 32.733 44.354 34.572 15.74 24.523 13.318 12.973 21.295 9.859 '
/>

<OrientationInterpolator DEF='EntityOrientationZ' key = '0 0.049 0.05
0.059 0.06 0.098 0.098 0.157 0.157 0.214 0.23 0.291 0.292 0.307 0.308
0.315 0.315 0.369 0.403 0.461 0.487 0.554 0.564 0.582 0.583 0.623 0.623
0.678 0.678 0.752 0.77 0.829 0.829 0.888 0.888 0.949 0.949 0.993 1 '
keyValue = ' 0 0 1 0.347 0 0 1 0.479 0 0 1 -0.5 0 0 1 -0.272 0 0 1 -
0.26 0 0 1 -0.247 0 0 1 -0.248 0 0 1 -0.25 0 0 1 -0.252 0 0 1 -0.252
0 0 1 -0.252 0 0 1 -0.252 0 0 1 -0.247 0 0 1 -0.186 0 0 1 -0.182 0
0 1 -0.136 0 0 1 -0.134 0 0 1 -0.141 0 0 1 -0.143 0 0 1 -0.143 0 0
1 -0.143 0 0 1 -0.143 0 0 1 -0.142 0 0 1 0.116 0 0 1 0.143 0 0 1
0.251 0 0 1 0.249 0 0 1 0.244 0 0 1 0.244 0 0 1 0.244 0 0 1 0.244 0
0 1 0.245 0 0 1 0.247 0 0 1 0.363 0 0 1 0.361 0 0 1 0.358 0 0 1 0.358
0 0 1 0.358 0 0 1 0.358 ' />

<OrientationInterpolator DEF='EntityOrientationY' key = '0 0.049 0.05
0.059 0.06 0.098 0.098 0.157 0.157 0.214 0.23 0.291 0.292 0.307 0.308

```

```

0.315 0.315 0.369 0.403 0.461 0.487 0.554 0.564 0.582 0.583 0.623 0.623
0.678 0.678 0.752 0.77 0.829 0.829 0.888 0.888 0.949 0.949 0.993 1 `
keyValue = ` 0 1 0 0.081 0 1 0 0.183 0 1 0 0.187 0 1 0 0.145 0 1 0
0.134 0 1 0 0.118 0 1 0 0.119 0 1 0 0.122 0 1 0 0.125 0 1 0 0.125 0
1 0 0.125 0 1 0 0.124 0 1 0 0.119 0 1 0 -0.006 0 1 0 -0.017 0 1 0 -
0.111 0 1 0 -0.114 0 1 0 -0.103 0 1 0 -0.1 0 1 0 -0.1 0 1 0 -0.1 0
1 0 -0.1 0 1 0 -0.101 0 1 0 -0.188 0 1 0 -0.183 0 1 0 -0.121 0 1 0 -
-0.123 0 1 0 -0.129 0 1 0 -0.129 0 1 0 -0.129 0 1 0 -0.129 0 1 0 -
0.128 0 1 0 -0.126 0 1 0 0.109 0 1 0 0.105 0 1 0 0.1 0 1 0 0.1 0 1
0 0.1 0 1 0 0.1 ` />

```

```

<OrientationInterpolator DEF='EntityOrientationX' key = `0 0.049 0.05
0.059 0.06 0.098 0.098 0.157 0.157 0.214 0.23 0.291 0.292 0.307 0.308
0.315 0.315 0.369 0.403 0.461 0.487 0.554 0.564 0.582 0.583 0.623 0.623
0.678 0.678 0.752 0.77 0.829 0.829 0.888 0.888 0.949 0.949 0.993 1 `
keyValue = ` 1 0 0 0.318 1 0 0 0.424 1 0 0 0.444 1 0 0 -0.351 1 0 0
-0.342 1 0 0 -0.332 1 0 0 -0.333 1 0 0 -0.334 1 0 0 -0.336 1 0 0 -
0.336 1 0 0 -0.336 1 0 0 -0.336 1 0 0 -0.333 1 0 0 -0.309 1 0 0 -
0.31 1 0 0 -0.329 1 0 0 -0.33 1 0 0 -0.326 1 0 0 -0.324 1 0 0 -0.324
1 0 0 -0.324 1 0 0 -0.324 1 0 0 -0.325 1 0 0 0.45 1 0 0 0.425 1 0 0
0.334 1 0 0 0.335 1 0 0 0.338 1 0 0 0.338 1 0 0 0.338 1 0 0 0.338 1
0 0 0.338 1 0 0 0.336 1 0 0 0.328 1 0 0 0.326 1 0 0 0.324 1 0 0 0.324
1 0 0 0.324 1 0 0 0.324 ` />

```

LIST OF REFERENCES

- Altova. (n.d.). *XML Editor: XMLSpy*. Retrieved May 18, 2020, from <https://www.altova.com/xmlspy-xml-editor>
- AmBrSoft. (2014, December). *Collinear 3-dimensional lines*. <http://www.ambrsoft.com/TrigoCalc/Line3D/LineColinear.htm>
- Arasu, A., & Manku, G. S. (2004). Approximate counts and quantiles over sliding windows. *Proceedings of the Twenty-Third ACM Symposium on Principles of Database Systems - PODS '04*, 286. <https://doi.org/10.1145/1055558.1055598>
- AVM International. (n.d.). *FRITZ!Box 7490*. Retrieved April 20, 2020, from <https://en.avm.de/products/fritzbox/fritzbox-7490/>
- Blais, C. L. (2018, September). *Rich semantic track (RST) ontology: Unified semantics and pragmatics for track data interchange*. https://wiki.nps.edu/pages/viewpage.action?pageId=1082228797&preview=/1082228797/1173261305/Blais_Defense_RichSemanticTrackJuly2018.pdf
- Blender Foundation. (n.d.). *Home page*. Retrieved March 25, 2020, from <https://www.blender.org/>
- Borenstein, N. S., & Freed, N. (n.d.). *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. Retrieved May 18, 2020, from <https://tools.ietf.org/html/rfc2045>
- Brook, M. A. (2015). *C2Sim: Complex Coalition Interoperation (Multiple Nations, Multiple Domains)*. STO-EN-MSG-141. <https://www.sto.nato.int/publications/STO%20Educational%20Notes/STO-EN-MSG-141/EN-MSG-141-05.pdf>
- Brutzman, D. (n.d.). *Videos/demonstrations/NRWG2020/README.md · master · Savage / Spiders3dPublic*. GitLab - Spiders 3D Public. Retrieved April 27, 2020, from <https://gitlab.nps.edu/Savage/Spiders3dPublic/-/blob/master/videos/demonstrations/NRWG2020/README.md>
- Brutzman, D., & Daly, L. (2007). *X3D*. Elsevier. <https://doi.org/10.1016/B978-0-12-088500-8.X5000-7>
- CadStudio. (n.d.). *VRMLout for AutoCAD, VRML Translator for Inventor*. Retrieved May 18, 2020, from <https://www.cadstudio.cz/vrmlout>
- Carlson, J. (2020). *Coderextreme/X3DJSONLD* [HTML]. <https://github.com/coderextreme/X3DJSONLD> (Original work published 2015)

- Carlsonsolutiondesign. (2019). *Carlsonsolutiondesign/x3dpsail* [Python].
<https://github.com/carlsonsolutiondesign/x3dpsail> (Original work published 2019)
- Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., & Ranzuglia, G. (2008). MeshLab: An Open-Source Mesh Processing Tool. *Eurographics Italian Chapter Conference*, 8 pages. <https://doi.org/10.2312/LOCALCHAPTEREVENTS/ITALCHAP/ITALIANCHAPCONF2008/129-136>
- Create3000. (n.d.-a). *Getting Started » X_ITE X3D Browser » CREATE3000*. Retrieved May 18, 2020, from http://create3000.de/x_ite/getting-started/
- Create3000. (n.d.-b). *Titania*. Titania X3D Editor. Retrieved March 25, 2020, from <http://create3000.de/>
- Debich, S. J. (2015). *The role of efficient XML interchange (EXI) in Navy wide-area network (WAN) optimization*. Naval Postgraduate School.
- GitHub. (n.d.-a). *Extrusion not working · Issue #26 · castle-engine/view3dscene*. GitHub. Retrieved May 18, 2020, from <https://github.com/castle-engine/view3dscene/issues/26>
- GitHub. (n.d.-b). *SenseGraphics/h3dapi*. GitHub. Retrieved May 10, 2020, from <https://github.com/SenseGraphics/h3dapi>
- Hamming, R. W. (2003). *Art of Doing Science and Engineering: Learning to Learn*. Taylor & Francis. <https://books.google.com/books?id=49QuCOLIJLUC>
- Hill, B. W. (2015). *Evaluation of efficient XML interchange (EXI) for large datasets and as an alternative to binary JSON encodings*. Naval Postgraduate School.
- IEEE. (2012). *IEEE Standard for Distributed Interactive Simulation—Application Protocols*. IEEE. <https://doi.org/10.1109/IEEESTD.2012.6387564>
- Instant Reality. (n.d.). *Instantreality.org*. Retrieved May 10, 2020, from <https://www.instantreality.org/>
- Josefsson, S. (2006). *The Base16, Base32, and Base64 Data Encodings*. <https://tools.ietf.org/html/rfc4648>
- Kamburelis, M. (n.d.). *View3dscene | Castle Game Engine*. Retrieved May 10, 2020, from <https://castle-engine.io/view3dscene.php>
- Kendig, K. (2000). *Is a 2000-Year-Old Formula Still Keeping Some Secrets?* https://www.maa.org/sites/default/files/images/upload_library/22/Ford/Kendig402-415.pdf

- Naftalin, M., & Wadler, P. (2007). *Java Generics and Collections*. O'Reilly.
<https://books.google.com/books?id=VUSbAgAAQBAJ>
- NATO S&T Organization. (2018). *NMSG Report Cards*.
<https://www.tecnologiaeinnovacion.defensa.gob.es/Lists/Publicaciones/Attachments/224/03%20NMSG%20Report%20Cards%20last%20update%2023%20Feb%202018.pdf>
- NPS Savage Research Group. (n.d.-a). *Savage Developers Guide*. Retrieved May 17, 2020, from <https://savage.nps.edu/Savage/developers.html#>
- NPS Savage Research Group. (n.d.-b). *Savage Developers Guide*. Retrieved May 18, 2020, from <https://savage.nps.edu/Savage/developers.html#Xj3D>
- NPS Savage Research Group. (n.d.-c). *X3D Validator*. Retrieved May 18, 2020, from <https://savage.nps.edu/X3dValidator>
- NPS Savage Research Group. (n.d.-d). *X3D-Edit 3.3 Authoring Tool for Extensible 3D (X3D) Graphics*. Retrieved March 25, 2020, from <https://savage.nps.edu/X3D-Edit/>
- Pitch Technologies. (n.d.). *Pitch Technologies – Pitch CDS Gateway – Simulation Security Solution for HLA*. Retrieved May 14, 2020, from <http://pitchtechnologies.com/products/pitch-cds-gateway/>
- Pullen, Dr. J. M., Douglas Corner, Dr. Samuel Singapogu, Dr. Curt Blais, Dr. Douglas Reece, & Jim Ruth. (2019). *Command and Control System to Simulation System Interoperation: Development of the C2SIM Standard*.
<https://www.semanticscholar.org/paper/Command-and-Control-System-to-Simulation-System-of-Pullen-Corner/ef10ec75b18d0b40bd2aad8d100f2e3473412d2>
- Pullen, M., Khimeche, L., & Galvin, K. (2018, December 3). *Validating a Command and Control-Simulation Interoperation Standard*. <https://c4i.gmu.edu/wp-content/uploads/C4ISeminar-3Dec18.pdf>
- RedSim. (n.d.). *DIS PDU Recorder*. Redsim. Retrieved May 18, 2020, from <http://www.redsim.com/products/dis-pdu-recorder.html>
- Schutt, T. (2019, August). *MV3500 2019 Assignments*. GitLab. https://gitlab.nps.edu/Savage/NetworkedGraphicsMV3500/-/blob/master/assignments/src/MV3500Cohort2019JulySeptember/projects/SchuttFetterolf/FinalProject_SchuttFetterolf_.pptx
- SISO. (n.d.-a). *C-DIS PDG*. Compressed DIS PDG. Retrieved May 11, 2020, from <https://www.sisostds.org/StandardsActivities/DevelopmentGroups/C-DISPDG.aspx>

- SISO. (n.d.-b). *SISO University—Bridges and Gateways*. Retrieved May 28, 2020, from <https://www.sisostds.org/Workshops/SISOUniversity.aspx>
- SISO. (2007). *SISO-REF-020-DRAFT: Reference for Guide: DIS Plain and Simple*. <https://savage.nps.edu/Savage/documents/SISO-REF-020-DRAFT%20DIS%20Plain%20and%20Simple-20131107.pdf>
- SISO. (2015). *SISO-STD-001-2015 GRIM RPR FOM*. SISO. https://www.sisostds.org/DigitalLibrary.aspx?Command=Core_Download&EntryId=30822
- Snyder, S. L. (2010). *Efficient XML Interchange (EXI) compression and performance benefits: Development, implementation and evaluation*. Naval Postgraduate School.
- Sons, K. (2019). *Supporting/xiot* [C++]. <https://github.com/Supporting/xiot> (Original work published 2013)
- SourceForge. (n.d.-a). *FreeWRL VRML/X3D browser*. SourceForge. Retrieved May 10, 2020, from <https://sourceforge.net/projects/freewrl/>
- SourceForge. (n.d.-b). *Xj3D*. SourceForge. Retrieved May 10, 2020, from <https://sourceforge.net/projects/xj3d/>
- Visco. (n.d.). *Visco—Ocatga Player*. Retrieved May 10, 2020, from <https://visco.no/OCTAGA>
- W3. (n.d.-a). *HTML 5.2*. Retrieved May 27, 2020, from <https://www.w3.org/TR/html52/>
- W3. (n.d.-b). *HTML 5.2: 1.1. Introduction*. Retrieved May 27, 2020, from <https://www.w3.org/TR/html52/introduction.html#background>
- W3. (n.d.-c). *Localization vs. Internationalization*. Retrieved April 8, 2020, from <https://www.w3.org/International/questions/qa-i18n.en>
- Web3D. (n.d.-a). *What is X3D? | Web3D Consortium*. Retrieved May 18, 2020, from <https://www.web3d.org/x3d/what-x3d>
- Web3D. (n.d.-b). *X3D Ontology for Semantic Web*. Retrieved May 18, 2020, from <https://www.web3d.org/x3d/content/semantics/semantics.html>
- Web3D. (n.d.-c). *X3D Resources*. Retrieved May 18, 2020, from <https://www.web3d.org/x3d/content/examples/X3dResources.html#QualityAssurance>
- Web3D. (n.d.-d). *X3D Tooltips version 4.0*. Retrieved May 18, 2020, from <https://www.web3d.org/x3d/content/X3dTooltips.html>

- Web3D. (n.d.-e). *X3D Version 4 Overview / Web3D Consortium*. Retrieved May 18, 2020, from <https://www.web3d.org/x3d4>
- Web3D. (n.d.-f). *X3D-Edit—NetBeans Plugin detail*. Retrieved March 25, 2020, from <http://plugins.netbeans.org/plugin/6191/x3d-edit>
- Web3D. (n.d.-g). *X3DJSAIL: X3D Java Scene Access Interface Library*. Retrieved May 18, 2020, from <https://www.web3d.org/specifications/java/X3DJSAIL.html>
- Web3D Consortium. (2020a, April 1). *Altova reports 5,367,804 users worldwide who have XMLSpy*. [Tweet]. Twitter. <https://twitter.com/Web3DConsortium/status/1245453673523638272>
- Web3D Consortium. (2020b, April 1). *Interesting student #X3D model*. [Tweet]. Twitter. <https://twitter.com/web3dconsortium/status/1245370306417418240>
- Williams, J. S. (2009). *Document-based message-centric security using XML authentication and encryption for coalition and interagency operations*. Naval Postgraduate School.
- WireShark. (n.d.). *Go Deep*. Retrieved January 28, 2020, from <https://www.wireshark.org/>
- X3dom. (n.d.). *Home Page*. Retrieved May 18, 2020, from <https://www.x3dom.org/>

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California