

Teaching Network Traffic Matrices in an Interactive Game Environment

Chasen Milner^{1,2}, Hayden Jananthan², Jeremy Kepner², Vijay Gadepally², Michael Jones²,
Peter Michaleas², Ritesh Patel², Sandeep Pisharody², Gabriel Wachman², Alex Pentland²
¹USAF, ²MIT

Abstract—The Internet has become a critical domain for modern society that requires ongoing efforts for its improvement and protection. Network traffic matrices are a powerful tool for understanding and analyzing networks and are broadly taught in online graph theory educational resources. Network traffic matrix concepts are rarely available in online computer network and cybersecurity educational resources. To fill this gap, an interactive game environment has been developed to teach the foundations of traffic matrices to the computer networking community. The game environment provides a convenient, broadly accessible, delivery mechanism that enables making material available rapidly to a wide audience. The core architecture of the game is a facility to add new network traffic matrix training modules via an easily editable JSON file. Using this facility an initial set of modules were rapidly created covering: basic traffic matrices, traffic patterns, security/defense/deterrence, a notional cyber attack, a distributed denial-of-service (DDoS) attack, and a variety of graph theory concepts. The game environment enables delivery in a wide range of contexts to enable rapid feedback and improvement. The game can be used as a core unit as part of a formal course or as a simple interactive introduction in a presentation.

Index Terms—Traffic matrix, adjacency matrix, education, serious games

I. INTRODUCTION

As the Internet becomes more critical for modern society this requires increasing efforts for its improvement and protection. Network traffic matrices (adjacency matrices in the graph theory community) are a powerful tool for understanding and analyzing networks (and graphs) [1]–[4]. Matrix based approaches have become even more powerful with the advent of high-performance analytic software standards like the GraphBLAS [5]–[8] available in many programming environments [9]–[15]. These technologies have enabled the anonymized real-time analysis of terabit scale networks with trillions of events [16]–[19].

This material is based upon work supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Under Secretary of Defense for Research and Engineering. Research was also sponsored by the United States Air Force Research Laboratory and the Department of the Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Department of the Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein. Use of this work is controlled by the human-to-human license listed in Exhibit 3 of <https://doi.org/10.48550/arXiv.2306.09267>

Briefly, an adjacency matrix

$$\mathbf{A}(i, j) = v$$

indicates that a graph has edge starting at vertex i and ending at vertex j with corresponding value v . A network traffic matrix is an adjacency matrix where the vertices are sources and destinations on a computer network and the weight represents the amount of information being sent from source i to destination j . Typically v is measured in terms of the number of bytes or data packets sent. Formally, i and j are chosen from pre-fixed initial segments of the positive integers. In computer networks, other labels for i and j are often used (such as strings) which can be handled with the more general associative array abstraction [2]. In this paper, the term *traffic matrix* is used regardless of whether the sources and destinations are denoted with integers or strings.

Adjacency matrix concepts are generally available in online graph theory education resources. Examination of relevant courses from some popular online learning platforms indicates that adjacency matrix concepts are introduced in some online courses [20]–[25]. Video resources are the most common way to present the educational material with text being much less common. The reliance on video is understandable with how visually motivated graph theory is.

In the network protection domain, adjacency matrices or their corresponding network traffic matrices are rarely taught in online cybersecurity classes. The Cybersecurity Labs and Resource Knowledge-base, or CLARK, is an ever growing online library of cybersecurity education [26] that contains over 1500 cyber educational resources encompassing over 25 topic categories. Over 150 educational resources are collected under the *Network Security* topic with three having matrix content. One resource presents adjacency matrices in the context of social network analysis [27], a second uses matrix-matrix multiply as example parallel computing application [28], and a third represents neural networks as matrices for machine learning [29]. At the time of writing, none of the CLARK cybersecurity resources present network traffic matrix concepts as a tool for protecting computer networks. This may be in part because available educational resources currently hosted on CLARK requires little beyond high school mathematics. This is to be expected, as the intended audience is cybersecurity experts and not scientists, engineers, or mathematicians. The mathematical sophistication of adjacency matrices can be a barrier to the network security audience which may have

limited exposure to advanced mathematics. Interactive game environments can be a powerful tool for distilling sophisticated concepts into an easily digestible format.

Serious games are a type of video game that focus on acquiring knowledge, often used for professional training and education [30]. Research over the past several decades has shown that bringing serious games into the classroom is increasing in popularity and can enhance learning, make lessons more enjoyable for the student, and increase motivation and engagement [31]. When looking at how students from different backgrounds benefit from game-based learning it has been shown that that cognitive, motivational, and behavioural outcomes of educational games show a moderate effect compared to conventional teaching methods [32]. Game-based education is not new to the cybersecurity field with notable card-based games Backdoors & Breaches [33] and Riskio [34] as examples of effective teaching tools.

Our *Traffic Warehouse* video game is a prototype designed to allow a user to be introduced to various network traffic matrix concepts. The game has a convenient broadly accessible delivery mechanism that enables making material available more rapidly to the intended audience as compared to other modalities such as static text or video. Specifically, the game is designed to be extensible by non-game developers so that it is possible to rapidly add new traffic matrix concepts. Using the online game approach the material can be made available to anyone in almost any context.

The rest of the paper is organized as follows. First, the extensible learning module JSON (JavaScript Object Notation) file is described as this is core educator interface to *Traffic Warehouse*. Next, a brief review of how the Godot and MagicaVoxel technologies were selected for this development effort. Some illustrative coding examples are provided to illustrate some details of the implementation. Finally, several learning modules are presented with their corresponding in-game screenshots.

II. EXTENSIBLE LEARNING MODULE FILE

The key design choice of the *Traffic Warehouse* game was to define the learning modules via easily editable JSON [35] files that a non-game developer could use to create new learning modules. Using this approach a student is able to load a set of JSON files that contain different traffic matrices with associated multiple choice questions.

Learning modules consist of a zip file containing multiple JSON files that the user can select and load into the game. *Traffic Warehouse* will take the zip file and load each of the JSON files contained in it and present them sequentially one at a time. To create a single matrix lesson there are example files that can be duplicated and modified. Given the core concept being illustrated is the notion of traffic matrices, a key feature is the size of the matrix. There are template JSON files for 6×6 or 10×10 matrices. Inside the JSON file the first fields are for the title of the lesson, the author, and the traffic matrix axis labels. There is a single list of axis labels that will be applied to both the vertical and horizontal axis. Shorter all

caps labels are easier to view in the game. In this example, the labels correspond to work station (WS), server (SRV), external (EXT), and adversary (ADV).

```
"name": "10x10 Template",
"size": "10x10",
"author": "Chasen Milner",
"axis_labels": [
  "WS1", "WS2", "WS3", "SRV1",
  "EXT1", "EXT2",
  "ADV1", "ADV2", "ADV3", "ADV4",
],
```

The next—and perhaps most critical field in the JSON file—specifies the network traffic matrix. With this field the number of packets sent between each source and destination is specified. While there is no hard limit in code, through testing it has been found that fewer than 15 packets between any source and destination displays well. The matrix is represented as a list of lists to make it intuitive for an educator to type out exactly what the student will see.

```
"traffic_matrix": [
  [1,0,0,0,0,0,0,0,0,2],
  [0,1,0,0,0,0,0,0,2,0],
  [0,0,1,0,0,0,0,2,0,0],
  [0,0,0,1,0,0,2,0,0,0],
  [0,0,0,0,1,2,0,0,0,0],
  [0,0,0,0,2,1,0,0,0,0],
  [0,0,0,2,0,0,1,0,0,0],
  [0,0,2,0,0,0,0,1,0,0],
  [0,2,0,0,0,0,0,0,1,0],
  [2,0,0,0,0,0,0,0,0,1],
],
```

After choosing the number of packets per matrix position, the next field specifies the colors of each entry in the matrix to grey (0), blue (1), or red (2). These colors can be an important aid for illustrating important cybersecurity concepts such as internal networks (blue) and adversarial networks (red).

```
"traffic_matrix_colors": [
  [0,0,0,0,0,0,2,2,2,2],
  [0,0,0,0,0,0,2,2,2,2],
  [0,0,0,0,0,0,2,2,2,2],
  [0,0,0,0,0,0,2,2,2,2],
  [0,0,0,0,0,0,0,0,0,0],
  [0,0,0,0,0,0,0,0,0,0],
  [1,1,1,1,0,0,0,0,0,0],
  [1,1,1,1,0,0,0,0,0,0],
  [1,1,1,1,0,0,0,0,0,0],
  [1,1,1,1,0,0,0,0,0,0],
],
```

The final fields in the JSON specify if there is a question, what the question is, the possible answers, and corresponding correct answer. *Traffic Warehouse* will randomize the list that has the answers when they are displayed, so the first element will not always be the first option given. Our choice to have three available multiple choice answers was deliberate as there is some evidence indicating that using a three choice multiple choice questions is beneficial in balancing the quality multiple choice questions against devaluing the assessment of the student's knowledge [36] [37] [38]. The ability to toggle a question on and off allows for a more interactive experience where an educator can have an open discussion or prompt an entire class through online polls.

	Godot	Unity	Unreal
Cost	Always Free	Free when making less than \$100k/yr	Free when making less than \$1mil
Language Used	C#, GDScript	C#	C++
Can Import .obj	Yes	Yes	Yes
Exports to Platform	HTML5, Windows, Mac, *NIX	HTML5, Windows, Mac, *NIX	HTML5, Windows, Mac, *NIX
Online Tutorials	Some	Many	Many
Asset Store	Almost non-existent	Many high quality assets	Many high quality assets

TABLE I: Comparison between the Godot engine and two other industry standards, Unity and Unreal. We compare the cost, the language used in-engine, the ability to import .obj files, what platforms that can the engine build for, the availability of online tutorials and training courses, and how robust the asset store is for purchasing premade assets.

	MagicaVoxel	Blender	Maya
Cost	Free to use	Free to use	\$1,875/yr
Model Creation	LEGO-like voxel building	Polygon mesh, digital sculpting	Polygon mesh, digital sculpting
Texture Creation	Paint-by-voxel, place colored voxel	UV Unwrapping, paint-on-model	UV Unwrapping, paint-on-model
Animation	Simple animations	Advanced animations	Advanced animations
Can export to .obj	Yes	Yes	Yes

TABLE II: Comparison between two industry standard 3D modeling programs and MagicaVoxel. We compare the cost, how models are made, how textures are created, animation capability, and if they can export to the model format we need for our game engine.

```
"has_question":true,
"question":"How many packets did WS1 send to ADV4?",
"answers":["0", "1", "2",],
"correct_answer_element":2,
```

JSON is a common, lightweight, human readable data interchange format that is well known in the information technology and cybersecurity fields. JSON is a plaintext file so the template can be edited with a simple text editor. Likewise, being plaintext allows for any security review to be accomplished quickly and efficiently. If a learning module needs to be brought into a restricted area it can be easily done so on printed paper and reviewed and approved by the appropriate security team member, then simply hand typed back on to the restricted machine.

III. GAME TECHNOLOGY

A variety of technologies can be used to create a video game environment. Game engines allow a programmer to code the environment, presentation, and dynamics of the game (see Table I). The 3D visual objects or *assets* in the game are typically created with a modeling program (see Table II). Care must be used to select game engine and modeling technologies that are appropriate to goals and scale of the game development effort. In this case, for a simple educational game, the emphasis is on availability and ease-of-use so that others can readily build on the work.

Traffic Warehouse uses Godot [39], a free and open-source game engine. The engine supports coding in either C# (for

```
static void HelloWorld()
{
    Console.WriteLine("Hello, world!");
}

static void Main(string[] args)
{
    MyMethod();
}
```

(a) C#

```
def HelloWorld():
    print('Hello, world!')

HelloWorld()
```

(b) Python

```
func _ready():
    HelloWorld()

func HelloWorld():
    print('Hello, world!')
```

(c) GDScript

Fig. 1: **Hello World in C#, Python, and GDScript.** This example illustrates some of the similarities between GDScript and Python when compared to C#.



Fig. 2: **Scene Tree Example.** This shows the scene tree for the training level of *Traffic Warehouse*. It has some elements that are exclusive to the training level but shares many others with a standard level.

performance) or in a scripting language called GDScript (for ease). *Traffic Warehouse* is written using GDScript, which is similar to Python and easy to learn (see Figure 1 for a comparison of C#, Python, and GDScript). Godot uses a node and scene based system to build games (Figure 2), similar to the way other engines treat prefabs and scriptable objects. The node/scene system is simple yet powerful, making for a very easy and clean workflow when rapidly developing games. With the need for new educational games in the cyberspace community, the ability to rapidly develop new educational games is valuable. Godot, when compared to staple game engines, such as Unity [40] or Unreal [41], tends to target

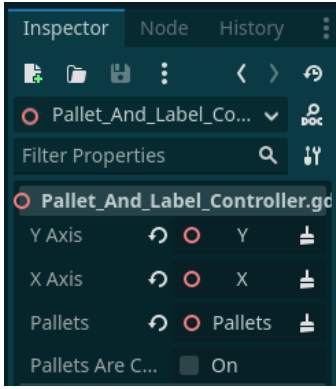


Fig. 3: **Export Variable Example.** This shows the Inspector tab which allows editing of various properties of our node. By manually exporting several variables they can be edited in this environment.

lower-end hardware, which makes Godot accessible to larger user base (Table I). Additionally, Godot is designed for fast switching between coding and visual scene editing, further speeding up a developer's workflow.

All of the visual assets for *Traffic Warehouse* were created in MagicaVoxel [42], a free-to-use voxel editor which enables rapid development of simple yet appealing 3D assets. While 3D modeling programs like Maya [43] and Blender [44] are commonly used in industry, they have many excellent features that would not be utilized in a project at this small scale. When deciding on a tool to create assets our effort focused on a few metrics such as cost, speed of asset creation, and the overall ease of use (see Table II). Another benefit to choosing a simpler voxel based modeling program is that future additions can be made by anyone without needing to know how to use a more advanced 3D modeling program. When provided with a voxel based program, similar canvas size, and a limited color palette a broad audience can create simple game assets in a fairly consistent artistic style.

IV. IMPLEMENTATION

A key element in educational game design is whether to adopt a direct or metaphorical approach to the learning task [45]–[49]. Network traffic and network traffic matrices are abstract concepts, so a physical metaphor was selected to aid in the understanding of these ideas. *Traffic Warehouse* presents a stylized shipping warehouse where each entry in the traffic matrix is represented as a grid of shipping pallets on the warehouse floor that can be loaded with boxes (packets) to be shipped. The shipping warehouse metaphor lends itself to a simple 3D design (floor, pallets, and boxes) and dynamics (placing boxes on pallets). Like most game engines, Godot can natively read in a JSON file and store it as a dictionary, from which the *Traffic Warehouse* learning modules can be instantiated using the shipping warehouse metaphor.

As an illustrative example of how different aspects of the game were created, consider how the matrix row and column label names and pallet colors are assigned using the JSON

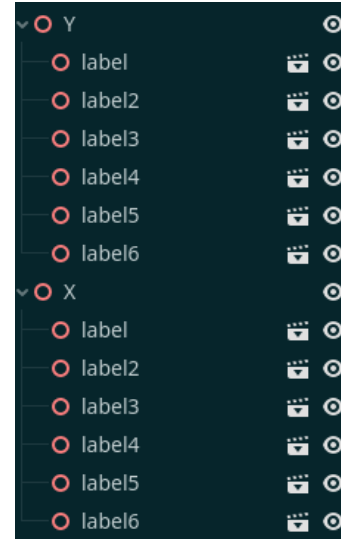


Fig. 4: **X and Y Nodes.** This shows X and Y nodes and their children, the label nodes.

learning module file. The following script is broken up into several parts in this paper but is a single file attached to the “Pallet_and_label_controller” node in the engine (Figure 2). In Godot a node is the smallest component that can be modified and used to build a scene. Several export variables are created to allow these variables be dynamically edited without having to edit the script as a whole (Figure 3). Additional variables are set to be assigned as soon as the node is loaded to get the level data and a list of all the child pallets that will be used later.

```
extends Node3D
@export var y_axis : Node3D
@export var x_axis : Node3D
@export var pallets : Node3D
@export var pallets_are_colored : bool = false
@onready var level_data : Node3D = $"../Data"
@onready var pallet_array : Array = pallets.get_children()
```

Next, an empty array is declared that will hold the pallet color codes. Different materials, or colors, are assigned to different variables. The different materials are preloaded so they can be quickly assigned to each pallet a set by the traffic matrix colors field from the JSON file.

```
var pallet_color_array : Array = []
var pallet_default_material : StandardMaterial3D =
    preload("res://Assets/Objects/pallet_material.tres")
var pallet_r_material : StandardMaterial3D =
    preload("res://Assets/Objects/pallet_material_r.tres")
var pallet_b_material : StandardMaterial3D =
    preload("res://Assets/Objects/pallet_material_b.tres")
var pallet_g_material : StandardMaterial3D =
    preload("res://Assets/Objects/pallet_material_g.tres")
var pallet_black_material : StandardMaterial3D =
    preload("res://Assets/Objects/pallet_material_black.tres")
```

The next step is to create a ready function. In Godot, `_ready()` is called as soon as the node enters a scene. It is useful when trying to let the node figure out where it is, to have it do an action immediately or, in this case, gather

data in preparation of performing its intended action. The ready function is pulling the “traffic_matrix_colors” from the pre-loaded JSON file stored in the “Data” node assigned to the variable “level_data” earlier in the script. After that is complete, set_labels() is called.

```
func _ready():
    for array in level_data.data["traffic_matrix_colors"]:
        pallet_color_array += array
    set_labels()
```

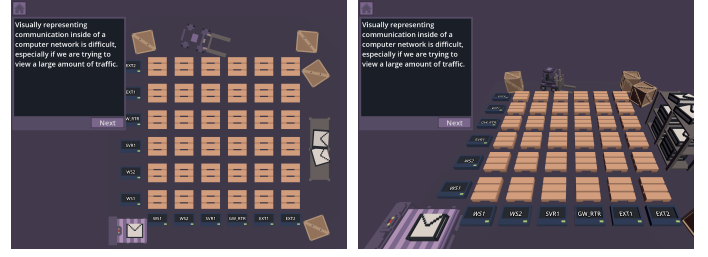
The set_labels() function retrieves the axis names from the “Data” node to assign the text labels along both the x and y axes. Earlier in the script the variables y_axis and x_axis were exported, so they could be assigned using the Inspector tab (Figure 3). These variables have been assigned to two nodes, Y and X, that have all of the label nodes under them as children (Figure 4).

```
func set_labels():
    var y_labels : Array = y_axis.get_children()
    var x_labels : Array = x_axis.get_children()
    if len(y_labels) != len(x_labels):
        printerr("Number of y labels does not match number of x labels!")
    elif len(level_data.data["axis_labels"]) != len(y_labels):
        printerr("Level data does not match number of labels!")
    else:
        var c : int = 0
        for label in level_data.data["axis_labels"]:
            y_labels[c].get_child(1).text = label
            x_labels[c].get_child(1).text = label
            c += 1
```

The last function in this example modifies a previously defined boolean controlling whether pallets are in their default or colored states. It is called whenever the toggle pallet color button is clicked. To ensure that each pallet gets the correct color the function iterates over the color array that contains all of the pallets on the level. This allows a step-by-step change of each pallet color to the one that was assigned in the JSON file.

```
func change_pallet_color():
    print("Change pallet color button")
    var c : int = 0
    if pallets_are_colored:
        print("Pallets are colored! Making them default")
        for color in pallet_color_array:
            pallet_array[c].get_child(0).material_override =
                pallet_default_material
            c += 1
        pallets_are_colored = false
    else:
        print("Pallets are default! Making them colored")
        for color in pallet_color_array:
            print("Matching color: " + str(color))
            match int(color):
                0: pallet_array[c].get_child(0).material_override =
                    pallet_g_material
                1: pallet_array[c].get_child(0).material_override =
                    pallet_b_material
                2: pallet_array[c].get_child(0).material_override =
                    pallet_r_material
                _: pallet_array[c].get_child(0).material_override =
                    pallet_black_material
            c += 1
        pallets_are_colored = true
```

The above example illustrates some aspects of the implementation of *Traffic Warehouse* in the Godot game engine



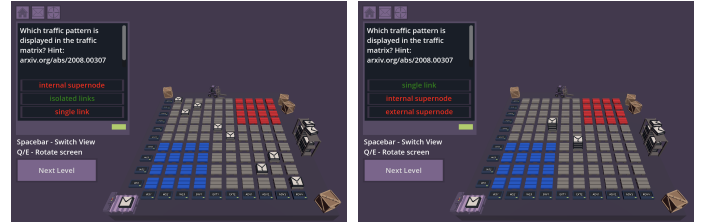
(a) 2D View.

(b) 3D View.



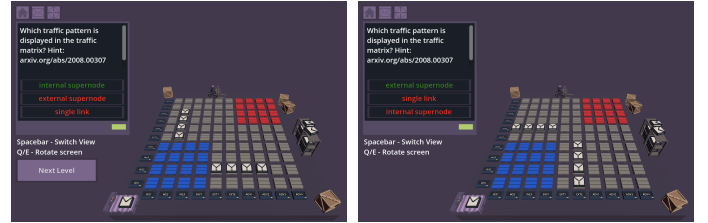
(c) Packets are all placed.

Fig. 5: Traffic Matrix Training. Training level of *Traffic Warehouse*. This level walks the player through what a traffic matrix is, why it is valuable to network security personnel, and how it will be represented in the game environment.



(a) Isolated Links.

(b) Single Links.



(c) Internal Supernode.

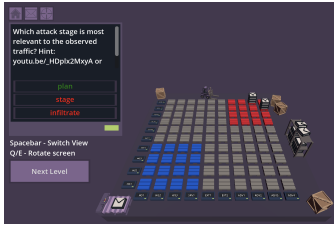
(d) External Supernode.

Fig. 6: Traffic Topologies. Traffic patterns representative of isolated links, single links, internal supernodes, and external supernodes shown on a 10×10 traffic matrix with a hint to an explanatory reference [50].

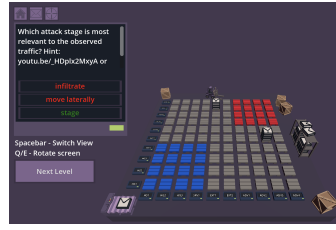
using the JSON learning module file.

V. LEARNING MODULES

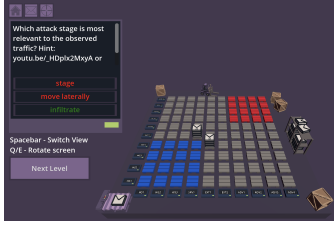
The goal of *Traffic Warehouse* is to allow the student to quickly engage with the learning modules in an intuitive manner. When the student starts the game they are first shown a network traffic matrix in a top-down 2D view. This view is how they would generally see a matrix in a spreadsheet, a textbook, or a presentation, which allows the student to



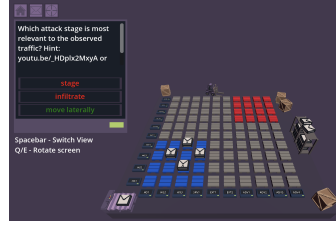
(a) Planning.



(b) Staging.

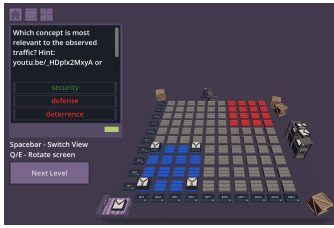


(c) Infiltration.

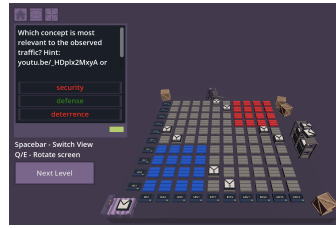


(d) Lateral Movement.

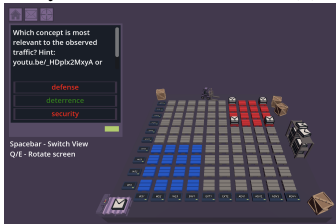
Fig. 7: **Notional Attack.** Traffic patterns relevant to the planning, staging, infiltration, and lateral movement stages of an attack shown on a 10×10 traffic matrix with hints to explanatory references [51], [52]



(a) Security.



(b) Defense.

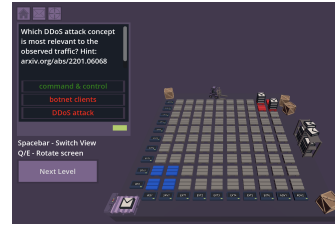


(c) Deterrence.

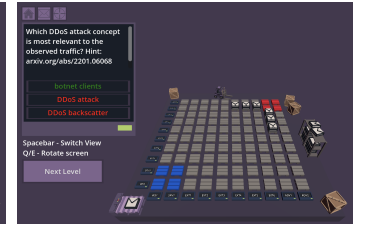
Fig. 8: **Network Security, Defense, and Deterrence.** Traffic patterns relevant to network security, defense, and deterrence approaches shown on a 10×10 traffic matrix with hints to explanatory references [51], [52]

connect back to those learning modes and relate them to what they see on the screen. The student has the ability to go into a 3D mode by pressing the spacebar key. The student can rotate the view using the Q and E keys. Switching between 2D and 3D and rotating the view allows the student to explore the network traffic matrix in new ways.

There is a single built-in module in *Traffic Warehouse* and that is the training level (Figure 5). This module walks the player through what a traffic matrix is, how to read one, how it is of value to them, and how it will be represented in the game environment. The training module also provides a space



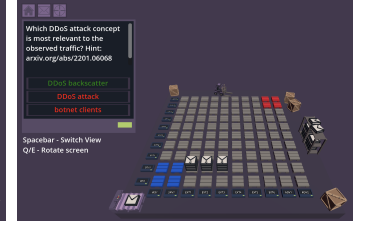
(a) Command and Control (C2).



(b) Botnet Clients.



(c) DDoS Attack.



(d) Backscatter.

Fig. 9: **DDoS Attack.** Traffic patterns relevant to the command-and-control (C2) servers, botnet clients, attack, and backscatter components of a distributed denial-of-service (DDoS) attack shown on a 10×10 traffic matrix with a hint to an explanatory references [52]

for the player to learn the controls of the game without needing to load in a learning module. Because of the flexible nature of the JSON module file, the potential number of modules is very large. Here a few will be described that illustrate the capabilities of *Traffic Warehouse*. For all the modules, the question type is the same “Which choice is the displayed traffic pattern most relevant to?” Other questions types are possible. This question type focuses on developing intuition in the student so they can connect the traffic patterns they are observing in a traffic matrix with common behaviors. Often a hint is provided that directs the student to an external resource which may help them with the question.

The basic traffic topologies module presents traffic patterns shown for isolated links, single links, internal supernodes, and external supernodes with additional color coding to help provide context for these patterns (Figures 6a through 6d, respectively). These traffic topologies can help a student gain further understanding of the relationship between the source and destination pairs on the traffic matrix and what it actually means when there is a connection between the two. After fully understanding the different topologies a student can move on to more advanced topics such as trying to analyze what is happening on a given matrix.

The notional attack module illustrates some of the common stages in a generic cyber attack. Each individual stage of a cyber attack can easily be demonstrated on a traffic matrix. First is the planning stage, which is done in adversarial space (Figure 7a). Second is staging, which takes place in greyspace (Figure 7b). Third is the infiltration stage, which happens at the border between grey and blue space (Figure 7c). The final stage is lateral movement, which happens inside blue space (Figure 7d). The individual stages can be represented

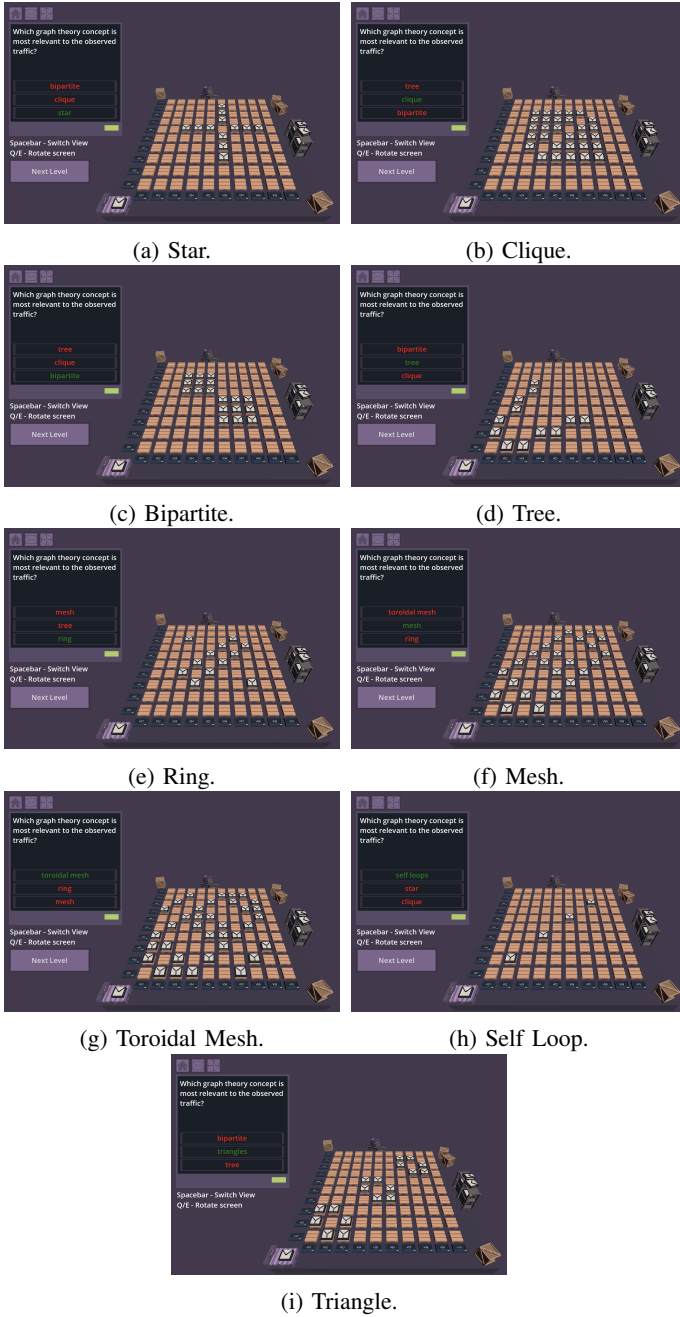


Fig. 10: **Graph Theory.** Traffic patterns relevant to various concepts in graph theory shown on a 10×10 traffic matrix.

as they are (with only the traffic associated with each stage shown) in an effort to explain how the traffic pattern would look like in a real network. After learning what the patterns look like individually, they could all be combined together or potentially mixed in with random background noise for a student to analyze and determine what is happening in the network.

A key concept in the protection of any domain is the distinction between (walls-in) security, (walls-out) defense, and deterrence [52]. The corresponding module illustrates these

more abstract concepts that are not associated with a specific attack, but places the broader idea of security, defense, and deterrence in a network traffic context. For security, the student can see that the traffic is operating within their own blue space, communicating with their own systems and ensuring no adversarial activity (Figure 8a). For defense, the student takes a step outside of their own network (Figure 8b). This illustrates that with community support from external sources, the student can identify threats to their network before they have the chance to enter their own secured space. Finally, deterrence is illustrated by showing that there is a credible activity in adversary space which arose as a response to unacceptable actions taken within the student's own network (Figure 8c).

Distributed denial-of-service (DDoS) attacks are amongst the most prevalent cyber attacks. The DDoS module illustrates some of the more common components of a DDoS attack from a botnet. Botnet command and control (C2) is shown by representing the communications in red space (Figure 9a). The communication from the C2 servers to the individual clients can be represented by identical communications between the C2 nodes and the botnet clients (Figure 9b). The attack is then represented by communication from the botnet clients to the blue controlled servers (Figure 9c), followed by the backscatter when the servers reply back to the illegitimate traffic (Figure 9d). Like other learning modules with multiple steps, after understanding these individual examples they could all be combined together or have background noise added to give a student even more of a challenge.

As a demonstration of the general flexibility of the *Traffic Warehouse* JSON file, a module was created to illustrate a wide range of graph theory concepts. In graph theory graphs are mathematical structures used to model pairwise relations between objects. Since the traffic matrix is simply a matrix filled with connections between two points it can represent different graphs in a way that a student may not have thought about which can help them further understand the concepts. This module demonstrates star, clique, bipartite, tree, ring, mesh, toroidal mesh, self loops, and triangle graphs (Figures 10a through 10i, respectively). The goal of providing these examples is to show that the information that can be displayed in *Traffic Warehouse* is not limited just to network communication in the cybersecurity community.

VI. CONCLUSIONS AND FUTURE WORK

Network defense is critical in effectively maintaining an organization. Providing the best education available to our network security experts serves to increase their ability to keep our infrastructure and data safe. Using educational gaming environments can be a powerful way to take complicated or abstract concepts and teach them in a simple and intuitive way that most will be able to understand. In addition to educating practitioners, these educational gaming environments can be used as demonstrations in presentations to help non-technical participants get an understanding of what is being explained. To fill this gap, the *Traffic Warehouse* interactive game environment has been developed to teach the foundations

of traffic matrices to the computer networking community. The game environment provides a convenient, broadly accessible, delivery mechanism that enables making material available rapidly to a wide audience. The core architecture of the game is a facility to add new network traffic matrix training modules via an easily editable JSON file. Using this facility an initial set of modules were rapidly created.

As with many educational or game development efforts, the potential for work and improvements is significant. These range from lower level improvements (e.g., expanding the range of colors and materials for various objects, hierarchical learning modules, and obfuscating question answers in the module file) to broader work on any number of concepts in the cybersecurity domain such as firewall configuration, reverse engineering, access control, registry or filesystem structure, and more. Furthermore, additional work should be done to find a rapid method of integrating educational games into already prepared course material and measuring the outcome and effect on the student.

ACKNOWLEDGMENTS

The authors wish to acknowledge the following individuals for their contributions and support: Daniel Andersen, William Arcand, Sean Atkins, David Bestor, William Bergeron, Chris Birardi, Bob Bond, Koley Borchard, Stephen Buckley, Chan-sup Byun, K Claffy, Cary Conrad, Timothy Davis, Chris Demchak, Garry Floyd, Jeff Gottschalk, Daniel Grant, Dhruv Gupta, Chris Hill, Michael Houle, Matthew Hubbell, Charles Leiserson, Kirsten Malvey, Lauren Milechin, Sanjeev Mohindra, Guillermo Morales, Andrew Morris, Julie Mullen, Heidi Perry, Christian Prothmann, Andrew Prout, Steve Rejto, Albert Reuther, Josh Rountree, Antonio Rosa, Daniela Rus, Mark Sherman, Scott Weed, Charles Yee, Marc Zissman.

REFERENCES

- [1] J. Kepner and J. Gilbert, *Graph algorithms in the language of linear algebra*. SIAM, 2011.
- [2] J. Kepner and H. Jananathan, *Mathematics of big data: Spreadsheets, databases, matrices, and graphs*. MIT Press, 2018.
- [3] G. Szárnyas, D. A. Bader, T. A. Davis, J. Kitchen, T. G. Mattson, S. McMillan, and E. Welch, "LAGraph: Linear algebra, network analysis libraries, and the study of graph algorithms," in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 243–252, IEEE, 2021.
- [4] D. A. Bader, *Massive Graph Analytics*. CRC Press, 2022.
- [5] T. Mattson, D. Bader, J. Berry, A. Buluc, J. Dongarra, C. Faloutsos, J. Feo, J. Gilbert, J. Gonzalez, B. Hendrickson, J. Kepner, C. Leiserson, A. Lumsdaine, D. Padua, S. Poole, S. Reinhardt, M. Stonebraker, S. Wallach, and A. Yoo, "Standards for graph algorithm primitives," in *2013 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–2, 2013.
- [6] J. Kepner, P. Aaltonen, D. Bader, A. Buluc, F. Franchetti, J. Gilbert, D. Hutchison, M. Kumar, A. Lumsdaine, H. Meyerhenke, S. McMillan, C. Yang, J. D. Owens, M. Zalewski, T. Mattson, and J. Moreira, "Mathematical foundations of the GraphBLAS," in *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–9, 2016.
- [7] A. Buluc, T. Mattson, S. McMillan, J. Moreira, and C. Yang, "Design of the GraphBLAS API for C," in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 643–652, 2017.
- [8] B. Brock, A. Buluc, T. G. Mattson, S. McMillan, and J. E. Moreira, "Introduction to GraphBLAS 2.0," in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 253–262, 2021.
- [9] T. A. Davis, "Algorithm 1000: SuiteSparse: GraphBLAS: Graph algorithms in the language of sparse linear algebra," *ACM Transactions on Mathematical Software (TOMS)*, vol. 45, no. 4, pp. 1–25, 2019.
- [10] P. Cailliau, T. Davis, V. Gadepally, J. Kepner, R. Lipman, J. Lovitz, and K. Ouaknine, "Redisgraph GraphBLAS enabled graph database," in *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 285–286, IEEE, 2019.
- [11] M. Pelletier, W. Kimmerer, T. A. Davis, and T. G. Mattson, "The GraphBLAS in Julia and Python: the PageRank and triangle centralities," in *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–7, 2021.
- [12] F. Dörre, A. Krause, D. Habich, and M. Junghanns, "A GraphBLAS implementation in pure Java," in *Proceedings of the 4th ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*, pp. 1–9, 2021.
- [13] B. Brock, S. McMillan, A. Buluc, T. G. Mattson, and J. E. Moreira, "GraphBLAS: C++ iterators for sparse matrices," in *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 238–246, IEEE, 2022.
- [14] A. Aadithya, A. Edpuganti, V. Rejathalal, S. Kumar, P. Poornachandran, et al., "An informal introduction to semirings for GraphBLAS using Matlab," in *2023 3rd International Conference on Intelligent Technologies (CONIT)*, pp. 1–6, IEEE, 2023.
- [15] J. Roose, M. Vaidya, P. Sadayappan, and S. Rajamanickam, "TenSQL: An SQL database built on GraphBLAS," in *2023 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–8, IEEE, 2023.
- [16] M. Jones, J. Kepner, D. Andersen, A. Buluc, C. Byun, K. Claffy, T. Davis, W. Arcand, J. Bernays, D. Bestor, W. Bergeron, V. Gadepally, M. Houle, M. Hubbell, H. Jananathan, A. Klein, C. Meiners, L. Milechin, J. Mullen, S. Pisharody, A. Prout, A. Reuther, A. Rosa, S. Samsi, J. Sreekanth, D. Stetson, C. Yee, and P. Michaleas, "GraphBLAS on the edge: Anonymized high performance streaming of network traffic," in *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–8, 2022.
- [17] M. Jones, J. Kepner, A. Prout, T. Davis, W. Arcand, D. Bestor, W. Bergeron, C. Byun, V. Gadepally, M. Houle, M. Hubbell, H. Jananathan, A. Klein, L. Milechin, G. Morales, J. Mullen, R. Patel, S. Pisharody, A. Reuther, A. Rosa, S. Samsi, C. Yee, and P. Michaleas, "Deployment of real-time network traffic analysis using GraphBLAS hypersparse matrices and D4M associative arrays," in *2023 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–8, 2023.
- [18] W. Bergeron, M. Jones, C. Barber, K. DeYoung, G. Amariuca, K. Ernst, N. Fleming, P. Michaleas, S. Pisharody, N. Wells, A. Rosa, E. Vasserman, and J. Kepner, "Hypersparse traffic matrix construction using GraphBLAS on a DPU," poster presented at *IEEE HPEC (arXiv:2310.18334)*, 2023.
- [19] J. Kepner, M. Jones, D. Andersen, A. Buluc, C. Byun, K. Claffy, T. Davis, W. Arcand, J. Bernays, D. Bestor, W. Bergeron, V. Gadepally, M. Houle, M. Hubbell, A. Klein, C. Meiners, L. Milechin, J. Mullen, S. Pisharody, A. Prout, A. Reuther, A. Rosa, S. Samsi, D. Stetson, A. Tse, C. Yee, and P. Michaleas, "Spatial temporal analysis of 40,000,000,000,000 Internet darkspace packets," in *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–8, 2021.
- [20] A. Meyer and A. Chlipala, "Mathematics for Computer Science." MIT OpenCourseWare, 2015.
- [21] J. Kepner, V. Gadepally, and H. Janathan, "Mathematics of Big Data and Machine Learning." MIT OpenCourseWare, 2020.
- [22] M. Fattah, "Graph Theory." Udemy, 2020.
- [23] N. Rhodes, "Algorithms on Graphs." Coursera, 2023.
- [24] A. Gupta, "Graph Analytics for Big Data." Coursera, 2023.
- [25] V. Gripon, P. Meyer, and N. Farrugia, "IMTx: Advanced Algorithmics and Graph Theory with Python." edX, 2023.
- [26] M. Dark, S. Kaza, and B. Taylor, "CLARK – the cybersecurity labs and resource knowledge-base – a living digital library," in *2018 USENIX Workshop on Advances in Security Education (ASE 18)*, (Baltimore, MD), USENIX Association, Aug. 2018.
- [27] J. Golbeck, "Predictive Analytics: Linear and Logistic Regression." CLARK: Cybersecurity Labs and Resource Knowledge-base, 2020.

- [28] J. Golbeck, "Map Reduce and Hadoop." CLARK: Cybersecurity Labs and Resource Knowledge-base, 2020.
- [29] L. Khan, "Probabilistic Models and Deep Neural Networks." CLARK: Cybersecurity Labs and Resource Knowledge-base, 2020.
- [30] J. Zeng, S. Parks, and J. Shang, "To learn scientifically, effectively, and enjoyably: A review of educational games," *Human Behavior and Emerging Technologies*, vol. 2, no. 2, pp. 186–195, 2020.
- [31] C. C. Ekin, E. Polat, and S. Hopcan, "Drawing the big picture of games in education: A topic modeling-based review of past 55 years," *Computers & Education*, vol. 194, p. 104700, 2023.
- [32] J. J. Michaela Artzmann, Lisette Hornstra and L. Kester, "Effects of games in STEM education: a meta-analysis on the moderating role of student background characteristics," *Studies in Science Education*, vol. 59, no. 1, pp. 109–145, 2023.
- [33] J. Young and S. Farshadkhah, "Backdoors & breaches: Using a tabletop exercise game to teach cybersecurity incident response," 11 2021.
- [34] S. Hart, A. Margheri, F. Paci, and V. Sassone, "Riskio: A serious game for cyber security awareness and education," *Computers & Security*, vol. 95, p. 101827, 2020.
- [35] T. Bray, "The JavaScript object notation (JSON) data interchange format," 2014.
- [36] T. M. Haladyna and S. M. Downing, "How many options is enough for a multiple-choice test item?," *Educational and Psychological Measurement*, vol. 53, no. 4, pp. 999–1010, 1993.
- [37] R. E. Landrum, J. R. Cashin, and K. S. Theis, "More evidence in favor of three-option multiple-choice tests," *Educational and Psychological Measurement*, vol. 53, no. 3, pp. 771–778, 1993.
- [38] A. Dehnad, H. Nasser, and A. F. Hosseini, "A comparison between three-and four-option multiple choice questions," *Procedia - Social and Behavioral Sciences*, vol. 98, pp. 398–403, 2014. Proceedings of the International Conference on Current Trends in ELT.
- [39] C. Bradfield, *Godot Engine Game Development Projects: Build Five Cross-platform 2D and 3D Games with Godot 3.0*. Packt Publishing Ltd, 2018.
- [40] W. Goldstone, *Unity game development essentials*. Packt Publishing Ltd, 2009.
- [41] J. Lee, *Learning Unreal Engine game development*. Packt Publishing Ltd, 2016.
- [42] "MagicaVoxel." <https://ephtracy.github.io/>, 2023. Accessed: 2023.
- [43] M. Ingrassia, *Maya for games: modeling and texturing techniques with Maya and Mudbox*. CRC Press, 2008.
- [44] L. Flavell, *Beginning Blender: open source 3D modeling, animation, and game design*. Apress, 2011.
- [45] P. P. Gómez-Martín, M. A. Gómez-Martín, P. Palmier Campos, and P. A. González-Calero, "Using metaphors in game-based education," in *International Conference on Technologies for E-Learning and Digital Entertainment*, pp. 477–488, Springer, 2007.
- [46] L. P. Rieber and D. Noah, "Games, simulations, and visual metaphors in education: antagonism between enjoyment and learning," *Educational Media International*, vol. 45, no. 2, pp. 77–92, 2008.
- [47] B. Law, "Puzzle games: a metaphor for computational thinking," in *European Conference on Games Based Learning*, p. 344, Academic Conferences International Limited, 2016.
- [48] S. Jackson, "Understanding IS/IT implementation through metaphors: A multi-metaphor stakeholder analysis in an educational setting," *Computers in Human Behavior*, vol. 55, pp. 1039–1051, 2016.
- [49] M. Allegra, A. Bongiovanni, G. Città, A. Cusimano, V. Dal Grande, M. Gentile, A. Kisslinger, D. La Guardia, G. Liguori, F. Lo Presti, *et al.*, "The role of metaphor in serious games design: the BubbleMumble case study," in *International Conference on Games and Learning Alliance*, pp. 198–207, Springer, 2021.
- [50] J. Kepner, C. Meiners, C. Byun, S. McGuire, T. Davis, W. Arcand, J. Bernays, D. Bestor, W. Bergeron, V. Gadepally, R. Harnasch, M. Hubbell, M. Houle, M. Jones, A. Kirby, A. Klein, L. Milechin, J. Mullen, A. Prout, A. Reuther, A. Rosa, S. Samsi, D. Stetson, A. Tse, C. Yee, and P. Michaleas, "Multi-temporal analysis and scaling relations of 100,000,000,000 network packets," in *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, IEEE, Sept. 2020.
- [51] J. Kepner, "Beyond Zero Botnets: Web3 Enabled Observe-Pursue-Counter Approach." TEDxBoston, June 2022.
- [52] J. Kepner, J. Bernays, S. Buckley, K. Cho, C. Conrad, L. Daigle, K. Erhardt, V. Gadepally, B. Greene, M. Jones, R. Knake, B. Maggs, P. Michaleas, C. Meiners, A. Morris, A. Pentland, S. Pisharody, S. Powazek, A. Prout, P. Reiner, K. Suzuki, K. Takhashi, T. Tauber, L. Walker, and D. Stetson, "Zero botnets: An observe-pursue-counter approach." Belfer Center Reports, 6 2021.