Naval Research Laboratory
Washington, DC 20375-5320

NRL/5540/MR--2022/6

Autonomous Cyber Response for Software-Defined Navy Networks

ALEXANDER VELAZQUEZ

Center for High Assurance Computer Systems Branch
Information Technology Division

September 25, 2022

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
25-09-2022	NRL Memorandum Report	April 18, 2022 – September 30, 2022

4. TITLE AND SUBTITLE

Autonomous Cyber Response for Software-Defined Navy Networks

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Alexander Velazquez

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
4X51

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
4555 Overlook Avenue, SW
Washington, DC 20375-5320

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/5540/MR--2022/6

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
4555 Overlook Avenue, SW
Washington, DC 20375-5320

10. SPONSOR / MONITOR'S ACRONYM(S)

NRL-NISE

11. SPONSOR / MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A: Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The Navy and Marine Corps increasingly rely on tactical networks for mission-critical functionality. Security and resilience of these networks require autonomous and automated capabilities. Cyberattacks that take place at machine speed cannot be addressed by cyber responses that are carried out at human speed. The complexity of these network also makes human-driven configuration, administration, and optimization unfeasible. In this project, we deploy a virtualization environment and develop a toolset to enable experimenters to rapidly deploy pre-configured but highly customizable experimentation environments that integrate software-defined networks (SDN) with autonomous cyber response capabilities. This will enable agile experimentation to validate SDN and autonomous cyber response concepts in the context of future Navy and Marine Corps tactical networks, which in turn can allow us to quantify overhead and improvements in security, resilience, and warfighter workload.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
					Alexander Velazquez
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**	U	15	**19b. TELEPHONE NUMBER** *(include area code)*
U	U	U			(202) 404-4879

This page intentionally left blank.

CONTENTS

This page intentionally left blank

EXECUTIVE SUMMARY

The Navy and Marine Corps increasingly rely on tactical networks for mission-critical functionality. Security and resilience of these networks require autonomous and automated capabilities. Cyberattacks that take place at machine speed cannot be addressed by cyber responses that are carried out at human speed. The complexity of these network also makes human-driven configuration, administration, and optimization unfeasible.

In this project, we deploy a virtualization environment and develop a toolset to enable experimenters to rapidly deploy pre-configured but highly customizable experimentation environments that integrate software-defined networks (SDN) with autonomous cyber response capabilities. This will enable agile experimentation to validate SDN and autonomous cyber response concepts in the context of future Navy and Marine Corps tactical networks, which in turn can allow us to quantify overhead and improvements in security, resilience, and warfighter workload.

This report serves to document our overall effort and key results in this NISE project.

This page intentionally left blank

AUTONOMOUS CYBER RESPONSE FOR SOFTWARE-DEFINED NAVY NETWORKS

1. INTRODUCTION AND PREVIOUS WORK

Naval superiority relies on complex weapons systems, information systems, and networks that need to support diverse and dynamic missions that evolve over time. These mission-support systems need to be resilient against cyberattacks and system failures. Adversaries are able to launch coordinated cyberattacks at high pace and volume, yet detection and recovery from attacks and failures rely on slow, human-driven processes. Thus, there is currently an imbalance between the pace at which cyberattacks can be carried out and the ability for state-of-the-art defenses (paired with human cyber warriors) to effectively detect and respond to cyber events. We need intelligent and proven autonomous detection and response capabilities in order to prevent and thwart sophisticated adversaries and to recover from system faults/failures. Furthermore, as military networks become more complex, the configuration, maintenance, and optimization of the network become an increasing burden for the warfighter. Autonomous network administration is an essential capability that would ease that burden as well as enable additional capabilities such as moving target defense.

In prior work, we developed the Autonomous Intelligent Resilient Security (AIRS) framework, an autonomous cyber response framework to defend systems from cyberattacks and system faults. This was done in collaboration with the Army Research Laboratory in order to target the software-defined network (SDN) layer of the Army's Command and Control / Internet of Battlefield Things (C2/IoBT) environment. Our research results demonstrated not only that SDN is a viable technology for tactical networks, but also that the AIRS framework could be used to effectively detect and autonomously respond to cyberattacks and system faults in a tactical environment.

Picking up from where the AIRS project left off, we have designed a new testbed and toolset that allows experimenters to deploy pre-configured but highly customizable experimentation environments that integrate SDN with autonomous cyber response capabilities. This enables agile experimentation to rapidly and effectively validate SDN and autonomous cyber response concepts in the context of future Navy and Marine Corps tactical networks and quantify overhead and improvements in security, resilience, and warfighter workload.

Whereas experiments and demonstrations for the AIRS project [1] focused on a fixed environment (based on the C2/IoBT environment) with a central node and two edge nodes, this effort takes this much further by greatly reducing the effort needed to deploy SDN and the AIRS framework (AIRS+SDN) in a wide variety of environments that represent the Naval and expeditionary domains. To this end, we conceptualized future Navy and Marine Corps tactical networks, which have more complexity and scale than the AIRS C2/IoBT environment.

Previous experimental results are summarized in [1, 2]. The work accomplished in this effort enables similar experiments and validation of AIRS+SDN in the Naval and expeditionary domains.

Manuscript approved September 22, 2022.

2. GOALS AND TECHNICAL ACHIEVEMENTS

2.1 High-level overview

We wish to demonstrate the feasibility and capabilities of software-defined networks (SDN) and the Autonomous Intelligent Resilient Security (AIRS) framework in a Naval and expeditionary context, and to do so in a highly flexible and repeatable way. To do this, we must conceptualize, implement, experiment with, and finally demonstrate a tactical network testbed that accurately captures the scale, complexity, and important characteristics of future naval and expeditionary networks.

In this effort, we strive to automate the labor-intensive process of setting up the experimentation environments in which one might want to deploy SDN together with the AIRS framework (AIRS+SDN) for testing, evaluation, and demonstration. The work documented in this report focuses more on the infrastructure and tooling around the setup and configuration of the testing/experimentation, rather than carrying out the testing/experimentation itself.

Our test and demonstration environment is implemented on a pair of high performance servers with plentiful hardware resources (each with: 2x16 CPU cores, 12x32 GB RAM, 8 10-Gigabit network interfaces, and over 8TB of SSD storage). The two servers are physically interconnected with a network switch. This hardware constitutes our physical infrastructure. The tools that we developed in this project are not specific to this physical infrastructure and can run on almost any hardware configuration, with some customization (although limited CPU/memory/disk resources will place an upper bound on how large the deployed environment can be). The particular virtualization environment software we chose to use for this project represents an evolution over previous iterations of the AIRS testbed. This rationale and consequences of changing virtualization environments are discussed in detail in Section 2.3. The virtualization environment software running on top of the physical infrastructure constitutes our virtualization infrastructure. On top of this virtualization infrastructure, we can spin up a real SDN composed of virtual machines (VMs), virtual network switches, and necessary software components (like the SDN controller software, AIRS monitors and agents, mission-related applications, and cyberattack scripts/agents).

The topology and configuration of the SDN/VMs/components will vary depending on the particular real-world system/environment it is modeling. As an outcome of this project, we wish have the capability to model not only a single system/environment; but rather, we wish to have the ability to model a wide array of environments that are representative within the Naval and expeditionary domains. A nice overview of network communication paradigms and potential bottlenecks between entities in expeditionary or carrier strike groups is laid out in [3]. We used this as the basis for our conceptualized pre-configured AIRS+SDN environments, although they can be customized further by an end-user/experimenter as needed.

2.2 Simulation vs. emulation vs. virtualization

Although some aspects of the conceptualized network environment are emulated (e.g., network characteristics of the SATCOM links), the SDN itself and all of the software running on the data plane is real, so the experiments run in real time. This ends up being the most time-consuming part of the experimentation lifecycle (running experiments and collecting the resulting data), but reducing the runtime is not something that can be easily done without fundamentally changing the testbed, and of course there are inherent advantages to the fact that it's a real SDN with real components, rather than a simulation.

Other than runtime experiments, another time-consuming aspect of operating the testbed is initial deployment and configuration. There are many VMs that need to be allocated, bootstrapped, and have their operating system (OS) customized and configured. Then, there is additional software in each VM to install and configure. Here, however, we can save a considerable amount of time by writing scripts and tools. In order to reduce the burden on the end-user and save as much time as possible, we have developed an automated deployment toolset to ease initial deployment, configuration, and customization of the experimentation environment on the testbed.

2.3 Migration to new virtualization environment

Previous iterations of the AIRS testbed have been based on different virtualization environments: first KVM, then Xen/Xenon [4], and most recently VMware ESXi. Each transition from one virtualization environment to the next was necessitated by a combination of technical requirements, design decisions, and maturation/evolution of these (and other) virtualization products over time.

When selecting a virtualization environment, there are always tradeoffs to consider. Given the goals of this project, we determined that migrating the testbed to a new virtualization environment would be beneficial, with the upsides heavily outweighing any potential downsides. The selected environment, Proxmox-VE [1], is mature, free and open source, and provides all of the features needed to support AIRS+SDN, while also providing additional automation and management capabilities that directly support key goals of this effort (i.e., enhanced ability to rapidly spin up and configure experimentation environments with minimal user effort). VMware ESXi is a powerful product, but it can be more difficult to use in our particular context and requires a paid license for most automation and centralized management capabilities (via its vCenter Server product), which is not only a financial burden, but also an additional configuration/management hurdle. Issues frequently arise months in the future as free, time-limited licenses expire and cause features to begin to silently fail. Any time and effort spent troubleshooting issues like this distracts from the work at hand and directly detracts from progress toward the goals of the project. Avoiding a paid license also sidesteps potential problems in the future – for example, even if there is no funding available for the project in the future, it still remains a viable product that can be demonstrated (with full set of capabilities) on short notice without the need to pay for, activate, or install a license.

2.4 Architecture

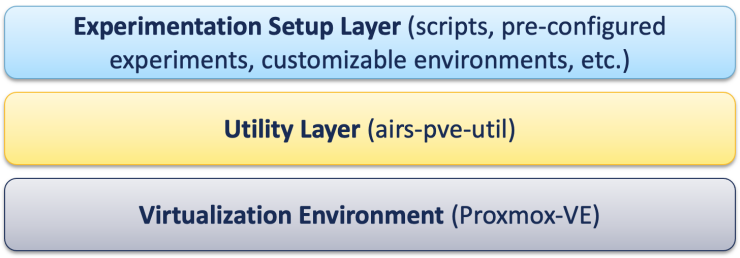

Fig. 1—The underlying virtualization environment provides a foundation for the utility layer, which in turn supports the experimentation setup layer.

Figure 1 shows the architecture that we implemented in order to realize our vision for agile experimentation with AIRS+SDN in the Naval and expeditionary domains.

[1] Proxmox-VE - Virtualization Management Platform: `https://www.proxmox.com/en/proxmox-ve`

On top of the virtualization environment, we implemented a Utility Layer that abstracts and consolidates the available functionality as needed for experimentation. Implementing this Utility Layer also serves to expand the action space that can be made available to the AIRS security manager and planner at runtime, in order to defend and repair the network. Actions (and meta-actions) that were previously unavailable or difficult to leverage due to the choice of virtualization environment are now available. However, additional work needs to be done to integrate some or all of these actions into the planner's hierarchical operational model and the execution platform within the security manager.

On top of the Utility Layer, we implemented an Experimentation Setup Layer that comprises various scripts, pre-configured experiments, and customizable environments. These allow an experimenter to easily and repeatably deploy and bootstrap a particular experiment design, or even customize it further for their particular needs.

2.5 Functionality of the Utility Layer

Table 1 lists the functions that are included in the Utility Layer library. This interface provides a convenient and useful layer of abstraction above the virtualization environment software itself.

Table 1—Functions included in Utility Layer library.

`is_bridgeid_valid`	Check if the given bridge ID is valid
`is_vmid_valid`	Check if the given VM ID is valid
`is_vmname_valid`	Check if the given VM name is valid
`does_exist_bridgeid`	Check if the given bridge ID is defined
`does_exist_vmid`	Check if the given VM ID is defined
`does_exist_vmname`	Check if the given VM name is in use
`get_vmids`	Get a list of all defined VM IDs
`get_vmnames`	Get a list of all VM names in use
`do_create_ovs_bridge`	Define a new network bridge (Open vSwitch)
`do_destroy_bridge`	Destroy a network bridge
`do_create_basic_vm`	Create a new VM with basic configuration
`do_convert_vm_to_tmpl`	Convert a VM to template
`do_clone_tmpl_to_new_vm`	Clone a new VM from a given template
`do_set_vm_autoboot`	Set VM autoboot setting
`do_start_vm`	Start the given VM
`do_start_vm_and_wait`	Start the given VM and block until it terminates
`do_start_vm_with_console`	Start the given VM and connect to its text-based console
`do_start_vm_with_console_and_wait`	Start the given VM, connect to its text-based console, and block until it terminates
`do_connect_vm_console`	Connect to text-based console of a running VM
`do_wait_for_vm`	Block until the given VM terminates
`do_shutdown_or_stop_vm`	Stop the given VM if it is running
`do_destroy_vm`	Destroy a VM
`do_add_vm_net`	Attach VM to a given network
`print_help`	Print all available utility functions

2.6 Functionality of the Experimentation Setup Layer

Table 2 lists all of the functions that are included in the Experimentation Setup Layer library. These functions provide mechanisms to create and cleanup network bridges, templates, and VMs; to start and stop the VMs; and finally, to create and destroy the environment as a whole.

Setting up an experiment is as easy as selecting a base configuration to use, customizing it if necessary, and then calling the `create` and `start` functions. Likewise, tearing down an experiment and freeing up any resources it was using is as easy as calling the `destroy` function.

Table 2—Functions included in Experimentation Setup Layer library.

`create`	Create all bridges, templates, and VMs
`destroy`	Destroy all bridges, templates, and VMs
`start`	Start all VMs asynchronously
`start_and_wait`	Start all VMs, connecting to the text-based console of each and blocking until it terminates
`start_and_wait_without_consoles`	Start all VMs, blocking for each one until it terminates
`stop`	Stop all VMs
`do_create_bridges`	Create bridges defined in the current configuration
`do_create_templates`	Create templates defined in the current configuration
`do_create_vms`	Create VMs defined in the current configuration
`do_cleanup_bridges`	Cleanup bridges defined in the current configuration
`do_cleanup_templates`	Cleanup templates defined in the current configuration
`do_cleanup_vms`	Cleanup VMs defined in the current configuration
`print_help`	Print all available experimentation setup functions

To encapsulate the functionality required by AIRS+SDN, two VM template images are provided. One of these, `airs-default`, is a relatively minimal Debian 11 installation with some customization and basic packages installed to provide a common baseline across all instantiated VMs. The second, `airs-framework-repo`, is based on `airs-default` but additionally contains source code, scripts, and Docker images that are used during the bootstrapping process to set up the SDN controller, switches, and AIRS components. Beyond these static resources, `airs-framework-repo` also becomes populated at deploy-time with dynamic instantiations of various configuration files (i.e., templates) that are shared by VMs that are part of a given experiment.

2.7 Experimentation environments and customizability

The Experimentation Setup Layer provides scripts that read a set of configuration variables (which can be customized) and expose commands to instantiate a fully functional experimentation environment on the testbed. For example, the script `setup-central-and-edge-nodes.sh` can be used to set up COC and NOTM nodes as in the AIRS C2/IoBT demonstration environment, or a central aircraft carrier node with a number of supporting ships as in the context of a carrier strike group [3]. The same script can be used to set up both environments; the only difference is the values of the configuration variables.

Using such a script is straightforward. One must first load a set of configuration variables, which can be done by manually defining the environment variables or by sourcing an existing file containing these variable values, and then interact with the script using its sub-commands. An example session is shown below:

```
# Load configuration variables
#source ./config/envvars-central-and-edge-nodes-marines-nrlnet.sh
# or
source ./config/envvars-central-and-edge-nodes-csg-nrlnet.sh

# Invoke the setup script via sub-commands
#./bin/setup-central-and-edge-nodes.sh {create,start,destroy}

# Set up the environment (first create, then bootstrap, and finally start)
./bin/setup-central-and-edge-nodes.sh create
./bin/setup-central-and-edge-nodes.sh start_and_wait
./bin/setup-central-and-edge-nodes.sh start

# Once done, tear down the environment
./bin/setup-central-and-edge-nodes.sh destroy
```

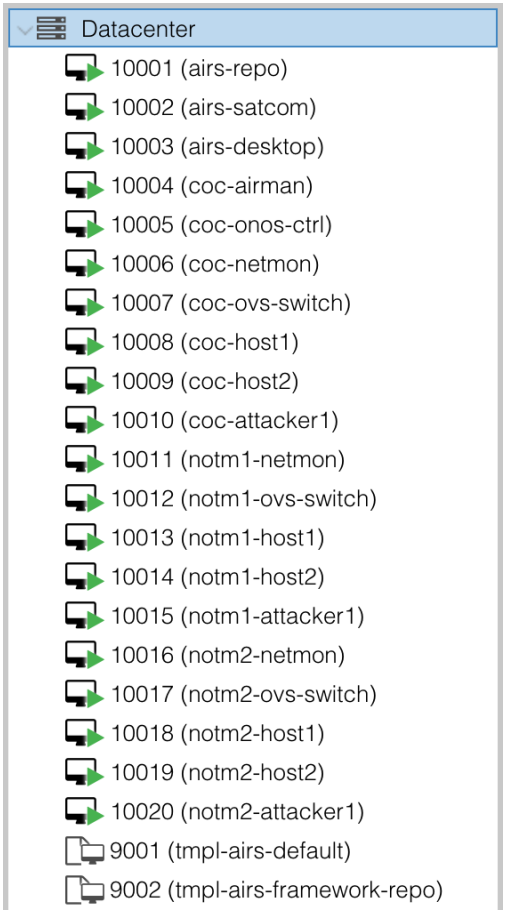

Fig. 2—This list of VMs is deployed when the setup script is invoked using a pre-defined set of configuration variables for the default AIRS C2/IoBT environment (`central-and-edge-nodes-marines-nrlnet`).

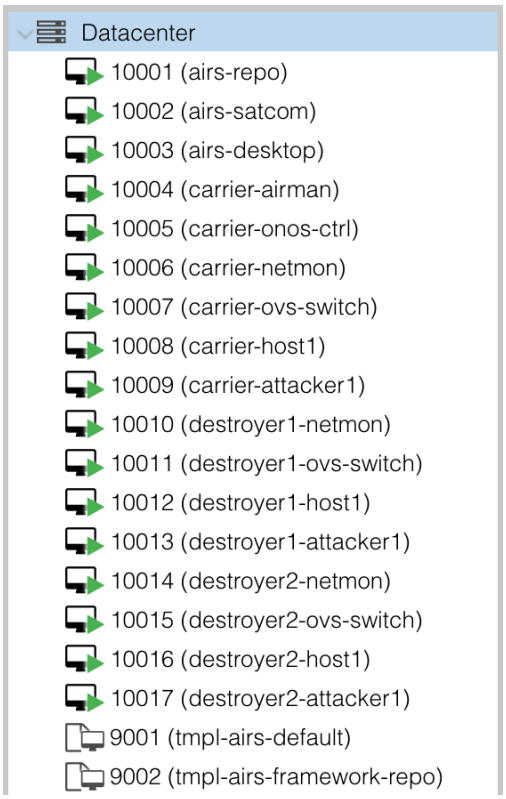

Fig. 3—This list of VMs is deployed when the setup script is invoked using a pre-defined set of configuration variables for a simplified Carrier Strike Group environment (`central-and-edge-nodes-csg-nrlnet`).

The resulting list of VMs after loading the default AIRS C2/IoBT configuration and running the setup script can be seen in Figure 2. Likewise, the list of VMs for a simplified version of the Carrier Strike Group configuration can be seen in Figure 3.

There are, of course, limitations to using a pre-configured environment. For example, it may not be possible to accurately model a given target system using the provided scripts and/or configurations (e.g., due to a complex topology or specialized requirements). However, in case a desired system cannot be effectively modeled using just the available scripts and configuration variables, achieving the desired outcome is not difficult. By following the example of the provided scripts, additional setup scripts can be developed that set up different topologies or entirely custom/one-off environments.

After successful deployment of an experimentation environment, the end-user/experimenter has a pristine instance of AIRS+SDN, fully configured in terms of the SDN controller, SDN switches, and AIRS components (including data plane components like network monitors, as well as management plane components like the security manager, planner, state manager, etc.). They also have access to a number of general-purpose VMs that are attached to SDN switch ports on the data plane. The exact number of general-purpose hosts depends on the selected configuration and any customization that was done to it. For example, the default Carrier Strike Group configuration creates five general-purpose data plane VMs per node (three mission-critical hosts and two attacker hosts per node), and the default AIRS C2/IoBT configuration creates three per node (two mission-critical, one attacker). To change the number of general-purpose VMs in an experiment, for

example to expand the C2/IoBT experiment with support for distributed denial-of-service attacks (i.e., with multiple attackers per node), one must simply edit the configuration to `export NUM_NODE_ATTACKERS=2` or a higher value. In terms of the network traffic that runs on the data plane (either attack traffic or "normal" background traffic), these general-purpose VMs are the place to orchestrate that type of traffic and the implementation of this is left to the experimenter. There are scripts for launching certain cyberattack scenarios on each VM under `/home/airs/bin/attacks`, and other software can be installed and configured as usual on a machine with a Debian-based OS.

Depending on what type of tactical network or system is being modeled, different workloads may need to be implemented on the data plane (on the aforementioned general-purpose hosts). For example, if we are modeling the network for a carrier strike group, it may be desirable to have a set of services running in the background so that we can measure the impact of cyberattacks and the autonomous responses. A set of enterprise services provided by NCES and CANES is summarized in [3] and includes, for example: E-Mail, Calendars, Chat, etc. To take the work done in this project a step further, a generic version of some of these services could be automatically deployed across the general-purpose VMs. For now, this step must be done by the end-user/experimenter after deployment and before commencing experimentation.

A SDN from the view of the SDN controller, ONOS [2], can be seen in Figure 4. A similar, but more comprehensive view, can be seen via the AIRS Mission Requirements Editor (MRE) GUI, shown in Figure 5. Both of these GUIs can be accessed by the end-user with no special configuration, using the `airs-desktop` VM that is deployed for convenience alongside the other VMs and contains a full graphical desktop environment with web browser, etc.

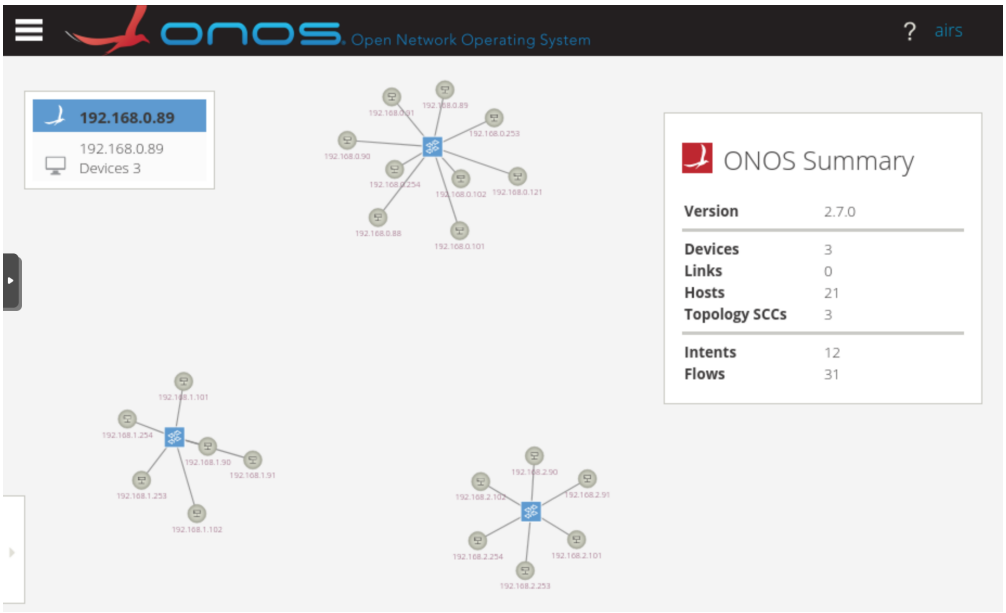

Fig. 4—A view of a data plane network from the perspective of the ONOS SDN controller.

In the context of a tactical network, long distance inter-node communication is typically implemented using SATCOM. In order to model the characteristics of inter-node traffic, we provide the `airs-satcom` VM, which is deployed alongside the other VMs and is inserted into the network traffic path and pre-configured

[2]ONOS - Open Network Operating System: `https://opennetworking.org/onos/`

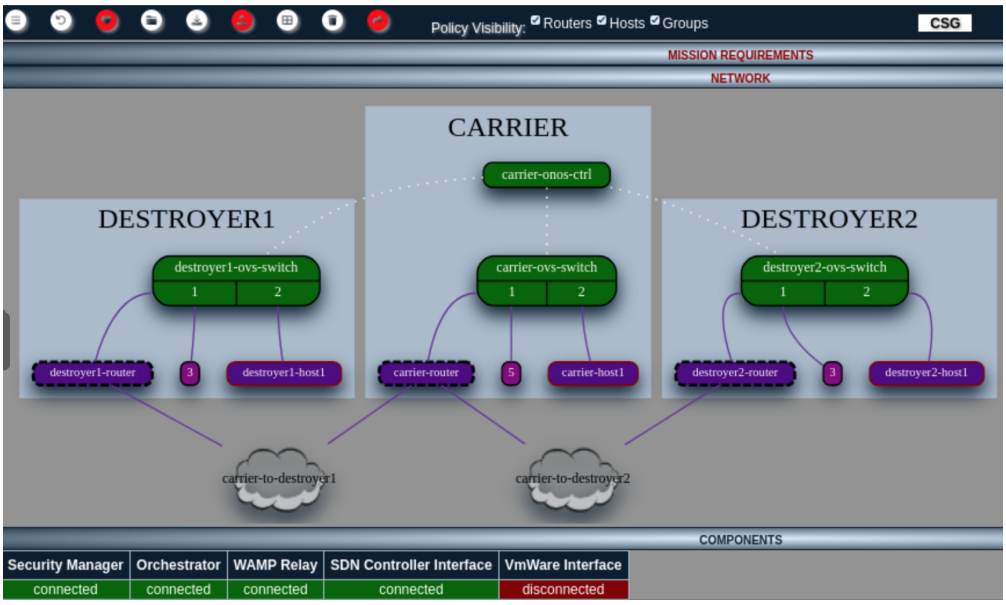

Fig. 5—A view of control plane and data plane networks, as seen in the AIRS MRE GUI.

to apply a delay to any network traffic that flows through it. This behavior can be customized further by the end-user/experimenter but provides a consistent starting point. The actual data transfer rate can be adjusted via configuration variables and sensible defaults have already been selected for each provided configuration. These default and recommended values are listed in Table 3. In the Carrier Strike Group environment, for example, the bottleneck data rate of the central node (the aircraft carrier) is recommended to be between 4 Mbps and 8 Mbps, while the bottleneck data rate of the edge nodes (support ships, e.g., cruisers or destroyers) is recommended to be between 256 kbps and 1 Mbps.

Table 3—Bottleneck data rates between nodes in two example experimentation environments.

Environment	Node	Min. rate	Default	Max. rate
AIRS C2/IoBT	*Central node*	0.5 Mbps	1000 kbps	2.0 Mbps
	Edge node	0.5 Mbps	500 kbps	1.0 Mbps
Carrier Strike Group	*Central node*	4 Mbps	4096 kbps	8 Mbps
	Edge node	256 kbps	512 kbps	1 Mbps

2.8 Results and demonstrations

As a result of the work done in this effort, additional experimentation of AIRS+SDN in the context of Navy and Marine Corps tactical networks can be carried out with lower effort and high degree of repeatability. Results like the ones presented in [1] can now be produced with much less effort on the part of the end-user/experimenter. What previously took hours or days to set up and configure can now be done automatically in minutes and with less chance of human error. Various examples of this new automated setup process have been recorded as short videos, in an effort to document one of the major outcomes of this project.

When carrying out experiments to test the effectiveness and performance of the autonomous cyber responses enabled by AIRS, it is necessary to have a library of cyberattack scenarios to test against. Here is a

list of cyberattacks that have already been implemented for AIRS to be tested against. This list is provided as a reasonable starting point, and additional attacks can be implemented and deployed on the general-purpose VMs or within the SDN control plane and deployed as part of an experimentation environment.

Table 4—Set of implemented cyberattack scenarios that can be used for test and evaluation.

Key	Description
`dp_dos`	Data plane denial-of-service attack
`dp_ddos`	Data plane distributed denial-of-service attack
`sdn_pktin`	Control plane SDN PACKET_IN flooding attack
`flowflood`	Control plane flow rule flooding attack

3. CONCLUSION AND FUTURE WORK

Currently, cyber responses are carried out by highly trained human cyber warriors. Even for low-level, relatively simple cyber events, it takes time to triage, investigate, and appropriately respond to the event. A basic response could take minutes to implement. For more complex cyber events, this could stretch to hours or days. Autonomous cyber response capabilities can significantly increase the tempo of cyber defense and lighten the load for cyber warriors by handling as much as possible at machine speed. Human expertise will still be necessary to monitor the system and/or approve the autonomous actions if desired. Autonomous cyber responses can take place within seconds of detecting cyberattacks or system failures. SDN is frequently deployed in the commercial sector to create agile networks that can be rapidly and reliably reconfigured through software. SDN together with autonomous cyber response capabilities can be a valuable technology for future tactical networks for the Navy and Marine Corps.

However, this is not technology that can be deployed immediately. It requires extensive testing, evaluation, and experimentation to prove the viability of AIRS+SDN in the context of tactical networks used by the Navy and Marine Corps. In this project, we have taken steps to get closer to this vision. We have redesigned the AIRS testbed while paying particular attention to automation and repeatability, to reduce the time and effort that is needed to set up these kinds of experiments.

The functionality developed in this project to support rapid and repeatable deployment of experimentation environments based on AIRS+SDN also expands the potential action space available to the planning components within AIRS itself. Some additional work would need to be done to fully integrate these actions into the planner's hierarchical operational model and the execution platform within the security manager.

Finally, developing additional experimental deployment topologies (modeled after the existing scripts that were developed as part of this effort), as well as developing additional cyberattack scenarios against which to test, could help mature this project even further.

ACKNOWLEDGMENTS

This project was funded by NRL through the NISE program. The AIRS project, which served as the foundation for the work done in this project, was funded by OUSD(R&E) between 2018 and 2021.

REFERENCES

1. A. Velazquez, B. Montrose, M. Li, J. Luo, M. Kang, S. Patra, and D. Nau, "ACRS4SDN: An Autonomous Cyber Response System for Software-Defined Networks," 2022. URL https://apps.dtic.mil/sti/citations/AD1166980.

2. S. Patra, A. Velazquez, M. Kang, and D. Nau, "Using Online Planning and Acting to Recover from Cyberattacks on Software-defined Networks," *Proceedings of the AAAI Conference on Artificial Intelligence* **35**(17), 15377–15384 (May 2021). URL https://ojs.aaai.org/index.php/AAAI/article/view/17806.

3. A. Deguzman, J. Ebken, N. Ho, R. Lai, D. Nunez, M. Raizada, P. Ross, and N. Tran, "Network Centric Communications for Expeditionary or Carrier Strike Groups," 2011. URL https://apps.dtic.mil/sti/citations/ADA555404.

4. M. Li and A. Velazquez, "Xenon Enterprise Scale Separation VMM Systems," 2019. URL https://apps.dtic.mil/sti/citations/AD1076348.