Standard Radio Nodes in the Defense Advanced Research Projects Agency Spectrum Collaboration Challenge

D. Alexander White Jr., J. Emery Annis, and Freemon F. Johnson

ABSTRACT

One of the major constructs of the Defense Advanced Research Projects Agency (DARPA) Spectrum Collaboration Challenge (SC2) framework was the standard radio node (SRN). The Johns Hopkins University Applied Physics Laboratory (APL) designed the Colosseum, the massive wireless test bed behind SC2, and the SRN within it. The SRN provided SC2 competitors a software-defined radio (SDR) as well as compute and storage node resources so that they could develop, test, and demonstrate collaborative intelligent radio network (CIRN) solutions. The SRN was designed to dynamically allocate and de-allocate competitors' container images while providing them complete access to and control of physically attached SDR and network resources. The SRN ensured the competition's security, integrity, and fairness and isolated each competitor's files and software. This article discusses the SRN's architecture and its supporting and commanding systems, including locally managed services and processes.

INTRODUCTION

The Defense Advanced Research Projects Agency (DARPA) designed the Spectrum Collaboration Challenge (SC2) to motivate competitors to develop collaborative dynamic spectrum-access algorithms for intelligent use of the radio frequency (RF) spectrum beyond a single radio. APL designed and built the collaborative intelligent radio network (CIRN) test bed, known as the Colosseum, for SC2. (For an overview of SC2 and the Colosseum, see the article by Coleman et al. in this issue.) By providing competitors access to equivalent hardware in the Colosseum, DARPA was able to evaluate CIRN algorithms independent of competitors' hardware. In the Colosseum, 128 nodes were available for competitor research. Each node, referred to as a standard radio node (SRN), comprised a server and a radio, as shown in Figure 1. The SRN is the main competitor platform and served as the competitors' interface to Colosseum resources. (See Figure 2 in the article by Coleman et al. in this issue for an illustration of the overall architecture of the Colosseum.)

The primary requirements of the SRN were to provide both the computing and graphics processing unit (GPU) system to enable competitors to develop and run advanced radio, collaboration, and machine learning algorithms for the competition as well as allow them full access to a software-defined radio (SDR) to transmit over the test bed. The SRN also had to be secure, protecting competitors' information and files while also ensuring that competitors could not access information and files on the APL network hosting the Colosseum.

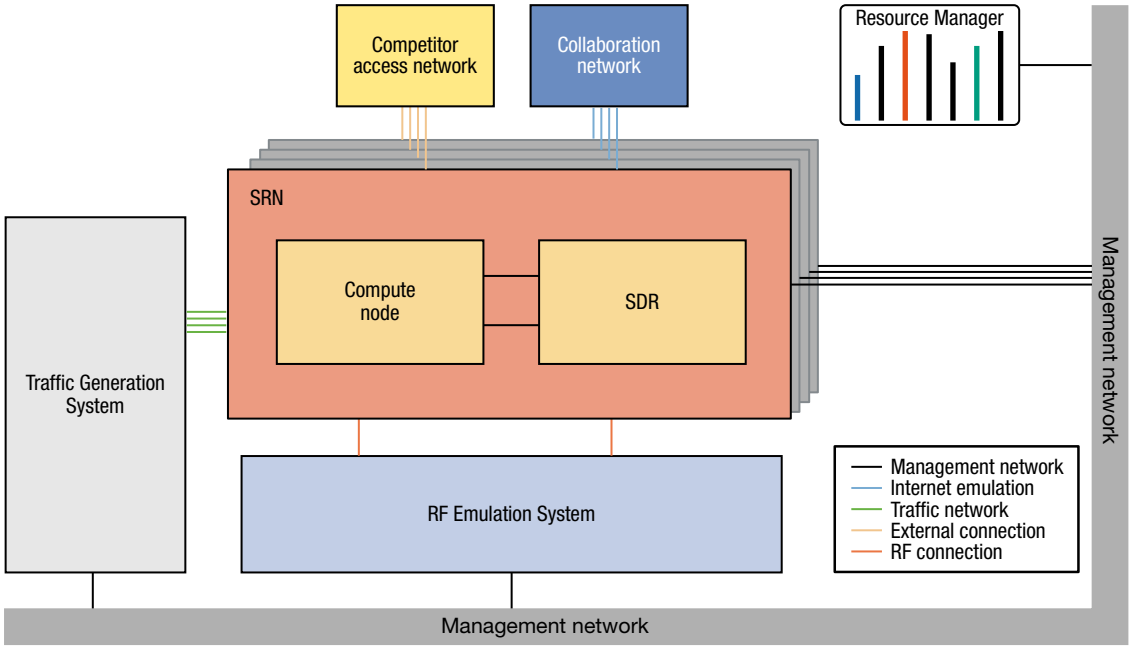

Figure 1. SRN architecture. Each SRN was a collection of general-purpose hardware and SDR components. Network interfaces provided connectivity to various Colosseum functions.

This article describes the overall SRN architecture in the Colosseum as well as the individual elements that enable remote research for collaborative radio networks, and it discusses the system's integrity and security.

SRN ARCHITECTURE

Each SRN included a general-purpose computing server with an Ethernet connection and USB interface to an SDR peripheral, in addition to Ethernet connections to the Resource Manager. The Resource Manager, as its names suggests, managed all resources across the Colosseum via its reservation system, ensuring fair resource allocation, automated orchestration of experiments, and verification of Colosseum operations. (See the article by Mok et al. in this issue for more detail on the Resource Manager.) Thus, the SRN's connection to the Resource Manager was persistent, enabling continuous monitoring of SRN status.

During an experiment, the SRNs had four additional interfaces: the collaboration network, the competitor access network (CAN), the Traffic Generation System, and the RF Emulation System. The collaboration network was an out-of-band communication channel among SRNs that allowed competitors to coordinate efficient use of the spectrum by dynamically reallocating resources based on load and criticality of transmission. The RF Emulation System provided the in-band (i.e., over-the-air) communication channel among the SRNs for real-time data exchanges. The CAN provided remote access during manual practice sessions for each of the SRNs in a reservation and provided access to the landing zone server, which mounts the competitor network-attached storage (NAS) for uploading of LXC images and downloading of logs and files. (See the article by Mok et al. in this issue for more detail on manual experiments.) The CAN interface was not allocated during competition scrimmages and formal events to restrict out-of-band collaboration to only the collaboration network. The Traffic Generation System delivered Internet Protocol (IP) traffic directly to the SRN with minimal routing and latency overhead since packet-level scoring was used to evaluate competitor algorithms. (For more information on scrimmages and events, see the article by Coleman et al in this issue. For more details on the RF Emulation System and the Traffic Generation System, see the articles by Barcklow et al. and Curtis et al., respectively.)

SRN CONTAINERS

To ensure competition security and integrity between, during, and after competitor experiments, the Resource Manager deployed competitor-supplied Linux containers to the SRNs (see Figure 2). A container is analogous to a virtual machine in that it provides full Linux functionality, enables portability, and eases deployment, but unlike a virtual machine, a container allows for direct access to host hardware. Also, a container is more lightweight than a virtual machine. This allows Linux users to easily create and manage system or application packages through the deployment of container images

on the bare-metal host. Furthermore, for SC2 the Linux containers provided security and integrity by confining competitors to their own files and information. Containers were used for two parallel instances on the SRN—the competitor algorithms and the collaboration server.

COMPETITOR CONTAINERS

Before an experiment, competitors uploaded a Linux container to the Colosseum and scheduled the experiment. During the experiment run time, competitors installed their server-side radio software, configured and programmed the SDR, and accessed approved network links (i.e., CAN, collaboration network, and Traffic Generation System). After each experiment, the SRN controller moved relevant logs and data to the competitor's shared network drive, the SRN was restored to a common baseline, and any competitor intellectual property was removed before the next experiment was executed (by the same or different competitor). Each SRN was permitted to host one competitor container at a time.

Since the container was the competitor's primary point of interaction with the Colosseum, the SRN needed to provide a first line of security. At a minimum, the SRN had to isolate the competitor container from the Colosseum's management systems, as well as ensure security and integrity among teams operating at the same time. One of the mechanisms used to provide this security was unprivileged containers.[1,2] Inside the container, the competitor had full access as root. However, outside the container, the same user had restricted privileges on the compute node. Even if the competitor managed to break out of the container, this setup prevented the competitor from having the ability to attack the underlying server or Colosseum network.

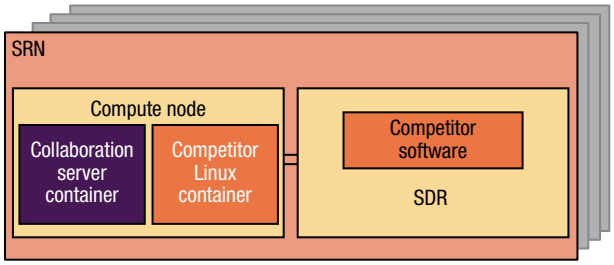

Figure 2. During an experiment, the Resource Manager allocated competitor-supplied Linux containers to the SRNs. The Linux container provided the competitor full access to the compute node and SDR.

COLLABORATION SERVER

In addition to the competitor container, each SRN could host a collaboration server (hosted in a parallel LXC container on the same server). Since only one collaboration server was required per experiment, the competitors specified the collaboration server host SRN with

COMPETITOR ACCESS AND COLLABORATION NETWORK

The competitor access and collaboration network was a single physical interface but was configured as many trunk ports to serve multiple functions and provide security and integrity during the competition. The two functions of this interface were remote competitor access and internet emulation for the collaboration network.

The CAN was accessed from the public internet through a Linux Secure Shell (SSH) gateway and firewall. Each competitor was provided a private internal subnet that connected each of their SRNs for a single reservation. The Linux SSH gateway was responsible for restricting access to each SRN based on the competitor's team ID.

Each SRN was configured with unique virtual local area networks (VLANs) specific to each competitor. During an SRN's idle state, the network remained shut down with no IP address configured. At the time of allocation, the SRN controller was informed of a reservation's team ID, which was used to configure a team-specific IP and VLAN and attach to the competitor's LXC container. This ensured that all SRNs reserved by a single team could communicate with each other and could be accessed from the public internet while also ensuring that no other team could access these SRNs even during simultaneous reservations.

The collaboration network provided competitors with an internet emulation network connected to at least one SRN per reservation. The internet emulation allowed variations of bandwidth, latency, and jitter to be applied to each interface, which increased the fidelity and forced competitors to collaborate using emulated wide area network (WAN) links. The collaboration network also connected the collaboration server. There were 128 collaboration servers, one corresponding to each SRN. At allocation time, one was selected as the server for each reservation and was restricted to only nodes in that reservation. The collaboration network was dynamically configured and connected to competitor containers using the SRN ID of the compute node that was hosting the collaboration server. This provided both for segmentation for simultaneous reservations and for competitor collaboration network connectivity without any required routes on competitor containers. The collaboration network was also monitored for competition scoring purposes so that scorers could track how often competitors attempted to collaborate and whether their requests were acknowledged and acted on. Packet captures recorded all traffic on the collaboration network for each reservation. After de-allocation, the files were copied offline for archiving and analysis.

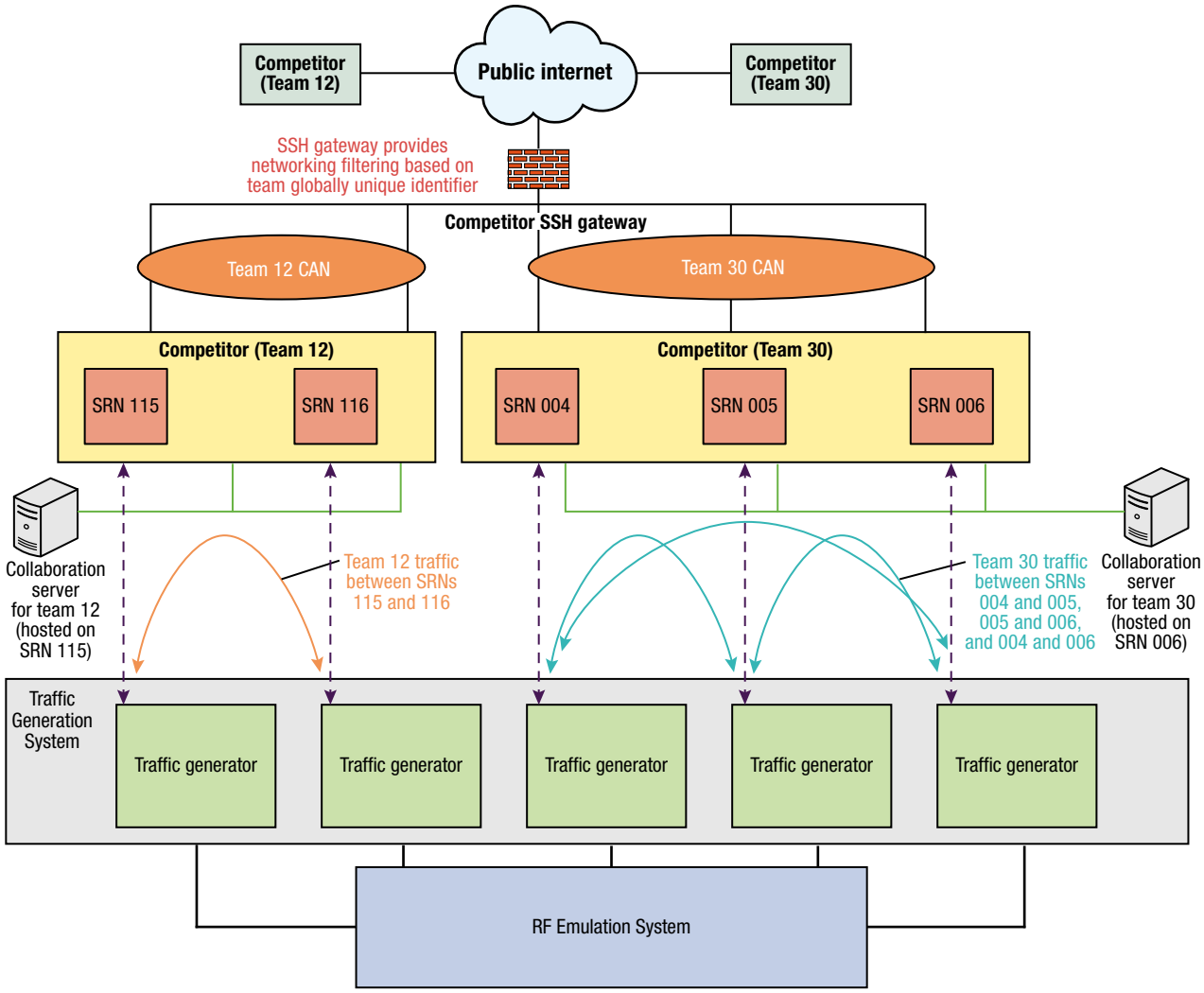

Figure 3. Illustration of SRN connections to the CAN, the collaboration network, and the traffic network for two teams. Team 12 and team 30 were in manual experiments at the same time. Team 12 was allocated SRNs 115 and 116, while team 30 was allocated SRNs 004, 005, and 006. A collaboration server was allocated to each team, and both the CAN and the collaboration gateways were connected. Further, traffic was connected to each team's LXC container and was passing over the wireless channel emulator.

their experiment request on the Colosseum website. The collaboration server acted as a publish/subscribe point for competitors so that competitors could register their IP address and query for a list of other competitors to facilitate real-time spectrum coordination.

TRAFFIC NETWORK

During the competition, competitors were scored on their ability to transmit IP traffic over the radio network. Thus, IP traffic had to be delivered to each SRN with minimal routing and latency overhead, and it had to be collected and measured in a similar fashion. Since the competitors were responsible for routing IP traffic over the wireless channel emulator, each SRN served as the default gateway for the sources and sinks of traffic. This resulted in the design choice of 128 independent traffic network VLANs corresponding to the 128 SRNs that would operate in the SC2 Colosseum. Each SRN was attached to only one traffic VLAN, and when an SRN was allocated, this Linux sub-interface was attached to the competitor container as an LXC "phys" interface. This means that the competitor essentially owned the management of the traffic interface during the reservation and could adjust bandwidth, duplex, or traffic-shaping characteristics as needed. It also ensured that there was minimal latency in traffic delivery and no unnecessary routed hops in the path. The traffic was sourced and synced by blade servers.

A sample experiment is illustrated in Figure 3 for two teams (team 12 and team 30) in concurrent manual experiments on parallel SRNs. Team 12 was allocated SRNs 115 and 116, while team 30 was allocated SRNs 004, 005, and 006. A collaboration server was

allocated to each team, and both the CAN and the collaboration gateways were connected. Further, traffic was connected to each team's LXC container and was passing through the RF Emulation System. If this were an automated experiment, such as during scrimmages and match sessions, the CAN would have remained disconnected for all SRNs and only one collaboration gateway would have been connected for each team. Further, the teams would have been placed into a single experiment and would have been allocated only one collaboration server for the entire experiment.

SRN CONTROLLER

The actions of the SRN host were directed by the Resource Manager and orchestration engine running on the Colosseum's management network. These actions included allocation and de-allocation of competitor reservations, networking configuration and connectivity, collaboration server configuration, and all administrative system monitoring. For the competitor, the SRN maintained the state of the container during the reservation, verified that the SDR was in a known state before and after reservations, and provided a Command Line Interface (CLI) for competitor manual experiments.

The SRN ran a local process, known as the SRN controller, that performed all management actions. It was commanded by the Resource Manager through a RESTful (representational state transfer) API. Some administrative functions on the SRN required root access, and this was provided by a separate process known as the Root Helper (see Figure 4).

The SRN controller was a system service that ran on each SRN bare-metal host and, as its name implies, controlled what the SRN did. The SRN controller was only in charge of its local SRN and only received RESTful formatted commands from the Resource Manager and any locally instantiated competitor container. It did not coordinate with any other SRN controller or other services within the Colosseum. The development team made the important design decision that the SRN controller would always be subordinate to the Resource Manager to ensure that system state was maintained. If the states of the Resource Manager and the SRN controller were out of sync, the SRN controller would align its state with that of the Resource Manager. Typically this process included ending any existing reservations, destroying hanging containers, and starting any processes required to match the state indicated by the Resource Manager.

The SRN controller was the parent process for the competitor's container. To limit the attack surface if a malevolent competitor broke out of their container, the SRN controller was launched with limited privileges. However, some of its tasks, such as ensuring that all log files from a competitor were removed from the local SRN after a reservation, required root-level access, so a second service, named Root Helper, was created with root access. Root Helper provided REST end points internally on the compute node and was accessible only by the SRN controller. These end points allowed the SRN controller to perform certain tasks that required root access.

The SRN controller had additional responsibilities during automated experiments since the competitor container did not have access to the CAN. To permit data to be saved from the containers, a temporary log directory was created on the compute node and mounted to the competitor's container. After the experiment completed and the competitor's container was destroyed, the SRN controller, with the help of the Root Helper, changed the ownership the log directory and moved it into the competitor's shared drive. The size of compute node's log partition was limited to 25 GB to prevent a single test from filling up the competitor's shared drive.

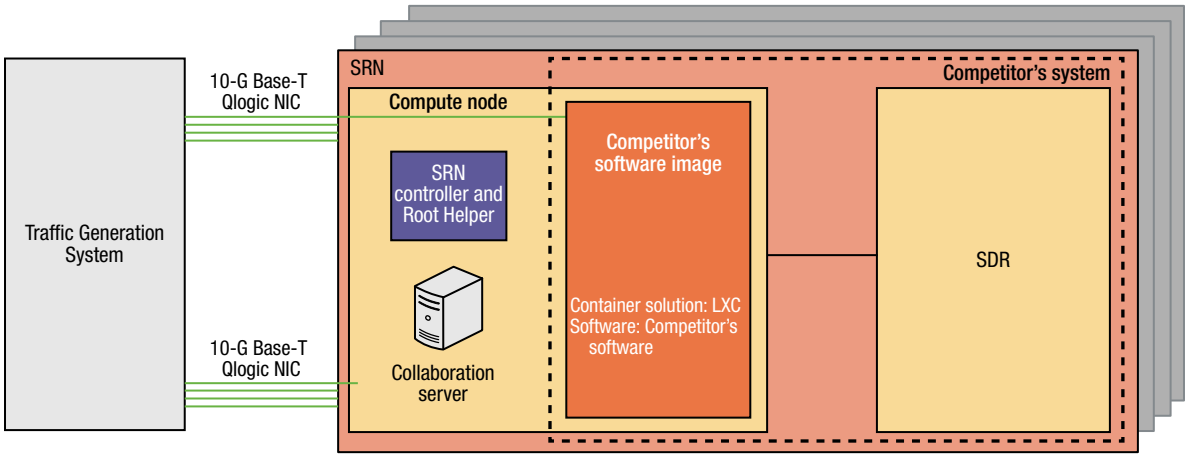

Figure 4. The SRN ran a local process, known as the SRN controller, that performed all management actions. Some administrative functions on the SRN required root access, and this was provided by a separate process known as the Root Helper.

RESERVATION INDEPENDENCE

One of the requirements was to ensure that competitor systems were always allocated into a predictable state and that there were no lingering configuration or data left from previous reservations. Use of a container provided much of the predictable state as long as the SRN controller destroyed and deleted all containers and log directories between reservations. The SRN controller was designed to allow only one reservation to be active at a time. If the SRN controller received an allocation command from the Resource Manager while already in a reservation, it immediately ended the current reservation and restored the compute node to baseline, including killing any hung processes and deleting any remaining competitor files.

SHARED DRIVE

A network file system (NFS) shared drive was created on a NAS to provide a centralized location for competitors to store their data. The shared drive was mounted to the landing zone server, which allowed competitors to access the shared drive independently of an active reservation. The drive was used to store images for reservation container allocation, miscellaneous competitor files, log files from each SRN, traffic generation statistics, and packet captures from the collaboration interface. During manual experiments, the shared drive was mounted to the competitor's containers. During automated experiments, competitors were provided a local partition on the SRN to store logs that would be copied to the NAS at de-allocation time.

RADIO COMMAND AND CONTROL API

Since all automated experiments were launched and commanded without user interaction, competitors required a way to trigger state changes within their containers. State changes included the allocation, run time, and de-allocation state of the container as well as initialization of the radio. It could also trigger the launching and stopping of RF and traffic scenarios.

To communicate with competitor systems, the SRN controller called a set of scripts on the competitor's container to signal state changes. These scripts were called the Radio API scripts. APL provided samples of Radio API scripts, but competitors could customize these scripts. The SRN controller expected a standard format returned from the Radio API scripts, but failure of customized scripts to adhere to this standard did not result in SRN operation failure.

The two best examples of Radio API scripts are start.sh and stop.sh. Start.sh was called when all the components in the Colosseum had been initialized for a reservation and at the exact moment that the traffic and RF scenarios were starting. The example start.sh script simply returned a code indicating acknowledgment. This script provided a hook for the competitor to start programs at the beginning of a scenario instead of just at the initialization of the container. Stop.sh was called at the end of the reservation and informed the competitor that the container would be destroyed in 2 minutes so that they could copy any files they wished to keep to the log directory in the container.

CONCLUSION

SC2 challenged competitors to develop and demonstrate novel dynamic and collaborative spectrum-sharing techniques. The SRN served as the competitor's primary interface for all development and testing of these algorithms and radios, as well as the range for competing in events. The SRN was secure and ensured competition integrity and fairness, as well as isolation of all competitor files and software at all times. The SRN was dynamic, supporting many reservations for many competitors over the competition period. Using containers for competitor images was integral to ensuring portability and also provided security and isolation. Multiple networks for each SRN provided isolation while also supporting collaboration and traffic generation. The successful SRN design offers much insight for the development of other test beds with dynamic instantiation and automation requirements involving multiple users and use cases.

ACKNOWLEDGMENTS: We thank Paul Tilghman (DARPA SC2 program manager) and Craig Pomeroy and Kevin Barone (Systems Engineering and Technical Assistance at DARPA) for their invaluable collaboration and support. We also thank the many APL SC2 contributors, whose names are listed on the inside back cover of this issue of the *Digest*. This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions, and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the US government.

REFERENCES

[1]M. Byazit. "Linux containers – A comparison of LXC and Docker." Robin. Jul. 26, 2016. https://robin.io/blog/linux-containers-comparison-lxc-docker/.

[2]S. Graber. "Custom user mappings in LXD containers." Stéphane Graber's website. Jun. 15, 2017. https://stgraber.org/2017/06/15/custom-user-mappings-in-lxd-containers/.

D. Alexander White Jr., Asymmetric Operations Sector, Johns Hopkins University Applied Physics Laboratory, Laurel, MD

Alex White is a Senior Professional Staff member and acting assistant supervisor of the Operational Systems Section in APL's Asymmetric Operations Sector. He has a BS in electrical engineering, from Washington University. Alex has over 20 years of systems engineering experience focused on military wireless communications systems. He has expertise in network architecture, performance, security, and integration and often consults with customers on network design and analysis. Alex also has deep knowledge of wireless communications protocols and technology development and experience leading and managing engineering teams. While leading software teams supporting DARPA's SC2 program, he integrated LXC deployment code to instantiate competitors' designs and developed code to start, collect data during, and clean up individual reservations. His email address is alex.white@jhuapl.edu.

J. Emery Annis, Asymmetric Operations Sector, Johns Hopkins University Applied Physics Laboratory, Laurel, MD

Emery Annis is a communications systems engineer at APL with a BS in electrical engineering from the University of Houston. He is currently studying for a master's in electrical engineering with a focus in communications from Johns Hopkins University. Emery has contributed to multiple efforts to develop network emulation environments including an Army Wideband Global Satellite (WGS) control system, the DARPA SC2 Colosseum, and an end-to-end networking analysis framework. He regularly contributes to efforts involving RF spectrum coexistence analysis and is interested in applying concepts from both the RF communications domain and IP networking to develop new standards and architectures centered on an intelligent control plane and network maneuver. His email address is emery.annis@jhuapl.edu.

Freemon F. Johnson, Asymmetric Operations Sector, Johns Hopkins University Applied Physics Laboratory, Laurel, MD

Freemon Johnson was a senior software engineer in APL's Asymmetric Operations Sector. He has a BS in electrical engineering from New Jersey Institute of Technology, an MS in communication and information studies from Rutgers University, and an MS in computer science from Johns Hopkins University. Freemon has over 21 years of experience that encompasses software development and information assurance. His experience can be categorized as follows: 15 years of systems engineering, 11 years of software development, and 4 years of agile development using Scrum/Kanban process methods. His last role at APL focused on multiple research tasks across software development, data engineering, and networking engineering. His work included machine and deep learning application development for cyber analytics, orchestration/automation, and security research for SDN and NFV. Freemon was a primary software developer for the DARPA SC2 Colosseum platform for the standard radio node (SRN) subsystem.