Radio Frequency Emulation System for the Defense Advanced Research Projects Agency Spectrum Collaboration Challenge

Daniel R. Barcklow, Lian E. Bloch, Stephen W. Sweeney, Brian E. Ahr, William J. La Cholter, Samuel Berhanu, Hakeem S. Bisyir, Jarriel D. Cook, and David M. Coleman

ABSTRACT

The Johns Hopkins University Applied Physics Laboratory (APL) designed and built a wireless communications research test bed, called the Colosseum, for the Defense Advanced Research Projects Agency (DARPA) Spectrum Collaboration Challenge (SC2). SC2 aimed to motivate research into autonomous wireless communication systems to uncover a new paradigm for managing the oversubscribed radio frequency (RF) spectrum. This article describes the Colosseum's RF Emulation System, which mimicked real-world phenomenon such as propagation delay, Doppler shift, and power attenuation between 128 two-channel radios, or 65,536 wireless communications channels. The RF Emulation System emulated isolated virtual environments across multiple concurrent experiments, enabling challenge competitors to research, develop, and test next-generation artificial intelligence solutions for wireless network systems.

INTRODUCTION

The Defense Advanced Research Projects Agency (DARPA) introduced the Spectrum Collaboration Challenge (SC2) in 2016, seeking to find new capabilities that enable more efficient allocation and use of the contested RF spectrum by taking advantage of artificial intelligence (AI). Central to the challenge was the Colosseum, a research test bed where competitors could research and develop their next-generation autonomous wireless communication systems, and a series of scenarios designed to mimic real-world challenges a network of collaborative autonomous radios would have to contend with. (See the article by Coleman et al. in this issue for an overview of the Colosseum and SC2 scenarios.)

Designed and built by APL, the test bed consisted of 128 two-channel radios and several subsystems, one of which was the RF Emulation System. As its name suggests, the RF Emulation System emulated realistic RF propagation effects among the full complement of antennae (65,536 wireless channels). The system provided users with the ability to emulate radio wave propagation in a 1-km by 1-km region with an excess delay of 5.12 μs. This was sufficient to emulate the first four taps of an LTE urban propagation model with vehicles moving less than 30 miles per hour (at f_c = 1 GHz) and first responder line-of-sight UHF communications during a wildfire, using full-motion video from overhead drones. The RF Emulation System, shown in Figure 1, received requests from the Resource Manager over the Colosseum intranet, and the Resource Manager received messages regarding resource availability and status back from the RF Emulation System Manager. (See the article by Mok et al. in this issue for more on the Resource

Manager.) The server read and processed scenario files, and the wireless channel emulator received a real-time data stream of scenario parameters.

This article discusses the design and performance of the two processes hosted on the application server: the RF Emulation System Manager and the channel update process (CUP). The wireless channel emulator, built and delivered by National Instruments, is outside the scope of this article; see Ref. 1 for more information on the wireless channel emulator.

REQUIREMENTS

The RF Emulation System had three major requirements for operation within the Colosseum.

1. **Provide RF channel emulation for each experiment**—The RF Emulation System had to emulate an RF channel between transmitter–receiver pairs to mimic the inherent pairwise interactions of independent radios in the real world. Emulation required a one-time configuration of the radios and a real-time data stream of parameters in the wireless channel emulator. The one-time configuration included radio parameters for standard radio nodes (SRNs) and the wireless channel emulator: center frequency, sampling rate, and antenna gain settings. (See the article by White at al. in this issue for more information on SRNs.) These parameters were set at the start of an experiment and remained constant throughout the experiment. Continuous application of channel updates followed the one-time configuration. These updates were applied at a rate of 1000 Hz to emulate the time-varying RF channel between all transmitter–receiver pairs in the experiment.

2. **Handle concurrent experiments**—The RF Emulation System had to handle multiple concurrent experiments. To be valid and repeatable, each experiment had to operate on a noninterference (i.e., isolated) basis. This meant that a transmitter or receiver in use in one experiment could not be used in another experiment at the same time.

3. **Monitor and report status**—The RF Emulation System monitored the status of the emulation hardware and reported it back to the Resource Manager. (See the article by Mok et al. in this issue for more information on the Resource Manager.) This status report included the start time of a scenario, an acknowledgment of the radios in use by a scenario, and when the scenario stops, whether from error, by the user, or because it completed under normal cases. Since the Colosseum Resource Manager was the single persistent interface to the user, the RF Emulation System Manager reported status back to the Resource Manager to maintain a list of available resources and to verify correct operation within the RF Emulation System for each experiment.

ARCHITECTURE

As shown in Figure 2, the RF Emulation System consisted of three processes/components: (1) the RF Emulation System Manager, (2) the CUP, (3) and a wireless channel emulator. APL designed and wrote the manager and the CUP, and they resided on the RF Emulation System server. During phase 1 of the program, APL integrated the wireless channel emulator. (See the articles by Freeman et al. and Plummer and Taylor in this issue for more information on project phases.) Each component is discussed in detail below.

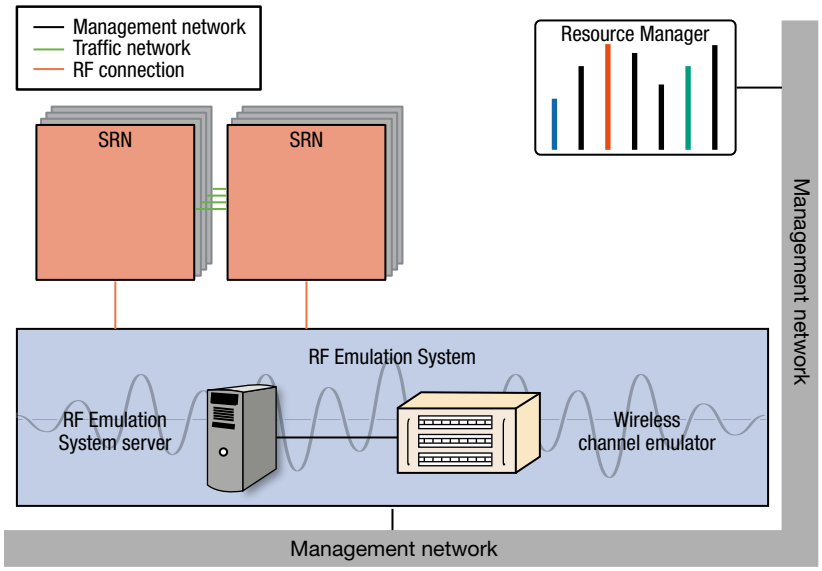

Figure 1. High-level diagram of the RF Emulation System. This Colosseum subsystem mimicked real-world RF phenomenon such as propagation delay, Doppler shift, and power attenuation between 128 two-channel radios, or 65,536 wireless communications channels.

RF Emulation System Manager

The RF Emulation System Manager had three interfaces: an Ethernet interface with the Resource Manager over the Colosseum intranet, a loop-back interface with the CUP residing on the same server with the RF Emulation System Manager, and an Ethernet interface with the wireless channel emulator. The two-way

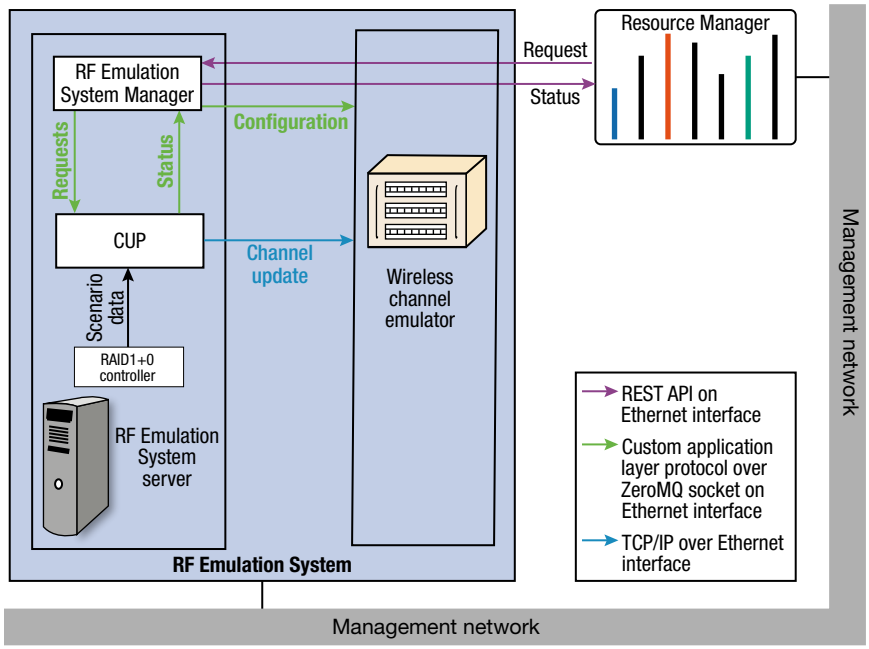

Figure 2. RF Emulation System architecture. The system consisted of three components: the RF Emulation System Manager, the CUP, and a wireless channel emulator.

connection between the RF Emulation System Manager and the Resource Manager used REST (representational state transfer) application programming interface end points.[2] The RF Emulation System Manager received requests from the Resource Manager, while the Resource Manager received messages regarding resource availability and status from the RF Emulation System Manager. The Ethernet interface between the RF Emulation System Manager and the wireless channel emulator used a custom application layer protocol over a ZeroMQ[3] socket connection. This interface was used to configure the radios within the wireless channel emulator, intended to be used at the beginning of an experiment. The third interface was the control and monitoring interface between the RF Emulation System Manager and the CUP, implemented as a ZeroMQ socket connection over a loop-back interface on the shared server. The RF Emulation System Manager is discussed in more detail later in this article.

Channel Update Process

The CUP had three interfaces, shown in Figure 2: the loop-back interface on the shared server with the RF Emulation System Manager, a RAID (redundant arrays of inexpensive disks) controller optimized for high-performance (capacity and latency) reading, and an Ethernet interface with the wireless channel emulator. As noted, the RF Emulation System Manager controlled and monitored the CUP by using a custom application layer protocol over a ZeroMQ socket connection on the loop-back interface. For the largest experiments with 128 SRNs, the RAID controller's required read rate exceeded 14 Gbps to meet the coherence rate of the SC2 scenarios (1 kHz). To achieve this high read speed, 10 solid-state drives with individual read speed ratings over 12 Gbps (Ref. 4) were combined in a RAID1+0 configuration.[5] Altogether, the storage array provided 10 TB of usable capacity and was able to maintain read speeds over 16 Gbps. Similarly, the high-capacity interface between the CUP and the wireless channel emulator exceeded 24 Gbps with 128 SRNs. Four 10.0-Gbps Ethernet connections using Transmission Control Protocol/Internet Protocol (TCP/IP) were established between the RF Emulation System server and the wireless channel emulator to meet this requirement while providing a continuous and reliable stream of channel updates for experiments. The CUP is discussed in more detail later in this article.

RF SYSTEM MANAGER

The RF System Manager was the liaison between the Resource Manager and the wireless channel emulator (see Figure 2). In this role, the RF System Manager had to configure radios in the wireless channel emulator ahead of the experiment, initiate and monitor the channel update process during the experiment, and return the radios in the wireless channel emulator to a default state at the end of the experiment. Since the wireless channel emulator was a shared resource, the RF System Manager had to execute its work across multiple independent and concurrently running experiments. The configuration, execution, and reset of the wireless channel emulator during an experiment was encapsulated in a user session within the RF Session Manager. The collections of sessions were managed by the RF System Manager.

As part of an experiment on the Colosseum, the Resource Manager invoked the RF Emulation System through a request into the RF Session Manager. After validation of the request, the RF Session Manager created a session for the experiment. The session was a finite state machine (see Figure 3) with 10 states that configured, updated, and reset the wireless channel emulator:

1. **Initialization**—In the initial state, the session was configured using the parameters within the request from the Resource Manager. These parameters

included the reservation identifier, the reference to the wireless channel emulator, and an object representing the session, which held the start and stop times as well as the competitor identification and a mapping of the radios to their current statuses. Additionally, a socket connection was established with the wireless channel emulator to enable radio configuration in the next state. Upon a successful port opening, the session transitioned to the Radio Configuration state. If any errors occurred, the session immediately transitioned to the Error Before CUP Began state.

2. **Radio Configuration**—Before an experiment, the radios in the wireless channel emulator inputs that corresponded to the individual competitor SRNs had to be configured to the appropriate sampling rate, center frequency, and gain settings on the transmit and receive ports. If errors were present during configuration, the session either transitioned to an error state or stayed in the Initialization state until the appropriate changes were made and the session could be verified and validated. If configuration was successful, a trigger was sent to move the session into the next state. At that time, the radios had been configured with the proper settings and the RF Session Manager had updated its mapping of radios to reflect the radios being used by this session.

3. **Invoke CUP**—At this stage, the CUP was notified of the request via an inter-process communication link. The request was populated with the session parameters including the experiment start time (used across the Colosseum to synchronize services; see the article by Mok et al. in this issue for details), the path to the scenario, and which radio nodes were being requested. If errors were present when interacting with the CUP, the session immediately transitioned to the error state; otherwise it moved to the next state.

4. **CUP Acceptance**—The CUP was given an opportunity to validate (or reject) the experiment request from the RF Session Manager before the experiment started. If the request was validated and accepted, the RF generation remained idle until the experiment start time (at which time the session would transition to the next state). If the CUP rejected the request, the session immediately transitioned to the error state.

5. **Error Before CUP Began**—When an error occurred before the CUP was invoked, the experiment was simply abandoned and the session transitioned to the closeout state.

6. **CUP Active**—At the experiment start time, the CUP triggered the session to move to its next state. This state allowed the RF Session Manager to monitor the status of the experiment. The session

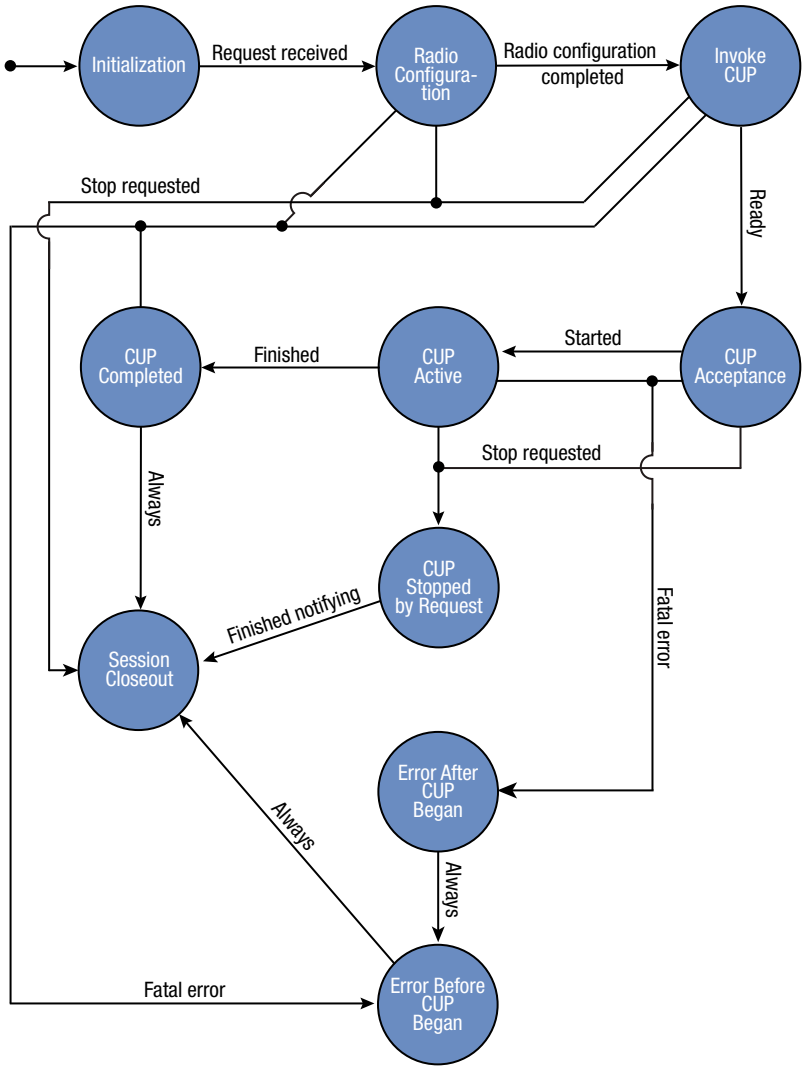

Figure 3. Session state machine. During an experiment on the Colosseum, the Resource Manager invoked the RF Emulation System through a request into the RF Session Manager. After validation of the request, the RF Session Manager created a session for the experiment. The session was a finite state machine with 10 states that configured, updated, and reset the wireless channel emulator.

remained in this state until the experiment expired, the user requested a stop, or an error occurred.

7. **CUP Completed**—When the experiment duration was complete, the CUP triggered the session to move to its completed state. The session entered this state only if the CUP had completed the entire experiment with no errors. The session always transitioned to the closeout state.

8. **CUP Stopped by Request**—In manual mode, users were able to asynchronously request an immediate expiration of the experiment. To account for this action, a request through the Resource Manager into the RF Session Manager to end an experiment early triggered the session to transition into this state. In this state, the CUP was immediately suspended and the wireless channel emulator channel was abandoned for that experiment. No wireless channel state was guaranteed while in this state. The session always transitioned next to the closeout state.

9. **Error After CUP Began**—When an error occurred during the experiment, the CUP triggered the session to transition to this error state. In this state, the RF Session Manager was able to collect valuable status information, such as last channel update applied or the radio in fault for error handling by the Resource Manager. After the information was collected, the session transitioned to the closeout state.

10. **Session Closeout**—The wireless channel emulator had to be reset before the session concluded. This included configuring the radios to a default state (e.g., center frequency, gain, and sampling rate) and clearing the wireless channel conditions that existed between radios. By taking this step, the session ensured that the radios were available for the next user and that the channels between radios were disconnected. Additionally, the RF Session Manager updated its internal radio mapping to reflect that these radios were no longer in use, so that checks for future session radio requests were allowed. The session immediately terminated after this work was complete. If an error occurred in this state, the Resource Manager was notified and the radios and channels were removed from the pool of available resources.

When a session transitioned to a new state, the RF Session Manager logged the change to both system logs and Splunk. (See the article by Plummer and Taylor in this issue for more details on Splunk.) The RF Session Manager ran on the Colosseum as a service—hence the necessity for system logs as opposed to standard output. These logs were very important, as they were the only place that showed what happened within the system immediately before an unforeseen error state occurred. Each log was denoted with a reservation identifier that was unique to a specific session. This information was helpful in the debugging process when an error occurred during the completion of a session, as the logs could be filtered by that unique identifier. In addition to the reservation identifier, each log included a timestamp and recorded the state transition occurring.

CHANNEL UPDATE PROCESS

Accurate representation and emulation of the physical environment for wireless communication required three components: (1) RF analysis of the environment, (2) a wireless channel emulator capable of representing the environment in the digital domain, and (3) a process to update the parameters of the wireless channel emulator. The output of the RF analysis stage was a power-delay profile (PDP)[6] for each wireless channel (transmitter–receiver pairing) across the entire experiment duration at regular intervals (often on the order of the channel coherence time[6]). The power-delay profile $y(t)$ was the convolution of the transmit power, $x(t)$, and the channel response, $h(\tau, t)$:

$$y(t) = x(t) \oplus h(\tau, t) \tag{1}$$

The channel response was a combination of slow fading, $s(t)$ such as atmospheric attenuation (i.e., oxygen and water vapor) and multipath components, $c(\tau, t)$:

$$h(\tau, t) = s(t) * c(\tau, t) \tag{2}$$

As such, a wireless channel emulator imparted a complex low-pass filter on the transmitted signal.

For the DARPA SC2 program, the PDP was instantiated in the wireless channel emulator as a finite impulse response (FIR) filter with four complex taps and a tapped delay line (TDL).[1] The approximated PDP with four complex coefficients and four delay values at a resolution of 1 ms was computed offline and stored on disk for the entire duration of each experiment. When an experiment was executed, the corresponding wireless channel parameter (WCP) file was read from disk and updates were sent to the wireless channel emulator at regular intervals. In this manner, an emulated wireless channel was established between a transmitter and receiver. To accommodate the 128 two-channel software-defined radios (SDRs) in the Colosseum, the RF Emulation System had to support up to 65,536 channel updates on each 1-ms interval.

Figure 4 shows the relationship between the radio channel pairs and their corresponding PDP that represented the emulated RF channel between the two radios.

To provide the flexibility to map any of the 256 source radio channels to any subset of the 256 potential des-

tination radio channels, a matrix of interconnections was employed. Each junction in the grid of interconnections represents a filter output. As described above, this FIR filter made realistic channel emulation between radio nodes possible. Figure 5 illustrates the interconnections between source radio channels and destination radio channels in the wireless channel emulator.

The following subsections detail the CUP phases and the novel algorithms used to meet the 1-ms interval for the world's largest channel emulator.

Data Storage

The WCP file contained the time series for the finite impulse response and tapped delay line complex taps. That is, for each interval (1 ms) the file specified four real (I), imaginary (Q), and delay (D) components for the emulated channel between each transmitter and receiver pair in the experiment. An experiment could include up to 128 two-channel radios and a maximum of 65,536 channel updates at 1-ms intervals. For example, an experiment in SC2 could have had a run time of 30.0 minutes, or 1.18 × 1011 channel updates across all 256 radios. Given the number of updates required for each experiment, data needed to be stored efficiently. The four real and imaginary components were 16-bit fixed point values representing the range (–1, +1), and the four delay components were nine bits each representing 1-μs increments up to 5.12 μs of total delay.[1] After data compression, each channel update—a single entry in the matrix shown in Figure 4—had a size of 20.5 bytes. Therefore, the exemplar experiment

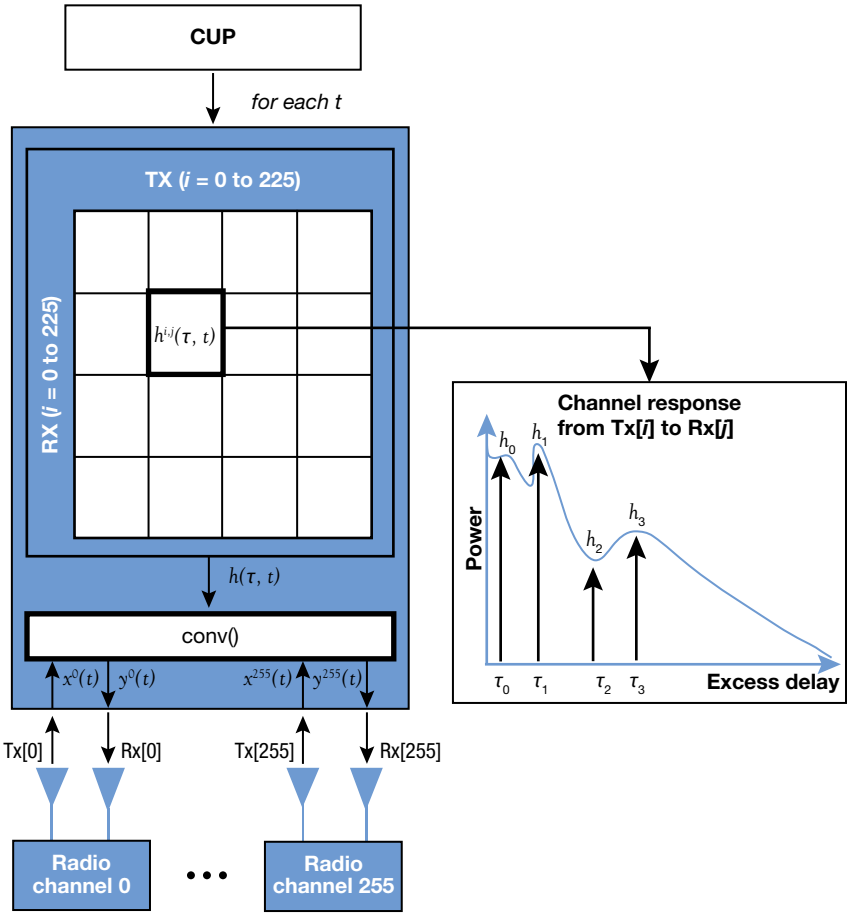

Figure 4. Wireless channel emulator's PDP matrix. Each junction in the grid contained the PDP that models the wireless channel between its corresponding radio channels.

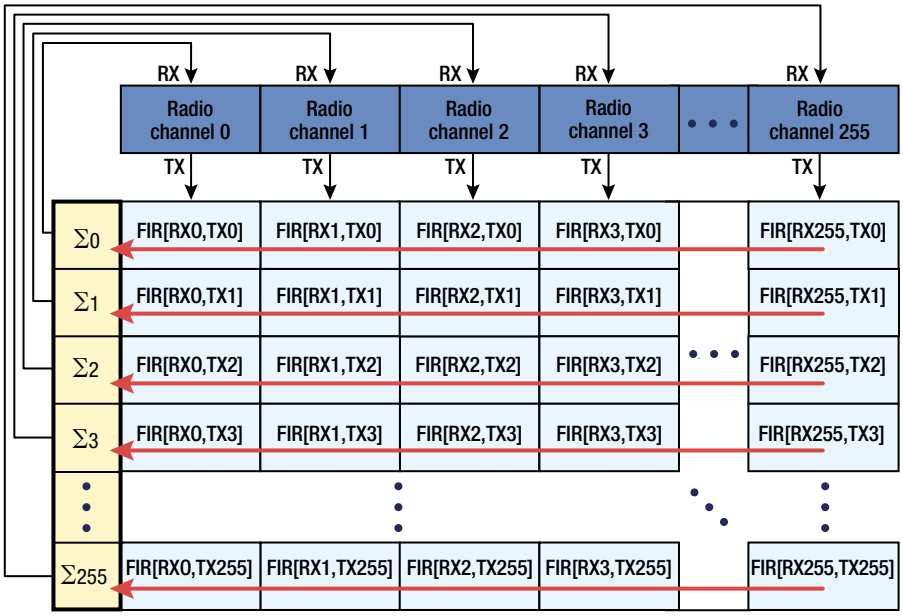

Figure 5. PDP matrix summing FIR output contributions for each radio channel.

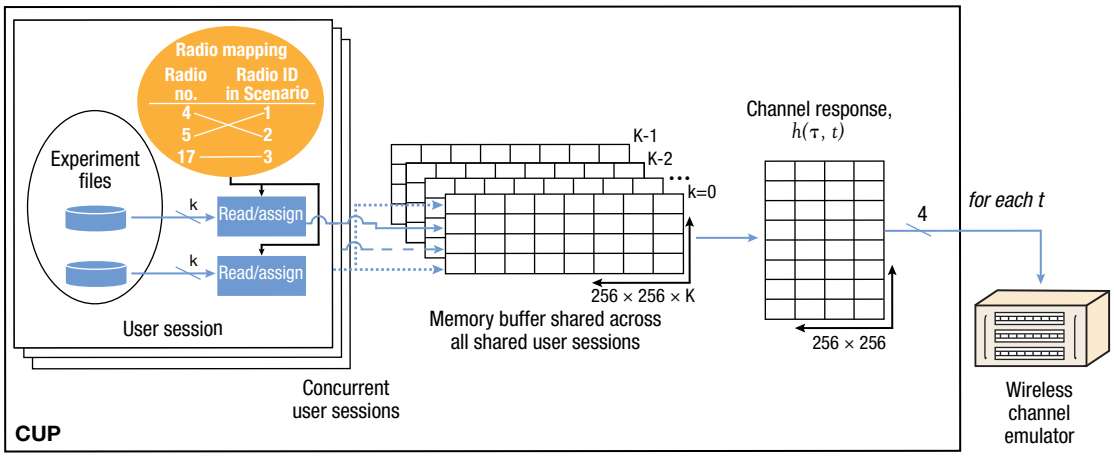

Figure 6. The four CUP phases—storage, reading/assignment, session abstraction, and sending.

had storage requirements of approximately 2.2 TB. For this reason, large storage arrays were used and were optimized for both reading and writing operations.

Reading and Assignment

Reading the WCP file from disk required specialized hardware and efficient parallel algorithms to meet the 1-ms latency requirement. Also required was an assignment of resources that mapped the general 1 to N radios in the WCP to the assigned resources on the Colosseum, as shown in Figure 6. This step required data processing between reading and sending each channel update to the wireless channel emulator. To reduce the impact of data processing on the strict 1-ms latency budget, multiple data processing threads were spun up at the start of an experiment. Each thread read a channel update ($N \times N \times 20.5$ bytes) from the WCP and mapped the generic transmitter–receiver pairs to assigned Colosseum resources. In this manner, a set of 1-ms updates were processed concurrently, which reduced bottlenecks in software. The lifetime of each thread was dictated by a session.

Multiple Session Abstraction

Unlike other Colosseum resources described in this issue (such as the Traffic Generation System and the Resource Manager), the wireless channel emulator was a physical shared resource and did not natively support multiple concurrent sessions. As such, abstraction of the multiple concurrent sessions was required, and a single time-ordered data stream defined the updates to the wireless channel emulator. APL developed a parallel algorithm that used a shared memory buffer between

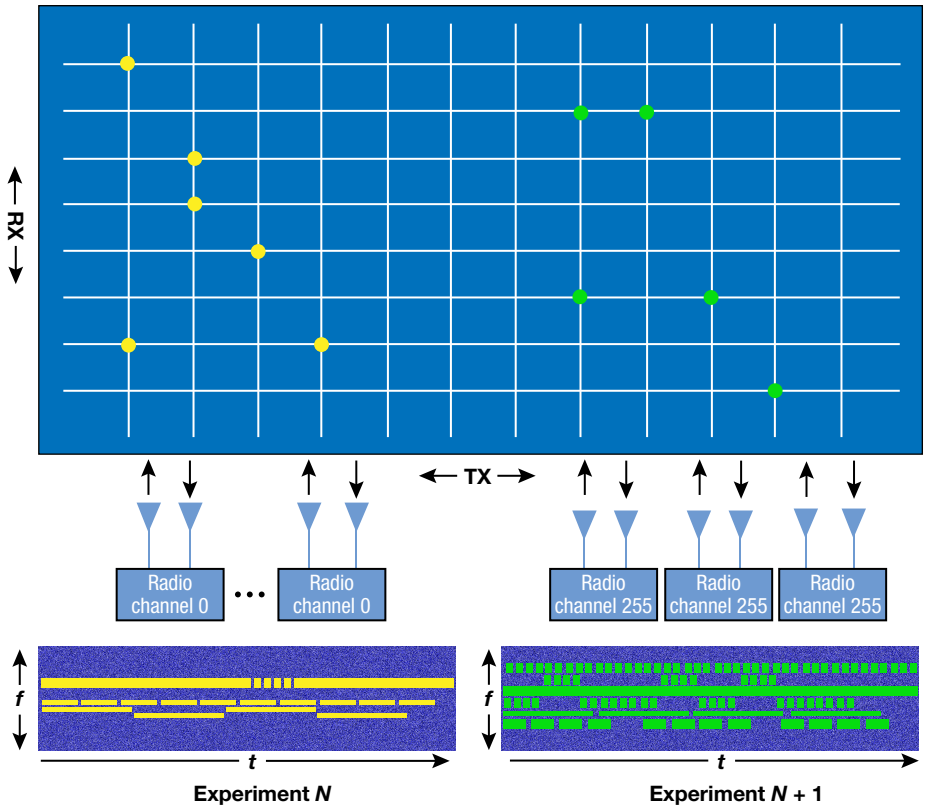

Figure 7. Coordination of tap updates. The figure shows that the two experiments started and stopped at different times and occupy similar frequency and time slots but could still run concurrently without interfering with each other.

the threads to hold the updates for an interval and still met the rigid 1-ms latency budget, as shown in Figure 6. The shared memory buffer was a set of 6000 tables (e.g., 6 seconds of channel updates) with 256 rows and 256 columns, where each entry held the four complex filter coefficients and four delay values (identical to the matrix illustrations in Figures 4 and 5). Given that each read thread worked on a unique time slice and each session was guaranteed to own a set of mutually exclusive SDRs, software locks were not required for the shared memory buffer. The algorithm focused on minimizing the amount of locks used, as wait times due to thread contention for resource access led to losses in performance.

Sending to the Wireless Channel Emulator

When the 1-ms timer expires, the appropriate table in the shared memory buffer locked and prepared for transmission to the wireless channel emulator using TCP/IP over four 10-Gb Ethernet connections. The table was split into four quadrants with respect to the transmitter numbers as shown Figure 6. When fully utilized, each data stream had a throughput of 4.19 Gb/s, and unused entries in the table were replaced with null values to guarantee that unused resources in the Colosseum had deterministic performance at all times.

The coordination of tap updates provided separate and isolated RF channels for each user request. As shown in Figure 7, two experiments could be run side by side without interfering with each other. The figure shows that the two experiments started and stopped at different times and occupied similar frequency and time slots. Without coordination between the RF Session Manager and wireless channel emulator, this type of channel allocation and emulation would not have been possible.

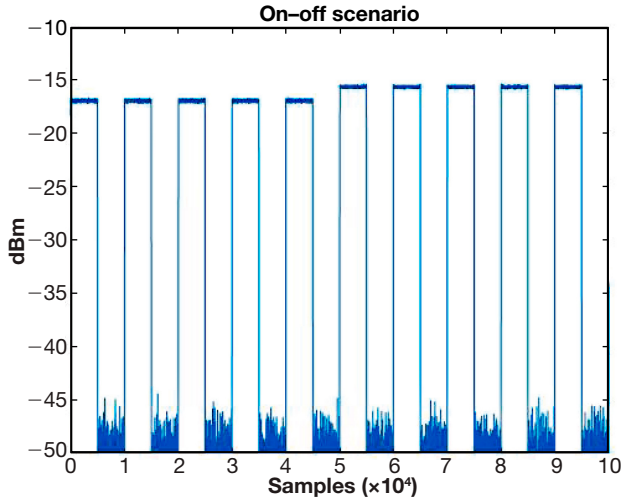

Figure 8. Time domain plot of update rate test results. This plot shows the received time domain signal from one SRN in the on–off update rate test. Measuring the on–off periods in the time-domain confirmed synchronization between the CUP and wireless channel emulator.

SYSTEM PERFORMANCE

The channel update rate within the wireless channel emulator was measured to ensure proper synchronization between the CUP and the wireless channel emulator's internal field-programmable gate array (FPGA) values. The wireless channel emulator was advertised to have a 1.0-kHz update rate.[1]

To evaluate this performance, APL created a custom RF scenario with channel coefficients alternating between $H(t) = 1.0$ and $H(t) = 0.0$. When $H(t) = 1.0$, the signal simply passed through the wireless channel emulator (see Equation 2). When $H(t) = 0.0$, the signal was blocked in the wireless channel emulator and only noise was present at the receiver. Signal processing blocks similar to those used in verification and validation testing were used to count the number of samples in each update. With an advertised 1-kHz update rate and a sample rate of 5 MHz, each channel update should have encompassed 5,000 samples.

Figure 8 shows the received time domain signal from one SRN in the on–off update rate test. Each segment lasted almost 5,000 samples. In an aggregate data collection of Colosseum performance with over 13,000 measurements, the update rate of the wireless channel emulator measured at 1.00002 kHz with a standard deviation of 0.08 Hz.

CONCLUSION

As part of the Colosseum test bed, the RF Emulation System provided accurate RF propagation between all radios for isolated yet concurrent experiments. The RF Emulation System contained three elements—the RF Emulation Manager, the CUP, and a wireless channel emulator. Wireless channels could be shared between a maximum of 128 two-channel radios simultaneously. The RF Emulation Manager handled experiment requests, configured the wireless channel emulator for each experiment, and monitored the status of the CUP during an experiment. In conjunction, the CUP and wireless channel emulator achieved a 1.0-kHz update rate for realistic RF emulation.

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]A. Chaudhari, D. Squires, and P. Tilghman, "Colosseum: A battleground for AI let loose on the RF spectrum," *Microwave J.*, vol. 61, no. 9, pp. 19–24, 2018.

[2]"REST API tutorial." https://restfulapi.net/ (accessed Jul. 12, 2019).

[3]"ZeroMQ.org, learn the basics." http://zeromq.org/intro:read-the-manual (accessed Jul. 12, 2019).

[4]"Dell 1.92TB SSD SAS read intensive 12Gbps 512e 2.5in hot-plug drive KPM5XRUG1T92." Dell. http://accessories.dell.com/sna/productdetail.aspx?c=us&l=en&s=bsdr&cs=04s2&sku=400-BBQP (accessed Jul. 12, 2019).

[5]R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau, "Redundant arrays of inexpensive disks (RAIDs)," in *Operating systems: Three Easy Pieces*. Madison, WI: Arpaci-Dusseau Books, 2015, ch. 38, pp. 1–18.

[6]T. S. Rappaport, *Wireless Communications: Principles and Practice*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 2002.

Daniel R. Barcklow, Asymmetric Operations Sector, Johns Hopkins University Applied Physics Laboratory, Laurel, MD

Daniel R. Barcklow is a wireless communications engineer in APL's Asymmetric Operations Sector. He holds a BS in computer engineering from George Mason University and expects to complete an MS in electrical engineering at Johns Hopkins University in 2019. Daniel has experience in field-programmable gate array (FPGA) and software development for wireless channel emulation, wireless receiver design, embedded signal processing, FPGA development, RFNoC block development on Ettus X310 software-defined radios, script development to protect Red Hat Enterprise Linux systems against software exploits and to limit system vulnerabilities, and formal testing on large Java-based systems. Before joining the Lab in 2016, he interned at NIST and Micron Technology. His email address is daniel.barcklow@jhuapl.edu.

Lian E. Bloch, Asymmetric Operations Sector, Johns Hopkins University Applied Physics Laboratory, Laurel, MD

Lian E. Bloch is a computer engineer in APL's Asymmetric Operations Sector. She completed a BS in computer engineering from Lehigh University in 2016 and is currently pursuing an MS in computer science with a telecommunications and networking focus from Johns Hopkins University. Bloch has experience with software development in Python and Java, as well as network management in Unix environments. Her email address is lian.bloch@jhuapl.edu.

Stephen W. Sweeney, Asymmetric Operations Sector, Johns Hopkins University Applied Physics Laboratory, Laurel, MD

Stephen W. Sweeney is an embedded communications and signals processing engineer at APL. He earned a BS in computer engineering from University of Maryland, Baltimore County and an MS in electrical engineering from Johns Hopkins University. He has contributed to multiple efforts to develop collaborative communications, including the DARPA SC2 Colosseum and Programmable Intelligent Collaborative Engagement Munitions (PICEM). He regularly contributes to efforts involving custom radio emulation design and implementation and is interested in signal detection and classification using traditional statistical methods as well as machine learning. His email address is stephen.sweeney@jhuapl.edu.

Brian E. Ahr, Asymmetric Operations Sector, Johns Hopkins University Applied Physics Laboratory, Laurel, MD

Brian E. Ahr is a software engineer in APL's Air and Missile Defense Sector. He earned a BS in computer science and mathematics from the University of Arizona and an MS in computer science from Johns Hopkins University. He has extensive experience with software design and development, modeling and simulation, cross-platform software development, C++, and Python. He has a passion for software engineering, building high-quality software, Linux, and open-source software. His work on SC2 focused on performance optimizations and correctness enhancements to the scenario data streaming component. His email address is brian.ahr@jhuapl.edu

William J. La Cholter, Asymmetric Operations Sector, Johns Hopkins University Applied Physics Laboratory, Laurel, MD

William J. La Cholter is a senior computer scientist in APL's Asymmetric Operations Sector. He has a BS in computer science and philosophy from the University of Maryland and an MS in computer science from Johns Hopkins University. Since 2011 he has been a member of the Senior Professional Staff at APL, where he has conducted research and development (R&D) in software diversity, cyber operations, malware attribution, and information security; performed security assessments of US government systems; and developed software for many domains and missions. For the DARPA SC2 program, Mr. La Cholter was a lead for a phase 1 RF Emulation System subteam, developer for the Traffic Generation System, and a software quality subject matter expert. He has led other APL teams as a section supervisor, project manager, and technical lead. Before joining APL, he conducted R&D in law enforcement systems, cross-domain solutions, security incident response, applied cryptography, adaptive networks, firewalls, and high-assurance operating systems. His email address is william.la.cholter@jhuapl.edu.

Samuel Berhanu, Asymmetric Operations Sector, Johns Hopkins University Applied Physics Laboratory, Laurel, MD

Samuel Berhanu is a software-defined radio engineer in APL's Asymmetric Operations Sector. He has a BS in engineering from Messiah College and an MEng in electrical engineering from Morgan State University. For the DARPA SC2 program, he contributed to Colosseum validation and verification efforts: debugging test apparatus built to interface with the RF Emulation System and migrating and modifying existing test artifacts for building and executing full Colosseum (128-node) validation and verification. Other experience includes implementing a Goertzel variant in a USRP X310 for holography

imaging using centimeter waves at near video frame rate; developing a 3G ALE demodulation and detector circuitry targeted for a Xilinx 7-series Virtex chip; and developing a test bench for verification of latency between crossbar and downstream AXI-stream-based blocks within the RFNoC framework in X310. Samuel is currently developing an N310 board support package for the OpenCPI framework. His email address is samuel.berhanu@jhuapl.edu.

Hakeem S. Bisyir, Asymmetric Operations Sector, Johns Hopkins University Applied Physics Laboratory, Laurel, MD

Hakeem S. Bisyir is a communication and networking engineer in APL's Asymmetric Operations Sector. He has a bachelors degree from Virginia Tech where he focused on signal processing and microcontroller programming. During his time at APL, Hakeem has worked on the system test architecture on the SC2 project, on establishing and distributing live wireless nodes throughout APL's campus, and on several networking and systems tasks. He has facilitated a machine learning workshop for his group and plans to pursue expertise in this field. His research areas of interest include RF signature tracking and encrypted network analysis. Hakeem led the characterization effort described in this article, designing both the scenario and container used to produce the timing metrics. His email address is hakeem.bisyir@jhuapl.edu.

Jarriel D. Cook, Asymmetric Operations Sector, Johns Hopkins University Applied Physics Laboratory, Laurel, MD

Jarriel D. Cook is the supervisor of the Wireless Cyber Capabilities Group in APL's Asymmetric Operations Sector. He has a BS in electrical engineering from University of Maryland, College Park and an MS in electrical and computer engineering from Johns Hopkins University. As supervisor, Jarriel aligns staff with appropriate tasking, coaches and mentors staff, reviews technical products, and works to foster an engaging and supportive work environment for staff. He collaborates with program management and sponsors to grow existing business relationships and develop new business opportunities. He has also served as a project manager, technical lead, and software developer on projects spanning Internet of Things protocols, software-defined radios, electronic warfare applications, and geospatial signal prediction algorithms. Before joining the Lab, Jarriel served in a variety of roles, including principal engineer, as a contractor for the US Naval Research Laboratory. His email address is jarriel.cook@jhuapl.edu.

David M. Coleman, Asymmetric Operations Sector, Johns Hopkins University Applied Physics Laboratory, Laurel, MD

David M. Coleman is research area lead for spectrum operations and supervisor of the Communication Systems Concepts Section in APL's Asymmetric Operations Sector. He earned a BS in computer engineering from Elizabethtown College, an MS in computer engineering from the University of Arkansas, and a PhD in electrical engineering from University of Maryland, College Park. For the DARPA SC2 effort, Dr. Coleman led the software development for the Colosseum's RF Emulation System as well as the Interface Control Document definition between this subsystem and the Colosseum. He also led code reviews, technical exchange meetings, and sponsor engagements. His previous experience includes work on Link-16 current and future capabilities to support autonomous platforms, wireless channel emulation capabilities in other hardware-in-the-loop test beds, and modeling and simulation of millimeter-wave and free-space optical backhaul networks. Dr. Coleman has presented at many conferences and has been published in journals and proceedings. His email address is david.coleman@jhuapl.edu.