RH RQ

Bringing great research ideas into open source communities

Abhimanyu Gosain

Where are we with wireless? How researchers are pushing forward the state of the art, and what that means for industry

+

Testing critical IoT systems

Measuring open source success

Yuga: A tool to help Rust developers write unsafe code more safely

Red Hat Research Quarterly
Volume 4:4 | February 2023 | ISSN 2691-5251

RESEARCH QUARTERLY

VOLUME 4:4

Table of Contents

Departments

04 From the director

05 News: Red Hat Collaboratory awards new funds to researchers

29 Column: Composing a research symphony

32 Research project updates

Features

08 Where are we with wireless? An interview with Abhimanyu Gosain

16 Testing critical IoT systems to mitigate network disruptions

20 Measuring open source success: developing analysis for actionable insights

24 Yuga: A tool to help Rust developers write unsafe code more safely

Red Hat

RESEARCH QUARTERLY

VOLUME 4:4

From the director

About the Author
Hugh Brock is the Research Director for Red Hat, coordinating Red Hat research and collaboration with universities, governments, and industry worldwide. A Red Hatter since 2002, Hugh brings intimate knowledge of the complex relationship between upstream projects and shippable products to the task of finding research to bring into the open source world.

Investments in university partnerships develop big ideas into working code

by Hugh Brock

It gives me great pleasure to announce the second round of the expanded Red Hat Collaboratory awards. This year we funded ten collaborative projects and nine speculative ones for $2.3 million, including a mix of new projects and continuations of the ones we supported in 2022. We have a nice review of them by RHRQ editor Shaun Strohmer on page 5.

Like everything we engage in at Red Hat Research, these projects are intended to push research ideas into open source, where they can develop into working code. We've now had this happen with at least three projects: Morphuzz (the QEMU fuzzer), Ceph storage caching, and the Unikernel project. Our projects on software and hardware co-design will soon follow. It's not easy finding engineers with time and energy to invest in these things, nor is it easy to convince researchers they should spend a little extra effort to make their results consumable by an open source community. But when we pull it off, the results make the effort very much worth it.

We also find interesting collaborative projects in all things related to edge and IoT, particularly in wireless. In this issue, we interview Abhimanyu (Manu) Gosain, who leads the Institute of the Wireless Internet of Things at Northeastern University. It's crucial to keep the data from an advanced factory or a research experiment local to the work, for both practical and regulatory reasons. Gosain talks about using AI inference on the user and control plane of fast wireless networks to make decisions in milliseconds—no time to send the data to the cloud for processing. What's more, the code making these decisions needs to be editable by ordinary software engineers, so there are now Python APIs to let cloud developers write apps that run in these situations.

You can see a practical example of this in the second IoT article in this issue. Research at Czech Technical University demonstrates automated testing of IoT devices used to monitor soldiers' health on the battlefield. Dubbed PatrIoT, the system researchers co-developed with Red Hat automates testing IoT devices in simulated unstable conditions, like those found during a natural disaster or on the battlefield.

Finally, as software does more and more, it is more and more important to help developers write better software. This is the motivation for Rust language, a compiled, high-performance language with the flexibility of low-level C, but with rules in place to prevent the common bugs even good C developers are prone to write. Rust does a good job providing guardrails to prevent unsafe code without being too constrained to be effective. However, there are times—writing API bindings for C code or device drivers, for example—when Rust programmers need to do unsafe things, and the guardrails are turned off. Our article on our joint project with Columbia professor Baishaki Ray describes Yuga, a code analyzer for Rust that specifically looks for errors in code flagged unsafe—the kind of code that would cause a real Rust program to crash, or worse. Tools like Yuga are essential for Rust to realize its potential as a complete upgrade from our current language tool bag, and we're proud to be funding its development. RH RQ

RESEARCH QUARTERLY

VOLUME 4:4

Red Hat Collaboratory awards new funds to research with industry impact

Projects in security, hybrid cloud networks, education, autonomous vehicles, and more receive resources for open source research.

by Shaun Strohmer

On January 26, 2023, the Red Hat Collaboratory at Boston University announced the recipients of its 2023 Research Incubation Awards.

The funding program, now in its second year, provides resources to research projects that focus on problems of distributed systems, security, operating systems, and networking whose solution shows promise for advancing the field and driving change in industry.

Expanding research at the Collaboratory will also create opportunities for Red Hat engineers and BU researchers to work together on new cloud development, testing, and experimentation, applying their results to new science and technology domains.

Awards in the second year of funding include renewals for several projects that received 2022 Research Incubation Awards and new submissions. Red Hat US Research Director Heidi Picher Dempsey said, "The Collaboratory projects in 2022 produced many interesting proofs of concept and demonstrations, ranging in scope from the chip level to multicloud federations. Experiments and prototyping in areas such as security, hybrid cloud networks, operations support, and tuning will continue in the second year of the Collaboratory. I am especially excited to see the collaborations reaching into other sciences and education, where applying computing power in new ways can benefit society directly."

The purpose of the Collaboratory is to enable research that would not be possible without collaboration between industry and academia. Resources from the MOC Alliance, the world's first open cloud for researchers, play a significant role in fostering this research.

About the Author
Shaun Strohmer is the editor of *Red Hat Research Quarterly*. She has worked as a writer and editor in academic publishing for over twenty years, and since 2014 she has focused on software development, cybersecurity, and computer science.

If you are interested in exploring collaboration opportunities on these or other research projects, please contact Heidi Picher Dempsey, US Research Director for Red Hat.

As BU professor and Collaboratory co-director Orran Krieger explained, "Collaboration between researchers and engineers creates projects with an impact. The capabilities of the MOC Alliance drive a range of projects that target real-world industry challenges made visible to researchers through the Alliance, such as AI Ops, cloud telemetry, disaggregated storage, OS support for low latency/high-bandwidth networking, and cloud security. Researchers can exploit these resources to explore how systems research can help with important social problems in areas like ecology, education, automotive, or smart cities and how to address emerging demands and technology changes that will increasingly drive future clouds, such as edge, streaming, or reconfigurable hardware."

The Collaboratory also continues to fund the large-scale AI for CloudOps project, which received a multiyear award in 2022. According to Dempsey, "The AI Operations project, which laid the foundation for radically changing how day-to-day operations are conducted in data centers, will continue to build in its second year and produce results that should be directly applicable to the MOC Alliance members."

Ten collaborative projects (small category) were selected to receive one year of funding with the option to renew. These projects combine the academic expertise of university faculty with the industry knowledge of Red Hat engineers. Of these projects, four are renewals of projects awarded Research Incubation Awards in 2022. The new projects in 2023 are:

Improving cybersecurity operations using knowledge graphs, David Starobinsky (BU), David Sastre Medina (RH), Zhenpen Shi (BU), Şevval Şimşek (BU)
Goal: Improve the workflow and performance of security operations centers, including automating several of its tasks, by leveraging the vast amount of structured and unstructured real-world data available on threats, attacks, and mitigation

Relational memory controller, Manos Athanassoulis (BU), Renato Mancuso (BU), Ulrich Drepper (RH), Ahmed Sanaullah (RH)
Goal: Enable the integration of the Relational Memory Engine (RME), an FPGA-based, hardware-based data transformation engine, with a memory controller

Toward on-the-fly reorganization of high-order data objects, Renato Mancuso (BU), Manos Athanassoulis (BU), Ulrich Drepper (RH), Ahmed Sanaullah (RH)
Goal: Investigate the design and

development of on-the-fly data reorganization engines to make the benefits of RME available to a broader set of applications, such as image manipulation, machine learning, and tensor analysis

HySe: Hypervisor security through component-wise fuzzing, Manuel Egele (BU), Muzammil Hussain (BU), and Bandan Das (BU)
Goal: Design, implement, and evaluate program analysis capabilities that allow the preemptive identification of bugs and vulnerabilities in hypervisor components that use interfaces identified as exposed to potential attackers

Prototyping a distributed, asynchronous workflow for iterative near-term ecological forecasting, Michael Dietze (BU), Christopher Tate (RH), Yannis Paschalidis (BU), Atefeh Hosseini (BU)
Goal: Prototype an accessible community infrastructure to generate ecological forecasts at scale, focusing on the development of a cloud-native workflow that can handle an asynchronous, event-driven, distributed approach to execution

Co-Ops: Collaborative OpenShift-based training of large open-source AI models at scale, Eshed Ohn-Bar (BU), Adam Smith (BU), Erik Erlandson (RH), Michael Clifford (RH), Lance Galletti (RH), Sanjay Arora (RH), Ruizhao Zhu (BU), Jimuyang Zhang (BU), Yuanming (John) Chai (BU)
Goal: Develop a set of open source, Red Hat-integrated tools for efficiently and flexibly facilitating diverse and modular collaboration when training AI models for autonomous driving at scale, emphasizing privacy-preserving knowledge sharing

Nine projects (speculative) related to exploratory research or MOC Alliance projects designed to initiate a collaboration also received Research Incubation Awards. Six of these awards will support projects with encouraging outcomes initially supported by the Collaboratory in 2022. The new speculative projects in 2023 are:

Minimal mobile systems via cloud-based adaptive task processing, Eshed Ohn-Bar (BU), Hee Jae Kim (BU), Lei Lai (BU), Sanjay Arora (RH), Bassel Mabsout (BU)
Goal: Build an efficient cloud-robot distributed computing platform for automatically offloading computationally intensive tasks to the cloud, improving performance and making low-cost, cloud-enabled robots accessible for a significantly larger set of users

Privacy-preserving, automated operational data sharing telemetry framework, Alan Liu (BU)
Goal: Develop an open source automated tracing system to collect, process, and anonymize operational data, focusing on identifying and testing privacy preservation models

Open source infrastructure for secure educational data management to optimize treatment and identification of students with learning disabilities, Hank Fien (BU), Ola Ozernov-Palchick (BU), Kasey Tenggren (BU)
Goal: Integrating student data

in a privacy-preserving way for analysis of students' risk profile and the likelihood of an individual student responding positively to a particular intervention

All software developed by these projects will be available under an open source license, and all results will be publicly available. Learn more by visiting Red Hat's research project pages, and stay tuned to *Red Hat Research Quarterly* and the Red Hat Research website (research.redhat.com) for more opportunities to learn about these projects and get involved.

Where are we **with wireless?**

How researchers are pushing forward the state of the art, and what that means for industry

An interview with ***Abhimanyu (Manu) Gosain***

conducted by ***Heidi Picher Dempsey***

RESEARCH QUARTERLY

VOLUME 4:4

Red Hat

Interview

Abhimanyu (Manu) Gosain is a Senior Director for the Institute of the Wireless Internet of Things at Northeastern University, co-chair for the FCC 6G Technology Advisory Council, and Senior Advisor for the National Telecommunications and Information Administration (US Department of Commerce) and the Office of the Undersecretary of Defense (US Department of Defense). Manu spoke with Heidi Dempsey, US Research Director for Red Hat, about how university research fits in the technology innovation cycle and the paradigm shifts that will shape the next generation of wireless protocols.

About the Author
Heidi Picher Dempsey
is the US Research Director for Red Hat. She seeks out and grows projects with academic and industry partners in areas such as operating systems, hybrid clouds, performance optimization, networking, AI, and security.

Heidi Dempsey: So you're a wireless computing expert. You've done papers and given talks. You're on government advisory committees and leading extensive research programs that are international and national in scope. What got you interested in computing and wireless to start?

Abhimanyu Gosain: Dialup was my first foray into networking and the internet. Your phone made that weird sound and opened a window to the world. The journey from me sitting 12,000 miles away in New Delhi, India, to where I am today wouldn't have been possible if we weren't connected by global networks. This obviously predates Zoom and online platforms, but that was still a huge part of being able to come to the US.

I got a small graduate assistantship helping a principal investigator who had embarked on the almost impossible mission of building a community wireless network that could be used for academic research. There were lots of learnings and growth and setting lots of single-board PCs on fire—

Heidi Dempsey: Literally?

Abhimanyu Gosain: Yes, literally! I was weatherizing and hardening systems, then working on all the software around making things communicate. I could implement what we'd learned in the books—like Shannon's information theory and the Fourier transform—then actually see those things use the wireless channel. That got me hooked.

Heidi Dempsey: Just to clarify for our younger readers: When you talk about dialup, you're talking about a time before you were even in university. So do you think you were always a nerd, or did you become a nerd because of dialup?

Abhimanyu Gosain: Ha! I think "became" would be the operative term. I was into soccer and breaking every windowpane in my neighborhood. My parents were a little more relaxed once I got hooked on computers because I could cause less damage. I did run up a long bill doing some online gaming, and they approached me with the four-figure bill. That's how charging worked back in the day—unlimited data wasn't a thing. As economies of scale have grown, we're down to negative average revenue per user for our networks, which is a challenge. Back in the day, the cost was borne by the customers. Now, infrastructure providers bear that cost.

Heidi Dempsey: That's especially true for wireless, where the cost is the device and the subscription, but there's a huge amount of infrastructure behind that, making it all possible.

I ask about your past because one of the really interesting things about your job is that you have to talk to government people. You have to talk about policy, but you also need to have

Instead of selling a solution or telling people how cool the technology is, I try to listen.

excellent nerd cred and be able to talk about the details of hardware and software. How does that work?

Abhimanyu Gosain: Over the past five years, during my transition from being a pure technologist to supporting federal agencies, regulatory agencies, and different consortiums and boards, I've learned that the most important thing is identifying the problem you are trying to solve. Instead of selling a solution or telling people how cool the technology is, I try to listen. Whether I'm talking to a Department of Defense person from a warfighter perspective or an operations perspective, they're going to share their challenges.

I'll take a very simple example: spectrum management. The year is 2022, and believe it or not, the most advanced army services in the world still manage the spectrum on Excel spreadsheets and sticky notes. On average, a spectrum management officer is staring at seven different screens because seven different isolated systems are operating at any given time. To absorb that amount of information and then render the intelligence or the data upstream to your commanders or stakeholders as a human-based decision—it's a huge bottleneck. Now I understand the problem, and it's not just a hypothesis or an algorithm I'm trying to develop.

That's what we bring to our students at Northeastern University. Co-op programs allow for hands-on experience at the graduate and undergraduate levels to deal with real problems. And the philosophy we have at the Institute for the Wireless Internet of Things is to solve real-world problems and do it at a system level. We're basing everything on very strong fundamental science, then taking the next step to build prototypes or reference platforms or architectures. That's where we lean on the open source community a lot.

Heidi Dempsey: That's really cool! That's what we're doing in Red Hat Research collaborations with Northeastern and other universities: let students take what they learn in the classroom and read in books and work with people hands-on in a real project. But I hadn't thought about expanding that past the engineering world to include thinking about spectrums and thinking about policy. The same approach can help even though the domain is different.

RESEARCH TESTBEDS

Heidi Dempsey: You're the czar of many testbeds that have lots of amazing capacities. One of the places we're using Red Hat OpenShift is in those testbeds. How does that tie into the challenge of finding a problem and solving it?

Abhimanyu Gosain: Look at what's happened with cloud computing over the last decade in terms of software and programmability. This new paradigm was developed and accelerated in the context of a living lab—essentially, a testbed. We've capitalized on those advancements in wireless research. Yes, we sacrifice performance, but our goal is not necessarily to have something operational or production-ready. That will be done by industry.

The testbed allows us to develop our technology at the pre-standardization curve. That's very different from where cloud orchestration networking companies are. Standardization and intellectual property are the biggest boon and curse in developing new wireless technology. We work through standard-development organizations like 3GPP [Third-Generation Partnership Project], which develops standards for wireless communications. If you enter a technology just a little bit late, you're going to run up against the problem of intellectual property: industry ends up in different silos and doesn't want to share ideas.

When we work at the pre-standardization, pre-commercial side of the technology, we can find and leverage large numbers of partners who are competitors with each other, but they all come to us with a common root-cause problem. The testbed can be the neutral environment that allows you to get fundamentals right. We can have a basis of agreement, for example, on where some additional performance enhancements happen if you use a MIMO [mutiple in multiple out antenna] system, which allows you to use the spectrum more efficiently. Then companies can add proprietary customizations on top: your custom schedules, algorithms, hardware optimization, acceleration. Companies will do what they each do best, but before that, research has made a measurable difference in the state of technology.

By the way, while we're doing all this, students and research staff are getting those learnings at the same time as industry. For the 5G and 6G generation going forward, that's going to be a massive paradigm shift. In the 2G, 3G, 4G days, university research lagged industry by at least 24 to 36 months. Industry didn't come to us, because we weren't at the cutting edge.

Heidi Dempsey: And in the early days you were trying to do it with hardware, setting up wireless testbeds that could be six miles wide or with drones flying around. That's a lot harder. You're not doing that now, right?

Abhimanyu Gosain: We're not doing that because the cloud's moving closer and closer to the edge. As that journey continues, software will be the dominant factor. Then we're going to see different forms of programmable compute—general purpose, GPUs, and FPGAs—and that's what allows us, our students, and our cohort of research partners to build stuff and even accelerate certain research innovations before industry gets there.

> Standardization and intellectual property are the biggest boon and curse in developing new wireless technology.

Heidi Dempsey: So software is eating the wireless world, too, not just the cloud world?

Abhimanyu Gosain: For better and worse. It's a new shift: with my training from 10 or 15 years ago, I wouldn't survive in that world today. That's why workforce development and training for students is a big issue. From a DevOps perspective or a system integration perspective in the telco world, at AT&T or Verizon or any CSP, they need a cloud engineer more than a hardware RF [radio frequency] engineer, and that's huge. For example, platforms like OpenShift are what you need if you're going to operate cloud container platforms and cloud paradigms and you're going to run network functions that serve a purpose in the wireless domain. Still, the operation, the execution, the security, the resiliency, and so on, all have to be done by a cloud professional.

Heidi Dempsey: So we have Kubernetes in the cloud, which is an example of what you were saying about different models eventually converging on one standard. Is something like that coming for how the edge and cloud will connect?

Abhimanyu Gosain: Right now, the edge is the biggest area of research, innovation, and focus—both from industry and academia. It boils down to two questions. From an application or enterprise use case perspective, how can you provide access to or make sense of data coming from user devices and do it in a very small time cycle? And from a security/privacy perspective, how do you keep that data on-prem?

Heidi Dempsey: Wait—on-prem and wireless? You just blew my mind. What does that mean?

Abhimanyu Gosain: Take an enterprise case. Let's say it's industry 4.0: a factory where you have UAVs [uncrewed

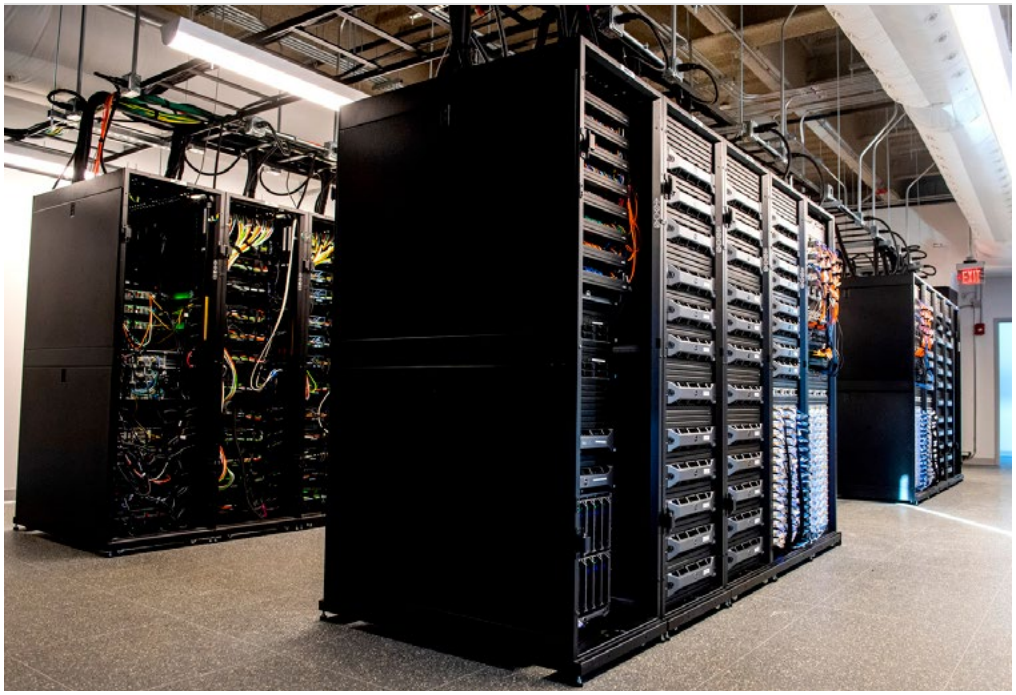
aerial vehicles] or robots working, or you have video streams where you're tracking what's going on. You don't want to move that data, but you want the ability to make decisions based on ingesting those data streams. And you have to do that at a fidelity and timescale that doesn't allow you to send that data into the cloud and back.

So you need some smartness, data preprocessing, or algorithm development implemented at the edge. At the enterprise level, you need visibility into that edge cluster. That's the data optimization perspective—the data security, privacy, decision-making, closed-loop piece. From a DevOps perspective, that's when you lean on the cloud a bit more.

In wireless, as with any network communication protocol, there's a control plane, a user plane, and a management plane. The user and the control planes are where those data operations have to be done as locally as possible. That also has ramifications on costs and on how much energy you're using. A lot of companies now understand that if we have a smart enough strategy on edge, we can move the needle again in terms of cost and performance, as well as experience.

Heidi Dempsey: I see now why you use the term edge and don't think so much about "wireless versus cloud." We're getting a lot of different clouds in a lot of different places that are rubbing up against each other, and we're trying to abstract as much as we can what's happening at the lower layers.

I was going to ask you why a cloud engineer should care about 5G versus

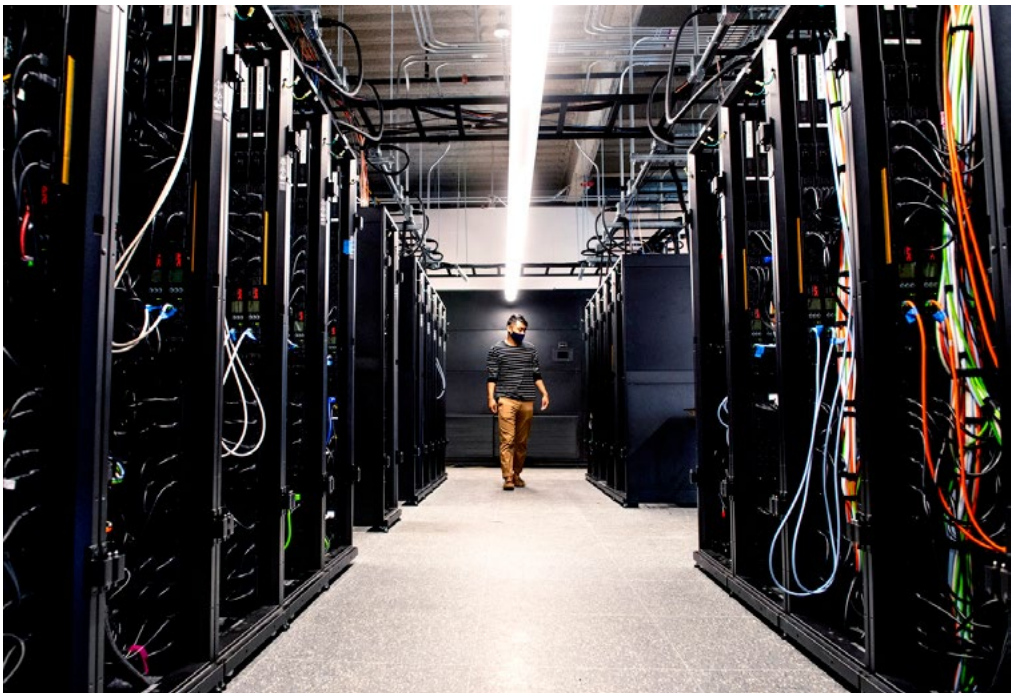


6G versus O-RAN [Open Radio Access Networks], but you already answered that. How much do you think they need to know? They can't be oblivious to wireless protocols and how they affect what you can do, but—

Abhimanyu Gosain: They can, actually! You mentioned a magic term getting a lot of attention in industry and government circles: O-RAN. We're now talking about abstracting away the key functionalities, like the wireless protocols, the wireless channel state information, and so on. Instead, we present it to you as an SDK with APIs, which is what you as a cloud engineer understand. For you, it's a socket at most.

Then you have the RIC, or RAN Intelligent Controller, which is not too different from an SDN controller. It abstracts the functionality of your RAN equipment, or radio, or what in 5G speak is called gNodeB, with all the murkiness of wireless, from channel state information to IQ samples to physical resource blocks, which is something you'll need—but you'd need to take a communications course to understand all those terms that I just used.

The beauty is that now the standard interfaces industry is developing abstract that information and present it to you as APIs that you can query and develop your own xApps or rApps, to use ORAN speak. Think of it as a closed-loop control system: AI and ML at each layer allow you to ingest data streams, absorb them, and then actuate or make a decision—either an optimization decision or an objective you want to meet. Then the tool will put that action back into the system, let the system behave, and report



that back to you. If it needs to happen on the scale of milliseconds, that's an xApp. If you can tolerate latency up to a second, that's an rApp.

So now a cloud engineer or anybody with a basic understanding of computer engineering can begin to wade into the waters of wireless communications. We have the infrastructure, the radios, and the cool technology, but for you to utilize that, you just come to us with the programmatic mindset, maybe learn Python, and then you have the ability to query APIs and modify network parameters at your fingertips. And the reactions are near real time, so you can see those changes implemented.

All of these network functions on a 5G architecture can be expressed or deployed as network functions on cloud orchestration platforms. At that point, the problem you face is the problem that you live with every single day, which is just workload management.

EDUCATION AND ACCESS

Heidi Dempsey: You've been in wireless for a long time, so I know you spend time up on roofs of buildings. You must have a good antenna story.

Abhimanyu Gosain: I'll give you one funny story and one cautionary tale. The moral of the cautionary tale is to always monitor the weather before you go up on rooftops, especially in Cambridge, and be aware of the lake effect. Unobstructed 20-, 30-, 40-mph gusts feel very different when you're standing on a 3m x 3m platform, holding on for dear life.

The funny story is how we ran afoul of the US Federal Communications Commission (FCC). The FCC dictates which wireless spectrum can be used, how it can be used, what power levels you can use it at, and so on. It's very serious business: billions of dollars are at stake, and you don't want to mess with public safety services or a CSP's ability to provide them.

A misconfigured parameter in one of our RAN files set one of our radios to radiate at a frequency used by a communication service provider, who then filed an exception request with the FCC complaining of a rogue transmitter. This led to the FCC sending out regulatory engineers with spectrum analyzers, pointing at different rooftops, and identifying us as the culprit. That was an awkward interaction.

This led to training courses for everybody and putting safeguards and checks in place. So now if you are applying for an FCC experimental program license and you see certain additional requirements, you have us to thank for that.

> We hear from many people in rural America, where they're still struggling to get 4G service.

Heidi Dempsey: It's good to know that you have such a significant impact on government!

Abhimanyu Gosain: The FCC is very supportive of research,

however. They want to allow wireless innovation to flourish.

Heidi Dempsey: Given that research and innovation are awesome, which we both believe, what advice would you give to people in university now or even before university who are computing on their own or playing around in AWS?

Abhimanyu Gosain: It's easy to say that you should get hands-on experience. There's nothing like trying to configure a server or a network interface. But I'm also cognizant that my worldview is shaped by sitting here at an R1 [top-tier US research] institution in the Northeast Corridor that is very well-resourced.

I co-chair the FCC Technology Advisory Council on 6G, and we hear from many people in rural America, where they're still struggling to get 4G service. They laugh when we tell them we're now looking at 6G.

Heidi Dempsey: That's really important. We see the same thing. If you're trying to reach out beyond the R1 level, it's a bigger hurdle to get people over. Doing workforce training and apprenticeships is a lot harder if you're thinking about how to do them in a research context

Abhimanyu Gosain: Growing that skilled workforce needs to start by teaching these concepts very early. Now we're slowly seeing Chromebooks and laptops given to third- and fourth-grade students, and that resource stays with them until they get out of high school. If you have the basic resources, the choices are endless. There are free

Manu Gosain presenting at Red Hat Research Day Europe in Brno, September 2022

online tutorials from places like MIT's OpenCourseWare. Covid has been a great equalizer because it has allowed people to access information that historically has been shared only at premier conferences where you have to pay thousands of dollars to attend. But now information is much more freely available. So there are no excuses at this point, if you have some semblance of resources or support.

Heidi Dempsey: Do you think the other direction is true too? For those of us who are already in the field, there are no excuses for not reaching out to people who don't have all that advantage, right?

Abhimanyu Gosain: Absolutely. I'll tie this back to one of the first questions you asked about working with government and working in technology. We're trying to solve a problem. I think sometimes in our field we're so focused on small bits and optimizing this, that, and the other that we forget the actual problem we're trying to solve.

I think we are on the precipice of actually democratizing access. Thanks to federal support and cabinet-level positions for science and technology, I think we're going to slowly start to fold in those who've been underserved and underrepresented for a while.

Heidi Dempsey: That's really inspiring! And it's a nice place to leave things. Thank you for taking the time to share your insights.

THE UNIVERSAL AI SYSTEM FOR HIGHER EDUCATION AND RESEARCH

NVIDIA DGX A100

Higher education and research institutions are the pioneers of innovation, entrusted to train future academics, faculty, and researchers on emerging technologies like AI, data analytics, scientific simulation, and visualization. These technologies require powerful compute infrastructure, enabling the fastest time to scientific exploration and insights. NVIDIA® DGX™ A100 unifies all workloads with top performance, simplifies infrastructure deployment, delivers cost savings, and equips the next generation with a powerful, state-of-the art GPU infrastructure.

Learn More About **DGX** @ nvda.ws/dgx-pod
Learn More About **DGX on OpenShift** @ nvda.ws/dgx-openshift

Red Hat

RESEARCH QUARTERLY

VOLUME 4:4

Feature

About the Author

Miroslav Bureš leads the System Testing IntelLigent Lab (STILL) at the Faculty of Electrical Engineering, Czech Technical University in Prague. His research focuses on system testing, IoT technology, and artificial intelligence, and their application in rescue missions, medicine, and defense.

Testing critical IoT systems to mitigate network disruptions

The Internet of Things brings new opportunities and new challenges for mission-critical applications where lives are at stake. Systematic testing can help.

by Miroslav Bureš

The Internet of Things (IoT) has significantly increased the capabilities of mission-critical systems in many domains. Integrated rescue systems, healthcare, defense, energy, and transportation benefit from using the IoT, enabling faster system reactions and better functionality for users. Instant situational overview, speedier information sharing, automated decisions, and shorter response times are just some of the possible enhancements.

A fundamental vulnerability of IoT systems is their reliance on data networks. IoT typically refers to devices using a public Internet; however, for some critical or defense systems, closed and more secured networks are used instead. Interruptions to network connectivity can happen in either environment for various reasons that can't be fully eliminated. Instead, IoT systems must be optimized to work even when network connections are weak or disrupted. Viable testing models for mission-critical IoT systems are essential for making them usable in real-world settings.

This article will introduce a technique for limited network connectivity testing and test case generation developed as part of the Quality Assurance for Internet of Things Technology project. This joint project of industry engineers and the Faculty of Electrical Engineering, Czech Technical University in Prague (FEE CTU) is funded by the Technology Agency of the Czech Republic, and it led to the development of the open source test-automation framework PatrIoT.

IOT IN COMBAT

These new opportunities afforded by IoT bring with them an increase in the complexity of mission-critical systems. The system attack surface also increases along with this complexity, as does the possibility of defects hidden in the system. Especially for critical systems relying on a data network, these may have serious consequences. Network signals can be weak due to energy limitations or terrain, or, in the case of defense systems, the signal can be jammed by the enemy, or parts of the system may be destroyed. Even in these scenarios, the system must be reliable enough to keep working, and this reliability must be properly tested. It's a critical system – lives might depend on its correct functionality!

For example, the Digital Triage Assistant (DTA), designed for military combat environments, is a potentially life-saving technology that must function in settings where disruptions are likely

to occur. The DTA is a joint project of the System Testing IntelLigent lab at FEE CTU in Prague, the NATO Allied Command Transformation Innovation Hub, the Czech Army, the University of Defense (Brno, CZ), Johns Hopkins University, and other partners. This project creates a sensor network collecting data about soldiers' vital signs to maximize their survival chances if they are wounded. The data are aggregated to a mobile back-end server, and soldier and unit status is estimated. Then the data are provided to different roles. A Combat Lifesaver wearing augmented reality glasses will see the positions and status of wounded mates through the forest or smoke. Unit commanders can see the positions of their soldiers in a map application. A surgeon waiting for a medevac to come to a dressing station can be better prepared by learning some indicative information about a soldier's health before they arrive to be examined.

Everything in this system can be mobile or disrupted. Soldiers move and can take cover, limiting the sensor signal. The back-end unit can be mounted to a vehicle and moved, and some units may require stealth mode. These are extreme examples, but in other critical IoT systems, network disruption situations happen on a daily basis. Weakly covered rural areas, no network coverage in tunnels in urban areas, cyberattacks, or even switching off a public mobile network for security reasons are all frequent causes of signal interruption.

TEST CASE CREATION

How to test an IoT system to be sure it works well in these situations? As test engineers, we are interested

Testing of DTA in summer 2021, Czech Republic. Soldiers in the unit are securing the area, and a Combat Lifesaver (CLS) starts examining a wounded soldier (this is simulated during the exercise).

The initial version of the body sensors in the DTA project. A FlexiGuard solution by the team from the Faculty of Biomedical Engineering, CTU in Prague, is currently used. These sensors will be replaced by a smart-textile-based variant or by sensors integrated directly into the ballistic protection.

A Czech Army rescue vehicle is coming to pick up the wounded soldiers during DTA testing.

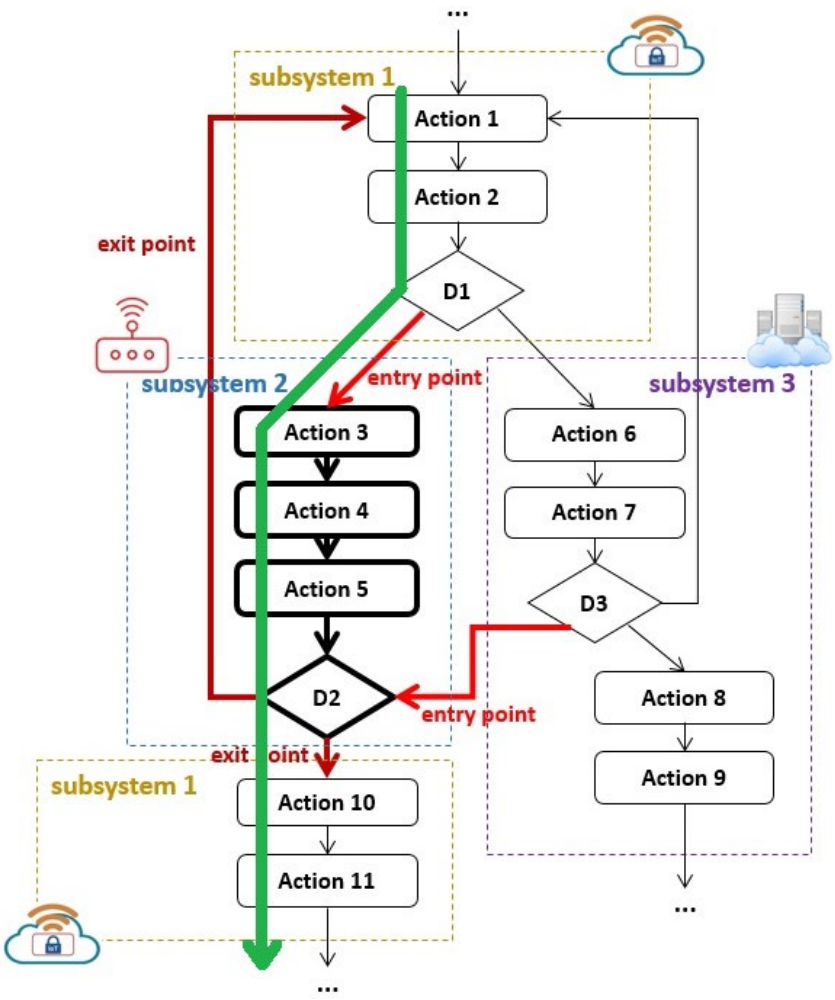


***Figure 1**. Example of a system model*

in two principal situations. What happens when the network connectivity is interrupted? And then, what happens when the connectivity is restored? We need to test these situations thoroughly.

The approach we use is model-based testing. We model an aspect of the tested system, in this case, a process to be tested by a model that is very similar to the UML activity diagram.

Figure 1 is a model of a tested system. In this example, subsystems 1 and 3 are connected to a stable network. Subsystem 2 is mobile, and its network connectivity can be interrupted. The entry point depicts the part of the process in which connectivity can be disrupted, and the exit point is where connectivity is restored. In tests, we try multiple possible sequences of entry and exit points in the schema. A green arrow depicts one possible test case.

The challenge we face is that we never know in which part of the system process such a disruption might occur. Using an automated test case generation tool for all scenarios is cost-prohibitive. We developed a new approach, creating unique algorithms to generate optimal—that is, the most relevant—test cases using the open source Oxygen platform. In the future, we will add more algorithms and generalize the technique to component failover testing, which will apply to more situations in critical systems testing.

To generate the test cases, we created a set of various algorithms, differing in their principles. In this scenario, a test case is a path through the system process. This portfolio comprises algorithms using classical graph traversal as well as AI-based representatives such as artificial ant colony optimization or genetic algorithms. For example, in the ant colony algorithm, we simulate artificial ants mimicking real ant behavior in nature. When ants find a discarded sandwich in the forest and they like it, they make a trail from their nest to the food source. To keep their mates on this trail, they deposit a pheromone path. The algorithm simulates this behavior. Artificial ants go through the tested system model and calculate the close-to-optimum set of tests together.

In the model, we estimate the probability that individual system components can be disconnected from the network. Using a threshold, we then model some realistic situations where this happened. To guarantee the strength of generated test cases, we use four levels of so-called test coverage criteria, a set of

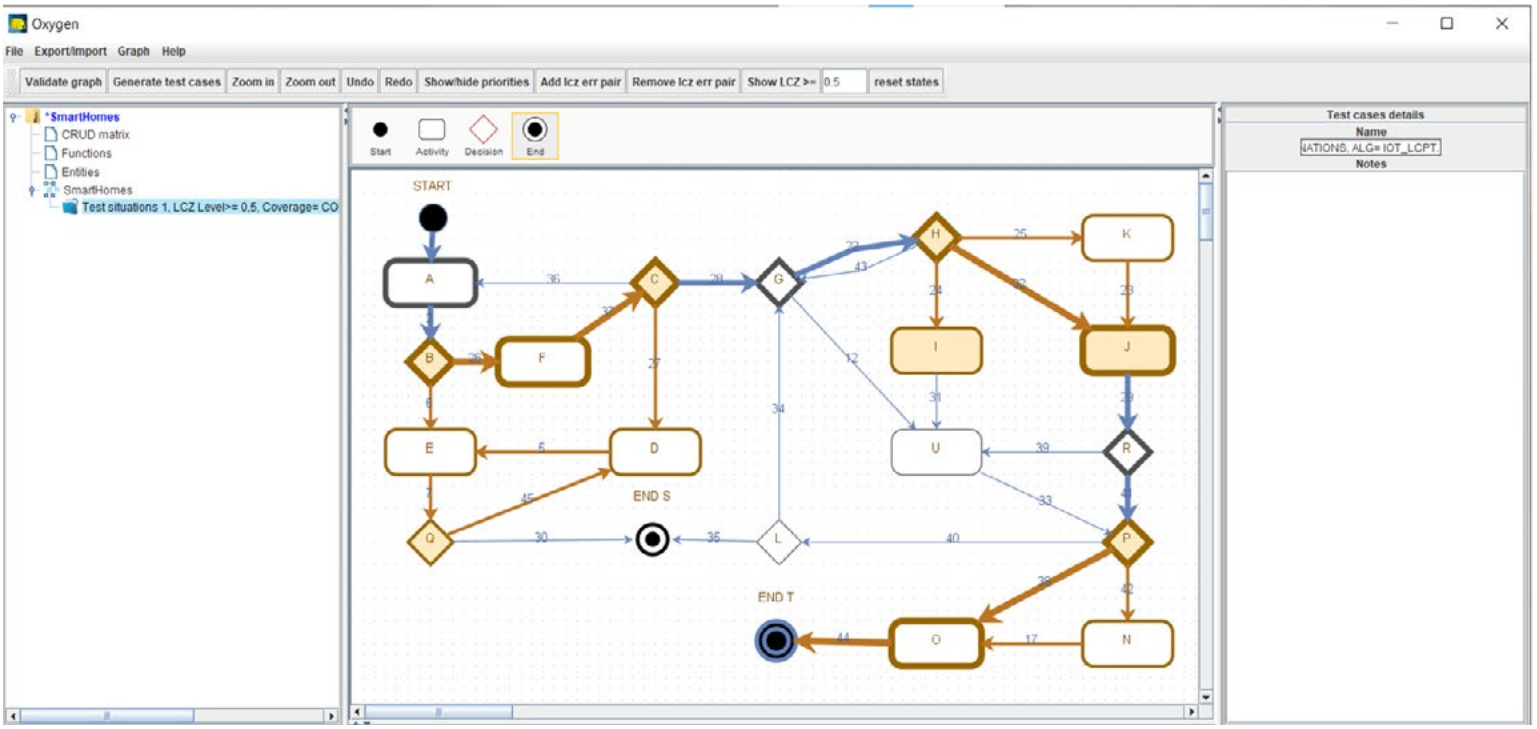


Figure 2. *Limited network connectivity testing technique in our model-based testing tool Oxygen. Source: System Testing IntelLigent Lab (STILL)*

rules that the test cases must satisfy. **Figure 2** is an example of tested system processes in Oxygen. Parts of the process likely to be disconnected from a network during the system run are visualized by brown parts of the model. When the test cases are computed, they can also be shown in the model. In this example, it's a bold path through the schema.

For example, in the DTA project, we tested a situation in which a soldier's ballistic protection, combined with hilly terrain, weakened the sensor's radio signal. Data flow became intermittent and was combined with a suboptimality in the server-side code. This led to an unnecessarily long timeout to restore the soldier's position in the map application once the signal was stronger. Because such disconnection can happen in many parts and variants of the process, it is much more effective to model the process and test these possibilities systematically than trying to simulate these situations randomly.

IMPLEMENTATION

However, these are details that test engineers do not need to consider to make use of the testing. From the engineers' viewpoint, they can model a system process in a graphical user interface, add information about the probability of network connectivity outage, simulate a particular situation, and let the machine do all the computations. When finished, they can visualize the produced test cases in the model, which is helpful for getting a good overview of the tests. The test cases can be exported in open formats based on XML, CSV, and JSON to be easily loaded into a test management or test automation tool.

With this systematic approach to testing and generating test cases, we can greatly increase the confidence that a limited network will not cause unexpected problems in the real-world operation of an IoT system.

Acknowledgments

Photo source: NATO Multimedia

Red Hat

RESEARCH QUARTERLY

VOLUME 4:4

Feature

Measuring open source success: developing analysis for actionable insights

Project Aspen plans to enable quantitative open source community health analysis for all.

by Cali Dolfi

About the Author
Cali Dolfi is a Data Scientist in the Open Source Program Office at Red Hat. She is actively involved in mentoring women in computer science and is a part of the CHAOSS community.

Organizations are increasingly adopting open source software development models and open source aspects of organizational culture. As a result, interest in how open source communities succeed is reaching an all-time high.

Until recent years, measuring the success of open source communities was haphazard and anecdotal. Ask someone what makes one community more successful than another, and you will likely get observations such as, "The software is great, so the community is too," or "The people in this community just mesh well." The problem with these evaluations is not that they are necessarily wrong, but that they don't provide information that others can use to reproduce successful results. What works for one community is not necessarily going to work for another.

Research universities, businesses, and other organizations interested in determining what makes open source projects successful have begun to collaborate on finding ways to measure aspects of community in a qualitative and data-driven way. One of the more prominent efforts is CHAOSS, a Linux Foundation project focused on creating metrics, metrics models, and software to better understand open source community health on a global scale. Unhealthy projects hurt both their communities and the organizations relying on those projects, so identifying measures of robustness isn't just an interesting project—it's critical to the open source ecosystem.

From the Red Hat Open Source Program Office (OSPO) perspective, CHAOSS was a very good answer to a pressing set of questions. First, how should community health be defined? Second, as metrics begin to take shape, how can we transition from reacting to one-off requests for data-based information about a given community to creating an entire process pipeline, literally and theoretically, for this work? The development of Project Aspen is the culmination of this pipeline, which will ultimately bring community data analysis to Red Hat and anyone else who can use it in the open source community.

COLLECTING DATA

In 2017, Harish Pillay of Red Hat's OSPO created Prospector, which was a huge inspiration

for what would become Project Aspen. Prospector aimed to present information from core data sources in a graphical dashboard, giving users thresholds to gauge whether they should do additional analysis. This resonated with CHAOSS' goal to better understand the health of open source communities, and Prospector was donated to CHAOSS and archived in July 2017. From a technical and theoretical standpoint, Project Aspen builds on the trail that Prospector first blazed.

Aspen is backed by a database generated from the Augur Project, a CHAOSS-based project that collects, organizes, and validates the completeness of open source software trace data. With this database, we can store all types of data points around the Git-based repositories from which we collect data, such as pull requests, reviews, and contributors. The data is already collected and cleaned, which, from a data science perspective, is where the most significant time drains occur. The continued data collection allows us to act agilely when questions arise. Over time, we will grow our pipeline to collect data from many other avenues in addition to Git-based repositories, such as Stack Overflow and Reddit.

As Augur regularly collects data on our selected repositories, the data is updated within a week and cleaned. With all the data collection and most preprocessing already complete, we are much better equipped to answer the analysis questions we receive and generate our own questions too. No matter where the questions come from, the same analysis process is necessary.

For every visualization or analysis, community leaders should consider these questions:

- What perspective are you looking to gain or give?
- What question can you directly answer from the data available to you?
- What assumptions am I making, and what biases may I hold?
- Who can I work with to get feedback and a different perspective?

Everyone's individual experiences and expertise impact the lens through which they look at a problem. Some people have experience in code review, while others' expertise lies in community management. How can we start comparing community aspects like apples to apples instead of oranges? Quantifying what people in different roles in open source are looking at when examining a project community can address this problem.

Community metrics empower all members to communicate in a common domain and share their unique expertise. Different perspectives lead to further insights, and Project Aspen uses data to make those insights more accessible to the entire community through data visualizations.

ASSUMPTIONS VS. ANALYSIS

Analysis is a tool for narrative building, not an oracle. Data analysis can help take the ambiguity and bias out of inferences we make, but interpreting data is not simple. A bar chart showing an increase in commits over time is not, by itself, a positive indicator of community health. Nor is a stable or decreasing number always a negative

Identifying measures of robustness isn't just an interesting project—it's critical to the open source ecosystem.

sign. What any chart gives you is more information and areas to explore.

For instance, you could build from a commits-over-time visualization, creating a graph that plots the "depth" of a commit, perhaps defined as the number of line changes. Or you could dive into the specific work of your community to see what these trends actually represent.

Comparing an issues-over-time graph (**Figure 1**) to an issues staleness graph (**Figure 2**) is a great illustration of why perspective matters. These visualizations reflect the same data but reveal completely different insights. From the issue staleness graph, we can see not only how many issues are open, but how many have been open for various time intervals.

Figure 1 shows that over many months there is relative consistency in how many issues are opened and closed. On the other hand, Figure 2 highlights the growing amount of issues that have been open for over 30 days. The same data populates each graph, but a fuller picture can only come from seeing both. By adding the perspective of the growth in issue staleness, communities can clearly see that there is a growing backlog of issues and take steps to understand what it means for their community. At that point, they will be well-equipped to devise a strategy and prioritize actions based on both good data and thoughtful analysis.

USING DATA WISELY

Including multiple points of view also provides much-needed insight and helps guard against false positives and gamification. Economists have

Figure 1. *Issues over time from 8Knot community data*

Figure 2. *Issue staleness from 8Knot community data*

a saying: "When a measure becomes a target, it ceases to be a good measure." In other words, measures used to reward performance create an incentive to manipulate measurement. As people learn which measures bring attention, money, or power, open source communities run the risk of encouraging actions taken just to play the system. Using multiple perspectives to define success will keep your metrics meaningful, so they have genuine value in maintaining your community.

To that end, Project Aspen is an exciting tool for building your own knowledge and making better decisions about communities. Whether you want to understand where your community is most vulnerable or the seasonality of activity within the community, having quality data to inform your analysis is essential. To see some of the work being done around community data analysis, please check out our GitHub organization or the demo 8Knot app instance.

MOC ALLIANCE
WORKSHOP

March 20th - 21st, 2023

Boston University George Sherman Union (GSU),
775 Commonwealth Ave, Boston, MA 02215

The workshop will bring together our unique community of Research IT, researchers, users, and industry to celebrate what has been accomplished and help define the MOC Alliance strategy moving forward

More info and registration:
www.massopen.cloud/2023-workshop

Follow the MOC Alliance on LinkedIn as they create the world's first open cloud

@mass-open-cloud

Housed at:

Boston University Rafik B. Hariri Institute for Computing and Computational Science & Engineering

MOC ALLIANCE

Red Hat

RESEARCH QUARTERLY

VOLUME 4:4

Feature

Yuga: A tool to help Rust developers write unsafe code more safely

Some bugs in unsafe Rust arise from errors so easy to make that they are easily overlooked. Researchers have developed a new analyzer to find them.

by Baishaki Ray, Vikram Nitin, Anne Mulhern, and Sanjay Arora

Rust, a programming language that did not exist just 10 years ago, is now well known and being adopted widely for a variety of projects, from static website engines to the Linux kernel. Rust is considered a safer alternative to languages like C and C++ due to its ownership model, which prevents the mutable sharing of objects. This restriction, which does not exist in C or C++, prevents a large class of programming errors. The ownership model also enables the automatic reclamation of previously allocated memory without garbage collection and without requiring the programmer to write explicit instructions for memory management.

The benefits of this language design must be paid for, in two ways. First, Rust is hard to learn and use, requiring every developer to become proficient with its lifetime annotations and associated concepts as well as its sophisticated type system. Second, sometimes the constraints imposed by Rust are simply too confining. When this is the case, the developer may take advantage of "unsafe Rust."

Unsafe Rust is actually unsafe. Within an unsafe code region, marked by the "unsafe" keyword, many safety guarantees provided by the Rust language do not apply. Unsafe Rust resembles C far more closely than safe Rust. For example, while Rust allows pointers, just as C does, only in unsafe Rust is it permitted to dereference a pointer, that is, to read from the location pointed at by the pointer under the assumption that the pointer points to a valid memory location and that memory location contains a value of the correct type.

While it may seem that any sensible developer would eschew unsafe Rust, it turns out that unsafe Rust can be very useful in particular situations. Obvious examples include bindings for C libraries and the manipulation of pointers that may be necessary to implement a data structure. When a developer chooses to use unsafe Rust in a library, it becomes their responsibility to ensure that the library offers all the guarantees to a client that it would necessarily have if the library were written without any unsafe regions. In other words, the client should be able to use the library in the same way regardless of whether or not the library contains unsafe regions. Ensuring this can be a formidable challenge for the library developer. This is evidenced by the increasing number of reported security vulnerabilities in Rust libraries, or "crates." In 2018, there were 26 reported vulnerabilities, but in 2021, there were 141.

This research project aims to help the Rust developer by automatically detecting errors in the implementation of unsafe regions in a Rust project.

IMPLEMENTATION

This project introduces the Rust analyzer and bug finder Yuga, which implements an analysis to detect

and report a set of characteristic unsafety-related bugs in Rust.

The analysis is a lightweight static analysis implemented as an extension of Rudra, an open source static Rust analyzer and bug finder implemented in Rust.

The analysis proceeds in several phases for a particular Rust crate:

1. Identify all struct definitions that contain pointers.

2. Identify all functions that have any of these structs in their signatures.

3. Examine each function signature to determine if it contains a characteristic bug pattern.

4. For any functions that contain a characteristic pattern, perform a further taint analysis on the function bodies to reduce the number of false positives.

Our tool uses two different Intermediate Representations (IRs) within the Rust compiler: the High-level IR (HIR) and the Mid-level IR (MIR). We use the HIR to get structure definitions and function signatures for steps 1-3 and the MIR to get function bodies in step 4. The entire system pipeline is shown in **Figure 1**.

RESULTS AND EVALUATION

From crates.io, the Rust package repository, we selected the top 1,600 most downloaded crates. From those, we excluded crates that would certainly fail to show any errors under our analysis, because they did not include the combination of pointers and lifetime annotations that our analysis examines.

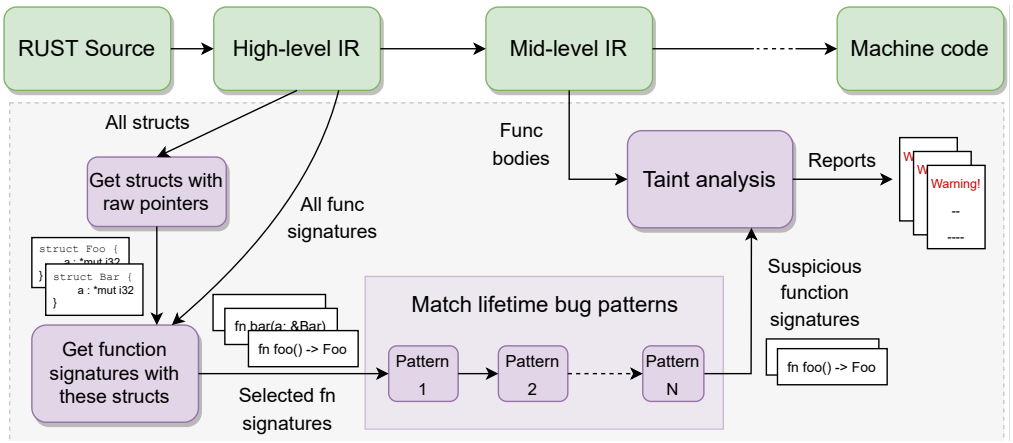


Figure 1. *Analysis framework*

These exclusions left us with 320 packages that might yield interesting errors. In these 320 packages, the analysis found 26 possible bugs. On examination, six of these proved real, 12 were false positives, and the remainder were a category we call "semantically safe." Semantically safe code is code in which there is a potential bug, but it can be proved that the code can never be used in conditions that would cause the identified bug to manifest. Eliminating semantically safe bugs automatically would require reasoning about the semantics of every method in a type's implementation.

BUG PATTERNS

In the tradition of the FindBugs tool, Yuga inspects code looking for characteristic bug patterns.

There are three patterns that Yuga looks for:

- The Get pattern
- The Set pattern
- The Create pattern

Bug pattern example: Get

In this section, we examine the Get pattern in detail. To discuss the pattern, we must first acquaint the reader a little more with the Rust language.

First, Rust places a very strong syntactic constraint on the references that may exist to a location in memory, which may be a simple value, such as an integer, or a more complex value, such as a struct. In a single, carefully calculated contiguous region of source code, it is forbidden to have a mutable reference and any other reference, either mutable or immutable, to the same memory location.

In early versions of Rust, the contiguous region was just the scope, demarcated by matching braces. Early Rust code was, consequently, rather full of matching braces used to divide a single large region into smaller ones so that two distinct references to the same memory could exist in the separate scopes defined by the braces. These days, Rust uses a liveness analysis to

```
fn main() {
    let mut value = 3;
    let ref1 = &mut value;
    let ref2 = &mut value;
    *ref1 += 1;
    *ref2 += 2;
}
```

Figure 2. *Rust code that attempts to allow two distinct mutators of a single memory location to interact*

determine its contiguous regions and is generally able to make them small enough so that extra braces are seldom required.

The reason for this restriction is to prevent two distinct mutators of a single memory location from interacting. This restriction:

- Prevents a large class of bugs and clears the programmer's mind of any concerns about that class of bugs since the compiler makes them impossible
- Allows aggressive compiler optimizations so that Rust programs can be fast

A very simple infraction of this rule is shown in **Figure 2**.

Here we have one mutable value, called "value," and two references to that value, both of which have been declared mutable, and both of which set that value.

The compiler forbids this: see **Figure 3.**

The reader is encouraged to copy this snippet of code into the Rust playground and explore variations on it, seeing which elicit a compiler error and which do not.

At first sight, and with this simple example, this prohibition may not appear terribly useful. But experienced developers will undoubtedly recall encounters with a variety of bugs that this restriction would have prevented. For example, there might be two nested loops where a reference to some struct exists in the outer loop, and a different reference to the same struct exists in the inner loop. The outer reference is overlooked, and the developer's mistake arises from considering only the behavior of the reference in the inner loop.

Just as importantly, the Rust developer knows that once the mutable reference is found, no other mutable reference exists. This is a luxury that the Python programmer or the C programmer does not have. This simplifies debugging and even eliminates certain types of bugs, which is why Rust is gaining popularity in settings like kernel code and security-critical applications.

A developer can rely on this assurance unless there is some unsafe Rust code that destroys this guarantee. In that case, the Rust developer's confidence is ill-founded.

To show how this can happen, we begin with the Rust struct in **Figure 4**.

The key point here is that on the outside Foo is Rust, but internally it contains a mutable pointer to a String, as the declaration **`inner: *mut String`** shows. So far, nothing is untoward, but remember that a pointer dereference is always considered unsafe. In other words, the pre-conditions for a bug have been established.

Note that the inner field of the Foo struct is private. This means that the developer can know all the behaviors of the Foo struct just by fully understanding its associated implementation.

Figure 5 adds a single method to the implementation.

The parameter declaration, **`&'b mut self`**, is a shorthand for **`self: &'b mut Foo`**, that is, self has the type Foo.

We add a convenience method for creating a Foo: see **Figure 6**.

```
error[E0499]: cannot borrow `value` as mutable more than once at a time
 --> src/main.rs:4:16
  |
3 |     let ref1 = &mut value;
  |                ---------- first mutable borrow occurs here
4 |     let ref2 = &mut value;
  |                ^^^^^^^^^^ second mutable borrow occurs here
5 |     *ref1 += 1;
  |     ---------- first borrow later used here

For more information about this error, try `rustc --explain E0499`.
```

Figure 3. *Rust compiler error from the code in Figure 2*

And then in **Figure 7**, we demonstrate that the Rust rules work and prevent us from having two mutable references to Foo's inner field at the same time.

We see the same compiler error as before: **`ref1`** and **`ref2`** are both mutable references to the same String, so this code is forbidden. The difference between this example and the original one is that the shared mutable reference is not to the simple integer value 3, but to the String pointed at by the inner field of the Foo struct.

However, this correctness depends on the developer getting the signature of the **`get()`** method exactly correct. There is a very natural mistake that the developer might have made. An incorrect **`get()`** method is shown in **Figure 8**.

The difference between the incorrect **`get()`** method and the correct one is only a single character. The method body is unchanged, and its behavior is the same. Yet with that change, the example **`main()`** method that previously would not compile does, and there are two coexisting mutable references to the same memory location.

A full explanation of why the change makes a difference is outside the scope of this article. However, the vital point is that in the correct implementation of the **`get()`** method, all the lifetime parameters, easily identified by their tick prefix, are the same parameter, **`'b`**. Contrariwise, in the incorrect implementation of the **`get()`** method, one of the lifetime parameters is the anonymous lifetime, **`'_`**, which is

```
struct Foo<'a> {
    inner: *mut String,
    #[allow(dead_code)]
    extra: &'a u64,  // ignore this value
}
```

Figure 4. *In some Rust code declared unsafe, the mutable pointer provides the pre-conditions for a bug*

```
impl <'a> Foo<'a> {
    fn get<'b>(&'b mut self) -> &'b mut String {
        unsafe {&mut *(self.inner)}
    }
}
```

Figure 5. *Adding a method to the code in Figure 4*

```
fn create<'a>(bar: &'a mut String, extra: &'a u64) -> Foo<'a> {
    Foo{inner: bar, extra}
}
```

Figure 6. *Now add a convenience method to the Figure 5 code*

```
fn main() {
    let mut value = "Hello".to_string();
    let extra = 32;
    let mut foo = create(&mut value, &extra);

    let ref1 = foo.get();
    let ref2 = foo.get();
    print!("{}, {}", ref1, ref2);
}
```

Figure 7. *Now add some code that should not compile, because it creates two mutable references to the same field at the same time*

```
impl <'a> Foo<'a> {
    fn get<'b>(&'_ mut self) -> &'b mut String {
        unsafe {&mut *(self.inner)}
    }
}
```

Figure 8. *The same convenience method as Figure 6, with one character changed that allows the buggy code to compile*

different from all the other lifetime parameters in the method signature.

In the incorrect example, by including the anonymous lifetime parameter, the developer has failed to properly inform the compiler that the memory contained in a Foo struct coincides with the memory in the String that the **`get()`** function returns. Consequently, the compiler does not prevent the incorrect code in the example,

and the contract between the Foo implementation and its clients is broken.

At this point, we have demonstrated that a pattern like the one above, where self's lifetime parameter is not the same as the returned value's lifetime parameter, may result in a bug. However, this pattern does not necessarily imply a bug. Whether or not there is a bug depends on how the body of the method is implemented.

Of the four steps of the analysis described earlier, it turns out that the first three have been completed.

1. The struct definition, Foo, has been identified as containing a pointer.

2. The `get()` method, which has a parameter of type Foo, has been identified.

3. The incorrect `get()` method has been shown to match a bug pattern.

Step 4, the taint analysis, is then run. It concludes that the incorrect `get()` method has been correctly identified because it determines that there is a flow from the self parameter to the result.

Both the Set and Create patterns have similar characteristics. Each pattern identifies a way in which the developer is likely to combine unsafe operations with incorrectly chosen lifetime annotations and thereby permit their code to be used so that it breaks Rust's safety guarantees.

CONCLUSION

Yuga's analysis successfully identifies bugs that other competing analyses, Rudra and MirChecker cannot. It is successful because it looks for the characteristic patterns of a particular class of bug, namely those where the developer specifies lifetime annotations incorrectly, so the compiler allows behaviors it should prevent. This can only happen in the presence of unsafe code; without unsafe code, the compiler will always be able to reject incorrect lifetime annotations.

Because it identifies likely bugs that can easily escape a developer's notice even over a considerable period of time, Yuga's analysis has the potential to be a practical tool for the working programmer. One drawback is that its false positive rate is inconveniently high. This false positive rate could be reduced by an analysis that was more precise than the existing taint analysis. Yuga's analysis could also be extended to a wider variety of likely bug patterns. Currently, all Yuga's bug patterns involve lifetime parameters, but mutability modifiers could also be considered.

About the Author
Baishaki Ray
is an associate professor of computer science at the Fu Foundation School of Engineering and Applied Science, Columbia University, New York.

About the Author
Vikram Nitin
is a fourth-year PhD student at Columbia University advised by Dr. Baishakhi Ray. His research interests are broadly in source code modeling and static analysis.

About the Author
Anne Mulhern
Dr. Mulhern, Red Hat Principal Software Engineer, is the technical lead of the Stratis project, which is written primarily in Rust.

About the Author
Sanjay Arora
works at Red Hat's AI Center of Excellence and is mainly interested in the application of machine learning to low-level systems.

RESEARCH QUARTERLY

VOLUME 4:4

Red Hat

Column

Composing a research symphony

Many talents contributed to one goal: a shared production-level research cloud.

by Heidi Picher Dempsey

It was a chilly morning at Boston University, and I was looking for a quiet place to gather my thoughts and do some writing. I passed two painters covering up scuffs on the white walls and a man with a floor machine busily tracing circles over the wide linoleum. Ducking into an empty conference room, I sat down and started banging away on my laptop. My creative reverie was shattered a minute later when a blast of Bach filled the room. I had wondered on previous visits whether the organ in the adjacent conference hall was ever used, and now I had my answer. At first I was annoyed by the disturbance, but I kept working, and soon I began to appreciate the interplay of swirling notes and coalescing sentences as the musician in the next room became my unwitting collaborator, bringing a new and very unusual contribution to this writing project.

Collaborators from Boston University (BU), Harvard University, and Red Hat have been leading a multiyear project to build and release a new computing environment that supports research and experimentation with a wide range of resources. The production part of this project, called the New England Research Cloud (NERC), depends on research IT professionals, software developers, systems engineers, project managers, faculty, and students for its success. It also depends on long-term partnerships with the Massachusetts Green High Performance Data Center (MGHPCC) and the MOC Alliance, as well as the National Science Foundation's continued dedication to Computer and Information Sciences Research (NSF CISE).

This is not a greenfield project. Each collaborator brought their own kit of tools and processes that served them well in their area of expertise. One of our biggest challenges was finding a way to fit the best pieces of each kit together in a new system, figuring out what was still missing, and working together

About the Author

Heidi Picher Dempsey is the US Research Director for Red Hat. She seeks out and grows projects with academic and industry partners in areas such as operating systems, hybrid clouds, performance optimization, networking, AI, and security.

to fill in the interstitial space so we could create a more solid system and open it to global researchers. For example, Red Hat's OpenShift software, which builds and manages Kubernetes containers, is already a commercial release with many successful users. Harvard and BU already manage thousands of research and student accounts with a website and ColdFront software that works well for their existing university research resources. But there was no existing way to connect these two solutions. We had to create the interfaces, software, procedures, and documentation to allow self-service for global researchers who would create and manage their own projects in a container environment. New roles and access control bindings had to be defined to maintain security in an environment that was no longer limited to a single community.

The Harvard and BU IT engineers, who knew what had worked well for faculty and students so far, collaborated with developers and systems engineers from BU and Red Hat, who knew how to implement new interfaces and connect them to existing APIs that managed container users and project namespaces. The NERC resources for each project had to be shared by all members of a project but not by other NERC users. The resources also needed to be managed by NERC administrators and monitored carefully to find and alert operations staff to potential issues before they caused problems for researchers and students. We needed the ability to track and report usage with an eye to predicting growth and bringing new resources into clusters quickly when necessary. Legal and regulatory requirements had to be met, and the whole project had to be tracked and managed carefully.

Most importantly, the project needed to follow open source practices adapted to deliver an entire hardware, software, and operations system. It should be possible for anyone to understand at a detailed level what the system components are and how to compose them from the bare hardware pieces through the operating system and application stack, including all the supporting production operations infrastructure. There are some intentional

exceptions to this rule necessary for privacy and security, but we default to open wherever possible.

Twelve public repositories and 18 named contributors later, it's possible to examine, use, and improve the detailed solution of this example problem for the OpenShift part of NERC. (An extensive project tracker built in Asana and many months' worth of meeting notes are not included in these repositories but can be shared with anyone joining the project.) Explore the OCP on NERC GitHub repository to see high-level documentation, detailed configurations, and references to other open source software projects like Vault, Prometheus, and Grafana that are essential parts of the solution. Go to the NERC website to request your own project, find user guides, and read more about this environment and the other NERC offerings, like OpenStack virtual machines and bare metal computing. On March 20-21, 2023, join the in-person MOC Alliance Workshop with engineers and researchers who are building even more varied ecosystems to support research and explore the future of open computing.

Building this environment has not been easy, and it has taken a continued concentrated effort from many people who were already busy with other critical projects and daily responsibilities. Success has depended on the team's willingness to question what we were doing, what was missing, how we could broaden our reach, how things could fail, and what we could do about it. The effort was often messy and underresourced, but the team's determination and dedication brought us closer to the goal. Although this project will remain an unfinished symphony for as long as research computing challenges evolve, we can all be proud of composing a new research and engineering score together.

Research project updates

Each quarter, Red Hat Research Quarterly highlights new and ongoing research collaborations from around the world. This quarter we highlight collaborative projects in Israel at The Technion, The Ben Gurion University of The Negev, Ariel University, Reichman University, and The Hebrew University.

PROJECT: CCO: Cloud cost optimizer

ACADEMIC INVESTIGATOR:
Prof. Assaf Schuster (Technion)

RED HAT INVESTIGATOR:
Ilya Kolchinsky

Researchers have completed the project successfully. CCO retrieves information regarding the prices and capacities of all instance types, regions, availability zones, and so forth, as advertised by the cloud provider. Based on this information, CCO calculates the optimal allocation of workload components to minimize the overall cost.

A fully functional version of the CCO supporting AWS, Azure, and hybrid mode (splitting the workload across multiple cloud providers) is now available for use. Incorporating advanced optimization techniques, CCO can create close-to-optimal deployment plans for workloads of hundreds to thousands of components within mere seconds. Watch for more details in an upcoming article of RHRQ, and visit the GitHub repository.

PROJECT: AppLearner: learn and predict the resource consumption patterns of your OpenShift application

ACADEMIC INVESTIGATOR:
Prof. Assaf Schuster (Technion)

RED HAT INVESTIGATORS:
Ilya Kolchinsky

This project targets the problem of accurately estimating resource requirements for workloads running over the Red Hat OpenShift Container Platform and adjusting these estimations during the course of application execution. For most real-life applications, it is notoriously difficult to manually estimate resource consumption patterns and thus tune the pod CPU/memory requirements accordingly to avoid under- and overutilization. While there are attempts to solve this problem by on-the-fly monitoring and adaptively scaling pods when changes are detected, these solutions could be inefficient when the application behavior is highly dynamic. Instead, AppLearner is proactively defining the provisioning plan for an application by

Contact academic@redhat.com for more information on any project described here, or explore more research projects at research.redhat.com.

learning and predicting its resource consumption patterns over time.

The project is now in the research phase with the goal of identifying the most promising approach to learning and predicting CPU and memory consumption of a sample set of workloads. We expect the early prototype to be available for use by Q4 of 2023.

PROJECT: SpotOS

ACADEMIC INVESTIGATOR:
Prof. Assaf Schuster (Technion)

RED HAT INVESTIGATORS:
Josh Salomon, Gabriel BenHanokh, Orit Wasserman, and Avishay Traeger

The SpotOS project aims to devise a distributed cloud-based operating system that uses unreliable or temporarily available resources to provide a reliable and scalable execution experience with a high quality of service by harnessing the power of spot instances, resources representing the currently unused cloud capacity. While spot instances are considerably cheaper than regular instances, the cloud provider can unexpectedly reclaim them anytime. In this case, a very limited time window is given to the running application to back up its current state. SpotOS aims to overcome this limitation by providing a reliable, adaptive, self-healing, user-transparent layer with spot instances serving as the underlying unreliable building blocks.

The work on SpotOS is ongoing in multiple directions. First, researchers are building the key innovation component: the EDM (external distributed memory). With EDM, the application state is split among multiple storage units (that could be hosted on regular instances, spot instances, or a mix of both) in sufficiently small chunks to complete the evacuation on time. A limited prototype of the EDM has been tested in the lab environment, and work has begun to make it applicable to real-life workloads and use cases. Another work direction revolves around the life migration of SpotOS applications. When a spot instance is down and the customer application has to be moved to a different spot, this transition must be as seamless as possible. Achieving this smooth migration requires technical innovations, which are the focus of this subproject.

Finally, the project team is building the controller for SpotOS, a component integrating and managing the framework's various components. This includes the EDM and the migration manager, as described above, as well as other independent projects such as CCO and AppLearner.

PROJECT: Cluster autoscaling DDoS attacks

ACADEMIC INVESTIGATOR:
Prof. Anat Bremler-Barr (Reichman University)

RED HAT INVESTIGATOR:
Benny Rochwerger

YoYo attack is a new type of DDoS (Distributed Denial-of-Service) attack based on abusing the auto-scaling mechanism by causing it to oscillate between scale-up and scale-down, thus causing economic damage (EDoS) or crashing the victim application by consuming its computation resources. As its name suggests, YoYo is based on periodic bursts of high and low traffic on the target. These bursts are short and hard to detect. YoYo attack is considerably more cost-effective than a regular DDoS on Kubernetes.

The goals of this project are as follows:

- Studying the YoYo attack and its impact on OpenShift-based infrastructure
- Devising a set of best practices for cluster admins to minimize the susceptibility of an OpenShift cluster to YoYo or similar attacks
- Investigating more general strategies for mitigating this attack clas

The project is now in advanced research stages, with several published papers mainly covering the first goal. Researchers plan to invest more resources in the second and third goals during 2023.

PROJECT: Tuning QUIC protocol for Ceph workloads

ACADEMIC INVESTIGATORS:
Prof. Anat Bremler-Barr and Jonathan Plotkin (Reichman University)

RED HAT INVESTIGATOR:
Yuval Lifshitz

QUIC (Quick UDP Internet Connections) is a general-purpose transport layer network protocol designed by Google offering significant advantages over TCP, such as greatly reduced latency. This project aims to utilize the strengths of QUIC for communication between the components of Red Hat Ceph Storage, particularly between Ceph Object Gateway (RGW) and Ceph clients. To that end, researchers will study the characteristics of the workloads in the relevant use cases and adapt and tune QUIC accordingly, possibly resulting in a new protocol variation and/or implementation.

This effort is still in its preliminary stages. The goals and milestones have been approved, and a graduate student allocated to the project is ready to start.

PROJECT: CVE mining and prediction

ACADEMIC INVESTIGATORS:
Prof. Anat Bremler-Barr and Dr. Tal Shapira (Reichman University)

RED HAT INVESTIGATOR:
Keith Grant

The CVE (Common Vulnerabilities and Exposures) database contains more than 190,000 vulnerability records, of which more than 20,000 CVEs were registered in 2021. A plethora of highly useful information and invaluable insights could be extracted from CVEs and similar data sources. Acquired knowledge could be used to predict future vulnerabilities or estimate the probability of an exploit of a particular software.

However, despite the immense potential of CVE analysis and mining, this area gets little attention. The main reason is the sheer difficulty of coping with such a large volume of highly unstructured information. This project aims to implement and apply an innovative approach to harvesting knowledge from CVEs by utilizing a knowledge graph that encodes the information in the form of entities and connections.

The proposed method uses NLP (natural language processing) to parse unstructured textual data from CVEs and populate a knowledge graph which could then be used for answering user queries and/or predicting the not-yet-known parts.

The project is in the early planning stage. The participating researchers are working closely with the Product Security team to identify the most promising and valuable directions and ways of applying the knowledge graph approach to mine the most important and relevant insights from the CVE database.

PROJECT: Service mesh performance study

ACADEMIC INVESTIGATORS:
Prof. Anat Bremler-Barr and Yaniv Naor (Reichman University)

RED HAT INVESTIGATOR:
Sanjeev Rampal

A service mesh is a dedicated infrastructure layer that controls service-to-service communication over a network. It provides a way to control how different parts of an application share data. In recent years, the number of Kubernetes service meshes and systems that adopt a service mesh has rapidly increased.

Since performance has a key role in every system, performance analysis and comparison between the leading service mesh technologies could benefit the community in at least two ways. First, a detailed comparative evaluation of the alternatives could help decide which service mesh to use for a particular workload. Second, such a study has the potential to reveal fundamental flaws and limitations in the way service mesh technologies function, thus paving the way for future research tasked with addressing these flaws and improving the current state of the art.

The goal of this project is to perform this analysis and empirical evaluation. The design and the planning are now complete, and the team is working on building the corresponding experimental setup.

PROJECT: Advanced proactive caching for heterogeneous storage systems

ACADEMIC INVESTIGATORS:
Dr. Gabriel Scalosub and Dr. Gil Einziger (Ben Gurion University of The Negev)

RED HAT INVESTIGATORS:
Guy Margalit, Josh Salomon, Orit Wasserman, and Gabriel BenHanokh

Caching is one of the most effective optimization techniques in large distributed systems. However, the standard approach in the industry still relies on relatively generic policies. These methods exhibit several drawbacks. Most importantly, they are non-adaptive and reactive rather than proactive, so they cannot leverage system-specific and workload-specific patterns for making caching decisions in advance.

This project seeks to improve the performance of NooBaa, an object data service for hybrid and multicloud environments, by developing novel caching frameworks that take into account request heterogeneity and perform proactive caching decisions (also referred to as speculative prefetching).

The intended contributions of this project are as follows:

- To close the above gap algorithmically by devising new approaches for caching in storage systems incorporating heterogeneity and proactivity
- To upstream our suggested solutions by extending NooBaa

The project is progressing toward completing these two goals by the end of 2023.

PROJECT: Software diagnosis with log files

ACADEMIC INVESTIGATORS:
Dr. Meir Kalech and Dr. Roni Stern (Ben Gurion University of The Negev)

RED HAT INVESTIGATOR:
Gil Klein

This project aims to create an automated tool to identify software failures and isolate the faulty software components (e.g., classes and functions) that caused the failure without using code coverage. The core idea is to leverage the information in the system log files instead and use it as an approximation of coverage. Such a solution could be used by QE engineers tasked with testing large distributed systems (such as Kubernetes/OpenShift) that cannot efficiently and scalably support coverage.

The project launched in January 2023.