Research Statement

Ennan Zhai

Computer systems nowadays play a critical role in our daily lives: they provide heterogeneous Internet services, process business transactions, and store personal and enterprise-scale information. In case of failure or compromise of any of these systems, catastrophic consequences would follow. Nevertheless, we are faced with an ever greater challenge to guarantee the security and reliability of computer systems, given the fact that they continue to increase in scale and complexity, which makes the systems fragile and vulnerable, and complicates fault prevention and diagnosis.

My research aims to tackle the challenge of building secure and reliable systems, by integrating the usage of a variety of techniques in areas including distributed systems, programming languages, verification and cryptography. Specifically, my research approach is to first identify important security and reliability problems, then design suitable solutions in principle, and finally build and refine practical systems based on the principle. I focus on targeting high-impact problems, with the goal of developing solutions that are both practical and effective.

Using this approach, I have designed, developed, and evaluated a number of new systems, including a cloud structural reliability auditing system (*INDaaS* [OSDI'14]), a language framework for cloud auditing (*RepAudit* [OOPSLA'17a]), a tracking-resistant anonymous reputation system (*AnonRep* [NSDI'16]), software configuration verification systems (*ConfigC* [CAV'16] and *ConfigV* [OOPSLA'17b]), a framework for automatic repair of firewalls (*FireMason* [FMCAD'17]), and a spam-resistant meta-data system (*SRaaS* [VLDB'17]). The application-specific focus of my approach has brought about fruitful collaborations with researchers from multiple fields, including systems, networking, programming languages, verification, cryptography, and database. Moreover, my research efforts have been recognized for their practical significance and impact; for example, AnonRep has been deployed by blockchain companies (Katalysis and Oxchains), and INDaaS has been reported in The Register [News'16].

In the following sections, I will discuss three pieces of my recent work in the areas of system reliability (§1), security and privacy (§2), as well as verifications (§3), respectively. In the final section (§4), I will outline my future research directions on topics, including programming languages, cybersecurity and system reliability.

1 Preventing Correlated Failures in Cloud-Scale Systems

Correlated failures in the cloud. Distributed systems today pervasively rely on redundancy techniques, *e.g.*, replicating important states and functionality across multiple servers, to ensure availability and reliability. In complex multi-layered hardware/software stacks, however—especially in the clouds where many independent businesses deploy interacting services on common infrastructure—seemingly independent systems may share deep, hidden dependencies such as network infrastructure and software components. The failure of these unexpected common dependencies can lead to a *correlated failure*—*i.e.*, the simultaneous unavailability of multiple end-hosts—which in turn causes service downtime, undermining redundancy efforts.

The state-of-the-art efforts on handling correlated failures focused on diagnosing root causes *after* service outages occur. However, these post-failure forensics (*e.g.*, accountability, provenance, and troubleshooting systems) not only require prolonged failure recovery time in the face of structurally complex services, but also fail to prevent expensive service downtime: the average cost of a datacenter outage in 2016 was $740,357.

Independence-as-a-Service (or INDaaS) [OSDI'14]. *Disease prevention is better than diagnosis.* I proposed and developed *Independence-as-a-Service* (or INDaaS), an architecture to audit the independence of redundant systems proactively, thus preventing correlated failures at the early stage. For a given redundant system, R, INDaaS first utilizes pluggable dependency acquisition modules to automatically collect the structural dependency information (including network and software dependencies) from a variety of sources underlying R. With this information, INDaaS then models the structure reliability of R using fault graph, which is a special data structure representing a system as a directed acyclic graph with logical gates. Finally, INDaaS analyzes the generated fault graph, not only quantifying R's independence, but also identifying dependencies potentially resulting in correlated failures.

However, correlated failures can be caused not only by dependencies within *one* single cloud provider, but also by dependencies across *multiple* different cloud providers. Recent years, increasingly more application providers (*e.g.*, iCloud and Zynga) redundantly rent different cloud providers (*e.g.*, Amazon S3 and Microsoft Azure storage); however, these cloud providers may share vulnerable third-party dependencies, such as buggy software libraries, DNS services, and IXP routers. Application providers cannot readily know how independent the lower-level cloud providers they build on redundantly really are. This is because the relevant common dependencies are often proprietary internal information, which cloud providers do not normally share. INDaaS employs a private set intersection cardinality protocol to evaluate the independence of all the alternative inter-cloud replication adoptions for application providers, while preserving the business privacy of cloud providers. With the help of INDaaS, an application provider can select the most independent inter-cloud replication deployment for her application, preventing correlated failure risks at th best efforts.

RepAudit [OOPSLA'17a]. For the single-cloud auditing case, INDaaS has three limitations. First, due to the lack of language support, it is difficult for the operators to express auditing tasks for diverse purposes. Second, the fault graph analysis approach in INDaaS is not scalable to the large cloud system with tens or hundreds of thousands of components. Finally, once the operators notice any correlated failures risks, asking them to manually improve the current deployment can be difficult and error-prone. I designed and developed a new system, named RepAudit, that is able to address the above three limitations. First, RepAudit offers a declarative domain-specific auditing language, RAL, for the operators to write auditing programs easily expressing diverse auditing tasks. Second, RepAudit presents a high-performance auditing engine that not only parses the input RAL-program, but also calls the corresponding auditing primitives that in principle transform fault graphs into Boolean formulas, and then solve the formulas by tweaking and leveraging various verification tools, such as the MinCostSAT solver and model counter. Such a design enables RepAudit to offer 100% accurate auditing results and 300$\times$ faster than INDaaS system. Finally, I equip RepAudit with a repair engine that accepts reliability goals specified by the operators and then automatically generates improvement plans meeting those goals, *e.g.*, for lower failure probability.

2 Tracking-Resistant Anonymous Reputation

User activity tracking in reputation systems. Online services (*e.g.*, eBay, Yelp, and Stack Overflow) employ reputation systems to help users evaluate information quality and incentivize civilized behavior, often by tallying feedback from other users (*e.g.*, "likes" or votes) and linking these scores to a user's long-term identity. User reputations increase or decrease based on this feedback, and reputation affects how widely a user's future posts are viewed. This long-term identity linkage between user behaviors and reputation, however, enables user tracking and appears at odds with strong privacy or anonymity. For example, attackers have successfully revealed eBay users' sensitive purchase histories by analyzing only pseudonyms' transactions and feedback. As privacy has become a major concern for online users, I aim at answering the question: is it possible to build a practical anonymous reputation system that combines the benefits of reputation with the privacy afforded by fully anonymous, unlinkable messaging? Achieving this goal is very challenging, since the requirement to associate users with their historical activities seems to preclude anonymity—*i.e.*, establishing reputation while maintaining unlinkability of activities appears to be a paradox.

Prior efforts have been proposed to address this problem, but none have yet proven practical or sufficiently general for realistic deployment. Specifically, the state-of-the-art efforts *either* rely on a centralized party that behaves honestly (*e.g.*, blind signature-based anonymous reputation), *or* fail to support negative feedback for confiscating reputations from misbehaving users.

AnonRep [NSDI'16]. I designed, developed and evaluated AnonRep, the first practical anonymous reputation system offering the benefits of reputation without enabling long-term tracking. AnonRep employs a small set of individual servers that run verifiable shuffle protocol to assign one-time pseudonyms to clients in different messaging rounds. With the one-time pseudonym, the clients can anonymously post messages, and can verifiably tag with their reputation scores without revealing their long-term identities. On the other hand, AnonRep adapts linkable ring signature primitive to reliably tally other users' feedback (*e.g.*, likes or votes) without knowing voters' identities, *while* maintaining security against score tampering or duplicate feedback. A working open-source prototype of AnonRep has been developed and my experiments demonstrate that AnonRep scales linearly with the number of participating users. My AnonRep prototype has been used by several blockchain startup companies, such as Katalysis and Oxchains.

3 Software/Network Configuration Correctness Reasoning

Software misconfiguration. System failures resulting from misconfiguration are one of the major reasons for the compromised reliability of software systems today. The state-of-the-art efforts either diagnose configuration errors after failures occur, or prevent simple configuration errors by analyzing both system code and configuration files. Verifying the correctness of system configurations via machine-checkable proof is challenging, because 1) software configurations are typically written in poorly structured and untyped "languages", and 2) specifying rules for configuration verification is difficult in practice.

ConfigV [OOPSLA'17b]. ConfigV is the first automatic framework for software configuration verification. ConfigV works in the following three steps. First, ConfigV parses a training set containing many configuration files (not necessarily all correct) into our proposed well-structured and probabilistically-typed intermediate representation. Second, based on the association rule learning algorithm, ConfigV learns, derives and refines rules (*i.e.*, specifications) from these intermediate representations. These rules describe which properties correct configuration files need to satisfy, for example: the configuration ordering, the configuration correlation, the configuration type, and the configuration pairing—none of the state-of-the-art efforts can detect or specify these errors. Finally, given a configuration of interest, C, ConfigV can verify whether C meets our learned rules. ConfigV was evaluated by verifying public configuration files on GitHub, and our results show that ConfigV successfully detected many configuration errors in systems on GitHub.

Misconfigured enterprise-scale firewalls. Firewalls are widely deployed to manage the enterprise networks. Because the enterprise-scale firewalls contain hundreds or thousands of rules, ensuring the correctness of firewalls—*i.e.*, the configuration rules meet the specifications of the operators—is an important but challenging problem. Although existing firewall diagnosis can identify potentially faulty rules, they offer the operators little or no help with automatically *fixing* these faulty rules. Fixing faulty rules in enterprise-scale firewalls is non-trivial, because: 1) it is difficult for the operators to easily specify the errors, and 2) a simple repair may introduce new problems in somewhere else.

FireMason [FMCAD'17]. FireMason is the first practical effort that offers automated repair by example for firewalls. Once a network operator observes undesired behavior in her network, she can provide input/output examples that comply with the intended behaviors, under the assumption that examples are much easier specified by the operator than conventional specifications (*e.g.*, written in the first-order logic). Based on the input/output examples, FireMason can automatically synthesize new firewall rules for the original firewall. This newly generated firewall rules correctly handle packets specified by the examples, while maintaining the rest of the behaviors of the original firewall (*i.e.*, without side-effect). Through a conversion of the firewalls to SMT formulas, we offer formal guarantees that the change (*i.e.*, repair) is correct. We evaluated FireMason based on real-world case studies from firewall repair questions on the Stack Overflow, and the results showed that FireMason can efficiently and accurately find correct repairs.

4 Future Research Plans

As computer systems become a greater part of our society, they will operate in a wider variety of networks and interact with more complex environments. Therefore, future computer systems will be faced with more security and reliability challenges. To meet these challenges, my future research targets three general directions: programming languages, cybersecurity and system reliability.

Direction 1 - Programming languages: New languages for security and reliability. Correctly developing secure and reliable systems is challenging, because the building blocks used to guarantee system security and reliability—*e.g.*, information flow control, advanced cryptographic protocols, and fault tolerance—have been designed too complex to be implemented correctly in practice. This challenge has led to a rise of developers who are writing buggy or vulnerable code in their systems. One of the ways to addressing this fundamental issue is to offer simple, expressive, and domain-specific languages (or abstractions) that alleviate the burden of writing complex systems, and allow focus on the problems at hand rather than tooling and language quirks. In addition to ease of use in the targeted problem domains, such abstractions and languages allow for verification of accurate specification implementation and guarantees of certain security properties. I am interested in constructing new abstractions and language frameworks that enable developers to implement and verify complex systems with ease and accuracy. My success of RepAudit is encouraging me to pursue this direction.

In addition, given the fact that misconfiguration has become one of the major root causes for system outages, the language support for easily writing configurations is important. However, compared to writing the system code, there are few language for configurations. A good configuration language should make configuration writing become easier and reliable, as well as make the configuration verifiable. I am interested in exploring new language abstractions for software and network configurations. Besides the language for the configuration, I am also interested in proposing configuration verification and repair tools. These tools can work together for the operators not only to easily write correct configuration, but also to verify and fix system configuration errors. My past experience in developing ConfigV and ConfigC indicates promise in the further exploration of such tooling.

Direction 2 - Cybersecurity: Securing Internet services via blockchain. Important Internet services for identity and data management are being faced with many threats. For example, attackers can construct man-in-the-middle (MITM) attacks to get the Certificated Authority (CA) to provide Internet users with forged public keys and decrypt private information. Distributed Denial of Service (DDoS) attacks have also crippled DNS providers, *e.g.*, Dyn, as recently as in 2016. The fundamental root causes of the above issues are either due to the inability to publicly perform system audits or because of single-point-failures. I am interested in addressing the above cybersecurity problems by leveraging blockchain technology to design such systems to be decentralized and publicly auditable, while maintaining user privacy. For example, in the MITM-attack case, a blockchain approach can make the CA system's ledger publicly auditable. Such a solution allows users to put their public keys in published blocks distributed over the participating nodes in a network, meaning that keys become immutable and easier for fake keys to be detected. Additionally, in the DDoS example above, a decentralized blockchain model makes it difficult for attackers to simultaneously overload all nodes in a distributed network. For applications in which user data is sensitive, I aim to utilize my experience in security area to design new cryptographic protocols that allow for privacy and anonymity preservation in the blockchain setting.

Direction 3 - System reliability: Predicating catastrophic failures caused in system runtime. Catastrophic failures—*i.e.*, service failures that affect all or the majority of users—resulting from errors introduced in system runtime have accounted for the largest proportion of real-world cloud outages. For example, new software bugs introduced by system updates can lead to correlated failures across global redundant instances, while misoperations of the datacenter operators can also result in cloud-scale outages. The state-of-the-art efforts are focused on either diagnosing root causes after outages occur, or proactively checking potential issues through static analysis. Bugs and misoperations introduced in system runtime are challenging to predicate ahead of time, because many of such issues can be triggered or observed only when the systems are actually run in the real environment. In other words, existing checking approaches are too "static" to capture these failure sources. I am interested in predicating catastrophic-failure sources by 1) emulating failure-inducing updates and operations based on tracing and log information, and 2) extracting potential failure sources based on the similarity between emulated updates and actual system executions. I am also interested in constructing runtime verification approaches to proactively alert the operators to potential catastrophic failure risks.

References

[OOPSLA'17a] Ennan Zhai, Ruzica Piskac, Ronghui Gu, Xun Lao, and Xi Wang. *An auditing language for preventing correlated failures in the cloud.* In 32th ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), Oct. 2017.

[OOPSLA'17b] Mark Santolucito, Ennan Zhai, Rahul Dhodapkar, Aaron Shim, and Ruzica Piskac *Synthesizing configuration file specifications with association rule learning.* In 32th ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), Oct. 2017.

[FMCAD'17] William Hallahan, Ennan Zhai, and Ruzica Piskac. *Automated analysis and repair by example for firewalls.* In 17th Formal Methods in Computer-Aided Design (FMCAD), Oct. 2017.

[VLDB'17] Ennan Zhai, Zhenhua Li, Zhenyu Li, Fan Wu, and Guihai Chen. *Resisting tag spam by leveraging implicit user behaviors.* In 43th International Conference on Very Large Data Bases (VLDB), Aug. 2017.

[CAV'16] Mark Santolucito, Ennan Zhai, and Ruzica Piskac *Probabilistic automated language learning for configuration files.* In 28th Computer Aided Verification (CAV), Jul. 2016.

[News'16] The bigger they get, the harder we fall: Thinking our way out of cloud crash. Available at: http://www.theregister.co.uk/2016/07/29/bryan_ford_bigger_icebergs/.

[NSDI'16] Ennan Zhai, David Isaac Wolinsky, Ruichuan Chen, Ewa Syta, Chao Teng, and Bryan Ford. *AnonRep: Towards tracking-resistant anonymous reputation.* In 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI), Mar. 2016.

[OSDI'14] Ennan Zhai, Ruichuan Chen, David Isaac Wolinsky, and Bryan Ford. *Heading off correlated failures through Independence-as-a-Service.* In 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI), Oct. 2014.