Research Statement

Behnaz Arzani
Behnaz.Arzani@microsoft.com

I am a systems and networking researcher focused on designing practical systems that improve the reliability, performance, and management of modern networks. The over-arching goal of my research is to design self-managing networks. Network management continues to be a difficult task given the number of devices, the added complexity from logical components that operate on top of the physical topology, the diversity of protocols, and the difficulty in obtaining a consistent snapshot of the network in its entirety. Our networks are evolving at a staggering pace and network management solutions need to be able to evolve with them. Creating a self-managing network requires accounting for these complexities and devising appropriate solutions with quantifiable risk to avoid unexpected behaviors. The advent of new programmable hardware, the deployment of programmable network interface cards, and advances in machine learning make it the right time re-think our approach to network management.

I use formal models to understand network protocols and build practical systems that improve network management. My research lies at the boundary of systems and theory. It stands out from most others in the field because it is rooted in (1) **systems perspectives** (drawing from my years of experience at Microsoft) to ensure that my solutions are deployable and grounded in practice while (2) **using formal models** to ensure that they have predictable behavior. The systems I design: (1) are high performance, scalable, and provably correct systems that help move networks to a desired state and (2) help identify new insights that allow networks to utilize low-overhead and scalable monitoring systems to make management more efficient. These are crucial properties of practical systems and are important components of a self-managing network. My work on 007 [2], which I deployed on one of Microsoft's data centers, is an example of a system that has these properties. 007 uses probability theory to provide performance guarantees and to quantify the conditions that are needed for 007 to achieve high accuracy.

In my research I aim to have real-world impact (e.g., [1, 2, 3, 4, 5]). My research experience has given me a unique perspective to be able to build systems that do exactly that. Through my research I have developed an understanding of many of the practical challenges of building *deployable* and *accurate* monitoring and diagnosis systems. For example, through my work in NetPoirot [1] I found that often, to troubleshoot, operators need information from components on a flow's path that are not part of their network. These components, in many cases, are managed by other organizations that are unwilling to share information or accept responsibility for the problem. A self-managed network will have to either include (1) monitoring components that can identify alternative monitoring that can be used to compensate for such lack of information (NetPoirot) or (2) if such a compromise is not possible, to allow for "brokering" of information across organizations in a way that is acceptable to all parties. I have found many similar examples throughout my work.

My style of research is highly collaborative and interdisciplinary. My solutions to networking problems often has leveraged knowledge from other areas including probability (007 [2] and [7]), optimization (DRVR [10] and FixRoute [13]), and machine learning (NetPoirot [1] and PrivateEye [4]). Not all systems can be modeled through a single theoretical technique. Similarly, solutions to many of the problems I've encountered has involved using techniques from other areas. In my research I have collaborated with researchers at Microsoft Research, Harvard, UCSD, Brown, and many students at the University of Pennsylvania. My familiarity with subjects such as theory (probability, optimization, game theory), machine learning, and verification has allowed me to quickly identify when each can be used to solve the problems I encounter, to pick up the necessary depth in order to use those techniques in practice, and to find collaborators that can help improve the systems that I build.

My time as an intern in Azure as well as a post-doctoral researcher at MSR has given me a unique perspective on the real problems that operators tackle which I believe can be immensely helpful in working on the types of problems I aim to solve (§4). My connections and close collaborations with industry enable me to have real impact with the research I conduct and to bring my research to the forefront of industry. I will also be able to use these ties as potential additional sources of funding.

In the following, I will present a summary of my past work (§1, §2, §3) and future research plans (§4).

1 Diagnosing failures in data centers

The goal of data center (DC) operators is to provide high availability to their customers. DC networks operate at massive scales. Failures in these networks are unavoidable, and when they occur, they can have a detrimental effect on availability. Finding the cause of problems in DC networks is akin to finding the proverbial needle in a haystack. There is a lot of existing work on diagnosis [6, 8], but most focus on finding the devices that have failed, whereas in practice operators also need to know which applications are impacted by the failure, and by how much.

I take a two-step approach to diagnosing *specific* applications and quantifying impact of individual failures:

Extracting more information from low-overhead/scalable monitoring systems [1]. The first step in diagnosis should be to find what part of the system is responsible for a problem. An important concern is that network connections often involve devices belonging to different organizations. These organizations may not be willing to share information, and when problems occur, their best interest may lie in blaming others. We need a solution that can be deployed from a single point in this ecosystem. Such a solution would allow operators to reason about the end-to-end environment without requiring support from the organizations involved.

I started with the following observation: TCP provides reliable delivery of data across networks that may drop packets. It is well known that TCP reacts to network failures. The key insight I had is that (perhaps surprisingly) this reaction can also be used, not only to identify network problems, but also to seperate network problems from those caused by the client or server. I built and deployed NetPoirot [1] to demonstrate this. NetPoirot was the first system to show that easily accessible signals from the TCP stack (e.g., number of duplicate ACKs, time spent in zero window probing) can facilitate diagnosis of a variety of issues and finding which entity is to be blamed for the problem. NetPoirot identifies whether a problem is caused by the network, server, or client. It eliminates the need for additional monitoring systems to be deployed throughout the network or on the server side. *NetPoirot was deployed in Microsoft's data centers and was used by engineers to find the cause of VM reboots for two years.*

NetPoirot's classification algorithm modifies a well-known machine learning algorithm, namely, random forests. I recognized that TCP statistics are well-suited to separate network failures from other failures and that information gain is higher when creating a binary classifier, as opposed to one that separates four classes of labels (network, server, client, non-failure). For these reasons, the input features to the algorithm would have higher information gain if the problem is solved as binary (instead of multi-class) classification and if it is first used to identify network failures from the remaining classes of labels. I took a divide and conquer approach. The first layer of the model classifies between network failures and other data points (server, client, non-failure). Next, the network failure data is removed from the training set, and the second-layer model is trained to classify between server-side failures and the remaining data points (client-side failures and non-failures). Finally, the last layer is trained to distinguish client-side failures from non-failures.

Finding the cause of every TCP packet drop [2]. Once the network is identified as the cause of a problem, we still need to uncover the devices in the network that are responsible. Besides, when multiple problematic devices are present in the network, we would like to be able to connect each of these devices to the applications it impacts. As one operator from Azure networking put it: "Fixes need to be prioritized based on customer impact. However, currently, we do not have a way to correlate customer impact with bad links."

Given the scale at which DCs operate, finding problematic devices is extremely difficult. What complicates the problem is that DCs use shallow buffers. Shallow buffers occasionally result in one-off packet drops that the diagnosis algorithm should not detect (since the drops are not caused by device failures). These packet drops act as "noise" to the diagnosis solutions and result in inaccuracies in solutions that have been proposed in the past [9]. 007 is a system that allows for uncovering the cause of problems for *specific applications* in a way that is insensitive to noisy packet drops, and that does not require major infrastructure changes. 007 achieves this goal by finding the cause of each TCP flow's packet drops. 007 is provably correct and to demonstrate its correctness *we deployed it in one of Microsoft's data centers for two months*. Its success also inspired an RDMA diagnosis system that incorporates the algorithms proposed in 007 to help the diagnosis of RDMA applications. The impact of this project was recognized by Victor Bahl, a distinguished scientist at Microsoft, in a blogpost published in 2018 [15].

To prove the correctness of 007, I recognized that whether a TCP flow will experience a packet drop on a given link can be modeled with a Bernoulli distribution. Using Kullback-Leibler divergence, I was able to show what the Signal to Noise Ratio (SNR) would have to be in order for 007 to be able to distinguish between *good* links (that drop packets due to congestion) and *bad* links (that drop packets due to systemic failures). I showed that the SNRs observed in today's data centers are large enough that 007 can find problematic links with almost perfect accuracy.

Other work on diagnosis [3, 6]. NetPoirot and 007 help diagnose failures during the execution of individual applications and assist operators in identifying each failure's impact. However, not all problems are due to packet drops. Routing loops, misrouted packets, and erroneous mappings of virtual IPs do occasionally occur. Diagnosis of such problems can be facilitated using systems like Everflow [14] that allow operators to run packet captures on multiple switches simultaneously. Everflow creates a copy of each packet at the switch and sends it to a collector. The result is a distributed packet capture that contains the trace of individual packets as they traverse the network.

Analysis of these distributed packet captures can be challenging. This is in part because, in some cases, the packets captured are dropped on their way to the collector. Also, there is no time synchronization across switches, and it is difficult to determine the path of a packet without additional topology and routing information. In our work, dShark [3], my collaborators and I created a framework for the analysis of distributed packet traces that addresses these problems for the operators. dShark allows operators to submit queries such as *Is there a routing loop in my network?* and analyzes the packet traces to return a yes/no response.

My collaborators and I also created VNet Pingmesh [6] which continuously monitors the health of routing, firewalls, and the logical path within individual customers' virtual networks. *It has allowed Microsoft's networking teams to be proactive about identifying problems and resolving them before they manifest in customer impact.*

2 Multi-path protocols

While quick diagnosis of failures helps reduce their impact on performance, it does not eliminate that impact. It is important to ensure the customer impact of failures is minimized while management is taking place. My work addresses improving the reliability of our communication networks in the face of failures to achieve this goal. Various proposals have been made in the past to improve the reliability of the network through network coding [16], fast re-routing [17], or multi-path flows [18]. Network coding comes with excessive overheads. Re-routing is often difficult as it requires moving large volumes of traffic away from parts of the network to others. Such significant changes to traffic flow can impact the operator's traffic engineering objectives as well as result in churn that can also be disruptive to user performance. In my work, I focus on multi-path protocols that enable users to achieve reliable performance over unreliable networks. I have looked at these protocols in the following contexts:

Multi-path routing in wireless mesh networks [10]. Wireless mesh networks allow for rapid and flexible deployments of communication facilities and have gotten renewed attention with increased investments of companies in edge computing and IoT. This potential notwithstanding the often erratic behavior of multi-hop wireless transmissions has a negative impact on the performance of applications that use them. In this work, I investigated the feasibility and benefits of a routing protocol that is aimed at making wireless mesh networks more predictable while preserving their efficiency and flexibility. My protocol's basic premise is the classical idea that a multi-path solution can offer resiliency to unexpected link variations. I demonstrate how routing decisions that account for link variability can be computed in a distributed fashion by formulating the routing problem as a convex optimization that can then be separated into multiple mini-optimization problems that each wireless hop can solve locally, independent of the other devices in the network (with minimal parameter passing with its local neighbors). The solution resulted in over 2X improvement compared to a single path approach that picked the most reliable path.

Analysis of the MPTCP protocol [11, 12]. While multi-path routing protocols are useful, they have their limitations. The most significant of such limitations is their impact on protocols such as TCP that are sensitive to packets arriving out of order. Alternatively, protocols such as MPTCP allow users to benefit from the existence of multiple concurrent paths while avoiding performance penalties by modifying the transport layer. MPTCP is currently deployed in Apple iPhones. But is the MPTCP design optimal? I found the answer to be no. My research showed that the choice of which sub-path is used first can have unintuitive consequences on the achieved throughput. I modeled the protocol's behavior analytically and found that the problem is due to the non-linear coupling between paths through the protocol's congestion control algorithm. These findings are interesting both from an operational and a design standpoint. They also show that we have not fully understood the implications of many of the design choices in the MPTCP protocol which may cause more significant unexpected problems down the line.

3 Managing VM security

Protecting VMs from being compromised is another one of operators' management goals. A compromised VM can damage other VMs and the DC infrastructure. When talking to operators, I found that less than 5% of VMs are protected by the compromise detection systems that providers offer. This is because customers do not want the operator to monitor their VMs. Operators have no recourse but to rely on customers to follow best practices (e.g., setting secure passwords and avoiding opening un-necessary ports on the VM).

The customer's need for privacy is understandable. However, oprators should still be able to protect all VMs. I am currently working on a privacy-preserving compromise detection system. The system monitors each VM's connection pattern (which is monitored for other purposes such as diagnosis) and detects when it is compromised. It uses the fact that operators have various internal services deployed in their DCs. These services are monitored extensively. The system learns how compromised VM's connection patterns change by using detections from the operator's internal VMs. It then applies what it has learned to customer VMs that are not monitored as aggressively.

I have devised a novel approach that encodes a VM's connection pattern and how it changes so that an ML algorithm can accurately predict whether a VM has been compromised. The system trades off precision for scalability and acts as a first line of defense in the DC. Other, more expensive algorithms such as Snort are then used to investigate further the VMs the system finds suspicious. My preliminary findings show that such a system is highly accurate in detecting various types of compromises that frequently occur in practice [4]. Specifically, the system achieves 95.77% true positive rate for a modest 1% false positive rate while imposing minimal CPU and memory overhead.

4 Future work

Problem Statement. My research agenda is to build on my prior work and answer the question: *How can our distributed systems and networks manage themselves?* The number of users of services such as Netflix, Amazon, and Twitter have enormously increased over the last decade. This increase is enabled by large-scale networks such as those managed by Google, Comcast, and AT&T. The increased usage of these services has also resulted in the increasing growth of our networks. Unlike operators in the 80s, who had to manage fewer than 100 switches, today's operators manage thousands, if not millions, of devices and links. With the increased popularity of Internet of Things, Video on Demand, and social media, the scale of our networks will continue to grow.

Azure, AWS, Google, Verizon, and other operators today employ hundreds of engineers whose job is to maintain and engineer various aspects of their operation. Networking research is decades old, and our operators continue to have to spend hours of work finding the root cause of failures, configuring the network, and fighting DDOS and other security breaches. I aim to answer the question: Can these operations be automated?

The advent of new machine learning techniques as well as new programmable data planes makes the time ripe to start to move our networks to a place where they can manage themselves. In theory, a self-managed network would identify where and when to execute management tasks. It would also identify and resolve problems when they occur. This process includes reconfiguring the network when necessary, predicting performance problems and mitigating them, capacity planning, scheduling configuration and network changes, and finally finding link failures, security breaches, and even congestion events. In practice, new technologies, such as programmable forwarding planes (e.g., P4), allow building measurements at a scale and precision that is not possible with legacy approaches (e.g., the perpetual traceroute).

Research areas. Some of the challenges that I will need to overcome include: 1) identifying management tasks in the day-to-day operations of various networks; 2) identifying what has prevented the automation of those tasks; 3) finding datasets that if available would have allowed for easier automation and/or better accuracy when automating management tasks; 4) building systems that automatically identify what datasets to collect and where to collect them in a way that provides the right balance between the granularity of the data collected and the cost of data collection; 5) building systems that help understand self-managing systems when something goes wrong; 6) building systems that analyze this data and determine where and what actions need to be taken to either resolve problems or move the network to a desired state; 7) devising risk assessment methods that would identify the risk of executing each of these commands in order to ensure minimal disruption to ongoing traffic. There are many more such challenges.

Prior work has laid the groundwork in the form of predictive models for application performance and providing high-level abstractions to encode operator objectives. My goal is to leverage some of these findings and design the missing pieces to build a working system that can be used in production networks. For example, moving to a self-managing network may require incremental changes to the current networked systems to facilitate the transition to a fully self-managed network. Identifying what these changes should be, how they are to be executed, and their impact is something that has yet to be addressed by current research. Finding what these changes should be requires a deep understanding of how current production networks operate and what their constraints are. My deep knowledge of these networks uniquely qualifies me to tackle this problem. Many of the challenges outlined above continue to remain unsolved. The solution to these challenges will also have to draw from research in optimization theory, probability, verification, and machine learning. For example, statistical and probabilistic techniques will

have to be used in order to identify the risk of each management operation the system will execute. Verification techniques will have to be used to ensure that the system remains in the correct state throughout these operations.

More immediate challenges. More immediate problems in this space that I will work to solve are the following:

AutoML in networking. ML is useful in automating many network management tasks. One of the challenges operators face in using ML techniques in large DCs is the lack of domain knowledge in machine learning. Progress is often bottlenecked by the limited human resources that know both domains well. Because of this, one of the future works that I plan to undertake is a domain-specific system that allows "machine learning for network experts". Specifically, my goal is to design a machine learning framework that allows domain experts in networking to use machine learning without being experts in machine learning themselves. For example, an operator that wants to solve a problem similar to NetPoirot [1] can use this system to auto-generate the necessary features and algorithms.

AutoML is currently being studied in ML communities, with Google offering a cloud AutoML framework that can be used in natural language processing, image processing, and language translation [19]. However, the current approaches to AutoML attempt to solve generic problems and do not consider experts' domain knowledge that could help improve both feature engineering and accuracy. My insight is that networking problems have a common underlying structure (e.g., topology, sources and destinations, etc.) and therefore allow for a degree of domain specificity that helps AutoML be effective.

Operators can provide insights within a domain specific language that describes their SLAs, the topology, and the data set. The framework can then assess whether an ML solution is appropriate. Once a determination is made that ML is the right approach, the framework can then automate feature engineering by exploring a range of possibilities and find the optimal ML model for the problem at hand. The operator SLAs can guide this process by limiting the search space, e.g. the system can stop the search as soon as it finds a model that satisfies the true positive, false positive, and performance SLAs the operator specified.

Diagnosis. Today's DC workloads involve various applications with complex dependencies on systems that are managed by different teams and services. One of the challenges in moving closer to a DC that can manage itself is inability to efficiently localize the cause of problems when they occur. Incidents are often bounced around across different teams and different groups until they reach the right individuals who can effectively mitigate and resolve the incident. During the summer of 2018, my intern and I did a measurement study of Azure DCs to understand this problem. Our results indicate that Azure engineers have to diagnose up to $24,000$ incidents per day (median $6,000$). We observed that 40% of incidents were routed to the wrong group *at least* once (one was misrouted 35 times). Each time an incident is misrouted, it is placed in the incident queue of a different group. We found that 20% of incidents experience at least an hour of queue time. This is the time during which an incident is not being investigated, and impact is ongoing. Misrouting causes an accumulation of such queue times which can exacerbate perceived customer impact. Furthermore, other incidents that occur afterwards are placed behind this incident in the queue which results in delayed diagnosis for those incidents as well. We found that misrouted tickets result in up to 300 hours of wasted engineering effort per day (median 50 hours).

These misroutings are in part due to the lack of a comprehensive monitoring system that can look at the system as a whole and take into account the different components that are involved in the execution of a particular task. Our interviews with 10 different groups in Azure and 50 engineers indicate that the problem is also in part due to a lack of comprehensive understanding of the system by individual groups, new hires, as well as the difficulty of correlating the monitoring data that *is* available to arrive at a definitive conclusion. I am currently collaborating with researchers at Harvard and the University of Pennsylvania to build a "meta-diagnosis system" that can integrate all the different monitoring information from all networking components and use it to pinpoint the right service to which to direct an incident. An idea we are currently exploring is to use per-system change point detection combined with other statistical techniques to identify which systems are experiencing problems at the time a ticket is created. When migrating to an autonomous network, this system can be used as the basis for automating diagnosis tasks.

Collobartions. My style of research is highly collaborative and I look forward to collaborating with colleagues in areas of theory (networking theory, game theory, control theory, algorithms, and probability theory), networking, distributed systems, verification, machine learning, programming languages, and security.

References

[1] **Behnaz Arzani**, Selim Ciraci, Boon Thau Loo, Geoff Outhred. *Taking the blame game out of data centers operations with NetPoirot*. Proceedings of the 2016 ACM SIGCOMM Conference

[2] **Behnaz Arzani**, Selim Ciraci, Luiz Chamon, Yibo Zhu, Hongqiang (Harry) Liu, Jitu Padhye, Boon Thau Loo, Geoff Outhred. *007 Democratically finding the cause of packet drops*. 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2018

[3] Da Yu, Yibo Zhu, **Behnaz Arzani**, Rodrigo Fonseca, Tianrong Zhang, Karl Deng, Lihua Yuan. *dShark: A General, Easy to Program and Scalable Framework for Analyzing In-network Packet Traces*. 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2019

[4] **Behnaz Arzani**, Selim Ciraci, Stefan Saroiu, Alec Wolman, Jay Stokes, Geoff Outhred, Lechao Diwu. *PrivateEye: Scalable Privacy-Preserving Compromise Detection In The Cloud.* In submission.

[5] Arjun Roy, Deepak Bansal, David Brumley, Harish Kumar Chandrappa, Parag Sharma, Rishabh Tewari, **Behnaz Arzani**, Alex C. Snoeren *Cloud Datacenter SDN Monitoring: Experiences and Challenges* 2018 Internet Measurement Conference (IMC)

[6] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, Alex Snoeren. *Passive realtime datacenter fault detection and localization.* 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2017

[7] Hui Zhang, **Behnaz Arzani**, Franjo Ivancic, Junghwan Rhee, Nipun Arora, Guofei Jiang. *Offline queries in software defined networks* US Patent

[8] Yin Zhang, Zihi Ge, Albert Greenberg, Mathew Roughan. *Network anomography*. Internet Measurement Conference, 2015

[9] Praveen Tammana, Rachit Agarwal, Myunjin Lee. *Simplifying datacenter network debugging with PathDump*. 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2016

[10] **Behnaz Arzani**, Roch Guerin, Alejandro Ribeiro. *A distributed routing protocol for predictable rates in wireless mesh networks.* ICNP, 2012

[11] **Behnaz Arzani**, Alex Gurney, Sitian Cheng, Roch Guerin, Boon Thau Loo. *Deconstructing MPTCP performance.* ICNP, 2014

[12] **Behnaz Arzani**, Alex Gurney, Sitian Cheng, Roch Guerin, Boon Thau Loo. *Impact of path characteristics and scheduling policies on MPTCP performance.* In Advanced Information Networking and Applications Workshops (WAINA), 2014

[13] **Behnaz Arzani**, Alex Gurney, Bo Li, Roch Guerin, Boon Thau Loo. Fixroute: A unified logic and numerical tool for provably safe internet traffic engineering. arXive (2015)

[14] Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, Ratul Mahajan, Dave Maltz, Lihua Yuan, Min Zhang, Ben Zhao, Haitao Zheng. *Packet-level telemetry in large datacenter networks.* Proceedings of the 2015 ACM SIGCOMM Conference

[15] Victor Bahl. *Microsoft Shines at NSDI 2018.* `https://www.microsoft.com/en-us/research/blog/microsoft-shines-nsdi-18/`

[16] Chou, Philip A., Yunnan Wu, and Kamal Jain. *Practical network coding.* Proceedings of the annual Allerton conference on communication control and computing. Vol. 41. No. 1. The University; 1998, 2003.

[17] Zhong, Z., Nelakuditi, Z., Yu, Y., Lee, S., Wang, J., and Chuah, C. N. *Failure inferencing based fast rerouting for handling transient link and node failures.* INFOCOM 2005.

[18] Raiciu, Costin, Christoph Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. *How hard can it be? designing and implementing a deployable multipath TCP.* In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, pp. 29-29. USENIX Association, 2012.

[19] Google's Cloud Auto-ML. `https://cloud.google.com/automl/`

[20] ARPANET. `https://en.wikipedia.org/wiki/ARPANET`