Before EDA, integrated circuits were designed by hand. – Daniel Nenni [73]

1 Introduction

The Internet has been an amazing success for nearly 50 years, partly because of a key design decision to make the network application agnostic. In other words, IP treats all applications whether remote telesurgery or Facebook as a group of addresses that send a bag of bits to each other. This allowed the Internet to start with apps like email, and later accommodate unforeseen applications like video. Today we take for granted services like Uber for hailing a cab but these run on cheap computers connected by an IP data center network. There many more exciting networked applications around the corner whether networks of self-driving cars, remote telesurgery, precision agriculture, and more we dont know of.

Many of these new applications are critical. They are so essential for society that they demand unprecedented levels of performance and reliability from the network. For example, a storage area network may require the network to deliver a disk block in < 100 usec, and cloud services aspire to 5 9s reliability. The requirements for networks of self-driving cars, for augmented reality, and for remote tele-surgery are still being formulated.

While such stringent requirements are typically associated with large enterprises such as Google and Facebook with deep pockets and armies of experts, what is surprising is how such requirements also flow to lower tier networks where expertise is scarce and cost is an issue. Consider for example, a surgeon doing remote telsurgery from Tamil Nadu to a village 100 miles away. Such a network has tremendous requirements on latency and jitter, and customizing such a network could help save costs or even lives.

Networks today are managed using a patchwork of complex, manual, and low-level mechanisms. Network architects produce designs largely by hand, guided by intuition and experience. Network operators reconfigure network devices constantly to match evolving needs, leading to configurations that are inefficient, difficult to understand, and ever more brittle.

As a result, networks today suffer from *high cost* (Google [46] reports a factor of two underutilization for expensive transoceanic links), *high latency* (Amazon [93] asserts that a 100ms increase in latency leads to a 1% drop in revenue), *lack of agility* (a startup [27] reports that some Fortune 500 companies take over a week to roll out minor configuration changes because they do not fully understand their potential impact), *fragility* (surveys [115] have shown that one in three networks have dozens of incidents per month that take over an hour to resolve), *insecurity* (denial-of-service attacks now account for 16% of network attacks [12]), and *high barriers to entry* (rural networks cannot afford to employ hundreds of highly-trained operators [38]).

Companies such as Microsoft that traditionally distributed software on disks have transitioned to a cloud-hosted service model through products such as Office365. Gartner estimates the size of the current public and private cloud markets at $139B and $10B respectively. The poor performance and fragility of networks has direct negative impacts on cloud computing. For example, Google aims to limit downtime to "a few minutes per month" [34]—anything more would hurt their bottom line as they make only a few cents on each search query.

In other fields, increasing scale and requirements led to intellectual advances in *design automation*. For example, challenges related to scaling hardware designs in the 1970s led to the creation of electronic design automation (EDA) as an academic discipline [87] as well an entire industry of EDA tools (e.g., high-level synthesis, functional verification). Similarly, increasingly stringent requirements on large-scale software led to the theory and practice of high-level languages and companion technologies (compilers, integrated development environments). McKeown [63] suggests there is a large opportunity for automation, noting that in both software and chip worlds a $10B tool industry supports a $100B software and chip industry, but that networks largely lack such tools.

The earliest EDA tools were produced academically. – EDA History [112].

Our Vision. Our central vision is to empower any enterprise whether small or large, whether a rural network in Mendocino County or a large data center in Oregon – to build and operate a customized network that provides the properties their application requires, while minimizing the number of manual effort required by network architects and operators.

To do so, we are inspired by the transformative effects of software tools and EDA, a field of *Network Design Automation* (NDA). We plan to develop novel science and practical tools that will revolutionize how networks are designed, managed, and evolved, thereby increasing the pace of innovation, reducing the cost of building secure, reliable, and performant services, and even lowering the barrier to entry for operators of rural networks and data centers, and possibly in the futiure ISPs and cellular networks.

Why Now? We believe the time is right for NDA, as evidenced by recent progress in academia and industry. In 2013, Google [46] and Microsoft [42] deployed new traffic engineering systems that compute efficient routes through the network automatically and without human intervention. They found that programming the network *as a whole* increased utilization and reduced cost by millions of dollars. We have built verification [50, 49, 114], testing [115, 13], and debugging [28] tools, along with many others, and we have also designed higher-level languages for expressing network policies [68, 3]. The number of papers on top-down design and verification at SIGCOMM, the flagship networking conference, grew from 1/30 before 2011 to 8/40 in 2016. Veriflow [50] and Forward Networks [110] received millions in venture capital funds in 2015 to verify network properties. In 2015, Cisco founded a design automation unit called Candid under Sundar Iyer [44], and Amazon hired Byron Cook to create a cloud verification group [21]. While there has been a flurry of recent activity in design tools for clouds, initial work has also begun on network design for rural networks barath.

What's Left? While these initial activities are promising, NDA will not happen without a significant, coherent, sustained effort that *democratizes* network design and operation tools so that every network, from the smallest rural network to the largest data center can systematically design and operate their networks to work at the highest levels of reliability and performance without a large team of operators that only the largest enterprises can afford.

In this context, the point solutions listed above are a great start, there are several gaps: *1. Fundamental questions unanswered:* For example, network reachability approaches [49, 50] break down in the presence of stateful devices such as firewalls. More generally, there are no fundamental models analogous to Turing machines, or general verification abstractions akin to Hoare logic [41]. *2. Most design is manual.* The design of a network, from its structure to its components to its configuration, is largely done by hand. *3. Isolated abstractions:* Existing abstractions [3, 9, 49, 15, 35] are promising but are neither complete nor interconnected. *4. Focus on data centers:* Most extant work ignores diverse networks such as rural networks. *5. Few connections between interdisciplinary areas:* NDA encompasses a range of disciplines spanning data mining to programming languages to ICDT and yet there have been few attempts to bring researchers in different disciplines to work together coherently. *6. Limited outreach:* Existing tools (e.g., Batfish [26], Propane [9] are still not directly applicable except to expert operators in larger networks. The goal of this NSF Large is to fill these gaps with a significant, coherent, and sustained effort in three areas:

1. Intellectual Merit: NDA Science. *We will develop a science of network design automation, inspired by programming methodologies but united by an emphasis on domain-specific languages and algorithmic search techniques.* On the surface, networks with their collections of routers, interface cards, and links do not resemble software. But one can view the entire network as a "program" that takes packets at the incoming edge of the network and outputs packets at the outgoing edge, after possibly rewriting some of the bits in the packet. For instance, a forwarding table on an individual router can be viewed as a "switch" statement on the packet's destination addresses, while a link between routers can be viewed as an assignment statement that modifies the packet's location. These analogies suggest a new research agenda that asks: what languages and tools are needed to program and reason precisely about networks? However, solutions cannot merely reuse existing ideas from programming languages and related fields. Instead, effective solutions for network programming and verification will require new abstractions that tackle the unique challenges of the domain: networks are fundamentally asynchronous and distributed, the state space of possible packets is huge, and the correctness and performance requirements are subtle and complex. At the same time, networks have domain-specific structure that can be exploited by tools. For example, an off-the-shelf theorem prover for verifying network reachability took more than *1 day* [58] but by cleverly exploiting equivalences between configurations, Yang and Lam were able to complete the same task in under *1 second* [114]. Table 1 summarizes some of the language design and scientifc questions we will address and Figure **??** relates these

Task	Abstraction	Sample Scientific Question
Topology Design	Topology Description Language	Can SAT Solvers aid rural topology design?
Network Verification	Policy Specification	Can Network Specific clustering be used to find anomalies without a specification?
Network Testing	Coverage Specifications	What does it mean to cover the control plne?
Network Management	Network Policy Language	How to apply stepwise refinement to networks?
Network Monitoring	Performance Query Language	How to apply performance assertions to networks?
Network Debugging	Application SLAs	Can machine learning be used to link network performance measures to application measures?
Failure Analysis	Cause-effect Graphs	Can NLP find causal relations from text?

Figure 1: Sample tasks, abstractions, and scientific questions for our proposed research on NDA.

questions to practical problems in various types of networks today.

2. Broader Impact: Network Design Tools. *We will build a suite of tools for networks to begin to catalyze an industry for Network Design Automation akin to the EDA industry.* We imagine network architects at Fortune 500 companies and rural ISPs using NDA languages to develop designs in hours or days, lowering cost and increasing robustness. We expect network operators to be able to evolve a network in an agile fashion—e.g., reacting to traffic spikes, responding to security threats, and rolling out new services. While we do not propose to deliver industrial-strength tools, we plan to collaborate with established companies (VMWare, Cisco, Google, and Microsoft) and startups (Candid and Forward Networks) via internships and joint projects (see letters attached to this proposal). We seek to develop the foundation of the NDA industry just as academia did for the EDA industry in the 1970s [87].

3. Broader Impact: NDA Education. *We will take first steps toward lowering the educational barriers for people around the world to operate networks.* As with the EDA and software revolutions, achieving wide impact will require educational materials to train the next generation to use NDA methodologies and technologies. To enfranchise operators, we will focus on producing open-source course material (inspired by Khan Academy) based on what we call *network scripting* that will teach how to manage the vast majority of simple networks found around the world, together with open-source simulators for easy network testing. We seek to complement existing efforts such as Cisco Academy [16] that have already had major impact in

disadvantaged societies.

Why an NSF Large? While individual researchers and companies can build impressive point solutions (e.g., [46]), it takes the efforts of several researchers working in concert to map out the landscape of NDA and define its interconnections (see Figure 2). Networking spans a range of disciplines including language design, verification, testing, debugging, and data mining. Our team brings together networking experts (Varghese, Govindan, both ACM Fellows for networking), with PL expert Millstein (who helped found the nascent field of network verification) and domain experts in ICDT (Raghavan), debugging (Netravali), and systems testing (Tamir) to facilitate interdisciplinary exploration at a much larger scale than is possible in a smaller NSF grant.

All six researchers, however, are in Los Angeles at UCLA and USC. Further, many of them have worked together successfully in research teams (e.g., Govindan-Varghese, Govindan-Millstein, Raghavan-Varghese) over many years and many projects. We believe colocation in Los Angeles facilitates genuine collaboration and frequent in-person meetings with students and PIs, in contrast to the more sprawling and comparitively isolated efforts of other multi-institution proposals.

Why not Industry? Companies, which generally operate on short-term schedules dictated by the lifecycles of their products, are not incentived to organize a large-scale effort such as NDA, let alone tackle difficult scientific questions. Instead, we are inspired by Berkeley's early development of open-source products (e.g., the VLSI Tarball [73]) and leadership (e.g., by Sangiovanni-Vincentelli and Newton) for EDA. We note that three of us (Govindan, Millstein, Varghese) have been working to help pioneer this area for the last five years; we feel it is time to take the next step to consolidate this field further

Related Efforts and Scope. First, the recent architectural shift known as software-defined networking (SDN) provides a set of basic mechanisms that allow network routers to be centrally programmed. However, new mechanisms alone are not sufficient: we contend that NDA is necessary to fulfill the promise of SDN, just as higher-level languages and tools were necessary to fulfill the promise of VLSI. Further, NDA does not depend on SDN. Our focus is on building tools for existing networks in the next 5 years while laying a scientific foundation for the next 10 years of networking research. Second, while NDA encompasses aspects of security that are directly impacted by network design and operations, such as ensuring isolation and detecting attacks, other aspects of security, such as encryption and authentication, are out of scope for this project. Existing efforts at UCLA [102] are already investigating these problems.

Third, we will focus in this grant on verification and automation of the design and operation of existing networks as opposed to more clean slate approaches being pursued by our colleagues at Cornell (Foster) and Princeton (Rexford, Walker). We fully intend to work with these colleagues wherever feasible, and will look for middle ground between their top-down language guided approaches. For instance, Millstein and Varghese have an NSF Medium working with Walker at Princeton on making the Propane language for router configurations more deployable.

Fourth, some of us (i.e., Varghese, Tamir) have expertise and interest in router hardware design and emerging languages for reconfingurable routers like P4 [10] and believe that Network Design Automation in its most general form also applies to building tools like router compilers. However, since tools for router design are cleanly separable from tools for network design, we will focus in this grant on network tools *except* for router hardware debugging primitives to assist network debugging.

2 Challenges

Many difficulties lay before us – Alberto Sangiovanni-Vincentelli, The Tides of EDA [87]

Networks today face daunting challenges because of *ad-hoc design methodologies*, *over-reliance on the human-in-the-loop*, and the *ineffectiveness of existing techniques*.

2.1 Ad-Hoc Design Methodologies

Networks are rife with ad hoc design methodologies, from topology design to policy.

C1: Topology Design is Heuristic: The physical topology of a network is essential for achieving good performance. If the topology lacks the capacity to meet demands, it will have to drop packets; if it does not provide a diverse set of forwarding paths, it will be unable to recover from failures. Yet there is immense pressure to keep topologies simple (to streamline operations) and small (to minimize cost). Most topology design today is done by hand, using simple heuristics that make it difficult to ensure that the topology will have the properties their designers envisage. A much better alternative would be to design the network topology and configurations together, using tools to *automatically synthesize* configurations that provide required performance characteristics while optimizing for simplicity and cost. While it is hardly surprising that topology design of a rural network [39] is fairly ad hoc, what is surprising is that systematic topology design tools even for large clouds l(e.g., Google's Condor [89]) are only gradually

C2: Network Policy is Implicit: In many networks, policy intent is absent or at best provided by incomplete and ambiguous English specifications [9]. However, the behavior of the network is defined by the low-level configurations installed on individual routers. Entries in configuration files added by humans (e.g., to override the firewall) are rarely removed due to fears that doing so may have unintended consequences. Different operators often have (erroneous) views of the way parts of the network should behave.

One may imagine that a central challenge for networks is verifying *asynchronous* distributed protocols such as BGP. However over the last 20 years many protocol bugs have either been fixed or worked around; so in this grant we concentrate on *configuration* verification and not *protocol* verification. Tools to synthesize or verify configurations would help reduce the fragility of networks today.

C3: Current Automation Tools are Limited to Clouds: The vast majority of recent work in network verification and automation targets data centers [49, 82, 116, 29]. However, the landscape of networks is diverse, and solutions for one class of networks do not directly apply to other classes. Raghavan at Berkeley has made similar observations about rural networks [39], but the problems are different. Automation is arguably even more important for rural networks are they impact more of the world.

2.2 Over-Reliance on Humans in the Loop

Networks are fragile due to a reliance on humans-in-the-loop in many phases of design and operation.

C4: Manual Configuration Causes Major Cloud Outages: Networks use distributed protocols to compute paths for forwarding traffic. However, these protocols are manually configured in complex ways to override the default least-cost routes. For example, customers often prefer certain routes for economic reasons (often by configuring a so-called "Local Preference" knob); for security reasons (often implemented by packet filters); and for spreading load (using a protocol called MPLS [66] that allows more flexible routing than traditional IP). To facilitate this kind of fine-grained control, routers provide various ways of partitioning sets of addresses or routes into equivalence classes for routing (so-called COMMUNITY tags) or isolation (so-called VLAN tags). The book by Stewart [97] describes the ingenious methods network operators use to tune routes and the BGP knobs they manipulate.

Over the years this need has produced a rich set of capabilities for network operators that they are accustomed to using and will not give up easily. Unfortunately, configuration through individual router settings is inherently difficult to understand and get right, as the space of possible routing and security policies is large. Configurations often have implicit invariants that must be respected across routers but are easy to get wrong—e.g., routers must agree on the meaning of a COMMUNITY tags, and access-control rules must be set consistently. Further, even when these invariants are maintained they often go wrong after failures or planned configuration updates. With the need to optimize service delivery, the velocity of change has increased [34] which increases the chance for errors. Finally, while many vendors are building ad-hoc tools for detection and remediation, bugs in these tools can exacerbate even small failures. For example, on April 11 2016, a simple error caused by removing an address block in the Google Compute Cluster [33] triggered bugs in automated fault detectors that resulted in a 53-minute global outage.

C5: Brain Drain Impacts Rural Networks: Almost four billion people, or two thirds of the worlds population, lack access to the Internet. Of those, over 90% are in the developing world, many of whom either live in rural locations far from existing Internet infrastructure or are too poor to afford connectivity. This sit-

uation is unfortunate because a 10% increase in broadband penetration is correlated with a 1.35% increase in GDP for developing countries [79, 84]. Rural networks operate under severe resource constraints and so must rely on aggressive traffic engineering especially when the weather changes.

Consider AirJaldi, a large rural wireless network operating in the Indian Himalayas whose upstream capacity totals 15Mbps. During a monsoon, thunderstorms disconnected several subscribers, indicating failure of the local power grid. Hours later, one of Airjaldi's two upstream providers went down. Unfortunately, this failure occured at peak usage time, forcing Airjaldi operators to cope with the loss of half its upstream bandwidth. Operators began routing traffic through aggressive content filtering proxies, and all but a few well-known ports (e.g., 80, 443, 456) were blocked. The expertise needed to provide this manual intervention is rare and expensive. Rural-to-urban migration and "brain drain" is a problem in rural areas globally. Offering wages competitive with urban areas is financially untenable. Allowing rural ISPs to automate the key aspects of network configuration could make their networks more sustainable.

Note that rural networks do not just exist in India and Kenya but also in rural areas of the United States. Consider the Motech ISP [39] in Sonoma County, California that one of our PIs (Raghavan) has considerable expertise with. It serves rural users across a rugged region of about 100 square miles. The challenge in such rural networks [39] is dynamic rerouting sometimes caused by weather and sometimes by simple reconfiguration: when the operator climbs a tree [39], they may find a better direction to aim their antenna. Unfortunately, these changes often require altering routes manually at each router which is done today by trial and error.

C6: Providing Performance Guarantees is a Black Art: Cloud tenants do not just want correctness but also performance quantified by service level agreements (SLAs): for example "Response times should be less than 1 msec" and "aggregate bandwidth to Azure Cloud Storage Service should be at least 1G." On the other hand cloud operators care about infrastructure health as quantified by internal performance metrics such as Inter-rack latency, router CPU utilization and router stability. While one normally thinks of such aggressive SLAs as being necessary for large clouds, they are also needed for rural networks. Consider, for example, the jitter requirements for remote telesurgery.

The key to providing such SLAs would be principled performance monitoring tools. Sadly they do not exist today. Instead, each application (e.g., Gmail, Bing, etc.) typically has an alert system. While alerts monitor the infrastructure, other ad hoc tools check for SLA violations to customers. Many of these tools use diverse data sources such as end-to-end probe results, router counters, and `syslog`s. All systems we know of only use some data sources and cannot correlate information across all of them. For example, there is no higher level way for a manager to ask: "which routers have come up and down more than 5 times in the last hour." This is complicated by the need to avoid false positives: an alert system that alarms too frequently will be ignored by operators.

C7: Data Center Failure Causes are Poorly Understood: While Data Center Analytics and Verification are the "first responders," better reliability will ultimately come from a deeper understanding of cloud failures. Clouds are complex behemoths and the root causes of failures can be surprising. The recent Google Compute Engine failure [33] provides a compelling example. The failure was not caused by a component failure but by the remediation system. The root cause was documented extensively [33] in *unstructured text*. One of the PIs (Govindan) spent months manual pouring over the post-mortem reports of Google's high profile failures in the last two years [34]. Such manual effort cannot scale to the large amount of post-mortem reports that exist in private and public clouds world-wide.

2.3 Traditional Design Techniques Do Not Work or Scale Well:

Simply repurposing tools pioneered for software and hardware design to NDA is untenable.

C8: Networks Lack Rigorous Foundations: Huge improvements in software reliability have come from *1.* bottom-up tools such as static checkers and fuzz testers that allow developers to gain confidence in the correctness of their code; and *2.* top-down high-level languages, both general-purpose (e.g., Java) and domain-specific (e.g., SQL), which make it easier to specify intent and obviate large classes of errors by construction. Both approaches rely on theoretical foundations developed over decades, including *logics* such as Hoare logic, techniques for *modular* program construction and analysis (e.g., isolating function

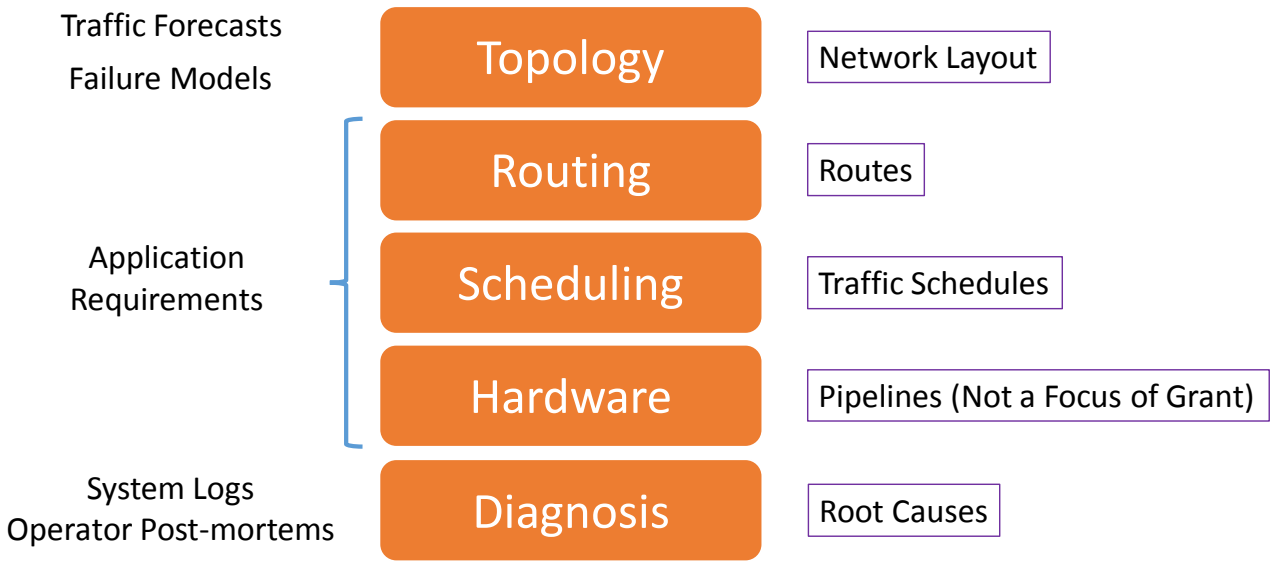

Figure 2: The layers of abstraction in NDA.

bodies from callers, type systems, and module systems with information hiding), and *abstractions* that make automated reasoning tractable and simplify specifications.

Unfortunately these theoretical foundations do not directly apply to networks—network "programs" are distributed, and most of the above approaches only apply to sequential programs, especially at scale, even if one assumes protocol convergence. Second, we do not understand enough about networks yet to have useful approaches to modular reasoning or abstraction. For example, the way that a node routes a particular packet can depend not only on the intended destination, but also on the source address, the nature of the network's relationships with its peers, and the current load on the network. A further complication is the diversity of vendor platforms (e.g., Cisco, Juniper, etc.).

We will concentrate less on rigorous foundations and more on building usable tools in our proposals. This is partly because foundations is already a major focus of our colleagues (e.g., Foster, Walker and Rexford) at the Cornell-Princeton network programming initiative]citecornellcenter). However, in pursuit of our tools, we will not hesitate to tackle fundamental questions when needed, leveraging the work of our colleagues wherever possible. For example, as we decribe later our initial work to find semantic differences between two routers in the same role which builds on the Minesweeper tool [8] from Princeton by essentially moving from SAT (Minesweeper, 1 counterexample) to AllSAT (our tool, all counterexamples, succinctly encoded).

3 Abstractions for NDA

Synthesis implies a bridge between two layers of abstraction – A. Sangiovanni-Vincentelli [87]

Abstractions were key to the success of EDA [60]. By steadily raising the level of abstraction for designers, EDA allowed complex chip designs to be developed using high-level behavioral models that are mapped to blocks, to logic gates, and finally to lists of wires. Although early efforts in EDA focused on bottom-up verification, later innovation allowed synthesis between abstraction levels.

Abstractions for NDA Our proposal seeks to develop an analogous but preliminary set of *interconnected* abstractions for NDA. Networks already have abstractions for packet delivery (TCP and IP [54]) and for packet forwarding (SDN-inspired flow rules [15, 35, 94]) but do not have abstractions for designing and configuring the network itself (or components thereof). To that end, we have identified an initial set of layers of a network that will frame our proposed research (Figure 2): *network diagnosis*, *router hardware*, (not a focus of this grant), *packet schedules*, *routes*, and *topology*.

We aim to define abstractions at each of these layers of a network, which can then be leveraged both for synthesis (from a higher to a lower layer) and verification (from a lower to a higher layer). Relative to EDA, these tasks are complicated by the fact that (a) networks exhibit fine-timescale dynamics and evolve over long timescales and (b) network design will need to incorporate human input in the design process. As such, synthesis and verification for NDA is a *continuous* activity over the entire network lifecycle. We overview the goals of each proposed abstraction in the following paragraphs; Section 4 describes our proposed work in detail and sets it in the context of related work.

NDA Research: Mapping Between Abstractions At the highest level, we assume that a network is meant to run networked applications with application level service level requirements often expressed by SLAs, ranging from loose (e.g., email, FTP, precision agriculture) to medium (video conferencing, VoIP, connected cards), to stringent (visualization, telesurgery) in terms of banswidth, latency, and jitter.

The first level abstraction (see topmost layer in Figure 2) then is a *network topology* to run the applications. In our vision, each abstraction layer corresponds to a tool. For example, we envision a topology tool wherein an architect specifies traffic growth forecasts, resiliency requirements and components like routers and links, and the tool produces the physical network. Many cloud providers including Google [89] and Microsoft already have such tools but these are proprietary and would not apply to a rural network like Air Jaldi.

Imagine an open source tool that can design a physical network whether for a rural network that uses VoIP (1000 users in 12 villages, using wireless links and FPGA based routers) or a larger data center network for Google (10, 000 users for Google Applications using high end Cisco routers). The Condor system at Google [89] recently proposed a delarative Topology Description Language (TDL) to specify abstract topology connectivity and uses a constraint solver to generate topologies **S4** but it only automates some limited aspects of topology design.

Once the physical network is laid out, the baton passes from the network architect to the network operator. Applications must specify their traffic demands and performance objectives. Merlin allows an operator to specify a policy to set up routes and packet schedules. Note that unlike conventional networks, these routes and schedules can be tuned to application objectives. Thus the first job is to layout application traffic on specific network paths subject to policy restrictions and to optimize enterprise objectives. For example, email and Voice Over IP could be "laid out" over the physical network where VoIP is sent on multiple paths while email is sent on only one path. Thus we envisage a routing tool that maps from application objectives and a physical network design to routes for each application traffic.

The next abstraction is *packet schedules*. While packet schedules are today largely decided by each router in decoupled fashion, work such as Fastpass [81] argues that even packet schedules could be computed by a tool and passed to each router. For example, the schedule could decide that in some routers, Voice Over IP gets priority scheduling over email. when the two traffic classes intersect at that router.

Of course, when churn occurs such as DoS or monsoon in a rural network or a fiber cut in a data center, the route are reconfigured automatically based on operator specification (ecommerce takes priority over web surfing during monsoon for rural networks, customer traffic gets priority over analytics in a data center. Another important kind of churn is the specification of new application requirements by the operator. For example, in a rural network if the operator adds a new tele-surgery application this can change the layout of the other packets and the schedules at each router.

While our example uses rural networks, the same idea applies to say Pfizer doing visualization of genome data and Augmented Reality for drug design, or Google adding connected cars, or John Deere adding precision agriculture.

These abstractions (topology, routes, schedule) compile down to a back end that can be a source routed

network with FPGA based routers (for rural networks), a conventional network with Cisco routers with BGP and scheduling parameters set by the higher abstraction levels, or even a P4 router based on say the Broadcom chip and routes set up by an OpenFlow controller. This is the next layer below that we call hardware.

In this layer, a router designer combines hardware *tiles* such as multiplexers and memories to make hardware blocks that comprise a pipeline *stage*. Designing router pipelines is a field that is greatly in need of tools and the ability for reseachers to build tools (as opposed to proprietary tools used at Cisco and Google) has had an inflection point since the advent of the P4 language [20].

For focus, this grant will **not focus on router design tools**. We will however, investigate debiugging primitives in router hardware) the bottom most layer especially as it relates to network debugging, the bottom most layer.

Various synthesis and verification tasks operate on the router and scheduling abstractions. One is the synthesis of router-level topology of an entire network from constraints on traffic and connectivity [89]. A related synthesis task requires the network architect to adapt to traffic changes via *traffic engineering* that maps measured *traffic matrices* over some duration to route configuration commands at routers. Abstractions also enable synthesis and verification between high-level network *policies* and their low-level configurations.

A network operator also needs to monitor and debug the network. Ideally, the operator should issue *network performance queries* at an autonomous system level (e.g., compute traffic matrix) and have these mapped to performance commands at each router (e.g., read byte counter). On a longer time scale, the network architect needs to extract, from logs of network faults captured (often annotated by a network operator), a higher-level *failure model*, which can be used to target efforts to improve reliability. We lump all these concerns into the bottommost level

Notice that our tools work on designing specialized networks within a single Autonomous Systems or AS where the architect and operator can control the entire network. Examples include rural networks, data center networks ad enterprise networks. Of course, individual networks are interconnected to form the Internet . We do not focus on such Network Design Automation "in the large" because we feel that is precisely the specialized context (rural networks, data center networks) that can allow novel language based synthesis and verification tools to work.

Our Approach: While there exist point solutions at some of these layers for certain tasks, these abstractions are neither *complete* or *interconnected*. We will produce a complete and interconnected set of abstractions from topology down to schedules. Interconnecting abstractions allows us to build an end-to-end system from routers to rural networks, as we propose in our evaluation.

4 New Directions for Mapping between NDA Abstractions

> *The technical community expanded to reach areas of expertise in non-linear and combinatorial optimization, control, artificial intelligence, and logic.* – The Tides of EDA [87]

The preceding section presented abstractions to address network design challenges. The utility of these abstractions depends upon methodologies to map between abstraction levels, as in EDA. Some mapping tasks are relatively straightforward, while others are risky, potentially transformative, and require new science. While the directions are necessarily diverse to address different stakeholders, they are united by common themes: *based on algorithmic search*, *interdisciplinary* (using multiple skill sets enabled by an NSF Large), and *domain-specific* (addressing the complexities but exploiting the structure of networks).

4.1 From Constraints to Wiring: Topology Synthesis

Tool: (Challenge C4, Foster, Govindan). A network architect must map a specification of a network in terms of constraints (that reflect both design objectives and resource limits) into a physical topology of routers and wires (**S4**).

Science: Past work [89] has focused on *data centers* and *connectivity* constraints and used mapping heuristics; we will generalize to *general networks* and *generic* constraints and map using Counter-Example Guided Inductive Synthesis (CEGIS).

Approach: Network topologies must balance competing objectives including high bandwidth, fault tolerance and low cost. Today, designing a large network topology can take a human expert up to six weeks. Although modeling and simulation tools are often used as aids, the design process is still largely done by hand. While topology design can be framed as combinatorial optimization [2, 100, 57], prior work has only been used to *understand* the structure of existing networks and not to *design* new ones.

We plan to develop a domain-specific programming language in which planners may specify constraints on the design. Our point of departure is recent work at Google on a domain-specific language (DSL) for data-center topologies called TDL [89]). TDL can be used to generate a specific class of topologies (Fat-Trees [1]) whose links and switches are relatively homogeneous, subject to connectivity constraints between different tiers of the Fat-Tree. We will design a more general DSL for wide-area networks (WAN) and enterprise networks that will support more general constraints essential for these networks such as latency, traffic-carrying capacity, costs, and limited optical touchdown points available from optical vendors.

We plan to use Counter-Example Guided Inductive Synthesis (CEGIS) [96] to search through the space of possible topologies, learning from counter-examples to quickly rule out topologies that do not provide required functionality. To make synthesis tractable, we will need to build tools that can rapidly answer reachability queries as well as quantitative questions involving bandwidth and fault tolerance. Finally, we will explore domain-specific heuristics such as generating topologies from a fixed class and exploiting its structure and symmetries, inspired by heuristically-optimal topologies [57]. We will also explore synergies with constraint-based approaches used in EDA for high-level synthesis (e.g., [19]).

4.2 From Configurations to Policies: Extending Network Verification

Tool: (Challenge C5, C7, Govindan, Millstein, Tamir, Varghese). A network architect must either map configurations at routers to a network policy (*verification*) or map a network policy to configurations at routers (*synthesis*) (**S5**).

Science: Past work in network verification is unscalable or makes assumptions that limits its applicability; we will develop general approaches to these problems such as *stepwise refinement* and *network specific data mining* to find bugs without specifications. We will also develop new algorithms to compactly encode the *semantic difference* between two routers.

Approach: Together with colleagues, we have helped pioneer the science of configuration verification (Header Space Analysis [49]); built tools for exhaustive data plane [49, 48, 58] and control plane [26, 25] verification; and applied the idea of automated testing to data planes [115]. In this thrust, we seek to improve the scalability of these tools and design missing pieces that prevent their application to a broader set of verification and synthesis problems in networking.

We propose new directions for verifying the data plane (the flow of data packets) and the control plane (the flow of routing packets).

1. Finding Bugs without Specifications: It has now become standard practice to use formal methods and "prove" that the existing router configurations work correctly across all packets for both data plane (e.g., Veriflow [50] and HSA [49]) and control plane (e.g., ERA [25], ARC [31], and Minesweeper [6]). However, this approach requires some form of specification to check the router or network model against. In our experience, such specifications are often missing, wrong, or at best partially correct (basic beliefs like "Customer Machines should reach DNS" are often correct [58] but are only *partial* specifications). This is possibly because networks are managed by multiple operators, and there is also much legacy code in configurations that current operators do not understand but are scared to change. This begs the question: can we find bugs without a specification?

The field of unsupervised learning and data mining suggests that one could *cluster* the set of routers in an operational network into partitions that we might term *roles*, confirm that roles are meaningful by looking for substantial agreement between routers in a role, and finally spotting bugs as *anomalies* where some

routers differ in some way from the majority of routers in the role. We tried straightforward approaches to clustering using off-the-shelf techniques like *k-means* and *Hamming distance* metrics, but failed to produce meaningful clusters despite tinkering with parameters.

This has led us to invent new *network specific clustering* algorithms. Note that the work in network verification such as Veriflow [50] and HSA [49] was able to scale because they exploited the structure of networks to scale verification beyond the limits and limitations of conventional verification tools like SAT Solvers and Symbolic Evaluation applied blindly to the networking domain. In the same way, we believe that clustering and data mining must be rethought to exploit the structure of the network domain.

Consider, for example, the following network specific clustering that we find works quite well in practice. We start with the observation that most names assigned to routers in organizations are meaningful. In UCLA, for example, a router could be named br1.boelter1.floor1, where the "br" stands for border router, and "boelter" is a building name. We have seen similar behavior in other universities and corporations. Thus, we divide the name into lexical tokens separated by delimiters (such as punctutation while ignoring the numbers). We then evaluate how meaningful each combination of tokens is as a role. We use statistical techiques analagous to that used by Engler in finding bugs in Linux [23] to evaluate the strength of each role as a hypothesis. Once we find a good role partition (measured by substantial agreement in some precise sense between routers in a role), we then look for bugs as deviant configurations [23] within each role. Note, however, that Engler's work has no concept of a role.

We have investigated two router specific clustering techniques. The first, described briefly above, is based on combining lexical tokens in router names. We try all combinations and thus do not make assumptions on where the role name occures whether as say boulter.br.ucla or br.boulter.ucla. This technique is based on the reasonable assumption that human operators assign router names based on some implied semantics. However, we have found that the technique is not perfect possibly because of drift and mergers: for example, in UCLA two sets of names "br.anderson" (the Anderson School) and "br.csb1" referred to the same role. This name convention is possibly caused by the fact that one set of routers is in the Anderson School in North Campus, and the CSB1 routers are in South Campus. Nevertheless, the routers in both "roles" are very similar.

We have also investigated a role definition based on distance from the border: most often the routers that interface to the ISPs perform one role, then come a set of core routers, then a set of department routers and so on. However, even this is not perfect because of the use of L2 switching technology that increases or decreases distance without necessarily differentiating roles. We are also investiagting combinations of conventional clustering (say based on Hamning distance of names) and say hop count and have found many new directions.

Our preliminary work at UCLA and some other networks, has identified new bugs that earlier technologies did not. Our grant plans to flesh out these preliminary ideas, evaluate them scientifically on a much larger set of networks, and make the tools available publicly.

2. Synactic and Semantic Differences: How can we tell whether a role is meaningful? For example, in UCLA it turns out that the pure router type (e.g. "br" or "bd") is not sufficiently meaningful but the combination of type and building (e.g., br.boulter) is. We measure goodness of a role by measuring the syntactic or semantic similarity of routers within the same role. A simple syntactic check for example is to check for similar definitions, e.g., of DNS servers. An anomaly then occurs when "most" (measured by a threshold) routers in a role have the same DNS server but a few do not.

A more profound difference is to measure the semantic difference in terms of forwarding behavior of two routers across corresponding interfaces (e.g., upwards towards the ISP). We leverage a tool called Minesweeper [6] but this tool uses an underlying SAT solver to provide a single packet counterexample when there is a difference. We have invented a new polynomial time algorithm to loop across Minesweeper to aggregate the counterexamples into sets of prefixes that are forwarded differently. We feel such semantic differences can not only find more meaningful anomalies but also help us understand the underlying semantics of each role.

Roughly the semantic difference algorithm is as follows. Assume we are looking for the set of destination prefixes that are different. We start by branching on prefixes that start with a 0 and those that start

with a 1. For each set, we ask Minesweeper whether the two routers are equivalent. If Minesweeper finds they are, we do a further check in Z3 see whether for all addresses that start with this prefix, there is some router environment and data packet which is forwarded differently by the two routers. If neither condition is true, we branch further. Stated recursively:

```
FindAllDiffs (R1, R2, P)  //* find all differences between router R1 and R2 for Destinatio
  If (For all addresses X that satisfies Prefix P, R1 and R2 have similar behavior) // Mine
          return; //no differences
  Else If (For all addresses X that satisfies Prefix P, R1 and R2 have differing behavior)
           Print (P); //everything in P is a difference
           return;
  FindAllDiffs (R1, R2, P0); //recurse on P0
  FindAllDiffs (R1, R2, P1); // recurse on P1
```

We have found that while the set of prefixes produced is fairly compact, one can compress further by using a bottom up dynamic programming algorithm to also exploit the fact that many sets of prefixes can be compactly expressed as difference of cubes notation [49, 58]. The algorithm walks the subtree of prefixes bottom up evaluating the costs of the two notations (positive form and difference form) to find the optimal "compression" in a manner evocative of Huffman coding,

3. Stateful Verification via Stepwise Refinement: Stepwise refinement dates back to Dijkstra and Wirth [113]. With Ryzhyk [85], we recently introduced refinement-based network programming as an intermediate stance between network verification (the user provides all configurations, the tool verifies) and synthesis (the user provides a specification, the tool synthesizes). Instead, the user interacts with the system at each level of refinement to design and prove correctness of that level. Step-wise refinement is especially promising because it may provide a way to address one of the major limitations of current tools: they cannot model stateful devices. In practice networks maintain state for several reasons (security, congestion control). However, if a router can no longer be modeled as a function on a single packet but rather as a function on sequences of packets, the state space becomes very large [105]. We hypothesize that step-wise refinement can escape this verification conundrum analogous to way a theorem prover can outperform a model checker if the user specifies high-level invariants. We will work with Ryzhyk at VMWare to verify Network Function Virtualization (NFV) devices such as Google's Maglev load balancer [22].

Moving to synthesis we plan to work on:

4. Synthesis for Rural Networks: We have already proposed extending the work in Propane on control plane synthesis from data center networks to ISPs in an NSF Medium grant (Butane [**?**]) with David Walker at Princeton. While Butane *extends* Propane to cover more use cases, in this grant we *simplify* Propane to handle simpler rural networks such as WISPs [39] (with PI Barath Raghavan). WISP operators in Africa, India, and rural US will not learn a new network policy language but may be willing to try a phone-based UI that verifies connectivity and suggests configuration changes in response to common events.[Phone-based UI? Seriously? These operators don't even have access to a desktop or laptop machine? –JNF]

We do not propose any work in configuration synthesis in this grant because we have already proposed generalizing the work in Propane [9] on control plane synthesis in an NSF Medium grant led by David Walker at Princeton that is joint with Millstein and Varghese.

4.3 From Configurations to Tests: Creating Automated Network Tests

Tool: (Challenge C5, C7, Millstein, Tamir, Varghese). Many vendors including Microsoft and Google have created VM frameworks to test routers and their configurations because verification only verifies a *model* at each router. . Unfortunately, these frameworks require the tester to manually supply tests. Instead, we suggest sutomated testing as in systems like Klee'[11].

Science: Inspired by our earlier work ATPG [115] for testing data planes and software notions of coverage (e.g., KLEE [11], we will automatically generate test *routing announcements* and emulate on UCLA and USC topologies.

Approach: Unlike KLEE, ATPG gets away with a much simpler algorithm based on set cover compared to say KLEE or DART because (in bug-free networks) there are no cycles (although network verification can and should check for such cycles first before applying such methods): hence the state space is finite (though large). KLEE and DART have a more complex set of heuristics to search the infinite space of executions caused by programs that can have loops.

However, BGP and OSPF provide an interesting intermediate ground; unlike loop-free data packet forwarding, BGP announcements can be batted back and forth before they converge and simple set cover is unlikely to work. On the other hand, most commonly used forms of BGP are not as complex as arbitrary code. Thus we seek intermediate methods that are tailored to networking without the complexity and generality of say KLEE or DART (which must handle arbitrary code). The intellectual component of this grant largely arises from navigating this tradeoff space.

While this approach is inspired by software testing, we are also inspired by hardware testing which has a prolific literature. PI Tamir has worked in the field of computer architecture and will pursue the connections between hardware notions of testability and coverage, and networks. [[Yuval add some stuff here]] showing how

4.4 From Network Queries to Router Commands: Performance Monitoring

Task: (Challenge C9, Foster, Weatherspoon). After designing a network, a *network operator* must monitor the network for performance problems such as traffic bursts and unexpected failures (**S7**).

Science: Past work [80] allows operators to specify higher level performance invariants for multiprocessor systems; we will generalize to allow performance assertions for networks.

Approach: We will develop a high-level query language and distributed run-time that exploints features provided on next-generation network hardware (Section **??**) but also for existing IP networks. It will map network-wide performance queries down to commands that can be executed on individual routes and switches. This challenge is similar to mapping network programs discussed previously, but adds several new dimensions, beacuse queries are typically quantitative and can often be answered only approximately. For example, we might implement a query that monitors a single flow by installing forwarding rules that tag and count matching packets, or one that computes the global traffic matrix query by aggregating counters polled from all devices at the edge [72].

While several existing systems [72, 36, 37] develop some basic machinery for answering queries, they are point solutions in that they only offer limited expressiveness and require SDN switches. We believe we can generalize Perl's notion of performance assertions that she used to monitor a multiprocessor system[80] to existing IP networks and to newer P4 switches, and use them to concisely express and efficiently evaluate "difficult" queries such as identifying bottlenecks or estimating available bandwidth.

We plan to compile performance assertions to existing router commands to poll router counters and logs, using a RegEx engine to correlate information across multiple devices. Before a router crashes or reboots, it will attempt to write a message to a `syslog` file. Hence, a query to find all routers that have port flapping events in the last hour, we can use a performance assertion about the rate of flap events, which can be computed from the `syslog` files for all routers. In addition to targeting existing routers, we also plan to take full advantage of emerging platforms such as P4, a goal we share with [72].

4.5 From Application Failures to Network Causes: Machine Learning for Debugging

Task: (Challenge C9, Govindan, Netravali, Varghese) Operators need techniques to tie application level problems to low level network issues (**S9**). Current debugging systems either opreate entirely at the network or entirely at the application layer while operators need to understand the connections.

Science: Machine learning is beginning to be used in networking to learn unstructured information. We propose using a combination of NLP and Machine Learning to .

Approach: Network failures and performance degradations are inevitable. Indeed, even with automated network design, demands on the network are continually in flux. Operator requirements change, the services that run atop networks (along with the corresponding network workloads they generate) vary, and

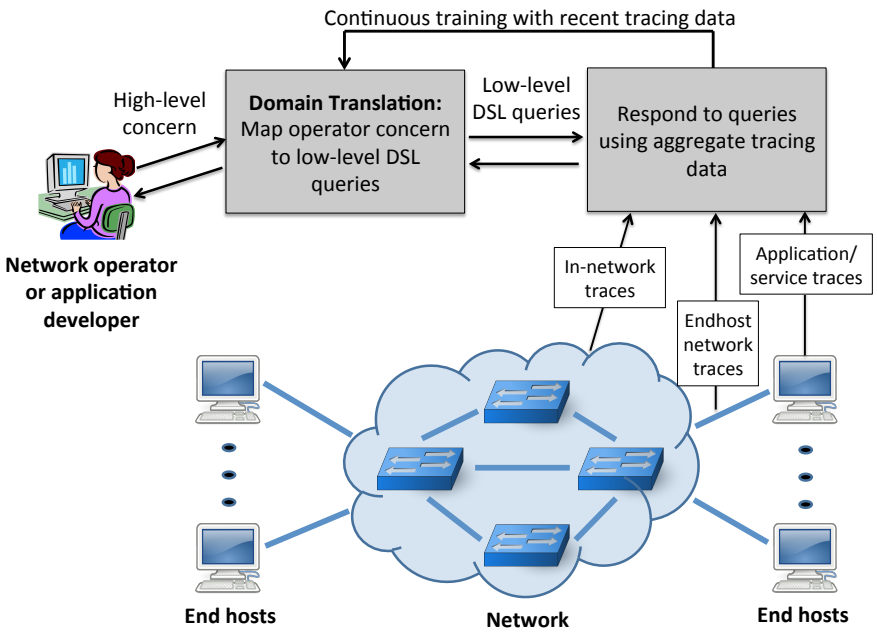

Figure 3: Overview of our proposal to automatically bridge the gap between operator/developer concerns and low-level tracing systems.

network devices can fail. As a result, the ability to monitor and debug networks and the services that they support is crucial for enabling functional and performant services.

Unsurprisingly, a multitude of systems have been developed to aid developers and operators in debugging each part of the ecosystem. Network debugging tools can be classified into two primary approaches: in-network techniques collect information directly in network switches (e.g., per-packet header manipulations, routes, etc.) [28, 37, 70, 71, 91], while endhost monitoring tracks data within the endhost's network stack or via end-to-end probes [69, 98, 99]. There also exist tracing tools for distributed systems and applications [59, 74, 90, 95]. These tools track source code line executions, data flows, and resource utilization patterns throughout the execution of a service or application. Each aforementioned debugger prunes the traced data to help developers and operators answer queries, typically specified using domain specific query languages (DSLs) [62, 70, 71]. For example, network operators can issue queries related to performance (e.g., queuing delays) or routing (e.g., loop detection), while application developers may ask about control flows and storage-layer data accesses in program execution.

Unfortunately, despite the existence of these well-optimized debugging systems, understanding bugs and performance degradations in networked systems remains challenging. The primary issue is that it is difficult to bridge the low-level information and DSL queries that each tool supports with the high-level concerns of operators and developers. Concurrent operations can span different applications, network stack layers, physical entities, network types, and administrative domains. Developers and operators are often well versed in only an individual part of the ecosystem, making it difficult to generate the appropriate cross-domain queries that make maximal use of the available information. For example, an application developer will likely ask questions of the form "I have observed an SLA violation in my service–what caused this?", rather than "Was my issue caused by an incast issue on switch X at time T?"

In this proposal, we seek to bridge the gap between low-level debugging systems and high-level operator concerns. In particular, our aim is to enable developers to specify high-level concerns and automatically receive actionable responses; the process of converting high-level concerns to DSL queries encompassing application, endhost, and in-network debuggers will be automated and hidden from developers and operators. Mapping high-level concerns to DSLs can be done in one of two ways: 1) using tracing data which

was collected during the execution in question, or 2) assuming that all possible tracing data is available. The former can use already collected data to provide a fast response, but may restrict the set of possible DSL queries, potentially affecting the response quality. In contrast, the latter approach isolates DSL query generation from the available data, ensuring that the optimal queries are considered; however, collecting the necessary data may be challenging as execution environments are difficult to reproduce. We plan to explore this tradeoff in more detail using concrete environments and real tracing data.

With either approach for generating DSL queries, a key challenge is to figure out how to partition a potentially ambiguous high-level concern into the appropriate DSLs. Prior systems have attempted to partition program logic across different network domains [5, 106]. However, these systems are designed to receive well-formed inputs (often expressed with a separate DSL), and partition logic across domains whose relationships are well understood (i.e., the control plane and the data plane). In contrast, our proposal may be presented with loosely-formed, high level concerns. In addition, we must map those concerns to cooperative DSL queries across environments that are largely dynamic and have been mostly treated in isolation to date.

To overcome these challenges, we propose two solutions. First, we plan to employ Natural Language Processing techniques to bridge query abstraction levels by mapping high-level operator concerns to DSL queries. Second, we will use tracing data collected at each point in the ecosystem (i.e., applications, end-hosts, and networks) to generate a model which relates low-level operations, tracing data, and DSL queries across domains. Given the dynamic nature of the end-to-end environment in terms of workloads, configurations, and available data, we propose that each component is continually trained to update its model with the latest tracing data. For example, training can use techniques like LSTM to prioritize recently observed tracing data without forgetting past (potentially sporadic) events [43]. Figure 3 depicts our proposed workflow.

Several properties of modern networks and distributed systems complicate this vision. First, how can we support broad sets of loosely-formed queries? Prior systems have attempted to automatically parse trouble tickets using NLP techniques based on fixed keywords and input structures [83]. In order to support arbitrary queries from each part of the ecosystem, we plan to develop models which relate keywords in each domain based on causal actions observed in the collected traces. Second, how can we collect sufficient training data to cover the wide range of possible developer queries. In addition to training with actual snapshot data from prior execution and bug reports, we plan to generate synthetic training data to highlight potential issues (e.g., resource contention, race conditions, etc.). Third, prior systems have shown that collecting comprehensive tracing data at any one part of a networked system poses scaling challenges, both for storage and processing [70, 90]. Our proposal requires the aggregation of tracing data from each part of the ecosystem. Because the importance of collected data is most effectively considered in aggregate, we must develop efficient pruning strategies that can scalably collect the important tracing information without sacrificing insights. Fourth, how can we handle information gaps in the ecosystem due to privacy concerns over tracing data or lack of administrative cooperation? Our models must be robust to varying amounts of contextual data and misreported tracing data.

Prior work has proposed using machine learning for classification to associate network-specific queries with tracing data from single machines [6]. Our proposal differs from these approaches in two key ways. First, we propose using machine learning in part as a language translation tool, connecting the abstract, high-level developer language to the low-level, structured query languages that existing tracing systems support. Debugging systems have continually tried to raise the level of abstractions for queries and user interfaces. However, in order to go the next step and connect every part of the ecosystem, we believe that tackling the unstructured concerns of operators is the next challenge. Second, our proposed model that relates DSL queries (and the corresponding tracing data) is not restricted to a single machine that is controlled by one administrative entity. Instead, we plan to merge data and DSL queries across different combinations of applications, machines, and networks, in order to accurately respond to high-level developer concerns with actionable suggestions.

4.6 From Failure Logs to Models: NLP for Extracting Cause-effect Patterns

Task: (Challenge C9, Govindan, Netravali, Varghese) Operators need techniques to analyze failures to prioritize network engineering and design decisions (**S9**).

Science: Relatively little research in Natural Language Processing has addressed the extraction of cause-effect relations in text. We will develop natural language learning methods that capture the linguistic patterns that denote causal relations that appear in network failure reports. Our research will help us suggest what *minimal* (because we believe it is unlikely that operators will adopt completely structured reporting) descriptions can be added to logs and failure reports to enable richer analyses.

Approach: At four or five nines availability, cloud providers can afford only a few minutes of outages per month [34], and even using NDA tools, it is likely that failures will happen. Today, cloud operators maintain logs that describe changes to the network and document analyses of failures. These logs are examined manually today [34]. We propose to exploit these logs to automatically (a) categorize root causes of failure, and (b) develop a succinct abstraction (a failure *cause-effect* graph) to represent each failure.

In preliminary work, we performed a detailed manual analysis of unstructured text in post-mortem reports of over 100 high-impact failure events within Googles network [34]. Here, we instead propose to use Natural Language Processing to develop failure abstractions. It will take new science in Information Extraction, Discourse Analysis and Domain Adaptation to develop natural language learning methods that accurately and robustly capture the *linguistic patterns* for *causal relations* that appear in network failure reports. Linguistic patterns for causal relations, for example, can appear in compound nouns ([EFFECT]$_{\text{NP}}$ *incident*), as subject-object pairs ([EFFECT]$_{\text{SUBJ}}$ *was triggered by* [CAUSE]$_{\text{OBJ}}$), or across sentence boundaries, with or without intervening discourse cues like "as a result" ([The guest OS rollout]$_{\text{CAUSE}}$ was applied to Cluster B at 10:10am. [Connectivity was lost]$_{\text{EFFECT}}$ at 10:11am.).

We believe that root-cause categorization is amenable to standard *text categorization* techniques [92]: given a bag-of-words representation of a post-mortem report failure DISCUSSION, supervised classification methods (e.g. SVMs [47], maximum entropy classification [77], deep averaging networks (DANs) [45]) can be applied to classify the text according to the type of the root cause it describes. The challenge here will be to develop sufficient training data for accuracy. We propose to use active learning methods [18, 61, 101] that select the most useful examples to add to the training set in each iteration. Finally, to discover new root-cause categories, (an instance of "concept drift" [111, 51]) we propose human-in-the-loop clustering with constraints [107, 108, 17, 64]: cluster the new and existing post-mortem reports and ask a networking expert to inspect the result, looking for partitions that contain network failure events that fall outside the current set of known root causes.

Beyond root-cause categorization, a useful abstraction that succinctly summarizes a single failure is a *cause-effect failure graph*, which relates failure causes (software or hardware components) to observed effects. This abstraction will permit network operators to analyze, across many failures, which components fail most often and under what conditions, enabling them to prioritize resources for reliability engineering and network re-design. From an NLP point of view, the extraction of cause-effect graphs is most easily viewed as a task in *information extraction* [14, 30, 88] in which limited kinds of domain-specific semantic content (i.e., network failure causes, locations, effects) are extracted from unstructured text (i.e., post-mortem report DISCUSSIONs), and transformed into structured formats (i.e., a cause-effect graph) suitable for analysis. Extracting cause-effect relations remains an open problem in NLP. We propose to build on research [53, 75, 76, 67, 40] in causal relation identification to develop domain-specific techniques that can identify both within-clause causal relations, inter-clausal and inter-sentential expressions of causality, as well as the coreferential relations among the entities, states and events involved.

Some of these approaches to NLP for network failure analysis were developed with Claire Cardie at Cornell, an NLP Expert. We also plan to consult Kai-Wei Chan, an NLP expert at UCLA.

5 Evaluation

We plan to evaluate the tasks presented in the preceding section as follows:

- **S1:** We will evaluate our topology synthesis tools on synthetic benchmarks, and will seek to deploy variants of these tools through collaborations with industry partners.

- **S2:** We will evaluate the extension of stepwise refinement to evaluate the Google Maglev load balancer, something beyond the capabilities of network verification tools today. We will work with Bjorner and Lopes at Microsoft to evaluate the new ideas for scalable verification based on transitive closure on the Microsoft data set we used in previous work [82].
- **S7:** We will instrument Gumaste's existing SDN testbed at IIT Mumbai with performance queries, and measure the overhead and completeness of the approach for several common monitoring tasks.
- **S8:** We will develop mining software that we will deploy at Google to evaluate whether we emulate all of PI Govindan's earlier manual analysis [34], while scaling to larger inputs.

Additionally, in the fourth year, we will deploy a end-to-end demonstration using a small cluster of new reconfigurable router designs (using an FPGA) at IIT Mumbai, add new router measurement capabilities using our compiler, design the network topology using our synthesis tool, synthesize configurations using stepwise refinement, and finally monitor the network using performance queries. While this only exercises a subset of our tasks, the others (e.g., failure analysis) can only be evaluated outside the laboratory.

6 Educational Activities

NDA is a radical departure from the status quo that will fundamentally change the kinds of skills needed to design and operate the networks of tomorrow. To address this need, we propose a set of educational initiatives designed to train students for the jobs of the future. A distinctive feature of these initiatives is that they target operators in the developing world and data center networks in the developing world.

6.1 Reaching Operators around the World

Our EEVBook will be designed to train NDA tool developers and, to some extent, network architects. We also plan to target network operators around the world with our second educational initiative.

We have had discussions with Michael Ginguld, an entrepreneur who believes that Internet access is a basic right [32] and has pioneered AirJaldi, an ISP in India, to make this vision real. Setting up networks around the world requires two things: infrastructure (which is receiving attention from Facebook [24]) and *operators who can cover the entire gamut of network operations* from climbing trees to network architecture and configuration. In NDA, we focus on the latter aspect: can we lower the barriers to operator training and network operations in rural India or Africa?

We are inspired by vignettes from the Cisco Academy [16] showing how transformative network operator training can be. These include Stephen Oriecki in Kenya, a graduate of the academy who persuades youth to choose the training center over "drugs in the street," and Farha Fathima in Sri Lanka who is one of the 28% of women graduates of the academy in the Asia Pacific region (where it is estimated that nearly 400,000 networking professionals will be required to meet demand). While the academy's achievements are impressive, its model has flaws: certification is expensive ($10,000), covers only Cisco gear, and teaches its graduates low-level primitives in Cisco routers.

Our allies who operate rural networks (Grameen, AirJaldi, Motech) around the world have more specialized needs than that provided by a Cisco academy-like education. We propose to explore whether NDA tools can raise the abstraction of network operations for rural networks.

Network Scripting: Simple Languages for Operators Our model for the development platform is the libraries and frameworks offered by Python for data mining—e.g., the IPython framework[1]. These tools have raised the level of abstraction for data analysis and enabled a new category of "data scientist" to exist. We aim for the same with our *network scripting* libraries: open-source software that aspiring operators like Stephen Oriecki and Farha Fathima can use both to learn NDA techniques and to use in the field.

Operators of rural networks, because they are called upon to do many tasks, are likely to find useful more familiar programming paradigms, rather than sophisticated specification languages [3, 9]. We aim to provide these operators with the ability to easily produce *network scripts* that automate common tasks. Analogous to the case for scripting in UNIX or IPython, we will develop high-level libraries for common

[1] `https://ipython.org`

design patterns like changing a peer and filtering at high traffic, with implementations that perform the appropriate low-level router commands. For instance, a network script can make a library call **No Transit** (X, Y) to state that traffic is disallowed between a Provider X and a Peer Y, and this will produce BGP commands at the ingress of X (to add a community tag) and at the egress (to filter based on this tag). For rural networks like Motech [39] the scripts will compile not to BGP but to commands for a UNIX router.

We will couple network scripting with a *simulator* that can be used by rural operators to test the impact of scripting commands on models of real networks, including their own. This can serve both as an educational tool and to help assess impact of configuration changes pre-deployment.

We hypothesize that a network scripting framework would allow a novice to more quickly and reliably learn how to operate a simple network. While this is a somewhat risky hypothesis, we propose to validate it as follows: Varghese and Millstein will look for 1-2 MS student who will first take the CCNA certification class (Santa Monica College close to UCLA offers such a class) and gain Cisco certification. We will use proposal funding to have the students visit Motech and Grameen (see letters). The student theses will be to understand needs, build a simulator, and develop a corresponding Network Scripting language (and compiler) for these networks. We will measure success via feedback from the researchers (Raghavan for Motech) and regional leaders (e.g., Hutchful for Grameen).

If successful, perhaps Network Scripting can be generalized to a broader class of networks, starting with home and small business networks and perhaps extending its reach to small data centers. Could such an approach enfranchise women and minority students in the US as well as around the world? We plan to examine this question by developing a scripting module for the successful SoNIC and PLMW workshops for minority and female students run by PIs Weatherspoon and Foster in recent years. Although these aspirations are perhaps lofty, we believe the simple starting point of libraries for rural networks is risky but achievable. Rural networks—kept working today by many operators with string and sealing wax [39]—can, we believe, benefit tremendously from *scripts and a simulator*.

Industry Outreach

We also plan to devote significant energy in connecting to industrial practitioners in addition to rural operators. To help learn about real-world problems and use-cases, we plan to organize workshops on NDA for industrial participants. We plan to invite leading network architects and operators to learn about what problems they are facing and explore whether NDA can help.

7 Leadership, Collaboration and Outreach

Center for Network Design Automation. To facilitate broader collaboration with industry and other faculty members, we will establish the *Center for Network Design Automation* (CNDA) at UCLA jointly with USC with George Varghese as Director at UCLA and Ramesh Govindan as co-Director at USC. The UCLA center will be modeled after earlier successful centers at UCLA such as CENS (which established sensor networks) which was also joint with USC.

Academic and Industry Engagement. This project has the potential to transform academia, by helping to establish a new field, and industry, by revolutionizing the way that networks are built and operated. To achieve this impact, however, requires engaging with stakeholders in each of these communities beyond the PIs. We have access to stakeholders at Microsoft, Google, Facebook, and Cisco to allow experimental validation at scale via interns. We will leverage our long-standing collaboration with MSR researchers in verification (Bjorner), software engineering (Godefroid), and networking (Mahajan). We will engage with other researchers such as Foster, Rexford, and Walker via the Cornell-Princeton Center.

Open-Source Software. To create, nurture, and grow a community of researchers working in Network Design Automation, we will encourage personnel to make software prototypes, mathematical models, topologies, traffic data, and other data sets associated with this project available on the project website under an open-source license. UCLA, where 4 of the PIs are based already has policies in place that permit release of software developed under sponsored research agreements. The PIs have a strong track record of releasing software prototypes and open data sets.

Collaborative Student Advising. Since we are in Los Angeles, we will strive to have at least half the students supported be jointly advised by a PI in UCLA and USC. It is students who are ultimately the bees that cross-pollinate researchers. We note that Govindan (USC) and Millstein (UCLA) have already demonstrated the effectiveness of this strategy in their initial work on network verification that produced tools like Batfish (Ari Fogel) and the PIC Interoperability Tool (Luis Pedrosa). Both Ari and Luis were jointly advised and the student meetings were both in USC and UCLA.

Tutorials and Mentoring Under-Represented Minorities. We will develop tutorials on NDA and present them at professional meetings such as POPL and SIGCOMM. PI Varghese has a strong track record of organizing such meetings including tutorials at FMCAD '13, SIGCOMM '15, and a session at the 2015 Microsoft Faculty Summit.

Annual Conference. To catalyze broader interest in NDA, we plan to organize an annual week-long conference, which will rotate between sites. We plan to invite leading network architects and operators to learn about the problems they are facing and explore how NDA can help. We will attempt to co-locate these workshops with existing events such as NANOG, ONS, or the P4 Workshop. The conference will provide an opportunity to give a status update on the project itself, to hear from outsiders working in related areas, and to allow an advisory committee of experts to meet and review our progress each year. We will invite the networking leads at Google, Microsoft, Forward Networks, and Cisco/Candid or their representatives (see letters) to serve on this committee to ensure we focus on problems that matter.

Outreach to the Developing World. The developing world is on the verge of being transformed by networked services. We believe the canvas of developing countries presents a unique opportunity for achieving broad adoption of our ideas and having positive societal impact. PI Barath Raghavan at USC will lead this effort. In addition, Raghavan will look to Jay Chen (NYU) and David Hutchful (Grameen) who have deep experience in networking in African contexts and who have offered to provide us guidance on aligning our research with impact on the ground. Beyond network scripting, we envision a User Interface (UI), one usable from a mobile device in the field, that, for example, makes it easy to add a new upstream provider or delete a link, obviating the need for more abstract specifications of regular expressions and route preferences. The Celerate system developed by Raghavan already provides a UI (that is used only for extracting, recording, and displaying the current topology and other network parameters); we envision extending it to synthesize configuration commands for BGP or low-level Linux route commands, using the idea of *network scripting* described in the educational section.

Research team. We have assembled a team from across 2 institutions on both coasts who wish to work together to develop the NDA vision *and* innovate in their own fields—*the special structure of the networking domain often reveals new approaches*. **Millstein** is an expert in programming language design [109, 65, 86], static program analysis, and verification [7, 55, 4, 56]. For the last 10 years, he has worked with **Govindan** and Ratul Mahajan at MSR to automatically identify security and interoperability errors [52, 78], and verify network configurations [26, 9]. **Govindan** managed the early development of the Routing Policy Specification Language (RPSL) which is used to coordinate routing policy across small providers. **Varghese** won the SIGCOMM Award for his work in making the Internet faster [104, 103] but has worked on bottom-up network verification for the last 6 years [49, 58, 82]. Further, his approach to interdisciplinary thinking via confluences [104] and his experience collaborating with theoreticians, architects, and database researchers will help meld this team into a synergistic unit. Varghese will coordinate the Los Angeles team. **Raghavan** was recently hired from ICSI and has worked with rural networks in Mendocino county and in Google data centers. **Netravali** is a recent UCLA hire from MIT where his PhD. dissertation focussed on debugging complex web systems; he plans to apply some of his insights in distributed systems to debugging networks from an application perspective. Finally, **Tamir** has extensive experience with switching hardware (he is the co-inventor of the well known Wavefront algorithm for scheduling crossbars that is used in PMC Sierra switches), computer architecture and testing. Tamir has been working with Millstein and Varghese on network operation for the last year, and provides a unique systems perspective on testing from an OS and Architecture perspective.

8 Intellectual Property and Ethics

The main artifacts produced by this expedition will be research papers, software tools, and an electronic textbook with accompanying teaching materials. We will create a project website with links to these artifacts. To encourage broad dissemination of our results, we will seek to publish in conferences, journals, and public respositories with reasonable copyright and open access policies (e.g., arXiv, ACM, USENIX, etc.) and will release our electronic textbook under an open-source license (e.g, Creative Commons). We will encourage project personnel to make software artifacts available under open-source licenses (Apache, BSD, GPL, etc.) to allow others to build on, adapt, and extend our work.

Although the research proposed here is primarily concerned with developing algorithms and tools, because networks play a critical role in modern society, there is also an implicit human dimension to our work. We will provide training for PhD students and postdoctoral resarchers on proper research conduct, paying special attention to specific issues related to network infrastructure—e.g., during an emergency, networks provide a critical line of communication for public safety officers, so ensuring that the network is reliable may literally save lives. We will seek out instutitional resources for guidance on any aspects of our research that impinge on further ethical issues.

9 Broader Impacts

This projects will have a number of positive impacts on society: 1. Public Clouds: We will partner with leading technology companies (e.g., Microsoft, Google, Huawei, and VMware) to develop, deploy, and evaluate NDA technology, with the goal of making them more efficient and reliable. 2. Operator Outreach: We will build a network scripting language and simulation tools for rural ISP operators (Grameen, Motech) and will work with them to improve the reliability and efficiency of their networks. We will also release training materials and tools online to reach operators in developing countries. 3. Technology Transfer: We will contribute to existing industrial efforts such as the P4 language and will also initiate new efforts to enhance the hardware primitives provided by router vendors especially for debugging. Varghese has a longstanding relationship with Cisco. 4. Education and Students: We will create a sustaining community of students who will drive the field forward via new courses that capture the key ideas behind Network Design Automation in a form that can be used and extended by the community. 6. Community Building: We will organize an annual meeting for industry and PhD students in NDA

References

[1] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. *SIGCOMM Comput. Commun. Rev.*, 38(4):63–74, August 2008.

[2] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74:47–97, Jan 2002.

[3] Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. NetKAT: Semantic foundations for networks. In *POPL*, January 2014.

[4] Chris Andreae, James Noble, Shane Markstrum, and Todd Millstein. A framework for implementing pluggable type systems. In *OOPSLA '06: Proceedings of the 21st annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 57–74, New York, NY, USA, 2006. ACM Press.

[5] Mina Tahmasbi Arashloo, Yaron Koral, Michael Greenberg, Jennifer Rexford, and David Walker. SNAP: Stateful Network-Wide Abstractions for Packet Processing. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM. ACM, 2016.

[6] Behnaz Arzani, Selim Ciraci, Boon Thau Loo, Assaf Schuster, and Geoff Outhred. Taking the Blame Game out of Data Centers Operations with NetPoirot. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM. ACM, 2016.

[7] Thomas Ball, Rupak Majumdar, Todd Millstein, and Sriram K. Rajamani. Automatic predicate abstraction of c programs. In *Proceedings of the ACM SIGPLAN 2001 Conference on Programming Language Design and Implementation*, PLDI '01, pages 203–213, New York, NY, USA, 2001. ACM.

[8] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. A general approach to network configuration verification. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 155–168, New York, NY, USA, 2017. ACM.

[9] Ryan Beckett, Ratul Mahajan, Todd Millstein, Jitendra Padhye, and David Walker. Don't mind the gap: Bridging network-wide objectives and device-level configurations. In *ACM SIGCOMM, Florianopolis, Brazil*, pages 328–341, August 2016.

[10] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014.

[11] Cristian Cadar, Daniel Dunbar, and Dawson Engler. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, pages 209–224, Berkeley, CA, USA, 2008. USENIX Association.

[12] Calyptix. Top 7 network attack types in 2016. Available at `http://www.calyptix.com/top-threats/top-7-network-attack-types-2016/`, June 2016.

[13] Marco Canini, Daniele Venzano, Peter Perešíni, Dejan Kostić, and Jennifer Rexford. A nice way to test openflow applications. In *NSDI*, 2012.

[14] Claire Cardie. Empirical methods in information extraction. *AI magazine*, 18(4):65, 1997.

[15] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane: Taking control of the enterprise. *SIGCOMM Comput. Commun. Rev.*, 37(4):1–12, August 2007.

[16] Cisco academy success stories. https://www.netacad.com/careers/success-stories/. Accessed: 2017-01-1.

[17] David Cohn, Rich Caruana, and Andrew McCallum. Semi-supervised clustering with user feedback. *Constrained Clustering: Advances in Algorithms, Theory, and Applications*, 4(1):17–32, 2003.

[18] David A Cohn, Zoubin Ghahramani, and Michael I Jordan. Active learning with statistical models. *Journal of artificial intelligence research*, 1996.

[19] Jason Cong and Zhiru Zhang. An efficient and versatile scheduling algorithm based on sdc formulation. In *Proceedings of the 43rd Annual Design Automation Conference*, DAC '06, pages 433–438, New York, NY, USA, 2006. ACM.

[20] P4 Consortium. P4-16 language specification. Available at http://p4.org/wp-content/uploads/2016/12/P4_16-prerelease-Dec_16.pdf, December 2016.

[21] Byron Cook. Available at http://www0.cs.ucl.ac.uk/staff/b.cook/, 2016.

[22] Daniel E. Eisenbud, Cheng Yi, Carlo Contavalli, Cody Smith, Roman Kononov, Eric Mann-Hielscher, Ardas Cilingiroglu, Bin Cheyney, Wentao Shang, and Jinnah Dylan Hosein. Maglev: A fast and reliable software network load balancer. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, NSDI'16, pages 523–535, Berkeley, CA, USA, 2016. USENIX Association.

[23] Dawson Engler, David Yu Chen, Seth Hallem, Andy Chou, and Benjamin Chelf. Bugs as deviant behavior: A general approach to inferring errors in systems code. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, SOSP '01, pages 57–72, New York, NY, USA, 2001. ACM.

[24] Facebook is launching rural internet access via satellite for africa. https://www.bloomberg.com/news/articles/2016-08-31/facebook-to-start-africa-satellite-this-week-to-find-rural-u Accessed: 2017-01-1.

[25] Seyed K. Fayaz, Tushar Sharma, Ari Fogel, Ratul Mahajan, Todd Millstein, Vyas Sekar, and George Varghese. Efficient network reachability analysis using a succinct control plane representation. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 217–232, Berkeley, CA, USA, 2016. USENIX Association.

[26] A. Fogel, S. Fung, L. Pedrosa, M. Walraed-Sullivan, R. Govindan, R. Mahajan, and T. Millstein. A General Approach to Network Configuration Analysis. In *Proc. Usenix Symposium on Networked Systems Design and Implementation (NSDI)*, 2015.

[27] Ari Fogel. Personal communication, 2016.

[28] Rodrigo Fonseca, George Porter, Randy H. Katz, Scott Shenker, and Ion Stoica. X-trace: A pervasive network tracing framework. In *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation*, NSDI'07, pages 20–20, Berkeley, CA, USA, 2007. USENIX Association.

[29] Nate Foster, Dexter Kozen, Matthew Milano, Alexandra Silva, and Laure Thompson. A coalgebraic decision procedure for NetKAT. In *POPL*, 2015.

[30] Dayne Freitag. Machine learning for information extraction in informal domains. *Machine learning*, 39(2-3):169–202, 2000.

[31] Aaron Gember-Jacobson, Raajay Viswanathan, Aditya Akella, and Ratul Mahajan. Fast control plane analysis using an abstract representation. In *Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference*, SIGCOMM '16, pages 300–313, New York, NY, USA, 2016. ACM.

[32] Solar-powered wi-fi: Another piece of the distributed energy framework. https://www.greentechmedia.com/articles/read/Solar-Powered-Wi-Fi-Another-Piece-of-the-Distributed-Energy-Framework. Accessed: 2017-01-1.

[33] Google compute cluster failure. https://status.cloud.google.com/incident/compute/16007?post-mortem. Accessed: 2016-05-1.

[34] R. Govindan, I. Minei, B. Koley, M. Kallahalla, and A. Vahdat. Evolve or die: High-availability design principles drawn from failures in a global-scale content provider. In *Proc. ACM SIGCOMM*, 2016.

[35] Albert Greenberg, Gisli Hjalmtysson, David A. Maltz, Andy Myers, Jennifer Rexford, Geoffrey Xie, Hong Yan, Jibin Zhan, and Hui Zhang. A clean slate 4d approach to network control and management. *SIGCOMM Comput. Commun. Rev.*, 35(5):41–54, October 2005.

[36] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Maziéres, and Nick McKeown. Where is the debugger for my software-defined network? In *ACM SIGCOMM HotSDN, Helsinki, Finland*, pages 55–60, August 2012.

[37] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Mazières, and Nick McKeown. I know what your packet did last hop: Using packet histories to troubleshoot networks. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 71–85, April 2014.

[38] Shaddi Hasan, Yahel Ben-David, Max Bittman, and Barath Raghavan. The challenges of scaling WISPs. In *Symposium on Computing for Development (DEV), London, UK*, pages 3–11, 2015.

[39] Shaddi Hasan, Yahel Ben-David, Max Bittman, and Barath Raghavan. The challenges of scaling wisps. In *Proceedings of the 2015 Annual Symposium on Computing for Development*, DEV '15, pages 3–11, New York, NY, USA, 2015. ACM.

[40] Christopher Hidey and Kathy McKeown. Identifying causal relations using parallel wikipedia articles. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1424–1433, Berlin, Germany, August 2016. Association for Computational Linguistics.

[41] C. A. R. Hoare. An axiomatic basis for computer programming. *CACM: Communications of the ACM*, 12, 1969.

[42] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven WAN. In *SIGCOMM*, 2013.

[43] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. Revisiting Readability: A Unified Framework for Predicting Text Quality. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP. Association for Computational Linguistics, 2018.

[44] Sundar Iyer. Available at http://yuba.stanford.edu/~sundaes/publications.html, 2016.

[45] Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. Deep unordered composition rivals syntactic methods for text classification. In *Association for Computational Linguistics*, 2015.

[46] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. B4: Experience with a globally-deployed software defined WAN. In *SIGCOMM*, 2013.

[47] Thorsten Joachims. *Text categorization with Support Vector Machines: Learning with many relevant features*, pages 137–142. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[48] Peyman Kazemian, Michael Chang, Hongyi Zeng, George Varghese, Nick McKeown, and Scott Whyte. Real time network policy checking using header space analysis. In *NSDI*, 2013.

[49] Peyman Kazemian, George Varghese, and Nick McKeown. Header space analysis: static checking for networks. In *NSDI*, 2012.

[50] Ahmed Khurshid, Xuan Zou, Wenxuan Zhou, Matthew Caesar, and P. Brighten Godfrey. Veriflow: verifying network-wide invariants in real time. In *NSDI*, 2013.

[51] Ralf Klinkenberg and Thorsten Joachims. Detecting concept drift with support vector machines. In *ICML*, pages 487–494, 2000.

[52] Nupur Kothari, Ratul Mahajan, Todd Millstein, Ramesh Govindan, and Madanlal Musuvathi. Finding protocol manipulation attacks. In Srinivasan Keshav, Jörg Liebeherr, John W. Byers, and Jeffrey C. Mogul, editors, *Proceedings of the ACM SIGCOMM 2011 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Toronto, ON, Canada, August 15-19, 2011*, pages 26–37. ACM, 2011.

[53] Zornitsa Kozareva. Cause-effect relation learning. In *Workshop Proceedings of TextGraphs-7: Graph-based Methods for Natural Language Processing*, pages 39–43, 2012.

[54] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach*. Addison-Wesley Publishing Company, 2009.

[55] Sorin Lerner, Todd Millstein, Erika Rice, and Craig Chambers. Automated soundness proofs for dataflow analyses and transformations via local rules. In *POPL '05: Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 364–377. ACM Press, 2005.

[56] Mohsen Lesani, Todd D. Millstein, and Jens Palsberg. Automatic atomicity verification for clients of concurrent data structures. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, volume 8559 of *Lecture Notes in Computer Science*, pages 550–567. Springer, 2014.

[57] Lun Li, David Alderson, Walter Willinger, and John Doyle. A first-principles approach to understanding the internet's router-level topology. *SIGCOMM Comput. Commun. Rev.*, 34(4):3–14, August 2004.

[58] Nuno P. Lopes, Nikolaj Bjørner, Patrice Godefroid, Karthick Jayaraman, and George Varghese. Checking beliefs in dynamic networks. In *NSDI*, 2015.

[59] Jonathan Mace, Ryan Roelke, and Rodrigo Fonseca. Pivot Tracing: Dynamic Causal Monitoring for Distributed Systems. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP. ACM, 2015.

[60] Sharad Malik. Network verification: Reflections from electronic design automation (eda). `http://research.microsoft.com/en-us/um/redmond/events/fs2015/speaker-slides/july8/Malik-Sharad_Faculty_Summit.pdf`. Accessed: 2016-05-01.

[61] Andrew Kachites McCallumzy and Kamal Nigamy. Employing em and pool-based active learning for text classification. In *Proc. International Conference on Machine Learning (ICML)*, pages 359–367. Citeseer, 1998.

[62] Steven McCanne and Van Jacobson. The BSD Packet Filter: A New Architecture for User-level Packet Capture. In *Proceedings of the USENIX Winter 1993 Conference Proceedings on USENIX Winter 1993 Conference Proceedings*, USENIX. USENIX Association, 1993.

[63] Nick McKeown. Mind the gap. In *SIGCOMM*, 2012. http://youtu.be/Ho239zpKMwQ.

[64] Ingo Mierswa, Michael Wurst, Ralf Klinkenberg, Martin Scholz, and Timm Euler. Yale: Rapid prototyping for complex data mining tasks. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 935–940, New York, NY, USA, 2006. ACM.

[65] Todd Millstein, Christopher Frost, Jason Ryder, and Alessandro Warth. Expressive and modular predicate dispatch for java. *ACM Trans. Program. Lang. Syst.*, 31(2):1–54, 2009.

[66] I. Minei and J. Lucek. *MPLS-Enabled Applications: Emerging Developments and New Technologies*. Wiley Inc., 3rd edition, 2015.

[67] Paramita Mirza and Sara Tonelli. An analysis of causality between events and its relation to temporal information. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 2097–2106, Dublin, Ireland, August 2014. Dublin City University and Association for Computational Linguistics.

[68] Christopher Monsanto, Nate Foster, Rob Harrison, and David Walker. A compiler and run-time system for network programs. In *POPL*, January 2012.

[69] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. Trumpet: Timely and Precise Triggers in Data Centers. In *Proceedings of the ACM Conference of the Special Interest Group on Data Communication (SIGCOMM '16)*, August 2016.

[70] Srinivas Narayana, Mina Tashmasbi Arashloo, Jennifer Rexford, and David Walker. Compiling Path Queries. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, NSDI. USENIX Association, 2016.

[71] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. Language-Directed Hardware Design for Network Performance Monitoring. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM. ACM, 2017.

[72] Srinivas Narayana, Mina Tahmasbi, Jennifer Rexford, and David Walker. Compiling path queries. In *USENIX NSDI, Santa Clara, CA*, pages 207–222, March 2016.

[73] Daniel Nenni. A brief history of EDA. Available at https://www.semiwiki.com/forum/content/1547-brief-history-eda.html, 2012.

[74] Ravi Netravali, Ameesh Goyal, James Mickens, and Hari Balakrishnan. Polaris: Faster Page Loads Using Fine-grained Dependency Tracking. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, NSDI. USENIX Association, 2016.

[75] Jong-Hoon Oh, Kentaro Torisawa, Chikara Hashimoto, Motoki Sano, Stijn De Saeger, and Kiyonori Ohtake. Why-question answering using intra- and inter-sentential causal relations. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1733–1743, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

[76] KU Leuven Oleksandr Kolomiyets, KU Leuven Marie-Francine Moens, University of Colorado at Boulder Martha Palmer, Brandeis University James Pustejovsky, and University of Alabama at Birmingham Steven Bethard, editors. *Proceedings of the EACL 2014 Workshop on Computational Approaches to Causality in Language (CAtoCL)*. Association for Computational Linguistics, Gothenburg, Sweden, April 2014.

[77] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up?: Sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, pages 79–86, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[78] Luis Pedrosa, Ari Fogel, Nupur Kothari, Ramesh Govindan, Ratul Mahajan, and Todd Millstein. Analyzing protocol implementations for interoperability. In *12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 15, Oakland, CA, USA, May 4-6, 2015*, pages 485–498, 2015.

[79] V. Perez and Y. Ben-David. Internet as freedom - does the internet enhance the freedoms people enjoy? In *Information Technology for Development*, 2012.

[80] Sharon E Perl and William E Weihl. Performance assertion checking. In *ACM SIGOPS Operating Systems Review*, volume 27, pages 134–145. ACM, 1994.

[81] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. Fastpass: A centralized "zero-queue" datacenter network. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 307–318, New York, NY, USA, 2014. ACM.

[82] Gordon D. Plotkin, Nikolaj Bjørner, Nuno P. Lopes, Andrey Rybalchenko, and George Varghese. Scaling network verification using symmetry and surgery. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL 2016, pages 69–83, New York, NY, USA, 2016. ACM.

[83] Rahul Potharaju, Navendu Jain, and Cristina Nita-Rotaru. Juggling the Jigsaw: Towards Automated Problem Inference from Network Trouble Tickets. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, NSDI. USENIX Association, 2013.

[84] R. Qiang, C. Rossotto and K. Kimura. Economic impacts of broadband. In *Information and Communications for Development*, 2009.

[85] L. Ryzhyk and et al. Correct by construction networks using stepwise refinement. In *NSDI*, 2017.

[86] Hesam Samimi, Ei Darli Aung, and Todd Millstein. Falling back on executable specifications. In *European Conference on Object-Oriented Programming (ECOOP)*, Lecture Notes in Computer Science. Springer, 2010.

[87] A. Sangiovanni-Vincentelli. The tides of eda. *IEEE Design and Test of Computers*, 2003.

[88] Sunita Sarawagi. Information extraction. *Foundations and trends in databases*, 1(3):261–377, 2008.

[89] Brandon Schlinker, Radhika Niranjan Mysore, Sean Smith, Jeffrey C. Mogul, Amin Vahdat, Minlan Yu, Ethan Katz-Bassett, and Michael Rubin. Condor: Better topologies through declarative design. *SIGCOMM Comput. Commun. Rev.*, 45(4):449–463, August 2015.

[90] Colin Scott, Aurojit Panda, Vjekoslav Brajkovic, George Necula, Arvind Krishnamurthy, and Scott Shenker. Minimizing Faulty Executions of Distributed Systems. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, NSDI. USENIX Association, 2016.

[91] Colin Scott, Andreas Wundsam, Barath Raghavan, Aurojit Panda, Andrew Or, Jefferson Lai, Eugene Huang, Zhi Liu, Ahmed El-Hassany, Sam Whitlock, H.B. Acharya, Kyriakos Zarifis, and Scott Shenker. Troubleshooting Blackbox SDN Control Software with Minimal Causal Sequences. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM. ACM, 2014.

[92] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47, March 2002.

[93] Nati Shalom. Gigaspaces blog post. Available at `http://blog.gigaspaces.com/amazon-found-every-100ms-of-latency-cost-them-1-in-sales/`, 2010.

[94] Scott Shenker. The Future of Networking and the Past of Protocols. `http://www.slideshare.net/martin_casado/sdn-abstractions`, 2011.

[95] Benjamin H Sigelman, Luiz Andre Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, and Chandan Shanbhag. Dapper, a large-scale distributed systems tracing infrastructure. Technical report, Technical report, Google, 2010.

[96] Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Sanjit Seshia, and Vijay Saraswat. Combinatorial sketching for finite programs. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XII, pages 404–415. ACM, 2006.

[97] John Stewart. *BGP4: Inter-Domain Routing in the Internet*. Morgan Kaufmann Publishers Inc., 2004.

[98] Praveen Tammana, Rachit Agarwal, and Myungjin Lee. CherryPick: Tracing Packet Trajectory in Software-defined Datacenter Networks. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR. ACM, 2015.

[99] Praveen Tammana, Rachit Agarwal, and Myungjin Lee. Simplifying Datacenter Network Debugging with Pathdump. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI. USENIX Association, 2016.

[100] H. Tangmunarunkit, R. Govindan, S. Jamin, and S. Shenkera nd W. Willinger. Network Topology Generators: Degree-Based vs. Structural. In *Proceedings of ACM SIGCOMM*, pages 188–195, Pittsburgh, PA, 2002.

[101] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *Journal of machine learning research*, 2(Nov):45–66, 2001.

[102] UCLA. Center for Information and Computation Security (CICS). Available at `http://web.cs.ucla.edu/security/`, 2016.

[103] George Varghese. *Network Algorithmics,: An Interdisciplinary Approach to Designing Fast Networked Devices (The Morgan Kaufmann Series in Networking)*. Addison-Wesley, San Francisco, CA, USA, 1999.

[104] George Varghese. Life in the fast lane: the confluence lens. In *SIGCOMM*, 2014. `conferences.sigcomm.org/sigcomm/2014/doc/slides/2.pdf`.

[105] Yaron Velner, Kalev Alpernas, Aurojit Panda, Alexander Rabinovich, Mooly Sagiv, Scott Shenker, and Sharon Shoham. Some complexity results for stateful network verification. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Eindhoven, The Netherlands*, pages 811–830, 2016.

[106] Andreas Voellmy, Junchang Wang, Y Richard Yang, Bryan Ford, and Paul Hudak. Maple: Simplifying SDN Programming Using Algorithmic Policies. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM. ACM, 2013.

[107] Kiri Wagstaff and Claire Cardie. Clustering with instance-level constraints. *AAAI/IAAI*, 1097, 2000.

[108] Kiri Wagstaff, Claire Cardie, Seth Rogers, Stefan Schrödl, et al. Constrained k-means clustering with background knowledge. In *ICML*, volume 1, pages 577–584, 2001.

[109] Alessandro Warth, Milan Stanojević, and Todd Millstein. Statically scoped object adaptation with expanders. In *OOPSLA '06: Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 37–56. ACM, 2006.

[110] Harrison Weber. Andreessen Horowitz-backed networking startup Forward Networks raises $11m. Available at `http://venturebeat.com/2014/11/26/andreessen-horowitz-backed-networking-startup-forward-networks-raises-11m/`, November 2014.

[111] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine learning*, 23(1):69–101, 1996.

[112] Wikipedia. Electronic design automation. Available at https://en.wikipedia.org/wiki/Electronic_design_automation, 2016.

[113] Niklaus Wirth. Program development by stepwise refinement. *Commun. ACM*, 14(4):221–227, April 1971.

[114] Hongkun Yang and S.S. Lam. Real-time verification of network properties using atomic predicates. In *ICNP*, 2013.

[115] Hongyi Zeng, Peyman Kazemian, George Varghese, and Nick McKeown. Automatic test packet generation. In *CoNEXT*, 2012.

[116] Hongyi Zeng, Shidong Zhang, Fei Ye, Vimalkumar Jeyakumar, Mickey Ju, Junda Liu, Nick McKeown, and Amin Vahdat. Libra: Divide and conquer to verify forwarding tables in huge networks. In *USENIX NSDI*, pages 87–99, 2014.