**LITERATURE SURVEY**

**1) “Network performance anomaly detection and localization,”**

**AUTHORS**: P. Barford, N. Duffield, A. Ron, and J. Sommers

Detecting the occurrence and location of performance anomalies (e.g., high jitter or loss events) is critical to ensuring the effective operation of network infrastructures. In this paper we present a framework for detecting and localizing performance anomalies based on using an active probe-enabled measurement infrastructure deployed on the periphery of a network. Our framework has three components: an algorithm for detecting performance anomalies on a path, an algorithm for selecting which paths to probe at a given time in order to detect performance anomalies (where a path is defined as the set of links between two measurement nodes), and an algorithm for identifying the links that are causing an identified anomaly on a path (i.e., localizing). The problem of detecting an anomaly on a path is addressed by comparing probe-based measures of performance characteristics with performance guarantees for the network (e.g., SLAs). The path selection algorithm is designed to enable a tradeoff between ensuring that all links in a network are frequently monitored to detect performance anomalies, while minimizing probing overhead. The localization algorithm is designed to use existing path measurement data in such a way as to minimize the number of paths necessary for additional probing in order to identify the link(s) responsible for an observed performance anomaly. We assess the feasibility of our framework and algorithms by implementing them in ns-2 and conducting a set of simulation-based experiments using several different network topologies. Our results show that our method is able to accurately detect and localize performance anomalies in a timely fashion and with lower probe and computational overheads than previously proposed methodologies.

2. **“Robust monitoring of link delays and faults in IP networks,”**

**AUTHORS**: Y. Bejerano and R. Rastogi

In this paper, we develop failure-resilient techniques for monitoring link delays and faults in a Service Provider or Enterprise IP network. Our two-phased approach attempts to minimize both the monitoring infrastructure costs as well as the additional traffic due to probe messages. In the first phase, we compute the locations of a minimal set of monitoring stations such that all network links are covered, even in the presence of several link failures. Subsequently, in the second phase, we compute a minimal set of probe messages that are transmitted by the stations to measure link delays and isolate network faults. We show that both the station selection problem as well as the probe assignment problem are NP-hard. We then propose greedy approximation algorithms that achieve a logarithmic approximation factor for the station selection problem and a constant factor for the probe assignment problem. These approximation ratios are provably very close to the best possible bounds for any algorithm.

3. **“Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs,”**

**AUTHORS**: C. Cadar, D. Dunbar, and D. Engler

We present a new symbolic execution tool, KLEE, capable of automatically generating tests that achieve high coverage on a diverse set of complex and environmentally-intensive programs. We used KLEE to thoroughly check all 89 stand-alone programs in the GNU COREUTILS utility suite, which form the core user-level environment installed on millions of UNIX systems, and arguably are the single most heavily tested set of open-source programs in existence. KLEE-generated tests achieve high line coverage — on average over 90% per tool (median: over 94%) — and significantly beat the coverage of the developers' own hand-written test suites. When we did the same for 75 equivalent tools in the BUSYBOX embedded system suite, results were even better, including 100% coverage on 31 of them. We also used KLEE as a bug finding tool, applying it to 452 applications (over 430K total lines of code), where it found 56 serious bugs, including three in COREUTILS that had been missed for over 15 years. Finally, we used KLEE to cross-check purportedly identical BUSY-BOX and COREUTILS utilities, finding functional correctness errors and a myriad of inconsistencies.

4. **“A NICE way to test Open Flow applications,”**

**AUTHORS**: M. Canini, D.Venzano, P. Peresini, D.Kostic, and J. Rexford,

The emergence of Open Flow-capable switches enables exciting new network functionality, at the risk of programming errors that make communication less reliable. The centralized programming model, where a single controller program manages the network, seems to reduce the likelihood of bugs. However, the system is inherently distributed and asynchronous, with events happening at different switches and end hosts, and inevitable delays affecting communication with the controller. In this paper, we present efficient, systematic techniques for testing unmodified controller programs. Our NICE tool applies model checking to explore the state space of the entire system--the controller, the switches, and the hosts. Scalability is the main challenge, given the diversity of data packets, the large system state, and the many possible event orderings. To address this, we propose a novel way to augment model checking with symbolic execution of event handlers (to identify representative packets that exercise code paths on the controller). We also present a simplified Open Flow switch model (to reduce the state space), and effective strategies for generating event interleaving likely to uncover bugs. Our prototype tests Python applications on the popular NOX platform. In testing three real applications--a MAC-learning switch, in-network server load balancing, and energy-efficient traffic engineering--we uncover eleven bugs.

**5. “Network tomography of binary network performance characteristics,”**

**AUTHORS**: N. Duffield

In network performance tomography, characteristics of the network interior, such as link loss and packet latency, are inferred from correlated end-to-end measurements. Most work to date is based on exploiting packet level correlations, e.g., of multicast packets or unicast emulations of them. However, these methods are often limited in scope-multicast is not widely deployed-or require deployment of additional hardware or software infrastructure. Some recent work has been successful in reaching a less detailed goal: identifying the lossiest network links using only uncorrelated end-to-end measurements. In this paper, we abstract the properties of network performance that allow this to be done and exploit them with a quick and simple inference algorithm that, with high likelihood, identifies the worst performing links. We give several examples of real network performance measures that exhibit the required properties. Moreover, the algorithm is sufficiently simple that we can analyze its performance explicitly.