The Case for Comprehensive Diagnostics

Chas DiFatta and Mark Poepping

Introduction

Creativity and invention can show us new capabilities, advanced engineering can make them practical, but it takes extensive monitoring and robust management to build the reliability required for a service to achieve status as a trustworthy utility.

The Internet has achieved great creative success, opening our eyes to radically new communication capabilities and distributed control opportunities. Yet for all the critical and commercial success of the past twenty years, we remain essentially unable to quantify or trace the actions of services and devices on the network. The Internet is a maelstrom of packets and protocols, an infrastructure built to speed it all along to quick completion, but constructed without detailed measures, without consistency of metrics or traces, with no visibility into how it's operating. While it appears true that most transactions complete as intended, and that most problems are transient and minor, there is no systemic data available to demonstrate by measures the actual state and trajectory of trustworthiness in the Internet. We have no assurance that we are actually talking with whom we intend, that our information is protected from those who would misuse it, or that our machines are in fact working on our behalf. This circumstance is especially problematic for security analysis, but it is also critical to performance and reliability management in any sort of critical control application (e.g. healthcare).

When there is a perceived problem with an application or supported service, a diagnostician must have the tools and information at her disposal to pinpoint the problem with reasonable certainty, in hopes of avoiding the problem in the future.

We propose creating a new capability, one to collect, manage, and correlate log and diagnostic event information to not only enable investigation of problems, but that can also support validation of correct operation in complex networked systems. If realized, the resulting capabilities will provide the visibility to understand, control and validate essential operation in the computing environments we increasingly rely upon.

Defining the problem

In today's Internet, software drives every capability and innovation is occurring at every layer in the application stack. Even a seemingly simple download of one web page often requires correct and efficient operation of dozens of software services on many different hosts, including the network links that connect them. Further, with the continuing advance of extremely small and highly functional embedded platforms, the number and range of involved hosts will continue to proliferate. As the drive for increased functionality continues so does the complexity in the system, and when failures occur it becomes increasingly difficult to trace the source of an error with certainty.

Failures have also become much more subtle. The most time-consuming failures are now usually not so straightforward as a prolonged error on a network link. They are often transient and may not be

entirely fatal for all relying applications or users. Many Internet protocols have been designed for overall resiliency, but individual users often still experience unusual failure conditions. Once a failure occurs, few implementations expose sufficient information to be reasonably diagnosed. Even for those where data is available, it is generally limited and not useful for correlation among multiple layers of the support infrastructure. In our current state, it is not at all uncommon to experience complex failures where each component in isolation appears to be functioning properly but the system as a whole is not delivering services as expected.

Further, user interface responses to error conditions are often cryptic. The best case scenario is for a user to receive an error message that will help them determine (and fix) the cause of an error. More likely however, the user sees only lack of success often presented as a very cryptic result (404: File not Found). If there is a help desk for the user to call, they are likely to have little useful information to present to a diagnostician, and often the only advice that can be given is to "try it again."

What is needed is a reliable, comprehensive diagnostic service that can report on the successes as well as failures in the enterprise application infrastructure. It should be able to provide information on the status of actual and individual transactions among customers in addition to providing uptime status of individual services. It should provide rich visibility at any point or abstraction required to troubleshoot a problem, including visibility on individual network flows, validity of specific authentication credentials, or the status of state machines for particular Internet protocols. It should be able to map diagnostic questions across these abstraction layers, correlating diagnostic information in time and topology to pinpoint errors. For safety and security, the service should be distributed, with powerful protections to match the importance of the services being managed.

Challenges in solving the problem

It is certain that this goal is broad and complex with significant long-term challenges to realization. One of the most subtle yet difficult challenges will be in addressing and leveraging the inertia and continuing effort of existing logging services. Almost all network, middleware, and application services now generate some diagnostic and/or performance data, often logged to a local file or perhaps stored via a network collection utility. There are also many monitoring systems available which will check system and service status. Given this, it is not unreasonable to wonder whether the problem is already solved. However, the best current systems are proprietary, and generally only minimally interoperable, even to basic issues of timestamps and naming. Most importantly, they seldom incorporate transactional data to enable diagnosis of specific user situations and tend to focus on only a single administrative domain, so it is generally not possible to incorporate data from areas where you do not have administrative control.

The key short-term technical challenges result mainly from the coordinating nature of a diagnostic service in leveraging data provided by other infrastructure and application services. The first challenge is to create a method of appropriately incorporating (or harvesting) existing diagnostic data from wherever it is currently available. It is important to realize that a fully incorporated diagnostic utility could generate an overwhelming amount of data, much of which will have very limited value, so the second challenge is to construct a capability to orchestrate and manage the data efficiently and for highest diagnostic value over time.

Even after you have all of the diagnostic information at hand, the final (and most important) challenge is to make sense of it – separating the wheat of relevant information from the chaff of unrelated events, finding the needle of cause in the haystack of effect. From a purely bottom-up standpoint, this is a daunting task indeed. We have a significant advantage however and we needn't work totally bottom-up. Remember that a key goal is to provide a diagnostician with the tools and information to diagnose real and perceived problems in the systems she supports. Our trail of log data is record of fact, but to gain the insight to pinpoint the source of a failure, we need to leverage knowledge of intent – a profile for what was *supposed to happen* as a benchmark against which to compare what actually did happen. Diagnosticians leverage their knowledge of application profiles informally as they investigate. Since we propose to automate the collection of facts, it follows that we should support collection of intents – a set of diagnostic profiles for supported applications to use in diagnostic analyses.

Crafting a New Approach: A Diagnostics Framework

We propose creating a new capability, a data framework and a technical implementation to enable the creation of a comprehensive diagnostic capability, one that will support collection, movement, storage, analysis, and long-term management of diagnostic and performance data to serve as a basis for enterprise and inter-enterprise forensics. Our framework consists of the follow key components:

- Common Event Record (CER) – The data structure common to all components of the diagnostic framework. A simple, powerful, and extensible schema to provide the basic normalization data for all diagnostic information.
- The Diagnostic Backplane – This is the network protocol and API components for transport and manipulation of CERs. It also provides an infrastructure for setup and management of the diagnostic system itself, including data flow manipulation, component configuration and data protection.
- Agents – This is the generic name for components that operate on CERs. They may create, direct, filter, compose, translate, or consume CERs. Agents also implement the interfaces through which the Diagnostic Framework interacts with other components in the system.
- Profiles – An application profile consists of a description of the expected behavior for (a piece of) the runtime of a given application, expressed in the terms of the Diagnostic Backplane. They are used to pinpoint operational breakdowns while diagnosing problems and are appropriate for comparing explicit expectations of application designers against the actual behaviors of the components they've built.

Conclusion

It is much easier to create something than it is to maintain and manage it. Inventors often do not consider the possibilities that their inventions may fail unexpectedly or be used ways not originally envisioned. Consequently, few tools are supplied with detailed specifications for their expected behavior, and fewer still have the capability to report on their own behavior.

We propose a fundamental shift to acknowledge operations management and diagnostic reporting as an essential requirement for any production system. The infrastructure we propose will enable a new generation of diagnostic applications and support a broad range of services, from real-time

troubleshooting to report generation to affirmation of correct operation in a supported service. The true power of this diagnostic capability will be realized when middleware and application services begin to leverage the feedback loop of diagnostic information to help guide their continuing product redevelopment and improvement efforts.

Whitepapers in Progress

- Architecture of the Diagnostics Backplane
- Key CER components to support correlation

Disclaimer

This material is based upon work supported by the National Science Foundation under Grant No. 0330626, Carnegie Mellon University, and Internet2. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.