THESIS

MONITORING AND CHARACTERIZING APPLICATION SERVICE AVAILABILITY

Submitted by

Daniel P. Rammer

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Spring 2018

ABSTRACT

MONITORING AND CHARACTERIZING APPLICATION SERVICE AVAILABILITY

Reliable detection of global application service availability remains an open problem on the Internet. Some availability issues are diagnosable by an administrator monitoring the service locally, but far more may be identified by monitoring user requests (ie. DNS / SSL misconfiguration). In this work we present Proddle, a distributed application layer measurement framework. The system periodically submits HTTP(S) requests from geographically diverse vantages to gather service availability information at the application layer. Using these measurements we reliably catalog application service unavailability events and identify their cause. Finally, analysis is performed to identify telling event characteristics including event frequency, duration, and visibility.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

In this work we aim to detect and classify application service unavailability events, which are a period of time where a service is unavailable from various geographic locations. Additionally, we identify characteristics of events to effectively predict future event behavior. This work focuses on HTTP(S) services. To achieve this we begin by developing an application layer network measurement infrastructure. Using this framework we monitor a number of service instances from geographically diverse regions to detect request failures. By performing analytics on these failures we identify unique events where a service was unavailable for a particular duration and identify the HTTP(S) request error. By analyzing characteristics of events we are able to predict event traits including frequency (how often events of this type occur), duration (how long events last), rate (number of failures per minute), and visibility (how widespread is the event viewable).

1.1 Service Motivation

Service administrators are unable to diagnose many user request failures without external information. While internal server errors, such as application failures, may be easily diagnosable many errors experienced by users such as DNS or SSL issues are difficult to be locally identified. Figure 1.1 shows a diagnosable DNS misconfiguration issue using distributed application layer measurements. In this example foo.com content is being served from the US East Coast. We collect measurements from four geographically diverse vantages, namely West US, Germany, Japan, and Australia. Measurements from the majority of vantages are successful but the vantage in Australia errors because it "Couldn't resolve hostname". With an alert, a domain administrator is provided actionable information that DNS is misconfigured in that region. Similarly, Figure 1.2 shows an example of a potential SSL configuration issue. In this example foo.com content is served by multiple locations, one on the US Eastern coast and the other in Southern India. Analysis of measurements shows content served by the Indian cache is successfully received whereas content served by the US East coast location fails because of an "Invalid SSL certificate". Again, domain administrators are provided actionable information and should easily resolve the issue. The examples, namely DNS and SSL failures, are presented because they are easily explained. However, any request failures by our measurement infrastructure are useful including redirect issues, general timeouts, or any variety of service failure.

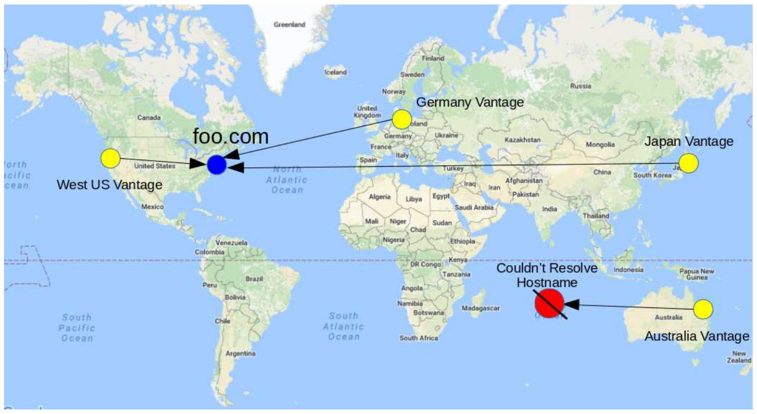

Figure 1.1: Example of detecting DNS misconfiguration

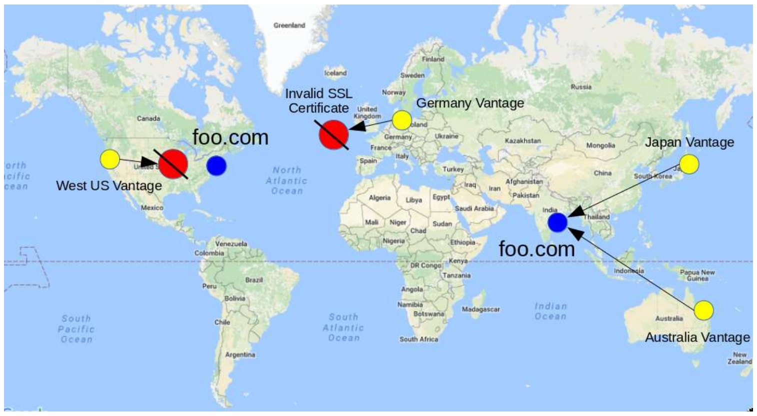

Figure 1.2: Example of detecting SSL misconfiguration

Another problem is that service users are often unaware of service unavailability cause, duration, visibility, etc. By analyzing characteristics of previous events we are able to reliably predict these attributes. For example, perhaps events similar to the "Invalid SSL certificate" example shown above are seen globally and last a duration of about two days. If the monitoring framework notices a particular domain has an "Invalid SSL certificate" it is able to alert users that the service will likely be unavailable for the duration of two days, and the effect will be seen globally. The obvious application is that users who rely on the service for critical functionality are able to redirect their work flow to account for the unavailability.

1.2 Related Work

Original attempts at network measurements like [1, 2] focused on developing tools for one-time measurements. The use of ICMP packets is central to common measurements like PING and traceroute. While the various ICMP based measurements were used successfully for many years, [3] shows that the accuracy of results when determining network connectivity can increase by 20%-30% when using TCP based measurements instead. This is commonly accepted since network administrators often route ICMP packets differently to reduce or entirely disallow measurements to infiltrate the network. Therefore measurements based on this protocol are often less effective. Additionally, network based measurements fail to successfully diagnose problems that may exist at the application layer.

Gradually, one-time tools tools evolved into distributed measurement collection frameworks. Work such as Scamper [4] and NIMI [5] aimed to make both ICMP and TCP based measurements available from a variety of geographically distributed vantages. In these frameworks, measurement definitions (with the exception of a few configuration options) are hard-coded. This constricts the ability to perform highly configurable and complex measurements. Scriptroute [6] rectifies this by allowing users to deploy measurement scripts. This allows essentially any measurement type/configuration supported by the script. The most recent success is RIPE's Atlas [7] framework. This framework allows users to execute highly configurable measurements at multiple vantages periodically. Additionally, it provides support for service based measurements (ie. HTTP(S), DNS, etc). These projects have shown the ability to scale effectively and provide reliable measurements. This work aims to build upon the most effective ideas presented in these frameworks by providing support for highly configurable, scriptable measurements. We go one step further by performing analysis on measurement results. In doing so we are able to draw complex conclusions about service unavailability causes, visibility, and duration among others.

There are many commercially motivated efforts for service/network monitoring including [8–14]. Commercial applications rely on customer subscriptions and as such only monitor paying services. They provide a large variety of measurement options including HTTP(S), DNS, SMTP, etc. Many also advertise custom customer specific measurements. Coverage is therefore extremely precise for subscribed customers and the result is an in-depth analysis of service availability/metrics available to each customer privately. Our application aims to provide a community service by monitoring a broad range of the most popular services. This work focuses solely on HTTP(S) measurements. However, the framework supports simple measurement implementation and deployment so extensibility is not a limitation. All of our measurements are make publicly available instantly after execution. Additionally, We perform periodic analysis on measurements to determine service unavailability events which are also immediately publicly available. By combining measurements from multiple services we are able to infer complex relationships. For example, the DynDNS DDoS attack in 2016.

Chapter 2

Measurement Infrastructure

Proddle is a distributed application layer measurement framework which we have developed and deployed. The decision to implement this application instead of leveraging other well established efforts hinges on providing a large variety of measurement types along with allowing the highest level of configuration for those measurements. Any conclusions drawn in this work are only as reliable as the measurements by which they are derived, therefore we design our measurements to execute similar to HTTP(S) requests from browsers. The three main components in the system are the vantages, bridge, and MongoDB or the backend database. The entire infrastructure is depicted in Figure 2.1. Below we described all components from an outside in approach. In each component section we explain the scalability of the component, thus proving the infrastructure's ability to scale as well.

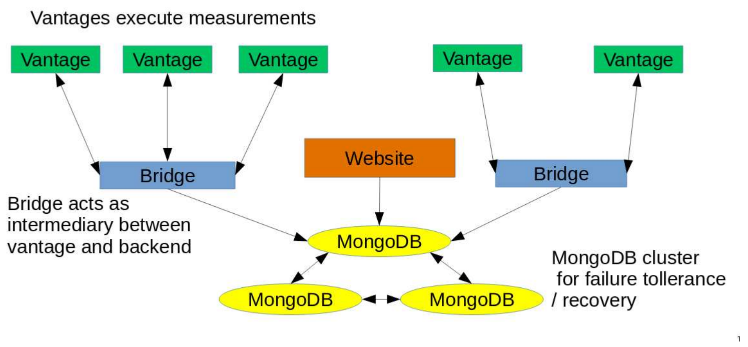
Vantages execute measurements
Vantage
Vantage
Vantage
Vantage
Vantage
Website
Bridge
Bridge
Bridge acts as intermediary between vantage and backend
MongoDB
MongoDB
MongoDB
MongoDB cluster for failure tollerance / recovery

Figure 2.1: Proddle architecture and component communication.

2.1 MongoDB

At the bottom of the framework is a MongoDB [15] cluster. MongoDB has a history of successful application in problems related to ours [16]. At the core, MongoDB is basically a distributed JSON document store. By using JSON as an internal storage format, the system removes the requirement of continually converting measurement result information between JSON and the relational data store format. This conversion process is not only removed from storing data in the database, we bypass it when providing information to users in JSON format as well. Another reason for choosing MongoDB is that the nature of Proddle's diverse measurement options would require a complex data model using any common relational database. Instead of traditional SQL tables, MongoDB refers to collections of records as documents (as that is how they are stored internally). MongoDB allows for records with varying fields to be stored and query-able within the same document. For example, in our measurements document we store records for HTTP measurements, which include fields for HTTP status code and content size among many others. In that same document we store records for traceroute measurements which include an entirely different set of result fields. MongoDB exposes a simple client application to query this document and to compare any fields available. Therefore we are provided an extremely simple interface for measurement storage and retrieval that can handle any fields that may be represented in JSON format.

2.2 Vantage

Vantages are the lightweight application instances which actively execute measurements. Measurements are scheduled to run at predefined intervals. Internally, this means a job is added to a threadpool. The threadpool provides two beneficial attributes. The first, having a configurable number of threads concurrently executing measurements effectively utilizes any desired fraction of available bandwidth. Additionally, the start time of each measurement is contingent on the internal threadpool implementation. Our implementation randomly chooses measurements from the execution queue, the result is a systematic execution delay for measurements. For example, 10,000 measurements are scheduled to be executed every 2 hours, in practice we see measurements can be executed anywhere in a 50 minute window after they are scheduled. Therefore when executing measurements from multiple vantages we see a variance of execution times. This increases the utility of the dataset by reducing the interval between measurements when multiple vantages are performing the same measurement. As seen in the results section, service unavailability duration is highly variable. By spacing out measurements we have a higher probability of viewing a specific event.

Measurement results are batched, compressed, and sent to a bridge application instance periodically. Additionally, vantages periodically poll bridges for measurement definition (ie. which domains to measure) and scheduling updates. The protocol used internally leverages hashing to reduce message size by providing a "diff" of the requested information.

2.3 Bridge

Bridges act as an intermediary, providing communication between the MongoDB backend and vantages. They play a few vital roles in the application. First they encapsulate database read/write operations. Meaning vantages never write to the database directly, instead data is sent to a bridge and inserted into the database from there. This reduces the complexity of the vantages to allow for their extremely lightweight resource footprint. Second, as briefly discussed in the vantage section, measurement definition and scheduling updates use a collection of hashing techniques to reduce bandwidth usage between vantages and the bridge.

2.4 Robustness

This system has been designed to continue operating regardless of failures in individual components. This is extremely important for a network measurement application. The system is tasked with monitoring service availability, as such it needs to be highly available itself. Any downtime could result in missing measurements that catalog a significant event. Additionally, users relying on the service expect it to be available. If it's not, it looses credibility. We have chosen to implement all components using Mozilla's Rust language [17]. This low-level language requires extensive error handling resulting in very resilient systems.

MongoDB provides failure tolerance through the notion of master/slave replication. In this paradigm all data in a master node is replicated to its slave nodes as well. It also provides data sharding functionality which allows sets of master/slave clusters to be responsibly for separate portions of the data. Combined these properties provide a robust, scalable backend for the system.

Two fail-over techniques are implemented in the vantage application to ensure message consistency in the event of any variety of communication failure between a vantage and bridges. The first is a simple bridge fail-over. Basically, we provide multiple bridge addresses when configuring the vantage application. A vantage will attempt to connect to each address in it's bridge configuration in a round-robin style, stopping once a successful communication occurs. This provides resilience during both intermittent link and bridge application failures. The second technique is flushing batched measurement results to disk if RAM hits a predefined usage threshold. This is important as the vantage application is designed to require minimal resources. If the application is unable to send the batch of measurement results to any of it's configured bridges it stores them and continues performing the measurements that have been scheduled. When communication with a bridge is restored any measurement results that have been written to disk are read and sent to the bridge.

Our current deployment consists of two separate bridge application instances This is vital to the scalability and resilience of the system as vantages rely on multiple communication channels as fail-over options. Additionally, as many vantages are initialized a single bridge application will be unable to handle the load. The deployment of multiple bridge instances assists the infrastructure in its scalability.

2.5 Security

In addition to resilience we strive to provide a secure system as well. There are two main constructs. The first is encapsulating database interaction within the bridge application. Obviously the goal of this work is to deploy an extensive application layer monitoring service. To achieve that we are required to deploy vantages in untrusted environments and thus subject ourselves to any variety of malicious actors. By forcing vantages to relay database read/write operations we can ensure validity in a trusted environment. This provides a layer of security as the vantage applications are unable to directly connect to the database, protecting our data from malicious deletion, corruption, etc.

Second we use the LetEncrypt [18] service to deploy and manage SSL certificates on all of the active bridge applications and the MongoDB instances. The benefits of SSL authentication / authorization are immense. In addition to verifying the hosts authenticity we are able to encrypt all of the data transfers in transit. Perhaps, this is currently not a high priority as all of the data is publicly available. In the future we foresee a number of "private" measurements required to expand the systems usability.

2.6 Libcurl

As stated earlier, we only leverage HTTP(S) GET measurements from Proddle for this particular work. At a low level we've used libcurl [19]. Using libcurl directly provides us many advantages over other techniques in terms of configuration and verbosity. Theoretically, all aspects of the request are able to be configured. We have spent considerable effort performing analysis of HTTP(S) request headers from Mozilla's Firefox and Google's Chrome to ensure our measurements most closely resemble browser requests. A few of the configuration options we use are listed in Table 2.1 below.

Table 2.1: A table containing libcurl configuration options and a short description.

Configuration Option	Description
HTTP User-Agent	String describing client OS and browser version
Follow Redirects	Allow request to follow up to 5 URL redirects
HTTP Transfer Encodings	Accept multiple encoding protocols
Upgrade Insecure Requests	Automatically attempt HTTPS requests on HTTP services

libcurl also makes a verbose variety of response attributes available. We are choosing to store a large fraction of those, excluding only those where their large size outweighs the utility. For example, the content of the request. The included attributes are listed below in Table 2.2.

2.7 Failure Detection

Here we will address the definition of measurement failures as it relates to this work. Since we are using libcurl for the HTTP(S) measurements a failure is defined as a libcurl failure. This means failures can be identified at multiple stages during the request. Individual error codes are explored further in the Chapter 4 Event Detection. Failures begin when resolving the hostname with DNS lookup issues and illegal URL formats. Then issues may arise during TLS/SSL certificate and peer certificate lookups and validations. After that there are server connection failures including general timeouts and the request reaching the maximum number of redirects (which we have set to 5). Finally, we see issues with data transfer and parsing such as unrecognized HTTP content. It should be noted that in the event of a measurement failure, a vantage will retry up to 2 more times with random delays introduced between attempts.

Table 2.2: A table containing libcurl response attributes and a short description.

Response Attribute	Description
HTTP Response Headers	Complete list of HTTP response headers
Content Size	Size in bytes (but not actual content)
Request Durations	Durations of total latency, hostname lookup, connection, etc
HTTP Redirects	List of HTTP URL redirects
Final Request URL	URL of final successful request
Final Request Host Address	IP address and port of final successful request

Chapter 3

Measurement Dataset

For this work we have decided to analyze a sample of just under 10 million HTTP GET measurements gathered during the one week interval of May 1st 2017 - May 8th 2017. We monitored the top 10,000 domains according to alexa.com [20] at 2 hour intervals. We performed measurements using each of 10 unique vantages deployed in geographically diverse locations. One vantage is deployed in the NetSec lab at Colorado State University in Fort Collins, CO. The other 9 were deployed on Microsoft's Azure cloud infrastructure with locations: East US, Central US, South Central US, West US, West Central US, West Europe, Southeast Asia, Australia Southeast, and Brazil South. All vantages are plotted on map below in Figure 3.1.

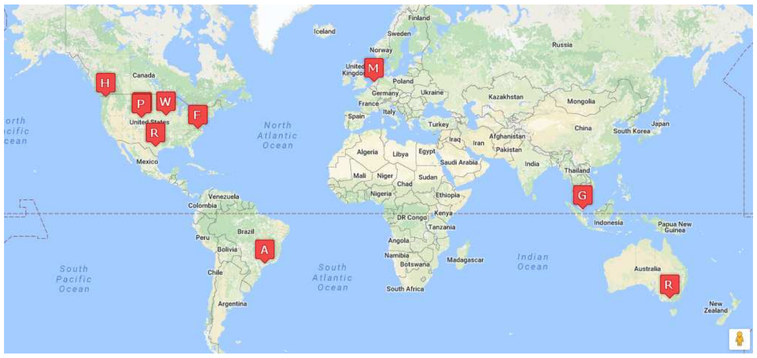

Figure 3.1: Proddle active vantage locations

Our initial analysis on this dataset revealed a failure rate (ratio of failed measurements to total measurements) of 2.525% over the entire dataset. There were 1239 domains that contained at least one failure. A plot of the failure rates for these domains ordered from highest to lowest is provided in Figure 3.2. Surprisingly, we see that 200 of these domains have a 100% failure rate, meaning all of the measurements performed for those domains failed. This is largely a result of alexa.com assigning all sub-domain traffic to the top level domains. For example traffic to a.foo.com and b.foo.com would be represented under the foo.com alexa domain. In this scenario the sub-domains, a.foo.com and b.foo.com, may serve content through HTTP whereas the top level domain, foo.com, may not. We have chosen to remove the measurements belonging to these 200 domains, resulting in a total failure rate drop to just 0.562%. We don't believe this reduces the utility of the dataset, as inclusion of the data would falsely skew our service availability findings.

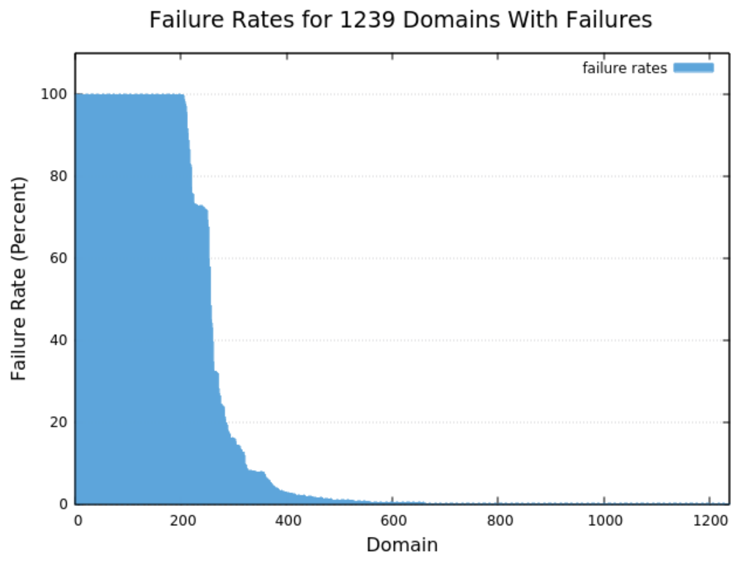

Figure 3.2: Total failure rates for each alexa.com domain.

Analysis into the 200 100% failure rate domains is provided in Figure 3.3 which shows unique error codes that were captured for the aforementioned domains. The largest portion of failures belongs to hostname resolution issues, an explanation for these cases is the instance described above where alexa.com rolls sub-domain traffic into the top level domain. Other failures include SSL certificate authentication issues, hitting the maximum number of redirects, and general timeouts among others.

Domains with failures are extremely diverse. The top 3 domains, in terms of failure counts during our sample, are runteki.com, nn.ru, and oeeee.com with 765, 762, and 748 failures respectively. The total number of measurements on a domain was around 800 during the sample period. There are a small number of domains that have an extremely high failure rate as seen here. Another notable set of domains includes yxngmwzubbaa.com and wipjyzwavojq.com. These domains exhibit similar characteristics to botnets using a DGA. While these examples are quite prevalent, they are also quite easy to discover as well because the alexa.com domain rank is highly variable. Even though these are quite irregular, we see no reason to exclude these domains from measurements as they are included in the alexa.com file and have successful measurements. A final example, includes gamestop.org, thepiratebay.org, and nintendo.com (among many others), which contained less than 5 failures during our measurement sample. We don't believe many of these cases to be terribly interesting, as experience shows a single vantage point may irregularly "timeout" on a measurement. Deeper analysis into the domains is beyond the scope of this work, but may be warranted for better insight into unavailability events.

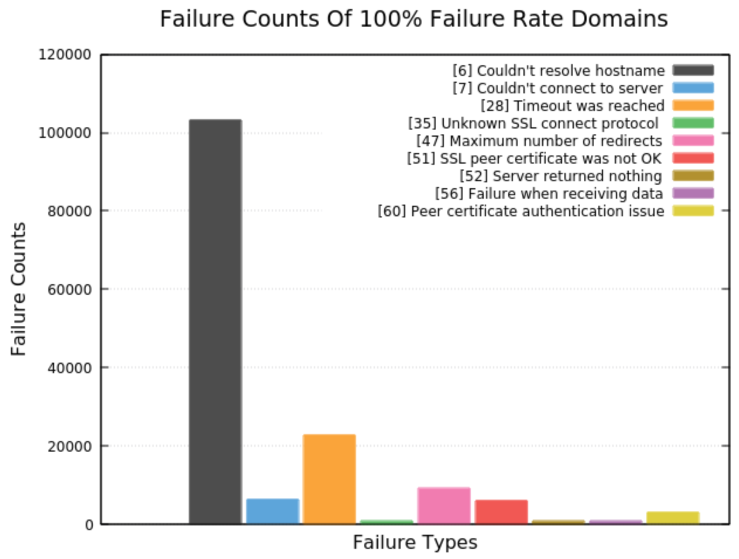

Figure 3.3: libcurl failure counts for each of the domains with a 100% failure rate.

Chapter 4

Event Detection

4.1 Methodology

We have chosen to leverage the DBSCAN [21] general clustering algorithm to effectively cluster individual measurement failures into service unavailability events. This algorithm has been studied extensively and has often produced better results than similar clustering algorithms [22–24]. Additionally it has been successfully applied to similar Internet based clustering applications [25]. Our current clustering criteria are quite elementary. We have chosen to focus on measurement failure timestamps, where failures that are close together are more likely to be clustered together. Usage of a clustering algorithm over other techniques, like manually designating clusters based on timestamps, was chosen to simplify implementation of more complex future analytics. There are many scenarios where we could extract more unintuitive relationships using different sets of clustering criteria. In terms of our clustering algorithm choice, we found DBSCAN provides many advantages, a few of which listed below.

- Does not require setting the number of clusters a priori
- Excels at grouping oddly shaped clusters
- Resistant to outliers by integration a notion of "noisy" points into the algorithm

The DBSCAN algorithm requires a user to define a distance function which calculates the distance between two points. Additionally, just two parameters are required, namely epsilon and min_points. Epsilon refers to the maximum distance allowed between two points for them to be in the same cluster. min_points is the minimum number of points required in a cluster.

Our distance function only takes the domain and the timestamp into account when determining the distance between two failures. Intuitively, this is all that is required to cluster failures into events as an event (as currently defined) is unable to span multiple domains. Using these two attributes we compute the distance as the difference between the two timestamps if the domains are the same; if not, the distance is not defined. We have chosen two use 3 as the min_points parameter to reduce false positives as the DBSCAN algorithm generally recommends a minimum value of 3. Finally, we address the epsilon value. The general recommendation is to use a kdistance plot and analyze sharp changes as permissible values. A k-distance plot is constructed by iterating through each data point and gathering the list of distances to the knearest neighbors (k being 3 in our instance). This results in a list of the distances to the k-nearest neighbors for each unique point. This list is then sorted and plotted. The idea is that sharp changes in the graph indicate epsilon values that will have a dramatic affect on the number of clusters / cluster size. The k-distance plot for our data indicates the 2 hour mark to be a good value to use as epsilon, the maximum distance between points in the same cluster. It is unsurprising that the 2 hour point is a good value because it's the interval at which measurements are executed. Our 2 hour measurement interval is a compromise between detecting transient failures (for example those due to routing convergences) and longer failures, and system resources. In our case, resources are the limiting factor and the interval can easily be altered pending resource availability. A shorter measurement interval such as 10 to 30 minutes is longer than a typical routing convergence, but may report false-positive failures for the monitored site. Alternatively an increasing measurement interval will require fewer resources but may fail to detect short-lived failures.

4.2 Results

Using this approach we were able to cluster the 54,929 failures in our dataset to 535 events spanning 342 unique domains. We can easily explore some example events in terms of the domains they belong to. The first example domain is creditonebank.com. The data noted two separate events, the first ran from May 1st at 12:08 AM to May 1st at 6:28 AM. In this event all 10 vantages were "Unable to resolve hostname" and there were 28 total measurement failures. In a separate event, the same domain had an "Unknown SSL protocol error in connect" failure from May 5th 8:07 PM to May 5th 10:47 PM encompassing 24 measurement failures. This exemplifies a situation where a single domain contained two separate unavailability events over the sample period. A second example is for the domain spiceworks.com. We noticed 4 vantages viewed a "URL using bad/illegal format" failure from 4 vantages May 3rd 8:28 AM to May 3rd 10:51 AM. We will explore these same events as we continue because they provide a good variety of event attributes.

Analysis of the events covering multiple facets presents interesting, even if intuitive, conclusions about failures per event, event duration, and the number of vantages which viewed each event. This information is included to lend credibility to the failure clustering techniques used. We see a large variety in event attributes which is expected.

We'll begin by addressing the number of failures per event. This data is displayed in Figure 4.1. We see the majority of the events consisting of less than 30 failures; then the data trends towards an exponential increase up to around 500 failures. We don't believe this view presents revolutionary findings, rather it reinforces the intuition one would expect.

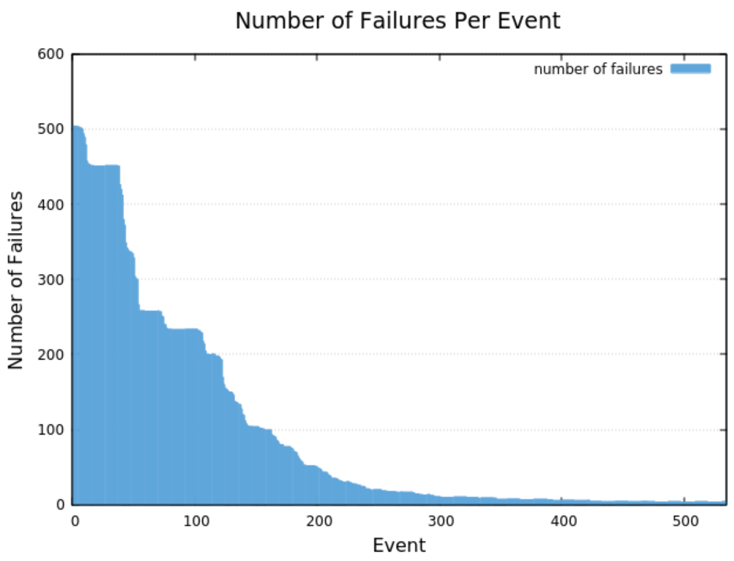

Figure 4.1: Total number of failures in each detected event sorted from greatest to least.

Figure 4.2 shows the number of vantages which viewed each particular event. This graph is rather interesting as we see large fluctuations between the number of vantages. It is reassuring that events are most commonly seen by all 10 (or even 9) of the vantages as that lends additional credibility to the event. The peak at 4 is explainable as a large number of events being short lived, therefore fewer vantages had the ability to measure within the given duration. An example of such an event is described earlier in this chapter as the lulumon.com event on May 5th (viewed by 2 vantages) lasted only 2 hours whereas the gamedog.cn event (viewed by all 10 vantages) lasted upwards of 5 days.

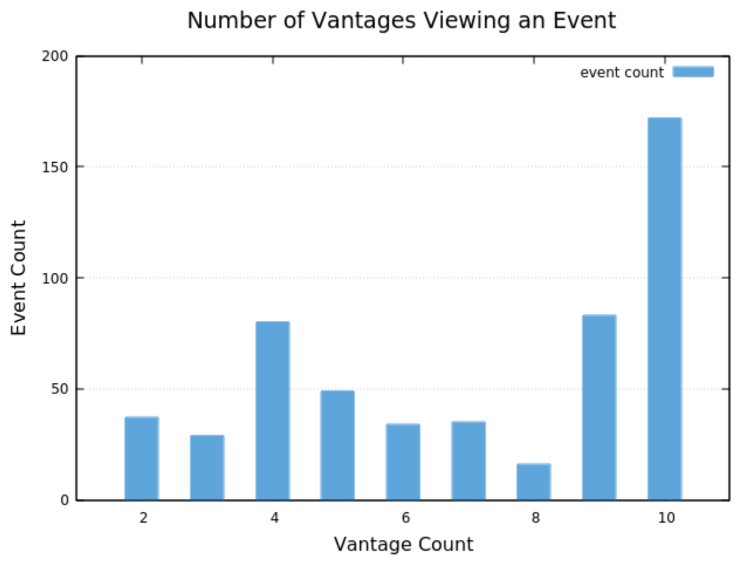

Figure 4.2: Total number of unique vantages that viewed each detected event.

Finally, we can analyze the durations for each event in Figure 4.3. Again, based on our previous data (number of failures per event, etc) this data is rather intuitive. The majority of events are relatively short lived, with an exponential increase in time as the data continues. Perhaps, the most noticeable facet of this data is the plateau around the 3000 minute mark. This is a construct of our sample window cutting off many events at the end of our 1 week duration.

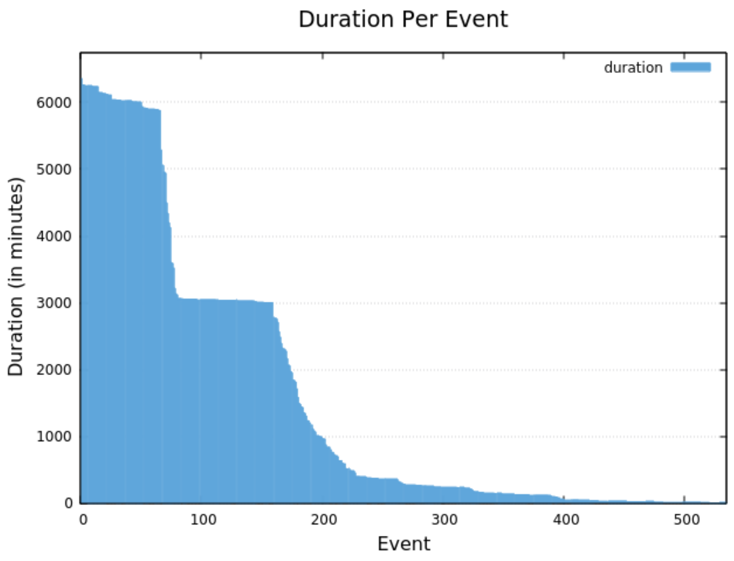

Figure 4.3: Duration (in minutes) of each detected event sorted greatest to least.

Chapter 5

Event Classification

5.1 Methodology

Classification of event cause is based directly on error codes/messages from libcurl. We have discovered 13 unique error codes (provided in Table 5.1). for measurement failures in the sample dataset. Of our 535 events 83% or 443 contain a single error code. This code is not necessarily the same for every event, just for failures within an individual event. These events are easily classifiable, as the cause is the single error code provided by their failures. Additionally, 15% or 81 contain 2 error codes where one error is a general timeout and related to the other, which we refer to as the primary error code. The remaining 2% of events that contain more than 2 error codes are currently unable to be classified. Our hypothesis is that the event detection algorithm is merging two actual events into one.

Table 5.1: A table containing libcurl error codes and the corresponding error messages.

Error Code	Error Message
3	URL using bad/illegal format
6	CouldnâĂŹt resolve hostname
7	CouldnâĂŹt connect to server
18	Transferred a partial file
23	Failed writing received data to disk/application
28	Timeout was reached
35	Unknown SSL protocol error in connect
47	Number of redirects hit maximum amount
51	SSL peer certificate was not OK
52	Server returned nothing
56	Failure when receiving data from peer
60	Peer certificate cannot be authenticated
61	Unrecognized HTTP Content or Transfer-encoding

5.2 Results

We have provided graph depicting the classification results is given in Figure 5.1 below. The x-axis represents the error codes for which events are classified and the y-axis is the number of events that fall into each classification. Classification frequencies range from the resounding majority, 249 events classified as "timeout" or error code 28, to the single event classified as error code 3 or "URL using illegal/bad format". Characteristics of each classification are explored in-depth in Chapter 6 Event Characteristics.

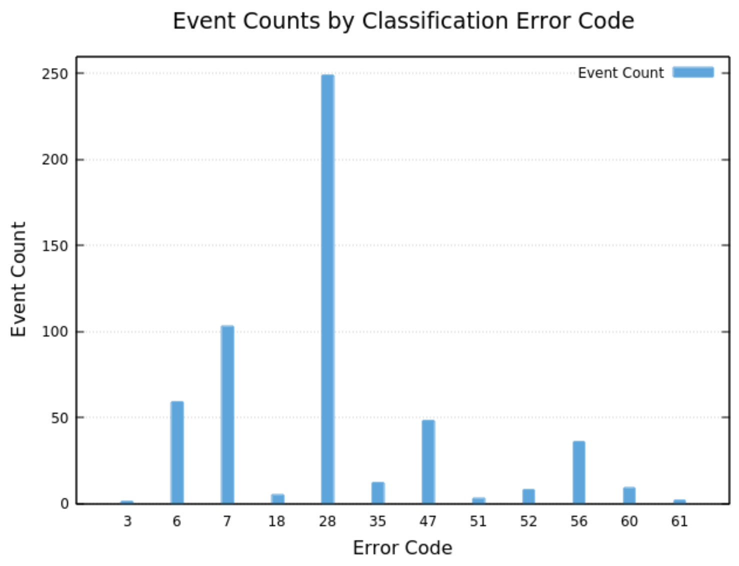

Figure 5.1: Number of detected events viewed by the libcurl error code they are classified under.

Chapter 6

Event Characteristics

In this chapter we choose to explore various characteristics of the event classifications. We have chosen to perform this analysis as a property of the Internet instead of focusing on a specific domain. That way results can be applied broadly. For example, the characteristics we have chosen to analyze can be applied whenever a failure occurs, even to sites that we don't necessarily monitor. We have identified four characteristics that present useful insight; namely frequency, duration, rate, and visibility. Explanations of each characteristic will be covered further as they are explored. Additionally we have devised a ranking system using High, Med (Medium), and Low and assigned one of these values for each characteristic for each classification. The aim is to use these rankings to construct a view of expected characteristics for each classification. Both service administrators and users will benefit. For example, when reporting a service unavailable to an administrator the service can gauge the potential visibility (or number of hosts that view an event). Similarly users experiencing an unavailable service may be able to understand the duration the service is expected to be unavailable for. All of this information is based on real observed events classified by the same failure cause.

Even though we have analyzed measurements for a relatively short time period (1 week) we are confident these results are representative of Internet failures. We performed over 800,000 measurements including almost 55,000 libcurl failures presenting an extremely large sample size. Our measurements were executed on the top 10,000 websites according to alexa.com which comprise a large majority of Internet requests. Finally, we experienced libcurl measurement failures at all stages of the HTTP request.

6.1 Frequency

Frequency refers to how often events of this classification occur. We have provided a graph showing this characteristic in Figure 6.1. Description of the graph has been omitted as it was covered extensively in Chapter 5 Event Classification.

Table 6.1 contains exact event counts and ranking values for each classification of events. The ranks are based on a visual interpretation of the data. Classifications 28 and 7 are ranked as High with 249 and 103 events respectively. Classifications 59, 47, and 56 are Med with 59, 48, and 36 events respectively. The rest of the classifications are ranked as Low.

Table 6.1: A table containing event counts for each libcurl error code/message classification.

Error Code/Message	Event Count	Class
(3) URL using bad/illegal format	1	Low
(6) Couldn't resolve hostname	59	Med
(7) Couldn't connect to server	103	High
(18) Transferred a partial file	5	Low
(28) Timeout was reached	249	High
(35) Unknown SSL connect protocol	12	Low
(47) Maximum number of redirects	48	Med
(51) SSL peer certificate was not OK	3	Low
(52) Server returned nothing	8	Low
(56) Failure when receiving data	36	Med
(60) Peer certificate authentication issue	9	Low
(61) Unrecognized HTTP content	2	Low

6.2 Duration

Duration refers to how long a particular event lasts. A graph showing the durations of events in each event classification is provided in Figure 6.2. Each value on the x-axis is an event that belongs to each classification and the corresponding y-axis value is the duration (in minutes) of that event. The graphs contain 9 separate lines, one for each of 8 unique error codes and the

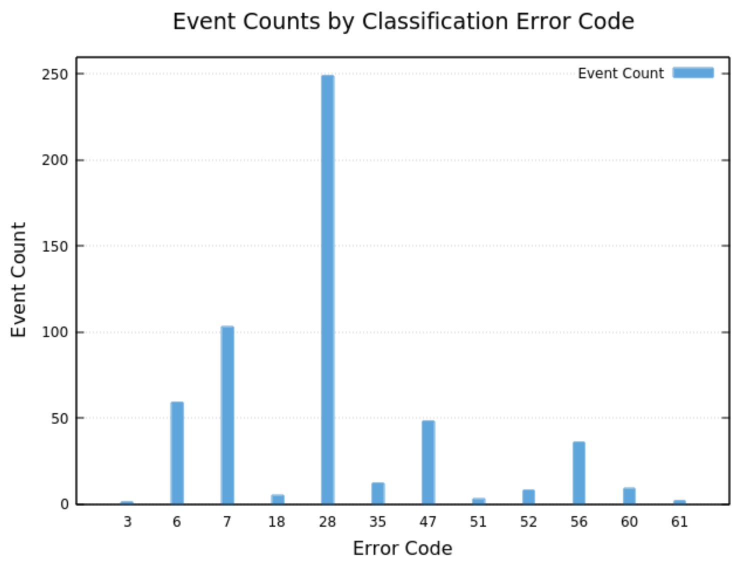

Figure 6.1: Number of detected events displayed as the libcurl error classification of the event.

final line encompasses the unrepresented error code classifications. Again, we see the plateau in durations at around the 3000 minute mark, again this is a construct of our 1 week sample window cutting off a number of events.

This graph provides a unique view of the large variances between event durations. For example, we see events like 47 and 7 that have relatively short tails resulting in a large negative slope. This attribute is contrasted by event classifications such as 28 and 6 that have longer tails and a more gradual decline. Using this information we can infer that events belonging to specific classifications will likely last longer than others. To quantify an event classifications duration we take the average of the duration of all events in that classification. The results are presented in Table 6.2 below. Again, we have chosen to split the durations into 3 ranks.

6.3 Rate

We next present the failure rate of event classifications as the number of failures per minute. The merit of this characteristic is to provide deeper understanding of the relationship between event size and event duration. For example, one would intuitively think the duration of an event is directly correlated to the number of failures in each event. We find that is not necessarily the case. The number of measurement failures is given in Figure 6.3. Each value on the x-axis represents a

Table 6.2: A table containing the average duration (in minutes) of an event for each libcurl error code/message classification.

Error Code/Message	Average Duration	Class
(3) URL using bad/illegal format	141	Low
(6) Couldn't resolve hostname	1583	Med
(7) Couldn't connect to server	2793	High
(18) Transferred a partial file	678	Low
(28) Timeout was reached	820	Low
(35) Unknown SSL connect protocol	1615	Med
(47) Maximum number of redirects	3096	High
(51) SSL peer certificate was not OK	2468	High
(52) Server returned nothing	1898	Med
(56) Failure when receiving data	923	Low
(60) Peer certificate authentication issue	774	Low
(61) Unrecognized HTTP content	4655	High

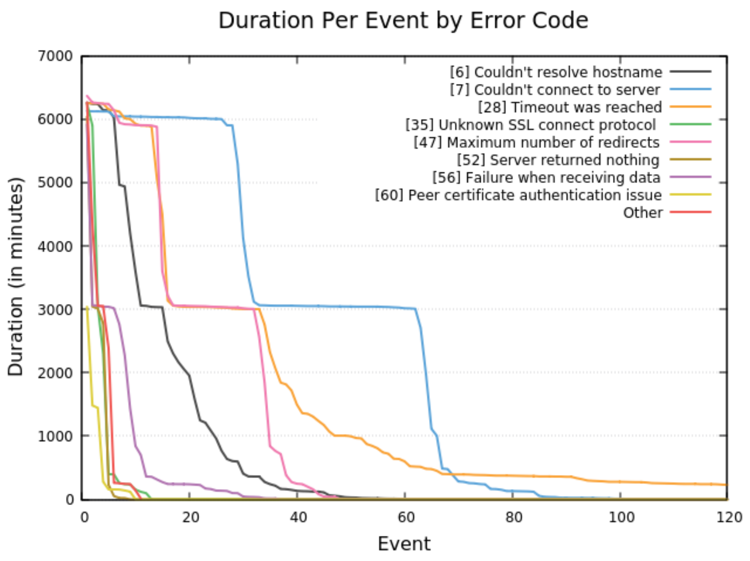

Figure 6.2: Duration (in minutes) of each detected event (sorted greatest to least) displayed by as the libcurl classification of the event.

single event, and the y-axis represents the number of measurement failures contained in that event. The interesting observation comes when comparing this graph to the durations presented in Figure 6.2. We see that the lines depicting event classification 28 are quite similar, meaning the number of failures and the duration of the event are highly correlated. When viewing the other lines we don't see this same level of correlation. We take this to mean that while events may be long lasting, they are not necessarily viewable by all vantages. The predefined 2 hour measurement interval means all vantages measure each domain the same amount of times within the interval, therefore for events with similar durations the only way the measurement failure counts would be different is that some vantages were having successful measurements within that duration.

Similar to previous characteristics we have provided Table 6.3 which contains the average failure rate value for each of the event classifications. In it we see values ranging from 0.137 to 0.028. We've decided to rank High as being above 0.1, Mid as 0.7 to 0.1, and Low as below 0.7.

Table 6.3: A table containing the average number of failures per event for each libcurl error code/message classification.

Error Code/Message	Average Duration	Class
(3) URL using bad/illegal format	0.028	Low
(6) Couldn't resolve hostname	0.121	High
(7) Couldn't connect to server	0.126	High
(18) Transferred a partial file	0.063	Low
(28) Timeout was reached	0.137	High
(35) Unknown SSL connect protocol	0.046	Low
(47) Maximum number of redirects	0.076	Med
(51) SSL peer certificate was not OK	0.164	High
(52) Server returned nothing	0.106	High
(56) Failure when receiving data	0.082	Med
(60) Peer certificate authentication issue	0.094	Med
(61) Unrecognized HTTP content	0.080	Med

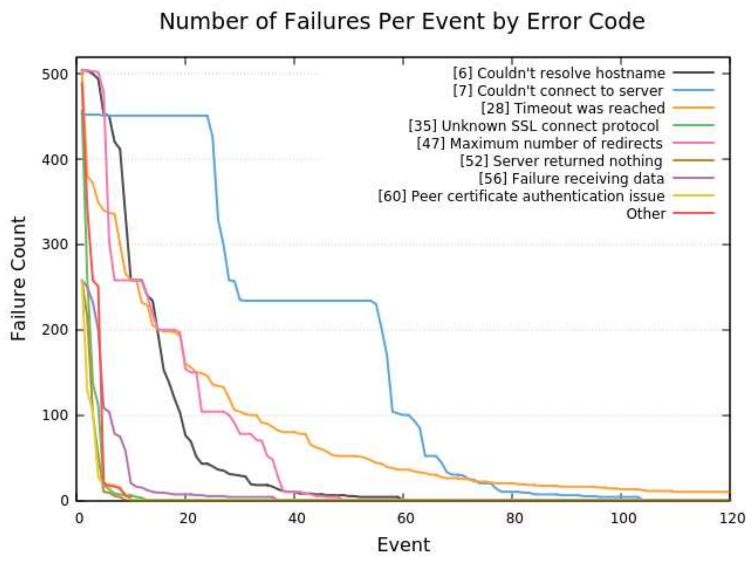

Figure 6.3: Number of failures per event (sorted greatest to least) displayed as the libcurl error classification of the event.

6.4 Visibility

Visibility is a simple characteristic, how many vantages view each event. We provide graph for this characteristic in Figure 6.4. On the x-axis are the number of vantages that viewed an event. The y-axis is the number of events. The various colors represent different event classifications as described in the legend.

In this graph we see that some event classifications are likely to be observed globally while others locally. This is determined by the number of vantages each event is viewable from. For example, we see events classified as 7 or "Couldn’t connect to server" as having a large number of events viewable by 9 or 10 vantages. Similarly 60 or "Peer certificate authentication issue" has all events viewable by all 10 vantages. Another interesting example is 47 or "Maximum number of redirects" where we see events are viewable by 2, 3, 4, or 10 vantages, but aren’t ever seen by 5 to 9 vantages. Rankings are displayed in Table 6.4. For ranking of this characteristic we have chosen anything above 9.0 to be High, between 5.0 and 9.0 to be Med, and anything below 5.0 is Low. We picked these numbers because there is a visible difference between the high and low values. 4, 9, and 10 vantages view a proportionally larger number of events (nearly double of the 4th highest) and 8 vantages view a very small number of events (less than half of the next smallest count).

Table 6.4: A table containing the average number of vantages that viewed an event for each libcurl error code/message classification.

Error Code/Message	Average Duration	Class
(3) URL using bad/illegal format	4.0	Low
(6) Couldn't resolve hostname	7.5	Med
(7) Couldn't connect to server	7.9	Med
(18) Transferred a partial file	9.8	High
(28) Timeout was reached	6.8	Med
(35) Unknown SSL connect protocol	6.7	Med
(47) Maximum number of redirects	6.1	Med
(51) SSL peer certificate was not OK	8.3	Med
(52) Server returned nothing	6.3	Med
(56) Failure when receiving data	5.8	Med
(60) Peer certificate authentication issue	9.8	High
(61) Unrecognized HTTP content	10.0	High

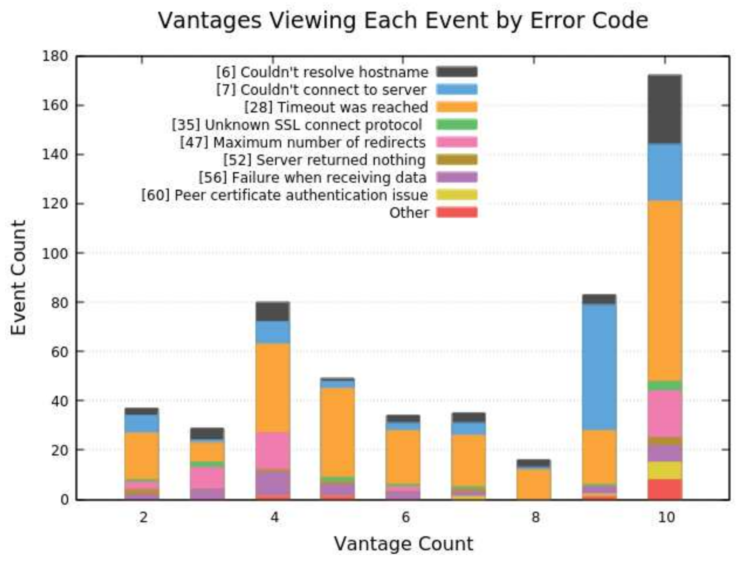

Figure 6.4: Number of vantages that viewed each event broken down the the specific libcurl error classification of the event.

6.5 Summary

In this section we present an aggregate of the characteristic rankings for each classification. It's provided in Table 6 below. The aim in providing this information is to effectively predict service unavailability based on event classification. This information is applicable for both domain administrators and service users alike. For example, if a measurement fails because "SSL peer certificate was not OK" (error code 51), we know this is an infrequent error that will likely last long and be viewable by most vantages. An administrator will find this information useful in that the unavailability is quantifiable and already diagnosed. They understand issues with the SSL certificate need to be resolved. Similarly service users, while likely uninterested in internal intricacies, can gauge the scope of service unavailability. In critical workflows this high level information can prove useful.

Table 6.5: A table containing an aggregation of characteristic rankings for each libcurl error code/message classification.

Error Code/Message	Frequency	Duration	Rate	Visibility
(3) URL using bad/illegal format	Low	Low	Low	Low
(6) Couldn't resolve hostname	Med	Med	High	Med
(7) Couldn't connect to server	High	High	High	Med
(18) Transferred a partial file	Low	Low	Low	High
(28) Timeout was reached	High	Low	High	Med
(35) Unknown SSL connect protocol	Low	Med	Low	Med
(47) Maximum number of redirects	Med	High	Med	Med
(51) SSL peer certificate was not OK	Low	High	High	Med
(52) Server returned nothing	Low	Med	High	Med
(56) Failure when receiving data	Med	Low	Med	Med
(60) Peer certificate authentication issue	Low	Low	Med	High
(61) Unrecognized HTTP content	Low	High	Med	High

Chapter 7

Use Cases

Given all of this information, it may be difficult to comprehensively understand how to present it in an actionable way to both service administrators and users alike. In this section we aim to explain using the two domain examples provided in Chapter 4 Event Detection.

If you recall, the first domain explored was creditonebank.com. There were two separate events, one spanning from May 1st 12:08 AM to May 1st 6:28 AM and the second from May 5th 8:07 PM to May 5th 10:47 PM. In this scenario we are able to provide actionable information to a service administrator for both events, the communication might go something like this. "We noticed creditonebank.com had hostname resolution issues from all 10 of our vantage locations during May 1st 12:08 AM to May 1st 6:28 AM". With this information a service administrator is able to make the correct changes to DNS configuration as necessary. The second event is perhaps more interesting. "During May 5th 8:07 PM to May 5th 10:47 PM we found the service was advertising unknown SSL protocols during connection". An event of this nature would likely go undetected without the information available through our system. This is because the users supported SSL protocols depend upon the medium used to retrieve the content. While developers often test with many browsers, it's unlikely they have been entirely comprehensive. In this instance it might be beneficial to add support for additional SSL protocols. In addition to service administrators, service users can benefit from the information we make available as well. For the first event, in near real-time, we can make available our predictions based on the characteristics we've explored. For example, "creditonebank.com is experiencing hostname resolution issues. Based on previous events we know this is a mildly frequent error. Additionally it will last for a medium duration (measurable in hours to days at most) and be viewable globally". This gives users the ability to take action on critical workflows relying on said service. Similarly the second event detected for this domain can be portrayed as "We're noticing some SSL protocol issues with creditonebank.com. This is a very infrequent event that will likely last hours to days and will not be viewable everywhere". If we wanted to provide more in-depth coverage of the event we could combine the information that all 10 vantages are viewing the event, but we notice that there are successful measurements within the event window. This could mean that a load balancer is redirecting requests, and the backend services donâĂŹt support the same SSL protocols. In this example we have provided information that is useful for both administrators and users.

The second example domain provided was spiceworks.com which contained a single event where 4 vantages noticed a "URL has bad/illegal format" from May 3rd 8:28 AM to May 3rd 10:51 AM. Under deeper analysis into the measurement failures we see that an external advertising website link failed to load correctly. This information is useful for the service administrator because a quantifiable error is restricting use of the site. The conversation might go "We noticed an external URL on spiceworks.com is illegally formatted". Many services rely on advertising revenue to remain profitable and as such this is an issue that should be fixed. This particular example is more usage for service administrators than users.

Chapter 8

Conclusion

In this thesis we presented Proddle, a reliable distributed application layer measurement framework. We cover the design decisions to improve over existing frameworks in the areas of scalability, security, and robustness. Using this infrastructure we are able to execute highly configurable, scriptable measurements. This work focuses on HTTP(S) GET requests. We successfully apply the DBSCAN clustering algorithm on one week of HTTP(S) GET measurement failures to identify service unavailability events. We then classified events, by their cause, based on the individual measurement failures that comprised them. Finally, we analyzed various characteristics of the classified events including frequency, duration, rate, and visibility. We then provided an aggregation of the various characteristics for service administrators and users.

Bibliography

[1] Savage, Stefan. "Sting: A TCP-based Network Measurement Tool." USENIX Symposium on Internet Technologies and Systems. Vol. 2. 1999.

[2] Luo, Xiapu, Edmond WW Chan, and Rocky KC Chang. "Design and Implementation of TCP Data Probes for Reliable and Metric-Rich Network Path Monitoring." USENIX Annual Technical Conference. 2009.

[3] Wenwei, Li, et al. "On evaluating the differences of TCP and ICMP in network measurement." Computer Communications 30.2 (2007): 428-439.

[4] Luckie, Matthew. "Scamper: a scalable and extensible packet prober for active measurement of the internet." Proceedings of the 10th ACM SIGCOMM conference on Internet measurement. ACM, 2010.

[5] Paxson, Vern, Andrew K. Adams, and Matt Mathis. "Experiences with NIMI." Applications and the Internet (SAINT) Workshops, 2002. Proceedings. 2002 Symposium on. IEEE, 2002.

[6] Spring, Neil T., David Wetherall, and Thomas E. Anderson. "Scriptroute: A Public Internet Measurement Facility." USENIX Symposium on Internet Technologies and Systems. 2003.

[7] Staff, R. N. "RIPE Atlas: A Global Internet Measurement Network." Internet Protocol Journal 18.3 (2015).

[8] site24x7. https://www.site24x7.com/

[9] uptimerobot. https://uptimerobot.com/

[10] uptrends. https://www.uptrends.com/

[11] binarycanary. https://www.binarycanary.com/

[12] alertra. https://www.alertra.com/

[13] statuscake. https://www.statuscake.com/

[14] thousandeyes. https://www.thousandeyes.com/

[15] MongoDB. https://www.mongodb.com/

[16] Van der Veen, Jan Sipke, Bram Van der Waaij, and Robert J. Meijer. "Sensor data storage performance: SQL or NoSQL, physical or virtual." Cloud computing (CLOUD), 2012 IEEE 5th international conference on. IEEE, 2012.

[17] Matsakis, Nicholas D., and Felix S. Klock II. "The rust language." ACM SIGAda Ada Letters. Vol. 34. No. 3. ACM, 2014.

[18] LetsEncrypt. https://letsencrypt.org/

[19] libcurl. https://curl.haxx.se/libcurl/

[20] Alexa.com. https://www.alexa.com/

[21] Ester, Martin, et al. "A density-based algorithm for discovering clusters in large spatial databases with noise." Kdd. Vol. 96. No. 34. 1996.

[22] Chakraborty, Sanjay, N. K. Nagwani, and Lopamudra Dey. "Performance comparison of incremental k-means and incremental dbscan algorithms." arXiv preprint arXiv:1406.4751 (2014).

[23] Suthar, Nidhi, Indr jeet Rajput, and Vinit kumar Gupta. "A Technical Survey on DBSCAN Clustering Algorithm." International Journal of Scientific and Engineering Research (2013).

[24] Chakraborty, Sanjay, and Naresh Kumar Nagwani. "Analysis and study of Incremental DB-SCAN clustering algorithm." arXiv preprint arXiv:1406.4754 (2014).

[25] Yang, Caihong, Fei Wang, and Benxiong Huang. "Internet traffic classification using dbscan." Information Engineering, 2009. ICIE'09. WASE International Conference on. Vol. 2. IEEE, 2009.