CANDID: Classifying Assets in Networks by Determining Importance and Dependencies

Scott Marshall

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2013-64
http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-64.html

May 15, 2013

CANDID: Classifying Assets in Networks by Determining Importance and Dependencies

by

Scott Michael Marshall

A thesis submitted in partial satisfaction of the
requirements for the degree of
Master of Science

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Sylvia Ratnasamy, Chair
Professor Vern Paxson, Co-chair
Professor Scott Shenker

Spring 2013

CANDID: Classifying Assets in Networks by Determining Importance and Dependencies

Abstract

CANDID: Classifying Assets in Networks by Determining Importance and Dependencies

by

Scott Michael Marshall

Master of Science in Computer Science

University of California, Berkeley

Professor Sylvia Ratnasamy, Chair

Professor Vern Paxson, Co-chair

This thesis introduces CANDID, a passive NetFlow-based network traffic analysis platform targeted at inferring relationships and dependencies among services running on hosts in enterprise networks. These networks present challenges of great scale, complexity, and nonstop dynamism, which hinder the ability for network administrators to maintain insight into the complex relationships that exist in these networks. Consequently, administrators do not always know how best to proceed if a network failure occurs. CANDID strives to empower administrators by illuminating these relationships, such that they will be prepared to remedy complex service failures.

The solutions presented here take the first steps towards understanding these complex in-network relationships, with a special focus on inferring one class of dependencies and detecting load balanced services. The focal point of this thesis is two radically different, yet complementary, strategies for inferring the presence of load balancing for pairs of systems.

A case study using real NetFlow data from the network located at Lawrence Berkeley National Lab is leveraged to validate the strategies presented here. Promising results indicate this problem space is rich with unanswered research questions and is worthy of further exploration.

Contents

List of Figures

List of Tables

Acknowledgments

The completion of this thesis, let alone my master's degree, would not have been possible without all of the guidance and support I have received from colleagues, family, and friends over the past two years (and in my life, in general). This section serves to acknowledge some of the important people that have impacted my life both personally and professionally, but my gratitude reaches far beyond the words on this page.

I am thrilled to have been co-advised by two amazing people, professors, and researchers: Dr. Sylvia Ratnasamy and Dr. Vern Paxson. Each advisor brought unique insights, perspectives, and strengths to the table, which have been great benefits to our collaboration. Both provided me phenomenal guidance, taught me how to become a better researcher, and were the inspiration of many of the ideas presented in this thesis. Vern provided me innumerable pointers for statistical techniques and taught me how to visually present data clearly & effectively. Sylvia showed me how to step back from a problem so as to not lose sight of our intuition & the bigger picture and could somehow always read my mind and knew what to say to keep me motivated when the challenges seemed insurmountable. I am immensely grateful to both Sylvia and Vern.

A special thank you to my family (Mom, Dad, and two brothers) for their continued love and support while I have continued to pursue my passion for learning. My parents taught me to always be a hard worker and to never give up, and that ongoing encouragement has made this thesis possible.

A special thank you to fellow graduate student Sangjin Han, who has not only become a good friend, but also collaborated with me on CANDID when I first started the research project in CS261N. Our joint work that semester formed what is now Chapter 5 and kicked off a path of research that has led to this thesis. Further, our ongoing research discussions have raised awareness to facets of research problems I hadn't considered, and have made CANDID better as a result.

Next, I want to thank all of my fellow graduate students in the UC Berkeley EECS NetSys Lab, who I don't just consider colleagues, but friends. All of you provided stimulating & insightful research discussions, lively & hilarious office conversations, and made our offices a fun place work. I wish you all the best in your future endeavors.

Last, but not at all least, I want to thank all of the close friends I have had from before I began my graduate studies. Each of you (listed alphabetically: Austin D., Chris C., David A., Rebecca W., and Shreyas S.) provided ongoing encouragement & support and curiosity for my research, all of which kept me motivated, even when tackling hard problems. It can be hard to come by good life-long friends, but I really have lucked out.

Thank you everyone!

Chapter 1

Introduction

Like the Internet, enterprise networks have grown significantly to hundreds of thousands of hosts and beyond, and Cisco Systems forecasts [2] that this growth is expected to continue for the foreseeable future. This increase in scale, combined with the dynamic nature of these networks, has made enterprise networks large and complex entities that are difficult to manage. In many circumstances, enterprise network administrators are not fully aware of what assets (service instances on network hosts) they have nor how these assets depend on and interact with one another. Therefore, administrators lack understanding of the roles network assets play towards supporting the missions of the enterprise. Consequently, administrators are unable to identify which assets are most critical and whether or not any resiliency is present (e.g., load balancing or failover) to protect those assets in the face of failures or attacks. Therefore, network administrators do not always know how best to proceed if an asset or network failure occurs, and security teams are unaware of which assets are in greatest need of security protection and monitoring.

To solve these problems, the goal of CANDID (**C**lassifying **A**ssets in **N**etworks by **D**etermining **I**mportance and **D**ependencies) is to passively analyze network traffic in an offline fashion, assign roles to assets (e.g., "authentication servers"), establish asset relationships (load balancing, failover, and dependencies), and rank assets by importance. In this thesis, I present several strategies for one phase of this analysis: establishing asset relationships, with a primary focus on identifying load balancing. The remainder of the CANDID solution is left for future work.

One aspect of CANDID that is different from many other projects in this space is that it only uses NetFlow [1, 56, 22, 21] records and not packet traces (headers and/or payloads) as the data source for network traffic classification and analysis. This is particularly important since most existing enterprise network infrastructure supports NetFlow-based measurement, which leads to a easy CANDID deployment strategy that does not require purchasing, configuring, and installing new infrastructure. While relying on flow data presents several challenges (see Chapter 4 for details), doing so does yield another benefit: use of flows in network measurement studies reduces privacy concerns since flows hide many user-identifying

characteristics. I take pride in presenting promising strategies that minimize abuse of user privacy.

To evaluate the methodologies described in this work, I performed a case study on Lawrence Berkeley National Lab (LBNL) SMTP [52] traffic. By analyzing anonymized NetFlow records reflecting SMTP traffic over several 24-hour windows, CANDID was able to identify several critical assets in the LBNL email processing system and inferred several instances of load balancing. To establish ground truth and to validate these results, I confirmed my interpretation of LBNL's SMTP behavior with LBNL network operations staff, who indicated that my conclusions were correct.

In summary, this thesis develops several strategies wherein CANDID successfully leverages passive NetFlow traffic analysis to infer relationships amongst assets in a real enterprise network. The remainder of this introductory chapter reiterates the problem statement and details the scope of this work in terms of the overall CANDID vision.

1.1 Problem Statement

As alluded to earlier in this chapter, my goals in this work are three-fold:

1. Identify assets in a network using only NetFlow traces.
2. Infer the complex relationships between those assets.
3. Determine the importance of individual assets and then rank them.

It is worth briefly noting here that there is more depth to these three goals than one might initially perceive: clearly defining the terms *asset*, *relationship*, and *importance* is an important component of any solution that attempts to tackle these goals. Detailed motivation for why it is important to address these goals is provided in Section 2.1.

1.2 Contributions

The contributions of this work span beyond the problem domain outlined by the three goals detailed in the previous section. In particular, some of these contributions come from the fact that this is a measurement study, and serve as solid techniques to use in other Internet and network measurement research. At a high level, these contributions can be summarized as follows:

- A sophisticated data handling pipeline that maps unidirectional NetFlow records from multiple network vantage points into a unified list of chronologically-ordered bidirectional network connections.

- Several data sanitization algorithms that can remedy common problems faced by researchers working with empirical data.
- Multiple strategies for identifying load-balanced assets which leverage a variety of connection attributes and characteristics.

Section 8.1 describes each contribution of this thesis in a more detailed fashion.

1.3 Thesis Summary

The scope of the CANDID project extends beyond that of this thesis alone; this thesis serves as a foundation of and a jumping-off point for further research towards a complete solution that encompasses all three aforementioned goals. While my initial research trajectory attempted to identify asset dependencies, I shifted my focus and instead decided to tackle inferring load balancing relationships, as this knowledge simplifies the network topology by aggregating assets known to be clustered into single entities, which in turn can simplify dependency analysis. Thus, the primary focus of this thesis is on understanding and inferring one type of asset relationship: load balancing, and while this may seem like a simple problem, it does not have a simple solution.

In this thesis, I develop a collection of strategies that take steps towards solving this problem, describe the pros and cons of each, and share the lessons I learned along the way. In particular, this thesis reveals that there is much more to modeling asset dependencies and relationships than natural intuition might first suggest. Readers should understand that while this work does not present a single comprehensive solution, there is research value in understanding the attempted techniques and solutions and their corresponding limitations. Further, the research community should take this work as a sign that this problem space is in fact quite challenging and deserves further attention. As I discuss in Chapter 8, I feel the best strategy for inferring load balancing is really a hybrid model that encompasses several of the techniques I designed.

The remainder of this thesis is organized as follows. Chapter 2 motivates this work and establishes key definitions and ideas used throughout the remainder of this thesis. In Chapter 3, I explore the expansive existing work in the research areas I build upon, while in Chapter 4, I provide context for NetFlow, illuminate the benefits and challenges of working with this type of measurement data, and describe the necessary data sanitation techniques required for subsequent analysis. Then, in Chapters 5, 6, and 7, I describe the three major strategies of analysis I have designed, providing design details and evaluations of each. Chapter 5 presents a technique for identifying asset dependencies from causal connection relationships, while Chapter 6 leverages similarities in connection distributions to infer instances of load balancing, and Chapter 7 revisits load balancing analysis through an alternative approach that deduces load balancing from hosts' peers. In Chapter 8, I recap the contributions of this work and discuss future directions, and I conclude in Chapter 9.

Chapter 2

CANDID Background

Before delving into the rich expanse of existing related research that precedes CANDID, it is important to motivate this work by better understanding the problems and challenges that have led to the goals outlined in Section 1.1. In addition, this chapter presents several key terms and concepts that are necessary to explain CANDID solutions in detail.

2.1 Motivation

To better understand the context in which the three goals from Section 1.1 originate, consider the following scenario:

> *You are a network administrator at a large corporation. Management tells you that to cut costs, you need to close down a company data center and move systems to a shared facility with minimal downtime. As you begin to plan the big move, you wonder...*
>
> 1. *I have limited space at the shared facility. Do I know what all the systems do? Are they all critical to daily operations, or can some be decommissioned?*
> 2. *Which assets can I relocate without impacting others? Are some in clustered configurations so they can be split apart during the move without impacting operations?*
> 3. *Is the network at the shared facility secure? My security team has a limited budget, so I need to tell them which assets need the most protection.*
>
> *You don't have the time to inspect each system individually nor do you have the option of deploying new specialized infrastructure capable of large-scale packet capture of the enterprise network. However, your current network infrastructure already supports traffic monitoring through use of NetFlow.*
>
> How might you answer these questions?

From this example, it should be clear that administrators face a number of uncertainties when working with networks of large scale. Even more importantly, the purpose of the fictitious scenario is to highlight some potential problems an administrator might face and the questions they might ask themselves when attempting to reach a resolution. Thus, it is important for administrators to be able to answer these questions because while they might have formal plans when new assets are initially deployed, these views are either inaccurate or quickly become outdated for a variety of reasons, and not knowing the answers to these questions presents big hurdles when problems arise.

Administrators Have Limited Visibility

While intuitively one might think that a network administrator has complete visibility into their network, this actually is not always the case. Consider, for example, that as IT divisions, which are spread across an enterprise, deploy new services, they may make use of an existing network resource but may not notify the resource owner that it is something their asset now relies on. At some point in the future, the owner may decommission the asset, not knowing that doing so would impact other services.

Another reason administrators can lack visibility is because they deploy assets they have not designed. Imagine a case where an appliance from a third-party vendor is purchased. Appliances are effectively black-box solutions: these systems can be poorly or incorrectly documented, such that an administrator deploying one really has no sense of the impact nor the dependencies the new asset has on existing network entities (e.g., perhaps a new network intrusion detection system which performs name lookups places a significant burden on DNS servers).

Finally, consider the case of a clustered asset: a service provided by several systems in either a *load balanced* or *failover* configuration (see Section 2.2 for definitions of these terms). Due to human error (e.g., either a failure to properly configure it or to test it prior to deployment), the cluster may not function as intended. For example, the transition from the active to the backup system during failover may not occur. This leads to yet another difference between the administrator's understanding of the system's behavior and actual real-world behavior.

Configurations Change

Even in cases where design plans and configuration specifications might exist when an asset is first added to a network, these truths do not necessarily hold indefinitely, as networks and assets are dynamic. One major cause of this dynamic nature is configuration changes, and these can take shape in several forms:

- To patch problems and minimize downtime, administrators hastily make changes to existing (documented) configurations, but in their haste, fail to document these mod-

ifications.

- Software patches may unknowingly or unintentionally modify existing well-understood asset behaviors and relationships.

It is also worth briefly noting that configuration details (e.g., service configuration files, firewall ACLs, monitoring parameters) for an asset are spread across multiple systems and *all* of these must be changed to avoid inconsistencies in both the administrator's view of the network and in asset behavior.

Problems Arise

The consequences of not knowing asset roles and lacking visibility into asset-asset relationships become particularly apparent when problems occur (and much to the displeasure of system and network administrators, problems do occur). Even in the case of simple hardware or software failures, accurate and complete network visibility is important as it helps guide the initial troubleshooting steps an administrator takes towards resolving the problem — *how can an administrator efficiently and effectively solve a problem if he/she does not know where to look?*

Unfortunately, the complexity of this situation spans far beyond the case of simple failures. Security teams provide protection for all systems in enterprise networks, but will usually add additional layers of security to critical assets. However, security teams need network visibility to know which assets are the critical assets, as a compromised key asset can lead to significant downtime and/or loss of revenue for the enterprise.

Finally, it is worth pointing out that due to asset dependencies, not all failures will be simple failures. These dependencies can establish long chains of dependent assets, such that the failure of one asset can lead to a "domino effect", wherein an entire sequence of assets subsequently fail.

I have argued how network and security administrators all have limited visibility into the roles their assets play, the importance of each asset, and the many complex inter-asset relationships that can exist in a large enterprise network. This inability to paint a complete picture of sophisticated dynamic relationships can lead to trouble when problems arise. Thus, it is for this reason — the need to understand asset roles and relationships — that I devised the goals outlined in the problem statement (Section 1.1) and CANDID takes steps towards accomplishing those goals.

2.2 Key Concepts

The remainder of this chapter shifts focus towards introducing concepts and defining terms that will be used throughout the remainder of this thesis.

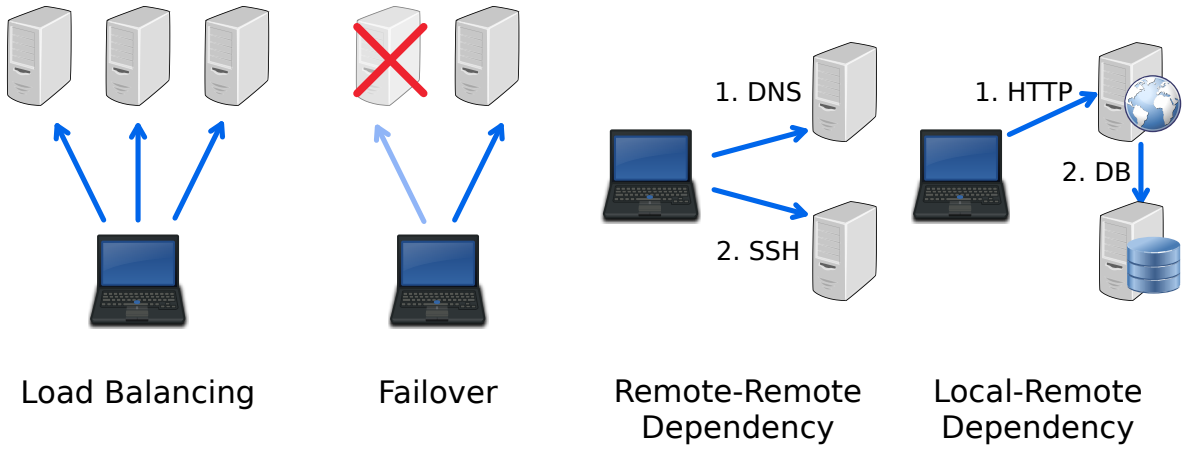

Figure 2.1: Four fundamental asset relationship types

Assets

The term *asset* in computer systems and networks can be thought of in many different ways; hardware (e.g., disks, hosts, or network appliances), software, network services, and information (e.g., confidential data or intellectual property) all fall under this umbrella. In the case of CANDID, an *asset* is defined as a network service provided by an individual *host entity* [1] denoted as a 3-tuple: ⟨L4 Protocol, IP Address, Destination Port⟩. Use of this same 3-tuple for distinguishing service roles in networks is common in previous work [13, 51, 17, 43].

In general, using the three-tuple to identify assets is a sensible choice: it is lightweight, intuitive, and is easy to determine (since it makes use of attributes present in NetFlow [1, 56, 22, 21] records from the IP header). The accepted downside of this choice, however, is that it cannot perfectly capture and express all service models, since there is not a one-to-one mapping between services and destination ports. For example, a web server may provide the same service to clients on multiple ports (e.g., 80 and 443) or an application that is not a web server could listen for clients on port 80 (as developers can choose to use any port(s) in an application).

Asset Relationships

CANDID examines four types of relationships that exist between assets, two of which (load balancing and failover) express *clustering relationships* and two of which (remote-remote dependency and local-remote dependency) express *dependency relationships*. These rela-

[1] A *host entity* is typically a single host. However, in the case of an asset provided by a cluster, the entire cluster is the *host entity* for that asset.

tionships are illustrated in Figure 2.1. The names and definitions for the two dependency relationships are from Orion [17].

Load Balancing In this relationship (depicted by the first diagram in Figure 2.1), the administrator wishes to distribute the client workload across a cluster of several servers to maximize throughput, minimize latency, and/or make use of fault-tolerance. A crucial characteristic [2] of load balancing is that all backend servers are configured to serve the same mission, typically with the same software and even the same hardware. However, the workload is not necessarily equally distributed among cluster members: load balancing workload distribution mechanisms can be static or dynamic in nature, and further, can make use of weighting. Load balancing can be achieved in several ways: (i) inline via a proxy or L4/L7 device (e.g., [20, 28, 14]), (ii) external servers routing requests (e.g., DNS round-robin), and/or (iii) directly by clients distributing the requests they issue (e.g., memcached client sharding [7]), but regardless of the mechanism, the key intuition is that all cluster members take on the same role since they all provide the same service.

Failover This is the second type of *clustered relationship* and is depicted in the second diagram of Figure 2.1. In a failover configuration, only one system is active at any given time (unlike load balancing, where several systems are active simultaneously). Failover provides fault-tolerance guarantees, but does not improve performance like load balancing does. Failover is frequently implemented by sending a heartbeat [59, 55] between the two systems: when the backup server detects that the primary server has failed (by a lack of heartbeat packets), it takes over the active role. The technique for identifying failover is similar to that of load balancing (in that failover systems exhibit the same service behavior), except for the fact that network connections to failover systems will be mutually exclusive rather than simultaneous (in time).

Remote-Remote Dependency A remote-remote (RR) dependency is one in which a system must first contact one host before issuing a request to the desired host (i.e., connections emit from the same client in an iterative fashion). The "remote-remote" name stems from the fact that the system relies on two remote (i.e., not local to the system) services to accomplish the desired task. For example, using the third scenario illustrated in Figure 2.1, the inferred dependency relationship is: *the client depends on the remote DNS service to use the remote SSH service.*

Local-Remote Dependency This is the second general type of *dependency relationship.* A local-remote (LR) dependency is where a system must issue a request to a remote system in order to complete an outstanding request issued to a local service. With LR dependencies, connections appear in a recursive pattern. This scenario is depicted

[2]Recall that a major focus of this thesis is to accurately and robustly identify characteristic(s) which act as indicators of *similar asset behavior.*

in the last diagram in Figure 2.1, wherein the web server needs a response from the database server in order to respond to the client. Here, the inferred dependency relationship is: *the web server depends on the database server*, but one could extend this to include the notion that *the client's task depends on the database server*.

Asset Importance

The final phase of CANDID analysis is to rank assets using a quantitative metric that captures the importance of each asset. The motivation behind establishing this single importance metric and ranking is that it is difficult for administrators to compare thousands of assets when they are each involved in complex combinations of relationships. A single asset importance metric provides network and security administrators at-a-glance illumination as to which assets are critical (i.e., if they fail or are attacked that the enterprise at large is subject to service disruptions, the inability to complete tasks, and/or financial loss).

Chapter 3

Related Work

CANDID spans across a multitude of network research areas: network measurement, general-purpose traffic classification, host role identification, host grouping, flow ranking, and application dependency analysis. This chapter serves to explore the contributions of existing literature to develop the context for the ideas that CANDID builds upon and to highlight the ways in which CANDID is different. An important takeaway from this chapter is that none of the existing research has made use of role identification in combination with host grouping to specifically identify cases of load balancing — as previously stated, this is the primary intent of this thesis and one of the goals of the overarching CANDID work. Subsequent sections of this chapter address related work in each of the six aforementioned research areas.

3.1 Network Measurement

Performing a network measurement study presents many challenges, particularly with respect to data handling: empirical data can have spikes, network traces (even when expressed as NetFlow records) are frequently large in size, and timestamps can be inaccurate.

Some fundamental network measurement concerns are outlined in [48]. While this work addresses a wide range of challenges (e.g., maintaining measurement metadata, performing consistency checks, ensuring experiment reproducibility), this thesis most closely relates to the points about the precision of system clocks and the presence of spikes and outliers in network trace data. The first point is particularly relevant to CANDID since several forms of analysis in this thesis rely on flow timestamps. As evidence that these concerns are in fact a legitimate problem, the LBNL flow data contained several cases where flows were of zero duration or where flow characteristics exhibited spikes. These issues are discussed in detail in Chapter 4.

Many of the analysis techniques presented in this thesis are based on statistical methodologies. In [46], three key ideas are presented that are reused in Chapter 6: the λ^2 metric

of discrepancy (which is used to quantitatively assess closeness-of-fit of two empirical distributions), an estimation of bin widths for the χ^2 test, and a quantitative mechanism for comparing random variables.

3.2 General-Purpose Traffic Classification

There is a large corpus of existing research that presents strategies for classifying network traffic, and the intention here is to provide context via a cross-section of that literature.

Kim et al. [37] provide a survey of several current traffic classification proposals, separated into two broad categories: flow-feature based and host-behavior based. The former describes classification strategies that leverage flow attributes (e.g., flow duration or packets transferred), while the latter considers the interactions hosts have with other hosts. CANDID leverages these same two categorizations: Chapter 6 describes strategies focused on flow attributes, while Chapter 7 focuses on hosts' interactions with their peers.

Unlike many other traffic classification approaches which assume complete bidirectional traffic visibility, [26] introduces a traffic classification technique for cases where only unidirectional traffic is visible (this frequently occurs at the network core where routes are asymmetric). The proposed solution makes use of k-means clustering on six traffic characteristics (packet count, mean packet size, mean payload size, bytes transferred, flow duration, and mean packet inter-arrival time). While the raw data used by CANDID is composed of unidirectional NetFlow records, CANDID first maps unidirectional flows into bidirectional connections (see Chapter 4 for details). However, since not all unidirectional flows have a corresponding flow to pair up with to form a bidirectional connection, one could consider using this strategy for those corner cases. CANDID does not do this since the fraction of unpaired flows in the LBNL dataset is small (see Chapter 4), but should other datasets have this characteristic, this approach is worth considering.

Moore and Papagiannaki highlight the issue that there is not a one-to-one mapping between networked applications and ports [42]. The proposed solution introduces eight classifiers that are applied to packet traces to determine the true application associated with a flow. Unfortunately, the approach makes use of DPI (deep packet inspection) in several classifiers by leveraging packet traces that include full payloads. Since CANDID minimizes user privacy concerns by only using NetFlow records, this solution is not applicable, and the aforementioned one-to-one mapping problem is a recognized shortcoming of the *asset* definition introduced in Section 2.2.

The authors of [13] propose several simple criteria to translate unidirectional flow records into bidirectional flows using a few flow characteristics: endpoint addresses, number of packets, flags, and timestamps. As previously mentioned, CANDID also accomplishes this same task, but does so by considering a different set of flow characteristics: endpoint addresses *and ports* and timestamps (but not packet count nor flags). A challenge faced with unidirectional to bidirectional flow mapping is establishing which endpoint is the server and which

is the client [1], and [13] proposes combining seven heuristics via Bayesian inference to make this determination. While CANDID design details are saved for Chapter 4, I propose a far simpler approach that proves to work well with the LBNL dataset.

Kannan et al. present a strategy for inferring application session structure in a semi-automated fashion [34, 35]. This work has two contributions: (i) causal aggregation of individual connections into user-initiated sessions, and (ii) extraction of application communication structure from these sessions. The first contribution leverages knowledge of Poisson arrival processes in combination with hypothesis testing to assess if a connection is independent of others versus caused by another. The second contribution infers application session structure by mapping state machines (where edges represent individual connections of a session) to applications. While CANDID does not make use of these contributions directly, it does leverage statistical methodologies similar to the ones presented in this work.

3.3 Host Role Identification

Basic traffic classification can be extended to infer the roles specific hosts have in a network. Role assignment is important for two reasons: it provides stronger inference into host behavior than simple IP↔service asset mappings, and it establishes a baseline of expected behavior for each system in the network. The latter feature is primarily useful in forensic contexts, where a change in a hosts' role may be a signal of malicious activity; CANDID is only concerned with the former benefit.

One proposal for host role assignment through behavioral classification is driven by forensic motivations [41]. Nonetheless, this proposal does determine host roles by processing NetFlow records to construct 3-tuples of the format ⟨L4 Protocol, Destination Port, Volume Tier⟩ for each host and uses those in a neural network to discover a host's most prominent behavioral pattern. The "Volume Tier" tuple component represents placement in one of five traffic volume tiers, which helps to constrain the set of tuples produced. However, between the need to train the neural network with baseline data and the potentially brittle selection of the five fixed traffic volume thresholds, this solution may not hold up well for datasets other than that used by the authors.

In [25], the authors outline nine features they use in combination with a clustering algorithm to assess host roles. While many of the nine features cannot be applied by CANDID because they require additional traffic visibility than that afforded by NetFlow (e.g., packet size distributions), some features overlap with a CANDID strategy. In particular, the authors surmise that host behavior is a function of the peers it communicates with, and CANDID leverages this same intuition (see Chapter 7 for details).

[1] While NetFlow could present this information (for TCP traffic), it does not.

3.4 Host Grouping

Host grouping makes use of one or more host characteristics to group hosts together that are determined to be similar in some way. For example, a host grouping algorithm could first assess host roles using a technique described in the previous section, then group hosts together that are found to have the same role. The ability to group hosts together is relevant to CANDID because this notion serves as the basis for identifying load balanced assets (see Chapter 6 and Chapter 7 for details).

Aiello et al. present two techniques for grouping hosts together into communities without first performing any host behavioral classification [9]. Their work presents two strategies for forming communities of hosts (but does not map these communities to load balancing or another clustering model): *popularity*, wherein the percentage of hosts that interact with all of the target hosts in the grouping must exceed a certain threshold for the group to be valid, and *frequency*, wherein two hosts are grouped if they communicate with one another during small time periods. CANDID does make use of a notion similar to the *popularity* concept, with the intuition that load balanced systems will have peers in common over long time scales. In contrast, the *frequency* notion is not used; this is the case for two reasons: (i) there is not any fundamental principal which indicates that load balanced systems will frequently communicate with one another (as intra-cluster communication patterns may heavily depend on the clustering implementation), and (ii) load balanced systems are likely members of the same IP subnet, such that these communication patterns will not be represented in the NetFlow records (see the discussion of the NetFlow "blind spot" in Chapter 4 for an explanation).

A similar approach to the *popularity* notion from [9] is detailed in [58]. Here, the authors propose making use of peer relationships from connection patterns to define host groups. The work develops a similarity metric for a host pair by computing the intersection of peers contacted by the two hosts in the host pair. This approach is similar to a CANDID approach, wherein the Jaccard Index is used to compute peer similarity (see Chapter 7). However, the difference between these two strategies is that the Jaccard Index computation used by CANDID encompasses *both* the intersection *and* union of peers for a host pair, and not just the intersection. Incorporation of the union is important, since it better illuminates the true peer relationship for a pair of assets.

Finally, BLINC [36] brings together many of the same analysis ideas used in CANDID. In particular, BLINC presents a multifaceted approach to classify systems and their applications, leveraging *social behavior* (the peers hosts communicate with), *functional behavior* (whether a host acts as a server, client, or both), and *application behavior* (source and destination addresses and ports, transport protocol, and mean packet size) to complete its analysis. BLINC establishes host groups by using host addresses to locate bipartite cliques, wherein each member of a collection of hosts acting as clients establishes connections with members of a collection of hosts acting as servers. As previously mentioned, CANDID does make use of *social behavior* to establish groups of load balanced hosts, but simplifies the

problem by working with assets in a pairwise manner rather than with full communication graphs. Also, while CANDID targets the enterprise LAN context, BLINC focuses on performing analysis from traffic on WAN access links. Finally, while CANDID does make use of application properties (e.g., request sizes) to aid in load balancing inference, it does not follow BLINC's footsteps with regards to all *application behavior*, as mean packet sizes are not available with NetFlow records.

3.5 Flow Ranking

As first mentioned in Section 1.1, one of the goals of CANDID is to determine the importance of each asset and then subsequently rank them. While this thesis does not address this importance metric nor the ranking mechanism, it is still worth surveying previous research in this space to build context for this problem space. However, unlike the CANDID goal, previous work has focused on ranking flows rather than assets.

FlowRank [60] establishes causal flow dependencies by how closely in time flows appear to each other (as well as how frequently this occurs), and subsequently ranks flows by applying an algorithm based on the PageRank [45] algorithm to the flow dependencies. The proposed algorithm works by looking at links between nodes (i.e., dependencies between flows), each with associated weights (e.g., traffic volume), to highlight the most important flows (e.g., elephant flows or flows representative of security threats).

The authors of FlowRank take a similar but slightly different approach to tackling the same problem as FlowRank in [61]. Here, a simpler flow dependency model is developed, which is based on the flow chain notion presented in Chapter 5. Unlike FlowRank, the authors of this work propose using the HITS [39] algorithm to rank the inferred flow dependencies. In a nutshell, the HITS algorithm assess importance by evaluating two node metrics: the authority value (how many other nodes are dependent on it) and the hub value (how many other nodes it depends on). Both the PageRank and HITS algorithms seem to be effective at ranking the importance of *flows* via their dependencies, but neither of these existing approaches addresses ranking *assets*; FlowRank briefly suggests that accumulation of important flows could map to high-value assets, but does not discuss the idea in detail.

3.6 Application Dependency Analysis

A large expanse of existing literature has explored techniques to perform application dependency analysis in both distributed systems and in networks. While CANDID is only concerned with the networking context, this section explores related research in both of these areas.

	Host-Based	Network-Based
Active	ADD [15] X-Trace [30]	
Passive	Sherlock [11, 10] Macroscope [51] Orion [17] Constellation [12, 10]	Kind [38] & Caracaş [16] eXpose [33] Dechouniotis [23] NSDMiner [43] Peddycord [49] **CANDID**

Table 3.1: Taxonomy of existing network dependency literature.

Distributed Systems

From a distributed systems context, work from Aguilera et al. [8] explores dependencies in distributed applications by analyzing RPC behavior. In particular, the authors present two strategies for identifying these dependencies: the presence of nested RPC calls (where one call depends on a complete request-response pair of a second RPC call in order to complete) and the cross-correlation of delays between RPC calls (as the time delay between dependent RPC calls is statistically significant as compared to the delay between RPC calls that are not related). The notion of nested calls is explored in a networking context in both NSDMiner [43], which is discussed later in this section, and in CANDID (see Chapter 5).

Network Applications

There are two major factors that are typically used to categorize tools which analyze network asset dependencies. The first factor is whether the approach is *active* or *passive*, that is, whether or not it generates network traffic and interacts with the network in some way in order to complete its analysis. The second factor is whether the approach is *host-based* or *network-based*. Host-based systems require instrumentation of each end-host (which can include application modification), and consequently, have visibility into information that network-based approaches do not. However, host-based approaches require greater deployment effort since modifications must be made at each end-host, whereas network-based approaches require deployment at only one (or a few) vantage points. Note that many host-based solutions still make use of a centralized service to aggregate data and to aid in the analysis process, and are not necessarily entirely distributed in operation. Table 3.1 shows this taxonomy.

CANDID is a *passive*, *network-based* solution. By only relying on the passive collection of NetFlow records from within the network, CANDID has a simple and feasible deployment strategy. While NetFlow records do not provide per-packet granularity, CANDID will show that lightweight flow-level data still illuminates network behavior and provides visibility into

relationships inside the network. The remainder of this section highlights the approaches and contributions of existing work outlined in Table 3.1 and details how they relate to CANDID.

Active Approaches

ADD [15] takes an active stance in determining application dependencies by perturbing systems with workloads of various intensities, and using application instrumentation to collect performance and availability metrics. Then, ADD uses a statistical model that relates the collected metrics to the level of perturbation to assess dependency relationships. While the design of ADD is intuitive, it has several potential limitations: (i) active perturbation can disrupt a production system (or, if a system is tested in isolation, it may not be representative of a true deployment), (ii) the accuracy of results is heavily dependent on monitoring all applications involved (if an application is not instrumented for monitoring, results may miss dependencies), and (iii) as an active solution, all endpoint systems must be instrumented for accuracy of results.

X-Trace [30] uncovers application dependencies by instrumenting the datapath. More specifically, X-Trace requires modifications to applications to tag requests with special metadata to allow them to be traced throughout the request-response lifecycle, thereby illuminating dependencies. While X-Trace's approach can be comprehensive, it is only successful if the entire datapath is instrumented [2] to tag requests.

Passive Approaches – Host-Based

Sherlock [11, 10] leverages packet capture running at each end-host to infer dependencies. Sherlock constructs a directed dependency graph by modeling component states (up, troubled, down) and the conditional probability of observing connections in a fixed dependency interval. Unfortunately, Sherlock has some drawbacks: (i) it only tackles remote-remote (RR) dependencies (and not local-remote (LR) dependencies), (ii) traceroute is required to determine the network topology, (iii) the dependency discovery algorithm uses a fixed time window when computing conditional probabilities of related connections (which can lead to a solution that lacks generality, hindering utility on traffic data from a wide range of environments), and (iv) while Sherlock's model can represent load balancing and failover relationships, the algorithm requires manual human input to indicate where/how clustering is used (i.e., inference of cluster activities is not automatic). While CANDID also uses a fixed interval (see Chapter 5), it does not require topology information and it shows great promise for automatically inferring instances of load balancing (see Chapters 6 and 7).

Macroscope [51], like Sherlock, makes use of end-host packet capture to complete dependency analysis. However, unlike Sherlock, Macroscope also collects connection tables from

[2] Interestingly, one could argue that the action of manually instrumenting the datapath is a way of documenting a subset of an application's dependencies, particularly those that are static and fundamental to the application's operation.

end-hosts to associate ongoing flows (from packet capture) with applications via process IDs, avoiding the shortcoming that port numbers do not have a one-to-one mapping with applications. Macroscope then uses these mappings as the first step towards uncovering application dependencies on network services. However, to account for ephemeral port usage, Macroscope uses the frequencies with which users and applications do and do not access services to further assess dependencies, with the intuition that repeated access of a service by an application from several users is a stronger indication of a dependency. While CANDID does not have access to application-specific behavior, I do think this is an excellent approach for overcoming potential effects of the one-to-one service/port mapping problem. It is worth noting, however, that CANDID avoids some concerns surrounding ephemeral source port usage by excluding the source port from the 3-tuple that defines an asset (Section 2.2).

Orion [17], which defined the remote-remote and local-remote dependency terms, is similar to [8] in that it leverages delay distributions to assess dependencies. Orion builds up probability distributions for delays in start times of flows, and looks for peaks in these delay distributions that would be indicative of a flow relationship. Unfortunately, Orion suffers from issues that overlap with Sherlock: (i) it contains several fixed constants (e.g., window size, minimum flow count) that are selected without data calibration, which might lead to a solution that lacks generality and robustness, and (ii) while it can group hosts for load balancing and failover, it requires configuration-specific out-of-band information (e.g., DNS naming patterns or manual human input) to assess cluster membership.

Constellation [12, 10] is similar to Orion — it infers dependencies by using the difference between flow start times. However, Constellation is different in two major ways: (i) it uses expectation maximization fitting to map observed delays to a set of models that indicate whether one set of packets has caused another set of packets to occur, and (ii) it runs in a fully-distributed fashion, not requiring a centralized server to extract dependencies from traffic collected by endpoints.

Passive Approaches – Network-Based

Both [38] and [16] take a similar approach to inferring network service dependencies. In both projects, the general idea is to look for flow correlations, wherein not only do two flows start closely-together in time, but also do so with great frequency. It can be difficult to perform this style of analysis when many flows are overlapping; in [16], the authors outline a rough strategy for working around this challenge, but indicate the complete solution will be published in a full paper at a future point in time.

eXpose [33] uncovers dependencies by finding flow groups that co-occur in windows of time with high frequency, and evaluates these co-occurrences using the JMeasure (a metric from the data-mining community used to quantify mutual information from two sources; i.e., the lack of independence between the two sources). While the eXpose authors present a strategy based on packet capture, eXpose will work correctly with NetFlow, lending itself nicely to a passive network-based solution. A potential downside of eXpose is that, like

Sherlock, it defines the fixed-window size as a constant, which can lead to a brittle solution. While CANDID takes an alternate strategy to understand asset dependencies, it does make use of the Jaccard Index (another similarity measure) to help illuminate instances of load balancing.

In [23], Dechouniotis et al. present a solution that identifies both RR and LR dependencies. Their approach maps basic flow attributes (source and destination addresses) to the logical definitions of local-remote and remote-remote dependencies. Then, rather than relying on the frequency of observations (like eXpose), their solution uses a collection of four confidence variables (that evaluate the relationship between the observation of a flow and any associated events) combined with timestamps along with a genetic algorithm to determine dependencies. A downside of this approach is that the genetic algorithm requires training data to be effective, which limits its usability as a general-purpose solution.

NSDMiner [43] illuminates LR dependencies by locating nested connections, wherein one complete request-response pair starts and completes between the request and response of another connection (matching the expected recursive connection pattern). The proposed algorithm accomplishes this by passively monitoring network traffic and processing flows in a chronological order, looking for matches of the source IP of a new flow with previous flows destined for that same IP address. This approach is similar to both [8] and CANDID (see Chapter 5). While NSDMiner is effective at tackling this one area of the network dependencies problem space, it does not determine RR dependencies, nor does it provide a technique for automatically detecting clusters.

Peddycord et al. present three improvements to NSDMiner in [49]. In the original NSDMiner solution, weighted directed graphs were constructed to assess the confidence of dependencies, where graph nodes represented services (weighted by the total number of accesses to the service) and graph edges represented service dependencies (weighted by the number of times one service accessed another while being used). This original strategy then computed ratios of edge to node weights to assess confidence in the discovered dependencies; the first improvement in [49] is that logarithmic weighting, where the base of the computed logarithm is the node weight, is used to compute these confidence metrics in lieu of simple ratios. This improves dependency inference accuracy, since it better accounts for the magnitude of graph weights. CANDID does not make use of logarithmic weighting, however, this general idea is worth considering in future work. The second improvement attempts to find dependencies that NSDMiner fails to find due to insufficient traffic, since infrequent network observations lead to weights of small magnitude in the weighted graph, which in turn leads to small confidence metrics. The proposed improvement boosts the confidence of dependencies of an infrequently-used service by seeing whether a similar set of hosts are involved in another well-known dependency. This style of inference may have potential benefits in overcoming the NetFlow "blind spot" (see Chapter 4), but is unexplored in this thesis. The third and final improvement is a basic attempt at identifying clusters by looking for equal probability in accessing one of several systems providing a service dependency. More specifically, the technique evaluates the computed dependency graph to locate services (i.e., graph nodes)

providing service on the same port number with roughly equal use by other services in the dependency graph (as indicated by graph edges). This approach is somewhat limited: not only does it require dependency analysis (the load balancing detection strategy presented in this thesis works independent of dependencies), but it also assumes that the client request workload is distributed evenly across all members of a load-balanced cluster (the solutions I present in thesis do not make that assumption). Given these limitations, I still consider the problem of performing passive general-purpose load balancing identification unsolved. Finally, while these additions do improve upon NSDMiner, Peddycord et al.'s solution still does not detect remote-remote dependencies.

3.7 Summary

This chapter has presented a wide array of existing literature that has explored a variety of problems and solutions that share one or more similarities with CANDID. In particular, the reviewed literature supports the solutions presented in subsequent chapters of this thesis by showing how similar flow and host characteristics can be leveraged to assess host and application behavior and relationships through passive traffic classification. Nonetheless, it is important to remember that none of this existing research has made use of role identification in combination with host grouping to specifically identify cases of load balancing, which is a key contribution of this thesis.

Chapter 4

Working with NetFlow

NetFlow is a standardized and frequently-used protocol and data format for IP traffic monitoring that was developed by Cisco Systems [1, 56, 22, 21]. This chapter both provides NetFlow context for readers unfamiliar with the technology and details several data sanitization and processing phases performed by CANDID that are required for subsequent asset relationship analysis. Further, this chapter serves to introduce the Lawrence Berkeley National Lab dataset used for the case study that is presented in subsequent chapters.

4.1 NetFlow Essentials

NetFlow is typically deployed as follows:

1. A set of network appliances, placed at various vantage points in a network, act as NetFlow *probes*, aggregating packets (oftentimes sampled [27], rather than all packets) into L4 flows, each of which is represented by a NetFlow *record*.

2. Once a flow has completed (e.g., TCP FIN flag is observed, no packets for the flow occur for a duration set by a timeout parameter, or the flow is evicted to make room in memory for sampling another flow), the probe transmits a UDP packet with the record to a centralized NetFlow *collector*.

   It is worth acknowledging that since UDP does not guarantee reliable delivery, some flow records may be lost during transmission. However, given that NetFlow often samples packets (and thus, may not reflect all flows), lack of complete flow visibility is already an acknowledged limitation of this measurement approach. Nonetheless, some NetFlow implementations [18, 5, 19] support the transmission of flow records to the collector via reliable SCTP [6].

3. The accumulated data is written to permanent stable storage by the collector.

4. Analysis tools, such as CANDID, make use of stored flows.

It is worth emphasizing that each NetFlow record represents a ***unidirectional*** flow. Mapping these unidirectional flows into bidirectional connections is a challenge that is addressed later in this chapter. Each NetFlow record contains basic aggregated statistics for an observed flow. While NetFlow record fields vary across protocol versions, the fields present in the NetFlow records used in this work are:

- Flow start & end timestamps
- Ingress & egress interface IDs (particularly relevant when the NetFlow *probe* is a router with many interfaces)
- Source & destination IP addresses and port numbers
- The L4 transport protocol (e.g., 6 for TCP)
- The union of all observed flags
- Total packet count and total (including headers) byte count [1]

4.2 NetFlow Benefits

As compared to other network measurement data collection techniques (i.e., packet headers or full-payload packet traces), NetFlow provides several clear advantages:

Widespread Support NetFlow [1, 56, 22, 21] and similar flow monitoring and reporting implementations (e.g., JFlow [32], NetStream [31], sFlow [50, 57], and IPFIX [54]) are popular protocols supported on a wide array of network devices, including switches, routers, middleboxes (e.g., [24]), and even end-hosts (e.g., [4] and [3]).

The fact that NetFlow is widely supported is particularly important, since it means most enterprises already have the infrastructure necessary to use CANDID. In contrast, enterprises that wish to deploy solutions that leverage packet capture and per-packet analysis oftentimes must purchase and deploy new specialized infrastructure specifically dedicated to this task (rather than simply "turning on" NetFlow in existing network components).

Privacy Since NetFlow provides flow statistics in aggregate, rather than per-packet details (e.g., sizes and timing) and packet contents, it minimizes exposure of information about users of the network and software details [40, 63].

Resource Footprint NetFlow requires a far smaller resource footprint (both for the capture process and for storage and subsequent processing) since it reports flow information in aggregate and because it is oftentimes sampled [27].

[1] These counts are most meaningful for TCP flows, where flow boundaries are well-defined.

Structure NetFlow records can be stored as well-structured lines in an ASCII text file, with each delimited record field as a member of a column. This makes NetFlow records easy to parse and process.

4.3 NetFlow Challenges

Unfortunately, even in light of the benefits outlined in the previous section, NetFlow has several limitations and poses several challenges:

Low Fidelity Leveraging an existing well-deployed traffic measurement technology and minimizing user privacy exposure through aggregated flow data does come at the cost of lower fidelity of measurement data. This lack of visibility into complete network behavior does pose some challenges: not only does it mask characteristics of individual packets in a flow (e.g., the distribution of packet sizes), it also masks the presence of multiple request-response pairs that might exist between two communicating hosts. For example, if communication between a web server and a database server is done with a pool of persistent connections, or if communication between a web client and web server is achieved with a single connection leveraging HTTP Keep-Alive [29], NetFlow will not capture the individual communication instances between the pair of hosts, and instead will simply provide start and end timestamps for the overarching flow. Both of these cases make it difficult to infer a relationship among request-response pairs between the web client & web server and between the web server & database server.

"Blind Spot" Unlike detailed application logging or packet capture, which are frequently performed at end-hosts, NetFlow probes are often located at the network core. Consequently, flow records will only represent inter-subnet traffic, but not intra-subnet communication. Further, it is likely that not all switches and routers in a network are configured to emit NetFlow records, meaning that even some inter-subnet traffic may not be visible in traffic traces. So, while this problem takes on many forms, it can generally be thought of as NetFlow having a "blind spot".

As an example, imagine a set of four assets spread across three subnets that are communicating. This network topology is depicted in Figure 4.1. In this scenario, Asset #1 in Subnet A needs to communicate with Asset #2 in Subnet B, which causes Asset #2 to communicate with Asset #3 in Subnet B, which then causes Asset #3 to communicate with Asset #4 in Subnet C. Ideally, a dependency analysis tool such as CANDID *should* flag this entire sequence of connections as a series of dependencies [2], but since Asset #2 and Asset #3 are members of the same subnet, the monitoring router does *not* observe any of their communication (though the router does observe

[2] Admittedly, the introduction of "chained" connections here is premature – see Chapter 5 for the complete design details.

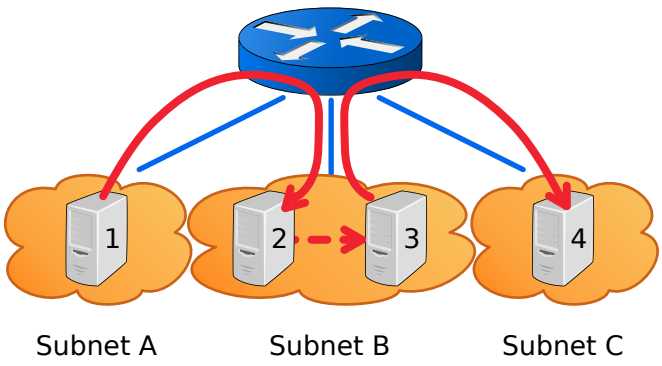

Figure 4.1: NetFlow "blind spot" traffic diagram, where intra-subnet communication in Subnet B between Assets #2 and #3 is not observed.

communication between Assets #1 and #2 and between Assets #3 and #4). Thus, in this case, communication between Assets #2 and #3 is unfortunately hidden in the NetFlow "blind spot".

Unidirectional NetFlow records represent unidirectional flows. While mapping unidirectional flows into bidirectional connections for analysis purposes is a challenge, unidirectional flow monitoring can also pose another issue. Network routes are known to be asymmetric [47], so a single vantage point may only capture one-half of a connection with a single unidirectional flow. While asymmetric routing tends to occur on the WAN, it can still occur on a LAN, so it is important to keep this issue in mind. These unidirectional flow visibility limitations represent another type of "blind spot".

Sampling As previously mentioned, NetFlow records are often derived from sampled packets, rather than all packets. Sampling is done to reduce the workload on the network probe. While NetFlow does have a smaller resource footprint than packet capture, routers and switches are simply not designed to monitor all traffic.

Although these limitations and challenges present several hurdles that make analysis difficult, CANDID resolves several of these issues and does yield promising results.

4.4 LBNL Dataset Overview

Subsequent chapters of this thesis present several strategies for inferring host relationships as steps towards solving all of the CANDID goals. A case study using real network data from Lawrence Berkeley National Lab (LBNL) is presented to evaluate these strategies. I believe the traffic from this large research institute is representative of a typical enterprise

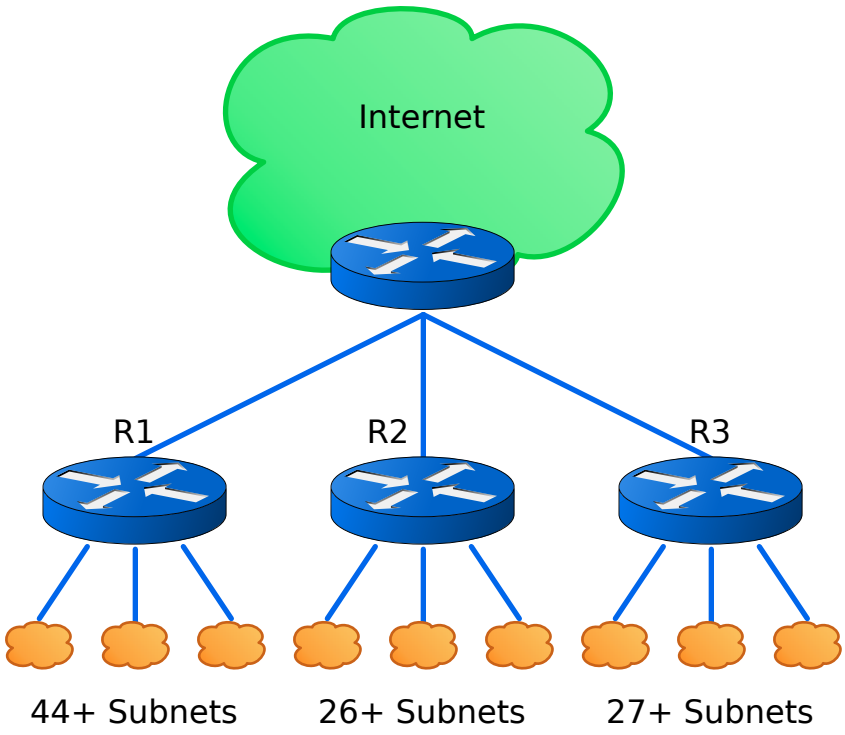

Figure 4.2: LBNL network topology and NetFlow vantage points.

network. The remainder of this section describes the LBNL data collection methodology and gives a summary of the specific dataset used in this work.

Traffic Collection Methodology

Network traffic at LBNL is recorded by three main border routers (labeled R1, R2, and R3) as shown in Figure 4.2. This specific configuration has three vantage points acting both as NetFlow probes and as collectors, such that NetFlow records are stored in three separate files, one for each router. Further, this topology leads to records for each router that include both communication with external hosts, as well as inter-subnet communication that may cross more than one router (which leads to duplicate flow entries from multiple routers observing the same flow). Finally, while NetFlow data is typically sampled due to control-plane overhead [27], the datasets used by CANDID are not sampled.

	June 1, 2011	June 2011
# of Flows	1,156,826	32,177,513
Data Transferred	38.19 GiB	916.98 GiB
# of Hosts	106,095	780,262
Internal	56,475 (53.23 %)	129,842 (16.64 %)
External	49,620 (46.77 %)	650,420 (83.36 %)

Table 4.1: General statistics of LBNL SMTP NetFlow data prior to conversion into bidirectional connections and data sanitization.

Dataset Summary

The data provided by LBNL represents all SMTP [52] traffic (records for TCP flows with source or destination port 25) for the month of June 2011. Flow records are broken up into files representing each day of the month, and in many cases, evaluations in this thesis focus on a single 24-hour time period: June 1, 2011. So, unless otherwise noted, readers may assume that any data presented in this thesis is that of June 1, 2011 only. Table 4.1 summarizes the flow data for both June 1, 2011 and the entire month of June 2011. The larger fraction of external hosts present in the month's span of flow records is likely due to a wider range of external domains participating in electronic mail exchange as compared to during the 24-hour period.

In this dataset, all source and destination IP addresses have been anonymized (the anonymization process is consistent across the source files from all three routers). Each IP address is represented as either `addr-intern-1234` (for internal hosts) or `addr-extern-1234` (for external hosts), where `1234` is a unique integer for each observed internal or external host. As a shorthand, future host references in this thesis will be expressed as `i1234` or `e1234`.

Readers may be troubled that provided traffic records focus on a single application protocol. However, since this work is just the first step of many towards solving all of the CANDID project goals, the choice to focus on a single application protocol helps keep the dataset manageable and reduces the level of human intervention needed with LBNL staff for establishing ground truth and validating results.

4.5 Data Handling and Sanitization

To overcome some of the challenges enumerated in Section 4.3 and to account for anomalies that are found in empirical data (e.g., invalid flow timestamps), CANDID performs several data transformation tasks prior to executing asset dependency or relationship analysis. Figure 4.3 provides a graphical overview of this data handling pipeline, and Table 4.2 (at the

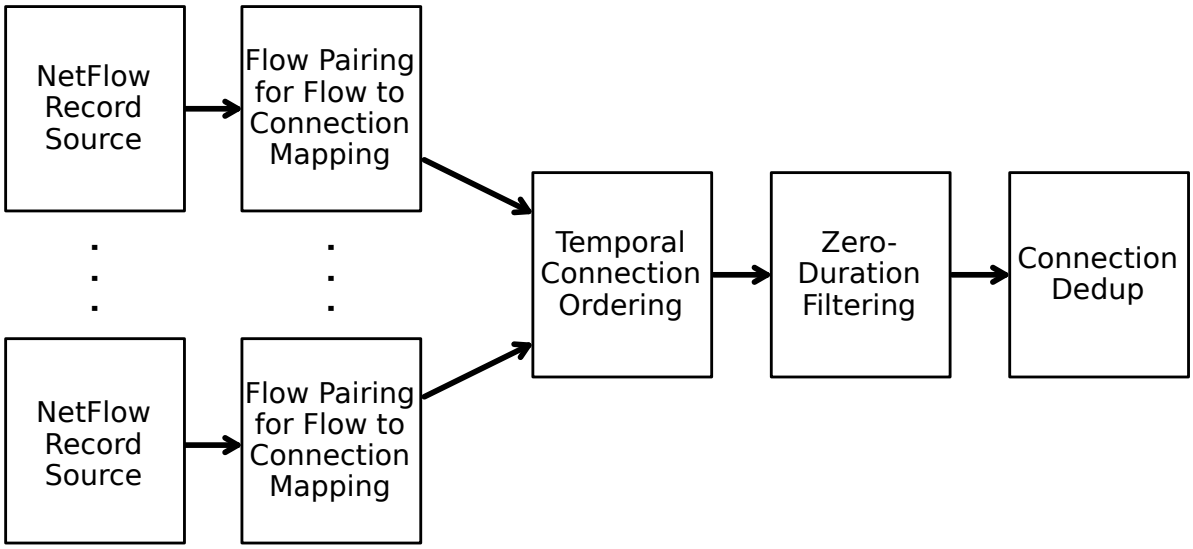

Figure 4.3: CANDID data handling and sanitization pipeline

end of this chapter on page 35) summarizes statistics of the dataset to illustrate the consequences of each data handling phase to ensure the empirical data is not being inappropriately treated (e.g., filtered too aggressively due to an incorrect assumption). The data processing and sanitization tasks are enumerated and explained in the remainder of this section (*note that some of the phases illustrated in Figure 4.3 encompass multiple tasks from this listing*).

Unproductive Connection Filtering

As a part of the flow pairing phase (described further in the next task), unproductive flows are discarded. An unproductive flow is one in which no meaningful IP-level communication occurs between the client and server.

Since the LBNL flow data used for the case study in this thesis is entirely composed of TCP traffic, a rather simplistic, yet robust, heuristic for identifying unproductive connections has been used: a productive connection must have at least two packets in each direction [3]. This approach was used in lieu of the NetFlow flags field, because while this field was present in the LBNL NetFlow records, it was always 0, and thus could not be used for this purpose.

A commonplace example in which unproductive communication occurs is when a system scans an entire subnet, looking for the presence of a service on a port or range of ports on each host. In this example, since many hosts will not provide such a service (i.e., it is unlikely

[3]This is derived from the notion that meaningful TCP communication requires connection establishment via the three-way TCP handshaking process (composed of the SYN, SYN/ACK, and ACK) followed by data transfer.

that every user's workstation is running a SMTP server), then flow records will show the scanning system's rejected connection attempts with each end-host. This is clearly evident in Table 4.2, where the first two columns reveal a dramatic drop in the number of servers from 60,176 down to 4,203.

Therefore, since these types of flows do not provide meaningful information about asset dependencies and relationships for current CANDID analysis strategies, CANDID discards these flows.

Unidirectional Flow Pairing

As noted previously in Section 4.1, NetFlow records only represent ***unidirectional*** communication. CANDID makes use of several flow characteristics in order to pair flows together such that these unidirectional flow pairs can be mapped into bidirectional *connections*. The criteria to pair flows is as follows:

1. Given a flow with start timestamp t_0 and 5-tuple ⟨L4 Protocol, Source IP Address, Source Port, Destination IP Address, Destination Port⟩, the set of candidate flows is composed of all flows with start times $t_1 > t_0$ and with mating 5-tuple ⟨L4 Protocol, Destination IP Address, Destination Port, Source IP Address, Source Port⟩ (*note that the source and destination have been swapped*).

2. Of all flows in the set of candidate flows, only those where $t_1 - t_0$ is less than the *match time threshold* [4] are eligible for pairing, where t_1 is the start time of the candidate flow.

3. Finally, of all eligible flows from the previous step, the flow pair whose $t_1 - t_0$ difference is smallest is selected for pairing. If the set of candidate flows from the first step is empty, then the flow is designated an *unpaired flow* and is discarded.

   Figure 4.4 shows empirical CDFs (ECDFs) of all candidate flow start time differences and matched flow start time differences. The figure reveals that the selected *match time threshold* is rather liberal with constructing flow pairings. A thorough study of false positives and the ideal value for this threshold is deferred for future work.

   If a matching candidate flow is found, then a connection is formed by combining data from the two flows to produce a unified set of properties. These connection properties and their derivations are:

   - Start timestamp, which is defined as the earlier of the start timestamps of the two composing flows.
   - End timestamp, which is defined as the later of the end timestamps of the two composing flows.

[4] This is a configurable constant that represents the maximum start time difference for a pair of flows. In this work, a value of 60 s was used.

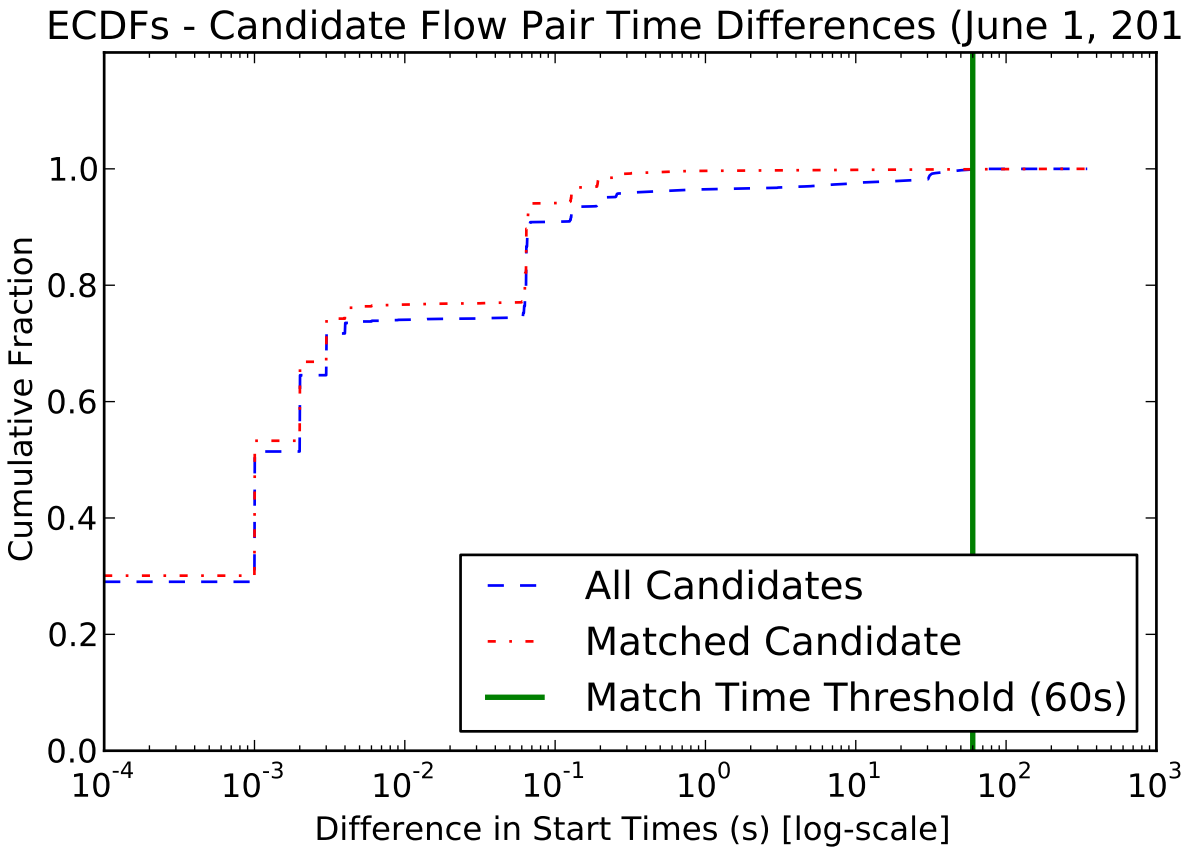

Figure 4.4: Empirical CDFs of flow start time differences ($t_1 - t_0$) for all candidates and flow pairings that are the best match. The selected 60 s *match time threshold* is also shown.

- Source interface, IP address, and port.
- Destination interface, IP address, and port.
- The L4 transport protocol.
- Source packet count and byte count (i.e., quantity of traffic sent by the source to the destination).
- Destination packet count and byte count.

Without unproductive flow filtering, a large number of *unpaired flows* can exist, since a flow may exist from a scanning system to each host on a subnet, but return flows from scanned hosts may not exist. Connections can still be constructed from these *unpaired flows* (this was done for the "All Connections" column of Table 4.2), but doing so provides little utility. Rather, what is useful is understanding the quantity of *unpaired flows* that exist *after* unproductive flows have been filtered out. In the case of the LBNL dataset, only a

small fraction (3.55 %) of all productive flows were *unpaired flows*. Several *unpaired flows* were manually inspected to ensure that the flow pairing algorithm had not made a mistake; the reviewed flows did in fact lack pairs for matching. A concrete cause for the presence of these *unpaired flows* is unclear and warrants further investigation in the future. These flows are not represented in the "Productive Connections" column (or any subsequent columns) in Table 4.2.

As depicted in Figure 4.3, each NetFlow record source (which may be from different vantage points or different time periods) is processed by the flow pairing data handling phase separately. One potential downside of performing flow pairing for each data source separately is that flow pairs along asymmetric paths will never be paired together to form a connection. However, based on the LBNL network topology depicted in Figure 4.2, asymmetric routes are not expected with the LBNL dataset. Exploring this design choice and the associated trade-offs is saved for future work.

Connection Directional Correction

A critical step of mapping pairs of flows into bidirectional connections is selecting which flow represents client→server communication and which flow represents server→client communication. While previous work [13] has indicated that assignment of client and server roles requires complex heuristics, the CANDID technique is based on two simple principles:

- Clients initiate network connections, and therefore, given a matched pair of flows, the flow with the earlier start timestamp represents client→server communication and the flow with the later start timestamp represents server→client communication.

- The LBNL data showed a significant number of connections (34.04 % of all productive connections) that upon human inspection appeared reversed. The exact cause of this is unclear — the timestamping process in network devices is known to be unreliable [48], but since connections are composed of two flows from the same network vantage point, it is unlikely that clock-related issues are the cause (unless each network interface has a separate clock). To correct for this direction problem, a frequent assumption regarding port numbers is made: ephemeral ports (≥ 1024) are associated with clients, while privileged ports (< 1024) are associated with servers. Thus, CANDID swaps the direction of any connection that is found to violate this assumption.

  Figure 4.5 uses a pair of ECDF curves to compare connection duration distributions for connections that did and did not require directional correction. While both ECDF curves have the same shape, connections with reversed directions typically have shorter durations than those with correct directions. However, beyond this minor difference, the ECDF comparison does not yield any striking properties that better illuminate causes of this behavior.

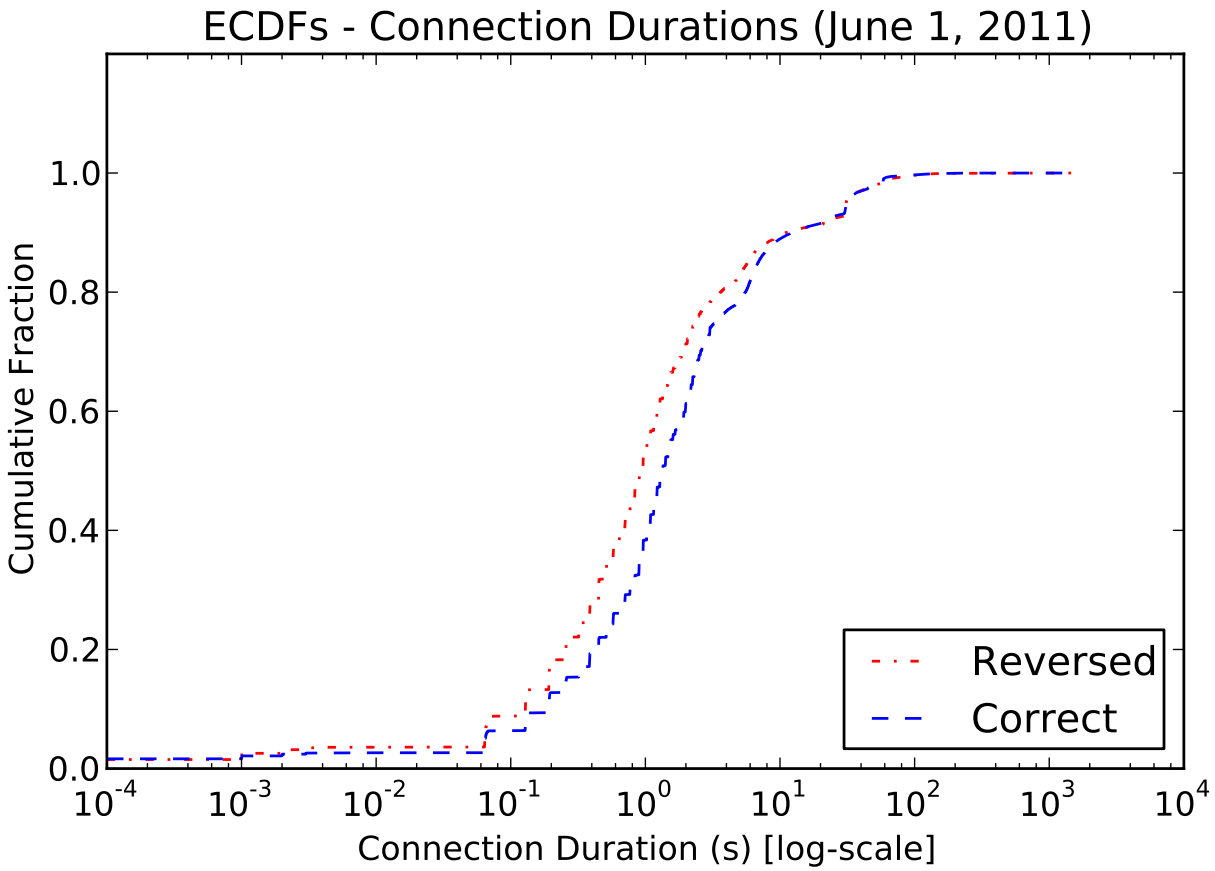

Figure 4.5: Empirical CDFs for connection duration distributions, comparing connections that require directional correction to those that were already correct.

This task completes the flow pairing phase, as flows have been mapped to connections and any connection direction errors have been corrected.

Connection Ordering

The purpose of this data handling stage is to combine connection records from separate NetFlow sources (e.g., multiple vantage points or windows of time) into a single temporally-ordered stream of connections. Fortunately, since the CANDID flow pairing phase ensures that the connections it produces are temporally-ordered by their start timestamps [5], this

[5]While unreliability of the timestamping process may yield minor ordering inconsistencies, the primary purpose of this processing phase is to bring together data from several time periods, so these inconsistencies are not of concern.

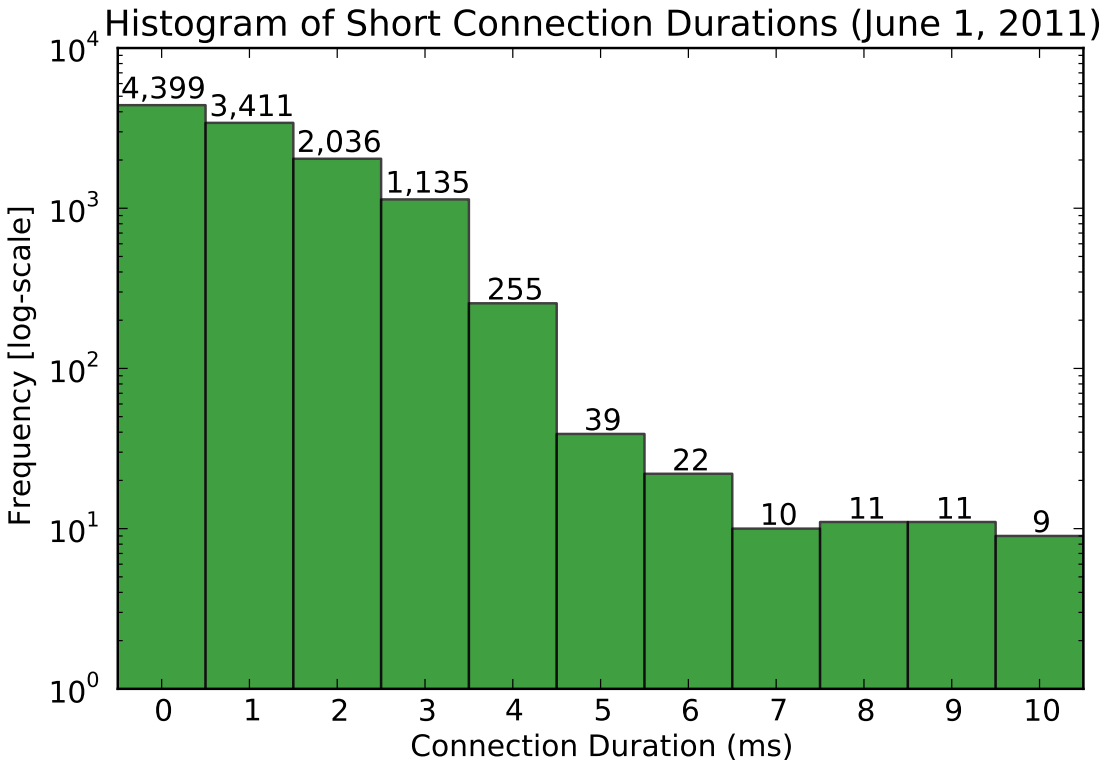

Figure 4.6: Histogram of connections with durations of 10 ms or less.

phase is quite simple: connection ordering checks the start timestamp of the next connection at each flow→connection mapper, and chooses the connection with the oldest timestamp.

Zero-Duration Filtering

The LBNL data revealed several connections (0.95 % of all productive connections) where the start and end timestamps are identical, yet an inconsequential quantity of data (i.e., hundreds to thousands of bytes) was transferred. Clearly, however, it is unreasonable to think that a connection of zero-duration can be established, let alone transfer any data.

As previously mentioned, clocks and the timestamping process in network devices have been known to be unreliable [48]. Therefore, it is reasonable to assume that such connections were derived from flows with incorrect timestamps. What is less, clear, however, is how to address these connections, and further, if other connections with extremely short lifetimes (e.g., 1 ms, 2 ms, . . .) exist, where to draw a boundary between acceptable and unacceptable connection lifetimes. This is particularly difficult to do, because while the speed of light does provide a lower bound on communication times, different enterprise network topologies

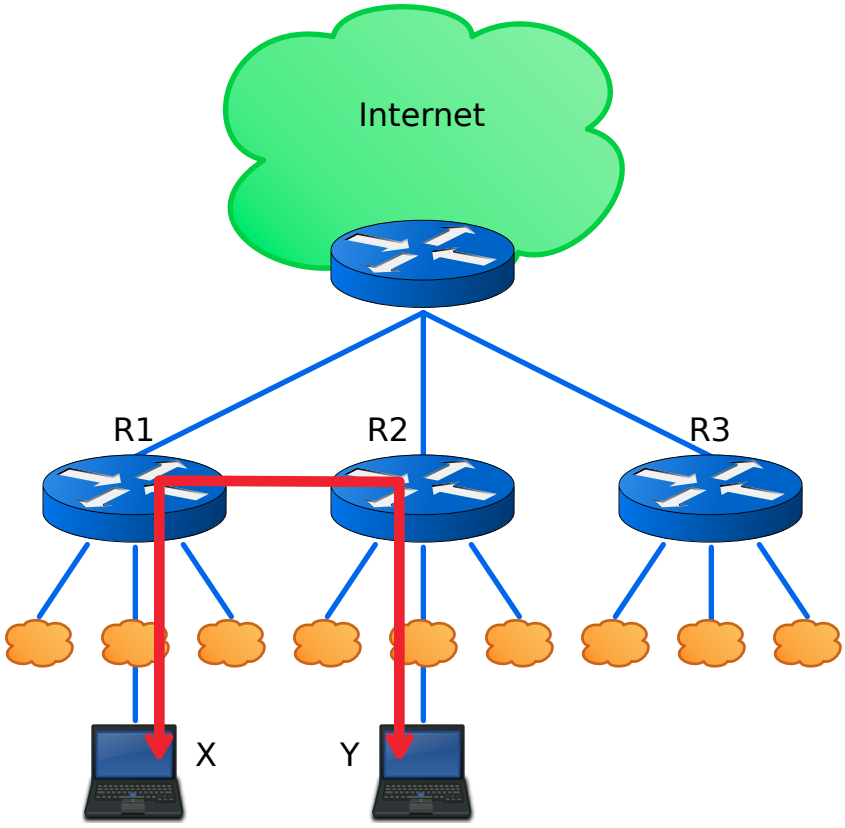

Figure 4.7: Flow duplication due to multiple vantage points along the same communication path.

and environments will have different minimum latency characteristics. In fact, Figure 4.6 provides a histogram of these short connections (those with duration ≤ 10 ms), and reveals that there are a significant number of connections with durations below 4 ms.

To simplify this problem space, the current CANDID implementation discards any connection with zero-duration, and does not apply any filtering to connections with extremely short lifetimes. Nonetheless, this issue is worth exploring further in future work.

Connection Deduplication

As illustrated in Figure 4.2, the LBNL network contains several NetFlow vantage points, and it is not unreasonable to assume that other enterprise network operators also perform traffic monitoring from multiple vantage points.

Consider the scenario depicted in Figure 4.7: a host `X` in a subnet attached to router `R1`

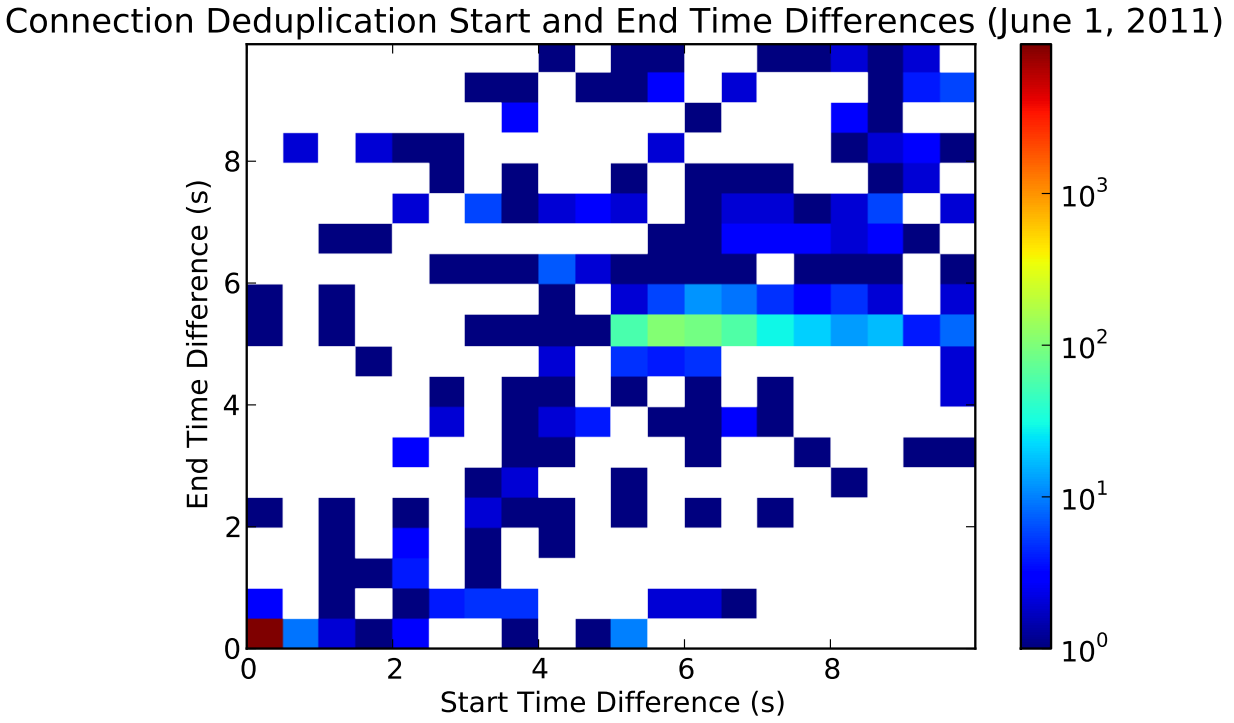

Figure 4.8: Heat map of start and end timestamp differences for candidate duplicate connections, with differences of 10 s or less.

communicates with a host `Y` in a subnet attached to router `R2`. Regardless of whether or not routers `R1` and `R2` are directly connected or communicate via a common peer, network traffic between hosts `X` and `Y` must pass through both routers `R1` and `R2`, and thus, both routers will produce flow records (and thus, connections) for this communication.

From the example scenario, it is easy to see how networks with multiple vantage points can lead to duplicate connection data, and the purpose of this connection deduplication data handling stage is to remove these duplicates. CANDID deems two connections duplicates of one another if all of the following criteria are met:

- Their 5-tuples ⟨L4 Protocol, Source IP Address, Source Port, Destination IP Address, Destination Port⟩ are identical.
- The difference in their start timestamps is less than or equal to the *similarity threshold* [6].

[6]This is a configurable constant that represents the maximum difference in connection timestamps that indicate connections are similar. In this work, a value of 500 ms was used.

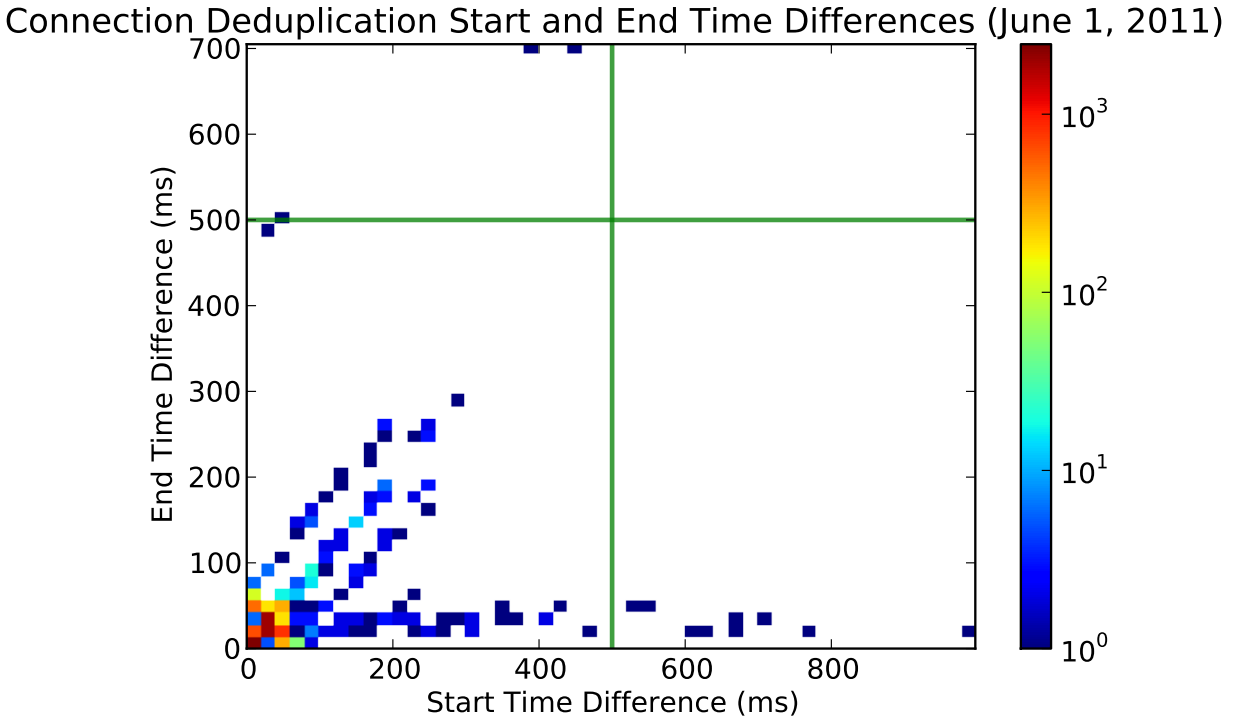

Figure 4.9: Heat map of start and end timestamp differences for candidate duplicate connections, with differences of 1 s or less. The 500 ms *similarity threshold* is marked by the crossing green lines.

- The difference in their end timestamps is less than or equal to the same *similarity threshold*.

Figures 4.8 and 4.9 provide heat maps showing the differences in start and end times for connections deemed similar by the aforementioned 5-tuple. Figure 4.8 includes connections with timestamp differences up to 10 s, while Figure 4.9 provides greater detail around the selected *similarity threshold* by only including connections with timestamp differences up to 1 s. The two heat maps clearly show that the majority of duplicate connection candidates have negligible timestamp differences, which shows that selection of 500 ms for the *similarity threshold* is a satisfactory choice.

Should CANDID identify two connections that are duplicates of one another, it discards one of them. The impacts of deduplication are shown in the last column of Table 4.2. Connection deduplication is the last CANDID data handling phase – connections that have satisfied all sanitization requirements are passed on to the asset relationship and dependency analysis components described in subsequent chapters.

	All Connections	Productive Connections	Non-Zero Duration	Unique Connections
# of Connections	679,526	463,953	459,554	449,909
Data Transferred	38.19 GiB	30.75 GiB	30.73 GiB	30.41 GiB
Per Connection (mean)				
Packets	84.95	104.59	105.43	106.40
Request Data	56.68 KiB	66.55 KiB	67.17 KiB	67.89 KiB
Response Data	2.24 KiB	2.93 KiB	2.96 KiB	2.99 KiB
Duration	4.05 s	3.82 s	3.85 s	3.93 s
Per Connection (median)				
Packets	10.0	32.0	32.0	33.0
Request Data	326 bytes	3.63 KiB	3.70 KiB	3.81 KiB
Response Data	272 bytes	1,012 bytes	1,020 bytes	1.00 KiB
Duration	0.71 s	1.15 s	1.15 s	1.21 s
# of Hosts	106,095	48,479	48,460	48,460
Internal	56,475 (53.23 %)	897 (1.85 %)	887 (1.83 %)	887 (1.83 %)
External	49,620 (46.77 %)	47,582 (98.15 %)	47,573 (98.17 %)	47,573 (98.17 %)
Client (only)	44,007 (41.48 %)	43,149 (89.01 %)	43,135 (89.01 %)	43,135 (89.01 %)
Server (only)	60,176 (56.72 %)	4,203 (8.67 %)	4,200 (8.67 %)	4,200 (8.67 %)
Client & Server	1,912 (1.80 %)	1,127 (2.32 %)	1,125 (2.32 %)	1,125 (2.32 %)

Table 4.2: General connection statistics of LBNL SMTP traffic after each data processing stage.

Chapter 5

Finding Dependencies via Connection Chains

This is the first of several chapters that presents the various CANDID analysis strategies. Each of these chapters is structured with the same three key components: (i) the intuition behind the strategy is introduced, (ii) details of the design are provided, and (iii) an evaluation, focused on the LBNL case study, is performed.

When this research venture first began, my initial goal was to tackle dependency inference, focusing only on local-remote (LR) dependencies. The basic idea for this approach is to use the element of time to establish causal relationships among connections.

5.1 Strategy Intuition

To better understand the intuition behind the dependency analysis approach detailed in this chapter, it is best to first recall the definition of LR dependencies by referring to the rightmost scenario depicted in Figure 2.1. In that example, the inferred relationship is: *the web server depends on the database server*, and the reason is because the web server needs a response from the database server in order to respond to the client.

The aforementioned example leads directly into the intuition behind this analysis strategy: a LR dependency will exist between two hosts if a pair of connections is observed to occur closely together in time with a shared "middle" host (e.g., the web server in the example), and the strength of that LR dependency is related to the frequency with which that observation is made. This intuition and related terms are formalized in the next section.

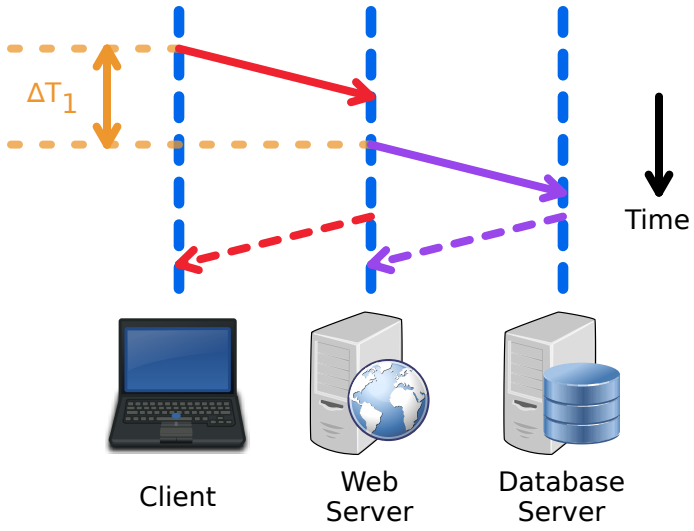

Figure 5.1: Connection chain of length 2 with a LR dependency.

5.2 Design Details

Inferring Causality

CANDID establishes the notion of causality for inferring LR dependencies though the idea of *connection chains*. Here, a *connection chain* is defined as a set of two or more temporally-ordered connections, wherein the destination host of one connection is the source host of the next connection. Further, the difference in start times of each pair of adjacent connections in the overall chain must be no greater than the time bound ΔT_1 [1]. Note, however, that this definition does *not* apply any constraints to the relative ending times of adjacent connections in the chain. The primary motivation for this decision stems from the fact that flow-level measurement data does not reflect individual request-response pairs, and moreover, that flow end times can be incorrect since connection termination is not the only mechanism for setting the flow end timestamp (e.g., eviction from the flow cache; see Section 4.1 for details).

As a simple, yet concrete, example of a connection chain, see Figure 5.1, which illustrates a connection chain of length 2. In this example, the client establishes a connection to the web server. At a later point in time, but within the time bound of ΔT_1, the same web server establishes a connection to the database server. From this observation, CANDID would identify a causal relation between the two connections and, thus, would infer that "the web server depends on the database server" (and, by extension, that "the client depends on the database server"). As previously noted, the timing and ordering of responses from the web

[1] A discussion regarding the selection of ΔT_1 and its effect on results follows later in this section.

server to the client and from the database server to the web server are unimportant, and thus, are shown as dashed lines (i.e., they may be temporally-ordered differently than as illustrated).

As chains are identified, CANDID tracks the number of times each chain occurs. The final output from CANDID for this analysis stage is a sorted list of all chains observed, with the chain that occurs most frequently at the top of the list.

Comparison to Existing Approaches

Previous literature, such as NSDMiner [43] and others (e.g., [49] and [8]), all use a similar idea of locating connection chains to perform dependency analysis. However, those approaches all rely on seeing fully-nested connections, wherein the request and response of one connection must occur after the start *and* before the end of the previous connection in the chain (and, more stringently, must occur before the response of the preceding connection in the chain). While the tighter bound of nested connections does indeed better match the intuition outlined at the start of this chapter, flow data can mask individual request-response pairs, and so CANDID introduces the modified variant of the technique which avoids the fully-nested requirement.

Selecting ΔT_1

Astute readers may be concerned with the selection of the ΔT_1 constant. Without a doubt, it is difficult, if not impossible, to arbitrarily (or intentionally) assign a single "one size fits all" value to a constant of this nature, where it would need to apply to all types of traffic for all applications in all enterprise networks. Making such a design choice can lead to a brittle solution that will inevitably fail to correctly infer asset relationships for cases where timing characteristics are at odds with the selected value for ΔT_1, leading to false positives and/or false negatives.

With this in context, a value of $\Delta T_1 = 5\,\mathrm{s}$ was selected for this work. The thought behind selecting this value is as follows: the value must be large enough to capture the latency of the enterprise network combined with application response time, but not so large that it casts too wide of a net such that CANDID infers nonexistent causal relationships. Ideally, however, it would be best to calibrate this constant through some means (e.g., evaluation of enterprise traffic with known causal connection chains).

5.3 Evaluation

The causal connection chain analysis technique described in the previous section was implemented and the June 1, 2011 SMTP dataset from LBNL was used to evaluate this strategy.

Case Study

At the start of the case study, the intention was to identify the entire LBNL incoming email processing chain, with the notion that there are several assets involved with receiving incoming messages, applying various stages of filtering (e.g., anti-spam and anti-virus), and then storage at the user's mail server. Unfortunately, after struggling to find evidence of these connection chains, LBNL operations staff confirmed that the majority of this incoming mail processing occurs on assets that are *all members of the same network subnet*. Therefore, due to the NetFlow "blind spot" (introduced in Section 4.3), CANDID is unable to illuminate these causal relationships.

Nonetheless, this strategy did reveal rather interesting information regarding the roles of hosts that appeared in the top six connection chains. Of all 1,967 connection chains, 539 unique chains were identified, and the top six connection chains represent 33 % of all observed chains:

- `i5`→`i63`→`i1` (8.8 % of all chains)
- `i3`→`i63`→`i1` (7.5 % of all chains)
- `i5`→`i25`→`e158` (4.8 % of all chains)
- `i3`→`i25`→`e158` (4.6 % of all chains)
- `i5`→`i35`→`i1` (3.8 % of all chains)
- `i3`→`i35`→`i1` (3.5 % of all chains)

These chains are visually illustrated in Figure 5.2. Observing that these six chains appear to come in pairs that share a common suffix and have similar frequencies gives a clue that the hosts `i3` and `i5` may be load balanced. While the purpose of the chaining strategy presented in this chapter is not to find load balanced hosts, the load balancing analysis strategies presented in Chapters 6 and 7 will show that hosts `i3` and `i5` are in fact load balanced.

Strengths

The greatest strength of this causal connection chain strategy is that it captures LR dependencies without applying constraints to flow end timestamps, which can mask individual request-response pairs or be unreliable. It would be relatively simple to modify this strategy to infer RR (remote-remote) dependencies by changing the common host requirement to apply to the source host of both connections in a pair (rather than the destination of one and the source of the other). However, due to several limitations with this general strategy (particularly concern for significant false positives or false negatives due to the ΔT_1 constant), this was not pursued further.

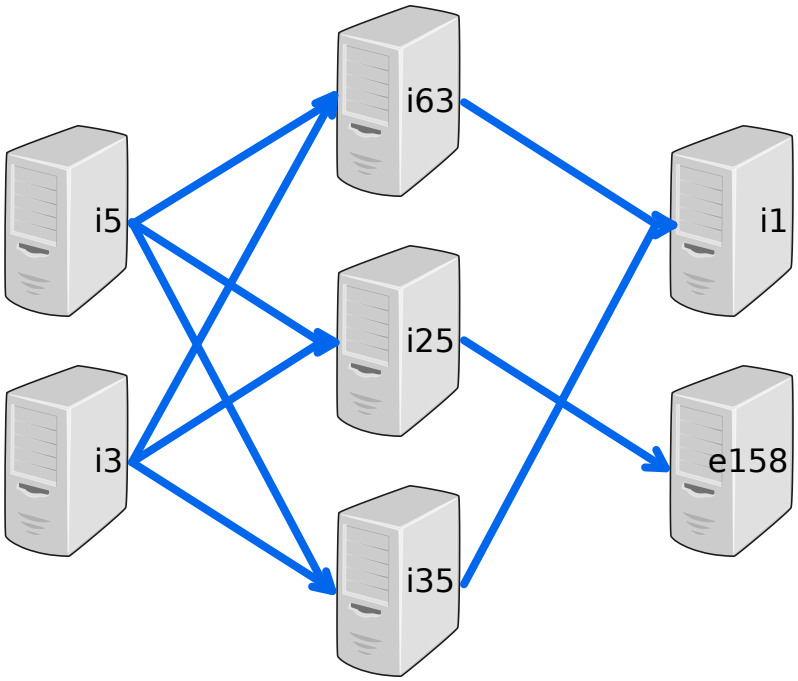

Figure 5.2: Asset dependencies inferred from the most frequent connection chains.

Limitations

As previously alluded to, this causal chain design suffers from several shortcomings:

- This initial implementation is naïve, in that the initiating client is included as a member of each connection chain. From one perspective, the identity of the client is relevant because different clients *could* induce different behavior from the same asset, leading to different suffixes to chains that start with the same first link. However, since many different clients can induce the same behavior from an asset, chains may exhibit common suffixes where only the initiating client is different.

  Therefore, the fact that this current implementation does not aggregate chains by common suffixes means that the number of unique chains presented in results is somewhat inflated. Enhancing chain aggregation to group chains by common suffixes is deferred for future work.

- As previously acknowledged, the presence of the ΔT_1 constant leads to a brittle design that may fail to correctly infer chains of all types (due to differences in network and/or application timing characteristics). While not reviewed in detail, tests showed that changing the value of ΔT_1 did impact results. Solutions that base their assessments

on distributions derived from the difference in start times (e.g., [8], Orion [17], and Sherlock [11]), as compared to this approach with a fixed threshold, are more robust in this regard and serve as guidance for future work.

- The strategy presented in this chapter only infers LR dependencies and not RR dependencies (though a rough starting point for a RR dependency solution based on this approach was previously mentioned when the strengths of this strategy were discussed).

- Identifying chains, like many other strategies, is vulnerable to the NetFlow "blind spot". It is particularly difficult to remedy this issue of missing data, as a researcher at best can make assumptions about whether or not any other communication *might have* existed at the point in time when flow data was collected. While statistical inference could potentially support filling in a few "gaps", that is not the case here; it would be difficult, if not impossible, to infer communication patterns since there is a complete lack of intra-subnet communication information.

5.4 Summary

This chapter presented a strategy for inferring LR dependencies by composing chains of connections linked by common hosts and similar starting timestamps. While this technique faced robustness challenges (e.g., selection of an appropriate value for the ΔT_1 constant), it does yield interesting results that aid in the exploration of load balancing conducted in future chapters of this thesis. Further, the approach described here shows great promise for future enhancements that will yield richer results.

The end of this chapter marks a change in course from focusing on asset dependencies to working towards robust strategies for inferring instances of load balancing. The motivation for this shift in focus is the thought that identifying the presence of asset clustering (i.e., load balancing and failover) would allow multiple assets acting as cluster members to be coalesced into a single asset entity, which should simplify dependency analysis. The remainder of this thesis focuses on those load balancing detection strategies, with all outstanding dependency and ranking concerns saved for future work.

Chapter 6

Detecting Load Balancing via Flow Characteristics

This chapter and the next chapter explore strategies for inferring the presence of load balancing from two different perspectives: this chapter focuses on flow attributes representative of application behavior (e.g., request and response sizes) that are indicators of load balancing, while the next chapter focuses on host attributes (e.g., the set of peers an asset communicates with) that do the same.

6.1 Strategy Intuition

The basic intuition behind the strategy presented in this chapter is as follows: if two (or more) assets are load balanced, then they have been configured to provide the exact same service to all of their clients. If these systems are providing the same service, then each instance of that service should exhibit the same behavior (i.e., the load balanced assets should all have the same role). To evaluate asset behavior from flow characteristics, only those characteristics that are a direct result of an asset's service behavior should be selected. In this work, three flow characteristics have been selected and are used:

Request Size The total number of bytes sent by the client (connection source) to the server (connection destination).

Response Size The total number of bytes sent by the server in response to the client.

Connection Duration The duration of the connection, which is defined as the difference between the connection's end and start timestamps.

Selection of the request and response size flow characteristics is an excellent starting point, as they directly map to application behavior. The motivation for including connection duration is more subtle: load balanced assets are likely located close to one another in the network,

such that any effects of network congestion (e.g., reduced throughput or elevated latency) or topological placement will likely impact all such assets.

If CANDID finds that these selected characteristics are similar for an asset pair, then CANDID infers that those assets are load balanced.

Readers should note that when pairs of assets are evaluated for similarity of the three aforementioned flow properties, the client and server behavior of assets are evaluated separately. In other words, if a service running on a host participates with other network entities both as a client (i.e., initiating connections) and as a server (i.e., responding to connection requests), then that asset will participate in two separate similarity comparisons: one for client activity and one for server activity. Client and server behavior are separated as these can reflect two significantly different aspects of the way in which a service interacts with the network.

6.2 Design Details

Pre-Filtering Candidate Assets

This analysis strategy functions by evaluating assets in a pairwise fashion: every asset is paired with every other asset and the load balancing assessment (to be described later in this chapter) is performed. Naturally, however, there is no reason to evaluate every single host in the empirical dataset. To begin with, all external hosts are excluded from this assessment, since the purpose of CANDID is to infer clustering of assets in an enterprise network, not outside of the network.

A slightly more subtle asset filtering choice is made with regards to asset popularity. It is reasonable to assume that services provided in a load balanced manner are popular and are frequently used. Therefore, to reduce the number of assets paired up for load balancing assessment since the number of pairs grows at a costly rate of $O(N^2)$, only *popular* assets are included in these pairings. CANDID deems an asset *popular* if it falls in the top 25 % of hosts when sorted by connection count. Selection of an appropriate *popularity* threshold remains an open question, however, intuition suggests that critical load balanced services will receive significant use, such that they will fall above the selected 25 % threshold. Moreover, so long as this threshold is not so small as to exclude any load balanced assets, it is merely an optimization. Table 6.1 summarizes the effects of this asset popularity filtering operation on both the number of assets involved as well as the number of connections those assets participate in. It is worth noting that while the asset counts are quartered (as expected), the connection counts are barely reduced, which shows that a majority of connections involve a small subset of all assets.

Asset Role	Assets		Connections	
	Total	Remaining	Total	Remaining
Client	796	204	205,451	203,728
Server	143	36	275,842	275,166

Table 6.1: Effects of filtering *internal* assets on connection popularity, with a threshold of keeping the top 25 %.

Building Flow Distributions

To perform statistical analysis on flow characteristics for each asset pair, CANDID assembles distributions for each desired flow characteristic (e.g., request size, response size, and connection duration) for each asset (treating client and server behavior separately, as previously noted). In other words, CANDID processes each connection instance and extracts the desired properties, which are then stored in 6 collections of values, which are the distributions of the flow attributes for the client and server participating in the connection.

Statistical Techniques

Once the set of candidate assets has been pre-filtered and distributions of flow characteristics for each asset have been assembled, CANDID begins statistical processing. More specifically, CANDID leverages several statistical goodness-of-fit techniques to assess whether the data values from the distributions for a pair of candidate assets are likely from the same distribution. If so, CANDID concludes that the assets represented by the pair in question are load balanced. The three goodness-of-fit tests CANDID uses are: (i) the two-sample Kolmogorov-Smirnov (K-S) test [53, §14.3.3], (ii) a two-sample variant of the well-known χ^2 test [53, pp. 733–734], and (iii) the λ^2 measure of discrepancy [46]. Note that for all of these tests, I use a significance level of $\alpha = 5\,\%$.

K-S Test

The two-sample K-S test is a hypothesis test that works as follows:

1. The test begins by creating empirical cumulative distribution functions (ECDFs) of the two empirical distributions being evaluated.

2. As if the two ECDFs are plotted and overlaid on the same set of axes, the test statistic D is computed as the maximum vertical distance between the two ECDFs.

3. The D measurement is compared to an estimated critical-value threshold derived from the number of samples [62], and if it is less than the threshold, then the test concludes that the two ECDFs are consistent with data values coming from the same distribution.

CANDID considers the two-sample K-S test to "pass" if it indicates that the two empirical data collections being compared are consistent with coming from the same distribution. So, when using the two-sample K-S test, CANDID considers two assets to be load balanced if and only if the test passes for all three (request size, response size, and connection duration) flow characteristics, each at the $\alpha = 5\,\%$ significance level. Failure of the test for any one of the three characteristics indicates the two assets in question are not providing identical service, and thus, are not load balanced.

χ^2 Test

CANDID makes use of a variant of the well-known χ^2 test designed for assessing goodness-of-fit of two empirical distributions. Without describing all of the details here (see [53, pp. 733–734] for details of the two-sample variant), this test functions as follows:

1. The two empirical distributions the test is evaluating are log_2-transformed (per [46]).

2. Histograms are created for the two transformed distributions from the previous step.

   It is critical that the same binning methodology (i.e., assignment of bin boundaries in the histogram) is used when creating the two histograms. Rather than using the suggested fixed bin-width (FBW) formula presented in [46], CANDID uses a fixed bin count (FBC) of 10 bins. The FBC binning methodology determines histogram bin boundaries as follows:

   a) All of the values from the two source empirical distributions are collected together (including repeating values) to form a single empirical distribution.

   b) The 9 deciles (i.e., 10 %, 20 %, ..., 90 %) for the single unified distribution are computed.

   c) The 9 values from the previous step join the 0 % and 100 % boundaries to define the edges of the 10 bins for the unified distribution.

   d) The computed boundaries for the 10 bins are applied to the two source empirical distributions separately, thereby constructing two histograms with identical bin boundaries for the test.

   So, in summary, the FBC methodology produces bins which are bounded by the deciles computed from combing the two distributions together. While this FBC binning strategy does not guarantee that each of the 10 bins will be the same width, it does ensure that each bin contains roughly the same number of values (which is ideal for the χ^2 test). Should this FBC binning process fail, due to the inability to create 10 unique bins (due to insufficient variance of the source data), or should any of the 10 bins in either histogram contain fewer than 5 items (per [44]), testing does not proceed (and a "no outcome" result is generated).

3. The test computes a squared and scaled difference in bin quantities for the two histograms, one bin pairing at a time, accumulating the total difference across the entire histogram pair as the test statistic, χ^2.

   Scaling is performed to account for the fact that the two source empirical distributions may have different numbers of values (i.e., the two assets being compared participated in an unequal number of connections). The complete equation for calculating the two-sample χ^2 test statistic is shown in Equation 6.1, where D_A represents the first empirical dataset and D_B represents the second (the $D_{A,Bin}$ notation represents a single histogram bin for the given dataset and the $|\cdots|$ notation represents the number of values in the specified quantity).

$$\chi^2 = \sum_{Bin=1}^{10} \frac{\left(\sqrt{\frac{|D_B|}{|D_A|}} \cdot |D_{A,Bin}| - \sqrt{\frac{|D_A|}{|D_B|}} \cdot |D_{B,Bin}|\right)^2}{|D_{A,Bin}| + |D_{A,Bin}|} \tag{6.1}$$

4. The χ^2 test statistic is compared to a critical-value, and if the statistic is less than this threshold, the test concludes that the two histograms provide insufficient evidence to indicate that the two empirical distributions differ.

Like the two-sample K-S test, CANDID considers this test to "pass" if it indicates the two flow characteristic distributions come from the same distribution. Also like the K-S test, CANDID deems two assets to be load balanced if and only if the χ^2 test passes for all three flow characteristics for a pair of assets.

λ^2 Measure of Discrepancy

This third approach is particularly interesting, because unlike the K-S and χ^2 tests, which are hypothesis tests that yield boolean results, this approach yields a quantitative *closeness-of-fit* metric. Two strengths of the λ^2 metric are: (i) it can be used to compare closeness-of-fit results for several source distributions, even when the number of empirical values or the number of histogram bins varies, and (ii) the test result includes both the discrepancy metric and the associated standard deviation, which enhances the ability to compare results. Concrete details on computing the λ^2 measure of discrepancy and the associated standard deviation are described in [46] — the purpose of this section is to show how CANDID makes use of this discrepancy metric to infer load balancing. This analysis strategy is designed as follows:

1. Like the χ^2 test, the λ^2 test metric and associated standard deviation σ_λ are computed from histograms assembled from the $\log_2$-transformed empirical data and the FBC binning strategy.

2. CANDID computes $\langle\ \lambda^2,\ \sigma_\lambda\ \rangle$ for all 3 flow characteristics for each asset pairing.

Asset Role	Total Pairs	Test Type	Passing Pairs
Client	20,706	K-S	301 (1.45 %)
		χ^2	0 (0.00 %)
Server	630	K-S	11 (1.75 %)
		χ^2	0 (0.00 %)

Table 6.2: Summary of load balancing inference results using the K-S and χ^2 tests for client and server asset pairs. λ^2 results are not included in this table since they are of a different form (ranking).

3. The $<_\sigma$ comparison operator, described in [46], is used to sort asset pairs using the computed $\langle\ \lambda^2,\ \sigma_\lambda\ \rangle$ in ascending order (such that the first item in the sorted result represents the closest or "best" fit and the last item represents the farthest or "poorest" fit).

4. The sorting operation is performed for each of the three flow characteristics independently, such that each asset pair has three "rankings" (e.g., positions in the three sorted lists).

5. The test result for an asset pair is the arithmetic mean of the three rankings computed in the previous step. Use of the arithmetic mean not only provides a single and easy-to-understand "ranking" by which all asset pairings can be compared, but also avoids giving preference or weight to any of the individual test results.

In the current implementation, human intervention is necessary to manually interpret test results to set a threshold in the sorted results that separates asset pairs that are likely load balanced from those that are not. However, given the relatively-small number of asset pairs that generally pass this test, this limitation is acceptable.

6.3 Evaluation

All three statistical techniques were implemented to perform load balancing inference as described in the previous section. The 24-hour set of NetFlow records representing SMTP traffic from LBNL (date June 1, 2011) was used to evaluate these three techniques.

Case Study

Table 6.2 summarizes the results of the K-S and χ^2 tests. Reviewing these results led to some immediate observations:

- There are significantly more asset pairs where the assets are acting as clients rather than as servers.

  Further investigation led me to realize that, unfortunately, one property of the LBNL dataset is that a majority of asset communication only appears as client-side behavior and not server-side behavior, due to the NetFlow "blind spot" problem.

  With this shortcoming duly noted, a new research question arose: *is client behavior an accurate indicator of whether or not the server is load balanced?* So, in the remainder of this thesis, the strategies for inferring load balancing are applied to client behavior, and server behavior when possible, to see if this research question can be answered.

- The K-S test, while passing for less than 2 % of all asset pairs, still yielded far more passing pairs than a human can reasonably validate.

- No asset pairs passed the χ^2 test. This is discussed further in the *Limitations* section.

To aid in the validation process, which as previously mentioned, requires communication with LBNL operations staff, I elected to try a different approach. Rather than blindly applying these load balancing inference tests to all popular asset pairs (recall, this is the top 25 %), I elected to select assets participating in the largest number of connections and evaluated only those for the presence of load balancing. Histograms presenting asset connection counts are shown in Figures 6.1 and 6.2 (on pages 52 and 53) for clients and servers, respectively. Curiously, the client connection count histogram shows that the popular assets appear in pairs with similar connection counts, which hints that some or all of these top pairs may be load balanced. On the other hand, the server connection count histogram does not show this pairing pattern. Also, readers will recall that in Chapter 5, two hosts (`i3` & `i5`) appeared in balanced chains with common suffixes and since these assets also appear in Figure 6.1 in a paired fashion, it is likely that these assets are load balanced.

Communication with LBNL operations staff confirmed the presence of three load balanced asset pairs:

1. `i3` & `i5`
2. `i37` & `i50`
3. `i3306` & `i56474`

Other load balanced assets may exist in the LBNL network, however, due to the network's scale, we could not confirm the existence of any other instances of load balancing. Thus, readers should be aware this evaluation does not consider false negatives nor false positives [1] and that the remainder of this chapter focuses only on the three known load balanced asset pairs.

[1] Given the significant number of asset pairs passing the K-S test (Table 6.2), false positives are likely.

Asset Pair		# Connections	K-S Test	χ^2 Test	λ^2 Rank
i3 & i5	Client	134,602	Pass	Fail	1 of 19
	Server	274	Pass	Fail	No Result
i37 & i50	Client	33,153	Fail	Fail	2 of 19
3306 & i56474	Server	261	Pass	Fail	No Result

Table 6.3: Summary of connection counts and statistical load balancing test results for known load balanced asset pairs.

Results Overview

Each of the statistical methodologies described in this chapter was applied to the three known load balanced asset pairs. Note that due to the NetFlow "blind spot", the LBNL data only has client behavior for the i37 & i50 asset pair and only server behavior for the i3306 & i56474 asset pair. Table 6.3 summarizes the test results of all three strategies for the known load balanced asset pairs and includes the total connection count for each asset pairing to give a sense of the effects of the NetFlow "blind spot" on observations of server behavior.

To visualize each of the distributions evaluated by the tests, a collection of empirical CDF (ECDF) plots of the request size, response size, and connection duration flow characteristics are presented in Figures 6.3 through 6.12 (pages 54 through 63). In these plots, the ECDF for each asset in a pairing is overlaid on the same axes, with the legend indicating the asset pairing in question. Note that the numbers in parenthesis in the legend indicate the number of connections the associated asset has participated in, and that in some ECDFs, the x-axis is log-scaled (see the axis label for details).

At a glance, Table 6.3 shows that: (i) the two-sample K-S test shows promising results since it passes for most of the known load balanced pairs, (ii) the χ^2 test is a point of concern since it never passes, and (iii) the λ^2 metric yields compelling results for client behavior but not for server behavior. Next, test results are explored in detail and reasons for test failures are examined.

Results in Detail

Perhaps the most striking load balancing relationship is that of the client behavior of the i3 & i5 asset pairing. Not only does this pairing pass the K-S test, it also scores the top rank for the λ^2 test. These results are well-justified, seeing as how the ECDFs in Figures 6.3, 6.4, and 6.5 reveal that the flow characteristic distributions for the two assets are nearly indistinguishable. This is a strong indication that the answer to the open research question, "*Is client behavior an accurate indicator of whether or not a server is load balanced?*" may in fact be *yes*. While the failure of the χ^2 test in this case may come as a surprise, a closer study of Figure 6.3 using a region of detail shown in Figure 6.13 reveals that the two flow

distributions are in fact noticeably different.

The server behavior ECDF plots of the asset pairs `i3` & `i5` and `i3306` & `i56474`, shown in Figures 6.6 and 6.7 and Figures 6.11 and 6.12, respectively, are less compelling. However, it is clear why both the χ^2 and λ^2 tests failed to produce results: the request size ECDFs (Figures 6.6 and 6.11) show essentially no variance, which means that the FBC binning strategy (which requires dataset variance to produce 10 distinct bins for the histogram) cannot produce histograms for these distributions. The fact that SMTP server requests lack variance suggests non-natural behavior, as the expectation is that a variety of message sizes should generate a wide variance in request sizes; combined with the observation that these assets participated in few connections, it is likely that the NetFlow "blind spot" is masking a majority of the relevant traffic.

While the λ^2 result for client behavior of the asset pair `i37` & `i50` is a good sign (second-highest rank, with `i3` & `i5` claiming the top rank), readers may be troubled that this asset pair fails the K-S test (and is the only asset pair to do so). However, a review of the request size distributions depicted in the ECDF of Figure 6.8 shows noticeable differences between the two distributions at the 10 % and 60 % points.

To confirm that my intuition for this general strategy (asserting that flow distributions will be similar for load balanced assets and different for assets that are not load balanced), I visually inspected two ECDFs for asset pairs that LBNL has confirmed are *not* load balanced. Figure 6.14 shows the request size distributions for the pairing `i3` & `i37` and Figure 6.15 shows the response size distributions for the pairing `i5` & `i50`. These two figures confirm my intuition: that flow characteristic distributions for assets that are not load balanced are clearly different. While this is only a visual inspection, it is reasonable to assume that the three statistical methodologies introduced in this chapter would confirm this result.

Strengths

In my mind, the results in Table 6.3 make one major strength of the presented solutions apparent: the λ^2 test. First, the test appears to be effective at whittling down the total number of candidate asset pairs (given that worst rank is 19, while Table 6.2 shows 20,706 total client asset pairs). Second, and perhaps more importantly, the combined ranking strategy does appear to be an effective mechanism for making sense of results, as two known load balanced pairs do hold the top two ranking positions.

Limitations

While the λ^2 metric of discrepancy does yield promising results, the solutions presented in this chapter do have several limitations.

As [48] aptly points out, the style of statistical testing used in this chapter (χ^2 in particular) is of the nature that it becomes extremely sensitive as the size of the dataset grows. Therefore, while unfortunate, it is not surprising in retrospect that these tests never yield

passing results, since they catch subtle differences in the ECDF plot pairs, even in situations where the curves appear near-identical. Also, as previously noted, both the χ^2 and λ^2 are unable to produce meaningful results for many asset pairs since the FBC binning algorithm requires many data points and significant variance to be effective.

More generally, however, the three statistical methodologies have a variety of trade-offs, which include:

- While the two-sample K-S test works well for distributions with both small and large numbers of elements, it is not a terribly sensitive test, given the significant number of passing client asset pairs in Table 6.2 and the likelihood that not all of those ∼300 pairs are load balanced.
- Both the χ^2 and λ^2 tests are unable to provide meaningful results for distributions with a small number of values or little variance, because it is difficult to create histograms from distributions with these properties. While the specific requirements of different binning strategies (e.g., FBC or FBW) may vary, the need to construct a histogram for these tests means that these limitations are fundamental for this class of tests.
- The K-S and χ^2 tests provide boolean results (due to their hypothesis testing nature), which are easy to interpret, but lack the ability to give any sense of the strength of the load balancing relationship (and thus, do not provide any guidance as to how to compare two different asset pairs that both "pass" a test).
- The λ^2 test illuminates the size of the discrepancy between two distributions, but does not provide any guidance as to where the boundary between "good" and "bad" fits is located.

In summary, there is not a single "one size fits all" goodness-of-fit nor closeness-of-fit statistical test. Consequently, in future work, it is worth pursuing a methodology that is a hybrid of these individual methodologies, that selects the best statistical test given a pair of distributions. The biggest hurdle in that work will be establishing a means of comparing results across all test strategies.

6.4 Summary

This chapter introduced the idea of using statistics to find cases of similar flow characteristics as a means of inferring asset load balancing. Of the three statistical methodologies presented, the λ^2 metric of discrepancy is the best and most promising approach for a robust solution. While the inability to observe a significant portion of the server-side behavior of known load balanced assets initially presented a challenge, this chapter showed that client-side behavior can still be a strong indicator of whether or not a pair of servers is load balanced. This novel contribution is a notion that has not been explored in previous work.

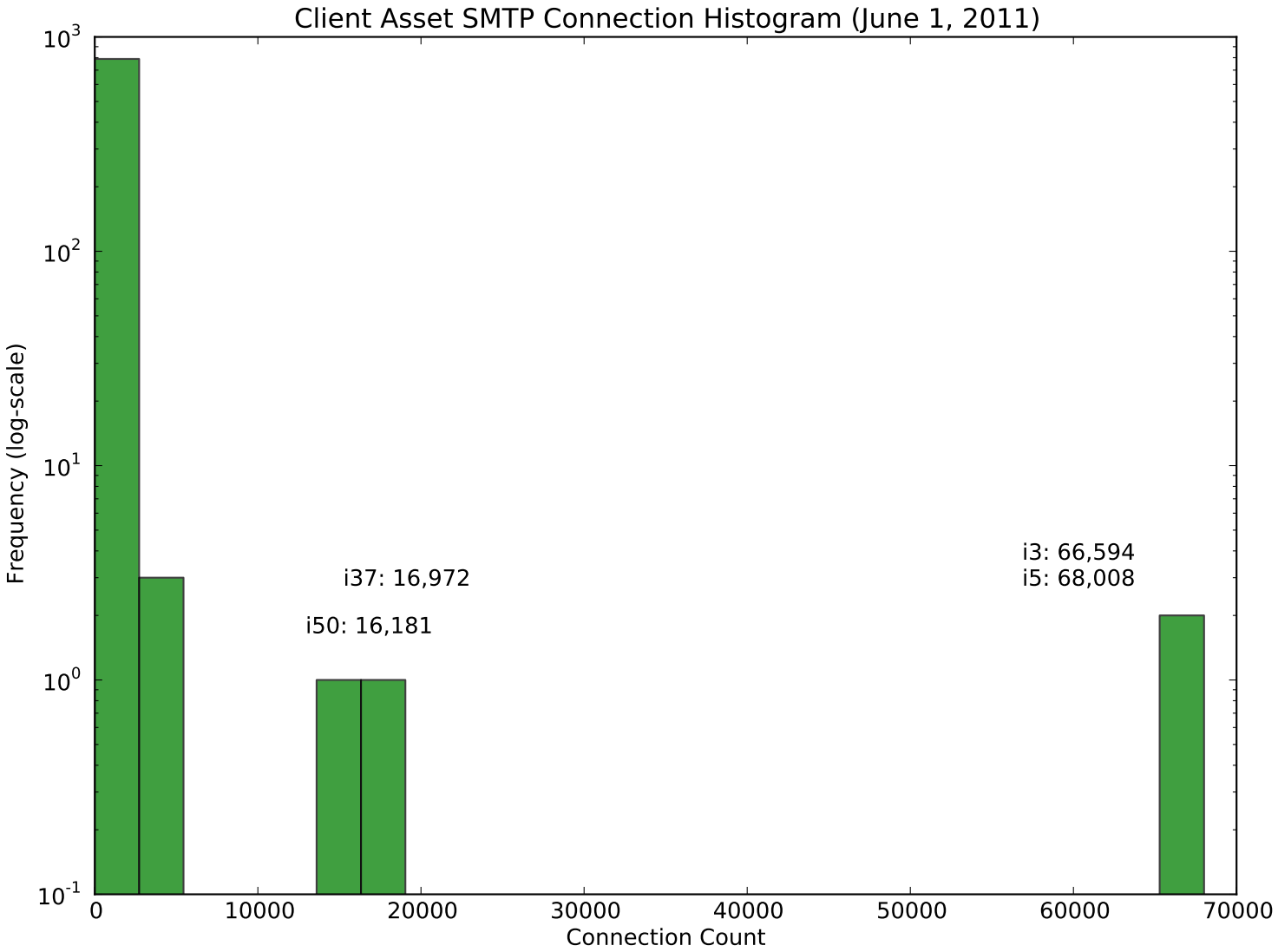

Figure 6.1: Histogram of LBNL SMTP traffic illuminating client assets with the top connection counts, where labels above the top bars indicate the associated asset and its connection count.

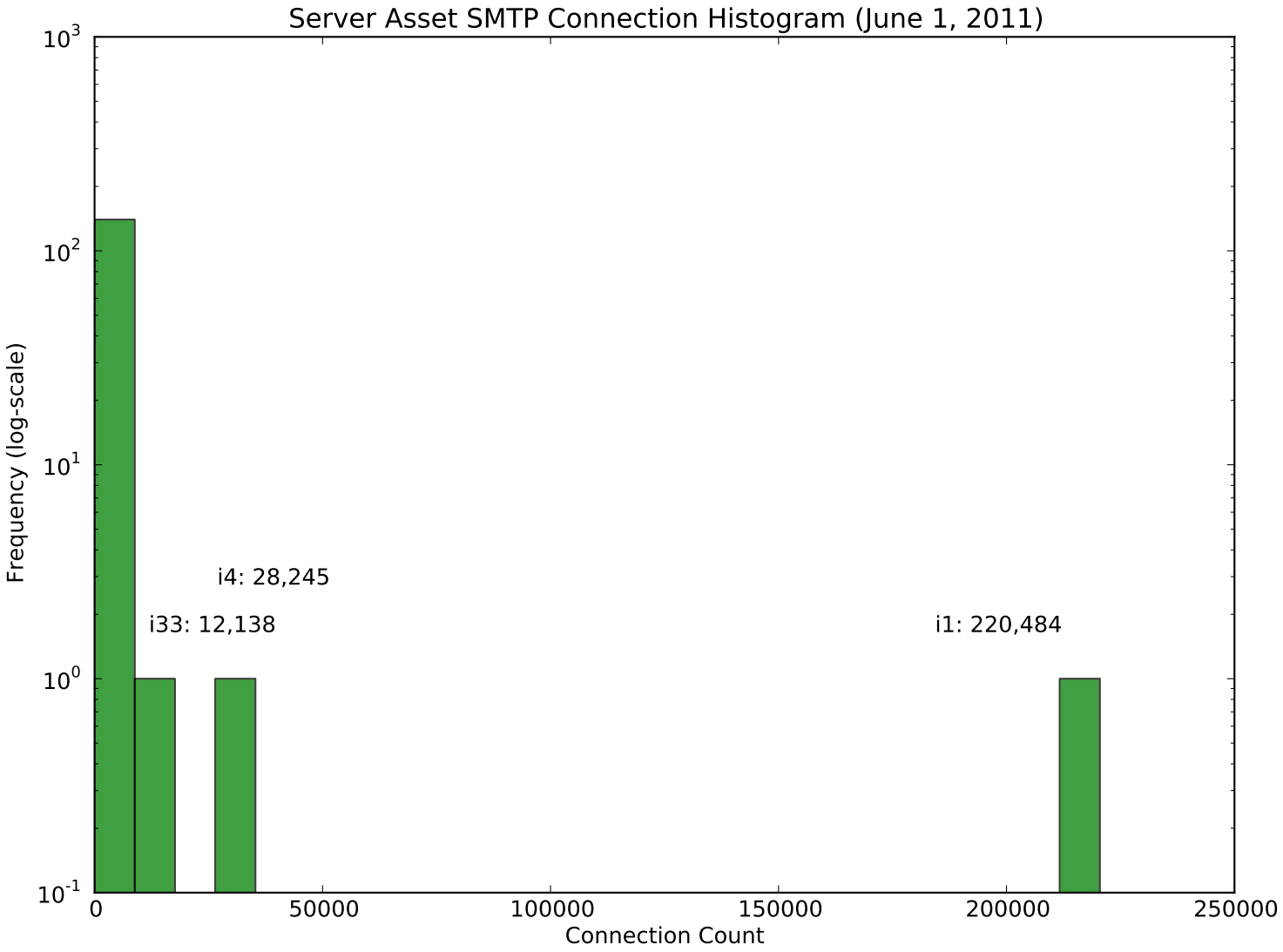

Figure 6.2: Histogram of LBNL SMTP traffic illuminating server assets with the top connection counts, where labels above the top bars indicate the associated asset and its connection count.

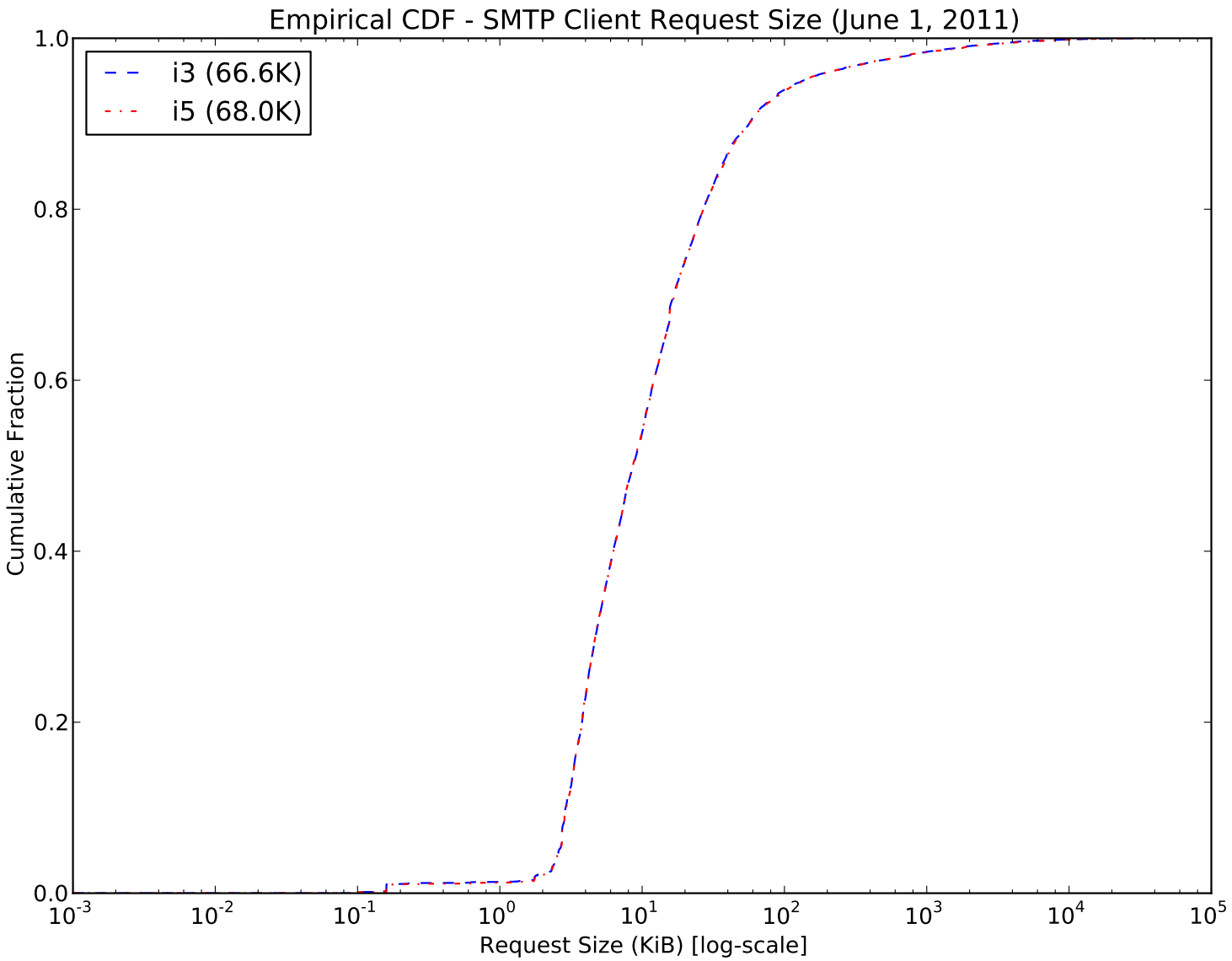

Figure 6.3: Request size empirical CDFs for SMTP client behavior of assets `i3` and `i5`.

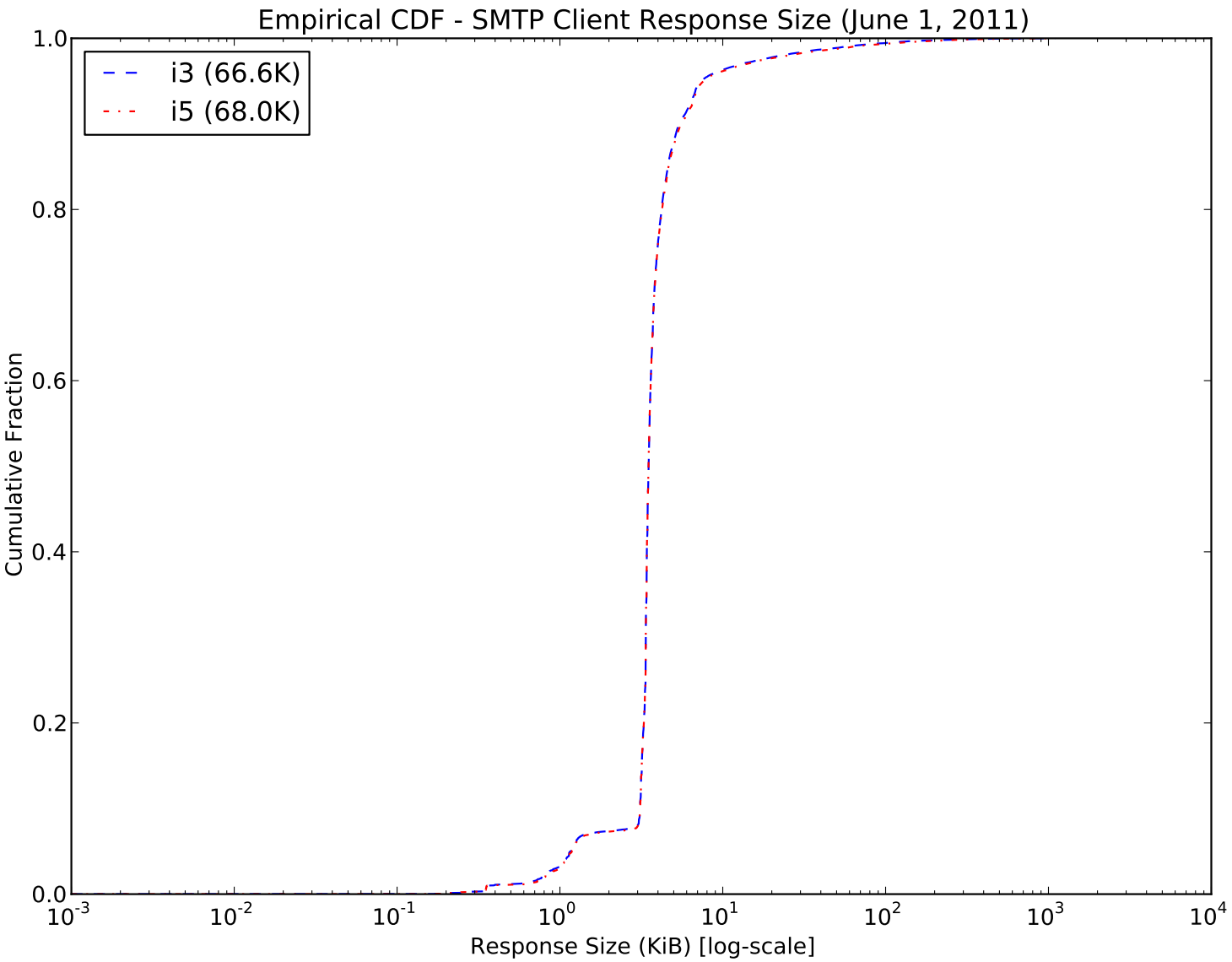

Figure 6.4: Response size empirical CDFs for SMTP client behavior of assets `i3` and `i5`.

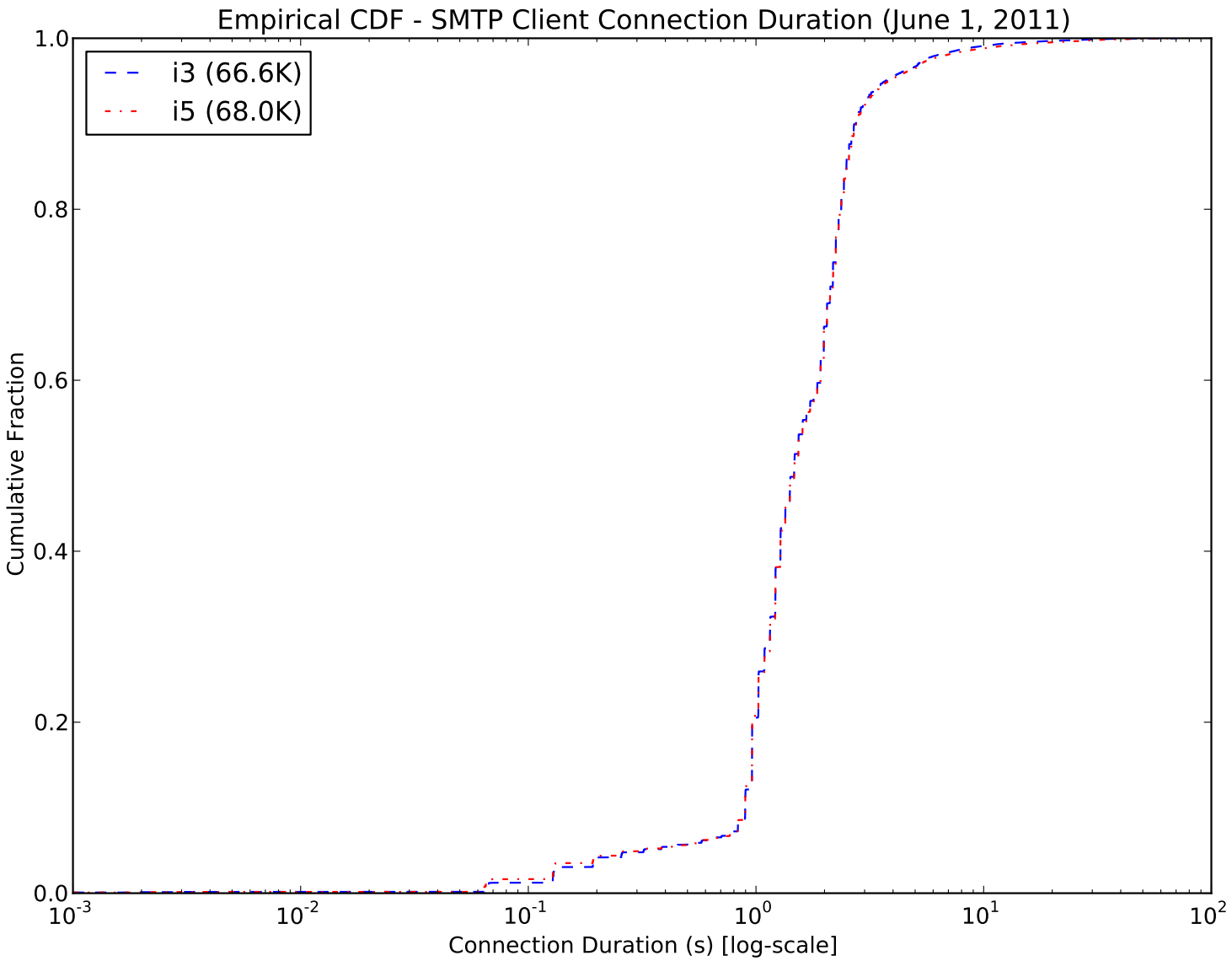

Figure 6.5: Connection duration empirical CDFs for SMTP client behavior of assets `i3` and `i5`.

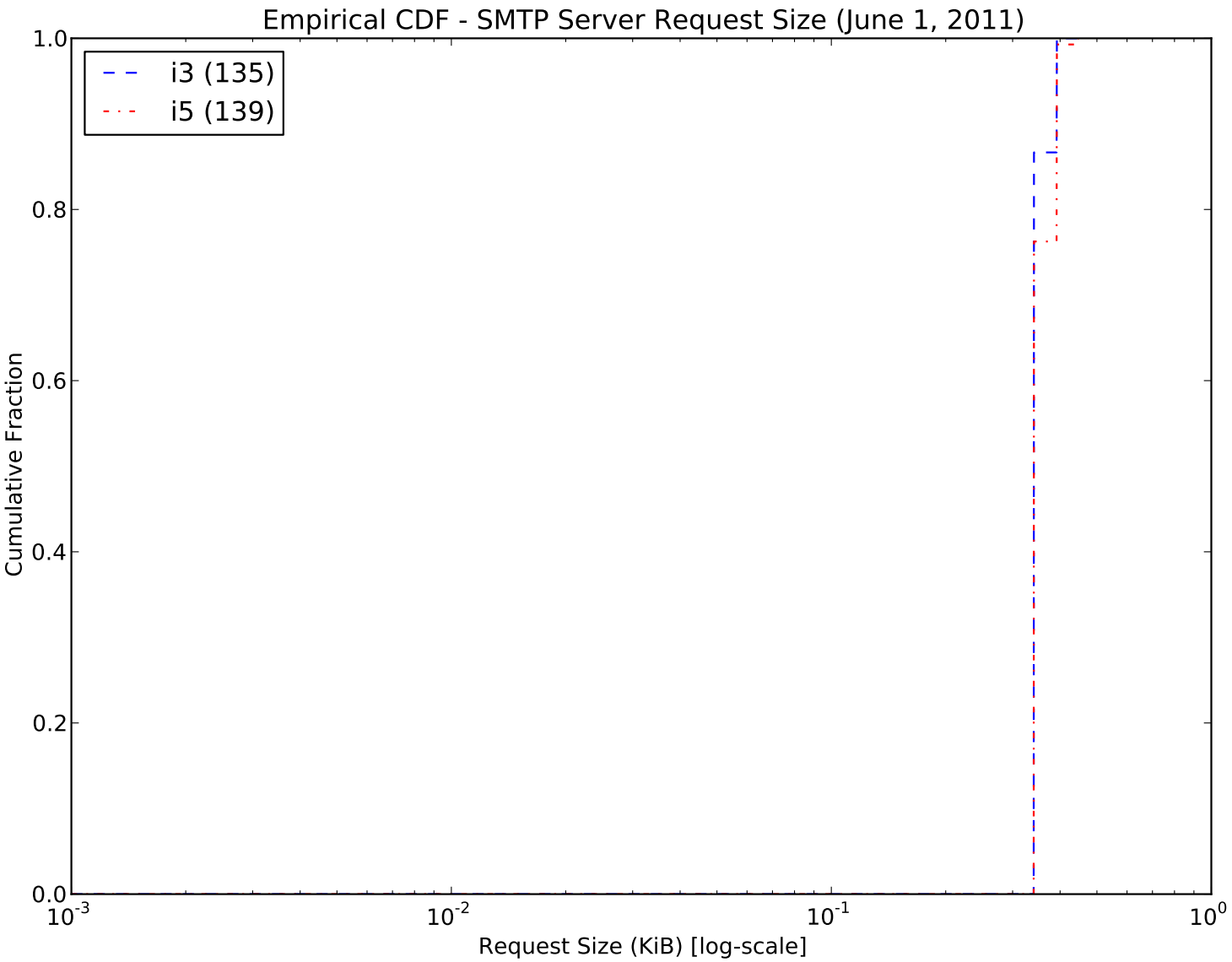

Figure 6.6: Request size empirical CDFs for SMTP server behavior of assets `i3` and `i5`.

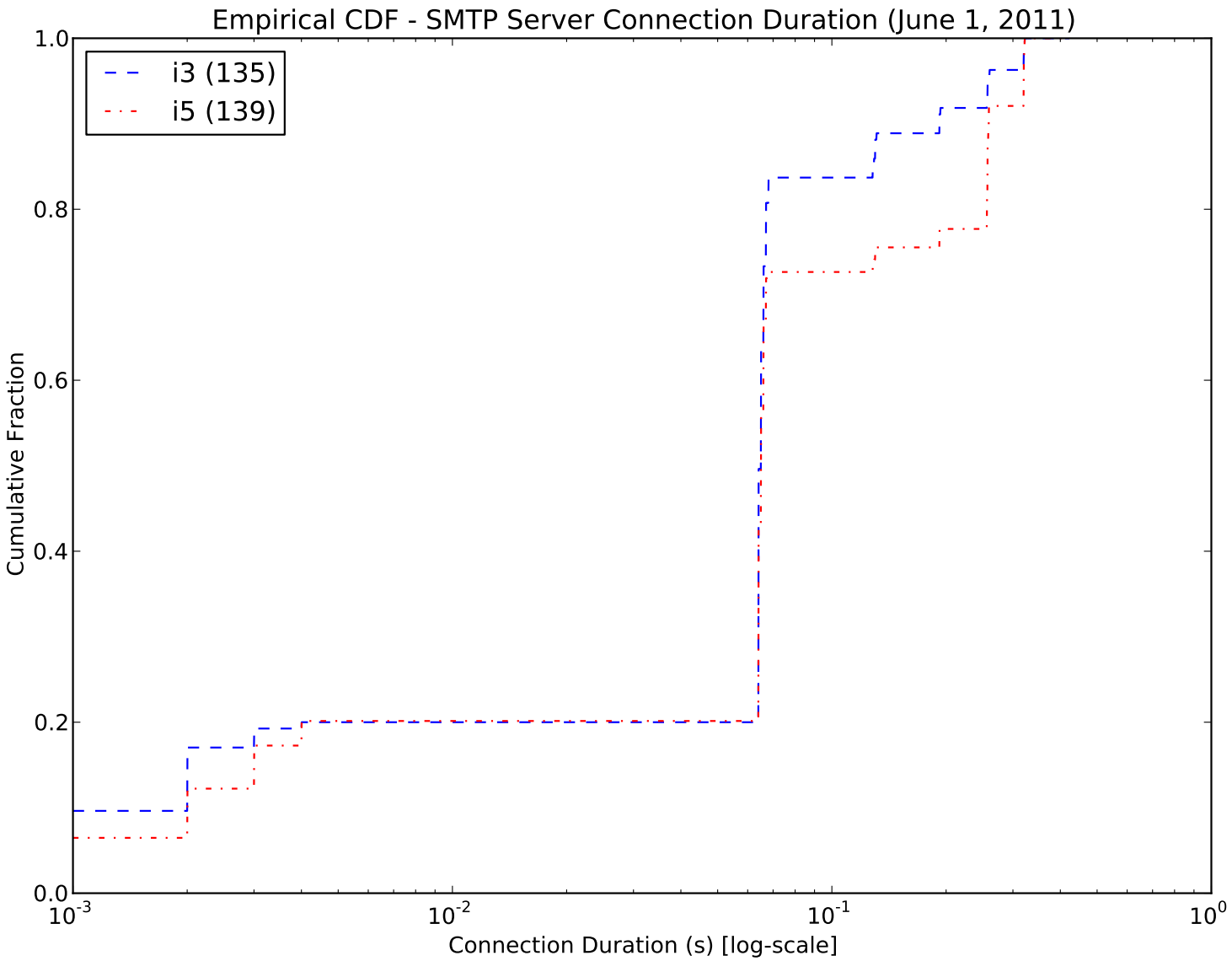

Figure 6.7: Connection duration empirical CDFs for SMTP server behavior of assets `i3` and `i5`.

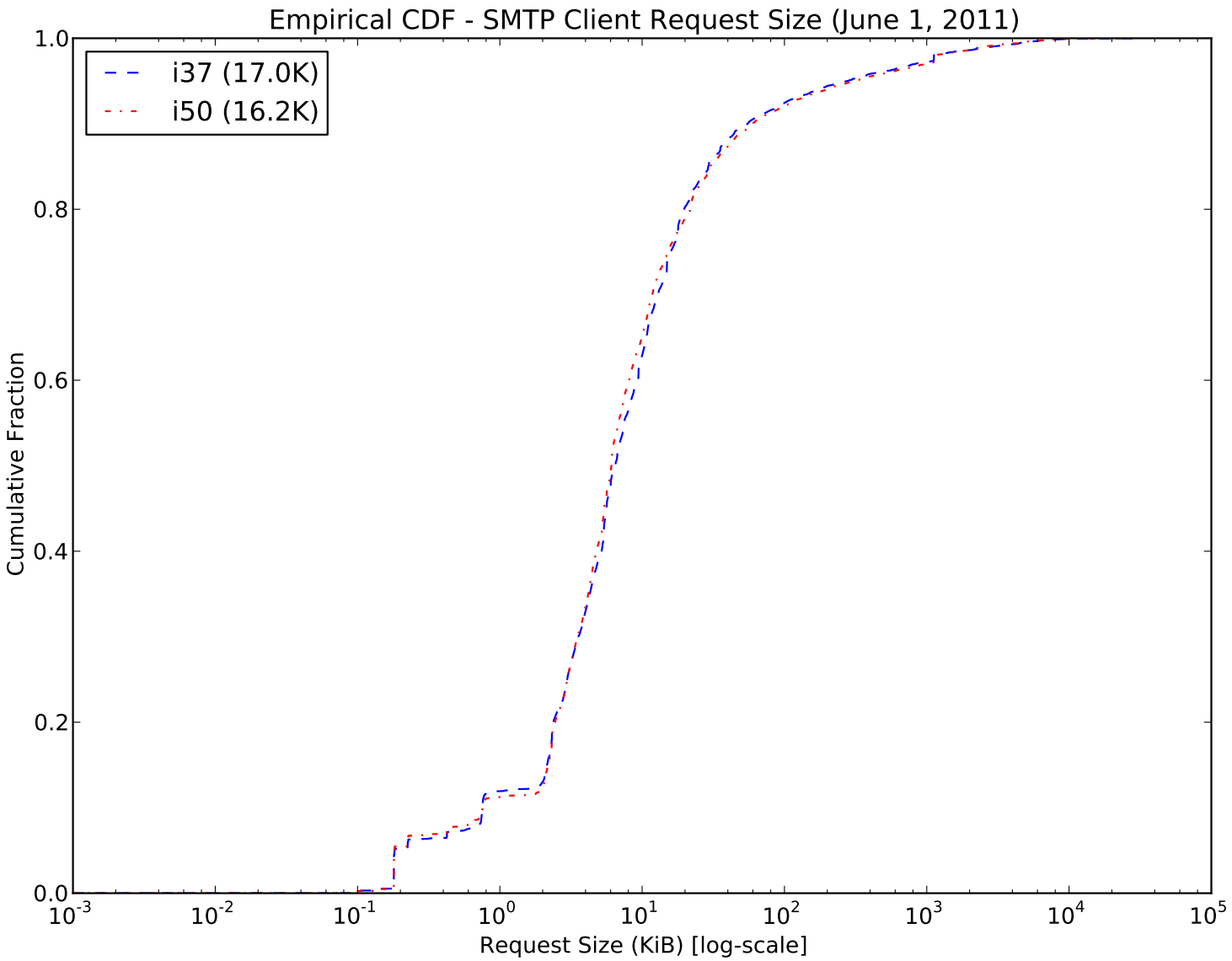

Figure 6.8: Request size empirical CDFs for SMTP client behavior of assets `i37` and `i50`.

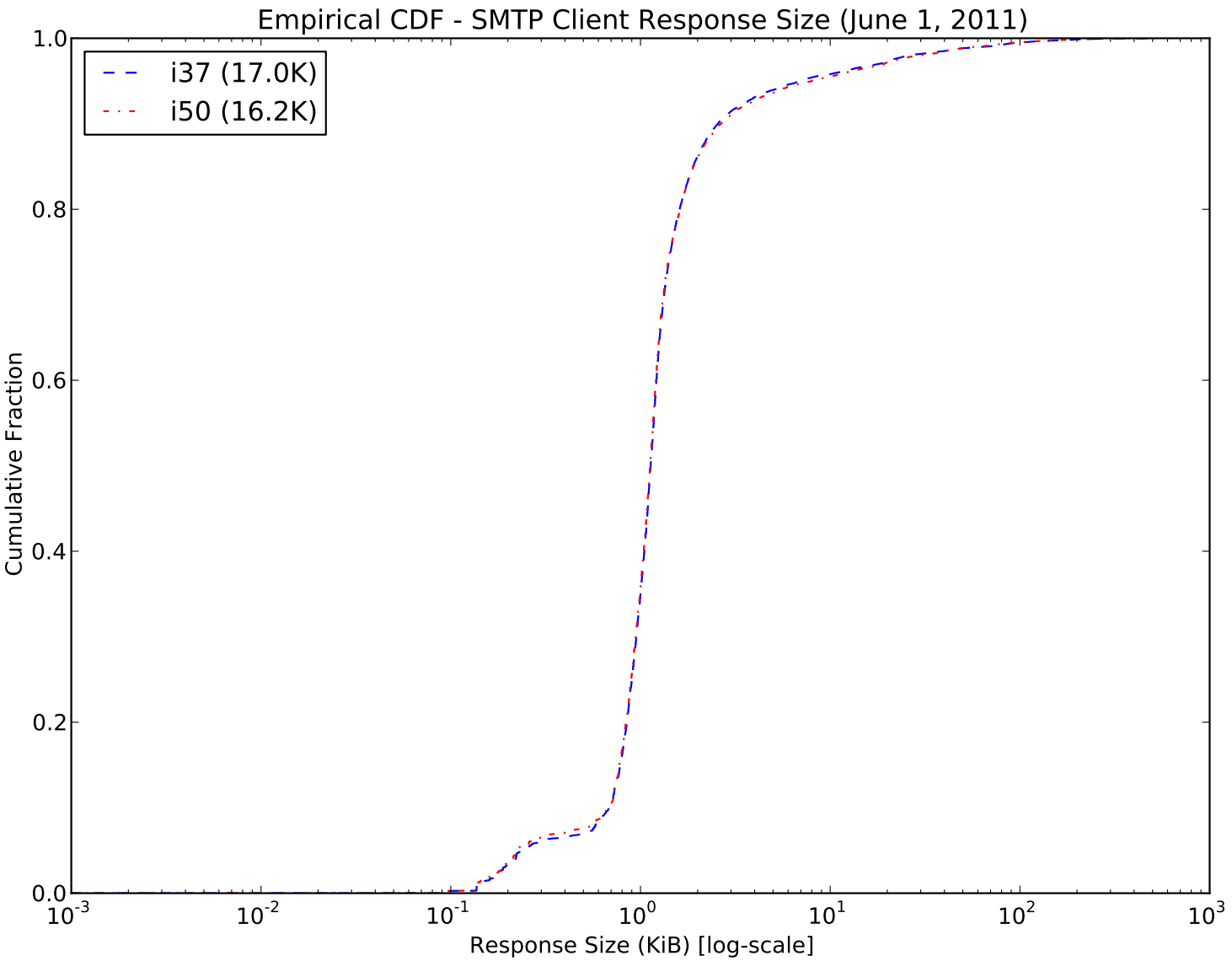

Figure 6.9: Response size empirical CDFs for SMTP client behavior of assets `i37` and `i50`.

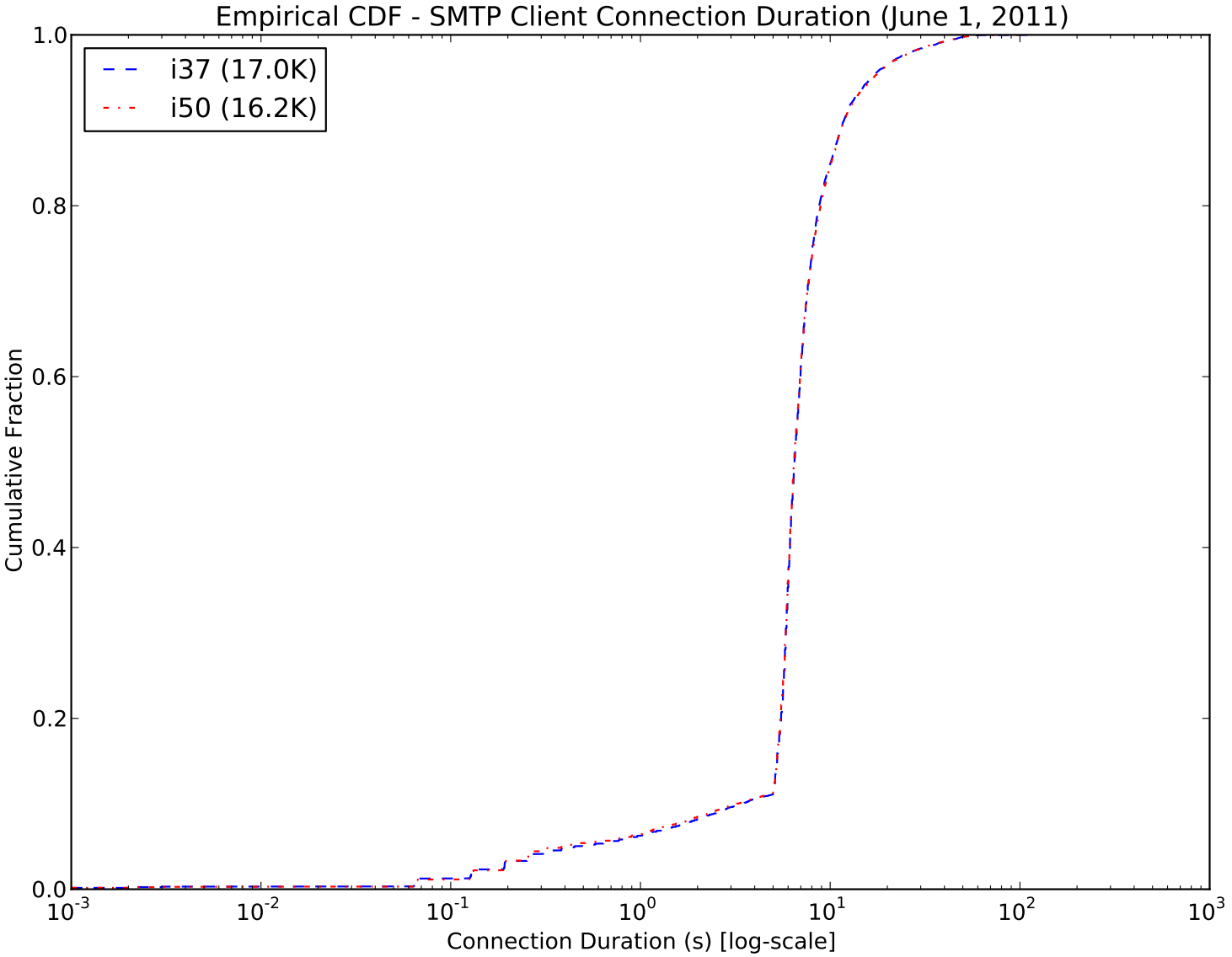

Figure 6.10: Connection duration empirical CDFs for SMTP client behavior of assets `i37` and `i50`.

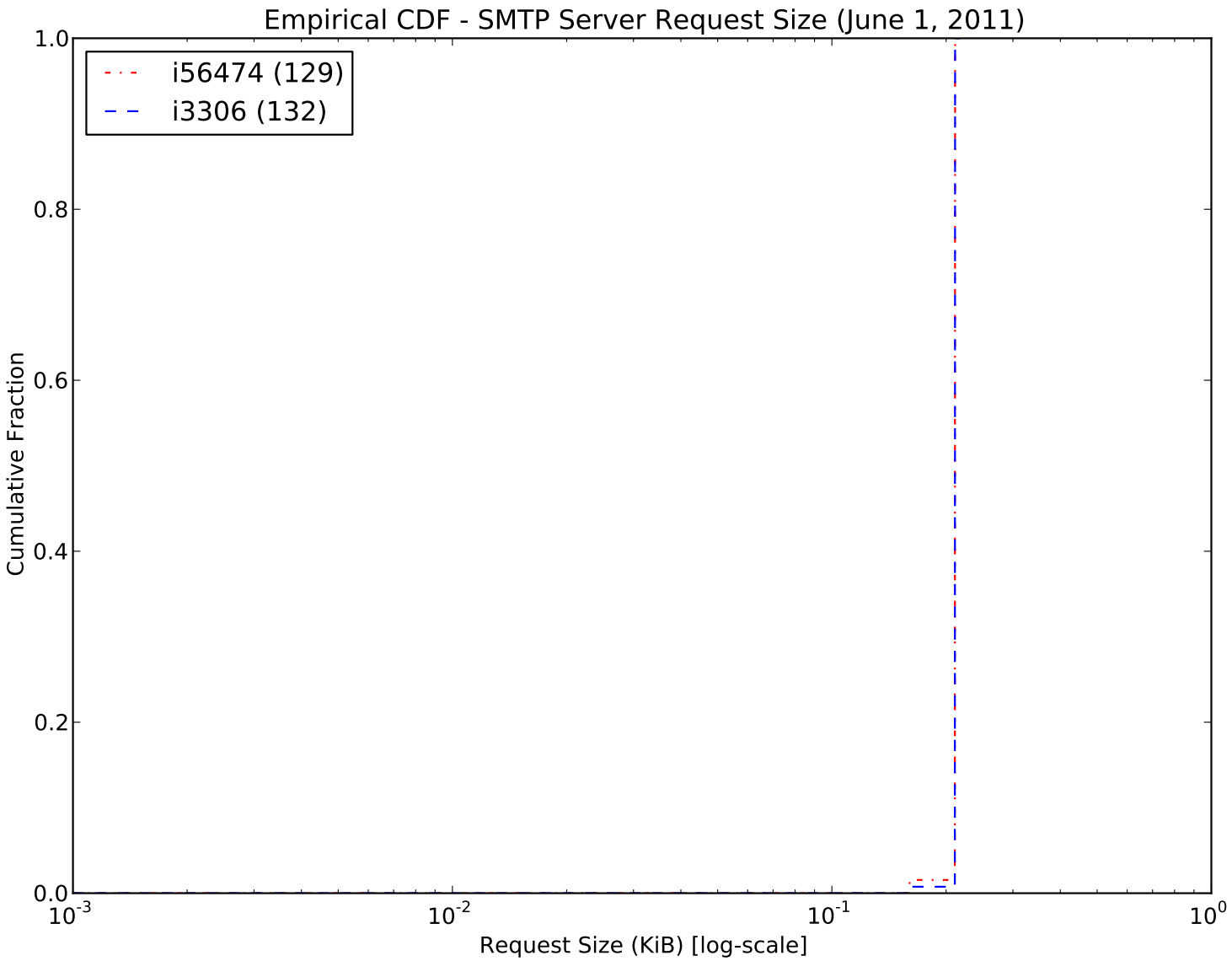

Figure 6.11: Request size empirical CDFs for SMTP server behavior of assets `i3306` and `i56474`.

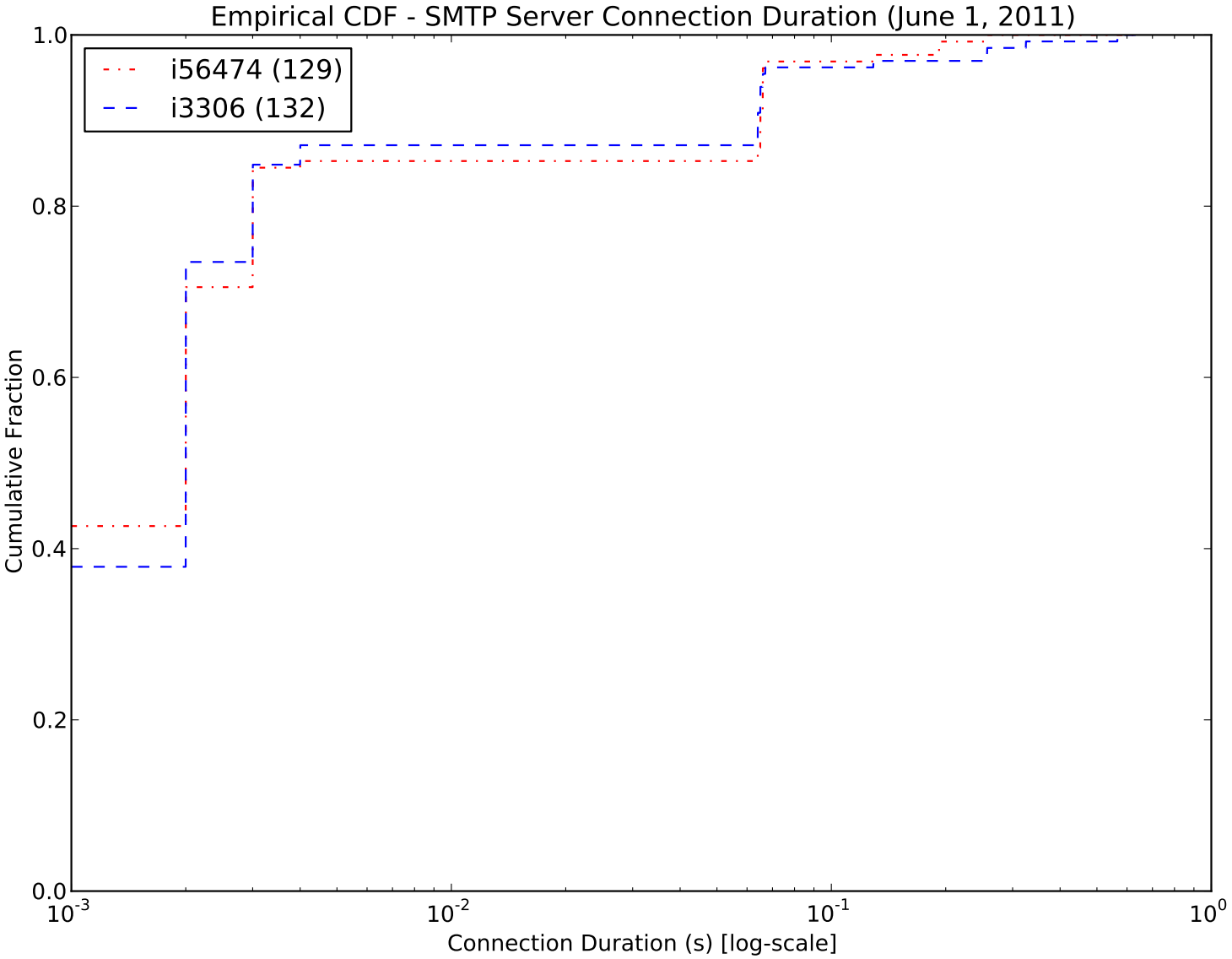

Figure 6.12: Connection duration empirical CDFs for SMTP server behavior of assets `i3306` and `i56474`.

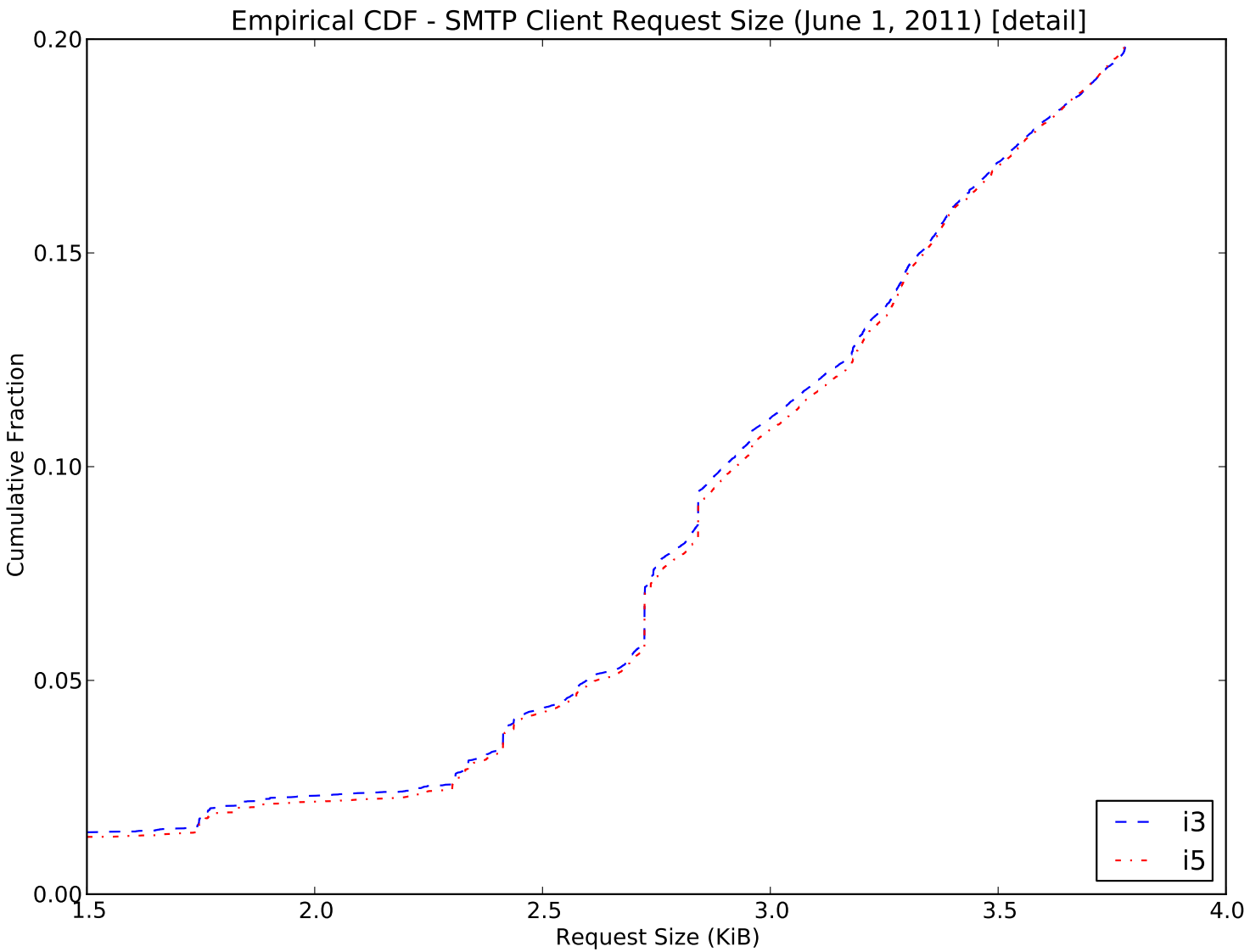

Figure 6.13: Region of detail illustrating the request size differences for SMTP client behavior of assets `i3` and `i5`.

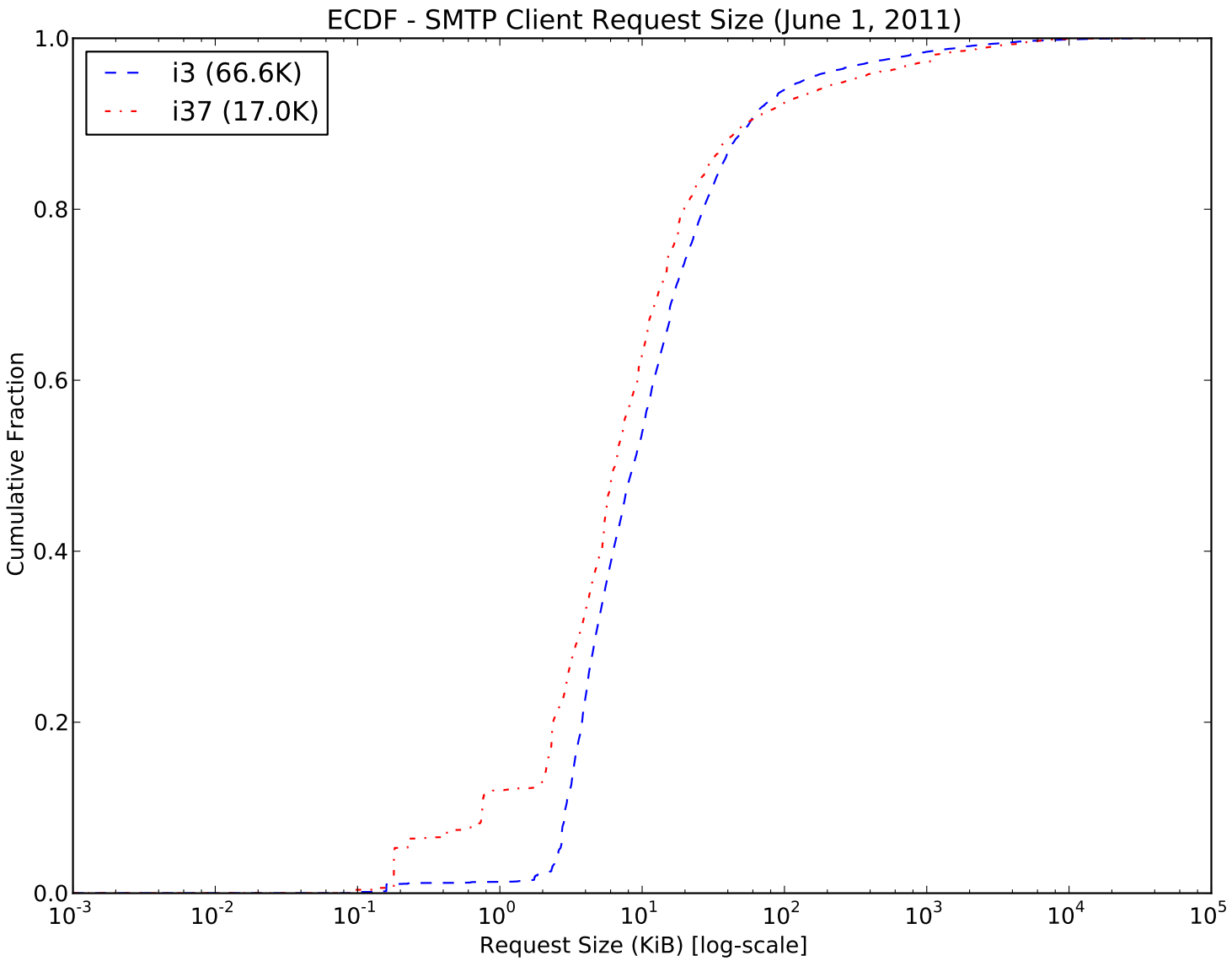

Figure 6.14: Request size ECDFs for SMTP client behavior of assets `i3` and `i37`, which are *not* load balanced.

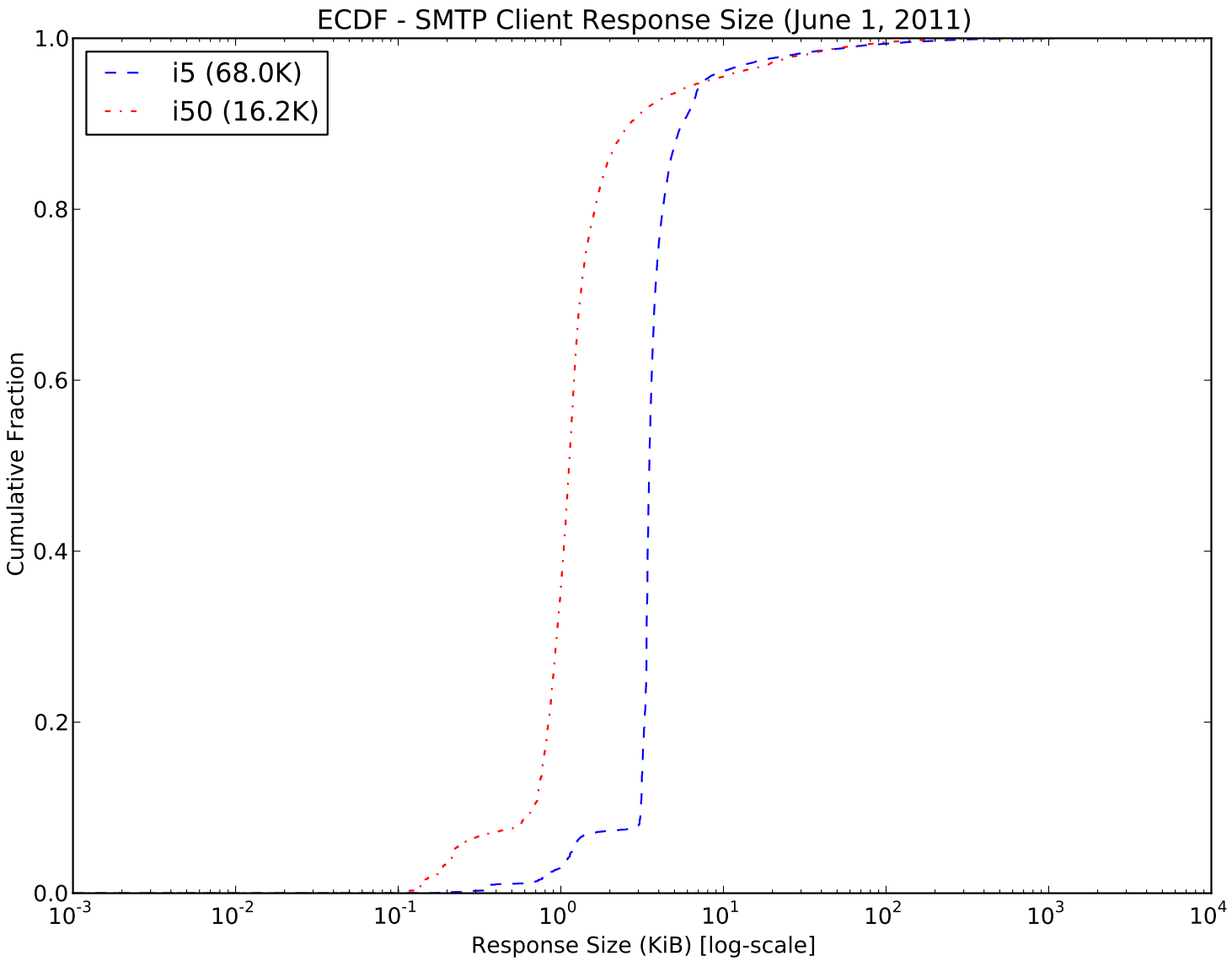

Figure 6.15: Response size ECDFs for SMTP client behavior of assets `i5` and `i50`, which are *not* load balanced.

Chapter 7

Detecting Load Balancing via Host Peers

This chapter explores load balancing inference from a different perspective than that of Chapter 6. In this chapter, I propose exploring similarities in host communication properties (i.e., the set of peers a host communicates with) as a means of assessing the presence of load balanced assets.

7.1 Strategy Intuition

Like the strategies in the previous chapter, the general approach taken here is to infer load balancing by looking for property similarity. However, rather than focusing on flow properties, the idea here is to focus on asset communication properties.

Given that two assets are load balanced (and thus, are providing identical service), intuition suggests the assets ought to exhibit similar social behaviors. In other words, I would expect all member assets of a load balanced cluster to communicate with a common set of hosts. Therefore, the strategies presented here infer asset load balancing by evaluating the similarity of peers [1].

7.2 Design Details

The strategies presented here are variations on two common themes: (i) constructing sets of peers each asset communicates with and (ii) using the Jaccard Index.

[1] This model assumes dynamic load balancing, where the mapping of incoming client requests to cluster members is not fixed (i.e., statically partitioned).

Given two sets, X and Y, the Jaccard Index, J, is defined as the size of their intersection divided by the size of their union:

$$J = \frac{|X \cap Y|}{|X \cup Y|} \tag{7.1}$$

In this work, I present several variations of the basic Jaccard Index computation shown in Equation 7.1 that leverage filtering and weighting in an effort to give more credit to peer relationships that occur repeatedly. Regardless of the equation used, these strategies all work similarly:

1. A set of peers for each asset is constructed by observing all connections in which the asset participates. Like in Chapter 6, the client and server behavior of assets are treated separately.

   Some of the variations of this strategy make use of the number of connections each peer participates in, rather than constructing a strict set of an asset's peers (i.e., counting an asset-peer relationship more than once vs. only once). This distinction will be made clear.

2. Assets considered for pairwise testing are pre-filtered just as in Chapter 6 (i.e., using the 25 % popularity filter). Then, each candidate asset is paired with every other asset, and the desired Jaccard computation is applied to the peer sets for the given asset pairing.

3. Results are sorted in descending order, so that the asset pairing with the greatest Jaccard value (indicative of a strong similarity in peer sets) appears at the top of the list with rank 1.

A third strategy that quantifies peer similarity by constructing distributions that capture the relative spread of connections across peers (rather than absolute connection counts) is in active research, and will not be presented here. However, preliminary results indicate this is a promising approach worth pursuing further.

7.3 Evaluation

The general peer set comparison strategy described in the previous section was implemented. By reviewing results from the LBNL case study and learning about limitations of the initial approach, I made several revisions to the base Jaccard Index computation (Equation 7.1). The case study will walk through the various equations and the reasoning behind each of the attempted variations.

Asset Pair	J_A Rank	J_B Rank	J_C Rank	J_D Rank	J_E Rank
`i3` & `i5`	57,283	1	1	8,696	8,231
`i37` & `i50`	72,512	No Result	2	11,360	14,368
Worst Rank	316,410	301	316,410	316,410	316,410

Table 7.1: Client asset rankings for a variety of Jaccard-derived peer similarity assessment strategies.

Asset Pair	J_A Rank	J_B Rank	J_C Rank	J_D Rank	J_E Rank
`i3` & `i5`	97	1	26	744	367
`i3306` & `i56474`	145	2	32	743	366
Worst Rank	10,153	11	10,153	10,153	10,153

Table 7.2: Server asset rankings for a variety of Jaccard-derived peer similarity assessment strategies.

Case Study

Results Overview

Tables 7.1 and 7.2 summarize the results of the case study using the same set of known load balanced asset pairs that were introduced in the previous chapter. Table 7.1 presents the rankings of known asset pairs based on client behavior (where peers are servers) for the various Jaccard-derived strategies, while Table 7.2 presents similar results for server behavior (where peers are clients). The last row in each table, labeled "Worst Rank", represents the poorest rank an asset pair could be assigned for the given test; this value is equal to the number of asset pairs evaluated.

Naturally, the results presented in Table 7.2 are more in line with the intuition introduced at the start of this chapter, since they evaluate server behavior. However, due to the aforementioned effects of the NetFlow "blind spot" on the LBNL data set, client-side behavior exhibits richer peer data. In fact, like in Chapter 6, one motive of this case study is to attempt to answer the previously-stated open research question, "*Is client behavior an accurate indicator of whether or not a server is load balanced?*".

Unmodified Jaccard, J_A

The unmodified version of the Jaccard Index is presented as approach J_A. Thus, in this case, Equation 7.1 is used without any modifications. Further, this approach uses strict sets of peers — even if a peer participates in multiple connections with the same assets, the J_A computation will only count that peer *once*, and not multiple times.

Results in the J_A column of Table 7.1 reveal inflated rankings for asset pairings that are known to be load balanced. Closer inspection of test results revealed that of the 316,410 asset pairs evaluated by this test, 57,208 (18.1 %) of them tied with the top score. In retrospect, the reason for this outcome is not surprising: any pair of assets that happens to have a single common peer (perhaps due to infrequent communication occurring during the selected 24-hour time period captured by the NetFlow data) will earn that top score. Clearly, this unmodified form of the Jaccard Index is not well-suited to accurately pinpointing the presence of load balancing. While this behavior is not as pronounced in the server results shown in the J_A column of Table 7.2, a review of the test results revealed similar effects.

Jaccard Variation J_B

The first Jaccard variant, J_B, had two changes from the base Jaccard expression that was presented as J_A and shown in Equation 7.1:

- The set computations were adjusted to be connection-count aware, such that if a peer participates in more than one connection with an asset, each connection instance is included in the Jaccard computation, rather than simply counting the peer once.

  This change was motivated by the intuition that repeated connections from a common peer ought to strengthen the confidence that a given pair of assets is load balanced. Otherwise, two different pairs of assets, each with a single common peer will earn the same score, regardless of how many times that peer communicates with each asset pairing.

- The numerator component of the Jaccard expression (i.e., the intersection term) is squared [2]. CANDID squares this component to give greater weight to similarity in the Jaccard computation. This modification is depicted in Equation 7.2.

$$J = \frac{|X \cap Y|^2}{|X \cup Y|} \tag{7.2}$$

Further, in Jaccard variant J_B, only asset pairs that are first deemed load balanced by the two-sample K-S test (Chapter 6) are evaluated. This change was made because results from the J_A test were flooded with many asset pairs which happened to have a single peer in common, thereby earning a "perfect" $J_A = 1.0$ score, which diluted any meaningful test output. By first applying the two-sample K-S test, the Jaccard search scope is narrowed to only include asset pairings that are likely load balanced, removing those spurious results.

A review of the J_B column in Tables 7.1 and 7.2 indicates that this first variation on the base Jaccard strategy is reasonably effective, as all asset pairs that yielded results occupy top ranking positions. More importantly, however, it validates the intuition outlined at the start of the chapter, which stated that common peers would be an indicator of load balancing.

[2] This is not a standardized modification of the Jaccard Index.

Jaccard Variation J_C

Due to the fact that the client behavior `i37` & `i50` asset pair does not pass the K-S test, it made sense to adjust strategy J_B. Rather than simply removing the K-S pre-filter, the new strategy takes a different approach. It is natural to think that, due to the finite quantity of measurement data used, some peers which communicate infrequently with a cluster may only be observed to have connections to one of the two assets in that cluster. Therefore, the Jaccard computation should only take peers into account that each asset in the pair communicates with frequently. Thus, this new strategy, J_C, is different from J_B in that: (i) it does not use the K-S test to whittle down the set of eligible asset pairings, and (ii) it only includes the top 25 % [3] most popular peers of each asset in the computation. Note that this peer popularity filter is independent of the candidate asset popularity filter described in Section 7.2, which is used for all five Jaccard tests presented in this chapter. The numerator in the J_C computation remains squared.

Based on the results, this strategy seems quite effective at inferring load balancing relationships from client behavior. Unfortunately, the server behavior results suffer in this case due to the removal of K-S pre-filtering.

Jaccard Variation J_D

The fourth Jaccard test, J_D, is quite similar to variation J_C, except for the fact that it does not square the numerator in the Jaccard computation, leading to an expression like Equation 7.1. The primary motivation for this variation was to bring the expression back towards the traditional Jaccard Index, which is well-understood.

A quick glance at the results shows that the ability to correctly infer load balancing from both client and server behavior is severely impacted when weighting of the intersection term is removed. More specifically, of the 316,410 host pairs evaluated, 8,650 (2.7 %) of them tied with the top score, which makes it difficult to interpret results. While it remains an open question of what the best approach to weighting the intersection component is to achieve appropriate score differentiation, this series of Jaccard variations shows that some form of weighting is required.

Jaccard Variation J_E

The final Jaccard variation explored is identical to strategy J_D, except that the set of peers included in the computation is not pre-filtered. This variation was completed to understand to what extent the pre-filtering impacts results.

The results for strategy J_E show little improvement over those of strategy J_D; in fact, asset pair `i37` & `i50` scores poorer with J_E. The reason for the minor shift in scores (and small improvement in 3 of the 4 asset pairs) has not been analyzed in detail; I suspect it must

[3] Evaluation of the most appropriate peer popularity threshold is deferred for future work.

stem from the intersection component growing far more quickly than the union component when the peer pre-filter is removed. This suggests that the selected peer popularity threshold of 25 % may not be the most appropriate value.

Jaccard Strategies Summary

Table 7.3 summarizes the properties of the base Jaccard Index and the four Jaccard variations that were just presented. In particular, the table focuses on whether or not any pre-filtering is applied to the asset pairs evaluated or peers included in the test computation, whether the computation is connection count aware (rather than counting a peer just once, regardless of the number of connections it participates in), and whether or not the intersection component is squared.

Results from the four Jaccard variants presented in this thesis lead to a few key ideas:

- Combining Jaccard testing with a flow-based analysis technique (e.g., K-S in J_B) can have merit if the right flow-based technique is selected.
- The basic Jaccard Index, J_A is not connection count aware, nor does not give weight to common peers. Counting connections is important, as it better highlights relationships with repeated connections from common peers, and some form of intersection weighting is necessary to best illuminate instances of load balancing.
- Pre-filtering the set of peers included in the computation warrants further exploration, particularly with respect to choosing an appropriate popularity threshold.

Strengths

The greatest strength of the Jaccard-derived strategies presented in this chapter is that they do effectively illuminate instances of load balancing for *both* client and server behavior. This, in essence, validates the initial intuition I described at the start of the chapter and suggests that an approach (even if not Jaccard-derived) which evaluates asset social behavior will be an effective mechanism for inferring load balancing.

Limitations

The five strategies presented in this chapter showcased how pre-filtering and weighting impact the strength with which an asset's peers illuminate a load balancing relationship. As a part of this iterative process, two shortcomings emerged:

- With the exception of J_B, all of these variations *only* consider social behavior, and disregard any knowledge of the service characteristics explored in Chapter 6.

Jaccard Variant	**Pre-Test Filtering?**	**Connection Count Aware?**	**Squared Intersection**
J_A	None	No	No
J_B	Only considers asset pairs for which K-S claims equal distributions.	Yes	Yes
J_C	Only includes the top 25 % peers of each asset.	Yes	Yes
J_D	Only includes the top 25 % peers of each asset.	Yes	No
J_E	None	Yes	No

Table 7.3: Key properties of Jaccard Index computation variations.

- Not giving stronger weight to the intersection over the union fails to illuminate load balancing, but the current weighting strategy (squaring the intersection term) also lacks a principled methodology.

7.4 Summary

This chapter took a radically different approach from the previous chapter, in that rather than inferring load balancing by evaluating similarities in flow characteristics, it presented strategies that attempt to illuminate load balancing via host social behaviors. Of the five strategies presented here, J_C (peer popularity pre-filtering and a squared intersection component) performed best for client behavior while J_B (K-S pair pre-filtering and a squared intersection component) performed best for server behavior. These strategies validated the initial intuition and provide guidance as to how best to proceed in future CANDID research.

Chapter 8

Discussion

The previous three chapters provided details of the approaches I have taken towards inferring LR (local-remote) dependencies and locating cases of load balanced assets. This chapter reviews the contributions of this thesis and paves the beginning of the path towards further research in this area.

8.1 CANDID Contributions

The contributions of this thesis span beyond the specific solutions presented in Chapters 5 through 7. In fact, perhaps the greatest contribution of this thesis is not one I initially set out to tackle: the idea that the behavior an asset partakes in as a client does directly reflect its role as a server. Thus, I see the contributions of this work to be as follows:

- As just noted, the realization and concrete evidence that network measurement of an asset's client behavior can be directly tied to it's behavior as a server.
- Two drastically different strategies for inferring load balanced assets, one which focuses on application behavioral characteristics from flow data (Chapter 6), while the other focuses on asset social behavior from connection peers (Chapter 7).
- A comprehensive data handling and sanitization pipeline that addresses challenges other researchers may face when working with unidirectional NetFlow records as well as multiple network vantage points (Chapter 4).
- A causal connection chain strategy that works towards identifying LR dependencies. Further, results from this work shed light on a new approach set aside for future work: aggregating chains by common suffixes as a way to separate clients from shared services with common dependency components.

8.2 A Fresh Take: a Hybrid Approach

I am currently investigating a new comprehensive solution for identifying load balanced assets. As this is currently in active research, the solution has not been fully formalized, nor are results available. However, it is worth briefly explaining my idea here, as it serves as a strong jumping-off point for future work. A review of Chapters 6 and 7 shows that there are currently two disparate solutions for inferring load balancing. What I have come to realize is that while both sets of techniques are based on correct intuition of load balancing behavior, neither solution alone is sufficient to correctly characterize the presence of load balancing. Therefore, I believe the best approach to inferring load balancing is a new hybrid approach that leverages three facets of host and application behavior:

- Application behaviors, based on flow characteristics, which are evaluated for similarity using the λ^2 metric of discrepancy (Chapter 6).
- Host social interactions, based on similarity of peer sets, via a weighted Jaccard Index (Chapter 7).
- Peer connection patterns, which capture the relative balance of connections from peers across a pair of potentially load-balanced assets. This is different from the previous point in that it is more concerned with *how* peers communicate with a pair of assets, rather than with *whom* those assets communicate.

When these three perspectives are combined, a tool like CANDID can confidently assess assets for load balancing relationships. The remainder of this exploration is deferred for future work.

8.3 Lemonade from Lemons?

Admittedly, initial results from this research showed limited promise, particularly due to the NetFlow "blind spot" and the use of the fixed ΔT_1 threshold. However, once I discovered that client-side behavior does appear to be an indicator of whether or not an asset is load balanced, I recall one of my advisors noting that we might be able to "make lemonade from lemons".

As I now reflect on the question, "Did I make lemonade from lemons?", I think the answer is a resounding *Yes*. Without a doubt, the contributions described in Section 8.1 are important, as are the individual results from the LBNL case study. However, I don't think any of these things alone are the "lemonade".

To me, the "lemonade" is the broader insight this work shares with the research community: (i) the idea that seemingly-trivial inference of load balancing via passive network measurement is a challenging research problem, (ii) that client network traffic *is* indicative

of server behavior, and (iii) the idea of a new hybrid approach (Section 8.2) towards solving this problem.

8.4 Future Work

This thesis has made great strides towards addressing the goals outlined in the problem statement from Section 1.1, yet at the same time, has raised new issues and avenues of research that are worthy of further exploration.

I see the two most-important tasks for the future to be: (i) completion of the hybrid design and peer connection distribution strategy introduced in Section 8.2 and (ii) evaluation of enterprise NetFlow records with a heterogeneous application mixture (i.e., not just SMTP), to validate that the intuition and assumptions introduced here apply to all network traffic.

More generally, some of the problems and tasks raised for future research include:

- Performing unidirectional flow pairing using all vantage points (rather than separately for each vantage point) to tackle route asymmetries.
- Evaluation of RR (remote-remote) dependencies and establishing an asset dependency ranking.
- Aggregation of connection chains by common suffixes (rather than by all chain members).
- Evaluation of causal connection chain membership via delay distributions (similar to Orion [17], Sherlock [11], and [8]), rather than a fixed threshold.
- Selection of an appropriate peer popularity threshold for the peer pre-filtering performed in Chapter 7.
- Selection of an appropriate weighting strategy to apply to the base Jaccard Index.
- A closer study of connections of short duration (i.e., 0 ms, 1 ms, 2 ms, ...) to characterize their cause and to establish an appropriate data sanitization strategy for this class of connections.

Interested readers and researchers are encouraged to explore any and all of these tasks.

Chapter 9

Conclusion

Enterprise networks are large, dynamic, and complex; system, network, and security administrators face the constant challenge of trying to track the networked assets they have and how these assets interact and depend on one another. With this limited visibility, and, consequently, the lack of a strong grasp of these issues, administrators are ill-equipped to tackle problems when they arise.

This thesis presents CANDID, a passive network-based approach for inferring asset relationships and dependencies in enterprise networks. In this work, three avenues of analysis are presented for inferring dependencies and detecting instances of asset load balancing, and each of these solutions is accompanied by a case study centered around NetFlow records for LBNL SMTP traffic.

Future work targets exploration of a new hybrid approach that introduces peer connection distributions and blends the individual strategies presented in this work together into a comprehensive solution.

Bibliography

[1] Cisco NetFlow. http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html.

[2] Cisco Systems Visual Networking Index (VNI) 2011-2016 Forecast Highlights. http://www.cisco.com/web/solutions/sp/vni/vni_forecast_highlights/index.html.

[3] fprobe: a libpcap-Based NetFlow Probe. http://fprobe.sourceforge.net.

[4] NetFlow iptables Module. http://sourceforge.net/projects/ipt-netflow/.

[5] Using SCTP with IPFlow. http://www.ipflow.utc.fr/index.php/SCTP, September 2005.

[6] Stream Control Transmission Protocol (RFC 4960). http://tools.ietf.org/html/rfc4960, September 2007.

[7] memcached - What is This Thing? http://code.google.com/p/memcached/wiki/NewOverview, August 2011.

[8] Aguilera, M. K., Mogul, J. C., Wiener, J. L., Reynolds, P., and Muthitacharoen, A. Performance Debugging for Distributed Systems of Black Boxes. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles* (October 2003), SOSP, ACM, pp. 74–89.

[9] Aiello, W., Kalmanek, C., McDaniel, P., Sen, S., Spatscheck, O., and Van der Merwe, J. Analysis of Communities of Interest in Data Networks. In *Passive and Active Network Measurement* (2005), C. Dovrolis, Ed., vol. 3431 of *Lecture Notes in Computer Science (LNCS)*, Springer Berlin Heidelberg, pp. 83–96.

[10] Bahl, P., Barham, P., Black, R., Chandra, R., Goldszmidt, M., Isaacs, R., Kandula, S., Li, L., MacCormick, J., Maltz, D. A., Mortier, R., Wawrzoniak, M., and Zhang, M. Discovering Dependencies for Network Management. In *The 5th ACM SIGCOMM Workshop on Hot Topics in Networks* (November 2006), HotNets, ACM, pp. 97–102.

[11] Bahl, P., Chandra, R., Greenberg, A., Kandula, S., Maltz, D. A., and Zhang, M. Towards Highly Reliable Enterprise Network Services via Inference of Multi-Level Dependencies. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (August 2007), SIGCOMM, ACM, pp. 13–24.

[12] Barham, P., Black, R., Goldszmidt, M., Isaacs, R., MacCormick, J., Mortier, R., and Simma, A. Constellation: Automated Discovery of Service and Host Dependencies in Networked Systems. Tech. Rep. MSR-TR-2008-67, Microsoft Research, 2008.

[13] Berthier, R., Cukier, M., Hiltunen, M., Kormann, D., Vesonder, G., and Sheleheda, D. Nfsight: NetFlow-based Network Awareness Tool. In *Proceedings of the 24th Large Installation System Administration Conference* (November 2010), LISA, USENIX Association.

[14] Brocade Communications Systems, Inc. Brocade ServerIron ADX 1000, 4000, and 10000 Application Delivery Switches. `http://www.brocade.com/downloads/documents/data_sheets/product_data_sheets/serveriron-adx-ds.pdf`, December 2012. Datasheet.

[15] Brown, A., Kar, G., and Keller, A. An Active Approach to Characterizing Dynamic Dependencies for Problem Determination in a Distributed Application Environment. In *Proceedings of the IEEE/IFIP International Symposium on Integrated Network Management* (May 2001), IEEE, pp. 377–390.

[16] Caracaş, A., Kind, A., Gantenbein, D., Fussenegger, S., and Dechouniotis, D. Mining Semantic Relations Using NetFlow. In *The 3rd IEEE/IFIP International Workshop on Business-driven IT Management* (April 2008), BDIM, IEEE, pp. 110–111.

[17] Chen, X., Zhang, M., Mao, Z. M., and Bahl, P. Automating Network Application Dependency Discovery: Experiences, Limitations, and New Solutions. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation* (December 2008), OSDI, USENIX Association, pp. 117–130.

[18] Cisco Systems, Inc. NetFlow Reliable Export with SCTP. `http://www.cisco.com/en/US/docs/ios/netflow/configuration/guide/nflow_export_sctp.pdf`, June 2006.

[19] Cisco Systems, Inc. Migrating from Traditional to Flexible NetFlow. `http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6555/ps6601/ps6965/white_paper_c11-545581.pdf`, June 2009. White Paper.

[20] Cisco Systems, Inc. Cisco ACE Application Control Engine Module for Cisco Catalyst 6500 Series Switches and Cisco 7600 Series Routers.

http://www.cisco.com/en/US/prod/collateral/modules/ps2706/ps6906/product_data_sheet0900aecd8045861b.pdf, July 2010. Datasheet.

[21] Cisco Systems, Inc. Cisco IOS NetFlow Version 9 Flow-Record Format. http://www.cisco.com/en/US/technologies/tk648/tk362/technologies_white_paper09186a00800a3db9.pdf, May 2011. White Paper.

[22] Cisco Systems, Inc. Introduction to Cisco IOS ® NetFlow. http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6555/ps6601/prod_white_paper0900aecd80406232.pdf, May 2012. White Paper.

[23] Dechouniotis, D., Leontiou, N., Dimitropoulos, X., Kind, A., and Denazis, S. Unveiling the Underlying Relationships Over a Network for Monitoring Purposes. *International Journal of Network Management 19*, 6 (2009), 513–526.

[24] Dell SonicWALL, Inc. SonicOS 5.8: NetFlow Reporting. https://www.sonicwall.com/app/projects/file_downloader/document_lib.php?t=PG&id=7, January 2012.

[25] Dewaele, G., Himura, Y., Borgnat, P., Fukuda, K., Abry, P., Michel, O., Fontugne, R., Cho, K., and Esaki, H. Unsupervised Host Behavior Classification from Connection Patterns. *International Journal of Network Management 20*, 5 (2010), 317–337.

[26] Erman, J., Mahanti, A., Arlitt, M., and Williamson, C. Identifying and Discriminating Between Web and Peer-to-Peer Traffic in the Network Core. In *Proceedings of the 16th International Conference on World Wide Web* (May 2007), WWW, ACM, pp. 883–892.

[27] Estan, C., Keys, K., Moore, D., and Varghese, G. Building a Better NetFlow. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (August/September 2004), SIGCOMM, ACM, pp. 245–256.

[28] F5 Networks, Inc. BIG-IP Local Traffic Manager. http://www.f5.com/pdf/products/big-ip-local-traffic-manager-ds.pdf, 2013. Datasheet.

[29] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. Hypertext Transfer Protocol -- HTTP/1.1 (RFC 2616). http://tools.ietf.org/html/rfc2616, June 1999.

[30] Fonseca, R., Porter, G., Katz, R. H., Shenker, S., and Stoica, I. X-Trace: A Pervasive Network Tracing Framework. In *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation* (April 2007), NSDI, USENIX Association, pp. 271–284.

[31] Huawei Technologies Co., Ltd. NetStream (Integrated) Technology White Paper. http://enterprise.huawei.com/ilink/enenterprise/download/HW_201022, September 2012. White Paper.

[32] Juniper Networks, Inc. Juniper Flow Monitoring. http://www.brocade.com/downloads/documents/data_sheets/product_data_sheets/serveriron-adx-ds.pdf, March 2011. Application Note.

[33] Kandula, S., Chandra, R., and Katabi, D. What's Going On? Learning Communication Rules in Edge Networks. In *Proceedings of the ACM SIGCOMM Conference on Data Communication* (August 2008), SIGCOMM, ACM, pp. 87–98.

[34] Kannan, J., Jung, J., Paxson, V., and Koksal, C. E. Semi-Automated Discovery of Application Session Structure. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement* (October 2006), IMC, ACM, pp. 119–132.

[35] Kannan, J. K., Jung, J., Paxson, V., and Koksal, C. E. Detecting Hidden Causality in Network Connections. Tech. Rep. UCB/EECS-2005-30, EECS Department, University of California, Berkeley, December 2005.

[36] Karagiannis, T., Papagiannaki, K., and Faloutsos, M. BLINC: Multilevel Traffic Classification in the Dark. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (August 2005), SIGCOMM, ACM, pp. 229–240.

[37] Kim, H., Claffy, K., Fomenkov, M., Barman, D., Faloutsos, M., and Lee, K. Internet Traffic Classification Demystified: Myths, Caveats, and the Best Practices. In *Proceedings of the ACM CoNEXT Conference* (December 2008), CoNEXT, ACM, pp. 11:1–11:12.

[38] Kind, A., Gantenbein, D., and Etoh, H. Relationship Discovery with NetFlow to Enable Business-Driven IT Management. In *The First IEEE/IFIP International Workshop on Business-Driven IT Management* (April 2006), BDIM, IEEE, pp. 63–70.

[39] Kleinberg, J. M. Authoritative Sources in a Hyperlinked Environment. *Journal of the ACM (JACM) 46*, 5 (September 1999).

[40] Kohno, T., Broido, A., and Claffy, K. Remote Physical Device Fingerprinting. *IEEE Transactions on Dependable and Secure Computing 2*, 2 (2005), 93–108.

[41] McHugh, J., McLeod, R., and Nagaonkar, V. Passive Network Forensics: Behavioural Classification of Network Hosts Based on Connection Patterns. *SIGOPS Operating Systems Review 42*, 3 (April 2008), 99–111.

[42] Moore, A. W., and Papagiannaki, K. Toward the Accurate Identification of Network Applications. In *Proceedings of the 6th International Workshop on Passive and Active Network Measurement* (March/April 2005), PAM, Springer, pp. 41–54.

[43] Natarajan, A., Ning, P., Liu, Y., Jajodia, S., and Hutchinson, S. E. NSDMiner: Automated Discovery of Network Service Dependencies. In *Proceedings of IEEE INFOCOM* (March 2012), IEEE, pp. 2507–2515.

[44] NIST/SEMATECH. Chi-Square Goodness-of-Fit Test. `http://www.itl.nist.gov/div898/handbook/eda/section3/eda35f.htm`, April 2012.

[45] Page, L., Brin, S., Motwani, R., and Winograd, T. The PageRank Citation Ranking: Bringing Order to the Web. Tech. Rep. 1999-66, Stanford InfoLab, November 1999.

[46] Paxson, V. Empirically-Derived Analytic Models of Wide-Area TCP Connections. *IEEE/ACM Transactions on Networking (TON) 2*, 4 (August 1994), 316–336.

[47] Paxson, V. End-to-End Routing Behavior in the Internet. *IEEE/ACM Transactions on Networking 5*, 5 (1997), 601–615.

[48] Paxson, V. Strategies for Sound Internet Measurement. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement* (October 2004), IMC, ACM, pp. 263–271.

[49] Peddycord, III, B., Ning, P., and Jajodia, S. On the Accurate Identification of Network Service Dependencies in Distributed Systems. In *Proceedings of the 26th Large Installation System Administration Conference* (December 2012), LISA, USENIX Association, pp. 181–194.

[50] Phaal, P., Panchen, S., and McKee, N. InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks (RFC 3176). `https://tools.ietf.org/html/rfc3176`, September 2011.

[51] Popa, L., Chun, B.-G., Stoica, I., Chandrashekar, J., and Taft, N. Macroscope: End-Point Approach to Networked Application Dependency Discovery. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies* (December 2009), CoNEXT, ACM, pp. 229–240.

[52] Postel, J. B. Simple Mail Transfer Protocol (RFC 821). `https://tools.ietf.org/html/rfc821`, August 1982.

[53] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. *Numerical Recipes: The Art of Scientific Computing*, 3rd ed. Cambridge University Press, New York, NY, USA, 2007.

[54] Quittek, J., Zseby, T., Claise, B., and Zander, S. Requirements for IP Flow Information Export (IPFIX) (RFC 3917). `http://tools.ietf.org/html/rfc3917`, October 2004.

[55] Robertson, A. The Evolution of the Linux-HA Project. In *Proceedings of the UKUUG LISA/Winter Conference - High-Availability and Reliability* (February 2004).

[56] Sadasivan, G., Valluri, V., and Djernaes, M. Cisco Systems NetFlow Services Export Version 9 (RFC 3954). `http://tools.ietf.org/html/rfc3954`, October 2004.

[57] sFlow.org. Traffic Monitoring Using sFlow ®. `http://www.sflow.org/sFlowOverview.pdf`, 2003.

[58] Tan, G., Poletto, M., Guttag, J., and Kaashoek, F. Role Classification of Hosts Within Enterprise Networks Based on Connection Patterns. In *Proceedings of the USENIX Annual Technical Conference* (June 2003), ATC, USENIX Association, pp. 15–28.

[59] Vogels, W., Dumitriu, D., Birman, K., Gamache, R., Massa, M., Short, R., Vert, J., Barrera, J., and Gray, J. The Design and Architecture of the Microsoft Cluster Service -- A Practical Approach to High-Availability and Scalability. In *Proceedings of the 28th International Symposium on Fault-Tolerant Computing* (June 1998), FTCS, IEEE Computer Society, pp. 422–431.

[60] Wang, S., State, R., Ourdane, M., and Engel, T. FlowRank: Ranking NetFlow Records. In *Proceedings of the 6th International Wireless Communications and Mobile Computing Conference* (June/July 2010), IWCMC, ACM, pp. 484–488.

[61] Wang, S., State, R., Ourdane, M., and Engel, T. Mining Netflow Records for Critical Network Activities. In *Mechanisms for Autonomous Management of Networks and Services*, B. Stiller and F. Turck, Eds., vol. 6155 of *Lecture Notes in Computer Science (LNCS)*. Springer Berlin Heidelberg, 2010, pp. 135–146.

[62] Wessel, P. Critical Values for the Two-Sample Kolmogorov-Smirnov Test (2-Sided). `http://www.soest.hawaii.edu/wessel/courses/gg313/Critical_KS.pdf`.

[63] Wright, C. V., Monrose, F., and Masson, G. M. On Inferring Application Protocol Behaviors in Encrypted Network Traffic. *Journal of Machine Learning Research 7* (December 2006), 2745–2769.