Constellation: automated discovery of service and host dependencies in networked systems

Paul Barham, Richard Black, Moises Goldszmidt, Rebecca Isaacs,
John MacCormick, Richard Mortier, Aleksandr Simma*
Microsoft Research

ABSTRACT

In a modern enterprise network of scale, dependencies between hosts and network services are surprisingly complex, typically undocumented, and rarely static. Even though network management and troubleshooting rely on this information, automated discovery and monitoring of these dependencies remains an unsolved problem. In the system we describe in this paper computers on the network cooperate to make this information available to all users of the network.

Constellation uses machine learning techniques to infer a network-wide map of the complex relationships between hosts and services using little more than the timings of packet transmission and reception. Statistical hypothesis testing on the resulting models provides a guaranteed confidence level for the accuracy of the result. The system is demonstrated against a substantial packet trace from an enterprise network.

1. INTRODUCTION

Shared network services enable great functional richness and flexibility for distributed applications. As a result apparently simple facilities, such as remote file sharing and email, invariably rely on multiple network services ranging from directory functions to authentication. As well as rendering effective management problematic, the size and sophistication of networked systems often leads to security vulnerabilities, inefficient troubleshooting and anomaly detection, and ultimately user frustration. This effect was noted by Lamport in his famous quote '*a distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable*' [10].

This paper presents *Constellation*, a tool that automates discovery of correlations between hosts and network services, and proactively infers a network-wide map of these complex relationships. Constellation is a peer-oriented distributed system in which peers share information about their traffic correlations. These correlations are inferred in a black-box fashion on each computer locally by performing a statistical test. As a result, neither application-specific monitoring devices, nor centralized "Reporting Managers", are required. Consequently, *any* computer in the network can obtain a global view of how its own traffic relies upon, or is produced by, remote services and computers that may not obviously be related.

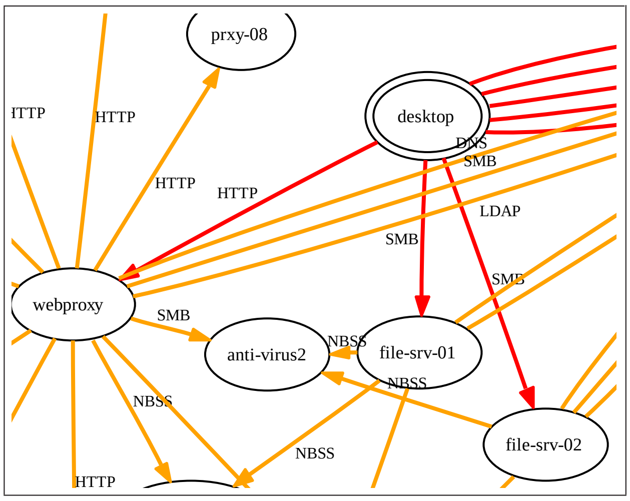


Figure 1: Small fragment of a constellation depicting the transitive correlations of a single computer in an enterprise network.

Figure 1 presents a small portion of a traffic visualization diagram which we term a *constellation*. The machine "desktop" (doubly-circled in the figure) is the root of the constellation. The other nodes represent servers on which the root depends (either directly or indirectly), while the edges reflect the corresponding services that are correlated. This picture was generated by the Constellation GUI, in which hovering the mouse over one edge highlights the related traffic. Constellation builds these correlation graphs using innovative machine learning techniques. Its lightweight, black-box approach allows deployment in a network regardless of the heterogeneity of host operating systems and applications.

The potential for end-users to view what services and computers they depend upon and hence identify the likely source of problems is a key advantage of Constellation over centralized solutions. IT management in a typical enter-

*John is now with Dickinson College, PA. Work done while a Researcher with Microsoft Research. Aleks is with the University of California, Berkeley, CA. Work done while an intern with Microsoft Research.

prise is extremely costly, with entire departments dedicated to maintaining the corporate network and associated infrastructure [9]. Troubleshooting is a particular challenge due to both the complexity and the opacity of systems that comprise products from multiple vendors and are frequently upgraded as technology changes. Furthermore, the problems experienced by end-users may be intermittent and hence difficult for IT support to track down, or can only be rectified after enough data is collected to identify the problem by which time a significant number of users are affected.

Current debugging capabilities for deployed networked systems are unsatisfactory. Individual applications may provide the ability to understand their own behaviour; a few simple general-purpose tools like *traceroute*, *tcpdump* and *ping* exist, while some systems make a variety of data available through SNMP. However, there is still a dearth of tools that enable users, operators and developers to understand the interactions between the systems they come into direct contact with and those not within their immediate view that are part of the infrastructure.

Paper outline

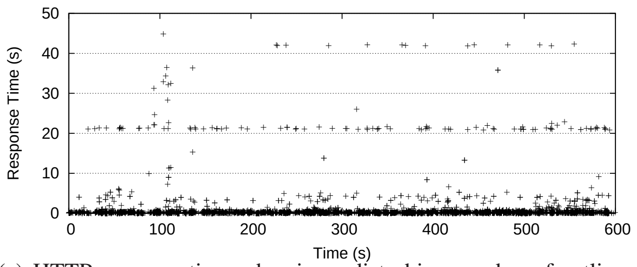
In the next section, we motivate Constellation by describing two case studies, and then in Section 3 introduce its software components, which include the processing that takes place on each computer locally, as well as a distributed search procedure for building network-wide constellations. At the core of the system is a method for learning a model of the relationships between input and output traffic streams, and applying hypothesis testing to this model to determine correlation. We describe this technique in Section 4. Section 5 presents an experimental validation using a carefully selected subset of a real network trace for which we have determined "truth", while Section 6 gives a qualitative evaluation of Constellation's performance across multiple computers using the same trace (for which we do not know absolute truth). Finally, we discuss some implementation issues in Section 7 and review related work in Section 8.

2. USAGE CASE STUDIES

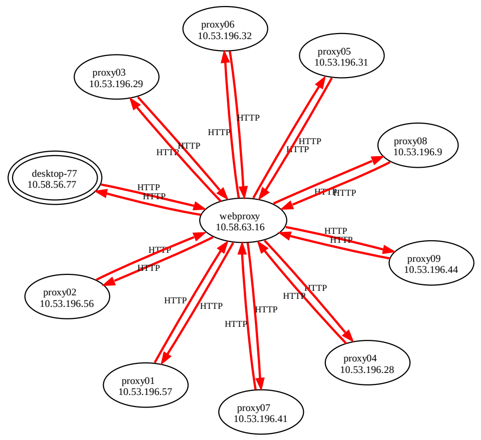
Throughout this paper, experimental analysis and results are obtained using a trace comprising headers and partial payloads of around 13 billion packets collected over a 3.5 week period in 2005 at Microsoft Research Cambridge. Although the capture site, which has approximately 500 networked machines, is physically distant from company headquarters, it is fully within Microsoft's global corporate network and so runs the same services and applications under the same global policy as the rest of the company. The trace covers every packet sent or received by the 500+ nodes on site and captures conversations with over 28000 off-site IP addresses.

2.1 End-user diagnosis of web proxy failure

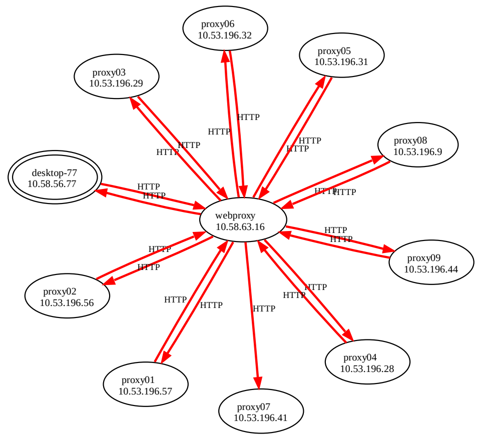
As an example of using Constellation for diagnosis we explore a sudden increase in HTTP response times for some clients that we detected in the trace. Figure 2(a) shows the HTTP response times of all clients during 5 minutes from this period. The plot suggests the presence of a problem with a small but significant proportion of HTTP requests that are observed to take over 20 seconds (and in some cases over 40 seconds) to be served. The way in which we investi-



(a) HTTP response times showing a disturbing number of outliers in bands at 20s and 40s.



(b) Hour 1: normal operations. HTTP requests are forwarded through the first-hop proxy to 9 second-hop proxies.



(c) Hour 2: *proxy07* has stopped responding to HTTP requests, which are still being forwarded to it from *webproxy*.

Figure 2: Investigation of web proxy issues.

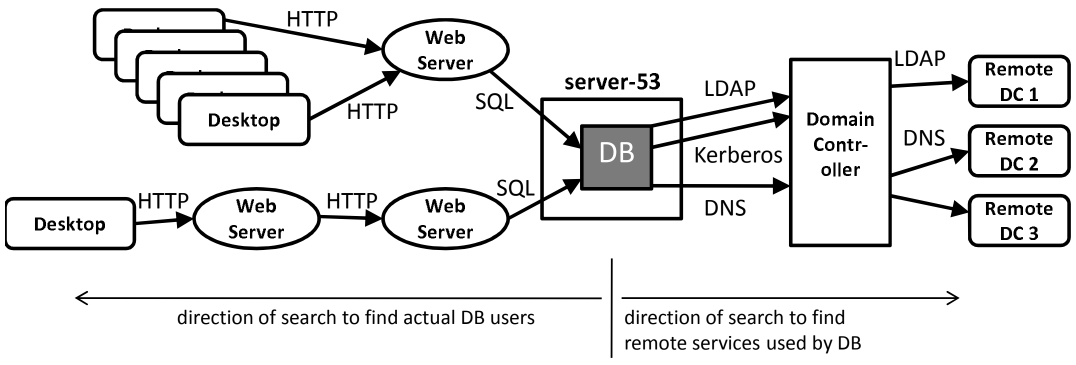


Figure 3: **Using Constellation for reconfiguration planning.**

gated these outlying points is illustrative of how Constellation could be used by an end-user to troubleshoot networking problems experienced on their desktop computer.

The constellations of Figure 2 are generated at the HTTP client *desktop-77*, and show that web traffic that is destined for the internet is directed first to a proxy called *webproxy* and from there forwarded to one of a bank of second-hop proxies denoted *proxy**. In a constellation graph, the arrows denote a correlation, therefore the pictures can be interpreted as showing us that on *webproxy* incoming HTTP traffic from *desktop-77* is correlated with outgoing HTTP traffic to *proxy01*, *proxy02* and so on.[1] Similarly, incoming HTTP traffic from the *proxy** servers, (which will in fact be the response packets), is correlated with the outgoing HTTP response traffic to the client *desktop-77*. These constellations were produced for one hour time periods both before (Figure 2(b)) and during (Figure 2(c)) the occurrence of slow web response times. It is immediately obvious from these pictures that *proxy07* has stopped sending HTTP responses in the second hour and is the most likely cause of the delays. For the human user sitting in front of *desktop-77* the fault can be identified at the time of experiencing the problem by comparing the HTTP-constellation from an hour ago, (when no problems were apparent), with the HTTP-constellation for the current time.

This is a straightforward diagnosis for the end-user given the information exposed by Constellation, but would be trickier for the system administrator to notice and track down. Although some clients will have experienced very slow response times during the outage, the majority received good service from the eight healthy proxies so the problem would have only manifested intermittently. To further complicate analysis of the issue, services other than HTTP, including Winsock control traffic, actually continued to work normally on the malfunctioning proxy during this period. In fact since the same load-balancing algorithm is used by the proxy server for both HTTP and Winsock control (which is used to establish non-HTTP tunnels to the external internet through the gateway *proxy** computers), the two Winsock control constellations from the same client for the same time periods are both identical to the HTTP constellation of Figure 2(b). Hence the only symptom is that one in nine web requests (approximately) experienced poor response times, while the server itself was still working normally for all other types of traffic.

[1]Strictly speaking correlation does not imply causation, e.g. scheduling artefacts could make two otherwise independent packet streams appear correlated. However, in the vast majority of cases that we have examined the presence of correlation strongly suggests a causal relationship.

Having pinpointed the problem source using Constellation, by manual inspection of the traces we were able to track it to an apparent IPSEC misconfiguration which caused *proxy07* to stop responding to TCP SYNs on port 80. After 20 seconds the proxy software apparently times out the `connect()` attempt and tries a different member of the load balancing group. It is impossible to tell how long it took the network administrators at the time to become aware of, to diagnose, and to rectify the problem, however the trace shows that *proxy07* continued to behave badly for over 48 hours.

Note that Constellation is not only empowering the savvy user but automated monitors can now alert the professional IT support organization of changes in the different user constellations enabling a faster, more focused, and more accurate diagnosis.

2.2 Reconfiguration planning

To illustrate how Constellation may be used in practice by a system administrator we use a hypothetical, but realistic, scenario.

Assume the administrator wishes to move a database running on *server-53* to another computer in a different part of the network with minimal disruption to its users. The query log shows only that *server-53* is accessed by several web servers, therefore the administrator cannot tell who the originators of these requests are and hence how many human users might be impacted by the planned reconfiguration. Inspection of logs from the relevant web servers would reveal this information, but in many situations will be a manual process requiring human intervention. For example, the

Type	Service	Peer	cPort	Description
OUTREQ	HTTP	server1	*	HTTP requests to specific server1
INREQ	HTTP	client1	*	HTTP requests from client1
INREQ	HTTP	client1	1234	HTTP requests on a single 5-tuple
INREQ	HTTP	*	*	All HTTP requests (at server)
OUTRES	HTTP	*	*	All HTTP responses (at server)
OUTREQ	SMB	*	*	All file browsing requests
INRES	SQL	dbhost	*	SQL responses from dbhost
OUTREQ	DNS	*	*	All outgoing DNS requests
INREQ	DNS	*	*	Incoming DNS responses (at server)
OUTRES	DNS	*	*	Outgoing DNS responses (at server)
INRES	DNS	*	*	Incoming DNS responses (at client)
...	...	...		...

Table 1: Examples of channel specifications.

database administrator may have to email several web server administrators to request access to, or a search of, their web logs.

For the purpose of this reconfiguration, the administrator may also wish to know which remote services the database makes use of. If *server-53* hosts more than one type of application server, it can be difficult to determine which remote invocations are triggered by database activity specifically. This traffic is likely to belong to network management services, for example we might see various Active Directory authentication services invoked. Such dependencies can be almost impossible to discover manually—they are likely to be either encoded in obscure configuration files or else configured dynamically according to some network-wide policy.

Constellation automates and simplifies finding this information by allowing the administrator to search "backwards" to identify the desktop computers that indirectly access the database on *server-53*, and to search "forwards" to find which remote services the database makes use of. The diagram in Figure 3 illustrates how these forwards and backwards searches might propagate from the database machine *server-53*.

3. BUILDING BLOCKS

The case studies of the previous section described some of the functions provided by Constellation from the point of view of the end-user and the system administrator. In this section we give an overview of the system and describe the mechanisms that enable this functionality.

Each participating host in the Constellation system runs a daemon that is responsible for learning the relationships between the traffic flows it originates or terminates. By querying this information, the system as a whole can explore possible transitive dependencies throughout the enterprise. Figure 4 shows the processing stages of a Constellation daemon: in the following we introduce the role played by each of the system components that are depicted in the diagram.

Channels

To establish a common vocabulary we first introduce a specific notion of a *channel* that we will use throughout the paper. A channel at a given host is a unidirectional flow of network packets sent or received by that host, identified by a direction and role (e.g. INREQ or OUTRES), the service, the remote peer, and the client port. Here, a service could be a protocol, application or network service, frequently determined in practice from the well-known TCP/UDP port number, other information in the packet header, or even from the operating system of the host. See Table 1 for examples.

Aggregation

Channel aggregation enables the model to treat the traffic of multiple peers as coming from or destined for a single entity. This is sometimes, but not always appropriate, depending on the nature of the traffic and the intended use of the resulting constellation.

An example of when aggregation is desirable can be seen in Figure 2. Here it is possible to track correlations for individual clients through the proxy, showing, for example, that client C1 was correlated with servers *proxy02* and *proxy04*, whereas C2 was correlated with *proxy01* and *proxy07*. However, for diagnosing the problem described in Section 2.1, the important information is that clients are correlated with all the upstream servers, since a request from any client might be sent to any server. Therefore on *webproxy* we aggregated all incoming HTTP requests into one channel and all outgoing responses into another (note however that on the client, *desktop-77*, HTTP traffic is not aggregated).

Filtering bursts

Constellation uses packet timing relationships to infer correlations that we actually expect to be exhibited between messages, i.e., between communications at layers above the network stack. Although the use of packets enables the Constellation daemon to be lightweight and non-intrusive, it has the drawback that the message-level semantics can be obscured when a large number of extraneous packets are generated, for example when transferring a large amount of data. Consequently, we use a low pass filter that removes "noise" packets and bursts while keeping the "signal": the details are described further in Section 7.3. This burst filtering is applied to individual flows before channels are aggregated.

Fitting probabilistic models

At the core of the Constellation system is a set of probabilistic models which describe the observed timing relationships between traffic on input channels and traffic on output channels. For scalability, each host maintains current and historical models of its own traffic, and allows other hosts to query this database.

Fitting models is done via the well-known Expectation Maximisation (EM) algorithm [5]. For a set of input channels and output channels we fit models using various priors[2] and distribution families. The EM algorithm finds the best parameters for each model, and by computing the *likelihood* each model assigns to the data, Constellation can select the best of these models.

[2]Initial parameter estimates.

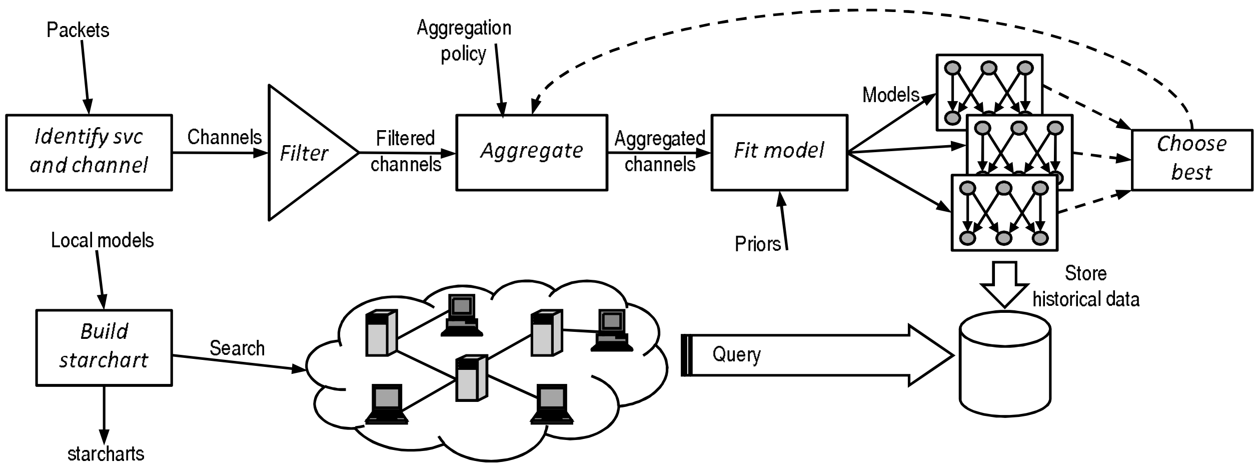


Figure 4: **Processing stages of a Constellation daemon. Packets with the same service and peer are grouped into channels, which are optionally filtered and aggregated. Correlations between input and output channels are then inferred from the packet timestamps by fitting probabilistic models. Each host exports current and historical correlation maps through a query interface, enabling a network-wide constellation to be constructed from any host by means of a directed search.**

Using the models, Constellation decides whether there is sufficient evidence that packets on an input channel result in packets on an output channel. Similarly, it can also determine when a model no longer accurately describes the currently observed traffic. Both these tasks are achieved using the likelihood score and statistical hypothesis testing.

Model selection

Due to the nature of machine learning algorithms, we must fix a particular aggregation strategy before fitting the model. As explained earlier, there are often multiple choices for aggregation. The model likelihood score can also be used to compare models fitted using different aggregation strategies or distributions and choose the one which best captures the traffic (the dotted lines in Figure 4 indicate that this component is under development and is not discussed further in this paper).

Queries

Using the model database, the Constellation daemon supports two primitive queries:

- Find outputs correlated with a specified input
- Find inputs correlated with a specified output

Inputs and outputs are specified using the channel syntax described earlier, and can include wildcards. Queries can be restricted to a particular time range, and results can be limited to the N strongest matches, or matches with a confidence grater than C. Executing a query is similar to running an SQL `SELECT` over a table of input to output correlation scores. The results tell us which channels are related on each machine individually.

Building a constellation

The Constellation client application can also perform a network-wide search to find related traffic, transitively following selected correlations from machine to machine in a breadth-first fashion, as well as following local "temporal" correlations on a single machine. The result is a directed graph containing services and computers that are transitively related to the traffic of the seed computer.

Many of the constellation diagrams throughout this paper, such as in Figures 1 and 2, show the graph generated by a *forwards* search initalized from outgoing traffic of the seed computer. It is also often useful to run the search *backwards* to find all possible transitive causes of a particular output request channel. We make use of the latter technique later in this paper (Section 6.1) to explore the relationship between name resolution and web browsing.

Both forwards and backwards search can involve following temporal correlations on the same machine. These occur when an input response channel is correlated with an output request channel, and indicate an ordering on service invocations by the client. Another common idiom for channel relationships is nesting, where incoming requests trigger outgoing requests. Nesting occurs on a hierarchical caching server such as the web proxy described earlier or in a typical DNS deployment.

Correlation scores are comparable across hosts (because they represent a measure of confidence), and so the simplest termination criterion for the search procedure is to stop at the host that has no outgoing channels correlated with the relevant incoming channel. More sophisticated schemes include weighting correlation strength according to the number of hops distant from the seed, or computing a cumulative probability score of the complete path from the seed to decide

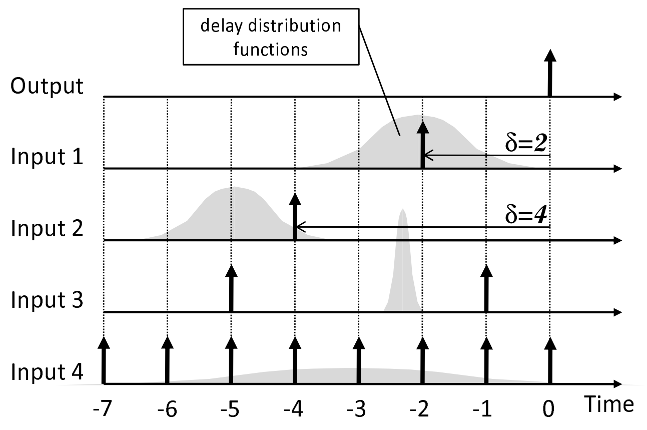


Figure 5: Constellation compares the relative probabilities that candidate inputs packets caused each output.

whether an edge should be added to the graph.

4. HOST TRAFFIC MODELS

Constellation is concerned with inferring the configuration and traffic interactions of *individual* machines, and then synthesizing this information to provide a global view. This section tries to impart an intuitive understanding of the model used to represent these local traffic interactions and to explain how it differs from previous work. The model is a crucial component of the system and so we also present a more rigorous introduction in Section 4.1. Further details, including formal properties of the model, are beyond the scope of this paper (which is focused on the system as a whole), but can be found in a separate publication [16].

Constellation learns the interactions between services by observing the timing relationships between their input and output traffic. In Figure 5, the output packet at time $t = 0$ may have been caused by any (or none) of the four inputs. Constellation assigns a probability to each candidate packet according to the time delta between input and output and a delay distribution function which is fitted to the observed samples.

In this example, the packet on input 1 with $\delta = 2$ is the most likely candidate. The packet on input 2 has $\delta = 4$ but the majority of observations we have seen previously have a $\delta \approxeq 5$. Input 3 is usually seen very close to $\delta = 2.3$, whilst input 4 packets are spread thinly across the whole window. Constellation uses the Expectation-Maximisation (EM) [5] algorithm to iteratively determine the most likely patterns of inputs and outputs.

An important difference between Constellation and related work is that the per-host models fitted by Constellation consider all recent input packets using Bayesian reasoning to identify the best explanation of each output packet. Earlier systems have used pairwise correlation techniques to determine whether a single input and output flow have similar timing patterns: Aguilera *et. al.* [1] used convolution of incoming and outgoing email timestamps to identify related messages that exhibited the nesting correlation idiom described in the previous section. Sherlock [2] collects pairwise co-occurrence[3] statistics at each host. These are then submitted to a central inference engine that aggregates observations across the enterprise and applies Bayesian reasoning to identify the most common configurations (e.g. primary and backup DNS servers) or the most probable root cause of problems. WAP5 [15] fits application-level input and output *messages* to a predetermined delay distribution that is parameterized in advance for specific applications.

Of course, output traffic is rarely explained perfectly by input traffic, and frequently there is no relationship at all (when output packets are "spontaneously" generated). To quantify the degree of correlation between an input I and an output O, we fit two models: one using all of the input channels, and another using all of the channels except I. The *difference* in accuracy of these two models at predicting the output traffic is a good metric of the correlation between I and O. In fact, Constellation applies standard statistical hypothesis tests to determine if the two models differ *significantly*, and this allows us to control the rate of false positives. This is a second major difference from previous work.

As a real-world example, Figure 6(a) shows the traffic patterns observed on a web server over the course of a typical hour. Each horizontal line shows request and response packets of a particular protocol. Although most human observers can observe subtle timing correlations between some of the eighteen protocols, there are also busy periods where it is difficult to make out any clear pattern. Even though individual interactions may frequently look ambiguous, evidence from the rest of the trace rapidly accumulates to identify the most probable cause.

Figure 6(b) shows a small excerpt of the web server plot annotated with arrows which show the explanation Constellation has attributed to each output packet. Constellation has determined that the HTTP request at the bottom left is the most probable explanation of the LSA (authentication) request, the SQL query and the SMTP conversation. Inspection of the configuration of this web server revealed that it hosts a site used for online booking of taxis which maintains bookings in a database and sends email when a new booking is made.

4.1 CT-NOR

The procedure we use for automatically discovering the dependencies[4] between the input channels and the output channels is well grounded in probability modeling and statistics. It comprises two steps. First we learn from the observed data a probability model of the interactions between inputs

[3] i.e. the number of input and output packets within 10 ms of each other.

[4] Note that *dependence* is used here in the statistical sense to mean that by knowing the inputs we can better predict the outputs.

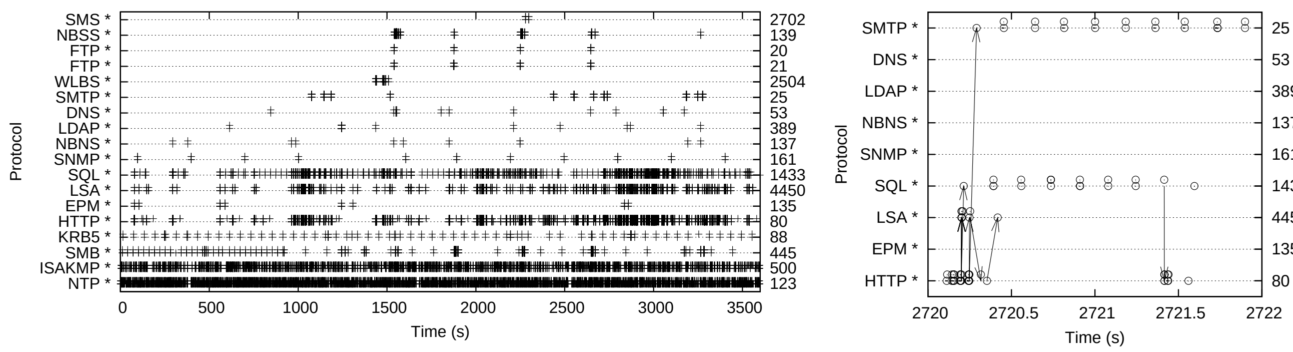


Figure 6: **Traffic patterns at a web server a) 1 hour overview, b) detail of HTTP request causing SQL and SMTP activity.**

and outputs, and then we use statistical hypothesis testing to establish the relevance of each input channel to each output channel.

Constellation uses a probabilistic model called *Continuous Time NoisyOr* (CT-NOR)[16], which considers a single output channel on a given host and simultaneously analyzes all of this host's input channels to determine which channel(s) best explain the actual observed occurrence of the output packets. Under this model, if o_l denotes the time of an output event then the probability that packet k in input channel j *generated* output packet l is distributed as a Poisson process with intensity parameter $p_k^{(j)}(o_l) = \pi^{(j)} f_\theta(o_l - i_k^{(j)})$.

In this equation $\pi^{(j)}$ represents the average number of output events that we expect each input event on channel j to be responsible for, and f_θ is the distribution of the delay between an input and the output events caused by it. This function will take as its argument the delay between the time of the output o_l and the time of the input $i_k^{(j)}$. Note that this makes intuitive sense: the probability that a given input packet caused a given output packet depends on both the expected number of packets it generates and the "distance" in time between them. The function $f_\theta(\Delta t)$ provides us with the opportunity of encoding prior information regarding the expected shape of the delay between the input and the output channels.

The parameter $\pi^{(j)}$ and the parameters of the function $f_\theta(\Delta t)$ are fitted automatically. To build the joint probability model of the output packets we recall that given a set of independent Poisson processes (denoted as *PP*) we can use the sum of their intensities and write $\{o_k\} \sim PP(\sum_j \sum_k p_k^{(j)}(\cdot))$ as the probability of the set of n outputs $\{o_k\}$, $1 \leq k \leq n$. Intuitively, given this independence between input channels, the model considers the packets in the output channel to be caused by the presence of any (a disjunction) of input packets in the input channels (with some uncertainty). Hence the name *NoisyOr*.

We are now ready to write the probability of observing a set $\{o_l\}$ of outputs, given a set of inputs i on all the channels

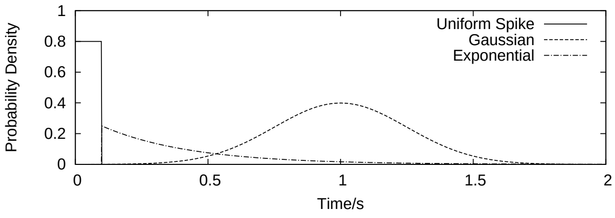


Figure 7: **Example delay distribution functions** $f_\theta(\Delta t)$

j. Let $\lambda = \sum_j \sum_k \pi^{(j)}$

$$P(o_1, \ldots, o_n | i) = \lambda^n \cdot e^{-\lambda} \prod_l \sum_{jk} \frac{\pi^{(j)} f_\theta(o_l - i_k^{(j)})}{\lambda} \qquad (1)$$

Eq. 1 is called the likelihood equation and it is important for both automatically fitting the parameters of the model and for discovering correlations.

For learning (fitting) the parameters, $\pi^{(j)}$ and the ones in $f_\theta(\Delta t)$, we find the values that maximize Eq. 1, as these parameters maximize the probability that our model generated the data we are observing. It is in this precise sense that our model best explains the data observed. To perform this maximization and fitting effectively, we follow standard machine learning and statistical techniques and use the Expectation Maximization algorithm [5].

The CT-NOR model is arguably the simplest probabilistic model that comprises the following characteristics: 1) it lends itself to statistical techniques so that we can offer guarantees in terms of the number of false positives in the dependency discovery task, 2) it takes into account the interactions between all input channels and 3) it enables the incorporation of expert knowledge on the structure of the expected delay distribution through the function $f_\theta(\Delta t)$.

4.1.1 Delay distributions

In our experiments and characterizations we determined two particularly useful delay distributions ($f_\theta(\Delta t)$). The first is a linear mixture of a uniform distribution on a small window, and a decaying exponential. The uniform component captures almost instantaneous forwarding dependencies (order of a millisecond) while the exponential component captures more distant interactions resulting from queuing or processing. The second useful distribution was a similar mixture, but with a Gaussian component for the more "distant" interactions (see figure 7). All the parameters of these distributions are fitted automatically whilst maximizing Eq. 1.

4.1.2 *Confidence thresholds*

To discover which input channels are relevant to the output channel we rely on statistical hypothesis testing on the the parameter $\pi^{(j)}$. Specifically, we use the Likelihood Ratio Test (LRT) [17]. The null hypothesis H_0 is that for a specific $j = J$, $\pi^{(j)} = 0$ (i.e. input J is not responsible for any output packets). To test the hypothesis we maximize Eq. 1 twice, once with the full set of inputs, and once with $\pi^{(j)} = 0$; if Λ is the ratio of the model likelihoods, then $2log(\Lambda)$ is distributed as χ^2 with one degree of freedom.

The result of this test is a "p-value" which gives (roughly speaking) the probability that a more extreme result would have occurred if the null hypothesis were true. To decide whether the input channel and the output channel are dependent, this p-value is compared with a χ^2 confidence threshold. Anything below the threshold is rejected and the channels are declared correlated.

4.1.3 *Multiple hypothesis testing*

Although the p-value tells us something about the probability that a specific case will be falsely rejected, when we construct a constellation we perform a large number of hypothesis tests and the probability of making an error could rapidly mount up. A concrete example illustrates this best: suppose we perform 1000 hypothesis tests of which we know that 200 are truly dependent channels and 800 are non-dependent, and that our test rejects a number of null hypotheses equal to the number of genuinely dependent cases, namely 200. With a p-value threshold of 5%, on average 40 of these (5% of 800) will be incorrect decisions—i.e. around 20% of our decisions are incorrect.

In practice, what we really care about is, *"Amongst the cases that my tests say are dependent, what is the proportion of incorrect decisions?"* This is known as the False Discovery Rate (FDR). Statisticians have been confronting this problem recently in the domain of genomics and have come up with various procedures for dealing with large numbers of hypothesis tests. In this work we follow the Benjamini-Hochberg procedure described in [4].

Using the p-values from the CT-NOR test in combination with the FDR procedure, we can compute suitable thresholds on each host for rendering constellations, with upper bounds (in expectation) on the number of false dependencies that will be included.

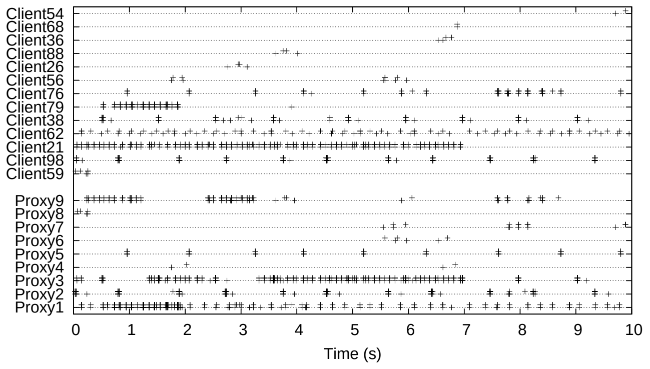


Figure 8: Activity plot of HTTP traffic at the caching web proxy. Clients are shown at the top and the nine upstream proxies in the bottom of the figure, with each point representing an HTTP packet either received or transmitted.

5. EXPERIMENTAL VALIDATION

In the field of machine learning "ground-truth" refers to those situations where we know the correct answers. The ground-truth for correlations in a real network is, in general, hard to extract. Relationships are often hidden inside the configuration files or source code of individual applications, and caching and load balancing can further obscure the actual underlying dependency. Fortunately, forwarding servers for a well-understood protocol can provide an exception to this situation, because received packets can be matched with transmitted packets by inspection of the payload.

In this section we validate the correlation test using forwarded HTTP traffic at a caching proxy server. We extracted ground-truth from a one hour period of the trace (10-11am on a Tuesday) by deep inspection of HTTP packets forwarded by the machine *webproxy* (which we also examined in the case study of Section 2.1). All HTTP requests to the external internet from machines on the trace site are directed to the proxy server, which will either respond from its own cache, or, as the constellation of Figure 2(b) shows, forward the request to one of an array of nine upstream proxies.

Because the web proxy performs load-balancing across all machines in the proxy bank, to avoid the uninteresting situation where every client is correlated with each of the upstream proxies (i.e. it would be impossible to make a mistake), we filtered the HTTP traffic by selectively dropping packets so each client uses no more than four or five of the proxies.

The ground-truth data set formed by this filtered HTTP traffic is an example of a particularly challenging environment for the correlation test. On the one hand the ground-truth is straightforward because it consists of only one style of correlation (nesting), the server is fairly lightly loaded, and requests are load-balanced across upstream servers at a coarse granularity. However on the other hand, the presence

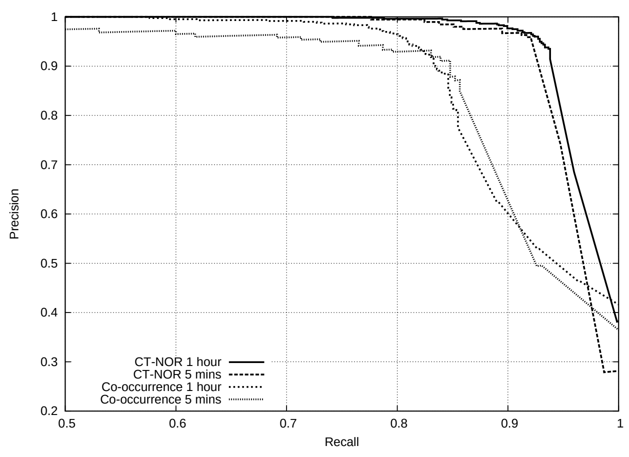


Figure 9: Ground-truth precision-recall for both CT-NOR and co-occurrence over the first 5 minutes of the hour and over the full hour. Note that for clarity the x-axis originates at 0.5 (at less than 50% recall we see almost 100% precision in all cases).

of a large amount of multiplexing between traffic to and from the 95 clients and 9 upstream proxies makes these relationships difficult to tease out at smaller timescales.

Figure 8 contains a 10 second segment of the HTTP traffic on *webproxy*. Since most clients use HTTP/1.0 there are frequently four simultaneous TCP connections per client and the median connection duration is 0.5s. The number of packets per connection shows the expected mix of short-lived connections and long downloads [11]. In contrast, HTTP/1.1 is frequently used between *webproxy* and each *proxy**. The median lifetime of these connections is close to 2 seconds and there are on average 30 connections active simultaneously between each pair of machines. Some of these connections transfer only a single HTTP object, whilst others carry many requests from multiple clients. A further complicating factor is that HTTP traffic comprises only 26% of the half a million packets that make up the data set. With an observed cache hit rate of 29%, this results in 4,246 instances of a forwarded HTTP request or response at an average of 1.18 per second over the hour.

5.1 Precision-recall

The statistical hypothesis test described in Section 4 generates a p-value, or confidence that a particular input and output are related. Comparing this score against a fixed threshold allows the user to arbitrarily trade off the accuracy of decisions against the number of dependencies identified. To assess the effectiveness of the test we borrow a technique from the field of information retrieval and plot the *precision* (the fraction of answers returned which are correct) against the *recall* (the fraction of all correct answers which were returned) as we vary the threshold from zero to infinity.

Figure 9 shows the precision-recall curve for CT-NOR when considering the full hour of the data set, as well as the first five minutes only. For comparison, we also include the results obtained when co-occurrence is used to decide correlation, configured with a window of 10 ms[2]. The knees of these curves suggest the best possible operating point for the tests over this data set, which for CT-NOR would be at just over 94% recall and 96% precision, and for co-occurrence at around 85% recall and 91% precision. From a machine learning perspective this represents a substantial difference in accuracy which is amplified when performing large numbers of hypothesis tests.

Trace Length	Target FDR	Num Tests	p-value cutoff	Actual FDR	Precision	Recall
5min	5%	220	5.2%	2.4%	97.6%	88.1%
	10%	220	11%	2.8%	97.2%	89.5%
1hour	5%	667	4.7%	2.9%	97.1%	91.3%
	10%	667	9.4%	3.4%	96.6%	92.1%

Table 2: Automatically chosen p-value thresholds and the resulting Precision/Recall and FDR statistics.

5.2 False discovery rates

Section 4.1.2 explained how the association of a confidence score with the outcome of the test provides a mechanism for choosing the correlation threshold. In terms of the precision-recall plot of Figure 9, this selects the operating point on the curve, and ideally we would automatically choose the "best" point irrespective of the duration, volume or type of traffic.

For the co-occurrence test we have no automatic way to choose an operating point. The scores are normalised relative to the "expected number of accidental collisions"[5], but do not represent p-values and hence can vary greatly from one dataset to another. To achieve a precision of 90% requires a threshold of 2.56 for the 5 minute trace but 3.89 for the one hour trace. Using the threshold from the 5 minute trace on the one hour trace results in a precision of just 44%!

In contrast, the CT-NOR test scores are p-values on an absolute scale and are therefore comparable across machines and datasets. For each of the ground-truth datasets we used the Benjamini-Hochberg procedure to choose an operating point (i.e. p-value threshold) guaranteeing false discovery rate less than 5% and 10%. Table 2 shows the resulting Precision/Recall and FDR statistics. In all cases we identify an operating point close to the optimum and stay well below the target error rate.

Using the p-values from the CT-NOR test in combination with the BH procedure, we can compute suitable thresholds on each host for rendering constellations, with guarantees (in expectation) on the number of false dependencies that will be included.

5.3 Search pruning efficiency

Another evaluation method compares the number of hosts considered when fault-finding. Figure 10 shows the aver-

[5]We use the same normalisation technique as described in [2].

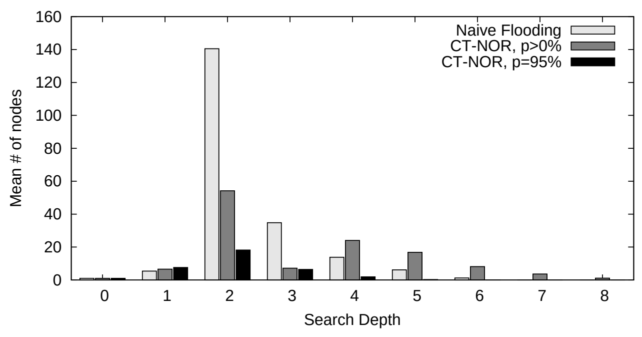


Figure 10: Pruning the search space using Constellation

age (over searches rooted at all hosts) number of hosts considered, against search depth, for three different algorithms. Note that, due to the nature of the trace, a large fraction of the nodes found early in the search will be offsite, preventing the search from continuing; with full network visibility we would expect the differences to be even more marked.

The naïve flooding is a baseline algorithm that searches every host to which a request has been made in the preceeding hour. Its inclusion illustrates (by its rapid growth) both the difficulty faced by administrators without tools such as Constellation, and (by the rapid curtailment) the limitations of the dataset. The middle bars are produced by a search using a co-occurrence scheme (in fact implemented by setting a CT-NOR threshold >0). The right-hand bars show the performance of the Constellation CT-NOR algorithm with a 95% confidence threshold. It can readily be seen that using Constellation to constrain the search to relevant hosts dramatically prunes the search space.

6. QUALITATIVE EVALUATION

This section describes a qualitative investigation of how a higher-level tool might make use of constellations. Figure 11 shows the entire constellation seeded from the computer *desktop-36* for a one hour period. The structure of this constellation is typical for client machines in the trace, with a large number of services invoked at the on-site domain controller, which in turn makes requests of upstream domain controllers for various authentication (LSA), security (Kerberos), RPC (EPM), name resolution (DNS) and directory (LDAP) services. The picture also shows that *desktop-36* is invoking services on email, web and file servers, with onwards dependencies from some of these machines. Note that the constellation actually contains both temporal and nesting correlations, however the graphical representation does not show this explicitly as it would make the picture hugely complicated.

When considering entire constellations we lack the means to extract ground-truth from the traces except for a very small set of services on individual computers. However, we can draw on our domain knowledge about likely relationships between hosts and services in the network to explore

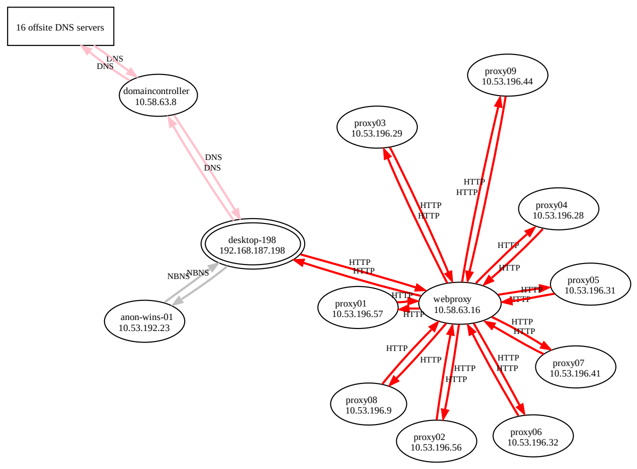


Figure 12: HTTP correlation constellation for one hour.

qualitatively whether Constellation exposes these relationships and in this section we examine three such scenarios.

All the constellations in this section were generated using CT-NOR with an exponential delay function and FDR of 5% at each host.

6.1 Name resolution and web browsing

The relationship between HTTP and DNS is well-known and does not need further elaboration here. We expect to find evidence of this in our traces, as well as a less universally common correlation between Netbios Name Service (NBNS) and HTTP, which is present due to the particular DNS and Active Directory configuration in the network.

To investigate the relationship between HTTP and name resolution we first examine the CT-NOR correlation models of all hosts in the network. Over a time period of one hour we find DNS response-to-HTTP request correlations for 47% of clients sending HTTP requests, while over 24 hours this proportion rises to 83%. The increase is explained by the effects of client-side caching of DNS name lookups (typically 15 minutes). The relationship between the Netbios name resolution protocol and HTTP requests is also significant: we find that 11% of clients issuing HTTP requests have the NBNS response-to-HTTP request correlation in their single hour constellation and 33% in their constellation for one day.

Secondly, we use the backwards mode of the constellation building algorithm, described in Section 3, to see how Constellation would answer the question “what hosts and services are required in order to browse the web?” We generate backwards constellations from a seed edge that is the outgoing HTTP request channel to the proxy server, and find that the resulting graphs fall into 3 main groups, depending on whether just HTTP is involved, or if name resolution using DNS or WINS is also used.

It is apparent from this that the essential infrastructure for web browsing from our site comprises the 10 proxy servers

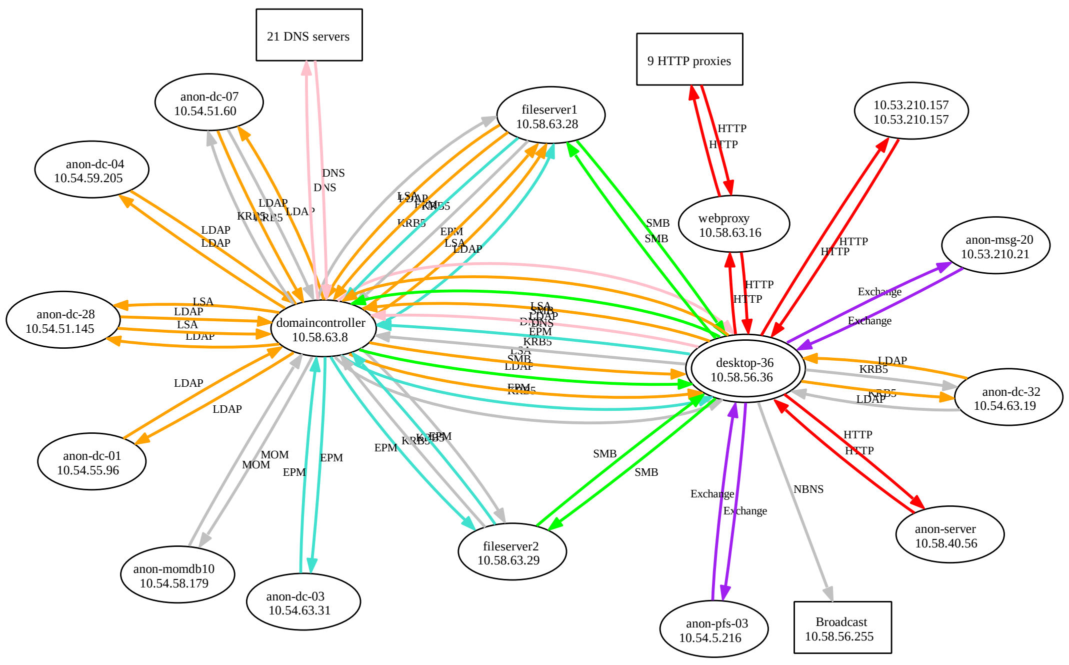


Figure 11: A constellation rooted at the host *desktop-36* highlights the large number of services invoked on the Domain Controller (*domaincontroller*) by the client, both by direct connection and also indirectly with transitive correlations via the two on-site file servers (*fileserver1* and *fileserver2*). Note that all the leaf nodes represent uninstrumented, off-site servers, and therefore do not show any further onward transitive correlations.

and the local domain controller. A large number of other DNS servers (168 over the course of the day) also appear in over 60% of these constellations, while 25 constellations contain the WINS servers used for Netbios Name Resolution. Figure 12 shows a typical “HTTP dependency” constellation containing DNS and WINS servers as well as the HTTP proxies.

These findings reassure us that Constellation observes the correlations we expect to see in this network, as well as shedding light on interesting aspects such as the volume of NBNS name lookup that took place for web browsing.

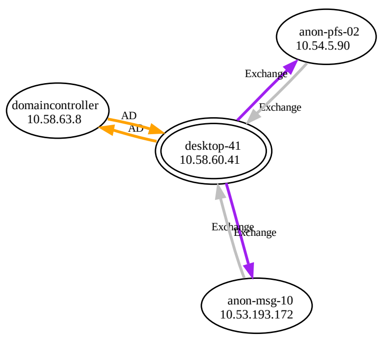
6.2 Email

The correlations for email are complex and highly variable over time. The email client used on all desktops in the network is Outlook, which maintains numerous, long-lived connections to a number of Exchange servers situated in an offsite data centre (and therefore not instrumented in the trace). The nature of the computers and services used by an Outlook client at any given time depends upon many things. For example, services such as the RPC endpoint mapper and DNS will be required when a new email session starts up. The global Active Directory catalog stores mailbox configuration and the global address book—the frequency of AD updates will impact communication by clients with domain controllers. Contact with a domain controller also occurs to invoke authentication services and protocols such as Netlogon and Kerberos. In the network, Exchange servers play dedicated roles, either supporting personal mailboxes or else shared email folders (public folders), and so the constellation for many clients includes not just their primary Exchange server, but possibly one or more public folder servers also.

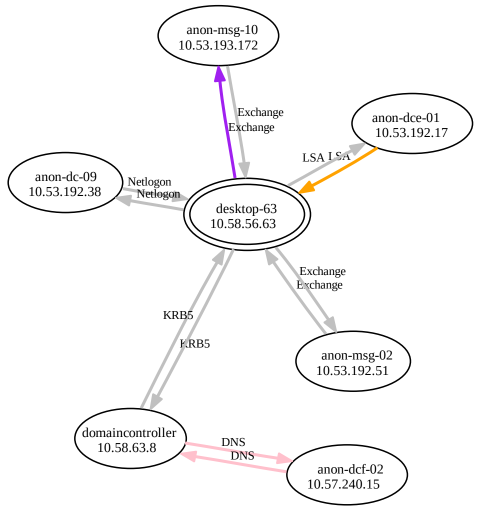
In contrast to the DNS/HTTP scenario we explored above, email TCP connections tend to be long-lived (on the order of hours or even days). Therefore we constructed backwards constellations for email over a period of one day rather than one hour. We found the vast majority of constellations to be very small, with an average of 4 nodes and 6 edges. However, as expected, the structure of these correlation graphs is a lot less consistent across the set of hosts than for HTTP. Nevertheless, it is notable that 90% of the constellations contain the DC, usually via Kerberos and DNS, in addition to their Exchange server, and one third also contain a Public Folder server. Other hosts found in the email constellations include file servers, web servers, non-local domain controllers and various security nodes. One of the common configurations for email dependency is depicted in Figure 13(a), and one of the more complex constellations that we found is shown in Figure 13(b).

6.3 Printing

The transitive correlation from a client to a printer via the print server is an obvious one that we expect Constellation to find. However the situation otherwise is not clear-cut, in part because of the way spooling is handled and in part because of the way printer status changes are communicated to



(a) Typical (simple) email constellation.



(b) Complex email constellation.

Figure 13: 24-hour Email backwards constellations

multiple clients (for example, when a user is watching the print queue status).

We discovered in the trace that print jobs are spooled no less than three times: first on the client computer, second on a central print server, and finally on the printer itself. The length of time between the initial print job being spooled to the server and from the server to the physical printer can often be several seconds. Handling this requires the appropriate delay distribution parameters and is an example of a situation in which a co-occurrence test will struggle to generate useful results without manual tuning of the window, whereas CT-NOR is able to adjust the distribution parameters automatically.

In addition, the print server makes callbacks to any client with an open job or status window on the printer when the printer changes status, and this callback is in the form of an RPC delivered to a named pipe over the remote filesystem protocol (SMB). Thus it appears that the print server acts as a filesystem client to a selection of desktop computers before and after submitting every job to a printer. As a result, the constellation seeded at a printing client may contain a selection of apparently unrelated client computers, as seen

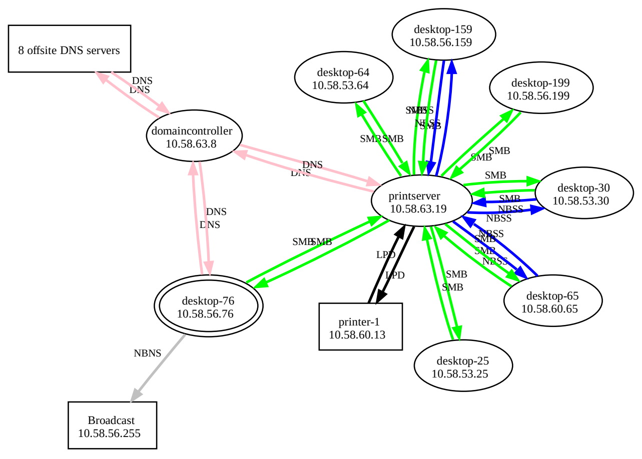


Figure 14: A printing dependency constellation containing 6 clients whose interactions with the print server *printserver* occur during the same time period as those of the root node, *desktop-76*.

in Figure 14.

This scenario highlights a subtlety in interpreting the output of Constellation. In some ways the unrelated clients, which might have jobs ahead in the queue, might reasonably be considered real dependencies since the print job of the constellation of interest may be dependent on the other jobs completing successfully. From another point of view these additional clients should not be entangled in a general constellation; we may not find useful a constellation in which every client that prints is correlated with every other client that prints. One resolution to this problem might be for the system to simply apply a “reasonable” policy such as excluding transitive desktop correlations that pass through a server. At this stage have made no such decisions, as we are still exploring the nature of the network dependencies that Constellation reveals and how the system might be usefully applied.

7. IMPLEMENTATION ISSUES

In this section we discuss a variety of practical considerations which we encountered whilst designing, implementing and evaluating our prototype system.

7.1 System architecture

The modular structure of the Constellation processing pipeline (see Figure 4) provides substantial flexibility in deployment options, allowing the system to cope with legacy environments and various security and privacy regimes.

Fully distributed deployment.

Constellation’s ideal deployment is as a distributed system with each host building its own traffic correlation database and providing an interface for search queries. The dataparallel and localised nature of the computation makes this organisation scale extremely well.

The daemons on individual machines obtain data from tcpdump/winpcap or (where operating system facilities such as strace, ptrace or Windows ETW exist) higher level network events. Additional metadata about services and channels can usually be obtained from tools such as ps and netstat and made available via the query interface.

Partially centralised deployments.

Where it is not possible to deploy daemons on all end-systems, Constellation can operate on packet captures from a spanning port on a nearby switch or router, or an underlying hypervisor. A single centralised machine then learns the traffic models for the whole subnet, and exports the interface to search queries for all those machines. For experimental expediency, most of the results in this paper were obtained using such a centralised prototype and offline packet captures from Microsoft Research Cambridge.

Clearly a combination of the distributed and centralised approaches is possible in which the search system is augmented to find the centralised host providing models for legacy hosts. An interesting option might be to infer models on the end-systems, but to centralise the results.

Security and privacy.

Security requirements, privacy concerns and trust levels within the network also influence deployment choices. A centralised (or logically centralised) system makes it easier to apply consistent access control to potentially sensitive information. Constellation, like netstat and other tools revealing potentially sensitive information, is unlikely to be deployed on the public internet.

Availability.

In a distributed deployment consideration must be given to the infrastructure necessary to communicate with the Constellation daemon on a machine; although we observe that if it is impossible to communicate with a Constellation daemon in the course of a search then it is straightforward to infer where the fault may lie!

It would be straightforward for a Constellation daemon to use standard techniques to improve availability, such as to replicate its model to its neigbours in some P2P system [13], enabling its model to be queried even when the machine is itself unavailable. We have yet to implement this in our prototype.

7.2 Performance

The CT-NOR correlation test has a computational complexity bounded above by $O(c_o b_o c_i (c_i + \log b_i))$ where c_i and c_o are the number of input and output channels respectively, and b_i and b_o are the number of packets on the busiest input and output respectively. Fortunately realistic data avoids this bound in the common case. For a typical hour of data in the Microsoft Research Cambridge trace 92% of hosts required less than one second of CPU with our prototype, and only four hosts required more than one minute. In a fully-distributed deployment, this corresponds to a CPU overhead of less than 1%. For a centralised deployment this means that a single machine normally can learn all the models for a subnet in real-time. These performance figures can also be substantially reduced if packet traces are filtered to remove bursts before fitting the model.

7.3 Burst filtering

Transport protocols like TCP often impose a burst structure on traffic, where each burst consists of several packets with very small inter-arrival times, and the slow-start and congestion window dynamics can produce gaps of an RTT in the middle of the packets which logically comprise a single message. Fragmentation of large datagrams can similarly result in bursts. These bursts can make it more difficult to observe timing correlations.

It is often possible to greatly reduce the data volume (and hence increase performance) by applying a simple filter to remove these bursts. We have explored the use of two filters; one using a low-pass frequency (LPF) design, and another which uses a very simple TCP-specific state machine.

We repeated the ground-truth experiments of Section 5 applying an LPF of 200 ms. The filter reduced the data volume by an order of magnitude, resulting in a much faster processing time. We found insignificant differences in the precision-recall curves for HTTP proxy traffic. Applying the same filter to the whole corpus we found that 98% of hosts complete processing of an hour's traffic using less than one second of CPU time, and the worst case host is reduced to under 8 seconds.

8. RELATED WORK

Constellation is a general-purpose, black-box tool for inferring dependencies, which leads necessarily to a statistical approach and hence some degree of uncertainty in the results. This contrasts with approaches that monitor system and network activity to explicitly track causal paths, for example Pinpoint [6] and Magpie [3]. The latter techniques are highly accurate, but require supporting mechanisms such as request identifier propagation or event logging on all participating systems.

A large number of commercial products exist for monitoring of network, services and applications in the enterprise, typified by HP OpenView. These tend to be customised to specific software and operating environments, in which the expected behavior is well understood. Constellation applies machine learning to deduce the dependencies that occur without assuming a particular environment and set of services or applications of interest.

Project5 [1] and WAP5 [15] make use of packet timestamps for correlation in a similar fashion to Constellation. Those systems aim to expose delays and bottlenecks in local and wide area networks respectively, by recording the application-level messages sent and received by a process (which may comprise multiple network packets). Various linking algorithms are then applied to determine the most likely "causal path", with estimated delays at each hop. In contrast, Constellation makes no attempt to infer a single causal path. Rather, the dependency test of Section 3 identifies all pairs of correlated channels at each host, enabling a complete picture of the inter-dependence of all services in a network.

The Sherlock system [2] can be viewed as complementary to Constellation. It computes global inference graphs centrally using the "service level dependency graphs"[6] learnt by distributed agents. As a result, the system can tolerate more inaccuracy of the local correlation test and the authors of Sherlock have found a co-occurrence-based test works well for their goals of problem detection and localization.

Nevertheless, our experience has demonstrated that co-occurrence can perform poorly in certain situations. In particular, we generated two one-day constellations from the seed host *desktop-36*, whose one-hour constellation is shown in Figure 11. One of these used CT-NOR with an FDR of 5%, while the other used co-occurrence, having manually determined the correlation threshold that gave an almost identical constellation for the one-hour case. In comparison with the one-hour constellation, the full day CT-NOR graph contained 171 additional servers, of which the vast majority—150—were upstream DNS servers. For co-occurrence the full day graph included 105 new nodes plus 108 additional upstream DNS servers. Out of these 105, two thirds turned out to be client (desktop) computers. This result strongly implies that almost all of these correlations are unlikely to be correct. It seems that over a longer time period the likelihood of chance co-occurrence between channels is problematic for the pairwise test.

The need for efficient network diagnosis tools and architectures that enable network management has been stressed before [7, 8, 18]. These papers propose augmenting the network with "a knowledge plane", separate and alongside the existing network, reporting on its current status. In contrast, Constellation provides a basic service for inferring network dependencies over which more sophisticated network management utilities can be built, and it has the advantage of being very easy to deploy over existing infrastructure.

Constellation aims at providing the user or network operator with the capability of comprehending and troubleshooting complex network behaviours. While end-user diagnosis tools have been proposed in the past [12, 14], these solutions identify problems of specific applications or protocols (e.g., TCP reordering, queuing and loss events) and cannot identify network-wide dependencies of host or services.

[6]In Constellation terms, this is equivalent to the constellation in which all channels (at both clients and servers) are aggregated for the service of interest.

9. CONCLUSION

Constellation is a new approach for inferring service dependencies in a computer network, deployable on existing infrastructure and using only lightweight monitoring. It automatically discovers the network-wide map of dependencies from a particular computer, removing the drudgery and guesswork of combing through tcpdump and Ethereal outputs to glean an understanding of how network services are related. In this paper we have presented results against a real network trace that are both accurate with respect to a ground-truth dataset, and that we have shown to be useful for end-users and network administrators alike.

10. REFERENCES

[1] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen. Performance debugging for distributed systems of black boxes. In *SOSP'03*, pages 74–89, Oct. 2003.

[2] V. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. Maltz, and M. Zhang. Towards highly reliable enterprise network services via inference of multi-level dependencies. In *SIGCOMM'07*, Aug. 2007.

[3] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier. Using Magpie for request extraction and workload modelling. In *OSDI'04*, Dec. 2004.

[4] Y. Benhamini and Y. Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society*, 57(2-3):125–133, 1995.

[5] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[6] M. Y. Chen, A. Accardi, E. Kıcıman, J. Lloyd, D. Patterson, A. Fox, and E. Brewer. Path-based failure and evolution management. In *NSDI'04*, pages 309–322, Mar. 2004.

[7] D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski. A knowledge plane for the Internet. In *SIGCOMM'03*, Aug. 2003.

[8] J. M. Hellerstein, V. Paxson, L. Peterson, T. Roscoe, S. Shenker, and D. Wetherall. The Network Oracle. In *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 2005.

[9] IT@Intel. Information Technology 2006 Performance Report. Technical report, Intel Corporation, 2006. `http://www.intel.com/it/pdf/2006-apr.pdf`.

[10] L. Lamport. Quarterly quote. *ACM SIGACT News*, 34, Mar. 2003.

[11] B. A. Mah. An empirical model of HTTP network traffic. In *INFOCOM (2)*, pages 592–600, 1997.

[12] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. User-level Internet path diagnosis. In *SOSP'03*, pages 106–119, Oct. 2003.

[13] D. Narayanan, A. Donnelly, R. Mortier, and A. Rowstron. Delay aware querying with Seaweed. In *VLDB*, Seoul, Korea, Sept. 2006.

[14] V. N. Padmanabhan, S. Ramabhadran, and J. Padhye. NetProfiler: Profiling wide-area networks using peer cooperation. In *IPTPS'05*, Feb. 2005.

[15] P. Reynolds, J. L. Wiener, J. C. Mogul, M. K. Aguilera, and A. Vahdat. WAP5: Black-box performance debugging for wide-area systems. In *WWW'06*, May 2006.

[16] A. Simma, M. Goldszmidt, J. MacCormick, P. Barham, R. Black, R. Isaacs, and R. Mortier. CT-NOR: representing and reasoning about events in continuous time. In *UAI'08*, July 2008.

[17] L. Wasserman. *All of Statistics*. Springer, 2004.

[18] M. Wawrzoniak, L. Peterson, and T. Roscoe. Sophia: An information plane for networked systems. In *HotNets-II*, 2003.