RZ 3714 (# 99724) 06/16/08
Computer Science 16 pages

Research Report

Investigating Indirect Dependencies in Bipartite Cliques of IT Infrastructure Topology

István Szombath

IBM Research GmbH
Zurich Research Laboratory
8803 Rüschlikon
Switzerland

Investigating Indirect Dependencies in Bipartite Cliques of IT Infrastructure Topology

István Szombath

June, 2008

Abstract

IT infrastructure analysis such as, impact analysis, root cause analysis, and compliance checking require deep and detailed information about the dependencies between infrastructure components. Among the passive observation methods, flow-based network traffic information is a promising source to identify and label dependencies, although the so called indirect dependencies are not directly observable from flow records. To identify indirect dependencies network traffic analysis is needed. In this article a method is introduced that can discover context sensitive indirect dependencies. The fundamental idea is to determine bipartite cliques and common interest groups of server pairs. From this information a regular expression is made, that can find indirect dependencies in flow records. After that the information can be propagated to CMDB to serve enhanced IT infrastructure analysis.

1 Introduction

Building and maintaining a business-driven IT infrastructure is a big challenge. System management provides a systematic approach for building and maintaining IT infrastructures and helps to keep costs at a reasonable level. The IT Infrastructure Library (ITIL) is the de facto standard in system management. The core element of ITIL is the Configuration Management Database (CMDB). An accurate, rich and up-to-date CMDB is a pre-requisite to efficient system management, because the CMDB is indispensable for analysing IT infrastructures, such as impact analysis, root cause analysis, and compliance checking.

Many IT infrastructure analysis approaches exist. Most of them require deep and detailed information about the dependencies between infrastructure components. Therefore without the knowledge of the IT infrastructure map, especially the service dependencies, accurate impact analysis and root cause analysis cannot be carried out.

Some methods exist to keep the CMDB accurate and up to date. The common denominator of these methods are that they all discover the surrounding environment and track changes, for instance, with continuous rediscovery. Discovery methods include scans, credential discoveries, configuration analysis, and passive observation. While scans and credential discoveries might cause unwanted load on the system and unacceptable security threads, passive observation minimizes overhead to the system and security problems. Passive discovery has many advantages, although the information provided might not be detailed enough, therefore for deep discoveries other methods might be needed as well. To assume passive observation is the first step toward deep discovery.

Among the passive observation methods, the flow-based network traffic information is a promising source to identify and label dependencies, although the so called indirect dependencies are not directly

observable from flow records. To identify indirect dependencies network traffic analysis is needed. For the analysis vast amounts of network traffic information has to be collected and previous research shows that NetFlow is suited for this.

It is assumed, that from flow record information can be mined that represents indirect dependencies. First a special type of context sensitive indirect dependency is introduced, than a method is introduced that helps discover context sensitive indirect dependencies. An algorithm is introduced that can determine the degrees of hosts, and most importantly, the common interest groups of server pairs. From this information a regular expression is made, that can find indirect dependencies. In this way, the discovery approach is a pattern creation/learning and matching approach. The information is mined from flow records (e.g. NetFlow records) and can be propagated to CMDB to serve enhanced IT infrastructure analysis.

First, in Section 2 existing concepts, methods and techniques are introduced. This includes the definition of relationship and dependency, introducing advanced dependency detection techniques, a short summary about network traffic information, and the discussion on configuration management, especialy the central repository of the infrastructure, the CMDB. In addition, IT infrastructure analysis CMDB requirements are also discussed.

In Section 3, concepts are discussed that are vital to build a prototype system. A proposal is discussed how to improve IT infrastructure analysis. The fundamental idea is to store relevant information in the Configuration Management Database (CMDB). For instance number of clients of servers proved to be a relevant information. Most importantly, the section reveals what are the possible indirect dependencies that can be collected from flow traffic information. In this way a novel approach is described how to find semantical dependencies, that cannot be found with existing dependency discovery techniques. The implementation of the prototype modules is also described.

Finally, Section 4 summarises the achievements of the article. Proposals are also described how to improve the prototype in later works.

2 Discovery Techniques and Configuration Management

ITIL defines a relationship as a connection or interaction between two people or things [9, p. 242]. In configuration management it is a link between two configuration items that identifies a (direct) dependency or a connection between them. A dependency is defined by ITIL as the direct or indirect reliance of one process or activity on another [9, p. 230]. In this way relationships (thus direct dependencies too) can directly extracted from network traffic flow data, such as NetFlow. Indirect dependencies however cannot be represented with one relationship only thus sophisticated analysis is needed to identify them.

In this subsection topology discovery techniques as well as related terms and methods are going to be discussed. The basic idea is to create and maintain the IT infrastructure map automatically. Topology discovery reveals relationships, thus it is a pre-requisite to discover indirect dependencies.

A network topology and infrastructure map might be able to have different meanings. Network topology and infrastructure map is a graph, where the nodes (vertices) of the graph are hosts in the infrastructure (e.g. an IP address or an application), and an edge means that two nodes are communicating with each other. The edges are directed, the source node of the edge is the source host, and the destination node is the destination host of the communication. In this way the graph is a logical view of the infrastructure.

Active Network Discovery It is based on the repeated querying of network components from one or more points in the network [4]. This includes credential-based and credential-less techniques.

Credential-less techniques include queries that do not require authentication, such as pings and port scans. Security risks are not present when using credential-less methods, however scans may cause false security alarms. The techniques collect limited information on the observed systems.

Credentialed techniques includes queries that require authentication (i.e. ssh query invoking netstat on a remote machine). The main disadvantage of this method is that the credentials have to be gathered, stored and updated. In a big enterprise environment this is infeasible and poses security risks.

Passive Network Discovery It extracts information by observing network traffic. The method is not intrusive, can run continuously, security related risks and resource usage are minimal. Three types of passive network traffic monitoring can distinguished [4]:

Payload Inspection that enables Application Layer inspection. This method gives the most detailed data, however, it takes the most computing power and for every protocol it requires a parser.

Header Inspection that enables Transport Layer inspection. This method has less overhead, but also gives less details. Due to significant increase of encrypted traffic the transportat layer is the highest layer that can provide useful information, thus header inspection has a potential.

Summary Records contains reduced size of data compare to header inspection, as certain records are aggregated that share the same common attributes. This is a very scalable method when planning network observation in enterprise networks.

Network monitoring capabilities are integrated into network devices such as router and switches, thus deploying such a system requires minimal effort.

Flow-Based Network Traffic Monitoring Monitoring computer network traffic is a common discipline in network management. One such approach is NetFlow protocol that is developed and standardized by Cisco Systems. NetFlow is a network feature to collect IP traffic information. For instance, if a Cisco router has NetFlow feature enabled it will send NetFlow records to a given destination. NetFlow initially implemented by Cisco, but now it emerged as an IETF (Internet Engineering Task Force) standard named Internet Protocol Flow Information eXport (IPFIX) [8].

A NetFlow record summarizes network traffic. The summary records contain information about end-to-end IP Flows. A single flow contains aggregated information from a set of packets that are exchanged between two endpoints. The usual fields of a NetFlow record are source IP address, destination IP address, source port number, destination port number, layer 3 protocol type, packet count, byte Count, start time of flow, and end time of flow.

2.1 Advanced Dependency Detection Methods

To reveal dependencies network topology has to be analysed. A challenging problem is to find indirect dependencies, that cannot be represented with a direct edge between components in the network infrastructure map. For instance an authentication server and a web server that are communicating through clients have an indirect dependency.

A wide variety of indirect dependency detection techniques exist. A promising approach is to identify correlated relationships. This means that an application server is (almost) always requested

after a web server request for instance. To find correlated relationships flow timings [1] [6] or data mining techniques [4] can also be used. A bit different approach but also dealing with time windows is Sherlock [2]. Their approaches and usability differs from each other, although they are common in that observe patterns, like two flows happening very often within a short interval, or in given time window.

Searching for patterns and classifying network traffic [7, 5] based on pattern matching not differs much from the previous approaches. BLINC is a promising research in this field. BLINC creates pattern semi-automatically and tries to match these patterns to the infrastructure map. If a matching is found a set of flows has been classified.

These approaches are promising but every classification can contain false positives and negatives:

False positives In a real IT infrastructure the background traffic is always noisy, that may result in false detection of correlations. Another reason for false positives could be that NetFlow is aggregating certain flows, thus hiding the exact flow start and end times. Correlation detection is sensitive to the size of the event/time-window, because bigger time windows can result more false positives.

False negatives In the presence of noise, statistical correlation requires a large number of observations, which is not always practical. For this reason, important but rarely observed patterns can be hidden. For instance a web server depends on an authentication server; for accessing the web server authentication is needed for every session. They can communicate not just directly but also through clients. Conditional dependencies also may stay hidden.

2.2 Configuration Management Database

CMDB is a centralized information repository to store relevant data about the IT infrastructure. CMDB is a database that contains all relevant details of each Configuration Item and details of the important relations between CIs (cite). A Configuration Item is defined as an asset, service component or other item that is, or will be, under the control of the Service Asset and Configuration Management. Configuration Items may be grouped and managed together. ...(cite) CMDB is indispensable for effective IT management. CMDB is the core of ITIL, thus it is a pre-requisite for all management process.

The real power of CMDB does not lie in its pure data, CIs and explicit relationships, but the power to create relationships between CIs not represented explicitly in the model. Implicit relationships can be represented by more explicit relationships.

2.3 CMDB IT infrastructure analysis Requirements

The most important analysis that CMDB can support is impact analysis (cite). For impact analysis two things are crucial: (1) the error propagation has to be evaluated to determine which events may result in a to service failure, and (2) the users affected on failure has to be estimated in order to create event priorization. Both objectives can be supported by CMDB.

According to ITIL [9, pg. 51] priority depends on the urgency and the impact of the event. In most cases impact is the users affected, thus the more users is affected by a particular (service) failure, the higher priority the event has. However in some cases even one user affected is unacceptable, depending who is that user (e.g. VIPs). Urgency reflects the maximal resolution time without SLA breach, then urgency can be dynamic.

Thus it can be assumed, that the most important metrics of a service is its degree (the number of users using that particular service). *Degree* is the number of users using that service.

User groups can be used to help incident resolution by filtering the same incidents from the queue (e.g. failure of a server is reported by many of its users). It also can be used to inform users that are affected by a service failure.

In this way in order to have an ITIL compatible IT management system, the CMDB has to support impact analysis that must be able to determine users affected in order to determine event priority and help management processes.

3 Enhanced Information Collection and Population Methods

In this section methods and ideas are proposed to implement a system that is capable of mining enhanced information based on passive observation (i.e., using aggregated network traffic information). Enhanced information can support improved IT infrastructure analysis and in this way help to get a better feedback, thus help to improve the efficiency of IT infrastructures.

3.1 Topology Analysis

An important pattern in the topology is to find the fully connected subgraphs or also called *cliques*. The reason for observing cliques can be a distributed system, malicious activity, sensor system, and p2p network (a dense subgraph, an 'almost clique' is shown on the left in Figure 1). Another variation of finding cliques is finding *bipartite cliques*. A bipartite clique is shown on Figure 1 (right), it has two disjunct set of nodes and the all nodes are connected to all other node from the opposite subset. The reason for observing (partial) bipartite clique might be:

Load Balancer More hosts are working together to provide a single service interface that can process requests from clients. If the observation interval is long enough, and the clients often use that service, a full bipartite clique might be observed.

Indirect Dependency It is common, that clients access an naming or authentication service just before accessing a particular service. For instance, a DNS request is observed before the access to a web server, or an authentication is needed before accessing a service. The services not necesarely communicate with each other directly, but they are depending on each other, because failure of service may cause the other service unreachable.

Common Interest Group It is possible, that a group of clients uses the same subset of services (i.e. their group of interests are the same). There is no dependency between the two services. This information can be useful for management processes.

Defect Observation of a clique; a clique exists in the infrastructure, but not all edges can be observed, not all the routers are NetFlow enabled.

Finding maximal cliques in a graph is NP-complete, thus finding these kinds of patterns is far from trivial. We argue that heuristics can be used to find 'important' bipartite cliques. The reason is that even an exponential algorithm can be effective when the input shows typical patterns.

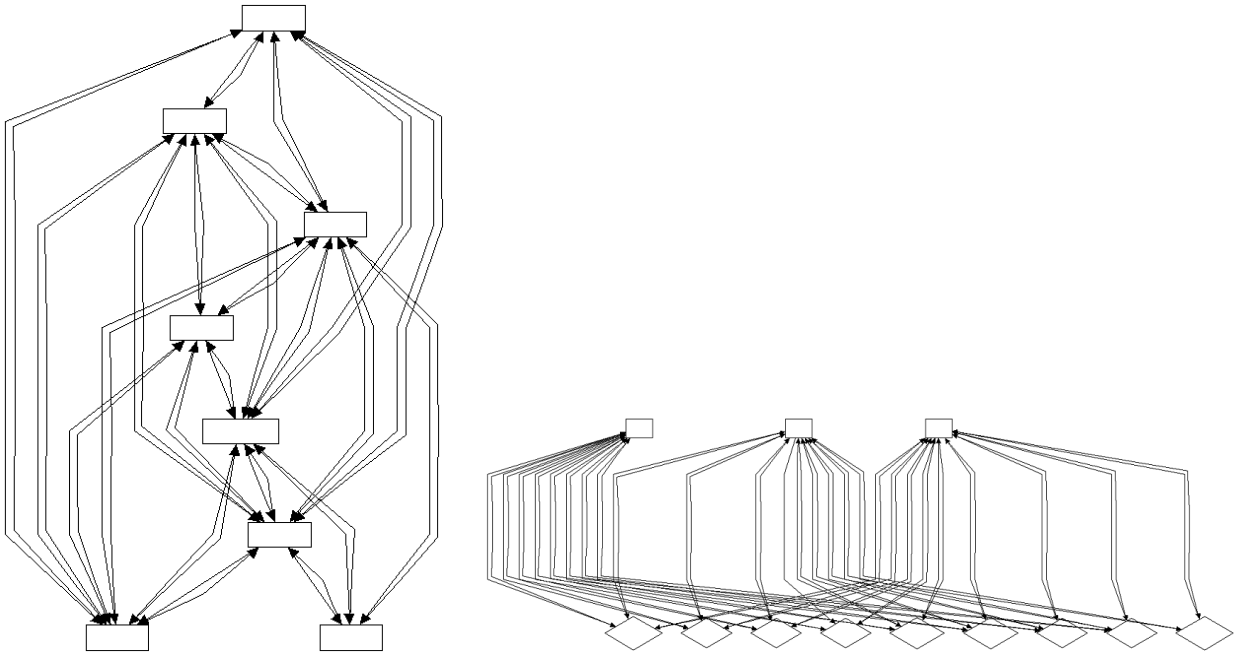

Figure 1: A dense subgraph in the network traffic topology (sensor network, left). Common interest group of clients at ZRL (Servers: boxes, Clients: rhombuses, right).

3.2 Peculiar Proprieties of the Infrastructure

In Figure 2 (left), it is shown that only few nodes have many neighbors (have a high degree[1], i.e. > 10, within 300s). Figure 2 (right) shows, that the most frequently used servers are accessed using well-known ports. Research shows that it is a good server identification heuristic to find hosts that accessed by the same service port frequently. Thus to classify servers, frequently used (or also called high degree) $ip : port$ tuples have to be identified. The majority of the hosts are not 'talking too much', thus most likely only minority of the nodes will have much higher degree than the average degree. That is why observing the neighborhood for instance, the 10% most popular servers can provide useful information.

If $ip : port$ tuples are observed more than 96% of the tuples have degree of one. Thus if an $ip : port$ has higher degree than $max(degrees)/20$ (12 in this case) it can be considered an important server (and the service access point is the $ip : port$). Algorithm 1 shows how to determine $ip : port \rightarrow degree$. The input is NetFlow records in CSV format. Every line of the CSV file is one flow record. The record has fields, such as destination and source IP address and port. In addition the algorithm creates an adjacency list of the graph that is a data structure, an orthogonal linked list that represents a graph. The list is very similar to the incident matrix. In the incident matrix the two dimensions of the matrix are the nodes (vertices) and the edges. The value of a matrix element is not zero (usually 1 or -1 depending on the direction of the edge) if the node in that row is associated with the edge in that column. The incident matrix can be implemented as a sparse matrix, because in a usual enterprise environment the incident matrix contains many zeros. A possible implementation of a sparse incident matrix is the adjacency list. In addition the degree of services can be populated into the CMDB to

[1]In a non-directed graph where all the edges haev wight of one degree is the neighbor of the node. If the graph is directed, the degree of a service is the number of hosts (clients) using that service.

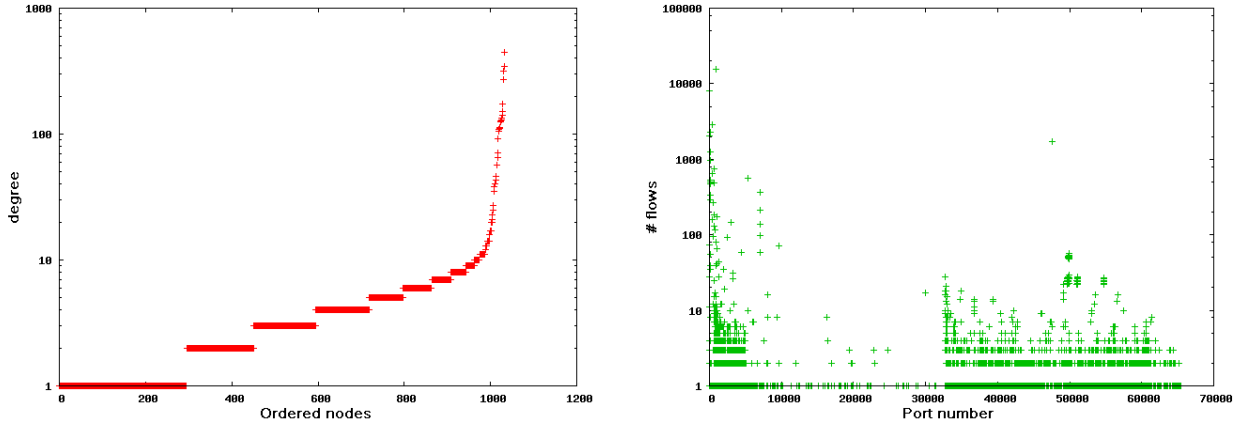

Figure 2: Degree of nodes (left), Port usage distribution (right)

support better impact estimation.

Algorithm 1 Determine degree of $ip : port$ tuples

Input: NetFlow records in CSV format
Output: Array d where $d_{ip:port}$ is the degree of $ip : port$

1: **for all** $line \in CSV$ **do**
2: $srcIP := line_{srcIP}$ {store source IP from CSV record}
3: $dstIP := line_{dstIP}$ {store destination IP}
4: $dstPort := line_{dstPort}$ {store destination port}
5: $SAP := dstIP : dstPort$ {SAP=Service Access Point, $ip : port$ tuple}
6: **if** $N_{SAP} = \emptyset$ **then**
7: $d_{SAP} := 1$
8: $N_{SAP} \leftarrow srcIP$ {N is the adjacency list, N_i is the i-th 'row'}
9: **else**
10: **if** $srcIP \notin N_{SAP}$ **then**
11: $d_{SAP} := d_{SAP} + 1$ {increment degree}
12: $N_{SAP} \leftarrow srcIP$ {store neighbors of a service}
13: **end if**
14: **end if**
15: **end for**
16: **return** d

In this way an effective method was introduced to classify servers without any preliminary knowledge. Although most of the clients 'don't talk too much' and this makes them impossible to classify. Another problem is, that there are some cases where a host is acting as a server and as a client as well. This can refer to service dependency, or it is a just a 'client on the host' (remote connection).

3.3 Searching for Bipartite Cliques

Finding bipartite cliques is important as they may reflect load balancers, common interest groups, and indirect dependencies.

IP Address (encoded)	Service Access Port	Service Name	Degree
14826	80	www-http	56
11095	88	Kerberos	57
8778	7000	afs3-fileserver	61
15015	445	Microsoft-DS	62
14791	80	www-http	63
14847	445	Microsoft-DS	64
17773	864	unknown	68
8771	7000	afs3-fileserver	71
11088	389	LDAP	72
8806	514	RSH	78
15015	389	LDAP	81
129017	138	NetBIOS	102
11165	5308	Cfengine	105
14861	137	NetBIOS	109
43001	138	NetBIOS	122
8834	53	DNS	167
15015	53	DNS	225
17773	138	NetBIOS	243
8834	138	NetBIOS	243
14896	138	NetBIOS	244

Table 1: Important services ($ip : port \rightarrow degree$)

To reduce the complexity of the graph, the *neighborhood* of a particular 'popular' server is observed (denote it with s where $s \in Servers$). It is assumed that only a minority of the servers (IP or IP:port tuples) are going to have high degree within a 300s interval (NetFlow default value). Thus the bipartite cliques most likely will be around popular servers (services). The server is typically accessed by clients and might be accessed by and accessing other services/servers. Let $N(s)$ denote neighbors of a server. The 'direct neighborhood' of the node s is $s \cup N(s)$. In the same way the size of the subgraph can be further extended: let $s \cup N(s) \cup \bigcup_{i \in N(s)} N(i)$ denote the *neighborhood* of s. In this way *neighborhood* is a 2-depth traversal starting from the node s. Our research shows that this graph is still large and, thus, a further heuristic is needed to reduce its size.

The goal is to find bipartite cliques, i.e. connections between servers and clients, thus server-to-client-to-server connection can provide useful information, but server-to-server-to-client connection chains are irrelevant in this case. For this reason the concept of the neighborhood is redefined:

$$s \cup N(s) \cup \bigcup_{i \in N(s) \cap Clients} N(i) \tag{1}$$

But still, the *spanning subgraph* determined by the nodes of the 'neighborhood' contains irrelevant data. Denote the spanning subgraph with H. Fist delete the edges from the spanning subgraph H, where the source or the destination node is more than two step away from the main server ($E_H \setminus \{(u,v) | u, v \notin N(s)\}$). After that nodes with degree of one can be deleted. This step is repeated. At the end, the subgraph will still contain the bipartite clique (it will contain server-to-client-to-server connections). For a server in the subgraph the number neighbor clients that have distance one from the main server is going to be the size of the common interest group of the clients with the main server.

In this way, the common interest group has many clients and only two servers. The graph is reduced significantly around a server. The subgraph is used for identify possible indirect dependencies.

Algorithm 2 Find Bipartite Cliques (where $|S| = 2$)

Input: NetFlow records in CSV format, d vector of $ip : port$ degrees, I adjacency list of the graph (logical topology of the IT infrastructure).

Output: $biCl$ matrix, where $biCl_{s,s'}$ is a set of clients that are using both s and s'.

```
1: m := max_∀i{d_i} {find maximum value in d vector}
2: threshold := m/20 {set threshold}
3: S := ∅
4: for all i : d_i do
5:    if d_i > threshold then
6:       S ← i_IP {if the degree is high then i_IP is a server}
7:    end if
8: end for
9: for all service ∈ S do
10:   s := service_ip
11:   Dist := ∅ {reset Dist data structure}
12:   Dist_s := 0 {s distance from s is zero}
13:   for all srcIP ∈ I_s do
14:      Dist_srcIP := 1 {neighbors of s have distance 1 from s}
15:   end for
16:   for all line ∈ CSV do
17:      srcIP := line_srcIP
18:      dstIP := line_dstIP
19:      if Dist_srcIP = 1 OR Dist_dstIP = 1 then {src or dst is neighbor of s}
20:         if dstIP ! = s AND srcIP ∉ S AND srcIP ∉ biCl_s,dstIP then {dst not s, src not
           server and src not in bipartite clique yet}
21:            biCl_s,dstIP ← srcIP {store IP participating in the bipartite clique}
22:         end if
23:      end if
24:   end for
25: end for
26: return biCl
```

The common sense dictates that bipartite cliques will be only observed near important services. The reason for that is that the server's degree has to be large if it participates in a 'big' bipartite clique (it has to have lots of 'client' neighbors). It is also trivial that server degree is larger or equal than service degree. The idea is to mark host that are providing a high degree service as servers. First the focus will be finding bipartite cliques, where the number of servers in the clique is two ($|S| = 2$, Figure 3 (left) shows such a bipartite clique). The reason for that is that the common interest group of clients (hosts) have to be determined in order to find load balanced services, indirect dependencies, etc. In this way the problem was reduced to find maximal vertex-independent paths with length two between the servers, where the vertexes between the servers has to be clients. Algorithm 2 describes how to determine $biCl$ matrix, where $biCl_{s,s'}$ is the set of clients that are using both s and s' (in a reverse order, thus s' first than s). The focus will be on a simple pattern. If $s' \longrightarrow s$ than it is enough

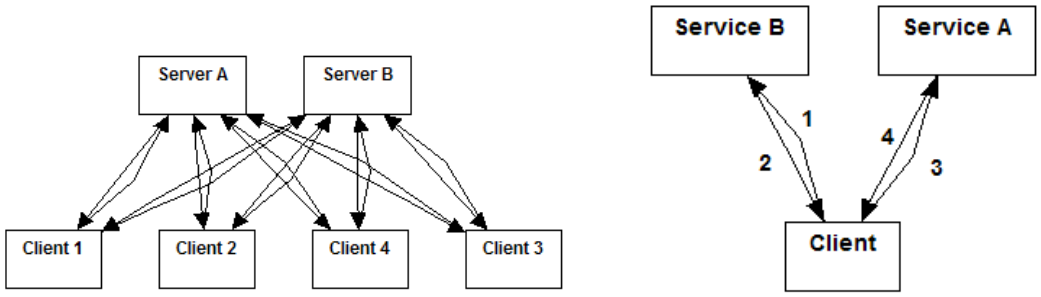

Figure 3: Topological view of a conditional indirect dependency: bipartite clique (where $|S| = 2$, left). Sequence of an indirect dependency (right).

to search for patterns such as:

$$\begin{aligned} s' &\rightarrow c \\ c &\rightarrow s \end{aligned}$$

The Algorithm 2 needs NetFlow records in CSV format, the d vector that is an output of Algorithm 1, and the I adjacency list. The adjacency list can be created similar such as the N adjacency list is created in Algorithm 1. The only difference is that on the y dimension N has $ip : port$ tuples while I has ip addresses on the y dimension.

In this way, the output basically holds the information of the two length vertex-independent paths between two vertexes (where a vertex in the middle of the path cannot be a 'server').

3.4 Conditional Indirect Dependencies

Critical control plane and session setup traffic is less frequent than normal traffic [1]. That is why many existing advanced dependency detection methods cannot classify indirect dependencies and they produce false negatives. A novel method is introduced to classify (a special kind of) *conditional indirect dependencies*. The bipartite cliques in the logical network topology are identified. The next step is to investigate the bipartite clique. The fundamental idea is to create a pattern based on the identified bipartite cliques and to find such patterns in the network traffic.

Assumed that *Service A* and *Service B* have an indirect dependency: *Service B* is a DNS server and *Service A* is a web server. Client connections to *Service A* are proceeded by connections to *Service B* (within a certain time window). To formalize this; if $ServiceB \rightarrow ServiceA$[2], within a given interval (300s) the following sequence can be observed for a lot of $c \in clients$:

$$\begin{aligned} c &\rightarrow ServiceB \\ &\vdots \\ ServiceB &\rightarrow c \\ &\vdots \\ c &\rightarrow ServiceA \\ &\vdots \\ ServiceA &\rightarrow c \end{aligned}$$

[2]The reason for the notation is that *Service B* is indirectly communicating with *Service A* (through many clients).

Figure 3 (left) shows the topology of such *indirect dependency*. On the right it also describes the sequential behavior of such an indirect dependency.

It is also possible to find correlation based on starting time of the flows. This method is better to find correlations, although it is yet unsolved how to create a single regular expression that will find only correlated conditional indirect dependencies. An idea to solve this problem is to find all possible patterns and investigate correlation on these patterns. This method is easy to implement although on-line processing is not feasible.

To match patterns like described a complex regular expression can be used. The regular expression below matches if there is an indirect dependency between $s2$ and $s1$ where c_i are the clients in the bipartite clique (using *perl* syntax).

```
/(?s)(c1|c2|..|cn),s2.*?s2,\1.*?\1,s1.*?s1.\1/
```

A more simple patterns looks like the following: it is assumed that s' and s has indirect dependency ($s' \longrightarrow s$). Then the regular expression is $s' \rightarrow c_i\ ,\ *\ ,\ c_i \rightarrow s$ where c_i is a 'variable' and $*$ is a wildcard. After the regular expression is created, it has to be evaluated on NetFlow records. If the regexp search matches for different 'clients', then the indirect dependency between the services are discovered, and the dependency can labeled as *Indirect Dependency*.

With the following code it is possible to find indirect dependencies in a CVS file where the third column is the starting time of the flow (first and second column is the source and destination IP address). The script has three parameters. If $s' \rightarrow s$ assumed, $\$a := s$ is the first parameter, $\$b := s'$ is the second parameter, and the clients participating in the bipartite clique are the third parameter ($\$c := c_1|c_2|\ldots|c_n$).

```
s/(?s)'$b',('$c'),([0-9\.]+)\n
(.*?\1,'$a',([0-9\.]+)\n)/$3/)
```

The pattern match will find relevant patterns but the correlation has to be investigated. For this, the starting times have to be stored (for every matched pattern) and if the difference between the starting times are less than one second they can be considered as correlated events.

However this *perl* script is a bit more sophisticated than described. It is designed to find all possible patterns, but for that the *g* parameter is not enough (it is problematic to find embedded or overlapping pattern). For the sake of simplicity to solve this problem pattern replacement is used instead of simple matching. In this way a recursive matching is implemented. To discuss this problem in full detail is out of the scope of this article, although it is important for future works to create a more efficient pattern matching.

In this way to discover indirect dependencies first the bipartite cliques has to be created (at least where the number of servers are two in the clique). In $biCl$ matrix $biCl_{s,s'}$ holds the information for the clients that are communicating with both s and s'. Where $|biCl_{s,s'}|$ is high a regular expression has to be created to evaluate the possible indirect dependency between s and s'. Thus all relevant server pairs have to be evaluated. But since there are not many servers with high degree, the number of regular expressions that has to be evaluated on the NetFlow records are not high (few hundred).

To conclude: if $|biCl_{s,s'}|$ is high, and if the regular expression search matches for different clients (where $c \in biCl_{s,s'}$ and the number of matches for different clients $\approx |biCl_{s,s'}|$),the dependency $s' \rightarrow s$ can be labeled as indirect.

In this way, the dependency detection has two phases. First, the pattern has to be created, after that it has to be evaluated. The current method does not consume a lot of resources, although the scalability has a downside, because the proposed method has exponential complexity. However in the

near future hardware acceleration can be used to speed up the IT infrastructure analysis and make it more efficient.

3.5 Populate the CMDB

In this subsection the implementation of enhanced information population into CMDB is going to be described. For this a third party application has to be developed that is able to populate information through a defined Application Programming Interface (API). If the CMDB meta-model is not capable of holding all the relevant data, then the data model has to be extend.

Important enhancements of the meta-model are to reflect *degree* for servers, services, and label dependencies as *Indirect Dependency* if needed.

3.6 Implementation

All the concepts, methods and implementation techniques are introduced to create a system that is capable of collecting information from flow records that can enhance analyses and import this information into the CMDB. The aim is to implement a pilot system that proofs the concepts: flow information holds relevant information, this information can be gathered, and it can support better IT infrastructure analysis.

First contribution is to create a graph that represents the logical topology of the IT infrastructure from flow records (e.g. NetFlow summary records provided by NetFlow enabled routers). After the graph is built, useful information can be extracted from the topology map and fed into the CMDB. The information extracted from the topology that is useful to an enhanced IT infrastructure analysis includes:

- determine degree of hosts and classify popular servers, and
- to extract and understand patterns from the infrastructure topology, most importantly search for bipartite cliques as an evidence of (conditional) indirect dependencies[3].

Figure 4 describes the operation of the implemented system that proves the concepts. It is assumed that the data model is already extended (0). The first step (1) is to determine the degree of services ($ip : port$ tuples). That information is propagated into the CMDB (2).

After that, servers are classified (3a). A server is a host (an ip address) where the $ip : port$ has high degree ($D_{ip:port} > max_{\forall i,p}(D_{i:p})/20$). Meanwhile a data structure has to be created, an adjacency list that represents the logical topology of the infrastructure (3b).

The next step is to determine the 'bipartite cliques' (4a). As mentioned this kind of cliques just contain two servers and as many clients as possible, although this information is enough to collect valuable information. It was also mentioned that basically a matrix is created ($biCl$), where the dimensions are the servers, and an item refers to a vertex independent two long paths between the servers (but paths that go through servers are not counted). Elements, where the size of the common interest groups is small (≤ 10), can be skipped (4b), because relevant information cannot be mined from them. This will reduce the size of $biCl$ significantly, thus only relevant server pairs will be investigated. In addition for all matrix elements $port$ information and the clients participating in the common interest groups are stored.

[3]Indirect dependencies that cannot be revealed using conventional detection techniques because they are sensitive to false negatives due to caching, unrevealed conditions, and under-sampling for instance.

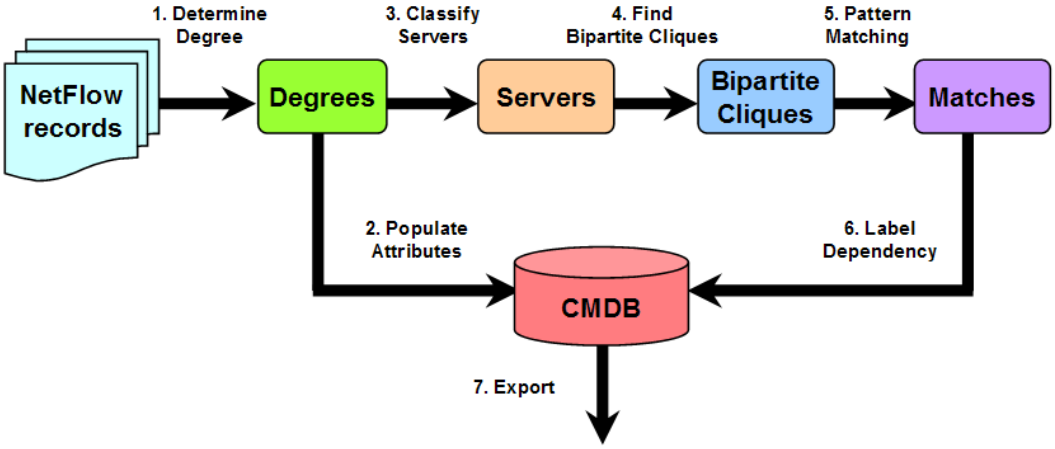

Figure 4: Architecture of the pilot system

All the server pairs have to be investigated where $biCl_{s,s'}$ is high. The semantical label for dependencies that were introduced in this work is *indirectdependency*. A regular expression has to be created (5a) that investigates the possible indirect dependency between the services. This regular expression has to be evaluated on flow records (5b). Upon match an indirect dependency has been found.

Finally the labeled dependencies have to be created or updated (6). To support ITIL processes the enhanced information can be exported from the CMDB (7). In this way CMDB stores information collected from flow records that can support better IT infrastructure analysis.

3.7 Results

The pilot IT Infrastructure used for the research, development and test implementation was the IT infrastructure of IBM Zurich Research Laboratory (IBM ZRL). The pilot system has to reflect typical features of a 'normal' enterprise class IT infrastructure, therefore representativeness is important. A subgraph from a logical topology that is extracted from 30s of ZRL's network traffic is shown on Figure 5 (left). The infrastructure is representative, because it is heterogeneous and complex, it uses diverse in terms of applications, and the infrastructure is complex even when looking at the macro and micro physical and logical structure. This makes the environment useful and feasible for a pilot system.

In ZRL's IT infrastructure, four routers have NetFlow enabled (all the traffic between the subnets and the ingoing/outgoing traffic is covered). During peak time (approximately at 10:30) the routers generate 366 flows per second, that means 34Kpkts/s and 254Mb/s.

The prototype system is fed with flow records from infrastructure at (**ZRL!**). In order to collect relevant information 100,000 flow records are parsed that are from a peak hour period (weekdays 10:00am). From the flow information:

- 1034 individual IP addresses are observed,
- 31,532 $ip : port$ tuples are observed,
- 89 important services ($ip : port$) are identified and the degrees of services are populated into

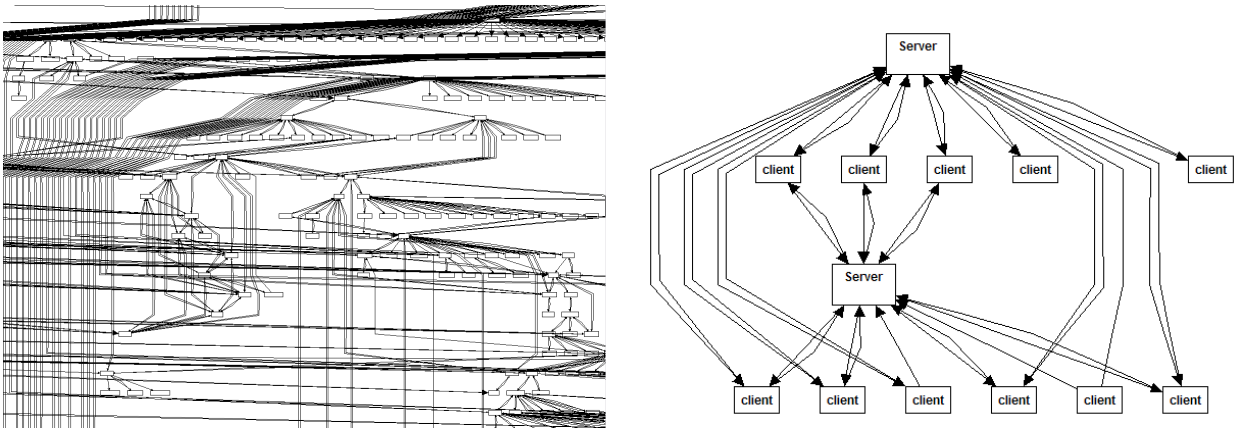

Figure 5: Subgraph of the logical topology extructed from 30s of ZRL network traffic (left). A load balanced service (Lotus Notes, right)

Tivoli Application Dependency Discovery Manager (TADDM)[4],

- 202 server (ip) pairs are found where the size of common interest group is not low (> 10).
- 14 indirect dependencies are found based on regular expression machining (and further processing), and
- the semantical dependencies are populated into TADDM.

The prototype is able to populate valuable information into the CMDB including the number of clients of services, and semantically enhanced dependencies. These populated information can be exported to support further analysis. Although validation of the populated semantical dependencies is problematic. The system is tested with authentic NetFlow data from a real life scenario. The problem with validation is that a well-documented infrastructure does not exists. That might be crucial to evaluate dependency discovery, i.e. to determine the number of false positives and negatives. For instance common sense dictates that the service port and the size of the common interest group is not sufficient in some cases to determine a load balanced services (although it was sufficient in the test dataset). All of the possible labeled dependencies cannot be validated using manual techniques. Other method is to compare this dependency detection to existing methods, but it is problematic, because this system is not designed to detect all kind of dependencies but to find a special type of dependency, that are unable to find with existing techniques. To summarize, all load balanced services and indirect dependencies that are found seems to be valid. For this some manual technique and semi-automatic investigations are used. The discovered conditional indirect dependencies cannot be found with existing methods, so the number of false negatives are not known.

In addition some limited flow classification is possible. For instance a high port can be classified when a high port communication is followed by an RPC or FTP request. This outcome is not among the key results however it might be important when considering improvements or reusability in related fields.

[4]CMDB solution by IBM

4 Conclusion

The article proposed a new technique to process information from flow records and populate this information to the CMDB. The collected and populated information includes the degree (the number of users of that service) of services. This information can support better impact estimation and can improve group management. Another achievement is the identification of semantic dependencies; a special case of indirect dependencies are identified where the servers are not communicating with each other directly but are linked through many clients. This case includes when the correlation is conditioned, due to caching for instance. The fundamental idea is that a special data structure has to be created that stores the two long vertex-independent paths across clients between two arbitrary servers. A novel approach is to create a pattern i.e., a regular expression, that is specific enough to keep complexity in hand, but abstract enough to identify patterns that cannot be identified with standard flow correlation techniques. After creating the pattern, pattern matching can take place, and upon match, the information can be populated into TADDM to support better IT infrastructure analysis.

The pilot system is implemented, and the populated information is 'ad-hoc' validated. The described novel dependency detection has two major phases: pattern learning and pattern matching. Pattern learning is needed to reduce the complexity of the pattern, to make it more specific. The system proves the concepts. However, to work out a mathematical proof a comprehensive validation is inevitable in the near future.

It is possible to keep this information up-to-date, although there are challenging parts. For example, it is possible to update the degree of services, although this function is not incremental yet, thus taking measurement during the same conditions is crucial (e.g. 100,000 flow records at 10:00 am on workdays). Another example is that each time semantical dependencies are labeled and populated, the labels however, do not decay. This means that if an old label does not stand anymore it will not be deleted. To summarize, the prototype is not incremental yet, thus to keep the information up-to-date full re-analysis is needed. Incremental model building is one possible way to improve the system both in speed and accuracy.

The degree of a service is most likely a metric that is changing over time. Methods on how to measure and model changing degree in time have not been found yet, however concepts can be re-used from intrusion detection [3] for instance, in a later work.

The aim of this article is to describe a novel dependency detection method that is able to find one particular dependency that is not possible to find with most existing detection techniques. Although a possible improvement is to extend the number of patterns and thus improve coverage. For this existing works have to be re-examined (such as [5]), and typical patterns have to be gathered. It is also possible with the introduced dependency detection method to classify high or random port number traffic such as RPC calls or passive FTP.

Network traffic analysis is a promising source to collect dependency information and to label dependencies in order to create a consistent notation system of semantical dependencies. This field will be more important in the future as it becomes increasingly important to understand the dependencies between components, although it requires more and more effort to keep the IT infrastructure up-to-date due to complexity.

Although the prototype requires some improvements to enhance efficiency and improve usability, it is capable of collecting relevant information from flow records. Furthermore it is now understood what the information needs of a CMDB are, in order to support enhanced IT infrastructure analysis. Finally a method was introduced on how to find indirect dependencies that cannot be identified using existing dependency detection methods.

References

[1] Paramvir Bahl, Paul Barham, Richard Black, Ranveer Chandra, Moises Goldszmidt, Rebecca Isaacs, Srikanth Kandula, Lun Li, John MacCormick, David A. Maltz, Richard Mortier, Mike Wawrzoniak, Ming Zhang, *Discovering Dependencies for Network Management*, Proceedings of the Fifth Workshop on Hot Topics in Networks (HotNets-V), November, 2006

[2] Paramvir Bahl, Ranveer Chandra, Albert Greenberg, Srikanth Kandula, David A. Maltz, Ming Zhang, *Towards Highly Reliable Enterprise Network Services Via Inference of Multi-level Dependencies*, SIGCOMM, pp. 13-24, 2007

[3] Marc Stoecklin, *Design and Implementation of a Flow-based Network Anomaly Detection Module* Maste's Theses project report (Responsible: Prof. Jean-Yves Le Boudec EPFL, Technical Advisor: Dr. Andreas Kind), February, 2007

[4] Alexandru Caracas, Andreas Kind, Dieter Gantenbein, Stefan Fussenegger, Dimitrios Dechouniotis, *Mining Semantic Relations using NetFlow*, Third IEEE/IFIP International Workshop on Business-driven IT Management, 2008

[5] Thomas Karagiannis,Konstantina Papagiannaki, Michalis Faloutsos, *BLINC: Multilevel Traffic Classification in the Dark*, ACM SIGCOMM Computer Communication Review, vol. 35, no. 4, pp. 229-240, October, 2005

[6] Andreas Kind, Dieter Gantenbein, Hiroaki Etoh, *Relationship Discovery with NetFlow to Enable Business-Driven IT Management*, 1st IEEE/IFIP Int. Workshop on Business-Driven IT Management, 2006

[7] Marios Iliofotou, Prashanth Pappu, Michalis Faloutsos, Michael Mitzenmacher, Sumeet Singh, George Varghese, *Network Monitoring using Traffic Dispersion Graphs (TDGs)*, Proceedings of the 7th ACM SIGCOMM conference on Internet measurement, pp. 315-320, 2007

[8] `http://www.ietf.org/html.charters/ipfix-charter.html` April 11, 2008

[9] Office of Government Commerce (OGC) *ITIL Service Operation*, London, The Stationery Office (TSO) , 2007

Acronyms

CI Configuration Item

CMDB Configuration Management Database

ITIL IT Infrastructure Library

TADDM Tivoli Application Dependency Discovery Manager