Mining Semantic Relations using NetFlow

Alexandru Caracaş, Andreas Kind, Dieter Gantenbein
{xan,ank,dga}@zurich.ibm.com

Stefan Fussenegger
stf@molindo.at

Dimitrios Dechouniotis
ddexouni@ece.upatras.gr

***Abstract*—Knowing the dependencies among computing assets and services provides insights into the computing and business landscape, therefore, facilitating low-risk timely changes in support of a business-driven IT management. In general, the results of a dependency analysis can be used for infrastructure reengineering, show evidence of policy and process compliance, and support assessments of business resilience. Current passive discovery approaches using network monitoring analyze only direct communication between assets and provide just a single-link mesh view. This work introduces a new algorithm based on NetFlow data preprocessed by the Aurora system developed at IBM Research to create a dependency model of the network. The algorithm uses time-based event correlation and the data mining concept of association rules to detect and classify dependencies that span two or more components. The advantages of the algorithm is that no access credentials are required and no packet payload inspection is performed. The suggested algorithm populates and maintains a dependency model of an observed network that describes dependencies among computer systems, software components, and services. The model combines the mined association rules that express relations between flows into dependencies, which are given intuitive semantics. Tests with simulated and authentic data prove the accuracy of the dependency mining algorithm.**

I. Introduction

There lies a high benefit for IT systems management in knowing the dependencies among assets and services: better insights into the computing and business landscape facilitating low-risk timely changes in support of a business-driven IT management. However, accurate discovery of meaningful dependencies among computing assets is a challenging task.

There are two main approaches to discover asset dependencies: *active querying of assets* and *monitoring network traffic*. Both approaches have shortcomings: obtaining credentials for active discovery using remote access to hundreds of machines is challenging and time consuming. Current passive discovery approaches using network monitoring analyze only direct communication between assets and provide merely a single-link mesh view. Most products deal with the gathering and reconciliation of dependency information from various data sources, partially restricted access to networked components hinders the creation of complete end-to-end dependency models. In this case, monitoring the network represents a promising source of complementary information.

Our contribution is a new algorithm employing passive network monitoring techniques based on NetFlow, requiring no access credentials, and which does not inspect packet payloads, hence, suitable also for encrypted traffic. The algorithm facilitates creating a dependency model of assets and services with relations spanning multiple components. Information obtained from the algorithm enhances detailed inventories of assets and services obtained from active discovery. We use NetFlow to surface network information which complements typical systems management data sources when the data discovered requires more complete and accurate information despite partial lack of access to certain application servers.

Figure 1 shows two systems on the left-hand side that communicate to the services running on ports 80 and 345 on the center system. The center system acts as a relay and uses the two systems on the right-hand side. The top right system is going to be stopped for maintenance. Based only on raw traffic data, it is not possible to determine which of the applications on the left-hand side will be affected by this action. Using the dependency detection algorithm one can infer which of the two servers will be affected.

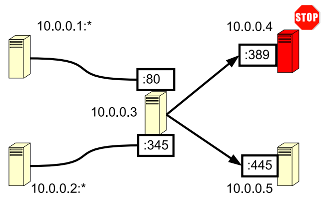

Fig. 1. Unknown affected servers based on raw monitoring.

Figure 2 depicts a possible result of the dependency analysis, which shows that only the upper left system is affected by the stopping of the right-hand system.

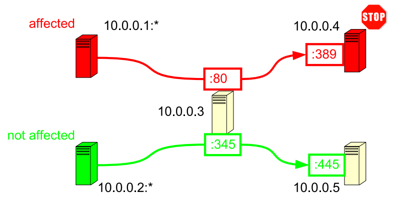

Fig. 2. Inference of affected servers based on dependency analysis.

II. Dependency Mining

Previous work [2] introduced the basic technique use of NetFlow to passively discover dependencies based on time-series correlated events. The work in [1] refined the initial ideas by using a fuzzy inference engine to distinguish meaningful relationships from noise. However, both techniques fail short if the communication is heavily aggregated (overlapping NetFlow events) as in the Figure 3.

A. Algorithm Idea

Our main algorithm idea is to identify correlated events and create association rules between pairs of flows in a dependency

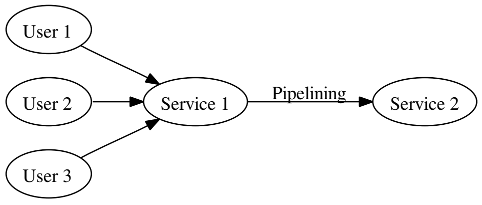

Fig. 3. Aggregation Problem

model. We extended the definition of correlated events from [2] to account for overlapping events (using both start and end times), see Figure 4. Such correlated events are then used to determine association rules of the form $f_1 \Rightarrow f_2$. In natural language, the rule states that if events from a flow f_1 are observed then those events are correlated with events from flow f_2 with a certain accuracy and correlation values.

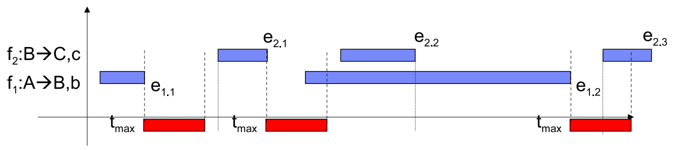

Fig. 4. Example of flow pairs

The accuracy of a rule is the number of instances that the rule predicts correctly, expressed as a proportion of all instances it applies to. The coverage of an association rule is the number of instances for which its prediction is correct [3].

Considering the example in Figure 4, the rule $f_1 \Rightarrow f_2$ has a coverage of 1 and an accuracy of 0.5. For the rule $f_2 \Rightarrow f_1$, the coverage is 2 and the accuracy is 0.67.

Based on the validity of association rules involving pairs of flows, we define semantic classes for relationship such as *strong dependency* and *conditional dependency*. Association rules are a powerful concept and are easily extensible to include boolean operators combining several flows, thus creating flow chains which show end-to-end business level semantic relationships. The full description of the algorithm for mining dependencies with combination of association rules and evaluation results will be published in full paper.

B. Implementation and Evaluation

The dependency mining algorithm is implemented in Java as a service with a loose coupling between input and output. There are several methods of feeding the algorithm with NetFlow data. The most important design and implementation consideration allows the algorithm to run continuously. For this purpose in-memory data structures are used to keep track of valid flow pairs. For each new flow event the statistics are updated. The resulting dependency rules can be queried at any time, and the results can be exported in various plain text formats or XML.

1) Example Results: The following shows an example dependency rule output in XML format. The dependency has a length of 2 and connects three end-points (or components) with a strong dependency semantic. The rule states that if a flow is observed from 10.0.0.1 to 10.0.0.3 on port 80, in 87.1% of the 27 instances of this observed flow, a flow from 10.0.0.3 to 10.0.0.4 on port 389 will be observed.

```
<dependency length="2" valid="true" coverage="27"
                         accuracy="0.871" semantic="strong">
    <component idref="0:10.0.0.1" />
    <component idref="0:10.0.0.3:tcp/80" />
    <component idref="0:10.0.0.4:tcp/389" />
</dependency>
```

The above rule shows evidence to support the business scenario described in the introduction by surfacing a strong dependency between assets.

An important remark is that new algorithm produces more false positives if the frequency of flow events is high. However, we argue that false positives can be preferred, for instance, when applied to business impact and root cause analysis.

Our approach produces the same correct results as the algorithm presented in [1]. Namely, we detect all known correlated flows with a very high confidence (on average 0.95) value and a coverage equal to the number of known introduced correlated event flows. The algorithm is able to converge quickly and identify dependencies with a high accuracy and coverage values based on the thresholds set.

III. Summary

We presented a new algorithm for dependency discovery using NetFlow data based on the ideas of Kind *et al.* [2], who suggested a dependency discovery algorithm using correlation of flow events. The new algorithm uses association rule mining, in which every dependency is backed by four different association rules. A lookup-table was devised in which every possible combination of rules is assigned to a semantic class. Besides determining whether a dependency exists, these semantic classes also allow conclusions about the dependency type. The dependencies discovered provide transparent assessment of business resilience and support timely business reengineering decisions.

One important finding is that a comprehensive, more accurate solution for dependency discovery requires knowledge from multiple sources using different techniques such as active discovery. By using an integrated approach, it would be possible to reconcile running software components with the dependencies discovered. For example, more than one port could be assigned to a software component, which would significantly reduce the number of actual dependencies.

References

[1] D. Dechouniotis, X. A. Dimitropoulos, A. Kind, and S. Denazis. Dependency detection using a fuzzy engine. In *Proc. of the 18th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM*, volume 4785 of *LNCS*, pages 110–121. Springer, 2007.

[2] A. Kind, D. Gantenbein, and H. Etoh. Relationship discovery with NetFlow to enable business-driven IT management. In *Proceedings of Business-Driven IT Manageent (BDIM'06)*, pages 63–70, December 2006.

[3] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, October 1999.