Efficient Multidimensional Aggregation for Large Scale Monitoring

Lautaro Dolberg, Jérôme François, Thomas Engel
University of Luxembourg
SnT - Interdiciplinary Centre for Security, Reliability and Trust.
Email: firstname.lastname@uni.lu

Abstract

Today, network monitoring becomes necessary on many levels: Internet Service Providers, large companies as well as smaller entities. Since network monitoring supports many applications in various fields (security, service provisioning, etc), it may consider multiple sources of information such as network traffic, user activity, network events and logs, etc. All these ones produce voluminous amount of data which need to be stored, visualized and analyzed for administration purposes. Various techniques to cope with scalability have been proposed as for example sampling or aggregation.

In this paper, we introduce an aggregation technique which is able to handle multiple kinds of dimension, *i.e.* features, like traffic capture or host locations, without giving any preference a priori to a particular feature for ordering the aggregation process among dimensions. Furthermore, feature space granularity is determined on the fly depending on the desired events to monitor. We propose optimizations to keep the computational overhead low.

In particular, the technique is applied to network related data involving multiple dimensions: source and destination IP addresses, services, geographical location of hosts, DNS names, etc. Thus, our approach is validated through multiple scenarios using different dimensions, measuring the impact of the aggregation process and the optimizations as well as by highlighting the ability to figure out important facts or changes in the network.

1 Introduction

Monitoring is a fundamental part of network management. It is essential for checking the network activity and status, e.g. tracking abnormal facts or changes (attacks, configuration errors, failures, etc). Several steps are required for this. First, data has to be collected from various sources and locations. Then, such data must be stored before it can be directly visualized or analized by human expert to provide summarized information, like alarms, to the network operational team.

For example, usual data collected in network might consist of full packet captures on a network or Netflow [9] records, for ISPs (Internet Service Provider), which are known to be helpful in network management context [10]. DNS traffic is also a valuable information in the security context for detecting botnets [6] and malicious domains hosting malware [1]. Geographical or network location of hosts might be helpful for placing servers at the right place in particular within a CDN (Content Distribution Network). Application level analysis can include monitoring of different exchange message types for a given protocol. For instance, a simple counter for error messages in SIP or HTTP may indicate some problems on the network. IDS (Intrusion Detection Systems) or firewall alerts are clearly relevant by nature.

Thus, there is plenty of valuable information which might be also correlated together, network traffic, DNS names or host locations, etc.

Since nowadays volume of such information grows rapidly, scalability represents a challenge. When information volume results to be massive, computing resources infrastructure (storage and analysis) demands for fine grained information, such as deep packet inspection, are too high to be practicable, in some cases inaccurate. For example, thanks to our partners involved in the operational field in Luxembourg, we obtain a Netflow and a DNS dataset. The average number of flows is 60,000/sec and can reach 100,000/sec. For forensics analysis, we faced with the a huge size of the DNS dataset including around 40M unique records over a 10 months period.

In this paper, we improve network monitoring by targeting scalability issues for storage, analysis and visualization of huge volumes of network related data. We analyze multiple sources such as traffic flows or DNS records to infer a global knowledge that might be helpful to identify particular events (normal or abnormal). Recent research has shown promising results for aggregation based techniques on macroscopic monitoring [8, 34, 20]. Although we explored aggregation over destination or source IP address separately in [34], this paper proposes a new method implemented as an open source tool which:

- handles multiple types of data
- defines a new tree based structure and algorithms to combine them into a single tree,
- is evaluated in multiple scenarios
- can be easily extended to include new features

Hence, we propose *MAM* (Multidimensional Aggregation Monitoring) is proposed for network related data aggregation including many features (also called dimensions) on very large collections. The goal of this paper is to present *MAM* from a formal point of view as well as from a practical point of view and study its viability for network monitoring purposes.

MAM can consider simultaneously the source and destination IP addresses and the ports of network traffic. A fundamental idea of our approach is to aggregate data, such as traffic load, over multiple dimensions without giving any preference to one. Considering the previous example, it means that the data will not necessarily be aggregated on source IP addresses first and then on ports. Usually, this decision is made by human experts using their own expertise. Some of them will monitor the usage per service first (ports) and then per IP addresses whereas others prefer to have statistics per IP addresses first and then the details for each service. In the first case, statistics about global traffic of an IP address are not directly available and need to be reconstructed by iterating over all ports. Adding more dimensions will add other levels that make the choice of ordering difficult and computational less efficient if some statistics need to be reconstructed afterwards.

Moreover, IP addresses monitoring usually relies on a subnet basis (/X networks) where the human administrator has to specify the prefix size X to use. In our case, the space of a dimension is not split a priori like a fixed size of subnets. The aggregation can thus lead to keep track of information related to IP subnets having different sizes.

In fact, the aggregation process is guided by the desired events to monitor, `i.e.` reaching a quantifiable visibility (5% of the traffic load in bytes or packets for instance) which will alleviate the administrator from both problematic decisions (order of dimensions and split over the space).

Scalability can be achieved by optimizing the usage of storage data structures. *MAM* leverages a tree based structure to store data in a hierarchical representation. It has a bounded size and different strategies are described to satisfy this constraint. Obviously, the strategy can impact the relevance of stored data as well as the aggregation process itself.

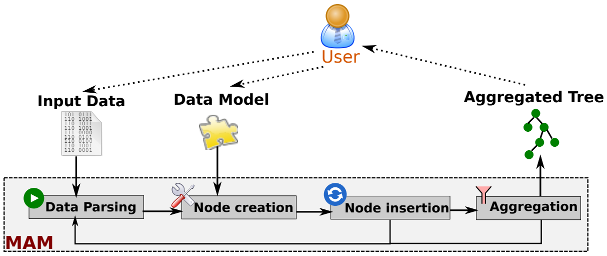


Figure 1: MAM overview

This paper is organized as follows. Section 2 overviews the proposed tool. Section 3 is related work. Section 4 formally defines the data structures for multidimensional aggregation. Section 5 explains spatial aggregation techniques. Section 6 proposes several strategies to cope with scalability. Section 7 is dedicated to the evaluation. Finally, Section 8 concludes the paper and describes our future work.

2 MAM Overview

Figure 1 highlights the main steps of our tool and so the main components. The objective is to aggregate multidimensional data into a single tree structure for each time window (temporal dimension). If the timing information is not provided, *MAM* can still be used to create a single tree for the whole dataset.

For being executed, the user has to provide input data as well as a data model describing the type of data and how to parse and aggregate it. Some data models are already provided with the tool as for example the IP address, services or geographical coordinates. Therefore, the first step is to parse the input data. For each data instance, (usually a line in a file), *MAM* creates a node accordingly. The latter is inserted into the current tree based on the hierarchy of each dimension (next sections provide details). These steps are repeated until the tree has to be compressed, `i.e.` nodes in the tree are aggregated from the leaves to the root for keeping solely relevant and aggregated information. This happens when the size of the tree is too high (online aggregation) for reducing resources consumption or at the end of a time window (simple aggregation). In this case, the tree is returned as a result to the user and *MAM* will continue the process starting from parsing next time window.

3 Related Work

Regarding scalability concerns, flow based information such as Netflow records [9], compresses IP traffic in a compact format [9]. Even mainly supported by most commercial routers, they are still known to require a large storage infrastructure, since Netflow records growth can be undefined.

Thus, sampling techniques have been proposed in the past [11, 29]. Sampling may clearly resolve the scalability issues by discarding large portion of data. However, the main drawback is the difficulty to set an appropriate sampling rate to discard enough information without impacting the relevance of observations made from the remaining data.

Using efficient data structures to optimize storage and access was also explored. Giura et al. [17] introduce NetStore, an efficient storage infrastructure for network flow data using a column-oriented storage that outperforms row-based techniques. BadHoods[27] performs aggregation on a subnet basis to demonstrate that malicious hosts tend to be close into the IP space but they highlight the difficulty to set appropriately the prefix size of subnets to monitor. In addition, Fenwick trees [12] handle single dimension by storing efficiently prefix sums for given values represented as a table.

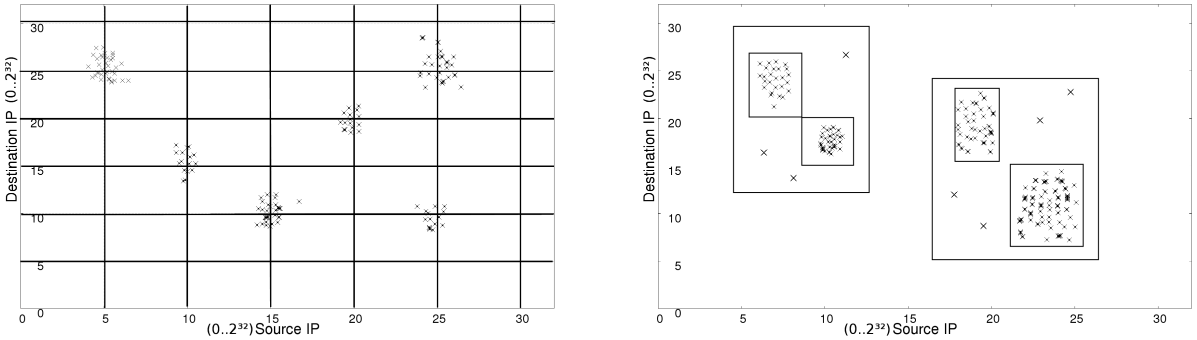
Flow aggregation based on entropy is proposed in [19] where the authors introduce a two dimensional hashtable (source and destination IP addresses). An alternative is to collect data from multiple sources as proposed in [35] to build a community graph for sharing information about host activities based on IP addresses. The scalability is addressed through data filtering as well as using multiple graph construction which can be achieved independently in parallel. In the past, hosts interaction is also a manner to compress data into a graph representation. This is known to be useful for detecting anomalies like botnets [15] in particular when instantiated with the MapReduce paradigm for being executed in a distributed fashion [14]. Another solution for traffic aggregation was presented in Aguri [8] and further on in our previous work Danak [34]. Both of them leverages a tree based structure for storing traffic data on a single dimension. IP addresses are represented by a tree following the common hierarchy of the subnets. Using such structures, it has been proved that anomalies in large scale networks can be tracked as for instance spam or distributed denial of service.

Other sources of information are relevant as noticed in introduction. Assuming DNS data, a context-aware clustering method is proposed in [31]. In fact, two trees are created. One for representing the IP subnets like in [8] and one for representing the DNS domains in a similar manner based on the parent-child relations between subdomains. Log analysis is also concerned and recent advances like [21] promote the efficient programming design as well as data structure for correlating log events in a scalable way.

From a general point of view, storing multi-dimensional data has been investigated in the past. While data structures cited in [5, 13] are plausible modeling tools for K-Dimensional spaces, they are recommended to be used on regular partitioned spaces. Therefore, the previous mentioned data structures requires a pre-established order among dimensions. Some contributions to this subject was made by [28] on Flamingo, and EtherApe [3, 33]. Regarding Data Cube [18], maintaining the whole cube structure is costly from a computational point of view. Furthermore, data requires to be stored within a relational structure and aggregation needs to be fully specified by the user in particular the granularity and the order among dimensions unlike our approach. Some cube approaches fail to process online data since they become no longer valid after new data has been added. In case of our tool, it is possible to append nodes on the fly without additional cost. Flamingo is a visualization tool for monitoring Internet traffic in real time, rendering 3D images. Aggregation is also applied by IP prefix, *i.e.* subnets, and the prefixes are chosen based on routing table entries. EtherApe is a network traffic monitoring tool. Network traffic is displayed as nodes, as for example IP addresses, and links with color codes to represent different protocols.

Some tools are also dedicated to visualization. ENAVis [23] represents multi-dimensional data into a graph where each node contains a single dimension. This helps to see relationships between users and hosts for example. Even if similar to our work, the main fundamental difference is that we chose a tree-based structure where nodes represent multi-dimensional instances of data. Furthermore, our approach integrates an online aggregation technique for automatically split the multi-dimensional space in subspaces. SIFT [36] includes IP subnet aggregation but the user still has to set the granularity of details. This is different from our approach, the same tree subnets can be visualized once they reach a certain threshold of measurable information like the traffic load.

Also our work is mainly complementary to [34, 8], *MAM* aims to aggregate relevant data on several spatial dimensions (IP addresses, ports, etc) like [23, 28]. However, it differs from these latter since the order between dimensions is not defined a priori as well as the space division is automatically determined.



(a) Regular IP Address space partitioning

(b) Dynamic IP Address space partitioning

Figure 2: Different IP Address space partitioning examples

4 Data Aggregation Structures

Aggregation consists to reduce granularity of data by grouping data instances according to a criteria, *i.e.* sharing similar properties.

For instance, aggregation over the IP address space could consider the shared prefix as the property leading to monitor network traffic by subnet. Defining the size of the subnet, equivalent to the prefix size, is quite arbitrary like /24, /28, etc. Changing this parameter leads to different observations regarding the context [27]. This is illustrated in Figure 2 with a naive case example where a regular partition of IP address space, in Figure 2(a), may not lead to identify the clear subgroups of hosts unlike a dynamic partitioning in Figure 2(b). For example, to monitor the hosts listed on Traffic Flow Table 1, monitoring /16 networks will give a general overview of the network activity which is probably too coarse and /24 seems more useful. However, using /24 or more can be too fine-grained at the Internet level. Moreover, since traffic is probably not well-balanced between machines and subnets, some of them should be more carefully monitored, equivalent to consider higher prefix size, while others may be monitored with a coarse-grained view (small prefix size).

To counter this problem, aggregation can be guided by the nature of the events to monitor. For example, the traffic load can be aggregated in order to observe phenomenas reaching a certain proportion of the entire traffic load or of IDS alerts.

For aggregation on a single dimension, a tree structure is suitable [34, 8]. Spatial representation of a bi-dimensional space can be done using a quad tree structure [13] where each internal node has exactly four children. Normally the space is recursively portioned into four quadrants or regions. Then for partitioning a three dimensional space a similar structure, an oct-tree, can be used. A general structure supporting M dimensions is a multidimensional tree (k-d Tree [5]) for k-dimensional space-partitioning. However, in our context, the dimensional space division is not known in advance and done on the fly when the tree is created.

In this paper, we consider features where it is possible to derive an underlying hierarchical relationships covering all potential data instances represented as nodes in a tree. Assuming two data nodes, one is qualified as more specific or there is no relationship between them. Formally the hierarchical relationship between two nodes n_i, n_j represent is given if there is a path from the root of the tree to n_j that passes through n_i. In that case, n_i is more general than n_j.

4.1 Single Dimension

Spatial and temporal aggregation on a single dimension for traffic flows was proposed in Aguri [8] and Danak[34]. Temporal aggregation splits the dataset into fixed size time windows on which spatial aggregation is applied. We extend the notion of spatial aggregation to multiple and generic dimensions (features).

IP address aggregation is performed by extracting the traffic volume (bytes or packets) for the source or destination addresses. Aggregation is based on a tree structure following the common subnet hierarchy where each node represents an IP subnet or a single address. The total volume of traffic transmitted is decomposed in particular volumes expressed as proportions or percentages. Nodes with a proportion of traffic lower than α are aggregated into their parents. An example of this is given in Figure 3 from Traffic Flow Table 1. In this small example, only nodes with more than 10% of the total traffic are kept. As highlighted, root concentrates the global traffic of a /17 network. Each node contains the following information:

1. Dimension name and value (i.e: {app:ROOT}, {src_ip:0.0.0.0/0}, etc)

Traffic Flow Table 1 Traffic flow example of a network nodes within 192.168.0.0/16 (Web and Mail services)

PORT	PROTO	KB	TIME	SOURCE	DEST
80	TCP	1491	2010−02−24 02:20:15	192.168.6.2	92.250.221.82
110	TCP	988	2010−02−24 02:20:19	192.168.8.2	92.250.223.87
443	TCP	902	2010−02−24 02:20:27	192.168.11.2	92.250.220.82
110	TCP	1513	2010−02−24 02:20:29	192.168.112.1	92.250.222.81
80	TCP	1205	2010−02−24 02:20:29	192.168.11.1	92.250.220.82
80	TCP	1491	2010−02−24 02:20:31	192.168.1.2	92.250.220.83
110	TCP	1467	2010−02−24 02:20:39	192.168.12.2	92.250.221.81
80	TCP	927	2010−02−24 02:20:39	192.168.12.2	92.250.220.82
443	TCP	1294	2010−02−24 02:20:39	192.168.11.1	92.250.223.82
110	TCP	940	2010−02−24 02:20:49	192.168.21.2	92.250.221.81
80	TCP	917	2010−02−24 02:20:49	192.168.23.1	92.250.220.82
443	TCP	460	2010−02−24 02:20:59	192.168.26.2	92.250.220.85

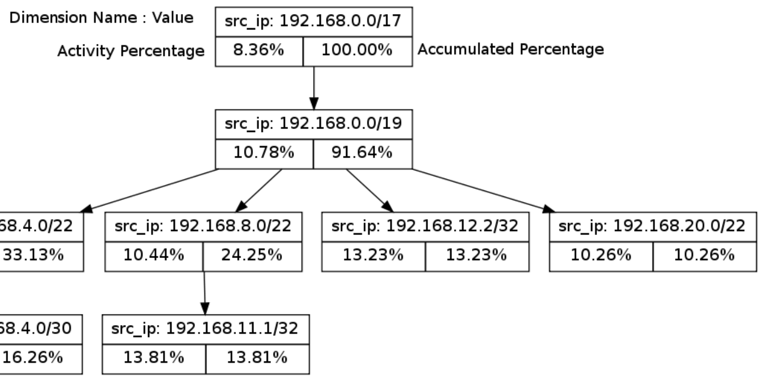


Figure 3: Single dimension tree (source IP addresses) based on Traffic Flow Table 1, activity volume: number of bytes, $\alpha = 10\%$

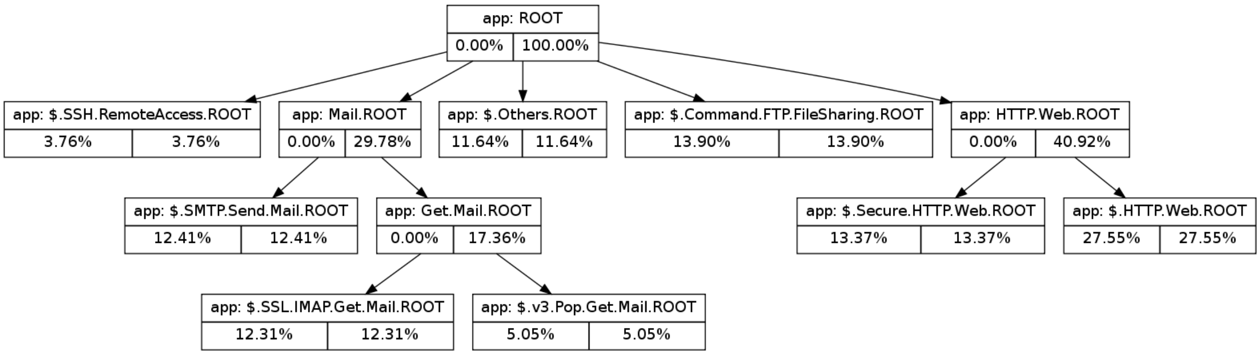


Figure 4: Single dimension tree (application) based on Traffic Flow Table 3, activity volume: number of bytes, $\alpha = 5\%$

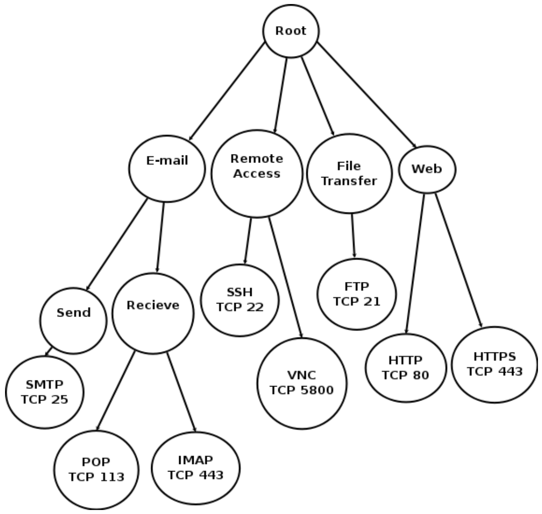


Figure 5: Example Application Taxonomy by TCP Port Numbers

2. Percentage of aggregated activity (activity volume defined as *vol*) for the current node

3. Cumulated percentage of activity of the node and its subtree defined as *acc_vol*

Intuitively, a single dimension tree represent a subset of the entire hierarchy (all the possible values) of a given feature like illustrated in Figure 3.

Formally an IP address single dimension tree of N nodes is [34]:

- A set of N nodes, where T = $\{n_0 \dots n_N\}$ and $n_i =< prefix_i, prefix\_length_i, vol_i >$. IP subnets are decomposed using CIDR format [16]. $prefix_i$ and $prefix\_length_i$ are the prefix value and size of node n_i while vol_i is the entire traffic load associated to the IP addresses included in the subnet n_i.

- A parent-child relationship where a $child : T \rightarrow \mathscr{P}(T)$ returns a set of child nodes for a given node.

Single dimension aggregation can also be done for many other attributes such as protocol messages, port numbers, spatial coordinates. Aggregation for TCP port numbers, using data from Traffic Flow Table 3, is highlighted in Figure 4. In this case, every node represents an application family or a specific one defined by the taxonomic classification in Figure 5. In our paper, a dimension is a feature which the values may be represented in a hierarchical tree with final values at the leaf nodes and group values as internal nodes.

This definition can be extended for a generic dimension tree as follow:

- A set of N nodes, where T = $\{n_0 \dots n_N\}$ and $n_i =< f_i, vol_i >$. Where f_i is an associative array modeling a dimension from a given traffic flow. For example in case of application port $f_i = \{app : value_i\}$ where *app* is a label and $value_i$ is the a string modeling a path in the taxonomic tree described in Figure 5. As mentioned before, it can a full path to a leaf or an intermediate branch describing a sub family of applications. For IP addresses, f_i is $\{prefix : prefix_i\ ,\ prefix\_length : prefix\_length_i\}$.

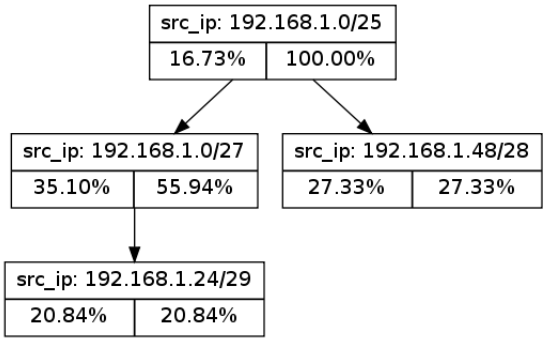


Figure 6: Single dimension tree (Source IP addresses) from Traffic Flow Table2

- A parent-child relationship where a $child : T \rightarrow \mathscr{P}(T)$ returns a set of child nodes for a given node.

Single dimension aggregation has proven to be an effective technique for practical network analysis methods and anomalous network traffic detection [34, 8]. However, information from network traffic includes more than one dimension. Furthermore, the anomalies can be present in a combination of dimensions.

A port scanning has to consider the application/service feature while IP scanning has to monitor IP addresses. Assuming a botnet doing port scanning from and to multiple IP addresses, all these features (source and destination IP addresses, destination ports) need to be monitored meanwhile to observe the attack activity globally. So predicting the single dimension or the combination of dimensions to monitor is hard.

Considering Traffic Flow Table 2 illustrating a naive case of several hosts performing a DDoS against a web server. IP address based aggregation (illustrated in Figure 6) cannot clearly detect it but aggregation using TCP port will do. Another scenario is Traffic Flow Table 3. In this case, a reduced group of hosts are targeting several applications which cannot be caught by aggregation on TCP ports. However, the multidimensional tree depicted in Figure 7 is able to detect this behavior. This highlights that 20% of the traffic is due to web session initiations (HTTP) in 192.168.0.0/20 subnet.

4.2 Multiple Dimensions

To extend the previous approach, we present an aggregation technique for monitoring network using multiple dimensions at the same time (eg: IP Address, Full Qualified Domain Name (FQDN), TCP ports, GPS Coordinates, etc). Multidimensional aggregation is performed by using a user-defined threshold α and an interval of η seconds. The latter defines the size of a time window. For each of them, the data is assembled into a tree composed of nodes, including multiple dimensions, where only those such that $acc\_vol > \alpha$ are kept.

In Figure 7, source and destination IP addresses and application from Traffic Flow Table 3 are aggregated.

Assuming a data instance that can be decomposed in many dimensions, a formal definition of a multidimensional tree of M dimensions and N nodes is as follows:

- A set of M associative arrays that model the M dimensions
- A set of N nodes, where T = $\{n_0 \dots n_N\}$ and $n_i = < \{f_{i_1} \dots f_{i_m}\}, vol_i >$, $f_{i_j} \in \{f_{i_1} \dots f_{i_m}\}$ is an associative array modeling the $j-th$ dimension according to the previous definition

  Thus, we can define $f_{i_1} = \{prefix : prefix_i \;,\; prefix\_length : prefix\_length_i\}$ for IP addresses. For the application/service level, $f_{i_3} = \{app : value_i\}$ where app is a fixed label and $value_i$ is the a string modeling a path in the taxonomic tree described in Figure 5.
- A parent-child relationship where a $child : T \rightarrow \mathscr{P}(T)$ returns a set of child nodes for a given node.

5 Tree Based representations

The core idea of a tree-based aggregation mechanism is to aggregate multi dimensional data for creating a summary of the network activity for each time window. Aggregation can be used for post-analysis or for real time monitoring. In the first scenario, memory and computational cost can be rise considerably for reaching a high level of precision. However in the case of real time monitoring execution time is a real constraint. Thus, two optimization strategies are introduced in Section 6 for real time computing that may lead to a particle approach for monitoring.

In this Section, we refer to the simple aggregation which is executed at the end of each time windows.

5.1 Simple aggregation overview

As long as a single data instance is inputed, a leaf node is built after extracting relevant information (dimensions and values). to be inserted at the right place (or updating the node in the tree if it already exists) creating intermediate nodes might be necessary, like nodes representing intermediate IP subnets. If the node already exists, vol_i is updated accordingly. For producing an outline afterwards, the tree is traversed post-order to aggregate nodes and to compute cumulative percentages (acc_vol). If the activity volume, vol_i, of a node n_i is less than the aggregation threshold, α, it is aggregated to its parent node, n_j. Thus, the node n_i is removed, vol_i is added to vol_j and all child nodes of n_i are attached to n_j. This allows to delete intermediate nodes which do not represent large activity volumes unlike their child nodes. Otherwise ($vol_i < \alpha$), the node is kept as it is.

Moreover, during the post-order traversal, activity volumes, vol_i are computed as percentages. In fact, they are stored as absolute values during the tree construction since the total activity volume is not known. At the end, thanks a global counter, $VOL = \sum_i vol_i$, each vol_i is updated accordingly, `i.e.` vol_i/VOL.

5.2 Directions

Due to the hierarchical relationships, we suppose there is a strict order relation between values of the same dimension. For constructing the multidimensional prefix tree structure, developing the concept of directions is necessary. Intuitively the directions correspond to find the correct path in the tree to attach a node or to navigate within the tree in order to access a given node.

Some dimensions are more likely to find or define a natural direction. For example, IP addresses have two directions, 0 or 1, modeling the next bit value. Regarding the subnet X.Y.Z.0/24, all IP addresses which the 25th bit is 1 (like X.Y.Z.128/25) will be placed on the left branch while every others which this bit is 0 will be placed on the right branch.

Other dimensions like UDP/TCP ports can be compared as integers but it is not so relevant as services of the same category like mail are not necessarily neighbors in the port range. In this case, a hierarchical classification, like in Figure 5, is required. In our tool the direction function may be customized by the user. Direction function is used to label the parent-child relationship on the multidimensional tree. Therefore, we also consider the longest common prefix which represents the most specific common ancestor between two nodes

Assuming $n_i = < \{f_{i_1} \dots f_{i_m}\}, vol_i >$ and $n_j = < \{f_{j_1} \dots f_{j_m}\}, vol_j >$, the longest common prefix is defined as:

$$lcp(n_i, n_j) = < \{f_{lcp_1} \dots f_{lcp_m}\} \quad (1)$$

where f_{lcp_i} is the most specific common part for the ith dimension, which corresponds to the longest sequence of

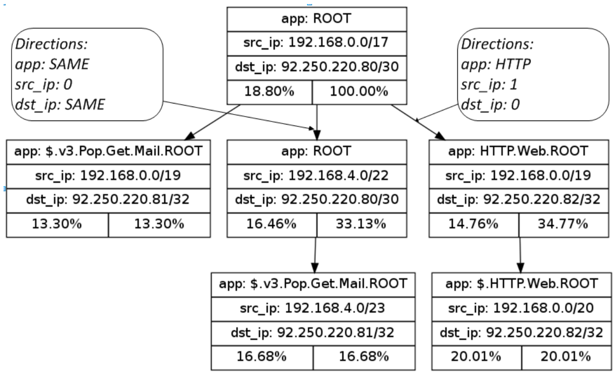


Figure 7: Multiple dimension aggregation based on Traffic Flow Table 3, $\alpha = 10\%$

Traffic Flow Table 2 Traffic Flow Table showing an example for a possible DDoS against a web server.

PORT	PROTO	KB	TIME	SOURCE	DEST
80	TCP	895	2010−02−24 02:20:59	192.168.1.17	92.250.220.82
80	TCP	47	2010−02−24 02:20:59	192.168.1.25	92.250.220.82
80	TCP	570	2010−02−24 02:20:59	192.168.1.45	92.250.220.82
80	TCP	952	2010−02−24 02:20:59	192.168.1.44	92.250.220.82
80	TCP	408	2010−02−24 02:20:59	192.168.1.61	92.250.220.82
80	TCP	609	2010−02−24 02:20:59	192.168.1.9	92.250.220.82
80	TCP	690	2010−02−24 02:20:59	192.168.1.15	92.250.220.82
80	TCP	88	2010−02−24 02:20:59	192.168.1.29	92.250.220.82
80	TCP	997	2010−02−24 02:20:59	192.168.1.27	92.250.220.82
80	TCP	650	2010−02−24 02:20:59	192.168.1.9	92.250.220.82
80	TCP	298	2010−02−24 02:20:59	192.168.1.46	92.250.220.82
80	TCP	502	2010−02−24 02:20:59	192.168.1.52	92.250.220.82

Traffic Flow Table 3 Traffic Flow Table example for a destination address being targeted by reduced group of host .

PORT	PROTO	KB	TIME	SOURCE	DEST
25	TCP	4660	2010−02−24 02:20:59	192.168.1.2	92.250.220.82
443	TCP	2417	2010−02−24 02:20:59	192.168.1.2	92.250.220.82
443	TCP	1945	2010−02−24 02:20:59	192.168.1.2	92.250.220.82
21	TCP	4206	2010−02−24 02:20:59	192.168.1.1	92.250.220.82
80	TCP	4336	2010−02−24 02:20:59	192.168.1.3	92.250.220.82
110	TCP	2110	2010−02−24 02:20:59	192.168.1.2	92.250.220.82
23	TCP	4257	2010−02−24 02:20:59	192.168.1.1	92.250.220.82
25	TCP	2005	2010−02−24 02:20:59	192.168.1.3	92.250.220.82
993	TCP	2434	2010−02−24 02:20:59	192.168.1.1	92.250.220.82
443	TCP	3270	2010−02−24 02:20:59	192.168.1.2	92.250.220.82
993	TCP	4775	2010−02−24 02:20:59	192.168.1.2	92.250.220.82
22	TCP	690	2010−02−24 02:20:59	192.168.1.3	92.250.220.82

directions. Therefore, for IP address, this is a sequence of bits which is similar to the standard definition.

Considering T_M a M dimensional tree, defined in Section 4.2, a multi-dimensional direction is defined as a tuple of M directions (one for each dimension):

$$\forall n_i \in T_M, n_j \in T_M, n_k \in T_M, n_j \in child(n_i), n_k \in child(n_i)$$
$$n_k \neq n_j \iff direction(n_i, n_k) \neq direction(n_j, n_k) \quad (2)$$

This corresponds to have only one child node per unique tuple of directions. Conceptually, a direction can be any tuples that allows to distinguish child nodes. Practically, it can usually match concrete value like the bit values for IP address or the application subclass for TCP ports. Directions are illustrated in figure 7 using a 3-tuple (application, source and destination IP addresses). The direction *SAME* was introduced to allow a child node to be different from its parent on a subset of dimension values while some remain the same. Preliminary tests show that this limits the number of internal nodes. However, at least one direction of the tuple must not be *SAME*.

5.3 Tree Construction

Intuitively, the tree construction is done by creating a root and then adding nodes or updating existing ones. Based on the directions, a pre-order traversal is done looking for a match to insert the node n_i (line 4 in Algorithm 1). Assuming that the traversal stops on the node n_j, there are different cases:

- if there is a perfect match (dimension values are the same), the activity volume is updated $vol_j \leftarrow vol_j + vol_i$. This is done in line 5 of algorithm 1.
- if n_i is a child n_j, a new child node is created by computing the directions tuple from n_j to n_i (there is not yet a node in this position otherwise the traversal should have continue). This is done in line of algorithm 1.
- otherwise, the traversal has followed the direction but ends in a too specific node (it happens because no all possible internal nodes are cerated for scalability issue, for instance X.Y.Z.0/24 may be a child of X.Y.0.0/16 on the IP address dimension). In this case, a new branching point (internal node) is created from $lcp(n_i, n_j)$. So n_i and n_j are the child nodes and directions are then computed, *i.e.* $direction(n_i, lcp(n_i, n_j))$ and $direction(n_j, lcp(n_i, n_j))$. This is done line 15 of algorithm 1.

Therefore, by construction, every node represents a subspace of its parent according to all dimensions.

Once the tree construction is finished (end of the time window), aggregation takes place. Aggregation is done by traversing the tree in post order to find nodes having a activity volume $vol_i \leq \alpha$. These nodes are aggregated into their parents. By doing this, directions are discarded since they are only needed for the construction and because they may not satisfy equation (2) due to the deletion of internal nodes. Therefore, the k-dimensional space is not divided a priori and the space is not split at regular intervals. This allows, again, to have an irregular granularity over the dimensions for efficiently monitoring of the targeted events.

Algorithm 1 Update Tree $insert\_node(tree, n_i)$

```
1: if $tree\ is\ empty$ then
2:    $tree.set\_root(n_i)$
3: else
4:    $n_i \leftarrow tree.search\_matching\_node(n_i)$
5:    if $n_j$ matches all directions then
6:       update $n_j$ volume {Perfect Match}
7:    else
8:       if $n_j$ is $n_i$ child then
9:          {Partial Match}
10:         for $dim \in n_i$ do
11:            add $n_i$ to $n_j$ childs
12:         end for
13:         update $tree$ set $branching\_point$ parent of $n_j$ and $n_i$
14:      else
15:         $branch \leftarrow$ empty node {New Branch case}
16:         for $dim \in n_j$ do
17:            $branch[dim] \leftarrow Direction_{dim}(n_j[dim], n_i[dim])$
18:         end for
19:         update $tree$ set $branching\_point$ parent of $n_j$ and $n_i$
20:      end if
21:   end if
22: end if
```

6 Online Tree Aggregation Strategies

Since memory consumption grows along with the size of input data, and so the size of the tree, we developed strategies for maintaining the tree structure under a predefined size. Once the number of nodes is higher than MAX_NODES, one of the following strategy is triggered:

- Root aggregation: this strategy performs the simple aggregation (as described in the previous section) from the root

- Least Recently Used (LRU): in this case the least recently used nodes are candidates

These methods are qualified as online because they perform pre-aggregation before the end of the time window for saving memory resources.

6.1 Root Aggregation

Root aggregation algorithm is shown in Algorithm 2. Every time the tree grows over the user defined threshold, the aggregation mechanism explained in Section 5.3 is triggered. It is the most simple solution but not an efficient mechanism since all nodes below the threshold α are aggregated while the objective is to reduce the number of nodes to MAX_NODES. This can be seen in line 5 of Algorithm 2. The cost of the aggregation mechanism triggered in line 6 is $O(n \times log(n))$ [8]. Its worst case complexity is $O(n^2 \times log(n))$ where n is the number of nodes. This is because worst case scenario represents triggering aggregation after every inserted node (For Loop in line 2). Details related to mechanisms to recover from direction incoherency entailed by internal nodes deletion are included in the source files provided (source file containing Tree class definition).

Algorithm 2 Build Tree $T(dimensions, data)$

```
1: $tree \leftarrow empty\_tree$
2: for $d \in data$ do
3:     $node \leftarrow build\_node(d)$
4:     $tree.insert\_tree(node)$
5:     if $tree.size > MAX\_NODES$ then
6:         $tree.aggregate()$
7:     end if
8: end for
```

6.2 LRU Aggregation

Algorithm 3 consists of triggering aggregation on the least recently used node only. The main idea behind this mechanism is to label every node with a timestamped tag that indicates the last time it was used. This is done in Algorithm 4 and used in line 4 in Algorithm 3 for updating the timestamp of nodes which have been gone through when a node is inserted or updated (i.e. its parent and ancestor nodes).

Aggregation is performed only for the least recently used node. To achieve that, a min-max heap is employed [2] structure using as key the timestamped tag present in each node. Algorithm 4 implements this mechanism extending Algorithm 1 functionality for labeling and maintaining the heap structure used for retrieve the LRU node. This corresponds to line 14 of Algorithm 4. This operation is based on updating a min max heap which has a complexity of $O(log_2(n))$ [2]. Assuming n, the number of nodes, the average size of a tree branch is $log(n)$. If every node in the branch has to be updated, an entry on the heap must be modified. Hence the subcost of that operation is $O(log^2(n))$ and in the worst case $O(n \times log(n))$ (a single branch of n nodes). To calculate this complexity, else branch in line 5 of Algorithm 4 is going to be executed. During the last for-cycle (line 14), the list $update\_nodes$ will contain $log(n)$ elements that corresponding to the explored path to place $node$ in $tree$.

After updating the last time used timestamp, the max heap is updated with a complexity of $O(log_2(n))$. So the total complexity is $O(log^2(n) + log(n)) = O(log^2(n)$.

Algorithm 3 Build Tree LRU $T(dimensions, data)$

```
1: $tree \leftarrow empty\_tree$
2: for $d \in data$ do
3:     $node \leftarrow build\_node(d)$
4:     $tree.update\_lru\_tree(node)$
5:     if $tree.size > MAX\_NODES$ then
6:         $candidate \leftarrow tree.lru\_heap.top()$
           Get the top element, candidate to be aggregated
7:         $candidate.aggregate()$
8:     end if
9: end for
```

Algorithm 4 Update Tree $update\_lru\_tree(tree, node)$

```
1: if $tree$ $is$ $empty$ then
2:     $node.ltu \leftarrow now$
3:     $update\_nodes \leftarrow [node]$
4:     $tree.set\_root(node)$
5: else
6:     $path \leftarrow tree.insert\_node(node)$
7:     for $n$ $in$ $reverse(path)$ do
8:         $n.ltu \leftarrow now$
9:         $now \leftarrow now + 1$
10:        $update\_nodes.append(n)$
11:    end for
12: end if
13: for $n$ $in$ $update\_nodes$ do
14:     $tree.ltu\_heap.update(n)$
15: end for
```

As highlighted in the *else* statement (line 5), the timestamp is updated such that the leaf node (inserted or updated) has a timestamp older that its ancestors (reverse path is constructed during the insertion itself without any additional cost). This ensures that a leaf node is always retrieved and aggregated in line 7 in Algorithm 3. Otherwise, such an element could be an internal node and aggregation may lead to remove entire subtrees.

6.3 Issues

Since online aggregation is done by sequentially reading data, a change of data ordering will produce different results. Inserting the same data instance at the end of the window will not have the same effect than at the beginning since the tree may have already been aggregated many times before. The impact of the data ordering will be evaluated in section 7.3.

7 Evaluation

In this Section we evaluate the following main aspects of the proposed tool:

- Scenarios showing aggregation benefits with multiple types of data:
  - Netflow
  - SIP Messages
  - Geographical coordinates associated to Netflow captures
  - DNS names associated to Netflow captures
- Performance of the proposed strategies of online aggregation
- Order of data impact on aggregation accuracy

For sake of clarity, only partial trees are shown in Figures. Except when mentioned, LRU strategy is applied.

7.1 Data sets

7.1.1 Netflow

NetFlow [9] was developed by Cisco Systems and is supported by many device vendors. Thus, Netflow or other flow-based approaches are now considered as a standard for IP monitoring. A flow record groups multiple packets sharing similar properties and in particular source and destination addresses, protocols and ports. Available information includes useful information like a timestamp, number of packets or bytes exchanged. The interested reader can refer to [9] for further information.

Real Netflow captures were provided by one major ISP in Luxembourg. As assumed to be free of attacks, we also inject a realist attack in the same manner as in [34] for assessing the ability of our approach to catch valuable information about anomalies.

The duration of the capture is 26 days from 01/30/2010 to 02/24/2010 with an average number of flows aroung 60,000/sec. A total of 279815 unique IP addresses using 64470 different UDP and TCP ports are represented.

The following information is extracted:

- Timestamp
- IPv4 Source Address
- IPv4 Destination Address
- TCP or UDP source port
- TCP or UDP destination port
- Traffic Volume in bytes is considered as the activity volume (*vol*)

7.1.2 DNS Data sets

This dataset consists is enhanced compared to the previous on by performing a reverse DNS look up [24, 4] for source and destination IPv4 addresses. Specifically, on reverse DNS look up the pointer to canonical name is retrieved (PTR) [26]. The dataset generated after this method is still representative since every Internet-reachable host should have a name according to [24, 4]. The hierarchical relationship on the DNS dimension is straightforward by using the traditional order between a domain and its subdomain.

7.1.3 Geographical coordinates Data sets

This dataset contains geographical coordinates associated to source and destination IPv4 address of every flow. This is done by using GeoIP database available in [25]. Therefore, the dataset includes the same information as the Netflow one and includes also the latitude and longitude for source and destination IPv4 addresses. Defining dimensions and directions for latitude or longitude is a bit harder as it is not a discrete space. When nodes are inserted in the tree in a way that a new one has to be created, the latter is a rectangle area such that it contains all child nodes using two directions (left and right and top and bottom). This corresponds to compute minimal and maximal values for both the longitude and latitude. Another approach could have considered a taxonomy per continent, country, region, city for example.

7.1.4 SIP Messages

SIP (Session Initiation Protocol) [32] is widely used in VoIP networks. SIP messages are divided into requests and responses. Keywords identify the type of a request such as `INVITE, OPTIONS, REGISTER, NOTIFY` while 3 digits numbers are used for responses (STATUS). The first digit, between 1 and 6, indicates the class (1xx: provisional, 2xx: success, 3xx: redirection, 4xx: client error, 5xx: server error, 6xx: global failure). Therefore, SIP Messages can be classified hierarchically according to the SIP Response Codes and SIP Request

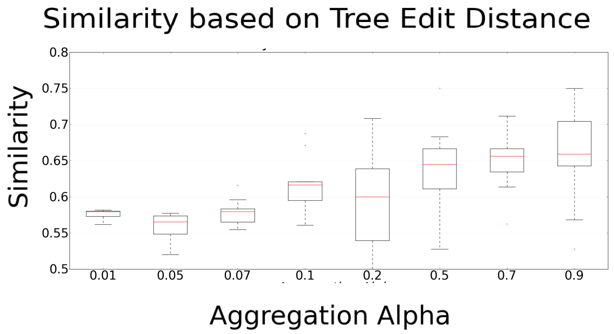


Figure 8: Similarity box plots for trees generated using Netflow samples compared to trees generated after a the same Netflow capture reversed for a 5 minutes window

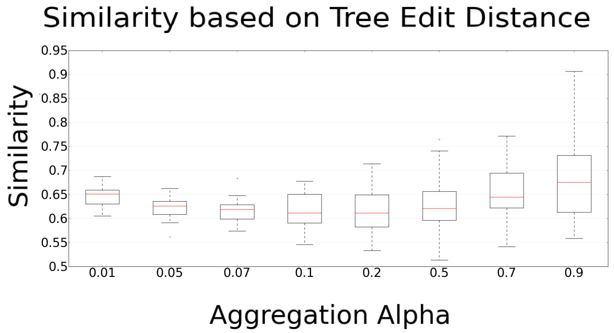


Figure 9: Similarity box plots for trees generated using Netflow samples compared to trees generated after the same randomized flows for a 5 minutes window

Methods described in [32] similar to a taxonomy like in Figure 5 where the first level of division from the root node will be the request and responses types.

This dataset was collected on a local testbed and has the following characteristics:

- 33952 SIP Messages, 17104 Requests, 16848 Responses
- 77 IP Addresses
- 38 Source IP Addresses, 266 source users
- 35 Destinations IP Addresses, 317 destination users.

7.2 Metrics

Trees are among the most common and well-studied combinatorial structures in computer science. The aggregated tree construction is sensitive to the data order. The goal is to measure the degree of similarity for trees built from the same data but in a different order. The Levenshtein distance [22, 7] is usually refereed as edit distance between two strings. It is defined as the minimum amount of operations to transform one string into the other (deletion, insertion and relabeling or substitution). As an example the strings ”sos” and ”sbs” have a distance of 1 by performing one substitution of ”o” into ”b”. This notion of distance was initially defined for strings where every operation has a single cost of 1. As a node is a more complex structure, thus deletions and insertions are always associated to a cost of 1. However substitutions in multidimensional nodes are decomposed into each dimension. According to the formal definition of a M multidimensional tree, a node was defined as $n_i \ = <\{f_{i_1}\dots f_{i_m}\}, vol_i>$. A distance function per each dimension can calculate the cost of an operation as:

$$\frac{\sum_{k=1}^{M} Distance_k(f_{1_k}, f_{2_k})}{m}$$

Thus, the distance function has to be defined for each dimension. In our case, we define it based on the longest common prefix between two IP addresses or on the longest common path in the taxonomy tree for ports.

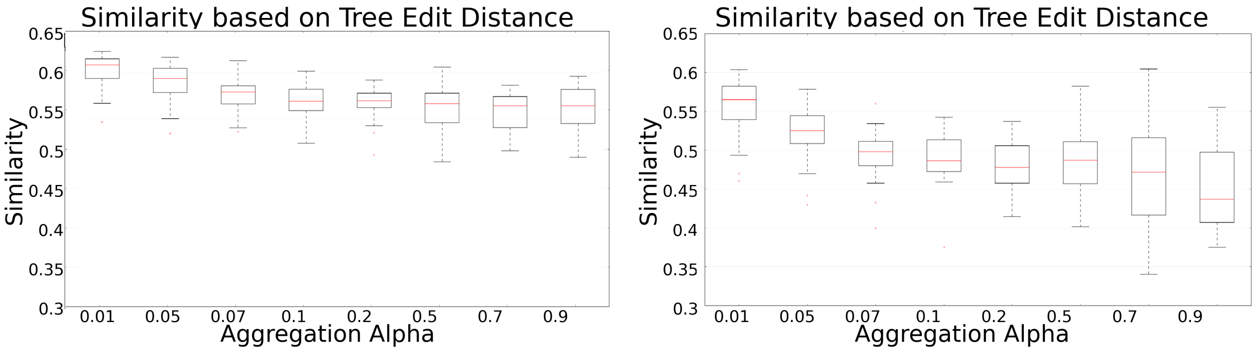
Tree distance can be computed by calculating a distance matrix as the result of the pair wise comparison between the nodes of the trees to compare. Then we calculate the average over the minimal distance of every row and column.

7.3 Order of data impact

In order to evaluate the impact of data order we used the similarity metrics described in Section 7.2. The focus is set on analyzing the distortion of the tree shape after altering the data order.

The evaluation was performed on 3 dimensions: source IPv4 address, destination IPv4 address and TCP port. For strengthen the evaluation, experiments are executed 1000 times with different samples and percentiles (25th, 50th, 75th) as well as minimal and maximal values are then calculated and represented in a box plot graph like Figure 8.

- Normal Sample vs Reversed Sample: This test consists in comparing trees generated from a 5 minutes sample ($\eta = 5min$) of the Netflow dataset and the same data, *i.e.* Netflow records, in the reverse order. In Figure 8, the similarity grows with aggregation ratio α except for small values. This behavior is logic as increasing the aggregating will keep track of global phenomenas which are usually present in the whole sample, `i.e.` not at a specific time unlike local events, and so less sensitive to the data order.



(a) 24 hours offset

(b) 36 hours offset

Figure 10: Similarity - trees generated using Netflow samples compared to trees generated after a Netflow capture from a offset using a 5 minutes window

- Normal Sample vs Random Sample: this test is similar but data are not reversed but randomized. The same observations can be made in Figure 9 even if the impact of α is less visible.

7.4 Scenarios

It is important to note that the similarity is a number between 0 and 1 with 1 when the trees are identical. We can observe from our experiments that the data order has an impact on the aggregation process itself since the value never reaches 1. However, it is important to verify that the similarity value is lower when data differs. Therefore, we can assumed that in most of cases, a similarity higher than 0.6 may reflect similar data but which may have been process in a different order.

7.4.1 Netflow

In order to evaluate the aggregation accuracy, several tests over the Netflow dataset were conducted. The evaluation consists in comparing similar and different traffic flows by observing the similarity variation. From the dataset, 50 time windows ($\eta = 5min$) are randomly picked up from working days between 01/30/2010 and 02/24/2010. Once the tree is built, it is compared with the tree associated to data with a 24 hours or 36 hours offset. Activity at the same time between two working days (24h offset) is usually considered as similar and so should exhibit a higher similarity than activity at a different times between two days (36h offset). Figure 10 shows that the similarity with a 36h offset decreases higher than with the 24h offset, leading to an easier differentiation. Even if the average similarity tends to be close between figures 10(a) and 10(b), the box plots clearly highlight a lower stability with a 36h offset which leads to the same conclusion. With respect to experiments on the data order (section 7.3), values are mainly lower than 0.6 for the 24h offset. Therefore, impact of the data order is lower than the impact of activity variation at the same time between two consecutive days. From this point of view, our approach is thus viable.

To evaluate the aggregation helpfulness in an malicious environment, the following experiment has been conducted:

- injection of DDoS attack directed against web servers running on TCP port 80 with a repeated sequence (3 packets and 128 bytes) sent by burst of 10 packets every 60-120 milliseconds. The attack was injected in two periods of 2 minutes each.
- 60 trees were generated, $T_0 \ldots T_{59}$, to summarize network activity ($\eta = 25s, \alpha = 0.05$) by monitoring both the source and destination IP addresses meanwhile
- we compute the edit distance as defined in Section 7.2 between two trees to figure out the attacks: $z[n] = EditDistance(T_n, T_{n+1})$

In figure 11, the two attack occurrences are clearly distinguishable by observing peaks due the high variation of the edit distance when the attacks start and end. Although more advance techniques, as for example machine learning based methods, could be leveraged and evaluated on other kinds of attacks, such a complete evaluation is not in the scope of the paper.

7.4.2 SIP Messages

In order to evaluate Aggregation accuracy for SIP messages, we tested the aggregation mechanism to detect high volume of error messages which is helpful for figuring out anomalies. As the testbed was locally made, we were able to generate anomalies (bypassing the registration). From our traces, we extracted one tree before

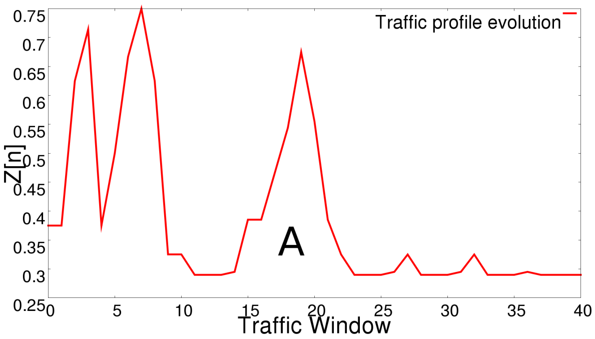


Figure 11: Distance Function Evaluation for aggregated trees generated from Flow capture of a TCP Flood attack

(figure 12) and after (figure 13) the problem occurs. In particular, a node containing an error (*ProxyAuthenticationRequired.ClientError*) appears. A simple counter per message type would also be able to detect it. However, the tree informs that a relatively big portion of traffic (7.32%) is concerned by this error on a single subnet, 192.168.1.0/29. It is helpful for identifying meanwhile the sources from where the errors come and so for recovering afterwards. Although this is a example to highlight the benefits of our approach, more advanced technique for analysis are possible. Retrieving automatically new nodes or only nodes representing errors could help an administrator.

From a general point of view, the trees produced by *MAM* can be manually or automatically analyzed. For manual analysis, this eases the understanding as the information is compacted and is sorted from top to bottom by the impact (accumulated volume of activity). Therefore, if an observation looks abnormal at a high level in the tree, finding the root cause requires to follow the parent-child relation. When reading the tree, the user can easily follow the interesting branches like those related to erroneous responses. Future visualizations techniques could propose a dynamic navigation within the tree by providing information on demand. Experiments where conducted using ($\eta = 120s - 360s, \alpha = 0.05$)

7.4.3 DNS names associated to netflow captures

For DNS names, the dimension *Domain*, representing a FQDN, takes in account top level domains and sub domains separated by dots. Each dot is another split between a parent and its child in the hierarchical tree. Figure 14 illustrates the approach using the dataset described in Section 7.1.2 by limiting our study to source IP addresses associated to the french top level domain (*.fr* domains). We also consider the domains of destination IP addresses. For example we can notice that most of traffic originating from *.fr* to *.com* domains targets *deezer*.

Assuming a fast-flux botnet [30], the DNS names and IP addresses should be monitored while considering the inverse of the DNS Time-To-Live should be considered as activity volumes. Using a short TTL and a large set of IP addresses (usually compromised machines), the attacker changes frequently the IP address associated to the domain hosting malware. By using the inverse of TTL, such behavior will appear using *MAM* by highlighting the problematic domain and IP addresses or subnets. Experiments were conducted summarizing network activity using parameters ($\eta = 0s - 300s, \alpha = 0.05 - 0.1$)

7.4.4 Geographical coordinates associated to Netflow captures

Evaluating results of geographical coordinates aggregation requires a more sophisticated display structure. Representing areas (ranges of geographical coordinates) in a map shows more information rather than displaying a tree with coordinates. Hence, the Google Maps API[1] was used to show a real map. Only flows having the origin and destination in a fixed area (France or Europe) are extracted. The color opacity reflects the aggregated traffic load by area. Figure 15(a) highlights that Luxembourg is the center of the most inner square which is natural as the dataset we used is provided by a Luxembourg operator, and so most of flows are from and to hosts in this country. As an other illustrative example, a traffic diagram of France is presented in figure 15(b) which logically highlights the concentration of network traffic around Paris. These experiments show, that without specifically guiding the aggregation by pre-defining geographical zones or using cities, our method is able to figure out automatically important parts, well known areas (around Paris) as well as others (for example in South of France in figure 15(b)).

As another illustrative case, a content provider may track his user location for optimizing the placement of his servers or to provide locally profiled content.

7.5 Performance

Several benchmarks have been done in a 8 core Intel(R) Xeon(R) CPU E5345@ 2.33GHz. They are based on the running times regarding the number of nodes in a tree. The Netflow dataset was used by considering source and destination addresses as well as destination ports. As introduced in Section 6, the worst case computational complexity of the LRU strategy is $n \times log(n)$. In figure 16(b), the empirical results are quite lower. This behavior can be explained because the worst case scenario is a pathological case that is not usually common found in

[1] https://developers.google.com/maps/

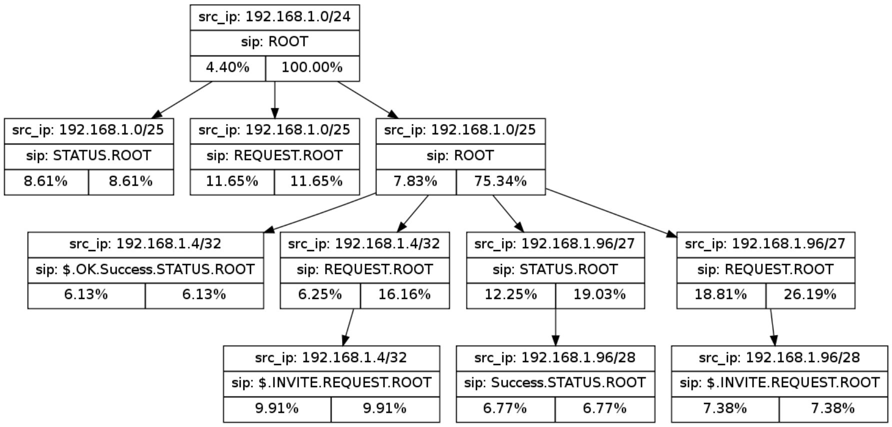


Figure 12: Aggregation for SIP conversations on networks 192.168.1.0/24, dimensions Message Type and source IP Address - no error (Partial view)

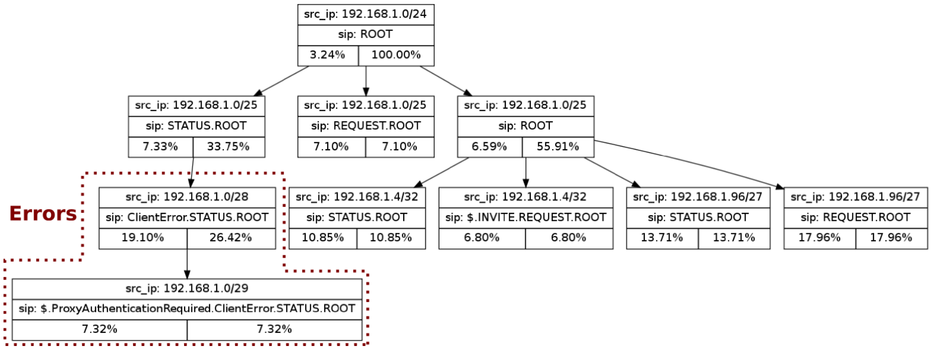


Figure 13: aggregation for SIP conversations on networks 192.168.1.0/24, dimensions Message Type and source IP Address - with errors (Partial view)

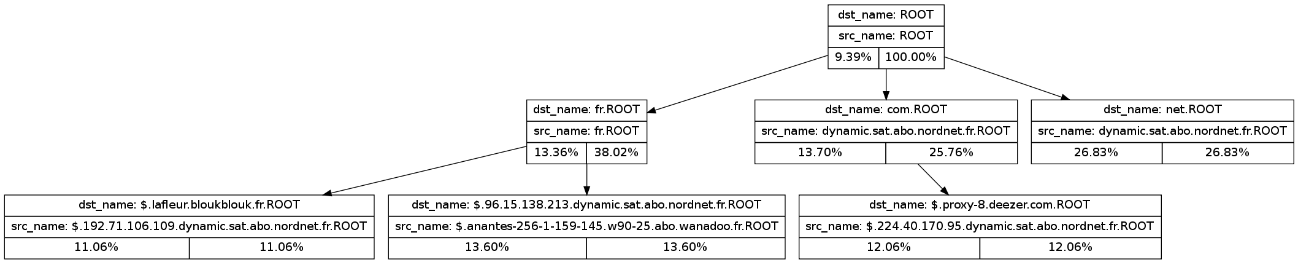
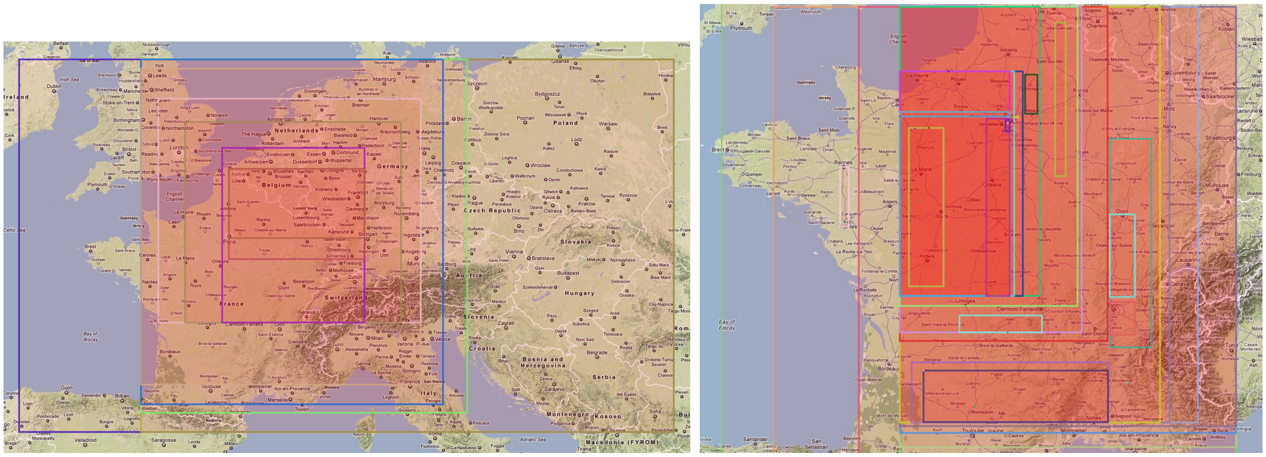

dst_name: ROOT
src_name: ROOT
9.39%
100.00%
dst_name: fr.ROOT
src_name: fr.ROOT
13.36%
38.02%
dst_name: com.ROOT
src_name: dynamic.sat.abo.nordnet.fr.ROOT
13.70%
25.76%
dst_name: net.ROOT
src_name: dynamic.sat.abo.nordnet.fr.ROOT
26.83%
26.83%
dst_name: $.lafleur.bloukblouk.fr.ROOT
src_name: $.192.71.106.109.dynamic.sat.abo.nordnet.fr.ROOT
11.06%
11.06%
dst_name: $.96.15.138.213.dynamic.sat.abo.nordnet.fr.ROOT
src_name: $.anantes-256-1-159-145.w90-25.abo.wanadoo.fr.ROOT
13.60%
13.60%
dst_name: $.proxy-8.deezer.com.ROOT
src_name: $.224.40.170.95.dynamic.sat.abo.nordnet.fr.ROOT
12.06%
12.06%

Figure 14: DNS names aggregation using as dimensions PTR records for source and destination (Partial view)



(a) Internal Traffic aggregation for Europe

(b) Internal Traffic aggregation for France

Figure 15: Aggregation example for Geographical coordinates associated to netflow captures aggregation for traffic within Europe.

the captures. Furthermore, this is coherent with the average computational complexity ($log^2(n)$) introduced in Section 6.

Figure 16(a) highlights similar results for the simple aggregation at the end of time windows. Therefore, algorithms are equivalent from a computational power side. Applying the LRU strategy does not entail any notable overhead. However, from a memory point of view, the LRU strategy saves a lot of memory space since the tree size is bounded by MAX_NODES.

The average size of the trees before and after the aggregation is given in table 1 when no LRU strategy is used during our previous experiments. First, using LRU strategy strongly limits the memory usage since the tree size before aggregation is quite higher 1000 (default value for MAX_NODES when LRU is applied) except for SIP since the dataset was generated on a local testbed. Therefore, in real scenarios, memory usage is highly reduced during the tree construction, which improves the scalability. α set to 5% provides good results in previous experiments while the number of nodes after the aggregation at the end of a time window is drastically reduced. The tree size never exceeds 90 nodes. Therefore, scalability is again increased from a storage point of view as well as for doing analysis (manual or automated) on the final built trees.

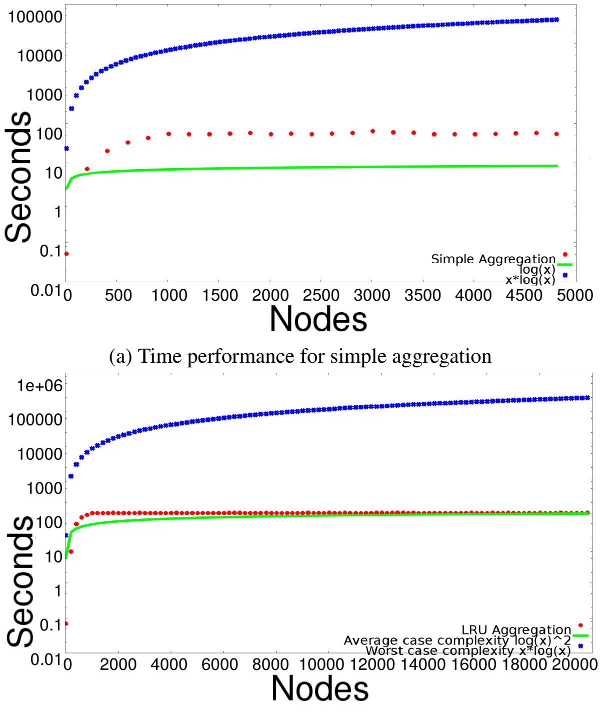
Support real-time flow aggregation is validated by associating Table 1 and figure 16(b), 3288 netflow records (average number of flows in a 5 minutes window) are always aggregated in less than 100 seconds, this allows to aggregate flows in real-time. Memory consumption can be computed as the sum of the dimensions data type size and the tree final size. For GPS coordinates every node holds 32 bits for 2 integers. Regarding the data structure 8 additional bytes and a list of pointers (4 bytes in 32bits) to the children are required. With an average tree size of 14664 nodes tree this results in $14664 \times (4+8+8) = 286.4MB$.

	Average tree size before aggregation	Average tree size after aggregation
Netflow	3288	90
DNS	8600	53
SIP	1077	18
Geographical	14664	45

Table 1: Tree size reduction using aggregation (average on all time windows) with $\alpha = 0.05$

8 Conclusion

In this paper, a multidimensional aggregation is introduced which is able to handle the multiple dimensions without having any predefined order. The aggregation increases the scalability by compressing data and by creating summarized outputs which are smaller and easier to analyse. Moreover, the aggregation is guided by the nature of the events to monitor, *i.e.* a configurable activity volume. This leads to split the space into partition of different sizes. Hence, distributed coordinated behaviors can be observed unlike traditional approaches relying on a regular space division which does not necessarily reflect the current distribution on the network activity. Based on formal definitions, this paper highlights the use of common dimensions related to network administra-



(a) Time performance for simple aggregation

(b) Time performance for LRU aggregation

Figure 16: Time performance for different strategies

tion like IP addresses, applications, DNS, but could be extended to any other kinds of dimensions such as polar coordinates by extending the feature class and implementing the requested methods in the API. We provide an open source tool[2] to export png file or Google Maps and provide ready to use implementations of IP addresses, applications, DNS and GPS features.

Experiments focus on sample scenarios where the aggregation reveals important facts or changes in the network. Theoretical as well as practical complexities were studied, in particular to prove the benefits of using a LRU strategy for improving the scalability of our approach. Furthermore, the data order problem due to such a process was evaluated and the results highlight a negligible impact. Similarly to our previous work on single dimension aggregation [34], aggregated trees are also good candidates for data mining algorithms and not exclusively dedicated to being processed by a human expert. Therefore, our next work will focus on this topic and on the definition of other dimensions like IPv6 addresses.

Acknowledgement: This work is partly funded by OUTSMART, a European FP7 project under the Future Internet Private Public Partnership programme. It is also supported by MOVE, a CORE project funded by FNR in Luxembourg.

References

[1] Antonakakis, M., Perdisci, R., Lee, W., Vasiloglou, II, N., and Dagon, D. Detecting malware domains at the upper dns hierarchy. In *Proceedings of the 20th USENIX conference on Security* (Berkeley, CA, USA, 2011), SEC'11, USENIX Association, pp. 27–27.

[2] Atkinson, M., Sack, J., Santoro, N., and Strothotte, T. Min-max heaps and generalized priority queues. *Communications of the ACM 29*, 10 (1986), 996–1000.

[3] Ball, R., Fink, G., and North, C. Home-centric visualization of network traffic for security administration. In *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security* (2004), ACM, pp. 55–64.

[4] Barr, D. RFC 1912: Common DNS operational and configuration errors.

[5] Bentley, J. Multidimensional binary search trees used for associative searching. *Communications of the ACM 18*, 9 (1975), 509–517.

[6] Bilge, L., Kirda, E., Kruegel, C., and Balduzzi, M. Exposure: Finding malicious domains using passive dns analysis. In *Proceedings of the Network and Distributed System Security Symposium (NDSS 2011)* (2011), The Internet Society.

[7] Bille, P. A survey on tree edit distance and related problems. *Theoretical computer science 337*, 1 (2005), 217–239.

[8] Cho, K., Kaizaki, R., and Kato, A. Aguri: An aggregation-based traffic profiler. In *Quality of Future Internet Services* (2001), Springer, pp. 222–242.

[9] Claise, B., Sadasivan, G., Valluri, V., and Djernaes, M. Rfc 3954: Cisco systems netflow services export version 9. Tech. rep., 2004.

[10] Drago, I., Sadre, R., and Pras, A. Report of the third workshop on the usage of netflow/ipfix in network management. *J. Netw. Syst. Manage. 19*, 4 (2011), 529–535.

[11] Estan, C., Keys, K., Moore, D., and Varghese, G. Building a better netflow. *ACM SIGCOMM Computer Communication Review 34*, 4 (2004), 245–256.

[12] Fenwick, P. A new data structure for cumulative frequency tables. *Software: Practice and Experience 24*, 3 (1994), 327–336.

[2] http://lorre.uni.lu/~jerome/files/mam.tar.gz

[13] FINKEL, R., AND BENTLEY, J. Quad trees a data structure for retrieval on composite keys. *Acta informatica 4*, 1 (1974), 1–9.

[14] FRANÇOIS, J., WANG, S., BRONZI, W., STATE, R., AND ENGEL, T. BotCloud: Detecting Botnets Using MapReduce. In *Workshop on Information Forensics and Security (WIFS)* (Foz do Iguaçu, Brazil, December 2011), IEEE, Ed.

[15] FRANÇOIS, J., WANG, S., STATE, R., AND ENGEL, T. BotTrack: Tracking Botnets using NetFlow and PageRank. In *IFIP/TC6 NETWORKING 2011* (Valencia, Spain, May 2011), Springer, Ed.

[16] FULLER, V., LI, T., YU, J., AND VARADHAN, K. Rfc1519: Classless inter-domain routing (cidr): an address assignment and aggregation strategy.

[17] GIURA, P., AND MEMON, N. Netstore: An efficient storage infrastructure for network forensics and monitoring. In *Recent Advances in Intrusion Detection* (2010), Springer, pp. 277–296.

[18] GRAY, J., CHAUDHURI, S., BOSWORTH, A., LAYMAN, A., REICHART, D., VENKATRAO, M., PELLOW, F., AND PIRAHESH, H. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Min. Knowl. Discov. 1*, 1 (1997), 29–53.

[19] HU, Y., CHIU, D.-M., AND LUI, J. C. S. Entropy based adaptive flow aggregation. *IEEE/ACM Trans. Netw. 17*, 3 (2009), 698–711.

[20] KAIZAKI, R., CHO, K., AND NAKAMURA, O. Detection of denial of service attacks using aguri. In *International Conference Telecommunications, Beijing China* (2002).

[21] KRIZAK, P. Log analysis and event correlation using variable temporal event correlator (vtec). In *Proceedings of the 24th international conference on Large installation system administration* (2010), LISA'10, USENIX Association.

[22] LEVENSHTEIN, V. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady* (1966), vol. 10, pp. 707–710.

[23] LIAO, Q., BLAICH, A., STRIEGEL, A., AND THAIN, D. Enavis: enterprise network activities visualization. In *Proceedings of the 22nd conference on Large installation system administration conference* (2008), LISA'08, USENIX Association, pp. 59–74.

[24] LOTTOR, M. RFC 1033: Domain administrators operations guide.

[25] MAXMIND, L. www.maxmind.com, 2007.

[26] MOCKAPETRIS, P. Rfc 1035: Domain names - implementation and specification.

[27] MOREIRA MOURA, G., SADRE, R., SPEROTTO, A., AND PRAS, A. Internet bad neighborhoods aggregation.

[28] OBERHEIDE, J., GOFF, M., AND KARIR, M. Flamingo: Visualizing internet traffic. In *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP* (2006), IEEE, pp. 150–161.

[29] PAREDES-OLIVA, I., BARLET-ROS, P., AND SOLÉ-PARETA, J. Portscan detection with sampled netflow. *Traffic Monitoring and Analysis* (2009), 26–33.

[30] PERDISCI, R., CORONA, I., DAGON, D., AND LEE, W. Detecting malicious flux service networks through passive analysis of recursive dns traces. In *Annual Computer Security Applications Conference - ACSAC* (2009), IEEE Computer Society.

[31] PLONKA, D., AND BARFORD, P. Context-aware clustering of dns query traffic. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement* (2008), IMC '08, ACM.

[32] ROSENBERG, J., SCHULZRINNE, H., CAMARILLO, G., JOHNSTON, A., PETERSON, J., SPARKS, R., HANDLEY, M., AND SCHOOLER, E. Rfc 3261 sip: Session initiation protocol.

[33] TOLEDO, J., ADRIGHEM, V., GHETTA, R., MANN, E., AND PETERS, F. Etherape: a graphical network monitor, accessed on 05/14/2012.

[34] WAGNER, C., FRANCOIS, J., ENGEL, T., ET AL. Danak: Finding the odd! In *Network and System Security (NSS), 2011 5th International Conference on* (2011), IEEE, pp. 161–168.

[35] WEIGERT, S., HILTUNEN, M., AND FETZER, C. Community-based analysis of netflow for early detection of security incidents. In *Proceedings of the 25th Large Installation System Administration Conference (LISA'11)* (2011), USENIX Association.

[36] YURCIK, W. Visualizing netflows for security at line speed: the sift tool suite. In *Proceedings of the 19th conference on Large Installation System Administration Conference - Volume 19* (2005), LISA '05, USENIX Association.