Securing Public Cloud Networks with Efficient Role-based Micro-Segmentation

Sathiya Kumaran Mani and Kevin Hsieh, *Microsoft;* Santiago Segarra, *Rice University;* Ranveer Chandra, *Microsoft;* Yajie Zhou, *University of Maryland;* Srikanth Kandula, *Microsoft*

https://www.usenix.org/conference/nsdi25/presentation/mani

This paper is included in the Proceedings of the 22nd USENIX Symposium on Networked Systems Design and Implementation.

April 28–30, 2025 • Philadelphia, PA, USA

978-1-939133-46-5

Open access to the Proceedings of the 22nd USENIX Symposium on Networked Systems Design and Implementation is sponsored by

Securing Public Cloud Networks with Efficient Role-based Micro-Segmentation

Sathiya Kumaran Mani*[1] Kevin Hsieh*[1] Santiago Segarra[2]
Ranveer Chandra[1] Yajie Zhou[3] Srikanth Kandula[1]
[1]*Microsoft* [2]*Rice University* [3]*University of Maryland*

Abstract

Securing network traffic within data centers is a critical and daunting challenge due to the increasing complexity and scale of modern public clouds. Micro-segmentation offers a promising solution by implementing fine-grained, workload-specific network security policies to mitigate potential attacks. However, the dynamic nature and large scale of deployments present significant obstacles in crafting precise security policies, limiting the practicality of this approach. To address these challenges, we introduce a novel system that efficiently processes vast volumes of network flow logs and effectively infers the roles of network endpoints. Our method integrates domain knowledge and communication patterns in a principled manner, facilitating the creation of micro-segmentation policies at a large scale. Evaluations with real-world deployment demonstrate that our solution significantly surpasses existing algorithms in role inference accuracy. We implement our solution as an end-to-end system and demonstrate that it is up to 21.5× more cost-efficient than Apache Flink, a widely used open-source stream processing system.

1 Introduction

The growth and criticality of public clouds are accompanied by persistent security threats. Prominent security incidents, such as those involving SolarWinds [2], Midnight Blizzard [71], and Snowflake [68], reveal a spectrum of attack vectors from compromised credentials to exploited software providers and trust chains. These incidents often lead to massive leaks of sensitive data, significant economic losses, and deterioration of customer trust [1, 86, 87]. Consequently, there is a pressing need for the development of enhanced security technologies to protect the integrity of data centers.

One promising solution that has emerged is *micro-segmentation* [89], a technique grounded in the zero-trust security model [63, 78]. Recognizing that preventing every cyberattack is a nearly unattainable task, micro-segmentation operates under the assumption that every endpoint can be compromised. It therefore aims to contain the impact of potential breaches by creating isolated μsegments within the network. Under this model, resources in different μsegments can only communicate with each other if explicitly permitted by a set of fine-grained, workload-specific policies. This restrictive approach significantly limits the potential damage of a breach. Several vendors have already introduced micro-segmentation offerings [28, 41, 55, 88]. The market for micro-segmentation is worth several billion dollars in annual revenue and boasts a robust growth rate of over +20% year-over-year [25].

However, two significant challenges arise when implementing micro-segmentation on a large scale: precise security policies and cost-effective monitoring. First, existing solutions [28, 41, 55, 88] rely on human administrators to assign a μsegment label to *every* endpoint. This μsegment creation process necessitates a deep understanding of the workloads and continuous updates to keep up with changes in software or environment. This manual approach does not scale for large systems that leverage millions of resources [72] and have large teams owning different parts of cloud software [19, 33, 45]. Mislabeling or outdated labels can result in legitimate access being denied or unnecessary exposure to potential attacks. Second, a micro-segmentation solution necessitates the capability to observe and monitor network behavior. This involves the near real-time gathering and analysis of network telemetry, a process that can inflate costs significantly. Indeed, leading solutions [28, 55] add a substantial *extra* cost to the infrastructure, such as virtual machines (VMs) and containers, ranging from 16% to as high as 71%. This financial burden presents a considerable obstacle to their extensive adoption.

Objectives and Techniques. We introduce ZTS (Zero Trust Segmentation), a novel micro-segmentation system designed for large-scale public cloud networks. The objective of ZTS is to facilitate the creation of precise, scalable security policies, while also offering a cost-effective solution for gaining visibility into network behavior and conducting efficient monitoring. ZTS consists of two core components:

(1) Deployment-Specialized Auto-Segmentation: To gen-

* These authors contributed equally.

erate μsegments on a large scale, we introduce a novel auto-segmentation algorithm predicated on the inference of network endpoints' roles. Our algorithm is grounded in the observation that within a large deployment, a multitude of network nodes often fulfill identical roles, such as replicas, microservices, lambdas, etc. These roles can be designated as the μsegments, as nodes performing the same function can adhere to the same network security policy. A key insight underpinning our role inference algorithm is that the definition of role varies across different functionalities and deployments. As such, our algorithm judiciously combines domain-specific knowledge, communication patterns, and deployment-specific information. This approach then generates an autoencoder that is uniquely tailored to each deployment, thereby enhancing its accuracy in inferring roles and μsegments.

(2) Cost-Effective Communication Graph Generator: To minimize the costs of observing and monitoring network behavior, we carefully design a unique system architecture that efficiently generates a complete communication graph from a large volume of network flow logs at a very low cost. Our architecture leverages compute endpoints (e.g., VMs) to process and batch a substantial number of network flow logs into a database endpoint (e.g., SQL servers) and employs advanced query optimization techniques to construct communication graphs. This design enables our system to accommodate large-scale deployments with a minimal amount of compute, memory, and network resources.

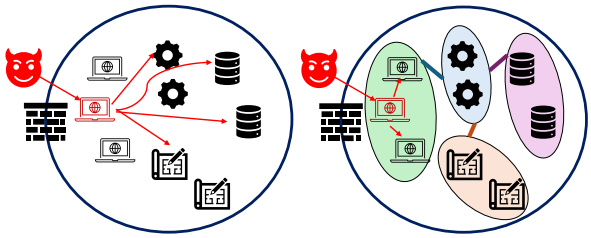
Implementation and Evaluation. We build ZTS as a complete micro-segmentation system that provides continuous communication graph generation, advanced role inference for auto-segmentation, and security policy authoring and enforcement. To evaluate the accuracy of our auto-segmentation algorithm, we curate ground truth data across 11 first-party and third-party deployments, encompassing communication graphs with tens of thousands of nodes and millions of edges. Our evaluation demonstrates that our role inference algorithm significantly outperforms state-of-the-art algorithms [56, 59, 72, 93], achieving an average Adjusted Rand Index (ARI) [53][1] improvement of 0.37.

To evaluate cost-effectiveness in network observation and monitoring, we implement an alternative communication graph generator using Apache Flink [38], a popular open-source stream processing system. Our evaluation across different deployments shows that ZTS is 7.5× faster than Apache Flink at only 35% of the cost, making it 21.5× more cost-efficient. Deploying ZTS only introduces an extra 0.5% of the cost of the VMs, which is significantly cheaper than existing micro-segmentation solutions.

Contributions. We make the following contributions:

- We introduce a novel algorithm that infers network endpoints' roles by incorporating domain-specific knowledge and deployment-specific information in a principled manner. This approach significantly enhances the accuracy and scalability of micro-segmentation.
- We design a cost-effective system architecture for generating communication graphs from network flow logs, which reduces resource usage and operational costs in implementing micro-segmentation at a large scale.
- We build an end-to-end micro-segmentation system and show significant improvements in role inference accuracy and cost-efficiency compared to existing solutions, validated using large-scale real-world deployments.

[1] ARI is a widely used metric for measuring clustering quality, adjusted for chance. It ranges from -0.5 for highly discordant clusterings to 1.0 for identical clusterings.



(a) Without Micro-segmentation (b) With Micro-segmentation

Figure 1: An example of how micro-segmentation prevents lateral movement by attackers.

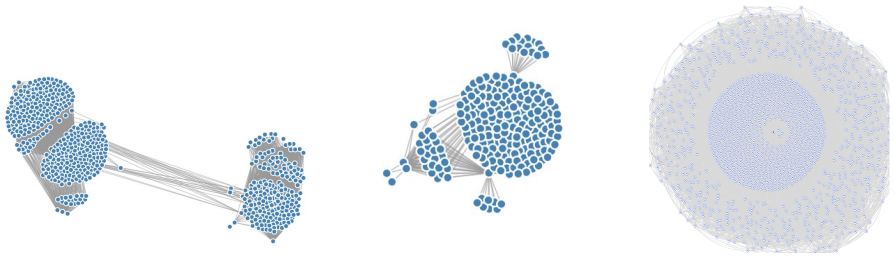
2 Background and Motivation

We begin by providing background on micro-segmentation, followed by a detailed discussion of the major challenges in realizing this concept in practice.

2.1 Micro-Segmentation

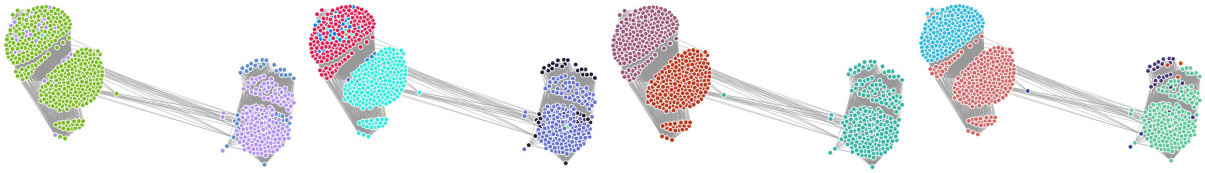
Micro-segmentation [89] is a security technique that involves dividing a network into smaller, more manageable, so-called, *μsegments*, each with its own set of security controls. The concept emerged as a response to the increasing complexity and interconnectivity of modern IT environments, where traditional perimeter-based defenses proved insufficient. As illustrated in Figure 1(a), a compromised network node permits an attacker to propagate laterally across other nodes without limitations, potentially resulting in significant data breaches and system damage. However, by implementing micro-segmentation, as depicted in Figure 1(b), each μsegment effectively minimizes the attack surface. This is achieved by confining communications to predefined pathways, thereby enhancing the containment of breaches. Today, micro-segmentation is considered a core component of zero-trust security architectures [63, 78].

Deploying a micro-segmentation solution involves two key phases to ensure effective implementation and enhanced security. The first phase is *μsegment definition and policy authoring*, where administrators conduct a comprehensive assess-



(a) K8s PaaS IP-graph (b) Portal IP-graph (c) KQuery IP-graph

Figure 2: Unsegmented IP-graphs for the three real-world deployments



(a) Simrank segmentation (b) Simrank++ segmentation (c) Conn.-weight. modularity (d) Byte-weighted modularity

Figure 3: Applying other segmentation strategies, inspired by prior work, on K8s PaaS's IP-graph

ment of the current network architecture and communication patterns. This is often facilitated by generating a comprehensive communication graph using network flow logs from the workloads. Following this assessment, administrators define the relationships between assets and μsegments, authoring security policies that specify which μsegments can communicate with each other and under what conditions. A pair of resources can communicate with each other only if explicitly allowed by the policies; i.e., the default will be to deny [34, 39].

The second phase is *implementation and monitoring*, where these security policies are enforced using a micro-segmentation platform, which may involve configuring virtual firewalls and security controls at the hypervisor or network level. Once implemented, the system should be continuously monitored to ensure compliance with the defined policies. Additionally, μsegments and policies need to be regularly updated to accommodate changes in configurations, deployments, and workloads.

2.2 Challenges in Authoring Security Policies

While creating micro-segmentation may appear straightforward in simple examples, such as in Figure 1, in practice, creating and maintaining μsegments can be a daunting challenge for administrators. To illustrate this issue, we examine three real-world deployments in Table 1.

Figure 2 depicts the unsegmented IP-level communication graphs for each deployment. As these graphs show, the number of nodes and communication patterns can quickly become complex. Our interviews with developers reveal that even they often do not fully understand these graphs, as communication can be initiated by various layers in a complex software stack. The current state-of-the-art requires a human

	#IPs mon.	Graph Size: #nodes (#edges)		#Records
		IP Graph	IP-port Graph	/minute
Portal	4	4K (5K)	13K (13K)	332
K8s PaaS	390	541 (12K)	1.3M (3M)	68K
KQuery	1400	6K (1.3M)	12M (79M)	2.3M

Table 1: Cloud deployments and some aspects of their communication graphs that we built and analyzed in this paper. The source telemetry is flow-level summaries every minute with schema as in Table 3.

administrator to manually tag each node with a μsegment label so that the system can suggest security policies. However, this approach is neither scalable nor error-free. Our investigation of real-world deployments (detailed in Table 5) reveals that only 12% to 23% of network nodes possess useful information—such as machine functions, tags, or function-based names—that could potentially serve as μsegment labels. However, the availability and consistency of such information exhibit considerable variation across different deployment scenarios. Furthermore, administrators need to stay updated on all role changes—such as when workloads migrate, scale up, or scale down [13], or when a software update alters communication behavior—which occurs frequently in large-scale deployments.

Study of Existing Role-Inference Algorithms. Could an algorithm assist with μsegment labeling? Fundamentally, there are far fewer roles than resources in a cloud setting because, for redundancy and scalability, it is common for multiple resources to share the same role. Intuitively, nodes that play the same role share communication patterns and may be inferred based on the similarity of neighbors a resource communicates with and the nature of the conversation. This is similar to the role inference problem in the graph mining literature [51, 58]. Figure 3 presents the outcomes using some established tech-

Core #	VM	Illumio	Illumio vs. VM	Akamai	Akamai vs. VM
2	$549		65%		71%
4	$1,100	$354	32%	$390	35%
8	$2,194		16%		18%

Table 2: Pricing comparison between Azure general-purpose VMs [7], Illumio Core [14], and Guardicore Segmentation [3]. All pricing represents annual costs, normalized to a single VM and shown in US Dollars. The pricing for Illumio Core and Guardicore Segmentation was from Azure Marketplace.

niques on one deployment. There are clear discrepancies among them, and it is unclear which one is more correct.

To deepen our understanding, we conduct a study utilizing popular, state-of-the-art role inference algorithms, including (1) Jaccard: similarity of neighboring nodes [59], (2) SimRank [56]: graph structural similarity, (3) GAS [93]: structural embeddings with graph neural networks, and (4) CloudCluster [72]: grouping nodes based on the adjacency matrix. We curate ground truth data for 11 real-world deployments (details in Section 6.2). The full results are summarized in Table 6 (Section 6.2). Our findings indicate that, although certain algorithms demonstrate good performance in specific deployments, none consistently achieve high performance across a diverse range of deployments. Additionally, the performance of all algorithms diminishes as the scale of deployments increases. Overall, existing algorithms tend to produce role inference results that significantly deviate from the ground truth, with none achieving an average Adjusted Rand Index (ARI) greater than 0.5.

Our analysis reveals that effective and consistent role inference for micro-segmentation remains a major challenge in practice, attributable in part to several factors: (1) The phenomenon driving role similarity in cloud communication graphs—namely, identical code executing across multiple resources for redundancy and scalability—does not stem from human interactions, unlike in social and product recommendation graphs [51]. (2) Structural resemblance within a communication graph does not guarantee role similarity; thus, an algorithm capable of harnessing additional communication attributes is necessary to differentiate such instances. (3) Given that the optimal role inference algorithm may vary among deployments or evolve over time, a dynamic, learning-based algorithm that adjusts according to feedback could potentially surpass all static approaches.

2.3 Cost of Micro-Segmentation Solutions

Micro-segmentation is an expanding market with solutions from multiple vendors [28, 41, 55, 88]. We examine the costs associated with two leading vendors specializing in micro-segmentation platforms [28, 55], comparing these expenses to the infrastructure costs of VMs with varying core counts.

As Table 2 shows, the pricing of both solutions is comparable and incurs significant extra costs, constituting 16% to 71% of the total VM costs. Although licensing fees may not directly equate to computational costs, this analysis highlights the substantial financial burden on organizations seeking to implement micro-segmentation.

Our analysis finds that the primary cost of micro-segmentation lies in the collection and analysis of network telemetry to achieve near real-time network visibility. This involves the continuous processing of extensive network flow logs to generate a comprehensive view. Initial attempts with off-the-shelf solutions, such as Apache Flink [38] and Apache Spark [92], proved to be cost-prohibitive, adding over 10% to the overall VM expenses.

Summary. The lack of an effective role inference algorithm for network endpoints, coupled with the high costs of network behavior monitoring, are significant barriers to achieving the full potential of micro-segmentation. These challenges form the core of our design requirements for ZTS.

3 Overview of ZTS

We introduce a novel end-to-end system, ZTS, specifically designed to facilitate micro-segmentation in public cloud environments at scale, while maintaining cost efficiency. The system achieves scalable security policy formulation with an innovative role inference algorithm, which integrates network domain knowledge and communication patterns based on the characteristics of each deployment. The algorithm allows for effective collaboration with deployment owners or network administrators to fulfill the diverse requirements of various deployments. Additionally, ZTS minimizes the costs associated with micro-segmentation through a cost-efficient communication graph generator. Figure 4 provides an overview of the ZTS architecture.

Telemetry and Graph Generator. The input to ZTS consists of network flow telemetry, which periodically summarizes the network flows generated by cloud assets such as VMs and containers within a deployment. Most major cloud providers offer this telemetry with minimal impact on customer workloads [10, 11, 17] (Section 5.1 provides more details). This telemetry stream is processed by ZTS's *communication graph generator* (① in Figure 4) to produce a communication graph, similar to those in Figure 2, where each node represents an IP address and each edge summarizes all flows between the corresponding IP pairs. Section 5.2 outlines the detailed system architecture of our communication graph generator.

Policy Authoring. During the security policy authoring phase, ZTS's *role inference trainer* (② in Figure 4) facilitates the creation of μsegments by inferring the roles of each network endpoint. The role inference trainer takes into account: (1) communication patterns from the graphs, (2) detailed information from flow telemetry (e.g., communication ports), (3) information from cloud resources (e.g., asset type), and (4)

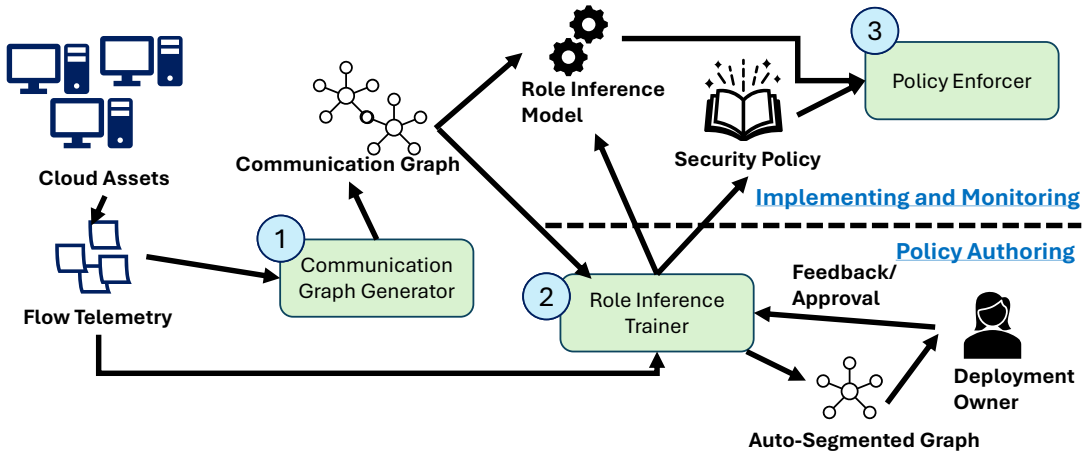


Figure 4: Overview of ZTS.

feedback from the deployment owner (e.g., tags of a subset of assets). The trainer iteratively collaborates with deployment owners to stabilize the micro-segmentation and suggests security policies for each μsegment based on the auto-segmented graphs. Section 4 describes the details of our role inference algorithm.

Implementing and Monitoring. The policy authoring phase generates a role inference model and a set of security policies. During the implementation phase, the *policy enforcer* (③ in Figure 4) maps these security policies from the μsegment level to the IP address level and writes the low-level policies into network security controllers (e.g., [8]) for enforcement. ZTS continuously detects role changes and infers roles for new network endpoints using the role inference model. These changes are applied by the policy enforcer and sent to deployment owners for periodic reviews and model retraining.

4 Role Inference

4.1 Problem Formulation

Our data collection procedure described in §5 enables the construction of a featured IP graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{A}, \mathbf{X})$. The node set $\mathcal{V}$ of cardinality $|\mathcal{V}| = N$ contains all IP addresses in the network, and a directed edge in $\mathcal{E}$ exists between two IPs if traffic was measured from one to the other. The total amount of traffic (measured in bytes) from node i to node j is encoded by the entry A_{ij} of the directed adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$. Notice that we can interpret the i-th row $\mathbf{a}_i^\top$ of $\mathbf{A}$ as features associated with node i capturing the total traffic that this node sent to every other IP in the network. The matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$ contains additional node features, where the i-th row $\mathbf{x}_i^\top$ collects the D features associated with IP i. These features encapsulate information beyond the total traffic sent to other IPs, which is already encoded in $\mathbf{A}$. This includes details such as the primary ports used by an IP, statistical information about the connections—such as the mean and variance of bytes per connection or bytes per packet—and even the count of graphlets [26] or motifs [84] within the communication graph to which a given node belongs. Additionally, they encompass system-related information, such as names and cloud resource SKU information. Our goal is to leverage the information in $\mathcal{G}$ to cluster the IPs into roles $\mathcal{R}$, where two IPs within the same cluster have similar activity. Ultimately, this can allow us to perform communication graph visualization and network management at the role level (instead of the IP level), thus facilitating the generation of rules and improving interpretability (§2.1). The procedure to achieve this goal is detailed next and summarized in Figure 5.

4.2 Infusing Domain Knowledge into Role Inference

In principle, we can concatenate $\mathbf{a}_i$ and $\mathbf{x}_i$ to generate a summary of the activity of IP i, and use these representations to cluster all IPs. However, notice that there is noise and redundancy in these representations, motivating a dimensionality reduction procedure. To be more precise, even if two IPs have the same role, we would not expect the corresponding rows of $\mathbf{A}$ to be identical [72]. Instead, we can think of these as noisy measurements of the canonical activity of that role. In order to recover these main modes of activity, we can perform principal component analysis (PCA). Similarly, the different features in $\mathbf{X}$ originate from the traffic data, so we can expect redundancy among the features, leading to an effectively rank deficient matrix. Based on these considerations, we perform a two-step dimensionality reduction procedure where we first apply a linear method (PCA) followed by a non-linear method (autoencoder). Formally, we consider the singular value decomposition (SVD) of $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^\top$ and keep only the top p components to form $\tilde{\mathbf{X}} = \mathbf{U}_p\Sigma_p$, where $\mathbf{U}_p \in \mathbb{R}^{N \times p}$ contains the p leading (left) singular vectors and $\Sigma_p \in \mathbb{R}^{p \times p}$ is a diagonal matrix containing the corresponding singular values. We select p such that the amount of variance that needs to

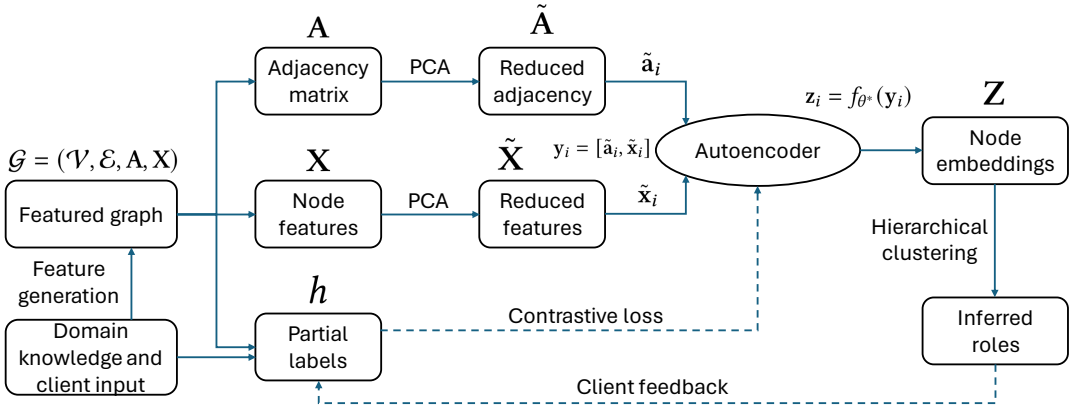


Figure 5: Role inference pipeline. Node embeddings, obtained through an autoencoder, are hierarchically clustered to group the nodes into role classes. Domain knowledge and potential client input is incorporated through feature generation and partial role labels.

be explained is greater than a pre-specified threshold, and we use 99% as a default value in our experiments. A similar procedure is followed to obtain $\tilde{\mathbf{A}} \in \mathbb{R}^{N \times p}$. We denote the concatenation of the i-th rows of $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{X}}$ by $\mathbf{y}_i = [\tilde{\mathbf{a}}_i, \tilde{\mathbf{x}}_i] \in \mathbb{R}^{2p}$.

We further reduce the dimensionality of $\mathbf{y}_i$ via an autoencoder regularized by a contrastive loss. To do so, we assume that we have access to the true roles of a small subset of the IPs. This can be achieved either by relying on domain-inspired rules, such as specific ports associated with pre-established roles, names given to the virtual machines, or an IP-based lookup table, or by obtaining feedback from the network administrator. We denote this partial labeling heuristic as $h : \mathcal{V} \to \mathcal{R} \cup \emptyset$ such that $\hat{r}_i = h(i)$ is the estimated role of node i. Notice that the heuristic might not have enough information to determine the role of a given node. In these cases, $\hat{r}_i = \emptyset$. Let us denote by $\mathcal{L}$ the set of nodes assigned some role label by this heuristic, i.e., $\mathcal{L} = \{i \in \mathcal{V} \,|\, \hat{r}_i \neq \emptyset\}$. Respectively denoting by $f_\theta : \mathbb{R}^{2p} \to \mathbb{R}^d$ and $g_\psi : \mathbb{R}^d \to \mathbb{R}^{2p}$ the parametric encoder and decoder (both multi-layer perceptrons) of our autoencoder, we train the parameters θ and ψ so as to minimize the loss

$$L(\theta, \psi) = \sum_{i \in \mathcal{V}} \|\mathbf{y}_i - g_\psi(f_\theta(\mathbf{y}_i))\|_2 - \tag{1}$$

$$\alpha \sum_{r \in \mathcal{R}} \sum_{\substack{i,i' \in \mathcal{L} \\ h(i)=h(i')=r}} \log \left(\frac{\exp(\mathrm{sim}(f_\theta(\mathbf{y}_i), f_\theta(\mathbf{y}_{i'}))/\tau)}{\sum_{\substack{i'' \in \mathcal{L} \\ h(i'') \neq r}} \exp(\mathrm{sim}(f_\theta(\mathbf{y}_i), f_\theta(\mathbf{y}_{i''}))/\tau)} \right).$$

In (1), sim is a pre-specified similarity measure in the embedding space (such as cosine similarity), τ is a scalar temperature parameter, and α controls the relative weight between both terms of the loss. Notice that the first term in (1) is the classical autoencoder loss that seeks to minimize the difference between the input and output, while the second term pushes together the embeddings of nodes known to have the same role (and separates those known to have different roles). Denoting by θ^* and ψ^* the parameters that minimize the loss in (1), we consider the embedding $\mathbf{z}_i = f_{\theta^*}(\mathbf{y}_i)$ as our concise representation of the activity of IP i, which we gather in the matrix $\mathbf{Z} \in \mathbb{R}^{N \times d}$. Finally, similar to prior work [72], we implement a hierarchical agglomerative clustering algorithm to $\mathbf{Z}$ to obtain our inferred roles for the IPs in the network.

We want to emphasize the following aspects of our role inference algorithm:

1) *User input and feedback*: The labeling heuristic h permits the incorporation of user input into the embeddings $\mathbf{Z}$ through the contrastive loss term in (1). Moreover, after executing the role inference procedure once, the user can be consulted regarding the clusters identified, in a manner similar to active learning [80]. This could involve querying the user about clusters that are the least separable. If discrepancies are found between the clustering output and the user's understanding, the user's knowledge can be fed back seamlessly into h, resulting in improved role inference.
2) *Incorporation of domain knowledge*: Unlike generic role inference methods for graphs [58], domain experts have prior knowledge of what shapes the role of an IP in a network in terms of, e.g., ports used and traffic statistics. Both the partial labeling heuristic h and the node features $\mathbf{X}$ are ideal vehicles to infuse this domain knowledge into our role inference method.
3) *Relation to existing work*: Our pipeline (Figure 5) subsumes most existing methods that cluster nodes based on structural similarities [49, 72]. For example, the method in CloudCluster [72] boils down to performing clustering directly on the rows of $\tilde{\mathbf{A}}$, the adjacency matrix after dimensionality reduction. The incorporation of the autoencoder and, more importantly, the consideration of node features $\mathbf{X}$ and partial labels h constitute key additions with respect to this baseline. We emphasize their value both with a toy example (§4.3) as well as in real-world networks (§6.2).

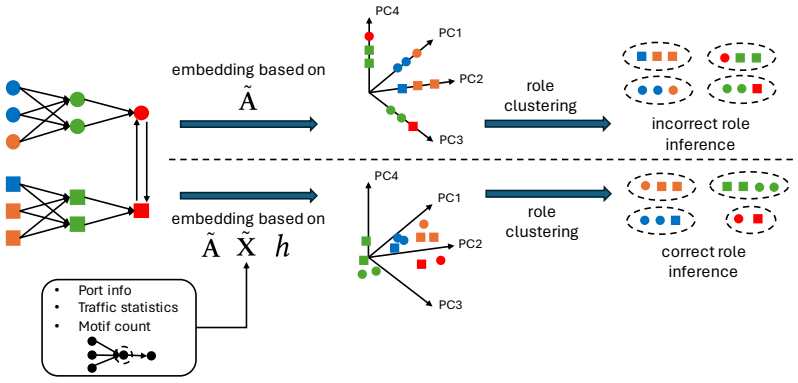


Figure 6: Illustrating the value of domain knowledge. The incorporation of domain knowledge through node features and partial labels leads to the correct inference of roles in the toy network under study.

4.3 Illustrating the value of our proposed pipeline

Let us consider a simple toy network shown in Figure 6-left, where the $N = 12$ nodes have been colored according to their role and the shapes (circles versus squares) represent the top and bottom subnetworks. Traffic originates from two classes of nodes corresponding to different services (blue and orange), this is then relayed through the green nodes to the red nodes. This can be an idealistic representation of a microservice architecture, where the blue and orange nodes are different microservices, the green nodes are the memory object caching systems, and the red nodes are the databases.

We first analyze the roles inferred in this toy network based solely on the adjacency (structural) embeddings $\tilde{\mathbf{A}}$ in Figure 5. Recall that, as explained in §4.2, this is effectively the method proposed in CloudCluster [72]. Assuming that all communication links in Figure 6 have similar total load, the adjacency matrix $\mathbf{A}$ has four dominant principal components. To see this, notice that the rows of $\mathbf{A}$ encode who a given node is talking to, and there are four distinct cases in our toy network (the green circles, the green squares, the red circle, or the red square). Moreover, in $\tilde{\mathbf{A}}$, all the nodes will have an embedding lying on top of one of these four principal components, as shown in Figure 6. Given this marked separation in $\tilde{\mathbf{A}}$, CloudCluster outputs the (incorrect) role labels in Figure 6-top right, irrespective of the specific hierarchical clustering algorithm of choice. This undesirable outcome is resolved by incorporation of $\mathbf{X}$ and h as in our proposed pipeline. For example, the incorporation of port-based features can help distinguish the blue from the orange nodes. Similarly, adding features based on small motifs such as the one illustrated in Figure 6 can bring the embeddings of all green nodes together (notice that only green nodes are at the center of these motifs). Additionally, partial labels of, e.g., blue nodes in the top and bottom parts of the network will help bring the embeddings of all blue nodes together through the contrastive loss in (1). As a result, the 4-dimensional embedding in Figure 6 is more informative than that solely based on $\tilde{\mathbf{A}}$, thus ultimately leading to the correct role inference.

Comparison (and combination) with other baselines. The description of the example above was centered on highlighting the benefits of our proposed pipeline with respect to CloudCluster [72], a state-of-the-art method for extracting roles developed within the networking community. However, it is also important to establish our advantages with respect to more traditional (not domain specific) role inference methods in graphs. This includes simple algorithms based on Jaccard similarity [59] as well as more sophisticated structural embeddings such as SimRank [56] and GAS [49]. Notice that two nodes are deemed close to each other by Jaccard if they share similar neighbors. This essentially means that the similarity of two nodes is given by the similarity between their corresponding rows of $\mathbf{A}$. Consequently, Jaccard inherits the same limitations established for CloudCluster in the example above. Other structural embeddings [30, 49, 56] go beyond comparing similarities in the one-hop neighborhoods and can detect that, e.g., both pairs of green nodes play a similar role in the network. Nonetheless, these methods do not incorporate node features or partial labels, thus falling short of our proposed solution. In our evaluation, we show how our proposed pipeline improves upon all these baselines in real networks. Finally, notice that if the structural embeddings output by any of the existing methods are preferred to the reduced adjacency in Figure 5, we can simply replace $\tilde{\mathbf{A}}$ with the embeddings from any of these methods and implement our pipeline *talis qualis*. In this sense, we can think of our pipeline as a principled way of combining structural embeddings with domain knowledge through node features and partial labels, irrespective of the structural embedding of choice.

5 Low-Cost Telemetry and Analytics

In this section, we discuss the rationale behind our choice of telemetry and our experience in building a cost-effective and scalable communication graph generator for ZTS.

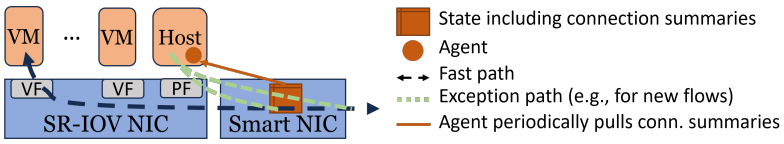


Figure 7: Illustrating how one can capture connection summaries with zero impact to VMs in a public cloud.

Time	Local		Remote		#Packets		#Bytes	
	IP	Port	IP	Port	Sent	Rcvd.	Sent	Rcvd.

Table 3: Schema of connection summaries.

	Azure	AWS	GCP
Name	NSG Flow Logs	VPC Flow Logs	
Agg. Intrvl	1 min	1 min	5s or higher
Content	As illustrated in Table 3		
Sampling	N/A	N/A	3% of Pkts, 50% of Flows
Price	about 0.5$/GB to collect		

Table 4: Details on available connection summaries at three large cloud providers.

5.1 Telemetry Source

As Section 3 mentions, the telemetry source for ZTS is network flow (or connection) summaries [10, 11, 17]. Tables 3 and 4 depict the data format and key details. We select this telemetry source because it (i) can be gathered with minimal disruption to customer workloads, (ii) is cost-effective, allowing for continuous collection, and (iii) is tamper-proof. Consequently, connection summaries are readily accessible to all customers across major cloud providers. Rather than developing a custom telemetry source, we opportunistically use the widely available connection summaries, which contributes to the practical viability of our system.

Cloud providers typically gather connection summaries using programmable NICs directly attached to all hosts in public clouds (Figure 7). These NICs perform various network functions [5, 6, 12, 35, 64, 75]. Connection summaries are stored in smartNIC memory, and an agent on the host periodically retrieves and forwards these summaries to a cloud store or service endpoint. Relative to capturing packet traces [21], logging in the kernel [62, 79] and sampling packets on switches [22], the method in Figure 7 has a few advantages. The size of the logs and memory usage are proportional to the number of concurrent flows. Since these cards already maintain per-flow state [23, 35], recording a few additional counters imposes minimal overhead. Importantly, the impact on customers is minimal; the performance interference from updating a few counters is negligible compared to the extensive network function processing occurring in network virtualization [23, 24, 43, 47]. Additionally, the same functionality can be implemented in the software stacks that handle network virtualization [43].

Collecting packet traces requires the network stack to copy packets, which is more costly compared to the method of generating flow summaries described in Figure 7, especially at NIC speeds reaching 400Gbps. Packet processing elements in switches typically do not have memory to spare to record per-flow state [15, 16], so they sample packets and summarize them at the control CPUs [22, 66]. Some cloud providers use a similar approach, sampling packets and flows to further reduce costs when generating connection summaries [11]. The cost-effective nature of connection summaries enables cloud providers to offer telemetry collection services at a price that makes continuous monitoring required for micro-segmentation feasible, typically $0.25 to $0.5 per GB of collected telemetry [4, 18, 20]. The total telemetry collection cost depends on the number of VMs and connection records generated per hour. In our study, this amounts to approximately $8 to $19 per VM per year. This telemetry is then securely stored to prevent tampering by customers, who do not have access to the host, ensuring availability even if VMs are breached. Customers incur additional storage costs for the telemetry, which can be minimized with an appropriate retention policy. These telemetry collection and storage costs are common to all micro-segmentation solutions and are in addition to the costs illustrated in Table 2. It is also possible that customers have telemetry collection and storage enabled for other monitoring use cases. Hence, we consider these costs orthogonal to our study's scope. Although alternatives like network stack hooks such as eBPF are beneficial, deploying agents on every VM in a public cloud is difficult due to the diverse OS types and the limited control providers have over guest OSes. Even if these agents are deployed, their telemetry can be compromised by malicious code within the VM, making them less reliable during breaches compared to the connection summaries used by our system.

5.2 System for Graph Generation

Our primary goal is to use systems that are available in most large public clouds today to build an analytics system that can analyze roughly 1000 VMs worth of telemetry (e.g., connection summaries at one minute granularity) using a handful of VMs worth of resources. This is roughly a 0.5% surcharge, and such a low COGS is crucial for extensive adoption.

The structure and volume of our telemetry source present challenges in designing a scalable, low-cost analytics system. Cloud providers organize telemetry into numerous small files (e.g., one JSON file per hour per VM) with connection summaries embedded as delimited strings within nested JSON structures. The structuring of the telemetry as many small files is a consequence of the design described in Figure 7, while the nested JSON keeps files compact in representing relationships between flows and various cloud resources. The

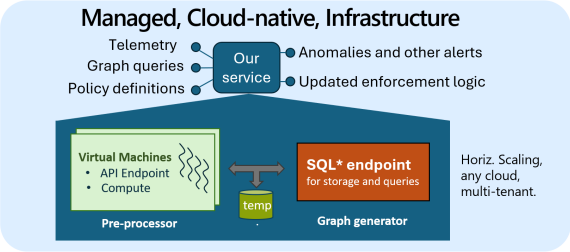


Figure 8: Our analytics system architected as a software-as-service (SaaS) that can adapt to load.

analytics system must efficiently process these small files, unroll nested JSON structures, and parse delimited strings to extract connection summaries. Additionally, it should integrate other information streams, such as IP Address Managers (IPAM) and Virtual Network (VNET) configurations, to enrich the summaries with domain knowledge for the role inference pipeline. The huge volume of connection summaries thus extracted (e.g., #records/min in Table 1) can result in very large communication graphs (e.g., an IP graph is roughly 10x to 100x the number of monitored VMs) requiring careful considerations to keep the processing time and cost low.

Given the telemetry structure, we faced a design choice between a stream processing system for consuming the large influx of small JSON files to generate graphs in real time, and a batch processing system that generates graphs from batched connection summaries. We chose the latter and separated the compute-intensive JSON processing from graph generation, resulting in two processing phases. This approach allows us to pipeline and parallelize the pre-processing stage, preparing batched connection summaries in an optimal format for large batch processing systems, resulting in low-latency processing.

Our system is architected as an always-on, cloud-native service as shown in Figure 8. We structure our system into two main components: (1) a pre-processor that leverages scalable compute resources to handle the large telemetry volume, (2) a graph generator based on a batch processing SQL system [9] to generate the communication graphs from the batched connection summaries. For low-latency graph generation, the batch processing system should be able to generate and complete an efficient query execution plan to process the connection summary batches into graphs. Naïvely, this is a group-by-aggregation query. That is, accumulate the byte, packet, and connection counts between pairs of nodes. The memory need is proportional to the number of node pairs in the graph which would be prohibitive for very large graphs. One mitigation we employ is to focus on the heavy hitters. That is, remote IPs and ephemeral ports that do not individually account for a sizable share of traffic are collapsed together. In fact, the graph sizes in Table 1 collapse IPs contributing less than 0.1% of bytes, packets or connections into one node in the IP-graph. Further, we optimize our query to use Common Table Expressions (CTEs) to eliminate intermediate materialization and to effectively use the system query planner. This approximation and the ability of the SQL system to handle large graphs by effectively using the available memory and disk resources allow us to construct communication graphs on subscriptions with 1000s of VMs in real time using just a few machines worth of resources.

We iterate over our system design and implementation over several developer months as shown in Figure 9, introducing several optimizations: (1) retiring our serverless PaaS system to a custom implementation of our pre-processor, (2) normalizing the tables and related query optimization to improve the batch processing performance and (3) optimizing resource management on the batch processing system. Such iterations enable us to keep pace with some of the large deployments in in Table 1 and keep cost low (e.g., 3 boxes for 5000 VMs).

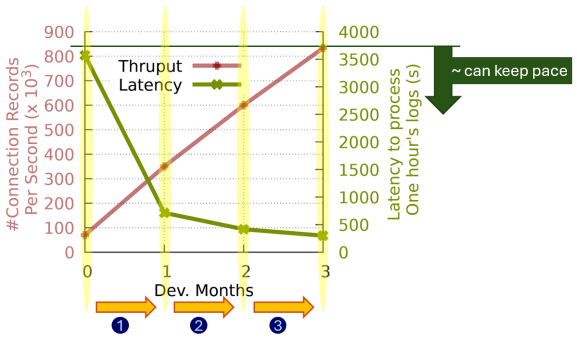


Figure 9: Iterative optimizations applied to our analytics system.

6 Evaluation

We seek to address the following key questions:

- Does the role inference algorithm developed by ZTS demonstrate significantly higher accuracy compared to existing algorithms in real-world applications?
- Does enhanced role inference contribute to the development of improved security policies?
- To what extent can ZTS's graph generator reduce costs compared to existing alternative systems?

6.1 Methodology

Datasets. To evaluate the effectiveness of our role inference algorithm, we curate a dataset comprising network telemetry from 11 real-world deployments within a large public cloud. This includes both first-party and third-party applications of varying sizes, patterns, and requirements. Due to infrastructure differences, the first six deployments utilize telemetry consisting of 5-tuples and flow summaries, while the latter five deployments use telemetry consisting of 5-tuples and process information, and the latter telemetry was also enabled at a larger scale. We conduct developer interviews to ascertain the role of each IP address in each deployment, which becomes the ground truth label in the communication graph. Table 5 summarizes the characteristics of the datasets.

	Telemetry	Graph Size		
		#nodes	#edges	#roles
Deployment A	Flow summary	200	500	14
Deployment B		200	5,000	28
Deployment C		500	9,000	21
Deployment D		200	3,000	28
Deployment E		100	100	12
Deployment F		100	2,000	17
Deployment G	Flow+Process summary	1,500	5,000	30
Deployment H		1,900	17,000	20
Deployment I		12,000	150,000	60
Deployment J		25,000	165,000	84
Deployment K		5,200	90,000	87

Table 5: Real-world deployments used in role inference evaluation: The graph sizes are rounded to preserve workload confidentiality.

Role-inference Baselines. We compare ZTS's role inference algorithm with four baselines that we discussed in §2.1 and §4.3: (1) Jaccard [59], (2) SimRank [56], (3) GAS [49], and (4) CloudCluster [72]. As for clustering algorithms, we use the hierarchical clustering algorithm described in CloudCluster [72] to infer roles for methods that output node embeddings (GAS, CloudCluster, and ours), and we use the Louvain community detection algorithm [37] for methods that output similarity scores (Jaccard and SimRank).

Hyperameters. We use the default hyperparameters from the corresponding papers for the baselines. For our algorithm, we set α in Equation 1 to 20 and τ to 0.05. Our autoencoder is a three-layer neural network with an encoded embedding size that is $\frac{2}{3}$ the size of the input embedding. We train our autoencoder for 1000 epochs with a learning rate of 0.002 and a weight decay of 0.001. To simulate existing asset tags and deployment of owner feedback, we randomly select 10% of the labels as h in our training pipeline. We find that our algorithm performs similarly well across a wide range of values for α, τ, and different autoencoder network architectures.

6.2 Role Inference Performance

Results Summary. Table 6 summarizes the Adjusted Rand Index (ARI) [53] between each role inference algorithm and ground-truth role labels. The ARI measures the quality of clustering and is widely used in role inference literature because it is adjusted for chance. We make three major observations.

First, ZTS's role inference algorithm significantly outperforms all baselines in all but one deployment. The average ARI of our algorithm is 0.77, while the average ARI for Jaccard, SimRank, GAS, and CloudCluster are only 0.33, 0.43, 0.39, and 0.34, respectively. This result demonstrates the effectiveness of ZTS's role inference algorithm in real-world deployments. The only deployment where ZTS does not outperform the best baseline is Deployment C, where our algorithm achieves an ARI of 0.96 compared to the best baseline's ARI of 0.97. This is because the ground truth of Deployment C is partially derived from the Jaccard method, which makes the Jaccard method perform particularly well in this deployment. Despite this, our algorithm still achieves a similar ARI in this deployment.

Second, the performance of state-of-the-art role inference algorithms varies drastically across different deployments. CloudCluster [72] generally performs well for smaller deployments (Deployments A to F) but lags behind other graph mining methods in larger deployments (Deployments G to K). This is partly because the adjacency matrix in larger deployments tends to be more sparse and noisy, and CloudCluster cannot find structural similarities effectively without diving deeper into the graph structures. The performance variation across deployments also underscores the criticality of incorporating domain-specific knowledge and developer insights into role inference processes, as varying deployment environments display distinct traits, and no single static algorithm can universally cater to all cases.

Third, all algorithms tend to perform better at smaller scales (A to F) than at larger scales (G to K). The average ARI of the baseline algorithms decreases from small to large deployments by 0.18, 0.19, 0.20, and 0.64 for Jaccard, SimRank, GAS, and CloudCluster, respectively. ZTS also sees performance degradation from small to large deployments, but at a more moderate 0.14, even with a much better starting point compared to the baselines. These results highlight the necessity of evaluating role-inference algorithms using large-scale deployments to better understand performance in practice. This also means that the performance gap between ZTS and other baseline algorithms increases with the deployment scale.

6.3 Role Inference and Security Policy

To evaluate the correlation between role inference performance and quality of the corresponding security policy, we conduct an assessment of *auto policy generation*. We collect telemetry data from the five larger deployments (Deployments G to K) over one day as the training set. We then infer the roles of each node daily using role inference algorithms. Cluster identities are aligned across different days based on node similarity. Each role is treated as a μsegment, allowing a pair of μsegments to communicate if such communication exists in the training set. We subsequently use telemetry data from the same deployment, four days after the training set, to evaluate the policy violation rate. This rate is defined as the fraction of edges that violate the security policy derived from the training set and the inferred roles. Table 7 summarizes the results.

As Table 7 shows, the security policy generated based on ZTS incurs a significantly lower policy violation rate than the policy generated using other baselines. This is mostly because, while the roles do not change often, the underlying communicating nodes can vary significantly across different days (e.g., talking to a different replica of the same role). If

	Jaccard	SimRank	GAS	CloudCluster	ZTS
Deployment A	0.47 ± 0.00	0.49 ± 0.01	0.50 ± 0.00	0.78 ± 0.00	$\mathbf{0.88 \pm 0.04}$
Deployment B	0.37 ± 0.00	0.33 ± 0.00	0.50 ± 0.09	0.81 ± 0.00	$\mathbf{0.85 \pm 0.03}$
Deployment C	$\mathbf{0.97 \pm 0.00}$	0.93 ± 0.00	0.89 ± 0.00	0.78 ± 0.00	0.96 ± 0.01
Deployment D	0.40 ± 0.00	0.06 ± 0.00	0.16 ± 0.06	0.50 ± 0.00	$\mathbf{0.63 \pm 0.05}$
Deployment E	0.55 ± 0.00	0.60 ± 0.00	0.33 ± 0.14	0.72 ± 0.00	$\mathbf{0.75 \pm 0.02}$
Deployment F	0.32 ± 0.00	0.10 ± 0.00	0.44 ± 0.06	0.73 ± 0.00	$\mathbf{0.90 \pm 0.03}$
Deployment G	0.44 ± 0.00	0.38 ± 0.02	0.41 ± 0.03	0.24 ± 0.00	$\mathbf{0.64 \pm 0.00}$
Deployment H	0.18 ± 0.00	0.14 ± 0.01	0.48 ± 0.01	0.18 ± 0.00	$\mathbf{0.61 \pm 0.03}$
Deployment I	0.41 ± 0.02	0.34 ± 0.01	0.25 ± 0.00	0.06 ± 0.00	$\mathbf{0.70 \pm 0.01}$
Deployment J	0.32 ± 0.00	0.14 ± 0.00	0.10 ± 0.01	-0.01 ± 0.00	$\mathbf{0.85 \pm 0.00}$
Deployment K	0.38 ± 0.00	0.13 ± 0.00	0.11 ± 0.02	-0.09 ± 0.00	$\mathbf{0.65 \pm 0.02}$

Table 6: The Adjusted Rand Index (ARI) between each role inference algorithm and ground-truth role labels. ARI is bounded below -0.5 for highly discordant clusterings and 1.0 for identical clusterings, being close to 0.0 for random labeling regardless of cluster or sample size. We report the mean and standard deviation across 10 runs for each experiment.

	Jaccard	SimRank	GAS	CloudCluster	ZTS
Deployment G	2.1%	11.1%	3.2%	38.4%	0.1%
Deployment H	3.5%	3.8%	2.3%	4.7%	0.2%
Deployment I	6.9%	5.2%	17.3%	6.1%	1.8%
Deployment J	6.5%	9.2%	8%	11.8%	0.7%
Deployment K	7.7%	8.3%	16%	20.7%	2.1%

Table 7: The policy violation rate of auto-generated security policies using role inference results and observed network behavior

the role inference algorithm does not assign the μsegment correctly, these changes will incur many policy violations. Note this evaluation is meant to assess how role inference algorithms can facilitate policy authoring, and the system is not designed to fully automate the process. Hence, a fraction of policy violations in a fully automated setting is expected.

6.4 Cost Effectiveness

To assess the cost of ZTS's graph generation system, we conducted a comparative analysis between our batch processing-based implementation and an alternative solution based on the open-source stream processing system, Apache Flink [38]. Our Flink implementation jointly handles the JSON file preprocessing and graph generation. Our choice of Flink for this alternative solution is based on two considerations. First, Flink is an enterprise-ready system with strong community support, making it a practical choice for our large-scale deployment. Second, we chose Flink over Spark [92], another popular open-source data processing system, because it has demonstrated superior performance in both streaming and batch processing with lower resource consumption [73], which aligns with our experiences with Spark in this context.

Our evaluation involved real-world telemetry data from regional deployments of two first-party services within a prominent public cloud. The details of these regional deployments are summarized in Table 8.

deployment	#nodes	data volume (#recs/hr)
Regional-Portal-Product (D1)	92	1.6M
Regional-K8s Paas (D2)	88	3.9M

Table 8: Characteristics of the regional deployments used in the evaluation of the analytics system. We use regional deployments to preserve workload confidentiality.

COGS. We run ZTS on an infrastructure consisting of a single 8-CPU VM with 32G of memory and 1 compute-unit of the batch processing system that costs in total $845/month. Similarly, Apache Flink runs on a single 64-CPU VM with 256G of memory, and this infrastructure costs in total $2406/month. We choose a larger VM for Apache Flink because its parallelism can benefit from more CPUs and memory so that it can achieve more competitive performance.

Dataset	ZTS	Flink
D1 (1hr)	78	344
D2 (1hr)	109	590
D1 (10hrs)	444	576
D2 (10hrs)	753	1195

Table 9: Processing completion times (in seconds) for the regional deployments. We report both single and multi-hour telemetry processing.

Completion Time. We compare the telemetry processing completion time for both 1-hour and 10-hour datasets across regional deployments. Our system consistently outperforms Apache Flink, as demonstrated in Table 9. Notably, Apache Flink can process a maximum of 5 hours' worth of data concurrently, resulting in its completion time for 10 hours of telemetry being merely double that of 1 hour. Conversely, ZTS delivers superior completion times in all instances while incurring only 35% of the COGS.

Scaling. We also evaluate the scaling behavior of both systems by scaling the telemetry volume by $10\times$ for the regional de-

ployments. We observe that our system outperforms Apache Flink even more significantly in this case, as shown in Table 10. ZTS is 7.5× faster than Apache Flink with only 35% of the COGS, or 21.5× more cost efficient. This is because Apache Flink is not designed to handle such large volumes of data, and its performance degrades significantly as the volume of data increases. In contrast, our system is designed to handle large volumes of data and can scale much better as the volume of data grows.

Dataset	ZTS	Flink
D2 (scaled by 10x)	765	5748

Table 10: Processing completion times (in seconds) for the scaled regional deployment.

It is worth noting that even when utilizing the SQL query optimizations discussed in Section 5.2, the Flink-based streaming system is not as cost-effective as our batch processing system for the scale of telemetry we consider. While Flink and Spark are designed for a wider range of use cases, our results highlight the importance of application-specific design and implementation to achieve significant cost savings, addressing a key obstacle to adopting micro-segmentation systems. We leave the exploration of our batch processing system's applicability to other use cases for future work.

7 Related work

Our key contributions are as follows:

- A novel role inference algorithm for nodes in communication graphs that incorporates domain knowledge and communication patterns.
- A low-cost system that constructs complete and dynamic communication graphs in cloud environments.
- Analysis over communication graphs from many large-scale real deployments.

Here we discuss related work in these areas.

Graph mining and analyses. Existing research on general temporal graph mining [29, 60, 74, 76] largely focuses on social networks, with limited studies addressing challenges in communication networks [57, 83]. Our work tackles novel issues specific to the networking context by analyzing network structures, communication patterns, and incorporating domain-specific knowledge to enhance the inference and application of security policies.

Network telemetry. Many systems analyze passively collected network-wide telemetry [42, 62, 65, 79], but have not achieved widespread deployment or consistent net positive revenue. Highly accurate and scalable sketching algorithms offer estimated answers to network telemetry queries [36, 67, 82, 91, 94], yet they lack generality, addressing only specific query types. Systems like PingMesh [48] and MALT [70] offer insights into network performance and topology but do not measure actual communication flow along data paths. Additionally, most research focuses on packet traces from a few collection points [32, 44, 54, 61, 69, 81], leaving the scalability and generalizability of these techniques to provide network-wide summaries uncertain.

Role inference in graphs and networks. Node embeddings have been utilized to encapsulate structural similarities within graphs [58]. These structural embeddings can subsequently be aggregated through clustering to facilitate role discovery or inference. The majority of existing methods adhere to a two-step procedure, initially discerning structural attributes associated with each node and subsequently transposing these attributes into an embedding space using various techniques [58]. These techniques encompass low-rank matrix factorization [50, 51], random walk-inspired methods [27, 77], and approaches based on (graph) neural networks [49, 85]. Within the domain of networking, the closest work CloudCluster [72], is also based on structural similarities. Section 4 and Section 6.2 discuss the shortcomings of these approaches and evaluate these predominant approaches. In contrast to these methods, our approach allows for the flexible integration of domain-specific knowledge through meticulously crafted node features coupled with a contrastive loss function. Furthermore, it is worth noting that should any existing structural embeddings be deemed more appropriate, our algorithm is designed to integrate them seamlessly into our processing pipeline.

Other studies have leveraged network traffic to infer attributes of certain segments within the communication graph. For instance, Xu et al. [90] classify internet hosts according to traffic similarities, utilizing this classification to identify malicious activities. In the realm of cloud services, patterns of network traffic have been instrumental in pinpointing network nodes or flows that may be compromised [31, 46, 52], as well as selecting suitable candidates for migration [40].

8 Conclusion

In this work, we show that realizing the vision of micro-segmentation at a large scale remains a difficult challenge. We present an end-to-end system that consists of an effective role inference algorithm for network communication graphs and a unique system implementation that drastically reduces the cost of generating communication graphs for network behavior understanding and monitoring. Using 11 large-scale real deployments, we demonstrate the system's capability to more accurately infer roles and generate communication graphs with minimal expense. We hope the findings and insights from our work will stimulate further research to ease the burden of securing public clouds with micro-segmentation.

Ethics: The use of production traces in this paper was governed by an institutional privacy review.

Acknowledgments

We extend our gratitude to our shepherd, Amin Vahdat, and the anonymous reviewers for providing invaluable and constructive feedback. Our thanks also go to our research, engineering and product collaborators, including Anatoliy Panasyuk, Ashish Bhargava, Eliran Azulai, Eli Schwartz, Ido Frizler, Trevor Eberl, Jamie Lee, Kiran Muthabatulla, Kris Zentner, Mariana Alanis Tamez, Prateesh Goyal, Priyatham Allala, Sachin Ashok, and Vahab Jabrayilov for their contributions to production traces, project planning, deployment automation, system optimization, and testing.

References

[1] 2017 Equifax data breach. https://en.wikipedia.org/wiki/2017_Equifax_data_breach, Retrieved on 2023-04.

[2] 2020 United States federal government data breach. https://en.wikipedia.org/wiki/2020_United_States_federal_government_data_breach, Retrieved on 2023-04.

[3] Akamai guardicore segmentation plans and pricing. https://azuremarketplace.microsoft.com/en-us/marketplace/apps/akamai-technologies.akamai_segmentation?tab=PlansAndPrice, Retrieved on 2024-09.

[4] Amazon CloudWatch Pricing - Amazon Web Service (AWS). https://aws.amazon.com/cloudwatch/pricing/.

[5] AWS: Data Transfer Costs for Common Architectures. https://go.aws/3cg5J3O.

[6] Azure: Bandwidth Pricing. https://bit.ly/3Cou81Z.

[7] Azure linux virtual machines pricing. https://azure.microsoft.com/en-us/pricing/details/virtual-machines/linux/, Retrieved on 2024-09.

[8] Azure network security groups. https://learn.microsoft.com/en-us/azure/virtual-network/network-security-groups-overview.

[9] Azure synapse analytics. https://learn.microsoft.com/en-us/azure/synapse-analytics/.

[10] Flow logging for network security groups. https://learn.microsoft.com/en-us/azure/network-watcher/network-watcher-nsg-flow-logging-overview.

[11] GCP: VPC flow logs. https://cloud.google.com/vpc/docs/flow-logs.

[12] Google Cloud: Bandwidth Pricing. https://bit.ly/3Cw83i9.

[13] Horizontal pod autoscaling. https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/.

[14] Illumio core plans and pricing. https://azuremarketplace.microsoft.com/en-us/marketplace/apps/illumioinc1629822633689.illumio_core?tab=PlansAndPrice, Retrieved on 2024-09.

[15] Intel tofino. https://intel.ly/3wxWT8w.

[16] Intel tofino 2. https://intel.ly/3QTeD6F.

[17] Logging IP traffic using VPC flow logs. https://docs.aws.amazon.com/vpc/latest/userguide/flow-logs.html.

[18] Network pricing | Google Cloud. https://cloud.google.com/vpc/network-pricing?hl=en#network-telemetry.

[19] A new walmart 'cloud factory' will accelerate digital innovation, boost business efficiency. https://shorturl.at/amwHI.

[20] Pricing - Network Watcher | Microsoft Azure. https://azure.microsoft.com/en-us/pricing/details/network-watcher/.

[21] tcpdump. http://ee.lbl.gov/tcpdump.tar.Z.

[22] Using netflow filtering or sampling to select the network traffic to track. https://rb.gy/83mcu.

[23] VFP: A virtual switch platform for host SDN in the public cloud. In *NSDI*, 2017.

[24] Azure accelerated networking: Smartnics in the public cloud. In *NSDI*, 2018.

[25] Microsegmentation - global strategic business report. https://www.researchandmarkets.com/report/microsegmentation, 2023.

[26] Nesreen K. Ahmed, Jennifer Neville, Ryan A. Rossi, and Nick G. Duffield. Efficient graphlet counting for large networks. In *2015 IEEE International Conference on Data Mining (ICDM)*, pages 1–10. IEEE Computer Society, 2015.

[27] Nesreen K. Ahmed, Ryan A. Rossi, John Boaz Lee, Theodore L. Willke, Rong Zhou, Xiangnan Kong, and Hoda Eldardiry. Role-based graph embeddings. *IEEE Transactions on Knowledge and Data Engineering*, 34(5):2401–2415, 2022.

[28] Akamai. Akamai Guardicore Segmentation. https://www.akamai.com/products/akamai-guardicore-segmentation.

[29] Alessia Antelmi, Gennaro Cordasco, Mirko Polato, Vittorio Scarano, Carmine Spagnuolo, and Dingqi Yang. A survey on hypergraph representation learning. *ACM Computing Surveys*, 56(1):1–38, 2023.

[30] Ioannis Antonellis, Hector Garcia Molina, and Chi Chao Chang. Simrank++: Query rewriting through link analysis of the click graph. In *VLDB Endowment*, 2008.

[31] Behnaz Arzani, Selim Ciraci, Stefan Saroiu, Alec Wolman, Jack Stokes, Geoff Outhred, and Lechao Diwu. PrivateEye: Scalable and Privacy-Preserving compromise detection in the cloud. In *NSDI*, 2020.

[32] Paramvir Bahl, Ranveer Chandra, Albert Greenberg, Srikanth Kandula, David Maltz, and Ming Zhang. Towards Highly Reliable Enterprise Network Services via Inference of Multi-level Dependencies. In *SIGCOMM*, 2007.

[33] Janet Bailey and Bradley Jensen. Walmart and azure. https://shorturl.at/vMTY0.

[34] Hitesh Ballani, Yatin Chawathe, Sylvia Ratnasamy, Timothy Roscoe, and Scott Shenker. Off by default! In *HotNets*, 2005.

[35] Deepak Bansal, Gerald DeGrace, Rishabh Tewari, Michal Zygmunt, James Grantham, Silvano Gai, Mario Baldi, Krishna Doddapaneni, Arun Selvarajan, Arunkumar Arumugam, Balakrishnan Raman, Avijit Gupta, Sachin Jain, Deven Jagasia, Evan Langlais, Pranjal Srivastava, Rishiraj Hazarika, Neeraj Motwani, Soumya Tiwari, Stewart Grant, Ranveer Chandra, and Srikanth Kandula. Disaggregating stateful network functions. In *NSDI*, 2023.

[36] Ran Ben Basat, Gil Einziger, Roy Friedman, Marcelo C Luizelli, and Erez Waisbard. Constant time updates in hierarchical heavy hitters. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 127–140, 2017.

[37] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Statistical Mechanics: Theory and Experiment*, 2008.

[38] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, et al. Apache Flink: Stream and batch processing in a single engine. In *ICDE*, 2015.

[39] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane: Taking Control of the Enterprise. *ACM SIGCOMM Computer Communication Review*, 37(4):1, October 2007.

[40] Marco Cello, Kang Xi, Jonathan H. Chao, and Mario Marchese. Traffic-aware clustering and vm migration in distributed data center. In *ACM SIGCOMM Workshop on Distributed Cloud Computing*, page 41–42, 2014.

[41] Cisco. Cisco Tetration. https://www.cisco.com/c/en_sg/products/data-center-analytics/tetration-analytics/index.html.

[42] Chuck Cranor, Yuan Gao, Theodore Johnson, Vlaidslav Shkapenyuk, and Oliver Spatscheck. Gigascope: High performance network monitoring with an sql interface. In *SIGMOD*, 2002.

[43] Michael Dalton et al. Andromeda: Performance, Isolation, and Velocity at Scale in Cloud Network Virtualization. In *NSDI*, 2018.

[44] Cristian Estan, Stefan Savage, and George Varghese. Automatically Inferring Patterns of Resource Consumption in Network Traffic. In *SIGCOMM*, 2003.

[45] Bob Evans. Walmart cio: We picked microsoft for huge cloud deal to accelerate digital transformation. https://shorturl.at/aABI3.

[46] Chuanpu Fu, Qi Li, and Ke Xu. Detecting unknown encrypted malicious traffic in real time via flow interaction graph analysis. In *30th Annual Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2023.

[47] Aaron Gember-Jacobson, Raajay Viswanathan, Chaithan Prakash, Robert Grandl, Junaid Khalid, Sourav Das, and Aditya Akella. Opennf: Enabling innovation in network function control. In *SIGCOMM*, 2014.

[48] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, Zhi-Wei Lin, and Varugis Kurien. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *SIGCOMM*, 2015.

[49] Xuan Guo, Wang Zhang, Wenjun Wang, Yang Yu, Yinghui Wang, and Pengfei Jiao. Role-oriented graph auto-encoder guided by structural information. In *Database Systems for Advanced Applications*, pages 466–481, 2020.

[50] Mark Heimann, Haoming Shen, Tara Safavi, and Danai Koutra. REGAL: Representation learning-based graph alignment. In *ACM International Conference on Information and Knowledge Management*, page 117–126, 2018.

[51] Keith Henderson, Brian Gallagher, Tina Eliassi-Rad, Hanghang Tong, Sugato Basu, Leman Akoglu, Danai Koutra, Christos Faloutsos, and Lei Li. Rolx: Structural role extraction & mining in large graphs. In *KDD*, 2012.

[52] Kevin Hsieh, Mike Wong, Santiago Segarra, Sathiya Kumaran Mani, Trevor Eberl, Anatoliy Panasyuk, Ravi Netravali, Ranveer Chandra, and Srikanth Kandula. NetVigil: Robust and low-cost anomaly detection for east-west data center security. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2024.

[53] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2:193–218, 1985.

[54] Marios Iliofotou, Prashant Pappu, Michalis Faloutsos, Michael Mitzenmacher, Sumeet Singh, and George Varghese. Network monitoring using traffic dispersion graphs (tdgs). In *IMC*, 2007.

[55] Illumio. Zero Trust: the security paradigm for the modern organization. https://www.illumio.com/solutions/zero-trust.

[56] Glen Jeh and Jennifer Widom. Simrank: A measure of structural-context similarity. In *KDD*, 2002.

[57] Weiwei Jiang. Graph-based deep learning for communication networks: A survey. *Computer Communications*, 185:40–54, 2022.

[58] Pengfei Jiao, Xuan Guo, Ting Pan, Wang Zhang, Yulong Pei, and Lin Pan. A survey on role-oriented network embedding. *IEEE Transactions on Big Data*, 8(4):933–952, 2021.

[59] Ruoming Jin, Victor E. Lee, and Hui Hong. Axiomatic ranking of network role similarity. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 922–930. ACM, 2011.

[60] Wei Ju, Zheng Fang, Yiyang Gu, Zequn Liu, Qingqing Long, Ziyue Qiao, Yifang Qin, Jianhao Shen, Fang Sun, Zhiping Xiao, et al. A comprehensive survey on deep graph representation learning. *Neural Networks*, page 106207, 2024.

[61] Srikanth Kandula, Ranveer Chandra, and Dina Katabi. What's Going On? Learning Communication Rules in Edge Networks. In *SIGCOMM*, 2008.

[62] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. The Nature of Datacenter Traffic: Measurements & Analysis. In *IMC*, 2009.

[63] John Kindervag, Stephanie Balaouras, and Lindsey Coit. Build security into your network's DNA: The zero trust network architecture. *Forrester Research Inc*, 27, 2010.

[64] Teemu Koponen, Keith Amidon, Peter Balland, Martin Casado, Anupam Chanda, Bryan Fulton, Igor Ganichev, Jesse Gross, Paul Ingram, Ethan Jackson, Andrew Lambeth, Romain Lenglet, Shih-Hao Li, Amar Padmanabhan, Justin Pettit, Ben Pfaff, Rajiv Ramanathan, Scott Shenker, Alan Shieh, Jeremy Stribling, Pankaj Thakkar, Dan Wendlandt, Alexander Yip, and Ronghua Zhang. Network virtualization in multi-tenant datacenters. In *NSDI*, 2014.

[65] Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining anomalies using traffic feature distributions. *SIGCOMM CCR*, 2005.

[66] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. Flowradar: A better netflow for data centers. In *NSDI*, 2016.

[67] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 101–114, 2016.

[68] Mandiant. UNC5537 targets Snowflake customer instances for data theft and extortion. https://cloud.google.com/blog/topics/threat-intelligence/unc5537-snowflake-data-theft-extortion, Retrieved on 2024-06.

[69] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: An ensemble of autoencoders for online network intrusion detection. In *NDSS*, 2018.

[70] Jeffrey C. Mogul, Drago Goricanec, Martin Pool, Anees Shaikh, Douglas Turk, Bikash Koley, and Xiaoxue Zhao. Experiences with modeling network topologies at multiple levels of abstraction. In *NSDI*, 2020.

[71] MSRC. Microsoft actions following attack by nation state actor midnight blizzard. https://msrc.microsoft.com/blog/2024/01/microsoft-actions-following-attack-by-nation-state-actor-midnight-blizzard/, Retrieved on 2024-01.

[72] Weiwu Pang, Sourav Panda, Muhammad Jehangir Amjad, Christophe Diot, and Ramesh Govindan. Cloudcluster: Unearthing the functional structure of a cloud

service. In Amar Phanishayee and Vyas Sekar, editors, *19th USENIX Symposium on Networked Systems Design and Implementation, (NSDI)*, pages 1213–1230. USENIX Association, 2022.

[73] Shelan Perera, Ashansa Perera, and Kamal Hakimzadeh. Reproducible experiments for comparing apache flink and apache spark on public clouds, 2016.

[74] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *KDD*, 2014.

[75] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado. The design and implementation of open vSwitch. In *NSDI*, 2015.

[76] Ahmad Rawashdeh and Anca Ralescu. Similarity measure for social networks – a brief survey. volume 1353, 2015.

[77] Leonardo F.R. Ribeiro, Pedro H.P. Saverese, and Daniel R. Figueiredo. struc2vec: Learning node representations from structural identity. In *KDD*. ACM, 2017.

[78] Scott Rose, Oliver Borchert, Stu Mitchell, and Sean Connelly. Zero trust architecture. Technical report, National Institute of Standards and Technology, 2020.

[79] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. Inside the social network's (datacenter) network. In *SIGCOMM*, 2015.

[80] Burr Settles. Active learning literature survey. 2009.

[81] S. Singh, F. Baboescu, G. Varghese, and J. Wang. Packet Classification Using Multidimensional Cutting. In *ACM SIGCOMM 2003*.

[82] Haifeng Sun, Qun Huang, Jinbo Sun, Wei Wang, Jiaheng Li, Fuliang Li, Yungang Bao, Xin Yao, and Gong Zhang. {AutoSketch}: Automatic {Sketch-Oriented} compiler for query-driven network telemetry. In *Proceedings of USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 1551–1572, 2024.

[83] Prohim Tam, Inseok Song, Seungwoo Kang, Seyha Ros, and Seokhoon Kim. Graph neural networks for intelligent modelling in network management and orchestration: a survey on communications. *Electronics*, 11(20):3371, 2022.

[84] Charalampos E. Tsourakakis, Jakub Pachocki, and Michael Mitzenmacher. Scalable motif-aware graph clustering. In *Proceedings of the 26th International Conference on World Wide Web (WWW)*, pages 1451–1460. ACM, 2017.

[85] Ke Tu, Peng Cui, Xiao Wang, Philip S. Yu, and Wenwu Zhu. Deep recursive network embedding with regular equivalence. In *KDD*, 2018.

[86] Verizon. 2023 data breach investigations report. https://www.verizon.com/business/resources/reports/dbir/.

[87] Verizon. Data breach investigations report: 2008 – 2022. https://www.verizon.com/business/resources/reports/2022/dbir/2022-data-breach-investigations-report-dbir.pdf.

[88] VMWare. VMware NSX. https://www.vmware.com/products/nsx.html.

[89] VMWare. Data center micro-segmentation: A software defined data center approach for a "zero trust" security strategy. Technical report, 2014. https://blogs.vmware.com/networkvirtualization/files/2014/06/VMware-SDDC-Micro-Segmentation-White-Paper.pdf, Retrieved on 2024-09.

[90] Kuai Xu, Feng Wang, and Lin Gu. Network-aware behavior clustering of internet end hosts. In *2011 proceedings ieee infocom*, pages 2078–2086. IEEE, 2011.

[91] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. Elastic sketch: Adaptive and fast network-wide measurements. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 561–575, 2018.

[92] M. Zaharia et al. Spark: Cluster Computing with Working Sets. Technical Report UCB/EECS-2010-53, EECS Department, University of California, Berkeley, 2010.

[93] Wang Zhang, Xuan Guo, Wenjun Wang, Qiang Tian, Lin Pan, and Pengfei Jiao. Role-based network embedding via structural features reconstruction with degree-regularized constraint. *Knowl. Based Syst.*, 218:106872, 2021.

[94] Fuheng Zhao, Punnal Ismail Khan, Divyakant Agrawal, Amr El Abbadi, Arpit Gupta, and Zaoxing Liu. Panakos: Chasing the tails for multidimensional data streams. *Proceedings of the VLDB Endowment*, 16(6):1291–1304, 2023.