NetPanel: Traffic Measurement of Exchange Online Service

Yu Chen, *Microsoft 365, China;* Liqun Li and Yu Kang, *Microsoft Research, China;* Boyang Zheng, Yehan Wang, More Zhou, Yuchao Dai, and Zhenguo Yang, *Microsoft 365, China;* Brad Rutkowski and Jeff Mealiffe, *Microsoft 365, USA;* Qingwei Lin, *Microsoft Research, China*

https://www.usenix.org/conference/nsdi23/presentation/chen-yu

This paper is included in the Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation.

April 17–19, 2023 • Boston, MA, USA

978-1-939133-33-5

Open access to the Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation is sponsored by

NetPanel: Traffic Measurement of Exchange Online Service

Yu Chen[†], Liqun Li[◇], Yu Kang[◇], Boyang Zheng[†], Yehan Wang[†], More Zhou[†]
Yuchao Dai[†], Zhenguo Yang[†], Brad Rutkowski[‡], Jeff Mealiffe[‡], Qingwei Lin[◇]
[†]*Microsoft 365, China,* [◇]*Microsoft Research, China,* [‡]*Microsoft 365, USA*

Abstract

Global cloud applications are composed of thousands of components. These components are constantly generating large volumes of network traffic, which is a major cost of cloud applications. Identifying the traffic contributors is a critical step before reducing the traffic cost. However, this is challenging because the measurement has to be component-level, cost-effective, and under strict resource restrictions. In this paper, we introduce NetPanel, which is a traffic measurement platform for the Exchange Online (EXO) service of Microsoft. NetPanel fuses three data sources, namely IPFIX, Event Tracing for Windows (ETW), and application logs, to jointly measure the service traffic at the component level, where each component is owned by a service team. NetPanel uses several schemes to reduce the measurement overhead.

NetPanel has been in operation for more than one year. It has been used to profile network traffic characteristics and traffic cost composition of EXO. With the insights obtained through NetPanel, we have saved millions of dollars in network resources. The overhead of running NetPanel is relatively small, which requires less than 1% CPU and disk I/O on production servers and less than 0.01% of EXO computation cores to process the data in our big-data platform.

1 Introduction

Cloud applications, such as Exchange Online (EXO), are composed of thousands of components running on hundreds of thousands of servers, developed and maintained by engineers from many different teams. In EXO, one component is a module that performs a specific function, as an entire or part of a process. The Internet Information Services (IIS) [4] based proxy, running on frontend (FE) servers, routes traffic for different components such as REST [23], EWS [5] and MAPI [6]. The traffic of each component is owned by a specific engineering team. These components are sending tremendous traffic across data centers, which incurs great costs. Any defect in a single component may lead to widespread traffic flood. Furthermore, due to the massive number of components, the limited shared bandwidth could be easily drained by low-priority traffic. In these cases, customers could suffer from long latency or even connection loss [7, 9, 12, 14]. For example, an incident caused by anomalous traffic was reported by Azure [2] on June 14^{th}, 2021 where some customers received errors when performing service management operations. The root cause was high CPU consumption and request timeouts caused by an unexpected surge in internal traffic. The issue was mitigated by adding rules to block internal traffic on a subset of backend servers.

Cloud application owners have built plenty of monitors for incidents and performance regressions [1, 3, 20]. Such monitors are typically based on availability or latency metrics, which are insensitive to traffic issues. Therefore, there is still undesired traffic caused by various reasons, such as code bugs or misconfigurations. Over time, these hidden bugs become extremely difficult to trace as everyone takes them as necessary bandwidth requirements. Unnecessarily more capacity planning budget is therefore needed in subsequent years. The extra cost will be millions of dollars per year given the application scale. For example, in one case, we caught a configuration error that nearly quadrupled its peak traffic. In another case, we found that one service was sending requests globally, while these requests can actually be handled within a location. These cases will be detailed in Section 5.2. Although there is existing work to detect anonymous traffic bursts [29, 31, 39, 44], few have provided insights for continuous traffic optimization. There is a body of work on misconfiguration detection [51, 53] and safe deployment [34], but their solutions are not specially designed for traffic-related issues. In particular, some traffic issues can only be identified through long-term continuous monitoring.

While it is vital to continuously monitor the traffic flow for a cloud application, the dynamic and heterogeneous nature of a global cloud application makes this difficult. An efficient traffic monitoring system for cloud applications must satisfy the following requirements:

(1) The measurement should provide component-level

results. To efficiently identify the owner of the traffic, component-level measurement is required. A component is typically owned by a single engineering team. Researchers have been exploiting network measurement tools such as IPFIX/Netflow in the last decade [22, 37, 55] to gain insights into the network traffic in large-scale cloud infrastructures. These approaches operate on network routers, so they cannot identify components at the application layer. Server network analysis tools [8, 11, 36, 43] can observe the traffic sent by each process. However, multiple components can share a single process. These tools cannot distinguish the traffic emitted by components sharing the same process.

(2) The size of daily measurement results should be small (in GB). To draw an effective conclusion on traffic observation, engineers need to query data over a long period of time. In addition, there are cases where successive interactive analysis is needed, such as the case we discuss in Section 5.2.1. A user query response should be returned within a few seconds. On the other hand, global cloud applications are constantly generating a vast amount of traffic logs. For instance, the size of IPFIX data is more than 10TB per day. Event Tracing for Windows (ETW) [11] data and application logs are in PB. Directly joining PB and TB data at a daily frequency is impractical due to resource constraints. Directly running queries on these log data imposes huge data processing costs and unacceptable query latency.

(3) The collector in production environment should run under strict resource restrictions. Cloud applications such as EXO provides high Service Level Agreements (SLA) [13] to the customers. Therefore, the service has strict restrictions in terms of the resource used by any single component on the servers to ensure a quick response to customer requests. To collect event logs, ETW needs to run on the servers by the side of service components. Pulling all the network metrics from ETW regularly is prohibited in the production environment because it will exhaust the CPU and disk I/O on the production servers.

To address the aforementioned challenges, we design NetPanel, a cost-effective continuous traffic profiling tool for EXO. NetPanel takes three data sources, including IPFIX, ETW, and application logs. We fuse these data sources to jointly provide component-level measurement results. We reduce the data size with feature translation, data splitting, and data aggregation to keep all measurement results at several GB per day, which will be detailed in Section 4. To reduce the resource consumption of ETW, we only retain the data for the top k ports obtained from IPFIX.

NetPanel has been safeguarding the network traffic of EXO for more than 1 year. It brought us valuable insights into our traffic and helped us save millions of dollars per year. We introduce 4 real-world cases in Section 5.2. NetPanel runs with negligible impact on our production servers (less than 1% increase in CPU and disk IO). The data processing cost for our big-data platform is also minor given the scale of EXO (less than 0.01% of EXO cores). For a query for data in a 60-day period, the response can be returned within 30 seconds.

Roles	Abbr.	Functionality
Frontend	FE	Connects with customers and routes customer requests
Backend	BE	Stores mailboxes, delivers emails, and provides site resilience
Active Directory	AD	Holds and queries customer metadata

Table 1: Server roles and their functionalities.

Our key contributions and insights in this work are summarized as follows:

- We discuss the requirements and challenges of measuring the traffic for a global scale application, i.e., EXO. We show that telemetry data should be attributed to an organizational structure, such as a team of engineers, to actually drive cost reduction. Moreover, daily data size should be small enough to provide insight into how traffic data changes over time.
- We present our novel traffic measurement design which fuses IPFIX, ETW, and application logs to achieve component-level measurement. We demonstrate that, with proper data volume reduction, it is feasible to join data sources across routers and servers. We show that cross-validating data for integrity is feasible and crucial.
- We share our observations on traffic characteristics of EXO in production environment. Specifically, we figure out that heavy hitters (top ports/components) are stable in EXO, and this feature can be used to reduce data volume. We also demonstrate how NetPanel can help reduce traffic costs through real-world case studies.

We believe that the experience of operating NetPanel provides valuable guidance to other cloud applications on how to monitor and optimize their network traffic.

2 Background

This section introduces the EXO service traffic and explains how these traffic flows are generated. Then, we share the measurement tools available in EXO and their capabilities.

2.1 EXO Service Traffic

EXO operates in numerous datacenters around the world. There are hundreds of thousands of servers all over the world serving its enormous user community. The servers are categorized into three server roles: frontend routing proxy (**FE**), backend mailbox (**BE**), and directory (**AD**), as summarized in Table 1. FE servers, which sit behind load-balancers, serve customer requests over direct connections. AD servers hold information about users, mailboxes, and other customer metadata. BE servers provide storage for mailboxes and are responsible for the delivery of emails to/from mailboxes. Multiple

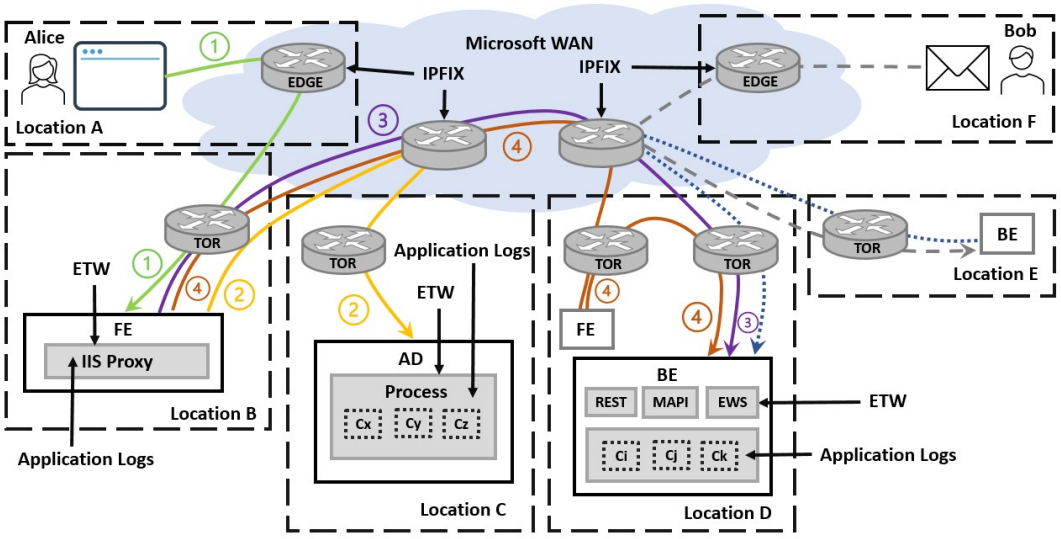


Figure 1: EXO traffic overview. When Bob sends an email to Alice, replication occurs, and Alice later reads the email replica. During the email-reading process, 6 Internal Long-haul traffic flows and 2 Internet traffic flows are generated.

copies of mailboxes are geographically-dispersed to provide high availability and site resilience. Each role of servers (i.e., FE, AD, or BE) hosts a group of services in order to provide functionalities as designed.

EXO servers communicate with each other when serving customers. We divide EXO traffic into two types: *WAN* traffic and *Metro* traffic. The WAN traffic goes through routers in WAN; and the rest, namely Metro traffic, travels within data-centers in the same location. We are particularly interested in the WAN traffic because it costs more than 90% of the annual bill. There are two subtypes of WAN traffic. The first is the traffic between EXO servers and user clients (Internet traffic) and the other among EXO servers (Internal Long-haul traffic). Internet traffic is responsible for around 10% of WAN traffic and Internal Long-haul traffic takes the rest.

The Internal Long-haul traffic are assigned to different priority tiers, *Tier 0* and *Tier 1*, by Bandwidth Broker [52]. Tier 0 is of higher priority with higher Quality of Service (QoS). Tier 1 traffic has a lower priority, and it is routed through sub-optimal paths and is dropped by routers first when congestion occurs.

2.2 How EXO Traffic Generated?

We use a typical scenario shown in Figure 1 to describe how the traffic is generated in EXO. In this scenario, Bob uses the Desktop client to send an email to another user Alice. Alice then reads her email through a web client. Bob in Location F sends the email to Alice's mailbox in Location E. The mailbox is replicated to Location D for high availability. These are annotated by the gray dashed line and the blue dotted line in Figure 1. Later, Alice reads her email from the mailbox. There are 4 steps in this email-reading process, detailed as follows:

Step 1: Alice uses the web client in Location A to send a request to the closest FE server. In this example, we assume the closest FE server resides in another Location B.

Step 2: The FE server talks to an AD server in Location C to query which BE server knows where the active copy of Alice's mailbox is hosted. In this case, the BE server in Location D is returned by the AD server.

Step 3: The FE server queries the BE server in Location D to ask which BE server hosts Alice's mailbox. In this example, the BE server in Location D happens to host Alice's mailbox, so it responds with itself.

Step 4: The FE server in Location B forwards the request to another FE server in Location D and that FE server in Location D will further transfer the request to the BE server in Location D. The mailbox's response is returned to Alice in the opposite direction along the paths of Step 1 and Step 4.

During the email-reading process, the traffic in Step 1 and the traffic from FE to the web client in the mailbox's response is Internet traffic because the FE server is talking to an external client outside of Microsoft. The traffic in Steps 2 and 3, and the traffic from the FE in Location B to the FE in Location D in Step 4 is Tier 0 traffic because the source and destination EXO servers are in different locations. In Step 4 (the orange arrow in Figure 1), the traffic between FE servers is Tier 0 traffic, while the traffic between FE and BE servers in Location D is Metro traffic. We use the example shown in Figure 1 only to explain the generation of Tier 0 traffic. Most

of the traffic would be Metro traffic when the servers involved are in the same location.

The replication traffic, represented as the blue dotted line in Figure 1, is Tier 1 traffic. Tier 1 traffic mostly consists of background traffic that does not serve user activities and therefore does not have rigorous latency requirements. Typical cases include: (1) data replications for high availability; (2) non-urgent mailbox migrations; (3) uploading logs.

In the example in Figure 1, Alice's request first hits the FE server in Location B (step 1), then is routed to the FE server in Location D (step 4), and finally reaches the BE server in Location D (step 4). The request is served by the REST [23] component on the BE server. In this situation, only the application logs of the FE servers capture this long-haul traffic between the two FE servers (Location B $\Rightarrow$ D). The components on BE servers in Location D, such as REST [23], MAPI [6], and EWS [5], are behind the FE proxy in the same location. As a result, the BE servers are unaware if the request originates from another location. Sometimes, multiple components on one BE server may even share the same process. Therefore, server network analysis tools [8, 11, 36, 43] are insufficient to provide component-level traffic measurements. In summary, application logs are necessary to perform component-level traffic measurements.

2.3 Traffic Measurements

There are two kinds of traffic measurement methods available in our data centers: *on-router* and *on-server* measurements.

One of the commonly used on-router measurements is flow monitoring [26, 35, 41]. There are two standards, i.e., Netflow [21] and IPFIX [15], that have been used for years. Flow monitoring samples packets with a certain probability and aggregate them into flows. A flow is a sequence of packets with the same IP 5 tuples (*src./dst. addresses*, *src./dst. ports*, and *protocol*). Flows are uploaded to a centralized storage.

When measurements are made on the servers, common toolkits include Tcpdump [8] and Event Tracing for Windows (ETW) [11]. Tcpdump is a well-known library that provides powerful packet analysis capabilities on Linux, while Windows uses ETW to collect system network events. These tools can monitor the traffic usage of all processes on a machine.

We annotate these measurement schemes in Figure 1. IPFIX collects traffic data on WAN routers. ETW collects system network events and provides traffic statistics for processes on servers. We further add application logs, which are generated within the services and are owned by different teams for debugging purposes. Application logs are request-focused. For a specific request, application logs record the timestamp, the component that serves the request, the local server name, the remote server name, the latency, the request and response content size, the remote port, etc.

We summarize the measurement capabilities of the three schemes in Table 2. TimeStamp, IP, Port, Process, and Traffic Size are general definitions. TimeStamp, an IP pair (source and destination), and a Port pair identify a unique flow. Process is the information of processes that are sending and receiving the traffic. We need a Differentiated Services Code Point (DSCP) tag [18] because Bandwidth Broker uses it for traffic QoS classification. Packets with different DSCP tags are classified into different priority tiers. IPFIX covers IP, port, and DSCP but cannot cover the process and component information which are available only on servers. ETW can further measure processes, but cannot cover the exact component as discussed in our example in Sec. 2.2. Application logs contain the request and response content sizes of components but not counting the sizes of the packet headers. In addition, application logs do not capture packet loss or retransmission. In conclusion, we need to fuse IPFIX, ETW, and application logs to achieve component-level measurement.

	Timestamp	IP	Port	DSCP	Process	Component	Traffic Size	Request Size
IPFIX	✓	✓	✓	✓	✗	✗	✓	✗
ETW	✓	✓	✓	✗	✓	✗	✓	✗
App logs	✓	✓	✓	✗	✓	✓	✗	✓

Table 2: Available Measurement Methods

3 Motivation and Design Goals

In this section, we state our motivation to design NetPanel and further define the goals to be met for our design along with the challenges to be resolved.

3.1 Motivation

In the EXO service, many components are working together to serve customers. These components send large amounts of traffic globally, which is very expensive. The large cost has motivated the application owner to understand the current traffic, reduce the traffic cost, and ensure there is no traffic waste. Furthermore, when a development team adds a new feature to reduce their traffic, they also need a tool to validate the traffic change of their component. Before NetPanel, each team only monitors their own request amount, leading to an isolated and incomplete view, which makes it hard to motivate traffic optimization efforts, verify data correctness, and detect traffic-related issues. On the other hand, without component-level information, it is non-actionable even if anomalous traffic is detected. With the increasing complexity of modern global-scaled software, this requirement becomes more and more urgent, which motivates us to build NetPanel.

3.2 Design Goals and Challenges

The design of NetPanel has to meet the following goals and address the corresponding challenges.

Goal-1: The measurement should provide component-level results. The overall EXO traffic should be divided into

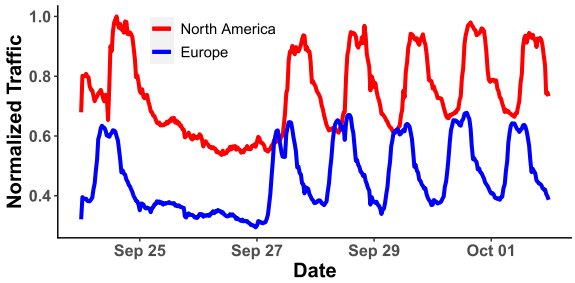
various components. EXO runs on hundreds of thousands of machines of three server roles. Machines of the same server role hold the same set of components. A component is owned by an engineering team. As long as the component's traffic is identified, the engineering team can take action to optimize its cost. NetPanel should provide the capability to establish the mapping between the components and their traffic flow.

Recent measurement works [33,45,46,48,54] are on-router measurements, so they cannot support measurement at component granularity. Traffic Refinery [19] identifies the component flows by inspecting DNS queries and manually specifying matches between components and their IP prefixes. This does not work in EXO because it assumes that each IP must correspond to at most one component. However, in our case, an IP can be shared by many components at the same time. Google uses Bandwidth Enforcer (BwE) [32] to allocate bandwidth at task granularity. A task may contain multiple processes, and therefore, BwE cannot be used to monitor the bandwidth for components sharing a process. NetPanel addresses this challenge by leveraging application logs to fill in the component property. We will describe how to jointly consider the three data sources, i.e., IPFIX, ETW, and application logs, to recover component-level traffic throughout Section 4.

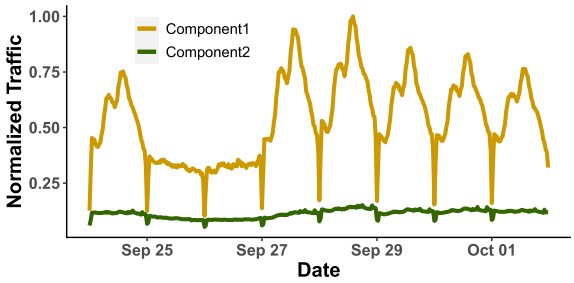
Goal-2: The daily measurement result size should be small (in GB). In EXO, IPFIX generates several TB of data every day, while the daily application log in PB. After compression, the data size will be reduced to ∼10%. However, IPFIX data is more than 10TB per day, so the data size will still be too large after compression. To draw an effective conclusion on traffic, engineers usually have to do consecutive analysis over long time intervals and compare the results. The system should provide quick responses to user queries and consume few resources. From our experience, we need to limit the result size to several GB per day, so that analysis over a long time interval is allowed.

The data used for analysis should cover a continuous time interval of at least several weeks to overcome the dynamic nature of network traffic. Figure 2a shows the traffic variation in EXO over a time interval of more than one week in North America and Europe. The traffic volume highly aligns with user activities, with more traffic during working hours and much less at night and on weekends. The valley values are nearly half of the peak values. The large fluctuation rate is likely to override the traffic change introduced by a new feature if we make queries only over a short time interval of a few hours. Moreover, different components can have different traffic patterns. We show the traffic patterns of two components in Figure 2b. The traffic of Component 1 fluctuates greatly, while that of Component 2 is relatively stable.

Many approaches have been proposed to reduce the data size. However, they cannot satisfy our requirements for different reasons. IBM cloud [39] is dealing with a 2.35TB log each day, but their approach only reports anomalies without any details on current and historical traffic usage. Analysis



(a) All geographical regions have fluctuations between daytime and night but follow a similar weekly pattern. The valley values could be half of the peak values.



(b) Different components have various traffic patterns.

Figure 2: Huge variations in time and components domain.

farm [47] proposes a cloud log analysis platform and aggregates IP addresses to IP-groups. We achieve something similar with feature translation (detailed in Section 4.2.1), but this alone would only reduce the data size to hundreds of GB per day. Anwar et.al. [16] claim to reduce the data size by up to 80% using different sampling frequencies and storage aggregation for different metrics. In our case, we collect only one metric but the data size must be reduced thousands of times. NetPanel addresses the challenge with multiple steps, which will be introduced in Section 4.2.

Goal-3: The collector in the production environment has to run under strict resource restrictions. The data collector has to run continuously in the production environment. There are strong resource restrictions (CPU, memory, disk I/O, etc.) for measurement tools in order to reserve as many resources as possible to serve user requests. The resource consumption of the collectors has to be very small.

We abandoned pulling all network metrics from ETW in the EXO production environments because of performance issues. In EXO, every single component should use no more than 5% CPU. It is restricted to log no more than 32MB of data on local disks every five minutes. Running ETW and writing all the metrics to the local disk exceeds the limit as shown in Section 6.1. The ETW data size for a single day is in PB. NetPanel reduces ETW collectors' resource consumption by only recording the traffic data of the top k ports. This greatly reduces the log size as well as the computation resource and disk I/O throughput. We explain how the top k ports are

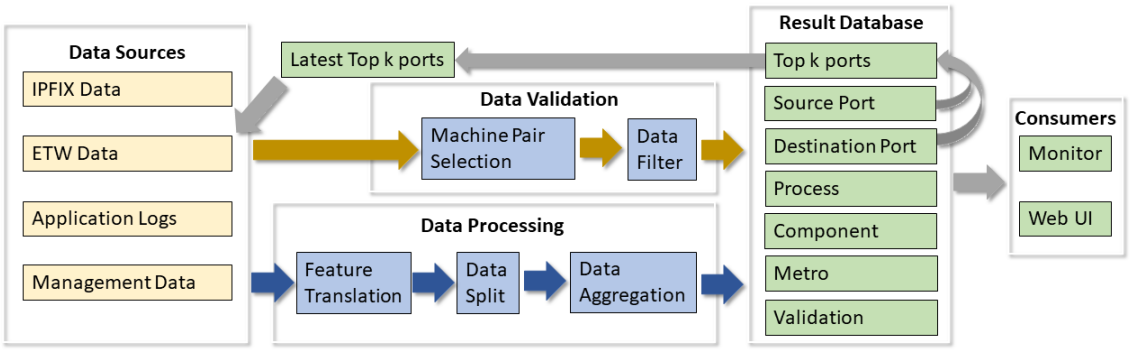


Figure 3: NetPanel Architecture Overview.

identified in Section 4.2.2.

4 System Design

Figure 3 shows the overview of our design. The data sources include IPFIX, ETW, application logs, and management data. There are two separate pipelines in the system. The first one (blue arrows) is responsible for reducing the data size while preserving important attributes. The second one (yellow arrows) is used to cross-validate the data from IPFIX and ETW to ensure data integrity.

The output of both pipelines is fed into the result database. The result database is a light-weighted database that can respond to web services as well as user queries in near real-time. We use Azure Data Explorer (Kusto) [10] for the result database. The result database contains tables for different features. The schema of the tables is available in Table 3. We use port tables in the result database to derive the top k ports with the largest traffic volume and use them as a filter for ETW collectors to reduce their resource consumption in the production environment (grey arrows). We also set up a monitoring service to help component owners detect anomalies and continuous upticks in their traffic usage. A WebUI is provided for traffic data visualization.

4.1 Data Sources

Before NetPanel, EXO deployed a background packet trace monitor on its servers to provide network statistics such as throughput, RTT, etc. However, without component-level information, this background tracking is not adequate to drive owners to optimize their traffic. This experience drives us to choose a different set of data sources for NetPanel.

NetPanel takes three measurement inputs: IPFIX, ETW, and application logs. The data is uploaded to COSMOS [38] on an hourly basis. COSMOS is a Hadoop-like distributed data storage and processing platform. We convert different types of data into a uniform format like Table 2. In addition to the measurement data, NetPanel also takes in management data. The management data contains the mapping between an IP address and its location and server role. The management data is uploaded daily because it is relatively static.

NetPanel handles the three data sources independently. IPFIX data size is more than 10TB per day. ETW and application logs data sizes are several PB per day. It is too expensive to directly join the three data sources based on shared features (i.e., TimeStamp, Port, IP, and Process). We thus process the raw data independently and only store aggregation results in the result database as detailed in the next section.

4.2 Data Processing

The data processing pipeline consists of three consecutive steps. (1) Feature translation: translate the raw information of the traffic data into a set of features. (2) Data split: divide the traffic data associated with source-destination port pairs into two separate views, i.e., source-port view and destination-port view. (3) Data aggregation: aggregate the traffic data into various feature tables.

4.2.1 Feature Translation

In Microsoft, different services occupy different blocks of IP ranges, so we use IP addresses[1] to retrieve EXO traffic. Then, we use management data to translate machine IPs into locations and server roles. The server role is among AD, FE, and BE. Location is the metropolitan area where the server locates. This translation reduces the storage requirements from trillions of IP pairs to millions of feature pairs.

We translate location pairs to Rate-Regions. Location pairs are only used to get the prices of traffic flows. A longer distance implies a higher price. We thus use a new feature called Rate-Region to replace the location pair of a flow. Azure charges Microsoft internal services a unified price ($/Mbps) for the flows traveling a geographical continent or an ocean, like North America, Europe, Atlantic, etc. There are only

[1]The IP addresses here are all Microsoft IPs.

Table	Data Source	Schema - Keys	Schema - Value (Bytes)
Source/Destination-Port	IPFIX	TimeStamp, ServerRole, RateRegion, Port, DSCP	TrafficSize
Process	ETW	TimeStamp, ServerRole, RateRegion, Port, Process	TrafficSize
Component	Application logs	TimeStamp, ServerRole, RateRegion, Port, Process, Component	RequestSize

Table 3: Columns in result tables.

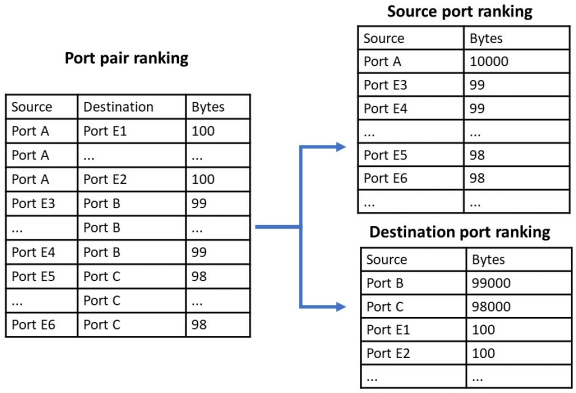


Figure 4: Split the port pair view into the source-port view and the destination-port view, separately. Port Es are ephemeral ports. Well-known ports A, B, and C pop up in the ranking list after the split.

about 10 Rate-Regions. Translating location pairs to Rate-Regions reduces the data size by ∼99.1%. We note that the feature translation does not hinder traffic debugging. One component is deployed on all machines of the same role. In case we detect anomalous traffic for one component in a Rate-Region, engineers can randomly pick a server in that Rate-Region and start the debugging from there.

4.2.2 Data Split and Aggregation

We aggregate the raw data based on the source port and destination port, separately, ranked based on the traffic volume. This results in two views: the source-port view and the destination-port view, which have two benefits: (1) significantly reduces the table size, i.e., from O(# source port $\times$ # destination port) to O(# source port + # destination port); (2) helps pop up "well-known" ports in ranking because the traffic of ephemeral ports will converge to relatively smaller numbers than the traffic of "well-known" ports after aggregation. An example of port view splitting is shown in Figure 4.

Once we have identified the "well-known" ports, we take an additional step to filter the ephemeral ports in the source-port table and destination-port table. For every single time slot, we aggregate all ports that contribute less than 1% of the total traffic to one record and mark it with a special tag. Recall that our goal is to reduce overall traffic cost, the threshold of 1% is small enough. We will show in Section 5.1.1 that a handful of ports dominate the traffic usage. The outputs are streamed into Kusto on a daily basis.

We aggregate the records from different data sources to obtain different feature tables. The schemas of these tables are shown in Table 3. We generate the Process table from ETW data, the Component table from application logs, and the Source-Port table and Destination-Port table from IPFIX data. We aggregate the TimeStamp with 5-minute intervals. TrafficSize is aggregated with sum operation for all the records with the same key. For example, the tuple of <TimeStamp, ServerRole, RateRegion, Port, DSCP> is the key for the Source-Port table. The TrafficSize in the Source/Destination-Port table and the Process table is the sum of network traffic while the RequestSize of the Component table is the sum of request/response content sizes, excluding packet headers.

4.3 Data Validation

As a global-scale system, there are many factors that result in data corruption such as broken hardware and data loss. During the development of NetPanel, we experienced a partial data loss of the IPFIX data due to the data collector pipeline change. In fact, data missing issues are non-trivial to detect due to the vast amount of data being processed. NetPanel resolves the problem by cross-validating the results obtained from its multiple data sources.

The key idea is to cross-validate the ETW data and IPFIX data because the traffic size captured with ETW should match that with IPFIX. However, because IPFIX does packet sampling while ETW captures all traffic, we need to recover IPFIX data before comparing them. The recovery function is shown in Eq. (1). We do not validate the traffic size using application logs, because they only capture the content sizes without the request headers.

$$IPFIXBytes = \frac{(PacketSize + HeaderSize) * PacketNumber}{SamplingRate} \quad (1)$$

PacketSize, PacketNumber, and SamplingRate are available from the IPFIX data. Note that we add the ethernet header length (i.e., HeaderSize) to PacketSize to get the actual ethernet frame size on the wire. Based on the law of large numbers, we believe that: Given a machine pair that continuously send a lot of traffic to each other, an effective estimation of IPFIX should be close to ETW data. We conducted an experiment to validate our recovery approach. The result shown in Figure 5

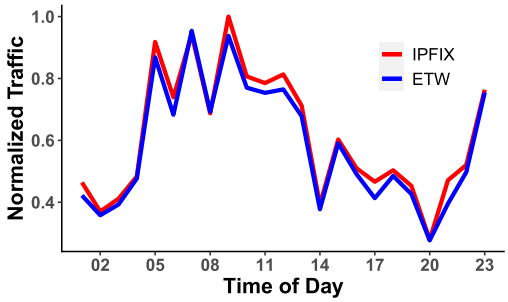


Figure 5: Recovered traffic for a single pair of machines in one day from IPFIX matches the traffic from ETW.

can confirm that the recovered IPFIX traffic size matches the ETW measurement precisely.

We selected the top 1000 pairs of machines in EXO that send the most traffic in a day for data validation. Machine pairs are re-selected daily as machine pairing relationships may change when machines become online or offline. After selecting the machine pairs, we insert their IPFIX data and ETW data into the validation table. Direct data comparison could be done with queries to the validation table. Data validation is done on a daily basis.

4.4 Result Database

The result database contains the Source-Port table, the Destination-Port table, the Process table, and the Component table. These tables are used to analyze traffic for component owners. A validation table is stored for data integrity checks. A top k ports table is created based on the Source-Port table and Destination-Port table. It picks the top k ports in both tables. These top k ports will be used as filters for the ETW collector in the future.

NetPanel uses the latest top k ports as a filter for the ETW collector to save production resources. These top k ports are usually obtained last day. IPFIX and Application Logs have been deployed in our environment for a long time. These data collectors have been optimized to guarantee no impact on SLA. ETW collector is a newly added collector running on production servers. As we will show in Section 6.1, directly collecting all ETW records will result in tens of times the CPU and disk IO usage. We must reduce ETW collector's resource consumption on production servers. With the observation in Section 5.1.1 that the top 10 ports cover more than 94% of the traffic and stay stable over weeks. We decide to use the Top-k algorithm as a filter in the ETW collector to reduce the data to be collected and thus reduce its resource consumption.

4.5 Component Traffic Estimation

We built a web UI to provide engineers with a convenient way to analyze the traffic. The dashboard shows component-level traffic like shown in Figure 2b (in Section 3). If the component monopolizes a process, its traffic could be directly obtained from the Process table as described in Table 3. Otherwise, we calculate its traffic in an approximated fashion. The details are presented in Algorithm 1.

Input: Component C
Output: Traffic size of Component C per second
1 **if** Process Contains C **then**
2 | **return** Process[C]
3 **else**
4 | P = Component C's RemotePort
5 | PortTraffic = Source-Port[P] + Destination-Port[P]
6 | **return** $\frac{Component[C,P]}{Component[*,P]} \times$ PortTraffic
7 **end**

Algorithm 1: Calculation Algorithm

If a component shares a process with other components, we have to estimate its network traffic following Line 4 to 7. In Line 4, we first find the port P used by component C with the Component table. Then, in Line 5, we calculate the total network traffic of this port P with the Source-Port table and Destination-Port table. In Line 6, Component[*, P] is the total request/response size of all components (* is the wildcard) going through port P. Component[C, P] is the request/response size of Component C going through port P. We use the ratio of Component[C, P] and Component[*, P], and the PortTraffic of P to estimate the TrafficSize of component C. The underlying assumption is that different components suffer from similar packet loss patterns (i.e., similar retry rate) and the packet header size is proportional to the payload size.

4.6 Monitoring

Aside from the dashboard, we also create monitors on the Source-Port table and Destination-Port table in the result database to safeguard the overall traffic usage of EXO. We now support two types of monitors, one to capture the static trend and the other to capture the dynamic changepoints.

We use Mann-Kendall trend test [25] to obtain the static trend and use LinkedIn's Greykite [27] to capture dynamic change points. The static trend monitor could help us find feature roll-out that potentially generates sub-optimal traffic. The dynamic change-points monitor could discover sudden bursts in traffic. These sudden bursts are usually caused by code regression or configuration errors.

5 Production Results

We used NetPanel to support traffic analysis in EXO. In this section, we share the observed traffic characteristics and use case studies to show the effectiveness of NetPanel.

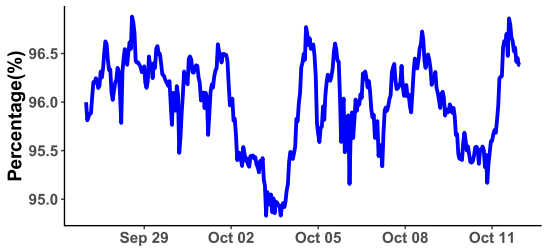


Figure 6: Traffic percentage taken by the top 10 ports. The percentage stays stable during the half-month observation.

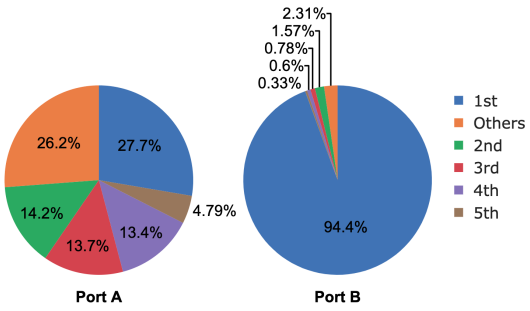


Figure 7: Component-level traffic distribution for the top 2 ports.

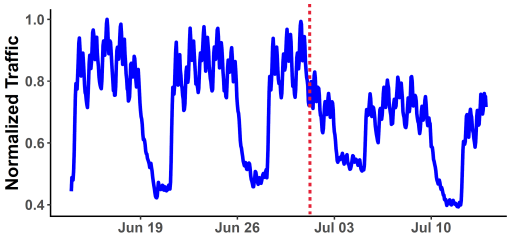


Figure 8: The traffic with destination port A decreased by 20% after switching from external to internal endpoint. The vertical dotted line indicates the recover time.

5.1 Traffic Overview

Recall that there are two tiers of traffic in Sec. 2.1. To use Tier 1 traffic, the owner team must register a specific port. All traffic to or from that port is labeled with a Tier 1 DSCP tag. The registration prevents other components from using the same port. Therefore, all Tier 1 traffic through that port is used exclusively by a single component. In the following, we only discuss the characteristics of Tier 0 traffic.

5.1.1 Several ports dominate the overall traffic

From the Source-Port and Destination-Port tables, we observe that several top ports contribute to most traffic usage. Figure 6 shows the percentage of total traffic generated by the top 10 ports over half a month period. The lowest value during this period is 94.8%. This result confirms that the Top-k ports filter algorithm could cover at least 94% of the total traffic for ETW collector when k = 10. Furthermore, the list of top ports remains unchanged during our observation. This observation supports us to use the latest top k ports obtained as a filter for the ETW collectors as described in Section 4.4. The concentrated traffic distribution at a few top ports saves us a lot of traffic optimization work. We can focus on a small number of ports when we want to reduce network traffic costs. We engage the partner teams that use these ports instead of calling every team in EXO.

5.1.2 Several components dominate the traffic of a top port

When multiple components share the same port (as in Figure 1 REST and EWS share the same port), there are many contributors to the port traffic. We analyze the traffic distribution for top ports. We show the traffic contribution of the top 5 components for the top 2 ports separately in Figure 7, labeled as A and B. For both ports, the contribution of the 6^{th} component is less than 5%. Investigation into lower-ranked components does not have a significant benefit in reducing overall traffic usage. With the help of NetPanel, engaging with a few partner teams is usually enough to optimize the traffic of a top port. For example, when we want to reduce the traffic for Port A, we only need to engage the owners of the top 5 components.

5.2 Case Studies

In this section, we show how we use NetPanel to refine the WAN traffic for EXO. We present 4 cases where we leveraged NetPanel for: (1) service traffic optimization; (2) legacy traffic discovery; (3) anomaly traffic burst detection; and (4) WAN feature validation. These actions save millions of dollars per year for EXO.

5.2.1 Service Traffic Optimization

In this case, we introduce how we use NetPanel to identify sub-optimal traffic and optimize the utilization of network resources. NetPanel provides long-term data visualization. This makes it easy for the application to identify its major traffic contributor. After engaging the owning team, it is easy to make a conclusion on whether the traffic is necessary.

During our investigation of the traffic composition of EXO, we discovered that the Tier 0 traffic from our BE servers to port A on FE servers is too large. Because EXO has optimized the internal service endpoint (URL) to serve requests with the nearest FE server, we expect that there should be little Tier 0 traffic from BE to FE. We then used the Component table to find the main contributors. We identified that a component

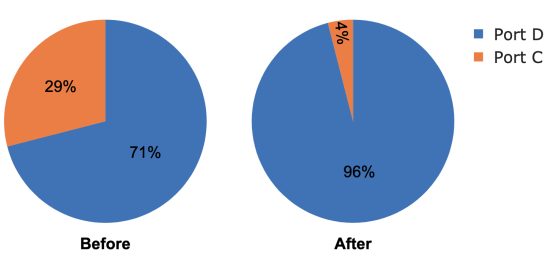


Figure 9: The ratio of the traffic from Port C to the traffic from Port D before and after the fix.

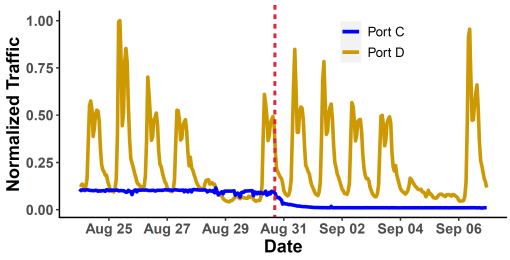


Figure 10: Traffic volume change with source port C and source port D after the removal of legacy code. The vertical dotted line indicates the change time.

that is used to extract plain text content from different formats of documents in emails took up most of the traffic. Then, we contacted the owning team and figured out that the component was using an external service endpoint. It had no control of where the requests were sent to and thus caused massive Tier 0 traffic worldwide. After identifying the issue, we worked together with the owning team to change to use an internal service endpoint so that this component could serve its requests within a single location. This action saved millions of dollars per year. The traffic change is shown in Figure 8. The traffic with destination port A decreased by 20% after the change. Prior to NetPanel, it relied heavily on code reviews to prevent such cases. This manual approach was likely to lead to omissions.

5.2.2 Legacy Traffic Discovery

There is another case where NetPanel helped discover legacy traffic and its source code. As the system grows, the source code becomes overwhelming. The work to remove outdated settings and services is necessary. However, it turns out that it is hard to discover legacies while do not break anything. We figured out there was one component on BE machines that was sending traffic to certificate authorities to validate some certificates that had been retired. We started from the continuous unexpected high traffic volume from port C in the NetPanel Source-Port table. According to our knowledge, EXO had finished the migration from Port C to Port D, so there should be very little traffic from port C, but massive

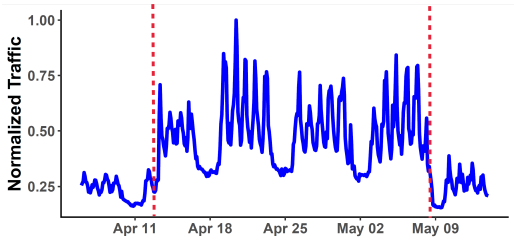


Figure 11: Traffic volume change of the log uploading service during the configuration error. The vertical dotted lines indicate the start time and the end time, respectively.

traffic from port D. The left pie chart in Figure 9 shows the ratio of the traffic from port C to the traffic from port D. The traffic from port C is 42% of the traffic from port D, which is much higher than our expectation. We used NetPanel to find the suspicious component and its owning team. After reaching out to the team, they investigated the issue and removed the legacy code. After the fix, the ratio of the traffic from port C to the traffic from port D became much smaller, where the traffic from port C was reduced to 4.2% of the traffic from port D, as shown in the right pie chart of Figure 9. The traffic of both ports during the fix is shown in Figure 10. After the removal, the traffic with source port C dropped to lower than 10% of the original traffic. That fix led to a savings of inbound capacity in the Gbps for Microsoft.

5.2.3 Anomaly Traffic Burst Detection

This case shows how we leverage NetPanel to detect an anomaly traffic burst caused by a configuration error. Some configuration errors at the application level may cause abnormal network behaviors. For example, a configuration error may cause the component to keep sending requests to an endpoint and thus leads to a traffic burst. These bugs are hard to detect unless the affected machines get so hot and break critical services [28]. In NetPanel, the change in request volume can be detected. When anomalies are reported by NetPanel, we could contact the component owner to debug the issue.

Figure 11 visualizes the traffic change during a configuration error for one component in EXO. An engineer accidentally changed a log uploading compression algorithm to an older version. The decrease in compression rate led to an increase in the total data volume sent to the log center and eventually hit the throttling limit. However, the component had a retry mechanism and kept sending data. This resulted in nearly quadruple the traffic volume sent during peak hours. The error was detected late after the change rolled out worldwide and drove a dramatic increase in traffic volume. We caught this abnormal burst and contacted the owner to dig into their component. After rolling back the change, the abnormal traffic disappeared and the traffic dropped back to the previous level. Without NetPanel, the problem could only be

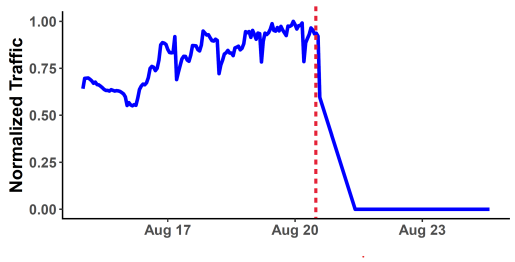


Figure 12: The volume change of Tier 0 traffic sent to registered ports after RWA devices support QoS Policy. The vertical dotted line indicates the change time.

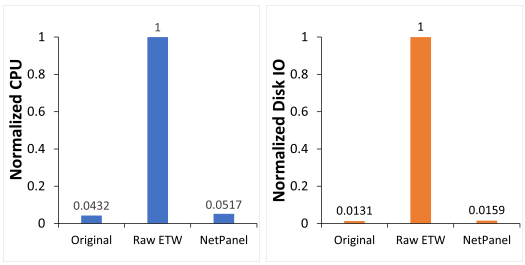


Figure 13: Normalized CPU and Disk IO usage on a BE machine when only services are running, when raw ETW is running, and when NetPanel is running.

discovered until the availability of the affected component is severely impacted, which would take a much longer time to fix the error and waste much more bandwidth.

5.2.4 WAN Feature Validation

This section introduces how NetPanel triggers a feature validation on newly deployed devices. When a new type of device starts to be deployed in the network, it may miss some functionality. Regional WAN Aggregator (RWA) is a kind of core router recently added to the Azure Network that sits on the top of datacenters and is responsible to connect the datacenter network to the backbone WAN. In this case, the newly deployed RWA devices do not support the QoS policy. They ignore the DSCP tag and treat all traffic with Tier 0 priority. NetPanel helped fix the bug at an early stage and avoid purchasing tens of millions of dollars in redundant capacity.

Bandwidth Broker routes traffic with different priorities. Our replication service has registered ports on the Bandwidth Broker service to lower the priority of non-urgent replications to Tier 1. However, from NetPanel data, we found that these ports have a significant amount of traffic classified as Tier 0. Together with the owning team, we picked a machine pair and captured packets on routers along the routing path. We found that the RWA device on the path does not support the QoS policy. Even though only 2 locations had RWA devices deployed at an early stage, this fix already saved several million dollars each year. If RWA devices were deployed worldwide, the value of this fix would be tens of millions of dollars per year. As shown in Figure 12, the Tier 0 traffic for those registered ports dropped to zero after the fix. NetPanel builds a map between components and ports, which makes it easy for component owners to monitor the priority tier of its traffic. Once any bug is introduced on the routing plane, they could discover and fix it quickly.

6 Evaluation

In this section, we evaluate the overhead of NetPanel in two categories: overhead inside and outside the production environment. The production environment serves customer requests and thus has strong restrictions on resource consumption, while the restrictions on overhead outside the production environment are more relaxed.

6.1 Overhead in the Production Environment

Figure 13 shows our evaluation of NetPanel overhead in the production environment. We ran the test on a single server. The overhead of NetPanel in the production environment is introduced by the ETW collector, which has extremely high CPU and disk IO consumption. If we enable ETW fully on machines, it will occupy 25× more CPU and 77× more disk IO compared to those when ETW is disabled. When we deploy NetPanel on production machines, less than 1% rise of the total available CPU and disk IO are observed because of retaining only top ports as mentioned in Section 4.4. It saves 99.1% CPU and 99.7% Disk IO compared with Raw ETW.

6.2 Overhead outside the Production Environment

The overhead outside the production environment includes two parts: data storage and computation. These are the overhead of processing and storing data in the big-data platform.

6.2.1 Data Storage

Table 4 shows the ratio between result data and raw data using our approach. The result size is 0.00361% of the origin for IPFIX, 0.00076% for ETW, and 0.00003% for application logs. The daily result sizes of all three data sources are in GB and are close to each other. The huge difference in the ratios is caused by the huge difference in the size of the original data. ETW data is much larger than IPFIX because it does not perform data sampling and covers Metro traffic (traffic within a location). Application log data is even larger because a machine pair could send multiple requests in one connection. The log data contains many extra columns for debugging.

Data	Size Ratio	Daily Calculation
IPFIX	0.00361%	1.1 hours
ETW	0.00076%	2.5 hours
Application Logs	0.00003%	6.8 hours

Table 4: Result data size ratio (compared to the raw data) and calculation time.

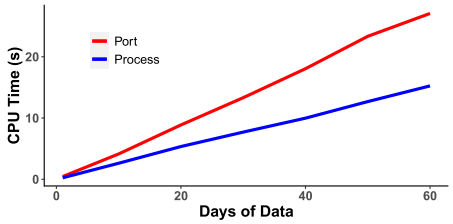


Figure 14: The CPU time for a query on a port and a query on a process increase linearly with the days of data. It takes 27 CPU seconds to get the 60 days of traffic for a port, and 15 CPU seconds to get the 60 days of traffic for a process.

6.2.2 Computation

The computational overhead consists of two parts. The first part is the overhead of background data processing and validation pipelines. The jobs run every day. The second part is the overhead of user queries.

We show the computational overhead for NetPanel's data processing and validation pipelines in Table 4. We use less than 0.01% of our production computation resources to process the data. It takes 1.1 hours to process IPFIX data, 1.6 hours to process ETW data, and 6.8 hours to process application logs every day. The validation pipeline takes around 1.5 hours. The tight resource restriction leads to a long calculation time. We make this trade-off because the goal of NetPanel is to support the reduction of traffic costs in the long run, and therefore a delay of hours is acceptable.

It typically takes no more than 30 seconds for NePanel to respond to a user query for the traffic of a set of specific ports or processes over months. We show the CPU time NetPanel needs to respond to a user query in Figure 14. The CPU time is proportional to data size. The actual waiting time is usually much smaller than the CPU time because multiple CPUs can calculate the result in parallel. The overall overhead of user queries is usually negligible.

7 Related Work

EXO runs on Azure architecture, further details of Azure architecture are shown in [30]. Gunawi [24] provided a summary of 597 cloud outages from 2009 to 2015. Ardelean [17] used Gmail as an example to analyze the performance of cloud applications. They studied the dynamic nature of cloud applications in short time intervals. There are multiple efforts to minimize the cost of operating data centers. Cascara [42] tried to optimize the edge cost. Yang et. al. [50] scheduled the bulk transfer among datacenters. ROTOS [49] designed an optical DCN architecture to improve power efficiency.

Large investments have been made in data-center monitoring. There is a body of work on server monitors for Linux and Windows [8, 11, 36, 43]. Trumpet [36] is a monitor that processes every packet at line rate on end-hosts and tests the presence of user-specified network events. PathDump [43] designs a server stack to retrieve metadata from arrived packets on edge devices to help debug network issues. Monitoring data-center networks has been a hot topic for years. Commonly used protocols include IPFIX/Netflow and sFlow [15, 22, 37]. A detailed review [26] of the general flow monitoring technique is provided. In recent years, researchers have refined flow monitoring for different purposes [33, 40]. To monitor all the flows without sampling, FlowRadar [33] encodes per-flow counters at switches and leverages the computing power at the remote collector to perform decoding. To handle the large-scale challenge of data-centers, many have turned to query-based solutions, including Stroboscope [46], OFRewind [48], and PacketScope [45]. Stroboscope [46] mirrors millisecond-long traffic slices on routers according to user queries to monitor network forwarding behavior including traffic paths, one-way delays, and load-balancing ratios. IBM [29] uses HTTP logs to detect component failure and provide reports over the last 48 hours when an anomaly is detected.

8 Conclusion

The vast amount of network traffic generated by the components in cloud applications consumes significant resources. It is vital to identify the composition of network traffic to reduce the cost. Component-level measurement is needed to drive the traffic optimization effort for systems developed by a large number of teams. NetPanel analyzes the network traffic for EXO at the component level and at a low cost. One primary challenge is caused by the huge amount of measurement data. We design several schemes to reduce the data size without losing fidelity. We discuss real cases where NetPanel is applied to save millions of dollars per year. We believe that the insights we gained during the design and operation of NetPanel provide valuable experience in traffic measurement and reduction of other cloud applications.

Acknowledgements

We thank the anonymous reviewers and our shepherd Srinivas Narayana, for their valuable comments and insightful recommendations to improve the quality of the paper. We thank all Microsoft engineers who have been closely working with us for their great support.

References

[1] Amazon cloud watcher. https://aws.amazon.com/cloudwatch/.

[2] Azure status history. https://status.azure.com/en-us/status/history.

[3] Google cloud monitoring. https://cloud.google.com/monitoring/.

[4] IIS Application Pool. https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2008-r2-and-2008/cc735247(v=ws.10).

[5] Outlook EWS Reference. https://docs.microsoft.com/en-us/exchange/client-developer/exchange-web-services/start-using-web-services-in-exchange.

[6] Outlook MAPI Reference. https://docs.microsoft.com/en-us/office/client-developer/outlook/mapi/outlook-mapi-reference.

[7] Salesforce disruption: Na2 and next day, cs1. http://iwgcr.org/salesforce-disruption/.

[8] TCPDUMP/LIBPCAP public repository. https://www.tcpdump.org/.

[9] Update on today's gmail outage. https://gmail.googleblog.com/2009/02/update-on-todays-gmail-outage.html.

[10] Azure Data Explorer: a big data analytics cloud platform optimized for interactive, adhoc queries over structured, semi-structured and unstructured data, 2018. https://azure.microsoft.com/mediahandler/files/resourcefiles/azure-data-explorer/Azure_Data_Explorer_white_paper.pdf.

[11] Event Tracing for Windows, 2018. https://docs.microsoft.com/en-us/windows/win32/etw/about-event-tracing.

[12] Aws outage analysis: December 15, 2021, 2021. https://https://www.thousandeyes.com/blog/aws-outage-analysis-december-15-2021/.

[13] Service Level Agreements (SLA) for Online Services, 2021. https://www.microsoft.com/licensing/docs/view/Service-Level-Agreements-SLA-for-Online-Services.

[14] us-central1: Google app engine standard experiencing increased latency and pending queue aborted error request rate., 2021. https://status.cloud.google.com/incidents/uaRinwS8pu2yyjdYbDaM.

[15] Paul Aitken, Benoît Claise, and Brian Trammell. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. RFC 7011, September 2013.

[16] Ali Anwar, Anca Sailer, Andrzej Kochut, and Ali R. Butt. Anatomy of cloud monitoring and metering: A case study and open problems. In Proceedings of the 6th Asia-Pacific Workshop on Systems, APSys '15, New York, NY, USA, 2015. Association for Computing Machinery.

[17] Dan Ardelean, Amer Diwan, and Chandra Erdman. Performance analysis of cloud applications. In 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), pages 405–417, Renton, WA, April 2018. USENIX Association.

[18] Fred Baker, David L. Black, Kathleen Nichols, and Steven L. Blake. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. RFC 2474, December 1998.

[19] Francesco Bronzino, Paul Schmitt, Sara Ayoubi, Hyojoon Kim, Renata Teixeira, and Nick Feamster. Traffic refinery: Cost-aware data representation for machine learning on network traffic. Proc. ACM Meas. Anal. Comput. Syst., 5(3), dec 2021.

[20] Matt Calder, Manuel Schroder, Ryan Gao, Ryan Stewart, Jitu Padhye, Ratul Mahajan, Ganesh Ananthanarayanan, and Ethan Katz-Bassett. Odin: Microsoft's scalable fault-tolerant cdn measurement system. In USENIX NSDI, April 2018.

[21] B. Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954, October 2004.

[22] Luca Deri, Ellie Chou, Zach Cherian, Kedar Karmarkar, and Mike Patterson. Increasing data center network visibility with cisco netflow-lite. In Proceedings of the 7th International Conference on Network and Services Management, CNSM '11, page 274–279, Laxenburg, AUT, 2011. International Federation for Information Processing.

[23] Roy Thomas Fielding. Architectural styles and the design of network-based software architectures. University of California, Irvine, 2000.

[24] Haryadi S. Gunawi, Mingzhe Hao, Riza O. Suminto, Agung Laksono, Anang D. Satria, Jeffry Adityatama, and Kurnia J. Eliazar. Why does the cloud stop computing? lessons from hundreds of service outages. In Proceedings of the Seventh ACM Symposium on Cloud Computing, SoCC '16, page 1–16, New York, NY, USA, 2016. Association for Computing Machinery.

[25] Khaled H. Hamed and A. Ramachandra Rao. A modified mann-kendall trend test for autocorrelated data. Journal of Hydrology, 204(1):182–196, 1998.

[26] Rick Hofstede, Pavel Čeleda, Brian Trammell, Idilio Drago, Ramin Sadre, Anna Sperotto, and Aiko Pras. Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. IEEE Communications Surveys Tutorials, 16(4):2037–2064, 2014.

[27] Reza Hosseini, Albert Chen, Kaixu Yang, Sayan Patra, and Rachit Arora. Greykite: a flexible, intuitive and fast forecasting library, 2021.

[28] Ryan Huang, Chuanxiong Guo, Lidong Zhou, Jay Lorch, Yingnong Dang, Murali Chintalapati, and Randolph Yao. Gray failure: The achilles' heel of cloud-scale systems. In Proceedings of the ACM Workshop on Hot Topics in Operating Systems (HotOS). ACM, May 2017.

[29] Mohammad S. Islam, William Pourmajidi, Lei Zhang, John Steinbacher, Tony Erwin, and Andriy Miranskyy. Anomaly detection in a large-scale cloud platform. In 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), pages 150–159, 2021.

[30] Karthick Jayaraman, Nikolaj Bjørner, Jitu Padhye, Amar Agrawal, Ashish Bhargava, Paul-Andre C Bissonnette, Shane Foster, Andrew Helwer, Mark Kasten, Ivan Lee, Anup Namdhari, Haseeb Niaz, Aniruddha Parkhi, Hanukumar Pinnamraju, Adrian Power, Neha Milind Raje, and Parag Sharma. Validating datacenters at scale. In Proceedings of the ACM Special Interest Group on Data Communication, SIGCOMM '19, page 200–213, New York, NY, USA, 2019. Association for Computing Machinery.

[31] Peter Kromkowski, Shaoran Li, Wenxi Zhao, Brendan Abraham, Austin Osborne, and Donald E. Brown. Evaluating statistical models for network traffic anomaly detection. In 2019 Systems and Information Engineering Design Symposium (SIEDS), pages 1–6, 2019.

[32] Alok Kumar, Sushant Jain, Uday Naik, Anand Raghuraman, Nikhil Kasinadhuni, Enrique Cauich Zermeno, C. Stephen Gunn, Jing Ai, Björn Carlin, Mihai Amarandei-Stavila, Mathieu Robin, Aspi Siganporia, Stephen Stuart, and Amin Vahdat. BwE: Flexible, Hierarchical Bandwidth Allocation for WAN Distributed Computing. In Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM '15, page 1–14, New York, NY, USA, 2015. Association for Computing Machinery.

[33] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. FlowRadar: A better NetFlow for data centers. In 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16), pages 311–324, Santa Clara, CA, March 2016. USENIX Association.

[34] Ze Li, Qian Cheng, Ken Hsieh, Yingnong Dang, Peng Huang, Pankaj Singh, Xinsheng Yang, Qingwei Lin, Youjiang Wu, Sebastien Levy, and Murali Chintalapati. Gandalf: An intelligent, End-To-End analytics service for safe deployment in Large-Scale cloud infrastructure. In 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20), 2020.

[35] 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, SIGCOMM '16, page 101–114, New York, NY, USA, 2016. Association for Computing Machinery.

[36] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. Trumpet: Timely and precise triggers in data centers. In Proceedings of the 2016 ACM SIGCOMM Conference, SIGCOMM '16, page 129–143, New York, NY, USA, 2016. Association for Computing Machinery.

[37] Sonia Panchen, Neil McKee, and Peter Phaal. InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks. RFC 3176, September 2001.

[38] Hiren Patel, Alekh Jindal, and Clemens Szyperski. Big data processing at microsoft: Hyper scale, massive complexity, and minimal cost. In Symposium on Cloud Computing, page 490. ACM, November 2019.

[39] William Pourmajidi, Andriy Miranskyy, John Steinbacher, Tony Erwin, and David Godwin. Dogfooding: Using ibm cloud services to monitor ibm cloud infrastructure. In Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering, CASCON '19, page 344–353, USA, 2019. IBM Corp.

[40] Jeff Rasley, Brent Stephens, Colin Dixon, Eric Rozner, Wes Felter, Kanak Agarwal, John Carter, and Rodrigo Fonseca. Planck: Millisecond-scale monitoring and control for commodity networks. In Proceedings of the 2014 ACM Conference on SIGCOMM, SIGCOMM '14, page 407–418, New York, NY, USA, 2014. Association for Computing Machinery.

[41] Vyas Sekar, Michael K. Reiter, and Hui Zhang. Revisiting the case for a minimalist approach for network flow monitoring. In Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement, IMC '10, page 328–341, New York, NY, USA, 2010. Association for Computing Machinery.

[42] Rachee Singh, Sharad Agarwal, Matt Calder, and Paramvir Bahl. Cost-effective cloud edge traffic engineering with cascara. In 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21), pages 201–216. USENIX Association, April 2021.

[43] Praveen Tammana, Rachit Agarwal, and Myungjin Lee. Simplifying datacenter network debugging with pathdump. In Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation, OSDI'16, page 233–248, USA, 2016. USENIX Association.

[44] Praveen Tammana, Rachit Agarwal, and Myungjin Lee. Distributed network monitoring and debugging with SwitchPointer. In 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), pages 453–456, Renton, WA, April 2018. USENIX Association.

[45] Ross Teixeira, Rob Harrison, Arpit Gupta, and Jennifer Rexford. Packetscope: Monitoring the packet lifecycle inside a switch. In Proceedings of the Symposium on SDN Research, SOSR '20, page 76–82, New York, NY, USA, 2020. Association for Computing Machinery.

[46] Olivier Tilmans, Tobias Bühler, Ingmar Poese, Stefano Vissicchio, and Laurent Vanbever. Stroboscope: Declarative network monitoring on a budget. In 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), pages 467–482, Renton, WA, April 2018. USENIX Association.

[47] Jianwen Wei, Yusu Zhao, Kaida Jiang, Rui Xie, and Yaohui Jin. Analysis farm: A cloud-based scalable aggregation and query platform for network log analysis. In 2011 International Conference on Cloud and Service Computing, pages 354–359, 2011.

[48] Andreas Wundsam, Dan Levin, Srini Seetharaman, and Anja Feldmann. Ofrewind: Enabling record and replay troubleshooting for networks. In 2011 USENIX Annual Technical Conference (USENIX ATC 11), Portland, OR, June 2011. USENIX Association.

[49] Xuwei Xue, Fulong Yan, Kristif Prifti, Fu Wang, Bitao Pan, Xiaotao Guo, Shaojuan Zhang, and Nicola Calabretta. Rotos: A reconfigurable and cost-effective architecture for high-performance optical data center networks. Journal of Lightwave Technology, 38(13):3485–3494, 2020.

[50] Zhenjie Yang, Yong Cui, Xin Wang, Yadong Liu, Minming Li, Shihan Xiao, and Chuming Li. Cost-efficient scheduling of bulk transfers in inter-datacenter wans. IEEE/ACM Transactions on Networking, 27(5):1973–1986, 2019.

[51] Jialu Zhang, Ruzica Piskac, Ennan Zhai, and Tianyin Xu. Static Detection of Silent Misconfigurations with Deep Interaction Analysis. In Proceedings of the 36th ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'21), October 2021.

[52] Lixia Zhang, Van Jacobson, and Kathleen Nichols. A Two-bit Differentiated Services Architecture for the Internet. RFC 2638, July 1999.

[53] Yuanliang Zhang, Haochen He, Owolabi Legunsen, Shanshan Li, Wei Dong, and Tianyin Xu. An Evolutionary Study of Configuration Design and Implementation in Cloud Systems. In Proceedings of the 43rd International Conference on Software Engineering (ICSE'21), May 2021.

[54] Yikai Zhao, Kaicheng Yang, Zirui Liu, Tong Yang, Li Chen, Shiyi Liu, Naiqian Zheng, Ruixin Wang, Hanbo Wu, Yi Wang, and Nicholas Zhang. LightGuardian: A Full-Visibility, lightweight, in-band telemetry system using sketchlets. In 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21), pages 991–1010. USENIX Association, April 2021.

[55] Tanja Zseby, Benoît Claise, Juergen Quittek, and Sebastian Zander. Requirements for IP Flow Information Export (IPFIX). RFC 3917, October 2004.