VIRTUAL EXPERIENCE
OCTOBER 11-14

2021 Fall
Technical Forum
SCTE® • NCTA • CABLELABS®

Reducing the cost of network traffic monitoring with AI

Petar Djukic
Director AI & Analytics
Ciena Canada
Ottawa ON, Canada
pdjukic@ciena.com

Maryam Amiri
Lead AI Engineer
Ciena Canada
Ottawa ON, Canada
maamiri@ciena.com

Wade Cherrington
Software Engineer
Ciena Canada
Ottawa ON, Canada
wcherrin@ciena.com

Table of Contents

List of Figures

List of Tables

1. Introduction

This paper describes three visionary approaches to reduce the amount of network telemetry. The problem we discuss is that internet protocol (IP) network monitoring requires collection and storage of a large volume of data. This is a big issue for service providers (SPs), but the solution is usually not thought of outside of conventional approaches. With the advent of artificial intelligence (AI) approaches, especially in estimation, interpolation and imputation, new solution avenues are becoming available.

IP network monitoring often implies the use of NetFlow or Internet Protocol Flow Information Export (IPFIX). However, as we show shortly the two alone are not enough to cover all use cases. NetFlow and IPFIX can be used to turn the network into a large collection of sensors collecting information about IP traffic, which can be used to monitor network usage, identify misconfigured network elements, identify compromised network end points and detect network attacks (Santos, 2016). However, high fidelity network monitoring with technologies such as IPFIX comes with many challenges due to the amount of data that is collected by network elements, then transmitted to where it can be stored and finally processed to get the insights that the operator is looking for. IPFIX also only gives a partial view of the network state and additional technologies are needed to build a complete view of the network.

1.1. Overview of IP Network Monitoring

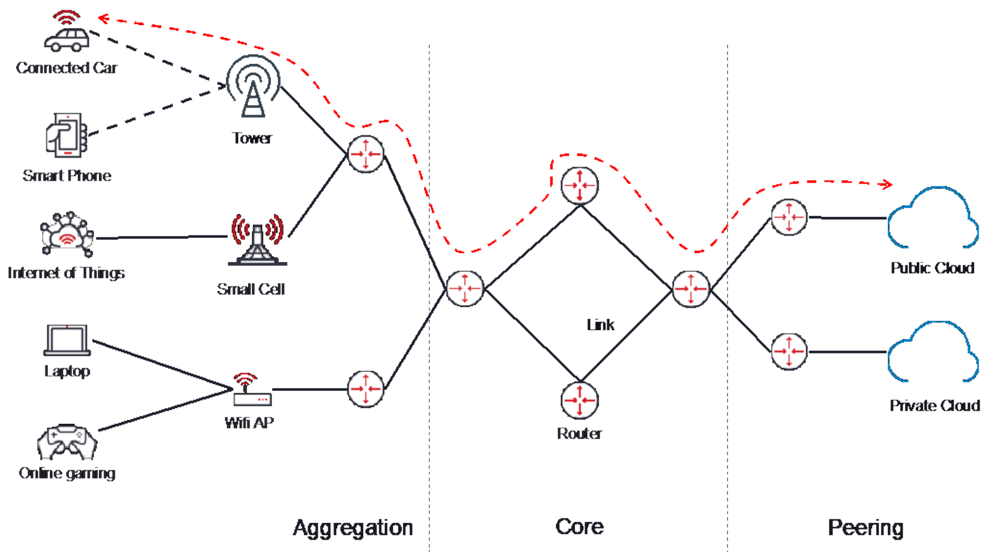


Figure 1 - IP Network

Figure 1 shows a simplified monitored IP network architecture. An IP flow is shown in red and refers to a distinct set of packets occurring in the same period that share a 5-tuple IP header (destination IP address, destination port, source IP address, source port and type-of-service). Aggregate flows, which combine traffic between points in the network may be used in core networks where there are too many flows to track individually. Flows traverse many network elements, but as the focus of this paper is the IP domain, routers and links are the important part of the figure. Routers and links are both physical elements, however they can have appropriate "digital twins" in network monitoring/inventory software, which is used to create a logical network topology.

There two types of network monitoring: active and passive (Network monitoring, n.d.). In this paper, we are considering passive approaches, which are based on measuring network information without disturbing existing network traffic. Passive IP monitoring can be done at an IP flow level, link level, network level or router level. Active approaches include injecting network packets to probe the network. Some of the approaches for network probing are MTR (Wikipedia, n.d.), based on Internet Control Message Protocol (ICMP) (Wikipedia, n.d.), Iperf (Wikipedia, n.d.), based on a proprietary packet generation and application level protocols, or others, based on RFC 2544 (Wikipedia, n.d.). Active network monitoring can benefit from the applications of AI shown here, but due to space and time limitations these are not discussed.

The goal of IP network monitoring is to get an accurate enough picture of the network state, so that network operators can implement their operational use cases (Quittek, Zseby, Claise, & Zander, 2004). IP network monitoring addresses many use cases:

- *Usage-based accounting* is a business model for selling IP services. A user or a user group is charged based on how much traffic was transmitted. For example, this traffic usage model is used by Amazon Web Services (AWS) to charge for transmitting data out of the cloud. To enable it, very accurate packet counts for the user are required at an ingress/egress point.
- *Traffic profiling* uses information collected about an IP flow to describe it succinctly so that its statistical profile can be used inside of a traffic model. The model of the IP flow can be used in network planning or network dimensioning. For example, the profile may have the average or peak traffic volume and does not require very precise measurements. Depending on how traffic profiling is used flows may be individual or aggregated.
- *Traffic engineering* uses the information collected about IP flows to control the network with the goal of optimizing network resources and traffic performance. Typical measurements are link utilization, traffic volumes between network nodes and routing information. While precise information may be required about flow routing, the information about link utilization and traffic volumes can be approximate. For example, it may be sufficient to know the largest flow on a "hot" link to initiate its route transfer to another part of the network.
- *Attack/Intrusion detection* uses the information about IP flows to detect unusual situations and suspicious flows and then monitors attacking flows. This use case may require stateful packet flow analysis, which requires deep packet inspection. However, to analyse traffic for anomalies all that may be required is a statistical model of the traffic, which can be learned with machine learning.
- *QoS monitoring* uses quality measurements of IP flows to validate QoS parameters negotiated during service level specification (packet loss, latency). QoS monitoring requires correlation of data from multiple observation points, which requires proper clock synchronization. As QoS monitoring is often specified in statistical terms, precisely

collected information is not required to enforce SLAs, rather it is important to build up a statistical profile of the observed QoS experienced by flows.

Each use case has its own requirements on the veracity and type of data required, meaning that there may be multiple protocols and software required to collect the information for any one of the use cases. We note that while some require precise collection and storage of data (usage-based accounting), others may only require precise data collection, while the stored data does not have to be prices (attack/intrusion detection), and some do not require precise collection or storage of data (traffic profiling, traffic engineering, QoS monitoring).

2. Data Collection for IP Network Monitoring

IP network observation is done by collecting information about the infrastructure (routers and their physical connections), information about IP flows traversing the infrastructure and information about how IP packets are routed through the network (routes). This paper implicitly talks about the network observations of the core network (shown in the middle of Figure 1), but the content of this paper also applies to the aggregation and peering parts of the network.

There is no one standardized method to collect all the information required for each of the network monitoring use cases. Typically, a combination of network monitoring protocols is used, and the information is collected by a software tool, provided by vendor specializing in this. Most networks are multi-vendor networks, presenting a commercial and technical challenges to equipment vendors to collect network information from other vendors' equipment. A third-party vendor would resolve those challenges and provide a data collection software. Collected information can be grouped into device, routing, link and flow information. The monitoring software tool correlates the information and presents it in a "single pane of glass". Users can then get subsets of data required for their use case.

2.1. Router information

Router information can be obtained with direct queries using their command-line interface (CLI) (Wikipedia, n.d.). CLI commands typically provide network element information (infrastructure information), which is not otherwise available through other means. For example, configuration, CPU utilization, or debug logs can be obtained in this way. Other information such as routing tables and link utilization are also available, however routers are typically not optimized to access this information through their CLI interfaces and that should be avoided. CLIs can change at the whim of the network device vendor, making it difficult to track changes for a 3rd party vendor providing the monitoring solution.

While not strictly required for IP flow monitoring, some of the information collected directly from the routers can be used for root cause analysis (RCA) of undesirable network behaviours. Particularly, system logs may be useful for this purpose.

2.1.1. Route information

IP routing information can be collected with routing protocol sniffing. Interior gateway protocols (IGPs) such as Open Shortest Path First (OSPF) (Wikipedia, n.d.) and Intermediate System to

Intermediate System (IS-IS) (Wikipedia, n.d.) are used by routers to exchange topology information in an autonomous system (AS), which enables calculation of forwarding tables on routers. A monitoring system sniffs the inter-router traffic for route update packets and builds the known network topology as a router would.

Using the sniffed network topology and shortest path routing, the monitoring system can in real-time build the routing matrix for the network, which is a "digital twin" representation of the network forwarding tables. The routing matrix relates flows to the paths they traverse in the network. Note that since IP uses dynamically changing hop-by-hop routing, the routing matrix is a delayed version of the real-time routing matrix and is only a close approximation of how packets traverse the network.

2.1.2. Link Information

Link information includes volume of packets carried on a link and link utilization. This information can be obtained by subscribing or polling Simple Network Management Protocol (SNMP) (Wikipedia, n.d.) or Network Configuration Protocol (NETCONF) (Wikipedia, n.d.) router's management interfaces. SNMP is an older protocol with many limitations. For example, it has a polling interval of 5-10 minutes, while NETCONF can be configured to stream messages, which are sent on change detected by the router. NETCONF can use the Yet Another Next Generation (YANG) (Wikipedia, n.d.) format to collect information from the network devices. Internet Engineering Task Force (IETF) is actively working on defining new data sources and formatting of their data across router equipment (IETF).

2.1.3. Flow information

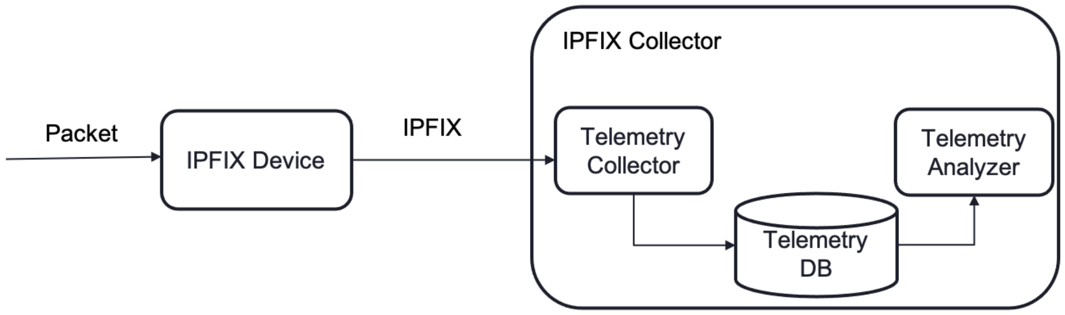


Figure 2 IPFIX Measurement Architecture

Flow information can be collected with NetFlow or IPFIX (Wikipedia), which are equivalent in functionality. IPFIX is the standardized method of doing it and we will limit our discussion to it, keeping in mind that problems with IPFIX also exist in NetFlow. Figure 2 illustrates key components of the IPFIX traffic measurement architecture. IPFIX device and IPFIX collector are two major components of the protocol.

- An IPFIX device is typically a part of routers or switches. It reports information about flows. A dedicated IPFIX device could also be installed to capture packets from the fiber tap or the mirrored port at a switch. The IPFIX device could be hardware based, or virtualized, so it could also be software installed on a datacenter server.

- The IPFIX collector gathers and analyzes IPFIX flows from multiple IPFIX devices through reliable transport protocols. A typical IPFIX telemetry data record consists of 5-tuple of IP/TCP/UDP header fields, the number of bytes, the number of packets, the flow start time, and the flow end time. In IPFIX, communication between the IPFIX device and the flow collector is done through reliable transport protocols such as Stream Control Transport Protocol (SCTP) or TCP2.

As we just showed, IPFIX way of collecting flow information requires installation of a specialized monitoring system with intermediate collectors, large volume data storage and an enormous number of computational resources to analyze the collected data. The data volume problems start at the network element where the software and hardware are typically not able to track all flows passing the element. Still, the aggregate volume of information may be such that the multiple collectors, distributed across the network, may have to be used. Finally, the volume of collected data makes it impractical to keep IPFIX collected data for long periods of time.

Due to hardware limitations, the monitoring system is usually unable to track a majority of the traffic. One practical approach to mitigate the collection overhead in IPFIX is a technique called threshold compression. In this technique the router reports only flows above a threshold to the collection station. The main disadvantage of this method is that the information of flows below the threshold are not sent. It is quite possible that up to 90% of the flows don’t cross the threshold and are not reported.

An alternative method of measuring traffic is use direct counters on traffic tunnels (aggregated flows). However, here we focus on standardized solutions such as IPFIX.

2.2. Overview of the paper

The rest of the paper is organized as follows. We start with a short description of how AI is implemented using deep neural networks (DNNs). Then we describe three ways to use DNNs to decrease the volume of telemetry and stored data, while keeping the fidelity of the data above what is required by traffic monitoring use cases. These are solutions to the problems with IPFIX that we just outlined.

Throughout the paper we cite Wikipedia and AI blogs for various AI concepts. While this may appear to not be the most scientifically sound, we found these articles easy to follow and they always link to the more complete computer science papers for the keen reader. There is much DNN jargon used in the paper. We introduce DNN-specific terms in quotes “” to emphasize their jargon origin.

3. Foundations of AI technologies

This section is intended as a general overview of AI technologies. Readers familiar with concepts of deep neural networks (DNNs) and machine learning (ML) can skip it. More information about DNNs can be found in (Goodfellow, Bengio, & Courville, 2016).

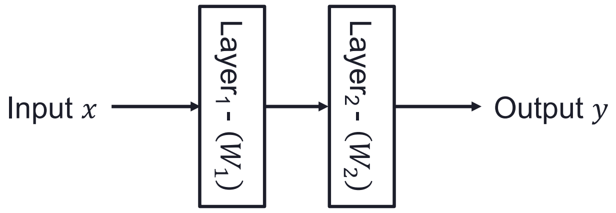
AI technologies are based on the use deep neural networks for machine learning. Machine learning is a computer science concept in which functional software blocks are created by showing the computer examples of correct outputs from inputs, instead of explicitly writing functions that instruct the computer on how to produce outputs from inputs in structured programming (Wikipedia, n.d.). Instead of coding the algorithm, a generic machine learning algorithm is "trained" with examples of what the correct outputs are for given inputs. In recent years, DNN technology has elevated machine learning to the level of human capability in some cognitive tasks. For example, it is now possible to train a DNN-based machine learning algorithm to read a paragraph of text and answer questions about more accurately than humans, or to categorize X-ray images better than a radiologist (Zhang, et al.).

DNN technology is based on basic linear algebra components – matrix multiplication and addition and basic calculus – derivatives. Most of the knowledge required for DNNs has been around for hundreds of years since the time of Carl Friedrich Gauss (Wikipedia, n.d.) and Issac Newton (Wikipedia, n.d.). What is new at this time is that the advances in parallel computing have made it possible to deal effectively with large matrices and train very large DNNs. The most common computing platform are the Graphic Processing Units (GPUs) (Wikipedia, n.d.), which can be used even beyond DNNs, and they are now being complemented with Tensor Processing Units (TPUs) (Wikipedia, n.d.), which are specialized computing units for DNNs. AI accelerators such as TPUs are now found almost anywhere from being embedded in laptops, cellphones, and dedicated data center servers.

For completeness and interest of the reader we now give a simplified overview if how DNNs make predictions and how they are trained.

3.1. How DNNs make prediction

A DNN is a set of algebraic equations that describes how outputs are determined from inputs using matrix operations. A graphical version of the DNN representation is shown Figure 3a, which shows the most basic type of building block for DNNs, known as "dense blocks". The 2-layer DNN shown in the figure is shallow. A typical may have dozens of layers (blocks).



a) Graphical Description of a 2-layer neural network

$$y = \max(0, W_2 \times \max(0, W_1 \times x + b_1) + b_2)$$

b) Mathematical Description of the 2-layer neural network

Figure 3 An example 2-layer DNN

The network in Figure 3a evaluates an equation involving linear algebra shown in Figure 3b. In this example equation, $W_1 \times x$ is a matrix multiplication (Wikipedia, n.d.) and the max function ensures that the result of all operations is positive. Terms b_1 and b_2 are called bias for the layer. The input to the network is x, while the output is y, so the equation describes the functional blocks used in the DNN. The output y is also called a prediction. The input x is a mathematical vector whose components is called a "features". Each feature is a separate input variable contributing to the output of the DNN.

The simple set of algebraic transformations in this example is very powerful as it can be shown mathematically that a neural network with enough layers – depth – can approximate any function. For this reason, DNNs are known as "universal approximators" (Hanin & Sellke, 2018).

A pictorial description of a DNN shown in Figure 3a can be translated into the above equation in Figure 3b by an AI engineer and then made into a software program that performs the set of algebraic equations. In practice, the software piece is simple to write using libraries such as TensorFlow (Abadi, et al., 2015) and PyTorch (Paszke, et al., 2019). The DNN can also be exported into the Open Neural Network Exchange (ONNX) (Open Neural Network Exchange, n.d.), which describes the equations and can be loaded into many DNN software frameworks.

3.2. How DNNs learn

So far, we have described the prediction or inference part of a DNN. If we know the weights of the DNN (e.g. matrices W_1 and W_2 in Figure 3) then upon receiving the inputs, the set of matrix calculations described by the DNN is performed to determine the outputs. The outputs of the DNN are called the "predictions". This process of making prediction is sometimes called "inference". Weights are determined during a process of training.

For example, if we have the function

$$y = 2x^2 + 3x,$$

we can generate a dataset of training samples shown in Table 1 in the two left-most columns. With the dataset, we can use a training function provided in open-source software such as TensorFlow (Abadi, et al., 2015) to determine a set of matrices W_1 and W_2 that result in the best approximation of the function. The training function is called "fit" as it fits the weights of the DNN to the dataset during the training. The fitting function minimizes the error of the predictions $\hat{y}$ for the dataset compared to actual values in the dataset y. The error can be measured with the Mean Absolute Percentage Error (MAPE) shown in the right-most column of Table 1.

Table 1 Example training dataset for $y = 2x^2 + 3x$

Input x	Actual output y	Predicted output $\hat{y}$	MAPE $\frac{\|\hat{y}-y\|}{y}$
1	5	4.5	10 %
2	14	15.6	11.5 %
3	27	25.4	5.9 %

The great DNN research achievement in recent years has been to devise an efficient training procedure that finds the set of internal weights W to minimize the error of the DNN. The training procedure uses "stochastic optimization" who's understanding essentially requires a PhD in mathematics or computer sciences. However, this understanding is not necessary to use DNNs as almost anyone who understands software development can write approximately 10 lines of code to create the DNN and train the function.

The training procedure is iterative and takes in a set of examples of inputs and outputs in batches. For each batch, the training procedure takes in the inputs and generates predictions using the current weights. The predictions from the DNN are compared with the known outputs to find the error in the predictions (the "loss" function) and this error is used to calculate the adjustment to the current weights. The adjustment is usually done using a gradient descent that takes a "learning rate" as an input and calculates the error of the predictions from the weights and number of predictions. The learning rate determines how quickly the descent happens and how closely to the best fit the training gets. The gradient of the whole DNN is calculated using "backpropagation" (Wikipedia, n.d.), which is an algorithm applied backwards through the DNN to differentiate it. Backpropagation is an example of automatic differentiation using the chain rule (Wikipedia, n.d.).

3.3. DNN Models

A DNN model is a trained DNN. A single DNN may have multiple models for different versions of the training data, or different versions of the training algorithms. Each version may have the same structure (number of matrices, size of matrices, and flow through the matrices), but the values in the internal matrices may be different. In a parallel to software development, DNN models have different versions, which presumably improve with higher version numbers. Unlike software, a DNN model is not guaranteed to work well over time, as the inputs may have significant changes in their statistical properties. An example would be traffic demands changing if a new data center peering point is added to the network.

Automating re-training of DNNs due to changes in input data is beyond the scope of this paper.

4. Reducing network monitoring data with AI

Now, we take a deep dive into three approaches for reducing network monitoring data volumes collected with IPFIX. Currently, this problem is resolved in unsatisfying ways. For example, IPFIX may not be active in the network at all, so end-to-end flow information is not collected. If IPFIX is turned on, it may not monitor all the flows – to reduce the pressure on the routers, some flows are simply reported. The collected data is often stored temporarily, or if it stored for longer time, it is reduced by averaging across days, weeks, or months.

We propose three new approaches to deal with the volume of data in the context of IPFIX. First, we consider how to reduce the volume of IPFIX measurements by using link volume information, without a substantial loss of end-to-end flow data quality. Second, we consider how to actively reduce the volume of IPFIX telemetry by taking advantage of correlations in the data. Third, we show how to use AI to compress stored network measurements without major loss of fidelity.

4.1. Reducing the amount of collected information

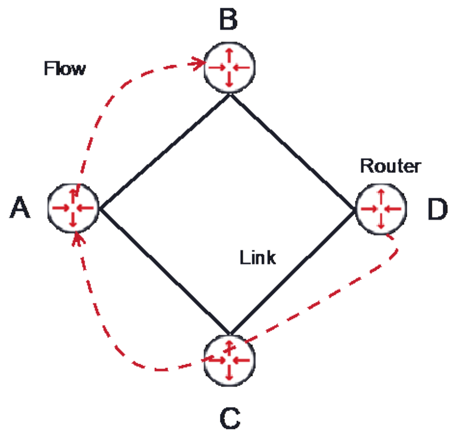


Figure 4 Relationship between links and end-to-end flows

Traffic profiling and traffic engineering are two major use cases for IP network monitoring. Each requires an estimate of the Origin-Destination (OD) traffic matrix, which describes the amount of traffic between each OD pair in the network. Figure 4 shows the relationship between OD pairs (end-to-end flows) and links. In the figure we have 4 routers A, B, C, D and 4 links AB, AC, BD and CD. In case of a core network, we would want to know the aggregate traffic between each OD pair. There are 12 relevant OD pairs, for example for router A there are AB, AC, AD. The premise of the idea in this section is that if we can deduce OD flows from link measurements, we can limit data collection to links. In the case of Figure 4, the amount of reduction would be 75 % (instead of collecting information on 12 OD pairs, we can collect information on 4 links).

Currently, IPFIX is a go to method to obtain OD pair information. IPFIX samples packets transiting through a given router and infers their origin and destination from packet headers. We now show that finding the entire traffic matrix can also be done by utilizing the available link counts and routing information.

Obtaining end-to-end traffic volumes from link measurements is a mathematical problem, which requires a matrix inversion. The instantaneous traffic matrix can be related to link measurements and the routing matrix with

$$\boldsymbol{y} \approx R\boldsymbol{x}$$

where $\boldsymbol{y}$ is the vector of measured link loads over links in the network, R is the routing matrix, and $\boldsymbol{x}$ is OD traffic matrix with one row corresponding to the demand of each OD pair. A flow in the matrix is denoted with row $x_i \in \boldsymbol{x}$ in the OD matrix x. The routing matrix is structured in a way that the link measurements $\boldsymbol{y}$ correspond to the sum of OD flows that traverse the link. Due to packet latencies and the random nature of OD pair traffic, the equation is approximate.

For the mathematically inclined, it may be obvious that the instantaneous traffic matrix can be estimated with

$$\boldsymbol{x} \approx R^{-1}\boldsymbol{y}$$

where R^{-1} is the "inverse" of the routing matrix. Alas, the routing matrix undetermined and is typically not invertible, so this solution is not possible.

One way to solve the undetermined equation is to find the OD traffic matrix $\widehat{\boldsymbol{x}}$, which minimizes a distance between the true end-to-end flow measurements and their estimate:

$$\widehat{\boldsymbol{x}} = \underset{\boldsymbol{x}}{\operatorname{argmin}} \|R\boldsymbol{x} - \boldsymbol{y}\|.$$

which is a problem solvable using AI technology. The equation reads: find an approximation of end-to-end flow volume, $\widehat{\boldsymbol{x}}$, which has the smallest error, compared to the true matrix $\boldsymbol{x}$. If we have many link measurements taken during different times of the day, the estimate becomes the maximum likelihood estimate of the end-to-end flows.

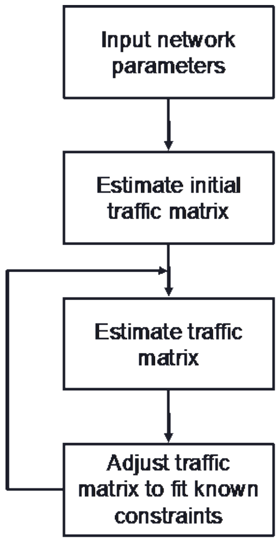

Input network parameters
Estimate initial traffic matrix
Estimate traffic matrix
Adjust traffic matrix to fit known constraints

Figure 5 Iterative estimation of the traffic matrix

The optimization approach alone does not provide the best results. We have found it necessary to also combine that approach with stochastic approaches to deal with the randomness in network measurements. The details of how that is done go beyond the scope of this paper due to the volume of mathematics involved.

The approach used above ignores many of the known network constraints since it is defined as an unconstrained optimization. To get around this, the AI method can be incorporated in an iterative procedure shown in Figure 5. The method takes an initial traffic matrix estimate and then uses an estimation procedure followed by an adjustment procedure. As the procedure goes on, it produces a sequence of the traffic matrix estimates $\hat{\boldsymbol{x}}_0, \ldots, \hat{\boldsymbol{x}}_n$, each of which is expected to be closer to the true traffic matrix $\boldsymbol{x}$. As the initial traffic matrix estimate and the estimate in the iterative step may produce a traffic matrix which may not match information known about the traffic (e.g. ingress/egress aggregate counts), an adjustment procedure that projects the estimate into the known constraints is used to fix this.

We evaluate the performance our methodology using real traffic traces from a backbone network. Our source of data is the IP-level traffic flow measurements collected form every point of presence (PoP) in the Abilene Internet2 back bone network (Roughan). Abilene is the major backbone network, connecting over 200 universities in the US, and peering with other research networks in Europe and Asia. At the time the data was collected, the Abilene network had 11 routers resulting in 121 origin–destination flows; there were 15 links in the network. We note that Abilene was collecting OD pair information using flow sampling technology and it was also simultaneously collecting link information, which allows us to estimate OD flows from link measurements and then use true flow measurements to evaluate the performance of the AI-based approach.

Table 2 summarizes the results we obtained on the Abilene dataset. We make several observations. First, the AI approach reduces the error in the estimate significantly (by 42%).

Second, the estimates are now in the range of what is acceptable for the traffic engineering task. Third, the amount of data collection has been reduced by 78%, as we only need packet counters from links.

Table 2 Summary of OD flow estimation from link data performance results

	MAPE	Reduction in MAPE
Non-AI approach	29.6%	-
AI Approach	17 %	42%

We think that this may be a promising approach to reduce operator dependence on IPFIX in estimating OD pair demands for the purposes of network planning and traffic engineering.

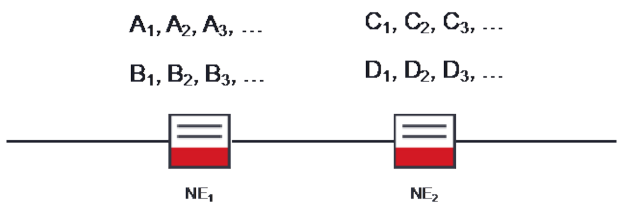
4.2. Reducing the amount of telemetry

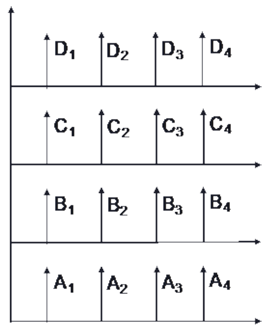
Another way to reduce the amount of IPFIX data is in network telemetry. Here we look at ways to reduce number of samples of the collected data. Unlike the previous method, which reduces the number of data sources, where the data is collected, this method reduces the amount of data collected by each data source.

The main idea is to reduce collection of data by decreasing the amount of information collected about some flow, and then to use the information collected from other sources (flows) and AI to infer what data that was not collect would be.

There are two uses for the method described in this section. First, it could be used to reduce the amount of telemetry data. Second, it could be used to increase the frequency of some measurements, while keeping the volume of telemetry about the same. The way this works is that we increase the frequency of measurements on some of the sources, while reducing the frequency of measurements of other sources. We then use AI to impute (estimate) the missing values in the sources with reduced measurement interval.

Network data samples are taken at a prescribed measurement interval (typically in the order of minutes). When picking the sampling interval, the network operator is typically not concerned about sampling interval from the point of view of the Nyquist criterion (Wikipedia), which is required to reconstruct the sampled data without the loss of information. The operators are not trying to reconstruct the data at the point of the collection and processing; data is typically collected for other purposes (forecasting, anomaly detection), so the precise reconstruction of the underlying random process is not important.





Spatially correlated time-series Reported time-series

Figure 6 Spatially and time correlated multi-variate time-series

As an example of time-series collection, we show two network elements and 4 time-series in Figure 6. NE_1 collects time-series A and B, while network element NE_2 collects time-series C and D. Time-series are on the same element are correlated, which means that information of both flows exists in either flow. For example, A(C) could be link utilization and packet latency could be series B(D). If link utilization is high, packet latency is also high, meaning that the two are correlated. Similarly, time-series on the same path are correlated. So, if A is link utilization on NE_1 and C is link utilization on NE_2, they are correlated due to the shared flows on those links. For example, if link utilization is A is high, this could be due to a large flow traversing NE_1, which is also traversing NE_2, so C is also high.

Data pruning

To reduce the information generated and transmitted by NEs we drop (prune) some of the samples. This is called "measurement sampling". Measurement sampling is a well-known technique used in both packet and flow-based measurement to reduce the data volumes required to report. The main idea in this technique is to take only a subset of packets or flows out of all packets or flows to obtain reasonable result for the measurement. For example, we could prune every k^{th} sample of each time-series. This is called subsampling and can be undone for each time-series using a low-pass filter, if the Nyquist criteria is satisfied for the subsampled time-series. *However, this is not what we do.*

Given measurement sampling, we propose a system architecture shown in Figure 7. In the network element, (1) the data pruning module receives the data from the data gathering module and removes some portion of the data before (2) transmitting it to the network analytics. In the network analytics module, a data recovery module (3) reconstructs (imputes) the data and passes it to the recovery evaluation module to (4) determine if the recovery was of high enough quality. Finally, the pruning logic module (5) instructs the data pruning module on how to prune data to improve performance.

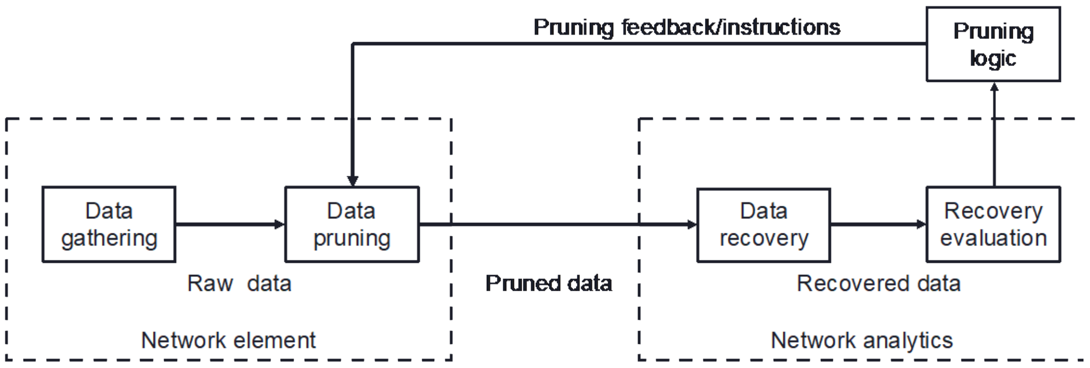


Figure 7 Pruning and reconstruction architecture

Correlation means that information about one time-series is available in another time-series. This is the fact we use when pruning and imputing missing information. We remove parts of a time-series so that some information is always available in another, correlated time-series. The information can be removed in many ways, but the easiest is to use an offset between the time-series when the k^th^ sample is removed. For the example in Figure 6, the samples can be reduced by only sending A_1, A_3, B_2, B_4, and C_1, C_3, D_2, D_4.

Data reconstruction

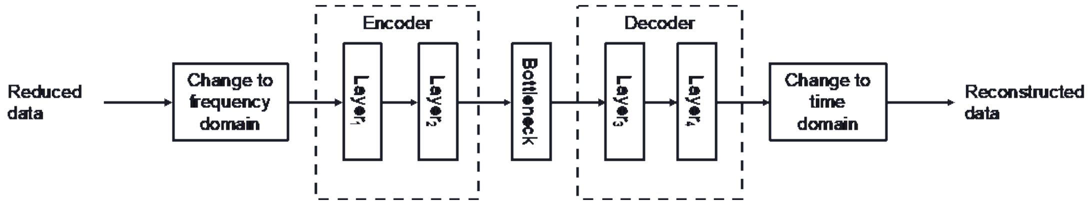


Figure 8 Reconstruction architecture

There are many ways of imputing the missing values received from the network elements. Here we discuss one way of doing it as a way of an example. The DNN treats the missing values as noise in the data and can work with any pruning strategy - the method does not require any information about how the data was pruned.

The AI reconstruction is shown in Figure 8. We use a multistage DNN. The pruned data is used as the input, while the reconstructed data is determined as the output. The method treats missing values as noise, so it is using a denoising method using an autoencoder (Wikipedia) (shown as the encode-bottleneck-decoder architecture). As the values are missing, the reduced data is first transformed in the frequency domain (using inverse Fourier transform or wavelet domain using wavelet transform). The transform into the frequency domain interlaces the missing and present values and so the structure of the data is pronounced even if some of the values are missing. The autoencoder structure denoises the frequency representation of the data thus emphasizing the structures in the data. The inverse Fourier transform then returns the denoised frequency domain data into the time domain.

Table 3 Summary of pruning/reconstruction performance results

Data reduction	5%	10%	20%
MAPE	4%	5%	9%

To evaluate this approach, we use the Abilene backbone dataset, which was discussed earlier. This is a relatively small dataset for this purpose, so learning to reconstruct the data is harder. Nevertheless, our analysis shows that data reduction is possible. For example, if we can tolerate a 10% reduction in data precision in data sources, which are being pruned, we can reduce the amount of transmitted data by 20%.

We think that this may be a promising approach to reduce the amount of telemetry used in IPFIX at a small loss of data precision.

4.3. Reducing the amount of stored information

Useful network data is high in volume, making it difficult to store for extended periods. Here we discuss compression of the data to make storage cheaper. For example, a network with a million data sources may generate 46Gb of 1-minute sampled data in one day, which translates to 16Tb of data per year. While the number of data sources may seem high, even in a traditional IP network this number would be inside the realm of possibility when considering each flow to be a data source. The number of data sources would be much higher when considering cloud services, IoT networks with billions of devices, multiple network layers, or high sampling network measurements (e.g., state of polarization, or wireless SNR measurements).

Using the example of 16Tb of data per year, the Amazon Web Services (AWS) S3 (AWS) cost to store it would be around $5000 in the first year and accumulating to $25,000 in year 5 of storing this data (AWS). Here, we describe a DNN-based lossy compression scheme to compress network data, which has a compression ratio of 100x-200x (Wikipedia). The one-time cost of compressing 16Tb using this compression scheme would be approximately $2.50 and the cost of storing the data for a year would be approximately $150-$250 for the one-year period, depending on the compression ratio. Over a 5-year period, the savings from compressing the data would be around $24,000, or 96%. These are significant savings that could be used in other business areas, instead of for simply storing data in the cloud.

Today’s solution is not to store the data, reducing the operator’s ability to make data-based decisions. Due to cost, network measurement data is not stored for extended periods, or it is aggregated in larger periods of time (daily, weekly, monthly, yearly), thus losing fidelity in an important historical record of what has happened in the network. The process of aggregation/averaging is a very crude way of lossy compression for time-series. For example, averaging represents a time-series with a single number over a period, so its accuracy is not good. The compression of averaging is not that good either - compressing 15-minute into a daily bin only has a compression ratio of 96, which we show is easily attained with neural networks.

Time-series compression with DNNs

We propose a method of storing network data in a deep-neural network, which greatly reduces the volume of information that needs to be stored. Figure 9 shows an example architecture for a network using the compression scheme. Data is collected from network elements (NEs), stored in temporary storage, compressed, and deployed as a deep neural network. Data is deleted after compression. Instead of using a database to store the data we use a server which hosts the DNN and allows querying of it to retrieve the data.

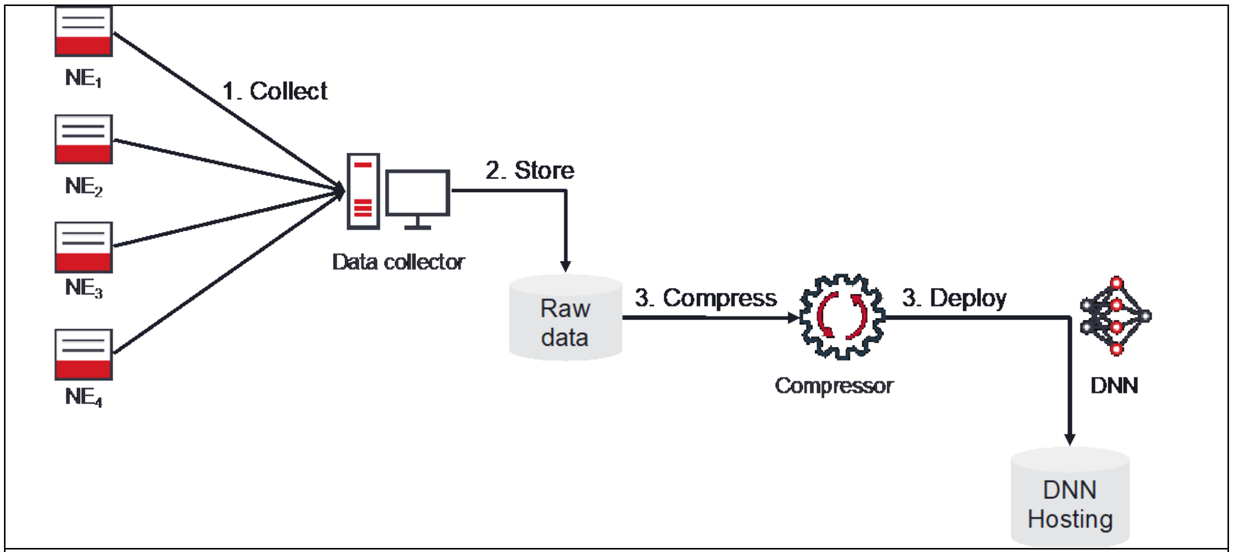


Figure 9 Network data collection infrastructure

The compression of the time-series is accomplished by using the ability of DNNs to memorize the mapping for any function. For example, we can train a DNN to memorize a relationship between a random index and a sequence of measurements. Suppose that the network data consists of sampled samples $\mathrm{ts}_1, \dots, \mathrm{ts}_n$. The samples are windowed into fragments $\mathrm{w}_i = (x_1, \dots, x_p)$ of p samples. Each fragment is associated with an index i.

Now we have a dataset where the features are the bits of the index i and the labels are the values in the fragment. We use a DNN to learn a function which maps a 32-bit integer into a p sample long fragment of a time-series. As an example, a 3-layer DNN is shown in Figure 10. Input to the DNN is the index of the bucket i and the output calculated by the DNN is the approximation of the time-series window $\widehat{\mathrm{w}}_i$

$$\widehat{\mathrm{w}}_i = W_3 \max\{0, W_2 \max\{0, W_1 i + b_1\} + b_2\} + b_3.$$

We note that due to lossy compression $\widehat{\mathrm{w}}_i \neq \mathrm{w}_i$, but their difference can be made arbitrarily small.

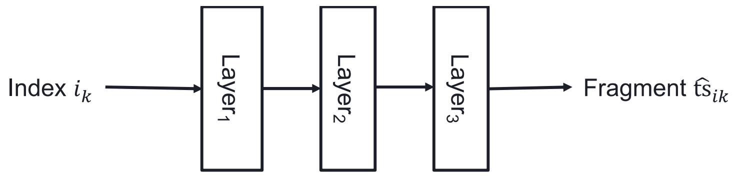
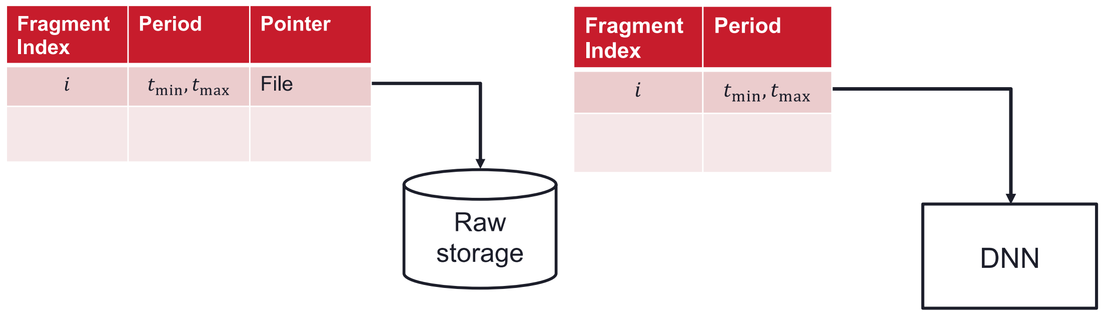


Figure 10 Decompression

Figure 11a compares the DNN approach to how data is stored today. Raw time-series are stored as fragments on disk and a table is used to relate times-series, periods, and the files on disk. Figure 11b shows how data is stored into a DNN. A table is utilized to relate the time-series, periods, and fragments with *indices*. The DNN contains the relationship between the index and the stored time-series. When an index is presented to the DNN, the DNN reconstructs the time-series bucket for that index.



a) Raw storage b) Compressed storage

Figure 11 Time-series storage

Achievable compression

The level of compression is strongly related to the level of achievable precision. In general, the stronger the compression, the less achievable precision is possible. This means that the compression can be used judiciously to save on space and time to train the DNNs. When comparing compression algorithms, an important metric is the compression ratio (Wikipedia). The compression ratio is the ratio of the size of the uncompressed data to the size of its compressed form. For example, if the original size of a dataset is 16 MB and its compressed size (size of the DNN) is 4 MB, the compression ratio would be 4x.

In many cases, the size of values in a time-series may vary significantly. Consider the case of “mice” and “elephant” flows in an IP network. There may be several orders of magnitude difference in the size of these flows. In the case of traffic engineering, it is much more important to know the size of large flows precisely than the size of small flows precisely. For example, if there are 100 small flows that can be compressed at 100x compression ratio and 1 large flow that can be compressed at compression ratio of 10x, to maintain its acceptably high precision for each set of flows, the average compression ratio compression could be 91x[1]. This should be compared to compressing all flows at the compression ratio of 10x, which the minimum required by elephant flows.

Table 4 Summary of compression performance results

Compression ratio	10x (XOR)	20x	40x	80x	128x	200x
MAPE	0%	0.5%	1%	2%	5%	10%

Table 4 show the compression achievable by DNNs. We used an IP-like dataset that we generated with a simulation. We use this dataset since we can control the size of the time-series, which makes it easier to see the performance of the compression scheme across different sizes. We also tried this on an IP dataset with similar results. The first column is for the XOR compression (Time-series compression algorithms, explained, n.d.), which is a lossless algorithm designed specifically for time-series. The MAPE for this algorithm is 0% since it is lossless. Next columns in the table use the compression we just showed. The compression varies from 20x to 200x, depending on the error in the estimates. We note that at 5% MAPE, which is very reasonable across a range of use cases the compression ratio is well over 100x.

We think that this may be a promising approach to reduce the disk space required to store IPFIX data at a small loss of data precision.

5. Summary

This paper has talked about various methods to reduce the amount of transmitted and stored telemetry data in IP network monitoring. We have described why network monitoring is important in relation to the monetization strategies used by network service providers. In most network monitoring use cases, the precision of the data is less important than the cost of collecting and storing the data. This introduces opportunities to trade off precision in the collected and stored IP telemetry with the cost of collection and storage.

We have reviewed 3 approaches that can reduce network telemetry 20% to 75% and the amount of stored IPFIX data by orders of magnitude. These are visionary applications of DNNs in network monitoring and the authors would appreciate feedback on their potential usefulness to service providers.

[1] We get this by calculating the compression ratio of large flows and small flows and then averaging out across all flows.

Abbreviations

AI	Artificial Intelligence
CLI	Command-line interface
CPU	Central Processing Unit
DNN	Deep neural network
GPU	Graphic Processing Units
IGP	Interior gateway protocols
IP	Internet Protocol
IPFIX	IP Flow Information Export
MAPE	Mean Absolute Percentage Error
ONNX	Open Neural Network Exchange
PoP	Point of Presence
RCA	Root Cause Analysis
S3	Simple Storage Service
SP	Service Providers
TPU	Tensor Processing Unit

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., . . . Jozefowicz, R. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems.* Retrieved from Software available from tensorflow.org: https://www.tensorflow.org/

AWS. (n.d.). *Amazon S3: Object storage built to retrieve any amount of data from anywhere.* Retrieved 07 21, 2021, from https://aws.amazon.com/s3/

AWS. (n.d.). *AWS Simple Monthly Calculator.* Retrieved from https://calculator.s3.amazonaws.com/index.html

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning.* MIT Press.

Hanin, B., & Sellke, M. (2018). *Approximating Continuous Functions by ReLU Nets of Minimal Width.* Retrieved from https://arxiv.org/abs/1710.11278

IETF. (n.d.). *Operations and Management Area Working Group (opsawg).* Retrieved 07 13, 2021, from https://datatracker.ietf.org/wg/opsawg/documents/

Network monitoring. (n.d.). Retrieved 07 06, 2021, from https://en.wikipedia.org/wiki/Network_monitoring

Open Neural Network Exchange. (n.d.). Retrieved from https://onnx.ai/

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., . . . DeVito, Z. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems 32*, (pp. 8024--8035).

Quittek, J., Zseby, T., Claise, B., & Zander, S. (2004). *Requirements for IP Flow Information Export (IPFIX).* Retrieved from https://www.rfc-editor.org/info/rfc3917

Roughan, M. (n.d.). *Internet Traffic Matrices.* Retrieved 07 16, 2021, from https://roughan.info/project/traffic_matrix/

Santos, O. (2016). *Network Security with NetFlow and IPFIX: Big Data Analytics for Information Security.* Cisco Press.

Time-series compression algorithms, explained. (n.d.). Retrieved 07 22, 2021, from https://blog.timescale.com/blog/time-series-compression-algorithms-explained/

Wikipedia. (n.d.). *Autoencoder.* Retrieved from https://en.wikipedia.org/wiki/Autoencoder

Wikipedia. (n.d.). *Automatic Differentiation*. Retrieved June 5, 2021, from https://en.wikipedia.org/wiki/Automatic_differentiation

Wikipedia. (n.d.). *Backpropagation*. Retrieved June 2, 2021, from https://en.wikipedia.org/wiki/Backpropagation

Wikipedia. (n.d.). *Benchmarking Methodology for Network Interconnect Devices*. Retrieved 07 09, 2021, from https://datatracker.ietf.org/doc/html/rfc2544

Wikipedia. (n.d.). *Carl Friedrich Gauss*. Retrieved June 2, 2021, from https://en.wikipedia.org/wiki/Carl_Friedrich_Gauss

Wikipedia. (n.d.). *Command-line interface*. Retrieved 07 06, 2021, from https://en.wikipedia.org/wiki/Command-line_interface

Wikipedia. (n.d.). *Data compression ratio.* Retrieved 07 21, 2021, from https://en.wikipedia.org/wiki/Data_compression_ratio

Wikipedia. (n.d.). *Graphics processing unit*. Retrieved June 2, 2021, from https://en.wikipedia.org/wiki/Graphics_processing_unit

Wikipedia. (n.d.). *Internet Control Message Protocol*. Retrieved 08 09, 2021, from https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol

Wikipedia. (n.d.). *IP Flow Information Export.* Retrieved 07 13, 2021, from https://en.wikipedia.org/wiki/IP_Flow_Information_Export

Wikipedia. (n.d.). *Iperf*. Retrieved 07 09, 2021, from https://en.wikipedia.org/wiki/Iperf

Wikipedia. (n.d.). *Isaac Newton*. Retrieved June 2, 2021, from https://en.wikipedia.org/wiki/Isaac_Newton

Wikipedia. (n.d.). *IS-IS*. Retrieved 07 06, 2021, from https://en.wikipedia.org/wiki/IS-IS

Wikipedia. (n.d.). *Matrix Multiplication*. Retrieved June 2, 2021, from https://en.wikipedia.org/wiki/Matrix_multiplication

Wikipedia. (n.d.). *MTR (software)*. Retrieved 07 09, 2021 , from https://en.wikipedia.org/wiki/MTR_(software)

Wikipedia. (n.d.). *NETCONF*. Retrieved 07 06, 2021, from https://en.wikipedia.org/wiki/NETCONF

Wikipedia. (n.d.). *Nyquist frequency.* Retrieved 07 16, 2021, from https://en.wikipedia.org/wiki/Nyquist_frequency

Wikipedia. (n.d.). *Open Shortest Path First*. Retrieved 07 06, 2021, from https://en.wikipedia.org/wiki/Open_Shortest_Path_First

Wikipedia. (n.d.). *Simple Network Management Protocol*. Retrieved 07 06, 2021, from https://en.wikipedia.org/wiki/Simple_Network_Management_Protocol

Wikipedia. (n.d.). *Structured programming*. Retrieved June 2, 2021, from https://en.wikipedia.org/wiki/Structured_programming

Wikipedia. (n.d.). *Tensor Processing Unit*. Retrieved June 2, 2021, from https://en.wikipedia.org/wiki/Tensor_Processing_Unit

Wikipedia. (n.d.). *YANG*. Retrieved 07 06, 2021, from https://en.wikipedia.org/wiki/YANG

Zhang, D., Mishra, S., Brynjolfsson, E., Etchemendy, J., Ganguli, D., Grosz, B., . . . Perrault, R. (n.d.). *The AI Index 2021 Annual Report.* Retrieved from arXiv: https://arxiv.org/abs/2103.06312